Approches a base de connaissances pour le test de circuits VLSI : application à la validation de prototypes dans le cadre d'un test sans contact

Meryem Marzouki

THESE

présentée par

Meryem MARZOUKI

pour obtenir le titre de DOCTEUR

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

(spécialité : informatique)

APPROCHES A BASE DE CONNAISSANCES POUR LE TEST DE CIRCUITS VLSI : APPLICATION A LA VALIDATION DE PROTOTYPES DANS LE CADRE D'UN TEST SANS CONTACT

date de soutenance : 06 février 1991

composition du jury :

MM.	COSTES Alain	Président et Rapporteur
	ABRAHAM Jacob A.	Rapporteur
	COLLIN Jean-Philippe	
Mme	CORDIER Marie-Odile	
Mr	COURTOIS Bernard	
Mme	SAUCIER Gabrielle	

Thèse préparée au sein du laboratoire TIM3

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

46 avenue Félix 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 Christiant
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.

AVANT-PROPOS

Cette thèse a été réalisée au sein de l'Equipe d'Architecture des Ordinateurs du laboratoire TIM3/IMAG. Les travaux dont elle témoigne ont été partiellement effectués dans le cadre des projets "ADVICE" (# 271 ESPRIT I-CCE) et "Localisation d'erreurs par MEB" (GCIS-CNRS).
J'ai personnellement bénéficié d'une bourse d'études dans le cadre de la coopération scientifique tuniso-française.

Je tiens en outre à remercier :

Mr Alain COSTES, Professeur à l'Institut National Polytechnique de Toulouse et Directeur du LAAS-CNRS, pour l'honneur qu'il me fait en présidant le jury de cette thèse. Je suis également très sensible au fait qu'il ait accepté d'en être rapporteur.

Mr Bernard COURTOIS, Directeur de Recherches au CNRS et Directeur du Laboratoire TIM3-IMAG, pour m'avoir accueillie au sein de l'équipe, et pour avoir dirigé mes travaux.

Mr Jacob A. ABRAHAM, Professor and Director of the Computer Engineering Research Center at the University of Texas at Austin, for his kind acceptance to act as a reviewer of this thesis. I hope this long, french document, hasn't been too tedious for him.

Mr Jean-Philippe COLLIN, Responsable du Département Technologie et Analyse de défaillances (1067) à IBM-France, dont j'ai déjà eu l'occasion d'apprécier la compétence et la gentillesse, Mme Marie-Odile CORDIER, Professeur à l'Université de Rennes I, et Mme Gabrielle SAUCIER, Professeur à l'Institut National Polytechnique de Grenoble et Directrice du Laboratoire CSI-INPG, pour avoir accepté de faire partie du jury de cette thèse.

.../...

Mes remerciements les plus chaleureux vont à Bernard COURTOIS et Jacques LAURENT.
Bernard m'a fait confiance et m'a donné confiance. Ce sont des choses qui comptent.
Jacques a fait plus que diriger le groupe de microscopie électronique : sa lecture de ma thèse a été minutieuse et précieuse, et je me souviens toujours avec plaisir de discussions épiques que, je l'espère, nous continuerons à entretenir. Ce sont des choses qui ne s'oublient pas.

Mes travaux ont largement bénéficié de discussions avec les membres du groupe de microscopie électronique. Gérard BAILLE, Isabelle GUIGUET, Dominique MICOLLET, Jeremy RUSSELL, Denis SAVART, m'ont beaucoup apporté. Avec Didier CONARD, j'ai en plus partagé les doutes et les difficultés, mais aussi l'enthousiasme et la bonne humeur.

Les trois années pendant lesquelles j'ai participé au projet ADVICE m'ont permis de connaître d'autres gens et d'autres méthodes de travail. C'est avec Marcello MELGARA que j'ai le plus travaillé, et j'ai pu apprécier sa gentillesse à cette occasion.

Enfin, je ne saurais oublier tous ceux, membres de l'Equipe d'Architecture des Ordinateurs, grâce à qui même les journées difficiles connaissaient de bons moments.

AVERTISSEMENT

Ce document est conçu en deux parties dont la lecture peut se faire de façon indépendante.
Chacune des deux parties possède donc son introduction, sa conclusion, ses références bibliographiques, sa table des matières, sa liste de figures, ainsi que sa propre numérotation de chapitres.

Cette structuration n'a pas été sans certaines redites, nécessaires à la compréhension de chaque partie comme une entité, mais peu nombreuses au demeurant, y compris dans les références bibliographiques.

D'autre part, le document aborde, et tente de mettre en rapport, les trois domaines de l'informatique et/ou de la microélectronique auxquels nous nous sommes intéressée : l'intelligence artificielle (plus précisément les systèmes à base de connaissances), le test et la testabilité des circuits intégrés, et le test par faisceau d'électrons.

Que les spécialistes de chacune de ces trois disciplines nous pardonnent si leur domaine n'est pas assez approfondi : plus que les différents chemins, c'est leur croisée qui nous a intéressée.
Mais que dire aux spécialistes des trois domaines, sinon que, si l'équilibre est certes plus aisé lorsque l'on prétend prendre plusieurs points d'appui, il est néanmoins plus long à trouver : un point peut devenir le centre du monde, deux points permettent de tracer une ligne droite sans grands risques, mais relier trois points non alignés résulte forcément en une ligne brisée, peut-être prémices d'un triangle infernal ; au delà de trois points, on peut sombrer dans la recherche de la quadrature du cercle, nous ne nous y risquerons pas.

ABSTRACT

Algorithmic methods, currently used for VLSI circuits testing, are facing two major problems which are, on the one hand the complexity of the devices under test, and on the other hand the incompleteness of the classical fault models.
These problems can be solved by using qualitative reasoning, based on partitioning and hierarchy concepts, and by taking into account very general fault models, like the occurrence of a discrepancy between the expected and the observed behavior of the device under test : such a solution is provided by the use of second generation knowledge-based systems.

The first part of this thesis is composed of the presentation of these systems and the assessment of the state of the art related to the use of knowledge-based approaches to electronic circuits and systems test and testability.

The second part of this thesis is concerned with prototype validation of VLSI circuits observed using a scanning electron microscope used in voltage contrast mode.
An overview of electron-beam testing possibilities and particularities is provided, before describing two different diagnostic approaches in this framework : the first approach is a knowledge-based one, while the second approach is based on the use of a fault dictionary.
These two approaches, as well as the two systems in which they have been embedded, are then evaluated and compared.

Keywords :
Second generation knowledge-based systems, Deep models, VLSI circuits testing, Diagnosis, Prolog, Prototype validation, Electron-beam testing.

RESUME

Les méthodes algorithmiques actuellement utilisées pour le test de circuits VLSI se heurtent à deux problèmes majeurs qui sont d'une part la complexité des dispositifs traités, et d'autre part l'imperfection des modèles de fautes pris en compte.
Ces problèmes peuvent être résolus par l'introduction de raisonnements qualitatifs, tirant efficacement parti des notions de partitionnement et de hiérarchie, et par la prise en compte de modèles de dysfonctionnement très généraux, comme la constatation d'une divergence entre le comportement attendu et le comportement observé du dispositif sous test : une telle solution est apportée par les systèmes à base de connaissances de seconde génération.

La présentation de ces systèmes, ainsi que l'état de l'art des approches à base de connaissances utilisées pour le test et la testabilité de systèmes électroniques, constituent la première partie du document.
La deuxième partie de la thèse est consacrée à la validation de prototypes de circuits VLSI observés à l'aide d'un microscope électronique à balayage utilisé en contraste de potentiel.
Après une présentation des possibilités et particularités du test par faisceau d'électrons, deux approches différentes du diagnostic sont décrites, ainsi que les deux systèmes dans lesquels elles ont respectivement été mises en oeuvre. La première est une approche à base de connaissances, et la deuxième une approche plus classique, basée sur l'utilisation d'un dictionnaire de fautes. Ces deux approches sont ensuite évaluées et comparées.

Mots clé :
Systèmes à base de connaissances de seconde génération, Modèles profonds, Test de circuits VLSI, Diagnostic, Prolog, Validation de prototypes, Test par faisceau d'électrons.

PREMIERE PARTIE
I.A. POUR LE TEST ET LA TESTABILITE DES SYSTEMES ELECTRONIQUES

TABLE DES MATIERES

I. INTRODUCTION A LA PREMIERE PARTIE 7

II. LES SYSTEMES A BASE DE CONNAISSANCES (SBC) 11
II.1 INTRODUCTION 11
II.2 NIVEAU OPERATIF : SYSTEMES DE PREMIERE GENERATION 12
II.2.1 Architecture générale d'un système expert 12
II.2.2 Acquisition des connaissances expertes 13
II.2.3 Représentation des connaissances 14
II.2.4 Utilisation des connaissances 15
II.3 CONNAISSANCES DE SURFACE ET CONNAISSANCES PROFONDES 16
II.4 NIVEAU DECISIONNEL : MODELES PROFONDS 18
II.4.1 Travaux de DAVIS et al. 18
II.4.2 Travaux de De KLEER et al. 23
II.4.3 Autres travaux 26
II.4.4 Bilan de l'approche modèles profonds 27
II.5 APPROCHE MIXTE : SYSTEMES DE SECONDE GENERATION 28
II.6 CONCLUSION 29

III. SBC POUR LA CONCEPTION SURE OU TESTABLE 31
III.1 INTRODUCTION 31
III.2 SYNTHESE AUTOMATIQUE 32
III.2.1 Généralités 32
III.2.2 D.A.A. (synthèse de chemins de données) 32
III.2.3 TDS (synthèse et analyse de timing) 33
III.3 CONCEPTION ADAPTEE AU TEST 35
III.3.1 Généralités 35
III.3.2 PLA-ESS (sélection d'une méthodologie de test de PLA) . . . 36
III.3.3 DFT-EXPERT (conception de circuits testables) 39
III.4 VERIFICATION DE REGLES DE DFT 40
III.4.1 Généralités 40
III.4.2 Vérificateur/analyseur de règles LSSD 40
III.4.3 ESTA (vérificateur de règles LSSD et BILBO) 43
III.5 CONCLUSION 45

IV. SBC POUR LA SIMULATION ET LA GENERATION DE VECTEURS 47

IV.1 INTRODUCTION 47
IV.2 GENERATION DE VECTEURS 48
IV.2.1 Généralités 48
IV.2.2 Le système THESEE 52
IV.2.3 Le système HITEST 53
IV.2.4 C-algorithme : génération concurrente 55
IV.3 SIMULATION 59
IV.3.1 Généralités 59
IV.3.2 Travaux de GULLICHSEN 61
IV.3.3 Travaux de GHOSH 64
IV.4 CONCLUSION 66

V. SBC POUR LE DIAGNOSTIC 69
V.1 INTRODUCTION 69
V.2 SYSTEMES A BASE DE CONNAISSANCES DE SURFACE 70
V.2.1 Généralités 70
V.2.2 Le système DART 71
V.2.3 Le système MIND 73
V.3 SYSTEMES A BASE DE CONNAISSANCES PROFONDES 75
V.3.1 Généralités 75
V.3.2 Le système FAULTFINDER 75
V.3.3 Travaux de THEARLING & IYER 77
V.4 CONCLUSION 81

VI. CONCLUSION DE LA PREMIERE PARTIE 83

Bibliographie 87

LISTE DES FIGURES

Figure II.1. Architecture générale d'un système expert 13

Figure II.2. Cycle de base d'un moteur d'inférences 16

Figure II.3. Exemple ... archétypique 21

Figure II.4. Treillis déduit de la figure II.3 26

Figure II.5. Déroulement des opérations dans le cadre d'une approche mixte 30

Figure III.1. Génération de chronogrammes avec TDS 34

Figure III.2. Matrice d'identification du PLA-exemple 37

Figure III.3. Contraintes initiales de l'utilisateur 38

Figure III.4. Comparaison MFT/UTS relativement aux contraintes de l'utilisateur 39

Figure III.5. Exemple de schéma et description correspondante 42

Figure III.6. Exemple de décomposition hiérarchique et graphe correspondant 44

Figure IV.1. Exemple de circuit et graphe de recherche du D-algorithme 50

Figure IV.2. Organisation du système HITEST 54

Figure IV.3. Exemple d'application du C-algorithme 56

Figure IV.4. Exemple de circuit simple et graphe ET-OU correspondant 58

Figure IV.5. Performances du simulateur de GULLICHSEN 64

Figure IV.6. Architecture générale du système de GHOSH 65

Figure V.1. Architecture du système MIND 74

Figure V.2. Noeuds génériques de l'arbre ET-OU 78

Figure V.3. Noeuds ET et noeuds OU pour des portes logiques élémentaires 79

Figure V.4. Arbre ET-OU et théorèmes dérivés d'un circuit exemple 80

I. INTRODUCTION A LA PREMIERE PARTIE

Depuis ses débuts, que l'on peut situer à la fin des années 50, l'intelligence artificielle (I.A.) constitue l'une des disciplines scientifiques les plus controversées.
Controversée essentiellement parce que trop porteuse d'espoirs : l'idée était bien belle, qui voulait faire "penser" les machines. Le seul commentaire que nous apporterons ici est que, si le fameux "test de Turing"[1] [Tur50] a bien failli succomber au charme d'une psychologue[2] [Wei66], cela montre seulement que, 40 ans plus tard, la conclusion d'Alan Turing ("quant à la question initiale, *les machines peuvent-elles penser*, elle n'a pas assez de sens pour mériter discussion" [Tur50], cité par [Lux90]) reste toujours vraie.

Une attitude, moins aventureuse, certes, mais peut-être plus saine, consiste à considérer l'intelligence artificielle d'un point de vue plus pragmatique : il s'agit, lorsqu'on est confronté à un problème trop complexe, non seulement par sa taille mais également par son mode de résolution, pour pouvoir le traiter par des méthodes déjà connues et éprouvées, d'envisager avec sérénité une réponse possible à la question qui a constitué la deuxième partie du titre de nombreux articles de revues et conférences : *can A.I. help ?*

Dans le cas qui nous intéresse, la question se traduit par "quels peuvent être les apports de l'I.A. au test et à la testabilité des systèmes électroniques?". Cette question peut encore se formuler ainsi : "en quoi l'I.A. peut-elle permettre de reculer les limites des méthodes employées actuellement pour le test et la testabilité des systèmes électroniques?".

Notre but, dans cette première partie du document, sera d'essayer d'apporter des éléments de réponse à cette question.
Pour ce faire, nous présenterons d'abord les principaux aspects du problème.

---

1. Méthode proposée par A. Turing pour déterminer si "une machine peut penser". Le déroulement du test nécessite deux personnes (que nous nommerons A et B) et la machine en question (que nous désignerons par C). La personne A, séparée de B et C, pose des questions à l'une ou l'autre sans savoir à "qui" elle s'adresse. Le but est que la machine réponde aux interrogations de la personne A de telle façon que celle-ci ne soit pas capable de reconnaître qui, des protagonistes B et C est l'être humain.
2. Il s'agit d'ELIZA, l'un des premiers programmes d'I.A., simulant le comportement d'un thérapeute selon la méthode de Roger. En fait, ELIZA est un programme tout à fait élémentaire, ne comportant aucun mécanisme de compréhension du langage, mais opérant par reconnaissance de mots-clé, et générant des réponses tellement vagues qu'elles peuvent passer pour raisonnables. Lorsqu'une phrase du "patient" ne comporte aucun mot-clé reconnaissable par ELIZA, le programme se contente de reprendre sous forme interrogative l'affirmation du "patient".

Nous nous intéressons donc aux systèmes électroniques d'une façon générale, et, bien qu'il sera très souvent question des circuits à haut niveau d'intégration, les cartes et systèmes électroniques complets seront également abordés.
Nous entendons "test" et "testabilité" comme les termes génériques de toutes les activités correspondant aux différentes phases du cycle de vie d'un système électronique, à savoir la conception (et plus particulièrement la conception adaptée à - ou en vue de - la testabilité), la simulation, la génération de vecteurs de test, et le diagnostic de fautes ou de défaillances (cas de l'analyse de défaillances).

Les problèmes qui se posent dans un tel cadre de travail découlent tous de la complexité des dispositifs traités et des exigences de qualité qui sont formulées.
Pour tenter d'évaluer la complexité du problème du test, il est nécessaire d'évaluer la complexité de l'algorithme de détection de fautes dans un circuit.
Il est généralement reconnu qu'un problème est inextricable s'il ne peut être résolu qu'en un temps exponentiel, c'est-à-dire un temps d'ordre k^n, où k est une constante donnée et n la taille du problème. Dans le cas du test de circuits intégrés, on pourra considérer la taille du problème comme étant le nombre de portes logiques et de connexions composant le circuit (lorsque la modélisation est effectuée au niveau portes logiques) [Fuj85].
Notre objet n'étant pas ici d'évaluer de façon formelle la complexité du problème du test, nous nous contenterons de référencer la suite de théorèmes établie par [Fuj85], et dont la preuve conduit à montrer que le problème de la détection de fautes est un problème NP-complet, donc appartenant à la classe des problèmes qui ne peuvent être résolus par un algorithme s'exécutant en un temps polynômial.

Qu'il s'agisse de détection de fautes ou de leur localisation, les méthodes classiques, pour essayer de surmonter la complexité des dispositifs traités, utilisent des hypothèses simplificatrices.
Ces hypothèses peuvent concerner le type de dispositifs traités (certaines méthodes, par exemple, ne traitent que les circuits combinatoires), leur niveau de représentation (relativement peu de méthodes considèrent le "niveau interrupteurs", encore appelé "niveau transistors"), et surtout le type de fautes prises en compte.
Cette dernière catégorie d'hypothèses a suscité plusieurs travaux en "modélisation de fautes", et les hypothèses les plus généralement formulées sont l'hypothèse de faute simple (une seule faute peut survenir à la fois dans le dispositif), et le fait que les fautes de type "collage à une valeur logique donnée" permettent de représenter la majorité des cas de fautes possibles.
D'autre part, les fautes sont souvent regroupées en classes d'équivalence lorsqu'elles peuvent avoir les mêmes effets, c'est-à-dire lorsqu'elles produisent les mêmes résultats aux sorties du circuit, ou plus généralement aux points observables du dispositif à tester.

Il est largement reconnu à présent que ces hypothèses simplificatrices ne permettent plus de répondre à des exigences de qualité qui se font de plus en plus drastiques, ni de

modéliser des fautes pouvant survenir dans des circuits fabriqués à l'aide de nouvelles technologies.
Plusieurs études ont donc été conduites pour essayer de remédier à ces lacunes présentées par les méthodes dites classiques. Ces études abordent notamment les aspects suivants [Rau87] :

- l'analyse de données volumineuses et, si possible, avec des temps de traitement très courts
- la recherche de configurations particulières dans des données volumineuses
- le raisonnement sur des informations incomplètes ou imprécises
- la mémorisation et le partage du savoir-faire

L'objectif de la première partie de la thèse est de faire le point sur les études existantes ou en cours.
Le chapitre II présente les systèmes à base de connaissances car ils constituent le domaine de l'intelligence artificielle le plus utilisé dans le cadre qui nous intéresse. Nous essayerons d'y spécifier les aspects les plus susceptibles d'être d'un apport réel pour le test et la testabilité des systèmes électroniques.
Les chapitres suivants nous permettront de dresser l'état de l'art des méthodes de test et de testabilité des systèmes électroniques utilisant des systèmes à base de connaissances.
Le chapitre III concerne les systèmes pour la conception et la conception pour la testabilité, le chapitre IV traite des systèmes pour la simulation et la génération de vecteurs de test. Enfin, le chapitre V est consacré aux systèmes pour le diagnostic de fautes, qu'il s'agisse de fautes de conception et de fabrication, ou de défaillances survenant au cours de la vie du système électronique étudié.

II. LES SYSTEMES A BASE DE CONNAISSANCES (SBC)

II.1 INTRODUCTION

Nous nous intéressons dans ce chapitre aux systèmes à base de connaissances en tant qu'approche *orientée intelligence artificielle* de la notion *d'assistance à l'opérateur.*
Le premier problème qui se dessine dans un tel cadre est donc l'évaluation du degré d'assistance que l'on attend du système à base de connaissances. Il s'agit alors de préciser à quel niveau, ou plus exactement *jusqu'à* quel niveau, un système informatique "intelligent" peut intervenir.

Pour répondre à cette question, psychologues et cogniticiens ont étudié - et étudient - le comportement humain dans le processus cognitif. Différents modèles, plus ou moins représentatifs, plus ou moins complexes, plus ou moins complets, ont été proposés. Parmi ces modèles, le modèle de performance humaine de Rasmussen [Boy88] distingue trois niveaux de comportement :

- le niveau des *habiletés* ("skill-based behavior")
- le niveau des *procédés* ("rule-based behavior")
- le niveau du *savoir* ("knowledge-based behavior")

Ces trois niveaux s'organisent dans la hiérarchie comportementale suivante : les habiletés correspondent au niveau *exécutif*, le plus bas ; les procédés correspondent au niveau *opératif* ; le savoir correspond au niveau *décisionnel*, le plus élevé, le plus abstrait, et le plus puissant.

Le niveau exécutif, lié chez l'humain aux activités sensori-motrices, et que l'on peut interpréter plus largement comme regroupant les tâches d'instrumentation effectuées par des capteurs/effecteurs, ne sera pas discuté ici.

Plus proches de nos objectifs sont les niveaux des procédés et du savoir :
Le niveau opératif correspond à un savoir-faire, acquis par expérience. Le terme anglais le désignant - "rule-based behavior" - exprime parfaitement le mécanisme de fonctionnement induit : application de règles ou de plans de conduite correspondant à une configuration donnée, et préalablement identifiée, de l'environnement. De ce point de vue, le niveau opératif permet de formaliser le comportement des systèmes à base de connaissances de première génération, encore appelés systèmes experts.

Le niveau décisionnel correspond véritablement au savoir, et permet seul de faire face à des situations imprévues, car il est basé sur une connaissance du milieu ou de l'environnement, et non pas sur une expérience des situations.

Dans la suite, nous nous intéresserons aux particularités présentées par chacun de ces deux niveaux, que ce soit par le type de connaissances qu'ils utilisent, les modes de raisonnement qu'ils induisent, la qualité d'assistance à l'opérateur qu'ils offrent, ou la façon dont ils sont mis en oeuvre dans les systèmes à base de connaissances.

II.2 NIVEAU OPERATIF : SYSTEMES DE PREMIERE GENERATION

Les systèmes à base de connaissances de première génération, ou systèmes experts, agissant au niveau opératif, sont les mieux connus à l'heure actuelle. Ils ont fait l'objet d'études nombreuses et importantes, ont permis le développement de concepts reconnus à présent comme essentiels dans l'évolution de l'intelligence artificielle, et connaissent même depuis quelques années un développement industriel.

Ces systèmes sont "experts" dans la mesure où ils sont censés :

- posséder la connaissance d'un expert humain dans un domaine
- reproduire le raisonnement de cet expert humain, confronté aux mêmes problèmes
- être capable de gérer des données et une connaissance incertaine
- Pouvoir justifier et/ou expliquer leurs conclusions

II.2.1 Architecture générale d'un système expert

L'architecture générale d'un système expert est présentée en figure II.1.

Les principales composantes en sont :

- la base de faits, ou mémoire de travail, contenant une image instantannée des objets du domaine traité et de leur état
- la (ou les) base de connaissances, ou mémoire à long terme. Elle est censée contenir la connaissance de l'expert sur le domaine
- le moteur d'inférences, agissant sur la base de faits, en utilisant la base de connaissances
- les interfaces, avec l'expert ou l'utilisateur du système, permettant le dialogue homme-machine et la mise à jour de la base de faits ainsi que de la base de connaissances

On distingue donc déjà la caractéristique majeure des systèmes experts par rapport aux systèmes informatiques plus classiques, qui consiste à séparer complètement les données (contenues dans la base de faits et la base de connaissances) du contrôle (assuré par le

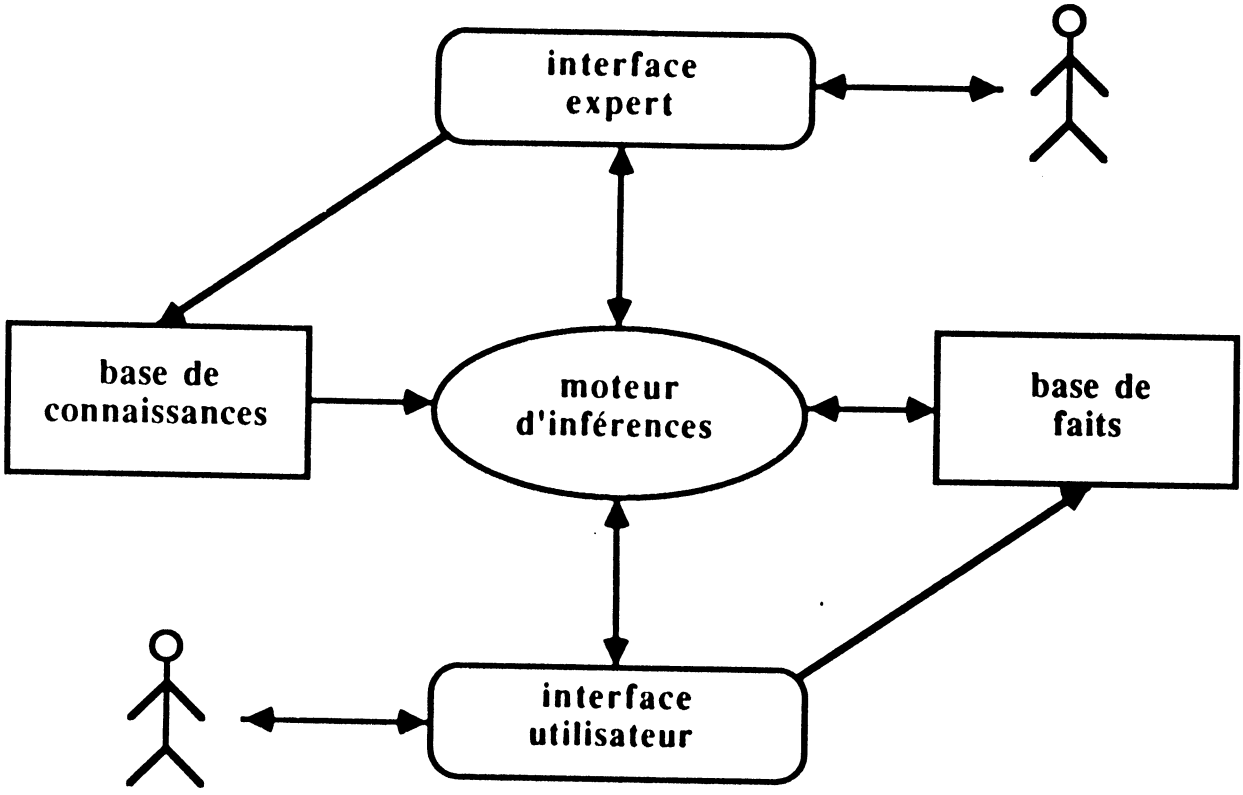


Figure II.1. Architecture générale d'un système expert

moteur d'inférence).

Les tâches les plus importantes dans la conception d'un système expert sont donc liées à la construction de ces composantes. Ces tâches sont décrites dans la suite.

II.2.2 Acquisition des connaissances expertes

Cette tâche va souvent constituer le goulet d'étranglement du système expert, puisque d'une part il s'agit de "faire dire" à l'expert choisi[3] toute la connaissance heuristique qu'il a de son domaine, connaissance dont parfois il ne soupçonne même pas l'étendue, ou du moins qui utilise des présupposés tellement intégrés dans son mode de reflexion qu'ils sont difficiles à identifier et formaliser, et d'autre part il n'existe pas de méthodes ou de méthodologies bien définies pour procéder à cette véritable opération d'extraction de la connaissance.

---

3. Le problème est encore plus complexe lorsque l'on veut considérer plusieurs experts, dont les itinéraires, l'expérience, ou tout simplement l'appréhension d'un cas, diffèrent, voire se contredisent.

Ceux que l'on appelle volontiers les "ingénieurs de la connaissance" doivent alors utiliser des procédés souvent empiriques, comme l'observation d'experts en situation, le "brainwriting", les interviews dirigées, les questionnaires à remplir, etc. [BFS88].

Mais, quel que soit le procédé employé, les problèmes qui se posent restent intrinsèques à la connaissance des experts eux-mêmes, puisqu'elle est *subjective* (la connaissance possédée par un expert n'est que la vision et l'expérience qu'a cet expert de son domaine, à un moment donné de son itinéraire professionnel), *volatile* (la connaissance est difficilement "capturable", et les experts qui la possèdent sont "mouvants"), et enfin *répartie* (la connaissance qu'a un expert n'est que partielle, et si l'on veut intégrer les connaissances de plusieurs experts, les recoupements nécessaires ne vont pas sans créer des conflits).

II.2.3 Représentation des connaissances

Une fois acquises et formalisées, les connaissances doivent être modélisées pour pouvoir être gérées et interprétées de façon automatique par le système.

Plusieurs formalismes de représentation des connaissances peuvent être utilisés, qui sont basés sur différents modèles logiques, parmi lesquels on trouve :

- *la logique propositionnelle* : système formel composé de propositions élémentaires à valeur de vérité binaire et des opérateurs ET, OU, NON, implication et équivalence
- *la logique du premier ordre* : le calcul des prédicats du premier ordre est un système formel composé de constantes, de variables, de prédicats pouvant contenir des variables et à valeur de vérité binaire, des opérateurs ET, OU, NON, implication et équivalence, et des quantificateurs existentiel et universel, ne pouvant porter que sur les variables
- *des logiques d'ordre supérieur*, généralement d'ordre 2. Dans ce cas, les prédicats peuvent eux-mêmes être des variables, et sont donc quantifiables
- *des logiques multivalentes*, dans lesquelles les valeurs de vérité ne sont pas forcément binaires, et il est donc possible de considérer d'autres valeurs que VRAI et FAUX
- *des logiques modales* : logiques étendues aux opérateurs intentionnels - modalités. C'est le cas des logiques temporelles, par exemple
- *des logiques de l'incertain*, permettant le traitement de connaissances incomplètes
- *des logiques floues*, permettant d'exprimer le degré de vérité d'une proposition

Pour plus de détails sur ces différents modèles logiques, voir [Som88].

On distingue principalement trois formalismes de représentation des connaissances, qui sont les faits, les règles et les objets. Notons que, souvent, un formalisme mixte peut constituer un bon compromis pour la représentation des connaissances.

Néanmoins, la grande majorité des systèmes experts existants sont des systèmes à règles de production. En effet, les systèmes de production, dont on peut trouver l'analyse détaillée dans [BaF82] ou [Lau82a] et [Lau82b] présentent les avantages suivants :

- *modularité* (ou granularité), puisque chaque règle de production constitue en elle-même un élément de connaissance
- *modifiabilité* aisée, qui est une conséquence de la modularité : la modification d'une règle ne doit pas influer sur les autres règles du système
- *lisibilité et facilité d'interprétation*, tant pour l'être humain que pour la machine, puisque d'une part le mode d'expression - *si conditions alors actions* - nous est naturel, et d'autre part le mode de raisonnement induit par ce formalisme - principalement *modus ponens* - est relativement simple à mettre en oeuvre
- *faculté d'auto-explication*, puisque la suite des règles appliquées peut fournir une trace du "raisonnement", encore que ce type d'explication reste assez simpliste dans de nombreux cas
- *efficacité*, enfin, si l'on se restreint au niveau de performance opératif défini plus haut.

II.2.4 Utilisation des connaissances

Quel que soit le formalisme employé pour représenter les connaissances, il faut concevoir des structures de contrôle permettant d'exploiter au mieux ces connaissances.

Bien sûr, les structures et le mécanisme de contrôle diffèreront selon le mode de représentation des connaissances. Néanmoins, le schéma de fonctionnement général reste le même : il s'agit, à partir de l'état courant de la base de faits, d'effectuer une transition en utilisant les données contenues dans la base de connaissances, pour arriver à un autre état de la mémoire de travail. Ce passage d'un état à un autre de la base de faits constitue un cycle de base du mécanisme de contrôle ou moteur d'inférences du système expert.

Dans le cas des systèmes de production, le cycle de base peut se concevoir comme montré en figure II.2.

Les cycles de base sont effectués soit jusqu'à ce qu'une solution au problème posé soit trouvée, soit jusqu'à blocage complet (aucune transition ne peut plus être déclenchée) : on dit alors que le système fonctionne en saturation.

Quoique des modes de contrôle mixtes soient possibles, on distingue deux principales stratégies de raisonnement, qui sont : le raisonnement en chaînage avant, ou guidé par les prémisses, dans lequel chaque transition consiste à déduire de nouveaux faits jusqu'à atteindre une solution, et le raisonnement en chaînage arrière, ou guidé par les buts, dans lequel chaque transition consiste à remplacer le but courant cherché par un ensemble (en fait, une conjonction ou une disjonction) de sous-buts à prouver Un sous-but est considéré

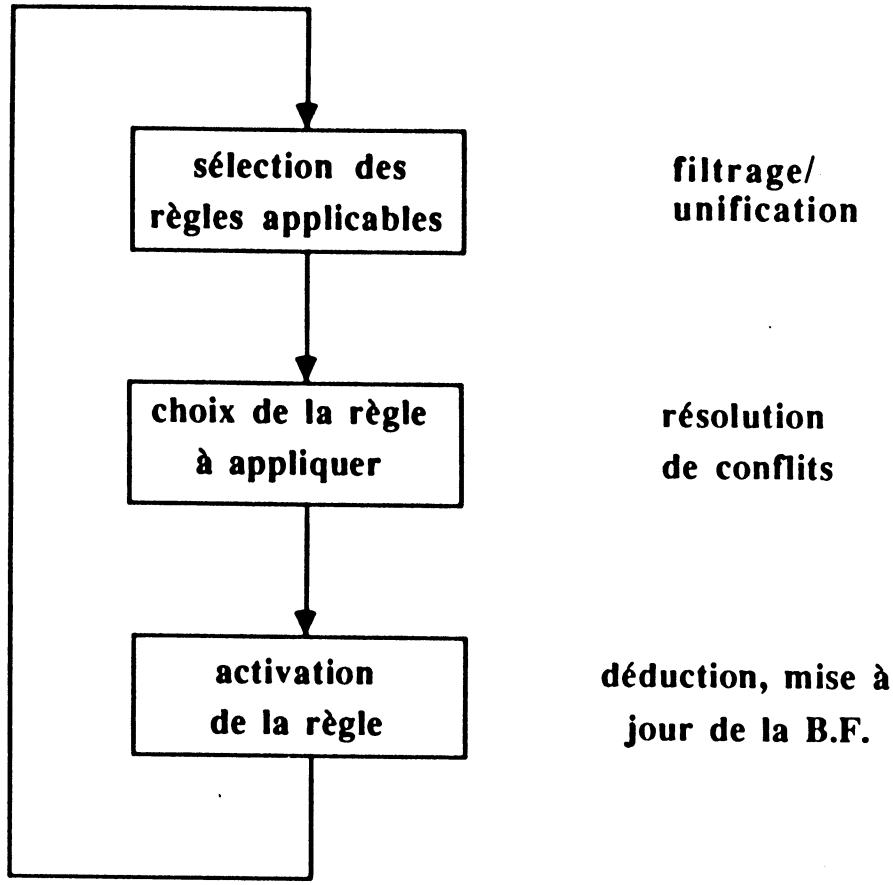


Figure II.2. Cycle de base d'un moteur d'inférences

comme prouvé lorsqu'il est présent dans la base de faits.

Si de plus il est possible, au cours du raisonnement, et en cas d'échec ou de blocage à un état intermédiaire, de restaurer un état antérieur de la base de faits, le moteur d'inférences peut effectuer ce que l'on nomme un "backtrack" (retour au dernier point de choix) d'état, et reprendre sa recherche de la solution dans une autre direction.

II.3 CONNAISSANCES DE SURFACE ET CONNAISSANCES PROFONDES

Dans [DaS83], Davis et Shrobe donnent l'une des premières explications de la distinction qui peut être faite entre ce que l'on nomme *connaissances de surface* ("shallow knowledge") et *connaissances profondes* ("deep knowledge") :

"Lorsqu'une machine aussi complexe qu'un ordinateur commence à présenter un dysfonctionnement, elle peut faire l'objet d'un diagnostic et d'une réparation de deux

façons différentes. Dans l'une des approches, un opérateur expérimenté en informatique *peut reconnaître les symptômes et entreprendre une action de maintenance en utilisant son expérience passée du problème. Dans la seconde approche, un* ingénieur électronicien expérimenté *sera capable de raisonner à sa manière sur le problème, en utilisant sa connaissance et sa compréhension de l'électronique digitale"*[4]

Davis et Shrobe nous semblent quelque peu mésestimer les facultés de raisonnement de l'opérateur, pour les besoins de la définition. Néanmoins, dans ce contexte, un système à base de connaissances de surface présentera le comportement de l'opérateur expérimenté en informatique, alors qu'un système à base de connaissances profondes reflètera plutôt le comportement de l'ingénieur électronicien expérimenté. On retrouve bien là la différenciation entre niveau opératif et niveau décisionnel.

Plus précisément, le concept de connaissances profondes a été introduit pour améliorer les résultats obtenus par ce que l'on a jusqu'il y a peu appelé l'"automatisation du processus de raisonnement" (typiquement, il s'agit des travaux développés sur les systèmes experts).
En effet, les premiers systèmes experts - ceux conçus dans les années 70 - étaient surtout basés sur des connaissances empiriques, leur manière de "raisonner" consistant le plus souvent à reconnaître une situation donnée (c'est-à-dire une collection de symptômes observés) parmi une liste d'états déterminés et préalablement identifiés, pour formuler les mêmes conclusions ou appliquer les mêmes traitements que ceux préconisés dans le cas particulier se présentant [BaF82].
Les systèmes experts répondant à ces caractéristiques, décrits en II.2, sont basés sur ce que l'on nomme des connaissances de surface, ou encore des connaissances associatives ("associational knowledge") [ChM85].

Plus récemment, de nouvelles voies de recherche ont été investiguées, partant de la constatation d'une part de la faiblesse d'une approche utilisant des connaissances empiriques, qui ne permettent pas au système informatique de raisonner, mais seulement de reproduire un comportement connu face à une situation connue, et d'autre part de la difficulté d'avoir cette connaissance accessible, complète, et utilisable [ChM85], cf. également § II.2.2.
Pour tenter de dépasser ces problèmes, et de faire face à d'autres nécessités, comme celle d'éviter le recours à des modèles de dysfonctionnement, ou celle de tenir compte des propriétés intrinsèques du dispositif en examen, une nouvelle approche des systèmes à base de connaissances a été définie, mettant en oeuvre le concept de connaissances profondes, concept qui a d'ailleurs induit un nouveau type de "raisonnement automatique", souvent

4. Cette citation, tirée de [DaS83], est une traduction de l'anglais.

appelé *raisonnement à base de modèles profonds* ("reasoning from first principles", ou encore "model-based reasoning") [Dav82], [DSH82].

Cette nouvelle approche repose sur la compréhension, l'interprétation et l'utilisation de la structure et du comportement du système en examen, considérant que cette compréhension conduit non seulement à une connaissance suffisante du système à l'étude, mais également à la capacité d'inférer un raisonnement sur ce système et son fonctionnement, ou ses dysfonctionnements [Dav84].

Les caractéristiques principales d'une telle approche sont :

- une nette distinction entre structure, fonctionnement, et comportement dans la représentation du système étudié
- une utilisation intensive de la hiérarchie de description et de niveaux d'abstraction multiples
- des corrélations possibles entre différentes vues ou facettes de la description du système
- une indépendance de la représentation du système par rapport à l'utilisation qui sera faite de cette modélisation

Ces caractéristiques, qui font de l'approche "raisonnement à base de modèles profonds" un intéressant et puissant paradigme pour la recherche dans le domaine des systèmes à base de connaissances, seront plus amplement développées et commentées dans la suite.

II.4 NIVEAU DECISIONNEL : MODELES PROFONDS

L'approche "modèles profonds" des systèmes à base de connaissances date du début des années 80. Malgré dix années de recherche, cette approche reste relativement peu répandue en Europe, et semble encore confinée à certains centres universitaires - certes prestigieux - des USA et du Canada. En conséquence, elle ne connaît pas encore - du moins à notre connaissance - d'application industrielle.

Les travaux les plus représentatifs dans ce domaine sont dus aux équipes de Randall DAVIS et Howard SHROBE (M.I.T.), Johan De KLEER (Xerox Parc), Michael GENESERETH (Université de Stanford), et Raymond REITER (Université de Toronto).
Ce sont principalement ces travaux qui seront exposés dans ce paragraphe.

II.4.1 Travaux de DAVIS et al.

Les travaux menés par l'équipe de DAVIS au M.I.T. sont à l'origine des idées fondamentales concernant le "raisonnement à partir de la structure et du comportement" du dispositif étudié.

Ces travaux ont été développés essentiellement en vue du diagnostic - localisation d'éléments fautifs - de systèmes électroniques, mais les théories qui en ont été issues sont suffisamment générales pour s'appliquer à une gamme de problèmes plus étendue.

Les concepts principaux mis en oeuvre dans cette approche adressent les trois grands problèmes du diagnostic, qui sont relatifs à [DaH88] :

- *la génération d'hypothèses* : étant donné un symptôme de dysfonctionnement du système, quels sont les composants susceptibles d'en être la cause ?
- *la vérification d'hypothèses* : étant donnés tous les composants susceptibles d'être cause d'un dysfonctionnement quelconque, quels sont ceux qui peuvent expliquer tous les dysfonctionnements observés ?
- *la discrimination d'hypothèses* : étant données les hypothèses vérifiées, quelles informations supplémentaires peuvent permettre d'aboutir effectivement au diagnostic ?

Ainsi formulées, ces trois tâches induisent déjà les présupposés, restrictions, et modes de fonctionnement de l'approche.

En effet, la notion même de symptôme de dysfonctionnement mérite tout d'abord d'être précisée, car il s'agit là de l'un des apports fondamentaux de l'approche "modèles profonds" par rapport aux systèmes de première génération : un dysfonctionnement quelconque, lorsqu'il est observé, se traduit uniquement par une *divergence* par rapport au comportement attendu du système, quelle que soit la façon dont cette divergence s'exprime (problème de la référence de bon fonctionnement) et quel que soit le niveau auquel elle s'exprime (problème des niveaux de représentation).
Ces deux problèmes - référence et niveaux de représentation - seront discutés plus loin (cf. § II.5).

D'autre part, la tâche de génération d'hypothèses requiert évidemment comme préalable l'observation des dysfonctionnements : il s'agit d'une étape précédant le diagnostic proprement dit, étape dite de détection d'erreurs, ou de collecte des symptômes. Les méthodes utilisées pour la détection sont très variables, et dépendent beaucoup du dispositif en examen. Néanmoins, la question qui concerne de près le diagnostic est celle du nombre et de la qualité des observations : plus ces observations seront significatives, et plus le diagnostic sera aisé et précis.

En outre, ces observations, pour permettre la détection d'erreurs, doivent être comparées à une référence. Pour ce faire, il faut se donner les moyens de spécifier le fonctionnement que l'on attend du dispositif en examen : il s'agit en général d'une simulation de son comportement correct.

De plus, lors de l'étape de discrimination d'hypothèses, le présupposé sous-jacent est clair dans l'approche proposée : un seul élément est responsable de tous les

dysfonctionnements possibles, et, par conséquent, ces différents dysfonctionnements ne peuvent être indépendants les uns des autres.
Ceci est, à notre avis, une restriction majeure de l'approche, encore que, dans le cas des systèmes électroniques, ce que l'on appelle communément "l'hypothèse de faute simple" constitue une restriction suffisamment plausible pour qu'elle soit couramment formulée.
Mais ce qui reste sans doute nécessaire pour simplifier des algorithmes "classiques" de diagnostic pourrait sans doute être évité dans une approche plus élaborée, en tirant justement parti des différents niveaux de représentation possibles du même dispositif, et d'une modélisation hiérarchique de ce dispositif.

Les travaux de DAVIS et al. se sont focalisés sur les deux premières tâches, c'est-à-dire la génération et la vérification d'hypothèses (c'est du moins ce qui ressort des différentes publications du groupe).

Ces travaux reposent sur une modélisation de la structure et du comportement du dispositif sous test. Par structure, il est entendu organisation physique et fonctionnelle des composants. La description du comportement revient à énoncer les contraintes - ou propriétés - qui doivent être respectées [DSH82], [DaS83], [Dav84].
Nous ne détaillerons pas plus cette modélisation, car il nous faudrait pour cela décrire une application particulière, mais les notions importantes y afférentes sont :

- une généralité suffisante pour qu'il ait été possible de définir un langage de description également utilisable pour la simulation, ce qui permet de garantir la cohérence globale du système
- un aspect fortement hiérarchique
- un aspect modulaire, utile pour la description structurée de dispositifs complexes
- des facilités de mise en correspondance des différentes descriptions possibles d'un même dispositif.

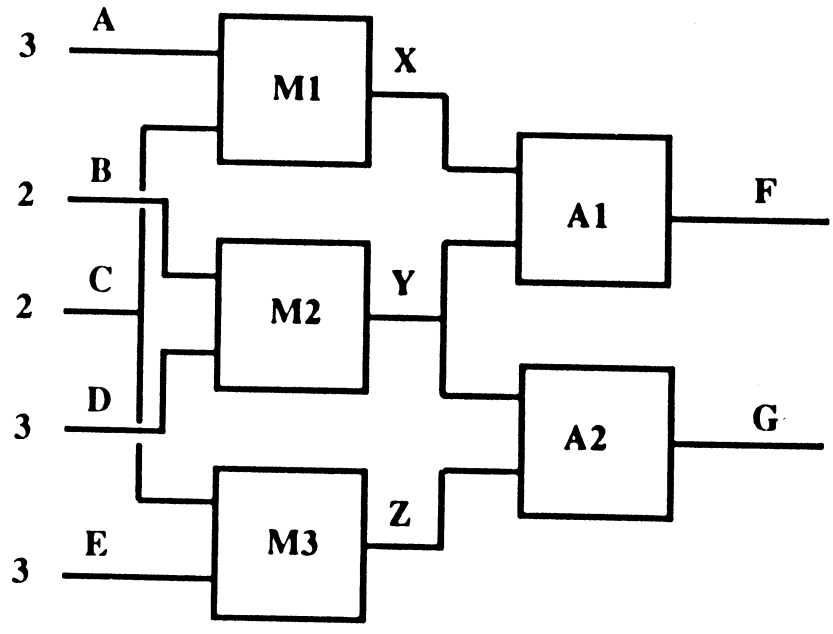
A l'aide de cette représentation, la génération et la vérification d'hypothèses se feront comme suit :

- *génération d'hypothèses :* il s'agit d'utiliser surtout la représentation de la structure (physique et fonctionnelle), c'est-à-dire la façon dont les composants du dispositif en examen sont interconnectés.
  A partir de chaque symptôme observé (donc du site de l'erreur), il faut collecter tous les composants ayant pu contribuer à causer cette erreur, parce que se trouvant sur un chemin - topologique et fonctionnel - dit d'"interaction causale".
  C'est donc une étape d'établissement de liens de causalité potentiels - en fait, de liens de dépendance -, permettant d'obtenir un ensemble de candidats.
  Différents ensembles de candidats - chacun correspondant à un symptôme - sont donc obtenus, et on restreint le nombre de ces candidats en effectuant l'intersection de tous

les ensembles. Il s'agit toujours de candidats potentiels.

- *vérification d'hypothèses :* pour effectuer cette tâche, une méthode dite de "suspension ou relaxation de contraintes" a été développée (rappelons que les contraintes permettent de spécifier le comportement).
  Pour chacun des candidats obtenus à la fin de l'étape de génération d'hypothèses, il s'agit de "suspendre" la contrainte représentant son comportement correct, et de vérifier s'il existe une configuration possible de comportement erroné telle que le symptôme dont ce candidat était potentiellement responsable est encore observable, auquel cas ce candidat peut être considéré comme un candidat possible. Si l'on ne trouve pas une telle configuration, le candidat doit être "disculpé".
  Cette étape, qui permet encore de réduire l'ensemble des candidats potentiellement fautifs, repose donc plus sur l'influence d'un composant sur le comportement du système complet que sur la dépendance topologique ou fonctionnelle des composants entre eux.

La figure II.3 montre l'exemple d'école utilisé par les équipes de DAVIS et De KLEER pour illustrer leurs travaux. Le dispositif à étudier est composé de deux additionneurs A1 et A2, et de trois multiplieurs M1, M2, et M3.



A, B, C, D, E : valeurs injectées
X, Y, Z : valeurs calculées
F, G : valeurs observées

Figure II.3. Exemple ... archétypique

La méthode de DAVIS et al., illustrée sur cet exemple, se déroule comme suit :

- **1er cas :**
  les observations donnent F=10 (au lieu de 12) et G=10 (au lieu de 12).

  * *génération d'hypothèses :*
    Le premier symptôme observé (F=10) permet de générer un premier ensemble de candidats : C1={A1,M1,M2}, puisque la valeur de F ne dépend que de l'un de ces trois composants.
    Le deuxième symptôme observé (G=10) permet de générer un deuxième ensemble de candidats : C2={A2,M2,M3}.
    L'application de l'hypothèse de faute simple permet de restreindre l'ensemble des candidats à : C=C1 ∩ C2={M2}.
    Fin de la génération d'hypothèses.

  Fin du processus, puisqu'on a pu trouver un candidat unique.

- **2ème cas :**
  les observations donnent F=10 (au lieu de 12) et G=12 (valeur correcte).

  * *génération d'hypothèses :*
    Le symptôme (F=10) permet de générer l'ensemble de candidats : C1={A1,M1,M2} (cf. 1er cas).
    Fin de la génération d'hypothèses (on n'observe pas d'autre symptôme), et C=C1={A1,M1,M2}.

  * *vérification d'hypothèses :*
    Suspension de la contrainte associée à M1 :
    étant données les valeurs injectées (A à E) et les valeurs que l'on peut calculer en utilisant le reste des contraintes (Y=6 et Z=6), il existe une valeur de X (X=4) telle que les observations (F=10 et G=12) restent cohérentes.
    M1 est donc maintenu comme candidat, et on a toujours : C=C1={A1,M1,M2}.
    Suspension de la contrainte associée à M2 :
    étant données les valeurs injectées (A à E) et les valeurs que l'on peut calculer en utilisant le reste des contraintes (X=6 et Z=6), il n'existe pas de valeur de Y telle que les observations (F=10 et G=12) restent cohérentes.
    M2 peut donc être disculpé, et on a : C={A1,M1}.
    Fin de la vérification d'hypothèses (inutile de suspendre la contrainte associée à A1, puisque ses entrées sont déterminées, et sa sortie observée).

  L'hypothèse de faute simple implique que le composant défaillant appartient à C, il faut donc discriminer les candidats A1 et M1.
  Fin du processus de génération et vérification d'hypothèses.

L'étape suivante - discrimination d'hypothèses - requiert, comme il a été dit plus haut, des informations supplémentaires et des procédures plus spécifiques.
Une méthode, proposée par [ShD83], consiste à générer des tests spécifiques, permettant de sensibiliser certains chemins dans le réseau de composants formant le système étudié.
L'intérêt de cette méthode est qu'elle est mise en oeuvre après les deux premières étapes, et par conséquent qu'elle bénéficie de la connaissance préalable de candidats potentiels.

L'idée est simple, puisqu'il s'agit de ne sensibiliser que des chemins ne comportant qu'un seul candidat à la fois. Lorsqu'on ne peut éviter d'avoir plus d'un candidat sur un même chemin, la méthode tire avantageusement parti de l'aspect hiérarchique de la description, par décomposition structurelle des candidats indésirables, n'empruntant que des chemins internes sûrs.

Cette méthode est donc basée sur le même principe que la suspension de contraintes, à la différence près qu'elle intervient au niveau structurel plutôt que comportemental.

II.4.2 Travaux de De KLEER et al.

L'approche suivie par De KLEER [DeW86] peut être considérée dans le même cadre que celle de DAVIS.
En effet, elle utilise également les notions de structure et de comportement traduit par une série de contraintes. Néanmoins, il s'agit d'une approche dans laquelle les étapes de génération et de vérification d'hypothèses ne sont pas effectuées successivement, mais combinées, grâce à l'utilisation de techniques ATMS ("Assumption-based Truth Maintenance System"), permettant de propager dans le réseau modélisé non seulement les valeurs observées, mais également les hypothèses formulées ("assumptions").
Dans la suite, nous emploierons le terme "contrainte" plutôt qu'hypothèse : une contrainte est associée à chacun des composants du système étudié, et traduit son comportement attendu. Une contrainte sera donc respectée si le composant correspondant est correct, elle sera transgressée s'il est défaillant.

La terminologie définie par De KLEER est la suivante :

- un *environnement* est un ensemble de contraintes toutes respectées
- un *candidat* est un ensemble de contraintes toutes transgressées
- un *conflit* est un ensemble de contraintes dont l'une au moins est trangressée (i.e. il y aurait incohérence, ou conflit, si toutes étaient respectées).
- soient OBS une observation donnée, et ENV un environnement. C(OBS,ENV) est vrai s'il est *cohérent d'observer OBS dans l'environnement ENV.*
- on définit de même P(OBS,ENV) comme *l'ensemble de toutes les prédictions de valeurs pouvant être inférées à partir de OBS dans l'environnement ENV.*

Une contrainte étant un composant du dispositif global étudié, un environnement, un candidat, ou un conflit, en tant qu'ensemble de contraintes, pourra être n'importe quel sous-ensemble de l'ensemble des contraintes.
Si le dispositif est décomposé en *n* éléments, il y a donc 2^n possibilités à examiner pour la détermination d'un candidat, d'un conflit, ou d'un environnement.
L'espace de recherche que l'on peut déduire de l'exemple de la figure II.3 est représenté dans l'approche de De KLEER par le treillis des relations entre sur-ensembles et sous-ensembles (figure II.4 [DeW86]).
Pour limiter l'aspect combinatoire de la recherche, des heuristiques sont utilisées, comme :

- le *principe de minimalité* : un conflit est tel que tout sur-ensemble de ce conflit est également un conflit. On considèrera donc uniquement les conflits de cardinalité minimale.

- la *monotonicité des prédictions par rapport aux observations* : étant données les mêmes conditions d'examen, les observations - mesures - sont cumulatives, et les prédictions P(OBS,ENV) ne peuvent être qu'incrémentales : il n'y a donc pas de remise en question possible des états et informations précédemment obtenus.

- la *monotonicité des prédictions par rapport aux environnements* : les prédictions P(OBS,ENV) sont incrémentales lorsque l'environnement s'élargit.

Ceci étant posé, la recherche des conflits se fait par calcul des "C(OBS,ENV)", pour les environnements possibles, en effectuant toutes les prédictions "P(OBS,ENV)" possibles.
Chaque fois qu'un nouveau conflit est identifié, les candidats sont générés de façon incrémentale, compte tenu des candidats générés à l'étape précédente, de la manière suivante : si un ancien candidat ne peut expliquer le nouveau conflit, il est remplacé par un ou plusieurs de ses sur-ensembles dont l'intersection avec le nouveau conflit n'est pas vide.[5]
Cette approche a été implémentée par De KLEER, et est connue sous le nom de GDE ("General Diagnostic Engine").
Un exemple permet d'illustrer son fonctionnement : considérons le dispositif de la figure II.3 ; les observations donnent F=10 (au lieu de 12) et G=12 (valeur correcte).

- *1er pas* (avant observation) : pas de conflit.
  le candidat minimal est donc $\varnothing$ ([] sur le treillis de la figure II.4).

5. On notera ici encore que le principe de minimalité implique une exploration du treillis "étage par étage", et par conséquent, la cardinalité des candidats ne peut être incrémentée que d'une unité à la fois, tant qu'il existe des candidats.

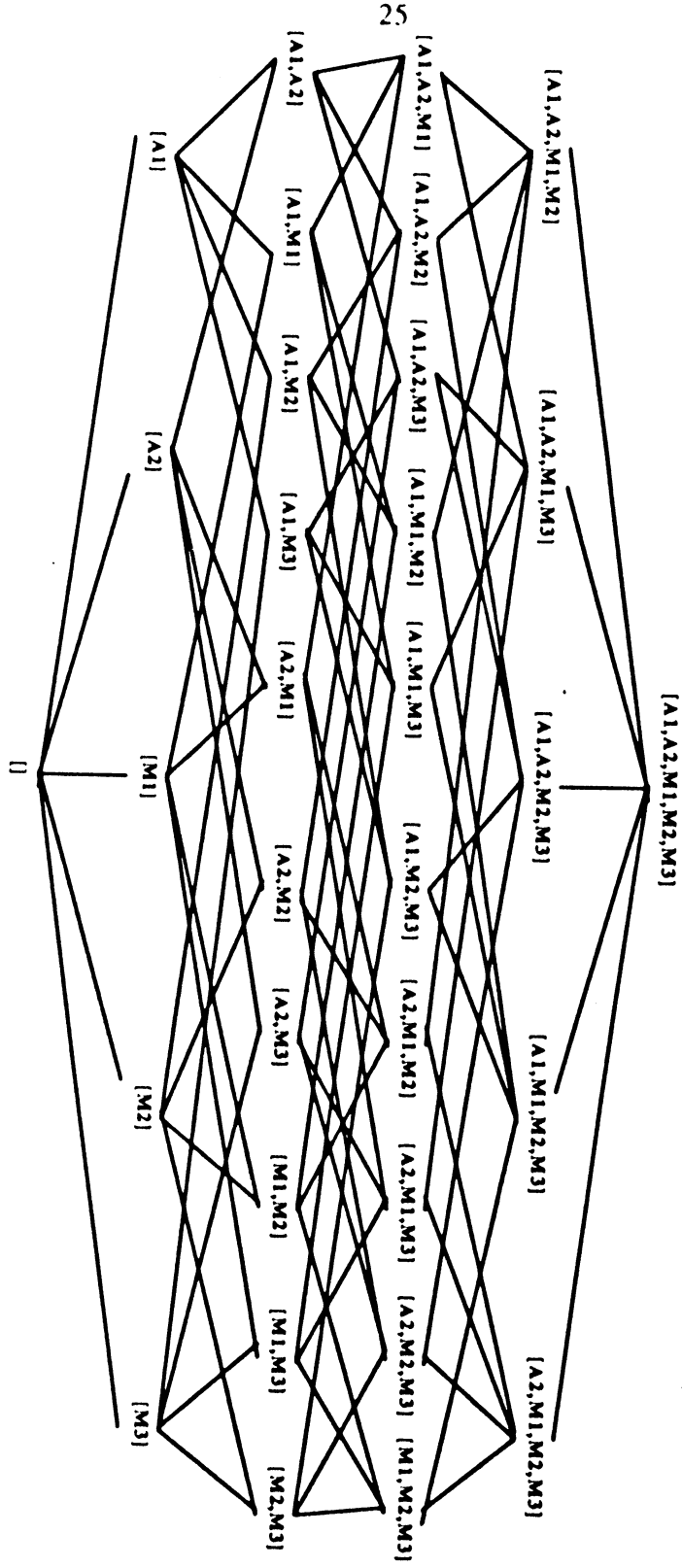


Figure II.4. Treillis déduit de la figure II.3

- *2ème pas :* observation F=10, conflit généré : <A1,M1,M2>
  le candidat minimal précédent (∅) ne peut pas expliquer ce conflit, ses sur-ensembles immédiats ([A1], [A2], [M1], [M2], [M3]) sont examinés.
  [A1], [M1], et [M2] expliquent le conflit, ils sont enregistrés.
  Ni [A2], ni [M3] ne l'explique ; seul [A2,M3] est l'un de leurs sur-ensembles qui ne soit pas sur-ensemble de [A1], [M1], ni [M2]. Mais on n'enregistre pas [A2,M3], car tous ses sur-ensembles sont également sur-ensembles de [A1], ou [M1], ou [M2] (principe de minimalité).
  On ne peut donc retenir comme candidats minimaux que [A1], ou [M1], ou [M2].

- *3ème pas :* observation G=12, sachant F=10 et [A1], [M1], ou [M2] candidats
  conflit généré : <A1,A2,M1,M3> (M2 est forcément correct).
  M2 ne peut expliquer ce conflit, on considère ses sur-ensembles immédiats [A1,M2], [A2,M2], [M1,M2], et [M2,M3].
  Chacun explique le nouveau conflit (i.e. a une intersection non vide avec l'ensemble {A1,A2,M1,M3} le représentant), mais [A1,M2] et [M1,M2] sont éliminés, car sur-ensembles des candidats [A1] et [M1] respectivement (principe de minimalité).
  On retient, des observations disponibles, les candidats possibles suivants :
  [A1], [M1], [A2,M2], et [M2,M3].

Cet exemple [DeW86], montre bien qu'il est possible de diagnostiquer aussi bien des fautes simples (A1 fautif, ou M1 fautif), que des fautes multiples (A2 et M2 simultanément fautifs, ou M2 et M3 simultanément fautifs).
Remarque : si l'on formule l'hypothèse de faute simple dans cette approche, on retrouve bien les résultats obtenus pas DAVIS dans le 2ème cas de l'exemple (A1 ou M1 fautif).

II.4.3 Autres travaux

Les autres travaux - notamment ceux de REITER et GENESERETH - seront évoqués rapidement, non pas car ils seraient d'un apport moindre, mais plutôt car les caractéristiques fondamentales des approches à base de modèles profonds ont déjà été exposées en II.4.1 et II.4.2.

La contribution de REITER [Rei87] réside dans la formalisation rigoureuse qu'il propose de ces approches, somme toute présentées de façon assez intuitive par DAVIS et De KLEER. Ainsi, REITER procède par abstraction des composants du système étudié en un ensemble de propriétés, la défaillance de l'un quelconque des composants étant représentée par un prédicat unique d'"anormalité" associé à ce composant [Rao89]. Le système étudié est donc représenté par un ensemble de formules du calcul des prédicats du premier ordre.

REITER définit un démonstrateur de théorèmes, formellement très puissant, mais pratiquement trop coûteux, qui, étant donnés les équations décrivant le dispositif, un ensemble d'observations et un ensemble de prédicats d'anormalité, détermine la cohérence de ce système de formules global, prouvant ainsi que les dysfonctionnements sont bien dus aux composants sur lesquels portent les prédicats d'anormalité.

Les travaux de GENESERETH, mis en oeuvre dans le système DART [Gen84], adressent plutôt le problème de la génération de tests du dispositif étudié.
Dans cette approche, la modélisation est entièrement basée sur le calcul des prédicats du premier ordre, qu'il s'agisse de représenter la structure, la fonction, ou le comportement.

Le système DART, après une première génération de candidats, essaie d'inférer les meilleurs tests permettant de discriminer ces candidats.
L'intérêt de cette méthode réside dans le fait qu'elle n'exploite justement que la description de la fonction et du comportement du dispositif étudié, les inférences étant effectuées grâce à une variante du principe de résolution [Rob65], appelée "resolution residue", la différence étant qu'il ne s'agit pas d'une technique de réfutation, mais plutôt d'une technique de preuve directe.

En ce sens, la méthode est bien plus puissante que celle utilisée par le D-algorithme [RBS67], par exemple, puisque tant le langage de représentation utilisé que le processus d'inférence sont indépendants du domaine d'application.

II.4.4 Bilan de l'approche modèles profonds

Les avantages de cette approche ont été largement présentés dans les paragraphes précédents. Pour en évaluer les limitations, il faudrait disposer d'informations sur des résultats d'application des méthodes présentées au diagnostic de circuits complexes réels, mais, à notre connaissance, de telles informations n'ont pas été rapportées dans la littérature.

Néanmoins, l'analyse que nous avons menée nous permet de conclure que des limitations existent, mais sont compensées par l'aspect fortement hiérarchisé de l'approche. De plus, la méthode agit à un niveau d'abstraction élevé : fonctionnel, et même comportemental. Le partitionnement, qui pourrait poser un problème réel, est déduit directement de l'approche de conception : la structure est donc immédiatement disponible. La versatilité de la méthode autorise en outre son application à plusieurs niveaux hiérarchiques d'abstraction sans modification majeure.

L'intérêt fondamental de cette approche réside donc dans le fait que sa mise en oeuvre est indépendante du type de dispositif auquel elle est appliquée. La taille importante des circuits actuels ne pose donc pas un problème insoluble. Par contre, comme il en est fait état dans [DaH88], la complexité d'un circuit, d'une partie de circuit, ou même d'un élément constitutif de base peut mettre en évidence les limites de cette approche si l'élément considéré ne peut être décrit facilement, notamment en ce qui concerne son comportement :

les circuits analogiques, ainsi que les circuits dédiés à une application particulière (ASICs), ou même certaines technologies de conception, peuvent entraîner des difficultés de modélisation.
Néanmoins, ces difficultés se situant non pas au niveau du circuit dans son ensemble, mais plutôt au niveau de composants de base (transistors, capacités, et parfois portes logiques), elles ne poseront pas de problèmes majeurs si l'on se contente d'un degré de résolution du diagnostic n'allant pas au delà de ces composants.

II.5 APPROCHE MIXTE : SYSTEMES DE SECONDE GENERATION

Les approches à base de "modèles profonds" tirent toute leur puissance justement du fait que les "premiers principes" sont exploités directement, et non pas au travers d'un processus de transformation humain dont les inconvénients résultent principalement en une perte - tant du point de vue quantitatif que du point de vue qualitatif - d'information, et en risque d'erreur.

Le problème majeur réside donc dans le modèle adopté : il faut que la modélisation soit suffisamment proche de la réalité pour bien la refléter, mais aussi que cette modélisation reste exploitable de façon automatique. En ce sens, les aspects hiérarchiques - plusieurs niveaux de détail possibles - et modulaires - séparation des représentations de la structure, de la fonction et du comportement - des modèles présentés permettent de faire face à cette exigence.

De plus, lorsque ces représentations et leurs modes d'exploitation sont basés sur des théories bien formalisées, comme le calcul des prédicats du premier ordre par exemple, ils peuvent permettre non seulement la description et le diagnostic du système étudié, mais également la réalisation d'un environnement complet, comprenant - dans le cas du test de systèmes électroniques, par exemple - des stratégies de simulation, des stratégies de génération de vecteurs de test, etc.

Par contre, la question de la référence de bon comportement reste ouverte dans le cas d'une approche "modèles profonds".
En effet, cette référence est en général constituée par des "valeurs attendues" issues d'un processus de simulation. Or il ne peut être question que d'une simulation du *comportement attendu* du dispositif, tel qu'il est décrit par la modélisation adoptée. Ceci implique qu'à un niveau hiérarchique de description donné, il n'existe aucun moyen, dans cette approche, de diagnostiquer des fautes dont l'effet est une modification de la structure - physique ou fonctionnelle - du dispositif, rendant le modèle incorrect.
Un exemple significatif est développé dans [Dav84]. Il s'agit, toujours dans le cadre du test de circuits intégrés, de l'apparition de fautes de type court-circuit entre deux lignes ("bridge fault"). Les mêmes difficultés se présentent pour les coupures de lignes.

Puisque des fautes de ce type ne peuvent être localisées car justement elles n'apparaissent pas dans la modélisation adoptée, une solution consiste à prévoir l'éventualité de leur apparition, puis à vérifier cette éventualité.
Il s'agit donc *d'augmenter le modèle* (par essence incomplet) d'un certain nombre d'hypothèses de fautes préalablement identifiées, et on retrouve là une approche connue et mise en oeuvre dans les systèmes de première génération.

Notre but n'étant pas ici de faire acte de prosélytisme en arguant de la suprématie de l'une ou l'autre des deux approches - à base de connaissances de surface ou à base de connaissances profondes -, nous plaiderons plutôt pour l'adoption d'une approche mixte, mise en oeuvre dans la démarche suivante (figure II.5), applicable à chaque niveau hiérarchique de représentation du dispositif étudié.

La démarche unifiée, telle que présentée en figure II.5, contient comme pré-supposé sous-jacent le fait que l'on désire obtenir le diagnostic le plus fin possible, puisque la fin de l'examen survient lorsque les seules erreurs constatées interviennent au niveau des chemins d'interaction entre composants. On peut néanmoins se contenter d'un diagnostic correspondant à un confinement de faute(s) dans un ou plusieurs composants, si le niveau hiérarchique de décomposition courant est estimé suffisant pour le traitement qui s'ensuivra (dans le cas du test de circuits intégrés, le traitement peut être tout simplement le remplacement des composants défaillants, et le diagnostic sera atteint lorsque le niveau hiérarchique de description courant correspondra à une décomposition en termes de "plus petites unités remplaçables").

Cette approche mixte a été mise en oeuvre dans le système PESTICIDE [MLC89] (cf. deuxième partie, chapitre IV, § IV.5.4).

II.6 CONCLUSION

Dans l'introduction de ce chapitre, il a été question du modèle de performance humaine de Rasmussen, dans lequel trois niveaux de comportement sont identifiés.
Nous nous sommes attachée à présenter le niveau des procédés - ou niveau opératif - et le niveau du savoir - ou niveau décisionnel - tels qu'ils sont mis en oeuvre dans les systèmes à base de connaissances, qu'il s'agisse de connaissances de surface représentant un savoir-faire ou de connaissances profondes représentant un savoir.
L'approche mixte que nous proposons dans le cadre de systèmes de seconde génération est rendue nécessaire par les inconvénients respectifs de chacune des méthodologies existantes.
Par les performances accrues qu'elle peut offrir, cette approche mixte vient corroborer l'effet synergique de l'association entre les niveaux du savoir et du savoir-faire, de la connaissance et de la compétence, du décisionnel et de l'opératif, de l'"épistémologiquement adéquat" et de l'"heuristiquement adéquat" [Hoc87].

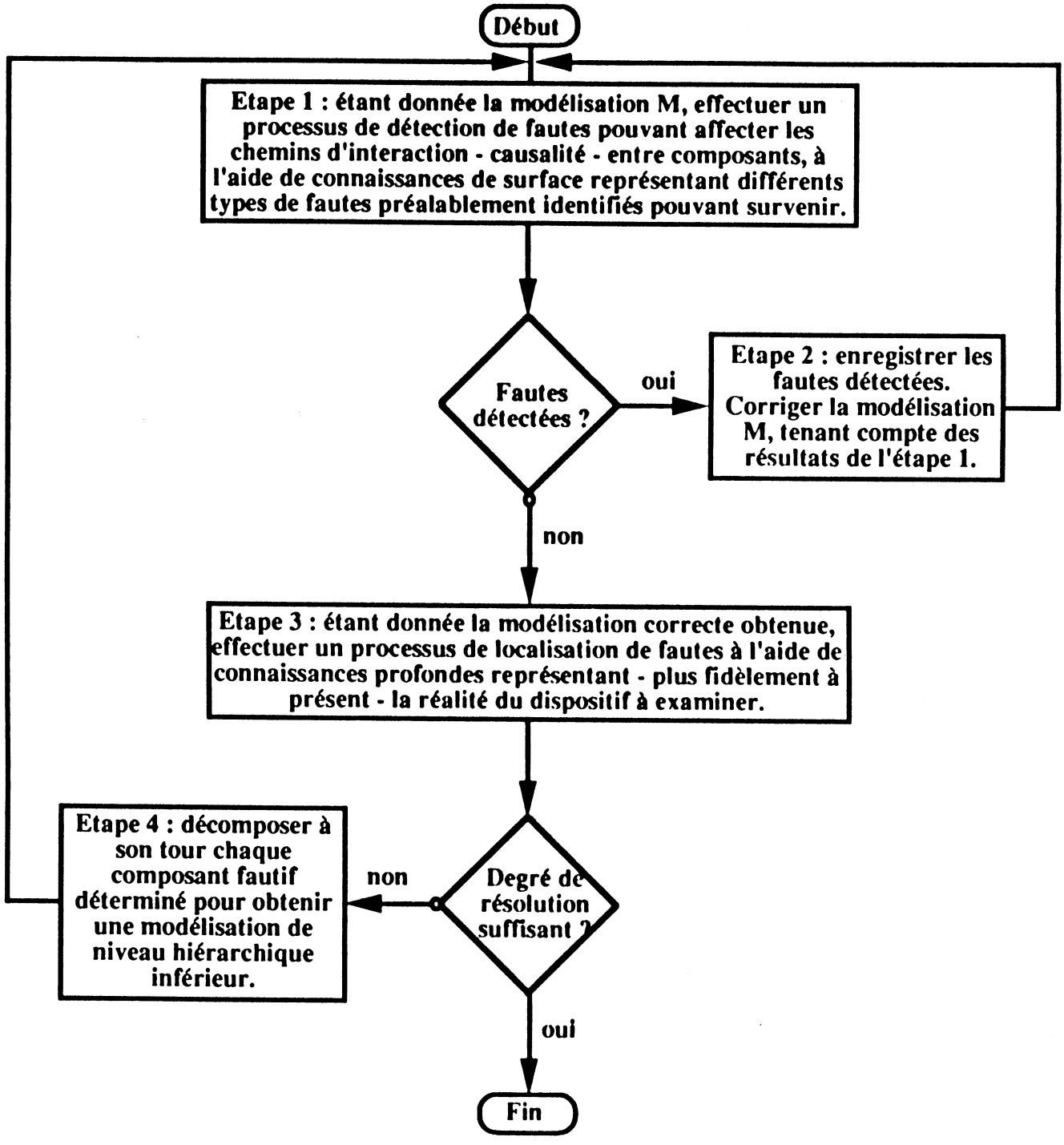


Figure II.5. Déroulement des opérations dans le cadre d'une approche mixte

III. SBC POUR LA CONCEPTION SURE OU TESTABLE

III.1 INTRODUCTION

Malgré le développement d'outils de CAO de plus en plus perfectionnés, dont les plus sophistiqués sont les compilateurs de silicium [Jer89], l'objectif du "zéro-défaut" n'est pas encore atteint pour les circuits VLSI, ni surtout pour les systèmes intégrés.
Néanmoins, d'importants progrès ont été et peuvent encore être réalisés pour tendre vers cet objectif. Ces progrès concernent essentiellement l'automatisation de la synthèse de circuits ou parties de circuits (cf. § III.2).

Mais, lorsque nous parlons de l'option "zéro-défaut", il ne s'agit que des défauts de conception. Même en supposant ceux-ci éliminés, des défauts de fabrication doivent être envisagés, ainsi que des défaillances survenant au cours de la vie du circuit.
Les circuits ont donc besoin d'être testés pour permettre la détection de ce type de défauts, et, lorsqu'il s'agit de circuits très complexes (VLSI), il est nécessaire de prévoir, dès la phase de conception, des dispositifs particuliers pour en assurer aisément la testabilité.

Des méthodes de conception adaptée au test ont été développées [WiP82]. Elles sont parfois spécifiques à certains types de synthèse (plusieurs d'entre elles ne sont prévues que pour des structures régulières), et ont chacune leurs avantages et leurs inconvénients. C'est pourquoi il n'est pas toujours facile de choisir la meilleure méthode de conception adaptée au test, et des systèmes de sélection assistée ou automatique ont été développés (cf. § III.3).

Enfin, une fois le choix de la méthode effectué, il faut l'implémenter correctement, et de la façon la moins coûteuse. Pour cela, certaines règles doivent être respectées, et étant donnée la complexité des circuits à traiter, des vérificateurs automatiques du respect de ces règles ont été conçus (cf. § III.4).

La suite de ce chapitre est consacrée à une revue de quelques-uns des systèmes présentant ces caractéristiques. Le point commun de tous ces systèmes est qu'ils démontrent les apports des techniques d'I.A. dans ce domaine, par les aspects qu'ils présentent tous : forte interactivité avec l'utilisateur, capacités d'explication, temps de traitement acceptables, confort d'utilisation, et parfois tout simplement résolution de problèmes auparavant insolubles, du moins d'un point de vue pratique.

III.2 SYNTHESE AUTOMATIQUE

III.2.1 Généralités

La synthèse de circuits, qui effectue la traduction d'une description d'un certain niveau d'abstraction en une description de plus bas niveau, comprenait jusqu'à récemment des tâches difficilement automatisables, car requérant les connaissances d'un expert du domaine. Le développement des systèmes à base de connaissances commence à permettre de remédier à ce problème.

On a pu ainsi assister à la conception et à l'utilisation de systèmes experts pour la synthèse logique de circuits ou parties de circuits, mais également de systèmes dédiés au placement de cellules et au routage de connexions, intervenant au niveau de la génération du plan de masse du circuit.
Qu'il s'agisse de synthèse logique ou de génération automatique, un système expert peut intervenir de trois façons différentes, qui sont [KeS87] : la création ou synthèse proprement dite (c'est le cas de D.A.A., développé au § III.2.2, TDS, décrit au § III.2.3, ou encore REDESIGN [StM84]), l'analyse critique (comme le fait le système CRITTER [Kel84]), et l'optimisation (comme dans SOCRATES [DeC85]).

III.2.2 D.A.A. (synthèse de chemins de données)

Le système D.A.A. ("Design Automation Assistant") [Kow84], [KGW85] est un outil d'aide à la synthèse logique de circuits et à son évaluation. L'objectif plus particulier de ce système est la synthèse automatique de chemins de données. Il a été développé par KOWALSKI à l'université de Carnegie-Mellon en 1984.

D.A.A. est un système à base de règles de production, formalisées par l'auteur après interview de nombreux concepteurs du secteur industriel.
Il contient 314 règles, dont 239 sont spécifiques au domaine d'application, c'est-à-dire aux tâches d'implantation du schéma, comme l'extension d'une conception partielle, le contrôle du contexte ou la mise à jour d'une base de données.
Les autres règles sont indépendantes du domaine, et ont pour rôle la gestion et l'utilisation des bases de connaissances du système.

Le fonctionnement du système D.A.A. est le suivant : à partir d'une description fonctionnelle de la machine-cible, une première étape de traduction est effectuée de façon à obtenir ce que l'auteur désigne par "value trace", utilisée pour réaliser, dans une deuxième étape, une allocation de ressources "grossière", pour obtenir une solution architecturale simple et non optimisée.
Ensuite vient la troisième étape, phase d'analyse effectivement experte, au cours de laquelle la solution architecturale simple sera optimisée à l'aide d'heuristiques de conception, pour produire la solution architecturale finale.

D.A.A. accepte en entrée la description fonctionnelle exprimée en langage ISPS ("Instruction Set Processor Specification"), laissant le soin à l'utilisateur du système d'effectuer la transformation, en amont, de la description comportementale.

D.A.A. est un exemple assez représentatif des "assistants de conception", et est bien accepté par ses utilisateurs, qui peuvent intervenir à plusieurs niveaux du processus de synthèse, grâce au caractère interactif de D.A.A.
Le système D.A.A. a été testé avec succès pour l'implémentation du jeu d'instructions du microprocesseur Motorola 6502 (3 heures de temps CPU sur VAX750), et du microprocesseur de l'IBM 370 (47 heures de temps CPU sur VAX11/780).
D.A.A. est un système écrit en langages OPS5 et C.

III.2.3 TDS (synthèse et analyse de timing)

Le système TDS ("Timing Design System") [KRK88], développé conjointement par le "Center for Intelligent Systems" de l'université Vanderbilt (Nashville) et la société NEC (Tokyo), est intéressant à deux titres puisque d'une part il intervient au niveau d'un système électronique et non pas d'un circuit intégré, et d'autre part il concerne les contraintes de synchronisation ("timing") devant être respectées par les signaux d'entrée/sortie des composants à interfacer.

TDS permet non seulement d'analyser le "timing" établi par un concepteur, mais également - et c'est l'apport majeur du système - de *générer* les "timing charts" ou chronogrammes.
Le processus de génération de chronogrammes effectué par TDS est représenté en figure III.1 [KRK88].

Les données d'entrée du système sont les spécifications de contraintes temporelles concernant les différents circuits et les bus devant assurer leur interfaçage. Ces spécifications, sous forme de clauses PROLOG, sont obtenues à partir des notices fournies par les différents constructeurs.

Lorsqu'il fonctionne en mode synthèse, TDS génère des chronogrammes valides par rapport à ses spécifications d'entrée et à l'architecture du système électronique dont il faut synchroniser les composants.
En mode diagnostic, TDS analyse les chronogrammes fournis par l'utilisateur pour vérifier que les contraintes relatives aux spécifications de chaque composant et à l'architecture du système global sont respectées.

TDS est un système à base de règles de production, développé en C-PROLOG sur VAX11/785, et comporte une base de spécifications, une base de connaissances heuristiques, un moteur d'inférences (en fait, quelques règles simples permettant d'utiliser l'interpréteur PROLOG [CIM85] en chaînage avant), ainsi qu'un échéancier pour le séquencement et la synchronisation des évènements.

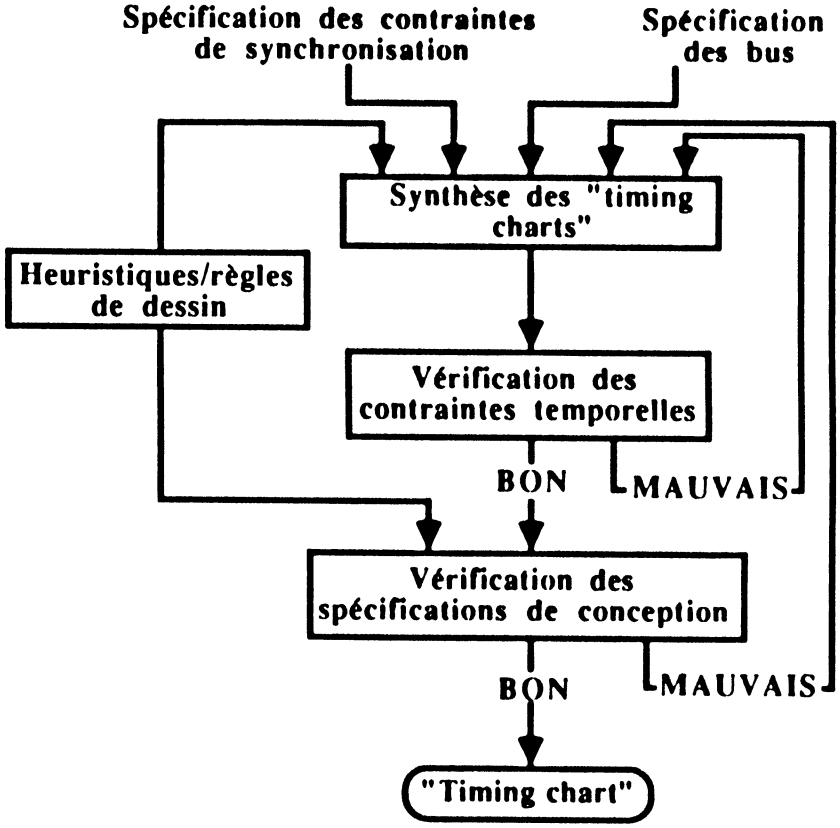


Figure III.1. Génération de chronogrammes avec TDS

Pour effectuer une synthèse, l'utilisateur commence par charger la base de spécifications adéquate. Il peut également fournir des contraintes temporelles supplémentaires sur n'importe quel signal d'entrée *ou de sortie*, puis TDS se charge de générer les contraintes relatives à tous les autres signaux de façon à respecter les contraintes de l'utilisateur, les spécifications des composants, et l'architecture du système global, en utilisant les heuristiques contenues dans sa base de connaissances.
Si aucune synchronisation valide ne peut être obtenue, TDS passe du mode synthèse au mode diagnostic, indiquant à l'utilisateur les problèmes rencontrés, de façon à ce que celui-ci puisse rectifier certaines des informations qu'il avait fournies au système, et une autre tentative de synthèse est effectuée.

Des résultats expérimentaux obtenus sur ordinateur DEC VAX 11/785, fournis dans [KRK88], ont permis de générer la synchronisation des signaux pour un cycle de lecture du microprocesseur NEC μpD41257 à mémoire RAM dynamique de 256K en 4.5 secondes de temps CPU, alors que la génération du cycle de base du microprocesseur INTEL 80286 a nécessité 20.4 secondes.

III.3 CONCEPTION ADAPTEE AU TEST

III.3.1 Généralités

La testabilité constitue de plus en plus l'aspect essentiel d'une bonne conception de circuit ou de système électronique. Elle est le reflet de deux notions, qui sont la contrôlabilité et l'observabilité.
La contrôlabilité permet de quantifier la possibilité de commander la logique interne du circuit à partir de ses entrées primaires.
L'observabilité permet de mesurer la possibilité d'observer la logique interne du circuit, ramenée au niveau de ses sorties primaires.

Deux types d'approches de conception adaptée au test (DFT : "Design For Testability") sont identifiées et exposées dans [WiP82] :

- *Les techniques de DFT ad hoc* comme le partitionnement (capacité de déconnecter logiquement une partie du circuit de façon à pouvoir la tester indépendamment du reste : approche "diviser pour régner"), l'insertion de points de test (entrées/sorties primaires supplémentaires), l'adoption d'une architecture structurée autour d'un bus principal, permettant l'accès à tous les modules, et l'analyse de signature, permettant la compaction des données observées, puisque la signature correspond au reste de la division d'un vecteur par un polynôme irréductible. Cette signature est obtenue en général à l'aide d'un registre à décalage à rebouclage linéaire (LFSR : Linear Feedback Shift Register"), dont l'implémentation doit être prévue dès les premiers stades de la conception. On verra dans la suite que la technique d'analyse de signature est également utilisée dans des approches plus structurées.

- *Les techniques de DFT structurées* : elles deviennent vraiment nécessaires lorsque l'on veut pouvoir appliquer les notions de DFT non plus à un problème précis, mais d'une façon plus générale et plus systématique, dans le but de réduire la complexité du test d'un circuit séquentiel à celle du test d'un circuit combinatoire.
  Le premier groupe de techniques structurées est constitué de méthodes qui sont toutes des variantes de la notion de "Scan Path" (chemin de balayage) [FWA75]. L'introduction d'un tel chemin, connectant tous les registres du circuit ou système, permet le balayage de tous les états internes, à la fois pour le contrôle et l'observation. Parmi les variantes de l'approche Scan Path, on peut distinguer la technique LSSD ("Level Sensitive Scan Design") [EiW77], qui est une façon de concevoir la logique séquentielle de sorte que d'une part un chemin de balayage existe ("scan design"), et d'autre part les opérations correctes ne dépendent pas des délais dans le circuit, mais uniquement des niveaux logiques ("level sensitive design").
  Le deuxième groupe de techniques structurées correspond à un test intégré, et regroupe sous le terme de BILBO ("Built-In Logic Block Observation") [KMZ79] les notions de LSSD, Scan Path, et analyse de signature. Cette technique consiste à intercaler un

registre dit "BILBO" entre chaque bloc de logique combinatoire. La particularité d'un registre BILBO est qu'il peut fonctionner dans l'un des quatre modes suivants : mode normal d'opération, mode LFSR, mode analyseur de signature, et mode "reset" de toutes les bascules du registre. Un registre BILBO résulte de la réorganisation d'un ensemble de bascules du circuit initial et de l'ajout de quelques portes logiques simples.

Chacune de ces techniques présente des inconvénients, qui sont le prix à payer pour les facilités de test qu'elles offrent. Il s'agit donc de trouver le meilleur compromis, en fonction des exigences du concepteur et des particularités du circuit ou système, entre par exemple l'augmentation de surface et du nombre de broches d'une part, et l'augmentation du taux de couverture de fautes et la diminution du coût de test d'autre part.

Le nombre de paramètres dont il faut tenir compte, la diversité des méthodes existantes, ainsi que la variété des circuits ou systèmes dans lesquels on veut les mettre en oeuvre rendent ce compromis difficile à atteindre.
Les systèmes à base de connaissances apportent une aide précieuse dans ce domaine, contribuant ainsi à une meilleure "hygiène de conception". Ces systèmes peuvent se limiter au *choix* de la méthode de DFT, comme PLA-ESS, décrit en III.3.2, ou TDES [AbB85], qui adresse d'autres structures que les PLAs, ou encore le système BIST [JoB87], spécifique aux méthodes d'autotest. D'autres systèmes proposent également une *implémentation* testable à l'aide de la méthode choisie : des exemples de tels systèmes sont DFT-EXPERT, décrit en III.3.3, ou encore TESTPERT [FuH86], qui constitue un module du compilateur de silicium SILC [BFR85].

III.3.2 PLA-ESS (sélection d'une méthodologie de test de PLA)

PLA-ESS (PLA Expert Synthesis System) [BrZ85], développé à l'université de Californie du Sud par BREUER et al., est un système permettant le choix d'une méthodologie pour rendre testable un PLA. Ce système est implémenté en langage LISP.

Les paramètres d'entrée de PLA-ESS, fournis par l'utilisateur, sont la description du PLA et les priorités ou contraintes à respecter.
Ces paramètres, ainsi que les caractéristiques des différentes méthodes de testabilité - contenues dans la base de connaissances du système -, sont utilisés pour construire une matrice d'évaluation, dont chaque ligne est constituée par le vecteur d'évaluation d'une méthode particulière.
Ce vecteur d'évaluation est une fonction du PLA et de la méthode elle-même, et chacune de ses composantes est la valeur prise par l'un des attributs permettant de caractériser les performances des différentes méthodes de testabilité.
Parmi ces attributs, on peut trouver le taux de couverture de fautes, le pourcentage d'augmentation de surface, le temps nécessaire pour un cycle de test ("test application time"), etc. (une liste exhaustive est fournie dans [BrZ85]).

La matrice d'évaluation est ensuite utilisée par une fonction assignant des crédits ou des pénalités à chaque méthode de test, en fonction de la distance mesurée entre chacun de ses attributs et les priorités de l'utilisateur.

A la fin de ce processus, l'une des situations suivantes peut survenir :

- Une solution unique existe : elle est identifiée, et transmise à l'utilisateur
- Plusieurs solutions existent : la meilleure - la plus proche des contraintes de l'utilisateur - est identifiée
- Il n'existe aucune solution : le système invoque un processus d'"analyse des raisons" de cet échec. Ce processus a pour objectif de déterminer où se situent exactement les problèmes - au niveau de quels attributs -, et de proposer à l'utilisateur, pour chaque méthode, une modification des contraintes associées aux attributs.
  Ce processus est fortement interactif, et repose sur un backtracking du raisonnement au niveau des points de choix erronés.

Lorsque le système arrive à une solution acceptée par l'utilisateur, il fournit également des indications sur la façon dont elle peut être implémentée.

Un exemple simple d'exécution, fourni dans [BrZ85], donne une vue d'ensemble des possibilités du système PLA-ESS :
Le PLA à traiter est chargé à partir d'un fichier ("personality file"), dans lequel il doit être préalablement décrit sous forme matricielle ("identification matrix"). Ainsi, le PLA à 5 entrées, 7 sorties et 9 monômes de l'exemple est décrit comme en figure III.2.

XX0X1	0000100
0X001	0100000
XX001	0001010
XX01X	0001000
0X010	1100000
0X011	0010000
XX100	0001011
00100	0010000
010X0	1000000

Figure III.2. Matrice d'identification du PLA-exemple

Le système disposant de plusieurs techniques de conception testable, il propose à l'utilisateur différents attributs - qui permettront la discrimination entre les techniques -, précisant pour chacun sa valeur courante, et lui demande de formuler ses contraintes en modifiant les valeurs des attributs concernés.
La deuxième étape consiste, pour l'utilisateur, à classer, en les pondérant, les attributs par ordre d'importance. A partir de ces données, le système entame un échange interactif avec

l'utilisateur, de façon à procéder au choix de la meilleure méthode. Cet échange consiste, dans le cas de l'exemple (caractérisé par les données résumées en figure III.3) en les étapes suivantes :

#	attribut	valeur	unité	pondération
1	génération tests	NON	(OUI/NON)	4
2	stockage tests	600	bits	7
3	tps d'appli. tests	0.8E-3	MS	9
4	autotest	NON	(OUI/NON)	2
5	test concurrent	NON	(OUI/NON)	2
6	maskage fautes	NON	(OUI/NON)	1
7	E/S additionnelles	5	# d'E/S	10
8	surface	100	%	80
9	taux couverture	99	%	70

Figure III.3. Contraintes initiales de l'utilisateur

- le critère le plus important (surface) est examiné. La technique permettant le minimum d'augmentation requiert une surface de 102.3%
- la modification est acceptée par l'utilisateur. La surface passe de 100% à 102.3%
- les autres critères sont examinés. Deux seules techniques satisfont à toutes les contraintes sauf une : la contrainte de surface. Ces deux techniques sont présentées à l'utilisateur
- l'utilisateur demande à examiner d'autres techniques, proches de ses contraintes, puis réclame de plus amples informations : comparaison des trois meilleures techniques, en l'occurrence MFT ("Multiple Fault Testable Design of PLAs"), UTS ("Universal Test Set"), et FIT ("Functional Independant Testing PLAs"). L'utilisateur choisit la méthode UTS en première approche, mais après consultation de sa description détaillée, n'est toujours pas satisfait par la contrainte de surface (112.3% pour UTS).
- une comparaison (figure III.4) entre MFT et UTS lui est proposée
- l'utilisateur finit par choisir la méthode MFT, malgré le nombre de vecteurs à stocker et le temps d'application de test plus importants
- fin de la session : le système modifie le PLA originel de façon à pouvoir implémenter la méthode MFT, et génère l'ensemble des vecteurs de test nécessaires à l'application de cette méthode.

attribut	valeur requise	pondération	valeur MFT	valeur UTS
génération tests	NON	0.1	NON	NON
stockage tests	600	0.1	868	296
tps d'appli. tests	0.8E-3	0.2	0.9E-3	0.7E-3
autotest	NON	0.5E-1	NON	NON
test concurrent	NON	0.2E-1	NON	NON
maskage fautes	NON	0.2E-1	NON	NON
E/S additionnelles	5	0.2	4	5
surface	102.3	2.1	102.3	112.3
taux couverture	99	1.9	100.0	99.5
logique additionnelle	--	--	186	183
coût conception	--	--	1	1
effet sur performances	--	--	1	1
complexité layout	--	--	1.5	2

Figure III.4. Comparaison MFT/UTS relativement aux contraintes de l'utilisateur

III.3.3 DFT-EXPERT (conception de circuits testables)

Le système DFT-EXPERT [BhP89a] est un ensemble de modules experts permettant la conception de circuits facilement testables à l'aide de méthodes connues de conception adaptée au test.
Ce système est développé par BHAWMIK et PALCHAUDHURI, à l'Institut Universitaire de Technologie de Kharagpur (Inde).

DFT-EXPERT est un système à base de règles de production, pour lesquelles la partie "action" est constituée d'appels à des procédures, dont l'exécution aura pour résultat la modification de la mémoire de travail du système (sa base de faits) [Bha88].
DFT-EXPERT a été développé à l'aide du générateur de systèmes experts KET (Kharagpur Expert system building Tool). Chaque règle, dans DFT-EXPERT, contient un "facteur de priorité", utile dans la phase de sélection des règles à appliquer. Le moteur d'inférences du système fonctionne en chaînage avant.

L'intérêt principal de DFT-EXPERT, contrairement à d'autres systèmes du même type, est de ne pas être spécifique à une méthode de DFT donnée.
Ainsi, étant données les priorités de l'utilisateur (maximiser le taux de couverture de fautes, minimiser les entrées/sorties additionnelles, minimiser la surface supplémentaire, etc.), exprimées à l'aide d'un "facteur de motivation" compris entre 0 et 1 pour chacun des paramètres, et la description du circuit au niveau transfert de registres, DFT-EXPERT va sélectionner les structures de DFT ("scan-paths", LFSRs, etc.) les mieux adaptées.

Le principe de fonctionnement de DFT-EXPERT est le suivant :

- Identification et classification des composants du circuits en "transporteurs de données" (Data Transporters ou DTs) et "processeurs de données" (Data Processors ou DPs), à l'aide de règles.
  Les DTs peuvent être des registres, multiplexeurs, etc., et les DPs des additionneurs, décodeurs, etc.
  Le partitionnement ainsi obtenu - ensemble de DTs et de DPs interconnectés - permettra d'identifier des chemins de test possibles (chaînes de DTs).
- Sélection de la méthode de test la mieux adaptée à chaque DP (les méthodes actuellement disponibles dans DFT-EXPERT sont relatives aux PLAs et à la logique aléatoire) [BhP89b].
- Proposition d'une configuration globale de DFT, en essayant de satisfaire les différents facteurs de motivation, et de maximiser le partage des structures de DFT, comme les registres de "scan-path" et les LFSRs, entre plusieurs DPs.
- Génération, à partir de la configuration obtenue, de séquences/plans de test du circuit, en résolvant les conflits relatifs au partage de ressources (les structures et chemins de DFT), conflits engendrés lorsqu'un certain degré de parallélisme (test simultané de plusieurs DPs) est envisagé.

DFT-EXPERT est actuellement en cours d'extension, pour permettre également la conception adaptée au test de structures RAM et ROM [BhP89a].

III.4 VERIFICATION DE REGLES DE DFT

III.4.1 Généralités

Lorsque l'implémentation d'une méthode de DFT n'est pas réalisée automatiquement, elle doit être nécessairement vérifiée. Les méthodes de DFT possédant leurs propres règles de conception, celles-ci peuvent être facilement encodées sous forme de règles d'inférence, car elles sont souvent indépendantes les unes des autres, et présentent des aspects beaucoup plus déclaratifs que procéduraux.
Deux exemples de systèmes experts de vérification de règles de DFT sont décrits en III.4.2 (LSSD) et III.4.3 (LSSD et BILBO). On peut égalemnet citer le système PROSPECT [LaC86], qui s'intéresse plus généralement aux techniques de Scan Path.

III.4.2 Vérificateur/analyseur de règles LSSD

L'objectif du vérificateur/analyseur de règles de DFT, développé par HORSTMANN à l'université de Syracuse en 1983 [Hor83], [Hor84], est l'automatisation du processus de conception adaptée au test, par des méthodes d'I.A.

Le principe de base de ce système est de vérifier que les règles de DFT qui lui sont fournies sont respectées par le dispositif à analyser. Si des règles sont transgressées, le système est capable de suggérer des alternatives pour la transformation en conséquence du schéma du dispositif.

Les caractéristiques de ce système sont les suivantes :

- il se restreint aux règles LSSD
- il est relativement indépendant de la technologie de fabrication du circuit (mis à part quelques caractéristiques permettant de tenir compte d'éléments comme les "OU câblés", les "ET câblés", etc.
- il peut générer des informations de contrôle, utilisables ultérieurement pour une génération automatique de vecteurs de test
- il accepte en entrée une description du circuit aux niveaux structurel, fonctionnel, et comportemental
- il est basé sur le calcul des prédicats du premier ordre (le système est implémenté en PROLOG [CIM85]).

Ce système comprend deux types de clauses, qui sont des clauses de description d'une part, et des clauses d'inférence d'autre part.

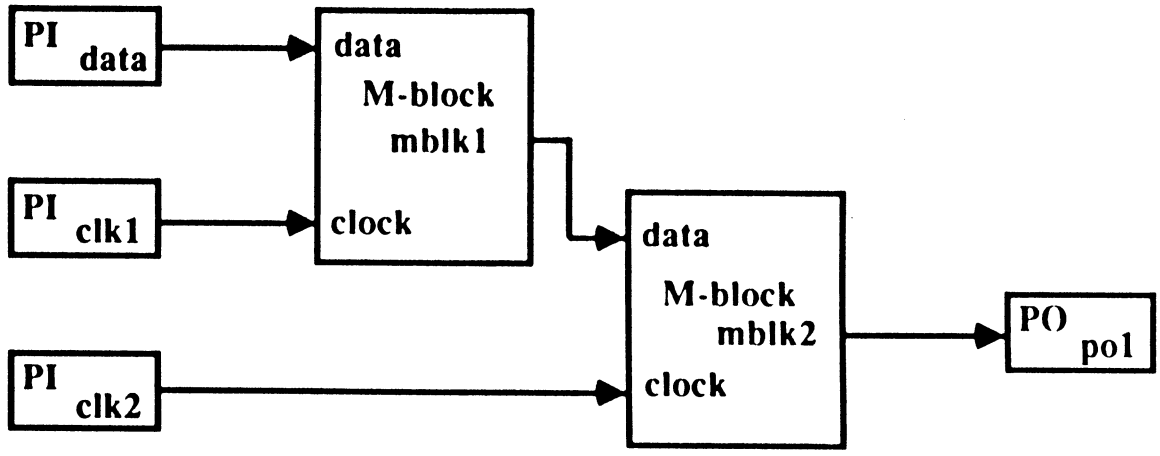
Les clauses de description concernent la structure, la fonction et le comportement du circuit étudié.
Dans le cas du schéma présenté en figure III.5 [Hor84], on peut identifier deux catégories de clauses de description.
Les clauses de type (1) représentent la fonction d'un noeud et de ses interconnexions. Elles sont générées automatiquement par un préprocesseur à partir de la description structurelle ou comportementale choisie, et sont de la forme suivante :

nom_noeud (fonction, liste_noeuds_amont, liste_noeuds_aval)

Les clauses de type (2) représentent le comportement spécifique d'un noeud, définissent les conditions spéciales relatives aux différentes broches d'entrée/sortie, et peuvent également traduire le comportement de la fonction réalisée par un noeud donné.

Les clauses d'inférence ont pour rôle essentiel de vérifier la cohérence des informations fournies au système relativement aux règles de DFT, et se divisent en cinq catégories :

- clauses de vérification de connexion entre deux noeuds
- clauses de vérification de la fonction exécutée par un noeud




```
(1)  data(pi,[],[c(mblk1,data,_)]).
(1)  clk1(pi,[],[c(mblk1,clock,_)]).
(1)  clk2(pi,[],[c(mblk2,clock,_)]).

(2)  clk1(clock).
(2)  clk2(clock).

(1)  mblk1(m-block,[c(data,data,_),c(clk1,clock,_)],[c(mblk2,data,_)]).
(1)  mblk2(m-block,[c(mblk1,data,_),c(clk2,clock,_)],[c(po1,_,_)]).

(1)  po1(po,[c(mblk2,_,_)],[]).
```

Figure III.5. Exemple de schéma et description correspondante

- clauses d'évaluation de fonction
- clauses de vérification du comportement d'un noeud suivant le type de signal reçu
- clauses d'évaluation de la fonction réalisée au niveau des broches

Le fonctionnement global du vérificateur/analyseur est le suivant :

- *Etape 1 :*
  un préprocesseur, ou un concepteur, décrit le circuit en termes de clauses PROLOG
- *Etape 2 :*
  l'ensemble des règles de DFT (avec séquences de contrôle et règles de transformation) à utiliser pour ce type de conception est identifié et fourni au vérificateur

- *Etape 3 :*
  appel à l'interpréteur PROLOG choisi pour la vérification proprement dite
- *Etape 4 :*
  si des sous-spécifications du modèle sont identifiées, le système procède soit par formulation d'hypothèses, soit par consultation du concepteur
- *Etape 5 :*
  chaque fois que la vérification d'une règle se solde par un échec, cette règle est transmise à l'analyseur de DFT qui identifie la partie du schéma à corriger
- *Etape 6 :*
  après analyse, le système, ou le concepteur, corrige le schéma de façon à le rendre conforme aux règles de DFT
- *Etape 7 :*
  itération des étapes 3 à 6 jusqu'à ce que toutes les règles aient été prouvées (au sens de PROLOG)
- *Etape 8 :*
  transmission des séquences de contrôle aux ATPG ou aux testeurs

III.4.3 ESTA (vérificateur de règles LSSD et BILBO)

ESTA (Expert System for Testability Automation) [CGG88], développé par CAMURATI et al. à l'Institut Polytechnique de Turin, est un système permettant de vérifier le respect des règles de DFT dans la conception de circuits VLSI.
Ces règles sont relatives aux méthodologies LSSD et BILBO. En ce sens, ESTA, écrit en PROLOG [CIM85], est proche du système décrit au § III.4.1, encore que plus perfectionné puisqu'il exploite la hiérarchie de conception du circuit à vérifier, à la fois pour la représentation des connaissances et les stratégies de contrôle.

Les connaissances sont représentées de façon hybride, à l'aide de règles de production et de schémas ("frames"). Pour ce faire, un langage spécifique, nommé ProTest (Prolog for Testability) a été développé à partir du langage PROLOG.
ProTest permet de tenir compte du concept de hiérarchie, et les primitives de conception sont décrites à l'aide de clauses PROLOG, alors que les "frames" sont utilisés pour décrire des modules d'un niveau d'abstraction plus élevé, ainsi que leur interconnexion.
Deux types de connaissances coexistent dans ESTA : des "connaissances à court terme", décrivant le circuit, et des "connaissances à long terme", représentant les règles LSSD et BILBO.

La vérification de la testabilité se fait en trois étapes, de façon à utiliser la hiérarchie de représentation. On distingue les modules primitifs ("feuilles" dans la hiérarchie de description) et les modules génériques (décomposables en modules génériques de niveau

inférieur ou en modules primitifs).
La hiérarchie de décomposition est exploitée dans le but de limiter la vérification proprement dite du respect des règles de DFT aux modules primitifs eux-mêmes et à leurs interconnexions. Ainsi, cette vérification n'est effectuée qu'une fois pour chaque module primitif, même si plusieurs de ses instances sont utilisées dans le circuit global.

Les trois étapes de vérification de la testabilité sont les suivantes :

- *Expansion* d'un module générique de niveau *i*, en le remplaçant par les interfaces des modules de niveau *i-1* dont il est composé
- *Vérification* d'un module primitif par activation des règles de vérification des méthodologies de DFT
- *Abstraction* d'un module - générique ou primitif - de niveau *i*, par reconstruction de ses interfaces avec le module générique de niveau *i+1* à la description duquel il participe.

Ces trois étapes ne se déroulent pas successivement, mais sont invoquées selon un parcours choisi du graphe correspondant à la décomposition hiérarchique du circuit à vérifier.
Ainsi, l'exemple présenté en figure III.6 [CGG88] est traité, selon un parcours "en profondeur d'abord" ("depth-first"), comme suit :

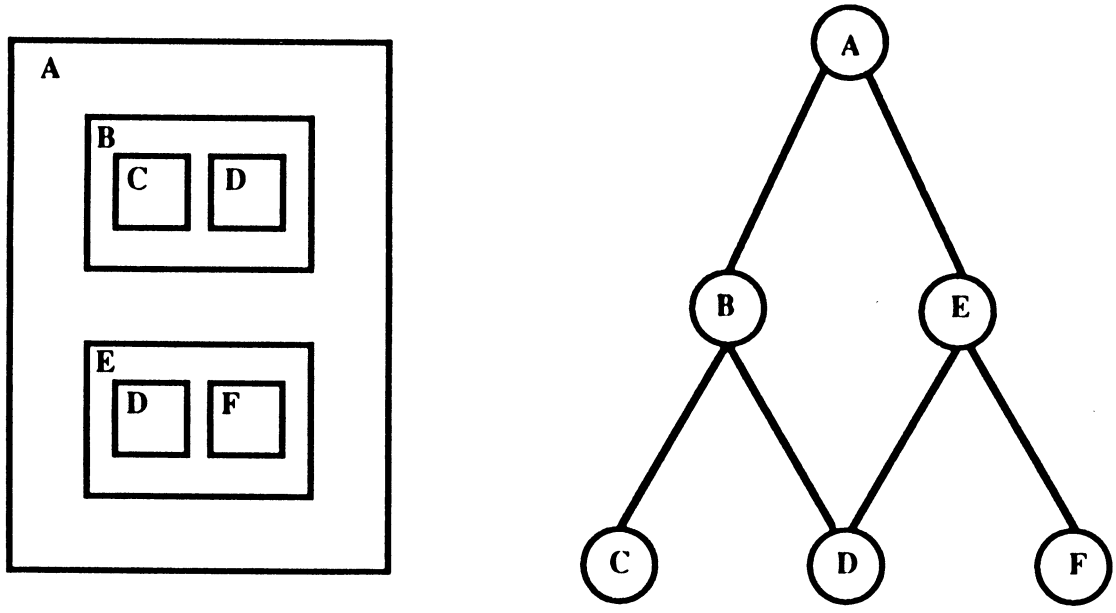


Figure III.6. Exemple de décomposition hiérarchique et graphe correspondant

- visite des noeuds A, B et C, successivement

- C primitif ==> vérification(C)
- abstraction(C)
- D primitif ==> vérification(D)
- abstraction(D)
- abstractions de C et D effectuées ==> expansion(B)
- vérification(B)
- abstraction(B)
- F primitif ==> vérification(F)
- abstraction(F)
- abstractions de D et F effectuées ==> expansion(E)
- vérification(E)
- abstraction(E)
- abstractions de B et E effectuées ==> expansion(A)
- vérification(A)

Notons qu'un module générique, une fois vérifié, peut être considéré comme un module primitif du point de vue de la vérification de sa testabilité : on comprend alors que le gain apporté par l'exploitation de la hiérarchie dans le processus de vérification peut devenir très important dans le cadre d'une conception de circuits modulaire et structurée.

III.5 CONCLUSION

Plusieurs systèmes experts ont été présentés dans ce chapitre, qu'ils soient dévolus à une conception automatique, donc correcte par construction une fois le processus de conception lui-même validé, ou à une conception testable.
Il n'y a pas lieu de décréter que l'un est meilleur que l'autre : chaque système a ses particularités, et répond à des besoins spécifiques. L'intérêt, déjà constaté à travers les priorités que s'assignent les développeurs d'outils de CAO et de compilateurs de silicium, réside plutôt dans la possibilité de disposer d'un ensemble cohérent d'outils.

On peut remarquer, au vu des références bibliographiques citées dans ce chapitre, que l'introduction de techniques d'I.A. pour la conception de circuits ou systèmes intégrés est assez récente. Ceci peut s'expliquer en partie par une certaine méfiance des concepteurs, mais surtout par le fait que ces techniques implémentent des approches de conception elles-mêmes récentes, comme l'approche DFT. En ce sens, les systèmes à base de connaissances permettent une utilisation plus facile des méthodes de conception adaptée au test, et contribuent à leur plus large diffusion.

IV. SBC POUR LA SIMULATION ET LA GENERATION DE VECTEURS

IV.1 INTRODUCTION

La génération de vecteurs de test se conçoit principalement pour la *détection* de fautes. Il s'agit de fautes - résiduelles - de conception pour les tests de mise au point, et de fautes de fabrication pour les tests dits "go/nogo" (tests de fin de fabrication). La génération de vecteurs de test pour ce dernier type de validation a été la plus fréquemment et la plus longuement traitée [LCG89].

Il a été dit dans l'introduction à la première partie que le problème de la détection de fautes - donc de la génération de vecteurs de test - est un problème NP-complet, même dans le cas de circuits combinatoires. Il s'agit donc d'un problème pour lequel l'emploi de techniques d'I.A., ou tout au moins d'un certain nombre de méthodes heuristiques, peut apporter un gain réel en efficacité.

Une génération exhaustive de tous les vecteurs possibles n'est évidemment pas raisonnablement envisageable pour des circuits complexes. C'est l'une des raisons[6] pour lesquelles des *modèles de fautes* sont utilisés de fac'on à restreindre l'espace de recherche.

De nombreuses méthodes de génération automatique de vecteurs de test ont été développées (cf. § IV.2). Ces méthodes, et les différents algorithmes qui les mettent en application, doivent être évalués non seulement du point de vue de leur coût, mais aussi et surtout du point de vue de l'analyse de la *qualité des tests générés*, encore appelée *taux de couverture de fautes*.

Cette analyse est généralement effectuée par *simulation de fautes* - dont ce n'est pas pour autant la seule finalité - (cf. § IV.3).

La suite de ce chapitre sera donc consacrée à la présentation de systèmes utilisant des techniques d'I.A. pour assurer les tâches de génération automatique de vecteurs de test et de simulation. Nous essaierons d'y dégager les apports spécifiques de ces techniques à de telles tâches.

---

6. Une autre raison en est la possibilité d'évaluer le taux de couverture de fautes.

IV.2 GENERATION DE VECTEURS

IV.2.1 Généralités

La génération de vecteurs de test est la tâche qui consiste à définir les valeurs à imposer aux entrées primaires d'un circuit, pour exhiber au niveau de ses sorties primaires les erreurs manifestant la présence de fautes dans le circuit.
On peut distinguer deux grandes classes de méthodes de génération de vecteurs de test, qui sont les *méthodes déterministes* et les *méthodes aléatoires*.

Les méthodes entrant dans la première catégorie sont dites déterministes car elles permettent de générer des vecteurs manifestant des fautes préalablement déterminées et modélisées. Ces méthodes se subdivisent à leur tour en différentes classes, suivant les modèles de fautes et le niveau d'abstraction concernés.
Ainsi, il existe des *méthodes déterministes fonctionnelles*, pour lesquelles les modèles de fautes sont relatifs à une *fonction* du circuit, exécutée de façon incorrecte [ThA78], [ThA79], et des *méthodes déterministes structurelles*, pour lesquelles les modèles de fautes sont relatifs à la *structure* du circuit [Cou81], à différents niveaux d'abstraction possibles (blocs, portes logiques, transistors).

Les méthodes aléatoires [DaT79] consistent à appliquer un certain nombre de combinaisons de vecteurs d'entrée, générés de façon aléatoire, au circuit à tester et à un dispositif de référence (circuit réputé bon ou simulateur), de façon simultanée.

L'analyse des sorties se fait par comparaison avec une référence (résultats de simulation, en général). Les sorties sont comparées soit sous leur forme "brute", soit après compaction (analyse de signature).
Dans le cas des méthodes aléatoires, une analyse statistique des sorties peut remplacer la comparaison à une référence [DaT79]. Cette méthode comporte néanmoins des limitations, en ce sens qu'elle peut résulter en une perte d'information.

Une évaluation de ces différentes méthodes [LCG89] conduit à rechercher de préférence, lorsque la structure du circuit est connue, un test structurel plutôt qu'un test fonctionnel ou aléatoire, d'une part à cause de la nature même des fautes recherchées, et d'autre part pour la robustesse et le bon taux de couverture de fautes qu'il permet d'assurer.
Pour toutes ces raisons, les méthodes déterministes structurelles de génération de vecteurs de test ont fait l'objet de la majeure partie des études théoriques et expérimentales pour leur développement et leur automatisation.

Les trois méthodes les plus connues de génération de vecteurs de test sont le D-algorithme [RBS67], PODEM ("Path-Oriented DEcision Making") [Goe81], et FAN [FuS83]. Ces trois méthodes ne traitent que les circuits combinatoires, mais des extensions aux circuits séquentiels ont été étudiées [BrF76], [Fuj85].

Le principe de fonctionnement de toutes ces méthodes est identique : étant donnée une faute à tester, on essaie de la manifester en remontant du siège de la faute jusqu'aux entrées primaires du circuit ("fault sensitizing"), de façon à générer les vecteurs de test recherchés. Il faut également propager l'effet de cette faute jusqu'aux sorties primaires du circuit ("fault propagation"), de façon à connaître la combinaison des valeurs des sorties en présence de la faute [KiM88].
Il est évident que, si des heuristiques ne sont pas utilisées pour limiter la recherche dans le graphe représentant le circuit, ce procédé de génération de vecteurs de test devient rapidement impraticable. Ce sont ces heuristiques qui vont différencier les trois méthodes citées plus haut.

Les trois principaux algorithmes, dont une présentation succinte est donnée dans la suite, effectuent une génération de vecteurs de test au niveau portes logiques. Les fautes considérées sont du type collage à 0 ou à 1, et, dans tous les cas, seule l'hypothèse de faute simple est considérée.[7]

Le D-algorithme [RBS67] a été conçu pour pallier l'inconvénient des méthodes les plus simples consistant à sensibiliser un seul chemin ("single-path sensitization"), du siège de la faute vers les sorties : en effet, il existe des fautes non redondantes, donc testables, mais pour lesquelles il est impossible de générer des vecteurs de test si un seul chemin doit être sensibilisé à la fois.

Le D-algorithme est donc une méthode de type "multiple-path sensitization". Il est basé sur une algèbre appelée D-calcul [RBS67], prenant ses valeurs dans l'ensemble $\{0,1,D,\overline{D},X\}$, où 0 et 1 sont les valeurs logiques binaires, D (respectivement $\overline{D}$) signifie que l'on observe la valeur 0 (respectivement 1) alors que la valeur correcte est 1 (respectivement 0), et X représente indifféremment la valeur 0 ou 1 ("don't care value").
Nous nous contenterons de présenter ici la façon dont les processus appelés "fault sensitizing" et "fault propagation" sont conduits dans le D-algorithme, des informations plus spécifiques sont données dans [RBS67], [BrF76], et [Fuj85].

Le graphe du circuit est tranformé en un graphe de recherche, dans lequel les noeuds sont constitués par les portes logiques, ainsi que les entrées et sorties primaires, et les arcs représentent les connexions dans le circuit (figure IV.1 [KiM88]).

La génération de vecteurs se déroule de la façon suivante :

---

7. Avec ce que cela comporte de limitations lorsqu'on s'intéresse à des dispositifs aussi complexes que les circuits actuels, qui sont de plus conçus à l'aide de nouvelles technologies (voir à ce sujet la deuxième partie, chapitre V).

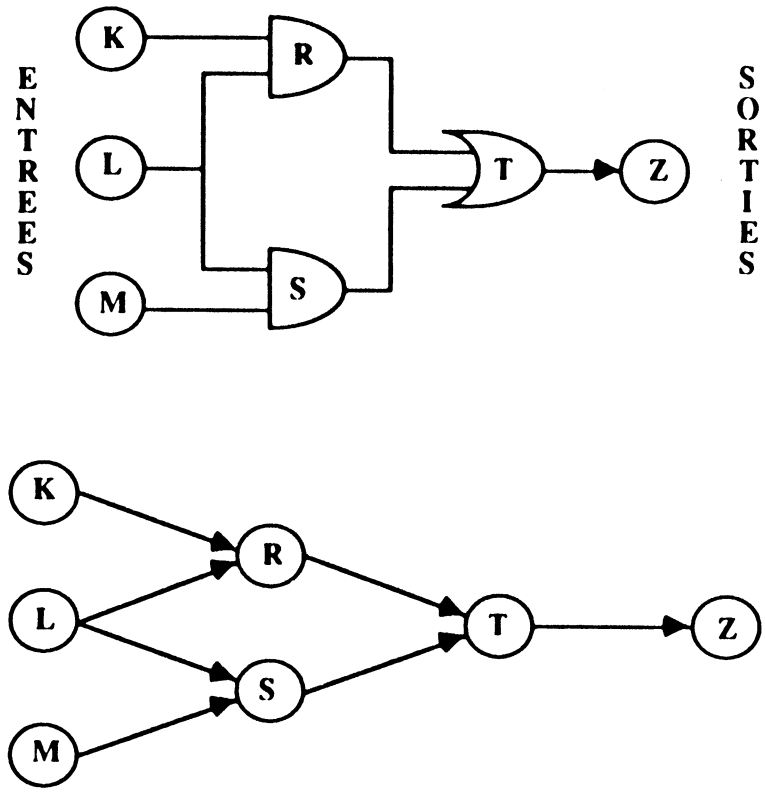


Figure IV.1. Exemple de circuit et graphe de recherche du D-algorithme

- *"Fault sensitizing" :*
  Tous les noeuds sont initialisés à la valeur X, puis la faute à tester est choisie par affectation de la valeur D (respectivement $\overline{D}$) pour un collage à 0 (respectivement 1) au siège de la faute. Ensuite, l'algorithme essaie d'affecter une valeur de l'ensemble $\{0,1,D,\overline{D}\}$ aux noeuds non encore assignés, par implication (en fonction des portes logiques composant le schéma), et par choix arbitraire entre plusieurs valeurs possibles, le cas échéant. Cette étape s'achève lorsqu'un vecteur a pu être généré, ou lorsque tout l'espace de recherche a été exhaustivement parcouru sans résultat, et dans ce dernier cas on peut conclure que la faute à tester est redondante.
  Si des conflits surviennent au cours du processus d'assignation des valeurs, l'algorithme remet en cause ses choix précédents ("backtracking").

- *"Fault propagation" :*
  Le D-algorithme conserve une liste de noeuds appelée D-frontière. Cette liste contient les portes logiques dont les sorties sont non encore spécifiées (valeur X), et dont les entrées ont été assignées à la valeur D ou $\overline{D}$. Le processus consiste à choisir un membre de la D-frontière et à propager les valeurs D et $\overline{D}$ jusqu'à au moins une sortie primaire.

Le D-algorithme permet de générer un vecteur de test pour toute faute de type collage, si cette faute n'est pas redondante. Il présente néanmoins les inconvénients majeurs suivants :

- Il n'existe pas de stratégie d'ordonnancement des tâches de "fault sensitizing" et "fault propagation".
- Il ne permet pas de détecter les conflits au plus tôt, car il ne tient pas vraiment compte du circuit dans sa globalité.
- Les conflits étant dûs à la présence de portes à sortance multiple reconvergente ("reconvergent fan-out"), ils deviennent extrêmement fréquents s'ils doivent être résolus en n'importe quel noeud du graphe.

L'algorithme PODEM [Goe81] a été développé pour améliorer le D-algorithme, notamment concernant le problème du nombre de conflits à résoudre, qui devient rédhibitoire pour des circuits comportant une grande quantité de portes OU exclusif.
Le principe de l'algorithme PODEM consiste à n'assigner des valeurs (donc prendre une décision) qu'aux entrées primaires du circuit, puis à propager ces valeurs par implication, de façon à limiter les remises en cause des choix ("backtracking") au niveau de ces entrées primaires.
Ceci implique un examen exhaustif de tous les vecteurs de test possibles, avec arrêt du processus dès que la faute à tester est rendue détectable par une valeur D ou $\overline{D}$ exhibée sur l'une des sorties primaires.
Dans le pire des cas (faute redondante), il faut examiner toutes les combinaisons possibles des entrées.
PODEM emploie, pour accélérer le processus de "fault sensitizing", certaines heuristiques basées sur les notions d'états contrôleurs de portes logiques, c'est-à-dire l'une des entrées à 0 (respectivement 1) pour une porte ET ou NON-ET (respectivement OU ou NON-OU) [Fuj85].

L'algorithme FAN [FuS83] va encore plus loin que PODEM dans la réduction des remises en cause de choix en cas de conflit. Pour ce faire, des heuristiques sont utilisées de façon à déterminer l'inexistence d'une solution le plus rapidement possible. Il en découle un certain nombre de stratégies que nous ne décrirons pas ici, mais qui sont détaillées dans [FuS83] et [Fuj85].

Un rapide bilan des problèmes posés par la génération de vecteurs de test, ainsi qu'un examen des méthodes existantes, conduisent aux conclusions suivantes [BGG88] :

- l'objectif, à chaque étape du processus, est la manifestation d'une seule faute à la fois
- la sémantique de la conception du circuit est souvent ignorée, du fait de ne considérer que le niveau portes logiques

- la hiérarchie de conception du circuit n'est pas du tout exploitée
- les heuristiques employées sont figées par l'algorithme, et ne peuvent être employées de façon dynamique
- les connaissances accumulées lors d'une étape de génération, pour une faute donnée, ne sont pas réutilisées dans les étapes ultérieures

On peut encore ajouter à ces remarques le fait que les algorithmes existants ont souvent été conçus par leurs auteurs en vue d'une efficacité maximale pour des types particuliers de circuits, ou parties de circuits. Il est donc plus intéressant d'offrir la possibilité d'appliquer différentes méthodes à différentes parties d'un même circuit.

L'emploi de techniques d'intelligence artificielle peut réduire ces inconvénients, justement en permettant d'exploiter au maximum d'une part les informations accumulées lors des différentes sessions de génération, et d'autre part la connaissance relative au circuit lui-même, issue du processus de conception.

Ceci se fait en tenant compte de la hiérarchie de conception (des exemples sont présentés en IV.2.2 et IV.2.3, on pourra consulter également [Kri87] et [Gup86], ou en se fixant comme objectif plusieurs fautes à détecter simultanément (cf. § IV.2.4, voir aussi [SWL86]).
En outre, l'utilisation de techniques d'intelligence artificielle n'est pas limitée au test structurel, puisque des systèmes à base de connaissances pour la génération de vecteurs de test fonctionnel existent également ([LBK88], [RLG89]).

IV.2.2 Le système THESEE

Le système THESEE [DGL87], développé par DELORME et al. pour la société CIMSA SINTRA est un système d'aide à la génération de vecteurs de test pour des cartes composées de circuits numériques. Ce système permet de combiner l'utilisation de procédures algorithmiques et de règles de production, selon le principe de fonctionnement suivant :
Etant donnée une carte à tester, la première étape consiste à effectuer un découpage fonctionnel de la carte en macro-composants (MCs), dont chacun constitue la plus petite entité testable. Ce découpage est réalisé à l'aide d'un système expert.
Ensuite, un générateur automatique de vecteurs de test (ATPG) est utilisé pour générer les vecteurs locaux à chaque MC.
Enfin, la troisième étape, utilisant un système expert, permet d'organiser les séquences de test global de la carte, par identification de chemins de test et des MCs les constituant.

La conception du système THESEE repose sur l'observation de la démarche d'un expert en situation.
La représentation des connaissances dans THESEE est basée sur la notion de modèles hiérarchisés, et la carte à tester est considérée à la fois selon différents points de vue

(physique, fonctionnel, etc.) et selon différents niveaux d'abstraction (carte, boitiers, portes logiques, etc.).
Il existe donc un modèle par point de vue, hiérarchisé en différents niveaux d'abstraction. Chacun des modèles est représenté sous forme de graphe, composé de noeuds, d'arcs et de connecteurs. Notons que la notion de connecteur a été ajoutée à la structure classique de graphe de façon à complètement définir l'environnement des noeuds et des graphes correspondant à chaque niveau d'abstraction : ainsi, tout transfert d'information entre noeuds et graphes se fait obligatoirement à travers les connecteurs.
Un modèle hiérarchisé est vu comme un ensemble de couples noeud/sous-graphe, où un noeud de niveau hiérarchique *i* est développé en sous-graphe de niveau *i-1*.

Le mode d'exploitation de cette modélisation est le suivant :
Le premier système expert (réalisant le découpage fonctionnel) procède par des techniques de recherche de transferts horizontaux (entre points de vue) et verticaux (entre niveaux d'abstraction) dans le réseau hiérarchisé de modèles, alors que le deuxième système expert (organisant les séquences de vecteurs de test générés localement à chaque macro-composant) procède par recherche de chemins de MCs, par détermination des conditions nécessaires pour obtenir les vecteurs aux points de contrôle de la carte, et par propagation de ces vecteurs jusqu'aux points observables (broches de sortie de la carte).

On peut donc identifier trois tâches expertes dans THESEE, qui sont : le découpage de la carte en MCs, l'identification et le classement de chemins de test selon des critères heuristiques fournis par un expert sous forme de règles, et le traitement d'un chemin de test consistant à gérer l'interfaçage avec les ATPGs et les simulateurs [DGL87].

Le système THESEE a été conçu sur machine SYMBOLICS, à l'aide d'outils généraux développés en parallèle. Ces outils sont : un système de représentation, à base d'objets de type "frame", des modèles hiérarchisés, et un système de représentation et de gestion des connaissances expertes, intégrant différents formalismes (frames, prédicats, et règles de production avec variables).
Les extensions prévues pour le système THESEE sont son intégration dans un système de CAO et son utilisation pour des tâches connexes, comme l'analyse de la testabilité de cartes électroniques.

IV.2.3 Le système HITEST

HITEST [Rob83], [Wha83], [Mau83] constitue l'un des outils de l'ensemble de logiciels développé par la société CIRRUS Computers à l'origine, actuellement commercialisé par la société GenRad sous le nom de "System Hilo". HITEST est un système de génération automatique de vecteurs de test utilisant des techniques d'intelligence artificielle. Ce système est composé d'un ensemble de modules, organisés autour d'une base de connaissances comportant la représentation du circuit à tester ainsi que des stratégies de test. Les différents modules sont représentés en figure IV.2 [Par88]. Le système HITEST permet

d'adresser des circuits combinatoires, des circuits conçus en utilisant des techniques de Scan-Path, et des circuits séquentiels structurés, mais n'incluant pas de stratégies de DFT.

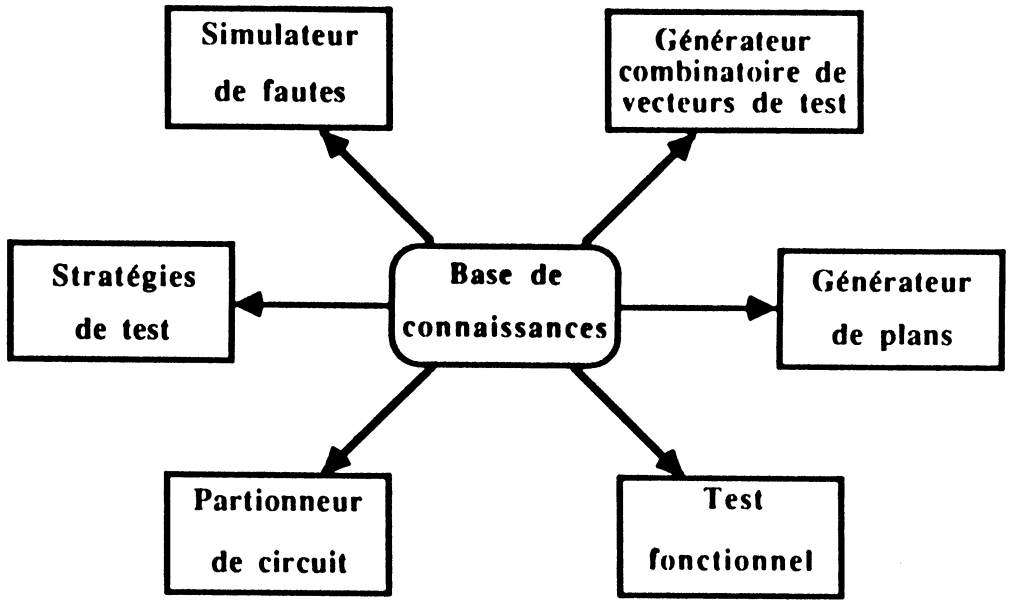


Figure IV.2. Organisation du système HITEST

Le problème de la génération de vecteurs est résolu par HITEST à l'aide d'une approche "diviser pour régner (divide and conquer)", en utilisant les différents modules de la façon suivante :
Les informations sur le circuit lui-même sont utilisées pour le partitionner (module "partitionneur de circuit "), réduisant le problème du test global en sous-problèmes plus facilement solubles. Chacun des composants ainsi obtenus sera traité par le module approprié : les parties combinatoires subiront un traitement purement algorithmique (module "générateur combinatoire de vecteurs de test") basé sur la méthode PODEM de génération de vecteurs de test, alors que les parties contenant des éléments de mémorisation seront contrôlées par le générateur de plans et seront soumises à un test fonctionnel.
Les séquences de test du circuit global seront ensuite organisées suivant des stratégies de test. Le taux de couverture de fautes permis par ces séquences pourra enfin être évalué par rapport aux modèles de fautes assignés comme objectif (module "simulateur de fautes").

Dans la version actuelle de HITEST, tous les modules ne sont pas encore disponibles. Néanmoins, une description de la base de connaissances, des stratégies de test, et du générateur de plans existe dans [Par88].
Les informations sur le circuit à tester, ainsi que les stratégies de test à suivre sont fournies par l'utilisateur, soit directement à HITEST, si cet utilisateur est suffisamment expérimenté,

soit par l'intermédiaire du système expert KNAC ("KNowledge ACquisition system"), qui fournit une aide à la génération de la base de connaissances de HITEST.
Ces informations seront organisées et représentées sous forme de "frames" composés de triplets "objet-attribut-valeur". Ceci permet une représentation hiérarchique, bénéficiant des avantages des représentations orientées-objets (notamment l'héritage d'attributs, et l'existence de "facettes" ou procédures attachées aux attributs, permettant la gestion de données incomplètes.

Le générateur de plans est utilisé pour contrôler l'évolution des états intermédiaires dans les circuits séquentiels. Il est basé sur une analyse de type "fins et moyens (means-end analysis)", basée sur l'évaluation des différences entre l'état courant et l'état final à atteindre. Ces différences sont progressivement réduites en appliquant l'ensemble des opérations dont la séquence est planifiée par le générateur.

Le principal intérêt de HITEST réside dans son utilisation pragmatique des approches orientées intelligence artificielle, qui se concrétise par l'exploitation industrielle du système par la société GenRad.

IV.2.4 C-algorithme : génération concurrente

Les travaux de YAU au Centre de Recherches AT&T de Princeton [Yau86] ont permis de développer une nouvelle méthode de génération de vecteurs de test, nommée C-algorithme ("Concurrent test generation Algorithm"). Cette méthode se conçoit comme une amélioration des algorithmes connus de génération de vecteurs (D-algorithme, PODEM, FAN, etc.), en ce sens que son processus de génération de vecteurs ne se fixe pas comme objectif *une* faute à tester à la fois (souvent choisie arbitrairement d'ailleurs), mais *toutes* les fautes non encore détectées.

Cette notion de base est somme toute peu nouvelle, puisque l'idée de détecter en une seule fois toutes les fautes simples sur un même chemin de sensibilisation date de la fin des années soixante [Arm66]. Néanmoins, l'intérêt de l'approche de YAU réside dans la façon dont cette notion est exploitée par le C-algorithme, qui permet de réduire, voire parfois d'éliminer, les inconvénients des méthodes non concurrentes de génération de vecteurs de test, dont les principaux sont les suivants :

- la taille de l'ensemble des vecteurs générés dépend de l'ordre dans lequel on veut détecter les fautes.
  En effet, considérons l'exemple de la figure IV.3 [Yau86], et supposons, dans un premier temps, que l'on veuille obtenir sur la sortie primaire "a" un "1" logique. Le vecteur d'entrée "10101010" permet d'atteindre cet objectif, créant ainsi 4 chemins sensibles (f-b-a, h-c-a, j-d-a, et m-e-a), sur lesquels il est possible de détecter 9 fautes de type collage à 0.

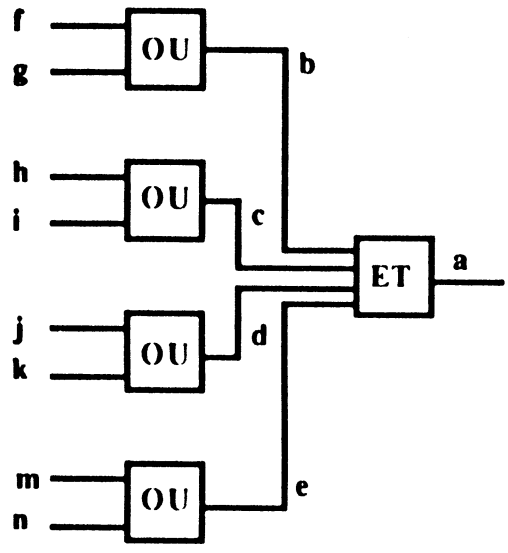


Figure IV.3. Exemple d'application du C-algorithme

Les 4 fautes de type collage à 0 restant à détecter peuvent l'être à l'aide d'un seul vecteur d'entrée ("01010101"), si l'on considère ces 4 fautes de façon *concurrente*. Par contre, une approche séquentielle aurait nécessité 4 vecteurs différents ("01101010", "11011010", "11110110", "11111101") pour tester ces 4 fautes, chacune étant considérée successivement.
Notons par ailleurs que, si l'on avait voulu d'abord obtenir sur la sortie primaire "a" un "0" au lieu d'un "1" logique, on n'aurait pu sensibiliser que deux chemins (par exemple f-b-a et g-b-a avec le vecteur "00101010") et ne détecter que 4 fautes de type collage à 1.

- il peut arriver que l'on détecte des fautes déjà détectées par un vecteur précédemment généré (le degré de redondance est estimé à 20% [Yau86]).
- lorsque la faute fixée comme objectif est une faute redondante (indétectable), trop de temps est perdu à essayer de générer un vecteur inexistant.
- la taille de l'ensemble des vecteurs générés est non optimale, ce qui cause une augmentation du temps de simulation de fautes.

L'approche de génération "concurrente", en considérant à chaque étape l'ensemble des fautes non encore détectées, génère au contraire des vecteurs permettant de sensibiliser plusieurs chemins à la fois, ce qui permet une couverture (donc une détection) possible d'un plus grand nombre de fautes simultanément, et résulte en une génération d'un ensemble de vecteurs de test réduit, sinon minimal ([Yau86] montre que cette réduction peut atteindre 50%).

Les techniques d'intelligence artificielle introduites dans la méthode de YAU sont relatives à la modélisation du problème et aux stratégies de recherche dans l'espace des solutions possibles.

La représentation du problème est effectuée sous forme de graphe ET-OU [Ric87], ce formalisme étant particulièrement adapté à la méthode utilisée - génération "concurrente" de vecteurs -, qui procède par réduction progressive du problème (décomposition en sous-problèmes jusqu'à atteindre un ensemble de primitives), et dont le principe est une propagation arrière ("backtracing") à partir des sorties primaires du circuit jusqu'à trouver les valeurs binaires à imposer aux entrées primaires.

Le graphe ET-OU est défini implicitement à partir de la structure du circuit considéré, et des modèles de fautes à détecter (en l'occurrence, les collages de lignes à 0 ou à 1), comme le montre l'exemple de la figure IV.4 [Yau86].

Dans ce graphe, les noeuds constituent les buts ou sous-buts à atteindre, et chacun des chemins possibles constitue une solution, mais il n'est évidemment pas nécessaire de toutes les énumérer (ce qui équivaudrait à un test exhaustif). Des alternatives ayant pour objet d'appliquer la valeur logique 1 (respectivement 0) sur toutes les entrées d'une porte OU (respectivement ET) peuvent d'emblée être éliminées, par exemple, puisqu'elles ne sont d'aucun apport pour la détection de fautes, surtout lorsqu'il s'agit de circuits ne comportant pas de porte à sortance multiple (circuits "fan-out free").

Le mode d'exploitation du graphe ET-OU, ainsi que la stratégie de génération de vecteurs de test suivie par le C-algorithme se résument dans les étapes suivantes, effectuées tant qu'il reste des fautes non encore détectées, ou que l'ensemble des solutions n'a pas été exhaustivement parcouru :

- **Etape 1 :**
  estimer le nombre maximum de nouvelles fautes détectables par le prochain vecteur, à l'aide d'heuristiques

- **Etape 2 :**
  chercher, en utilisant le graphe ET-OU implicite, la solution la plus efficace, toujours à l'aide d'heuristiques. Considérer d'abord un seul chemin de sensibilisation, puis des chemins multiples s'il subsiste des fautes non encore détectées

- **Etape 3 :**
  effectuer une simulation de fautes avec les tests générés, et mettre à jour l'ensemble des fautes non encore détectées

- **Etape 4 :**
  s'il s'avère que le test généré a permis la détection de nouvelles fautes, l'ajouter à l'ensemble des tests, sinon l'éliminer

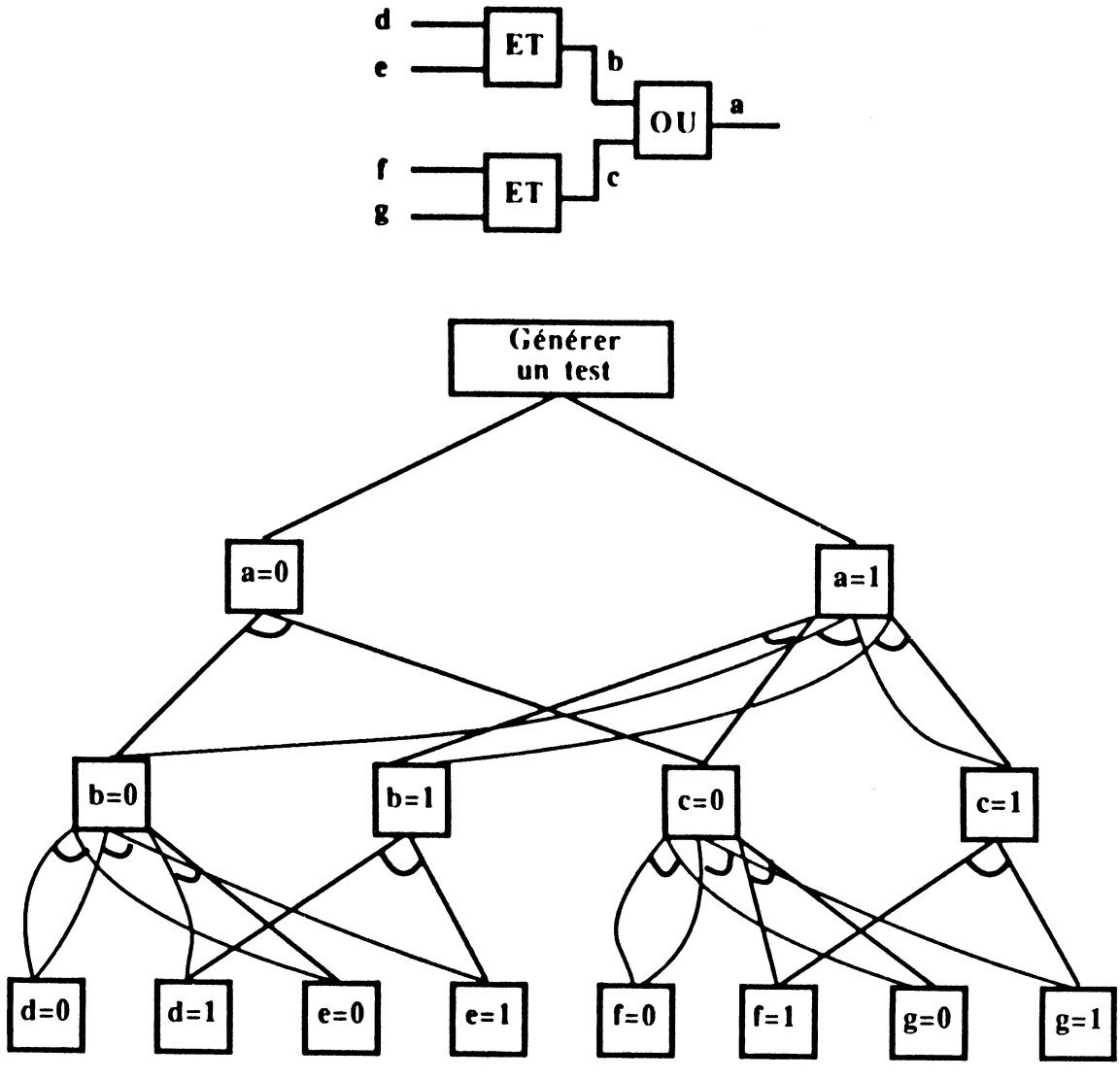


Figure IV.4. Exemple de circuit simple et graphe ET-OU correspondant

- **Etape 5 :**
  si tous les chemins de sensibilisation de multiplicité *i* ont été parcourus, déclarer comme redondantes toutes les fautes non détectées sur les lignes ayant moins de *i* chemins vers les sorties primaires, et éliminer ces fautes de l'ensemble des fautes non encore détectées.

Différentes heuristiques peuvent être utilisées pour guider la recherche dans l'espace des solutions possibles. Il peut s'agir de fonctions d'évaluation de coût, comme celle employée dans l'algorithme AO* [Nil80], ou de fonctions plus spécifiques comme la fonction "COVER" dont on trouvera le détail dans [Yau86], et dont le but est d'estimer le nombre maximum de fautes non encore détectées que l'application d'un vecteur de test particulier

peut exhiber ("couvrir").

Les principales qualités de la méthode de YAU se résument dans les points suivants :

- compacité des tests générés
- production d'un ensemble *ordonné* de vecteurs, suivant un taux de couverture de fautes décroissant, ce qui a pour double avantage d'une part de réduire le temps de simulation de fautes, et d'autre part d'éliminer d'abord les tests ayant le plus faible taux de couverture, s'il est nécessaire - pour des raisons de limitation de mémoire dans l'équipement de test, par exemple - de réduire l'ensemble des vecteurs à appliquer
- efficacité de l'algorithme de génération, grâce à la détection dynamique des fautes redondantes, par exemple.

IV.3 SIMULATION

IV.3.1 Généralités

La simulation de fautes sera vue principalement ici en tant que moyen d'évaluer le taux de couverture de fautes assuré par une méthode de génération automatique de vecteurs de test. Elle consiste à "injecter", dans la représentation du circuit à tester, des fautes préalablement établies suivant des modèles de fautes précis, et à simuler le comportement du circuit en présence de ces fautes. Ces modèles dépendent du niveau de représentation du circuit, et de la connaissance que l'on possède de sa description.

Les types de test auxquels nous nous intéressons ici (test de mise au point et test "go/nogo") supposent une connaissance suffisante de la structure du circuit pour que des modèles de fautes relatifs aux niveaux logique et électrique soient utilisables, dans le cadre d'un test structurel.
Les modèles du niveau logique sont les collages de lignes aux valeurs de potentiel 0 ou 1 ("stuck-at-0" et "stuck-at-1"), ce sont les plus couramment utilisés. Néanmoins, ces modèles étant insuffisants pour représenter la réalité des fautes pouvant survenir, notamment pour les circuits réalisés en technologie MOS [GCV80] (voir également à ce sujet le chapitre V de la deuxième partie), d'autres modèles correspondant à un niveau de description en termes de transistors, ont été développés et étudiés : il s'agit de transistor collé ouvert ("stuck-open"), collé fermé ("stuck-on"), de coupure ("open") et de court-circuit ("short").

Bien que ces modèles aient pour but de restreindre le nombre de fautes à simuler, la simulation de fautes reste un processus très long et très coûteux.
D'autres notions sont donc utilisées pour diminuer ce coût. Ainsi, les notions d'équivalence et de dominance de fautes [BrF76] sont-elles adoptées pour ne simuler qu'une faute par classe d'équivalence ou de dominance, les autres étant forcément détectées par les mêmes vecteurs de test que le représentant de la classe.

Diverses méthodes de simulation de fautes ont été développées, toutes ayant pour but une plus grande efficacité et un moindre coût du processus de simulation. Il s'en dégage trois méthodologies principales, qui sont : la *simulation parallèle*, la *simulation déductive*, et la *simulation concurrente* [BrF76], [Fuj85].

La simulation parallèle utilise les propriétés des instructions logiques manipulant des bits : partant du constat que ces instructions (ET, OU, OU exclusif) manipulent tous les bits d'un mot de façon identique, et indépendamment les uns des autres, les algorithmes de simulation parallèle permettent de traiter n différentes fautes simultanément, pour des mots de n bits. Si l'on veut donc simuler m différentes fautes pouvant survenir dans un mot de n bits, il suffira d'effectuer m/n simulations au lieu de n, l'injection de fautes se faisant par l'intermédiaire de "masques".

La simulation déductive effectue une simulation du comportement correct du circuit, et déduit pour chaque état courant - correct - du circuit toutes les fautes pouvant être détectées sur n'importe quelle connexion, par propagation de *listes de fautes* associées à chaque connexion. Ce principe permet d'effectuer la simulation en une seule passe, plus longue qu'une passe de simulation parallèle, certes, mais entraînant une très nette amélioration du temps global nécessaire, pour des résultats équivalents.

La simulation concurrente s'effectue également en une seule passe. Elle est basée sur une très simple constatation, qui est que la différence de comportement entre un circuit correct et un circuit comportant une faute est très ténue. La simulation concurrente consiste donc à simuler le comportement correct du circuit, et à n'en simuler - de façon concurrente - le comportement "fautif" que lorsque la différence entre ces deux comportement est effectivement observable, ce qui réduit considérablement le temps de calcul nécessaire.

Ces méthodes permettent d'adresser les circuits combinatoires, ou, à défaut, des circuits séquentiels conçus de façon à en faciliter la testabilité, notamment par des techniques de Scan-Path (cf. chapitre III).

Ce préambule à la simulation de fautes étant effectué, il s'agit d'identifier en quoi des techniques d'I.A. peuvent-elles aider à développer des méthodes de simulation de fautes moins coûteuses.
Deux points principaux justifient l'utilisation de méthodes orientées intelligence artificielle : il s'agit d'une part de l'emploi d'heuristiques, de façon dynamique au cours du processus de simulation, dans le but de restreindre la masse de calcul à effectuer, et d'autre part de l'utilisation la plus intensive possible des notions de partitionnement et de hiérarchie, pour effectuer des simulations partielles.

Le premier point (emploi d'heuristiques), préconisé par des chercheurs comme SCHULZ [AKM88], a pour but principal la restriction du nombre de lignes pour lesquelles une

simulation de fautes explicite doit être effectuée (il s'agit surtout des "pieds de fan-out", pour les portes à sortance multiple). Ces heuristiques sont citées dans [AKM88]. On en retiendra surtout ici l'aspect dynamique, puisque les critères d'applicabilité de ces heuristiques sont mis à jour au fur et à mesure de l'avancement du processus de simulation.

Le deuxième point (exploitation de la hiérarchie de description du circuit), préconisé par ROGERS [AKM88], permet de maîtriser l'aspect fortement combinatoire du processus de simulation de fautes. Il est également intéressant en ce sens qu'il permet d'adresser des circuits séquentiels, et de mixer différents niveaux de représentation et de simulation, dans le cadre d'un processus réparti.

Les deux travaux présentés dans la suite (§ IV.2.2 et § IV.2.3) illustrent ces deux apports essentiels de l'I.A. au processus de simulation.

IV.3.2 Travaux de GULLICHSEN

Le travail réalisé par Eric GULLICHSEN, à l'université de Victoria (Canada) [Gul85], consiste en une simulation de circuits à l'aide de méthodes heuristiques, élaborée en utilisant le langage PROLOG [CIM85].

GULLICHSEN résout le problème de la représentation des connaissances sur les caractéristiques fonctionnelles et physiques des circuits de la façon suivante :

- Les portes logiques sont représentées par leurs tables de vérité décrites comme des axiomes en langage PROLOG.

  Exemple : porte ET

  ```
  AND (IN(0,0) , OUT(0)).
  AND (IN(0,1) , OUT(0)).
  AND (IN(1,0) , OUT(0)).
  AND (IN(1,1) , OUT(1)).
  ```

- Les circuits combinatoires sont représentés comme un réseau de portes logiques, et font l'objet de clauses dont la partie gauche représente le circuit et la partie droite sa description en termes de portes logiques interconnectées.

Le processus de simulation est vu en tant que processus de satisfaction de contraintes imposées par les spécifications de fonctionnement du circuit, et fonctionne comme un "démonstrateur" de théorèmes.
La valeur du travail de GULLICHSEN tient aussi au fait qu'il permet, par des considérations d'abstraction et de hiérarchie, de simuler des circuits complexes en les découpant en plusieurs modules.
Ce démonstrateur de théorèmes permet en outre de faire de la simulation avec des valeurs inconnues ("don't know values") ou non significatives ("don't care values") qui constituent

des indéterminations, simplement par ajout d'axiomes supplémentaires :

Exemple : porte ET

Soient X les valeurs non significatives et U les valeurs inconnues. Les axiomes supplémentaires sont les suivants :

AND (IN(0,U) , OUT(0)).
AND (IN(U,0) , OUT(0)).
AND (IN(1,U) , OUT(U)).
AND (IN(U,1) , OUT(U)).
AND (IN(U,U) , OUT(U)).

Le démonstrateur de théorèmes développé par GULLICHSEN n'est pas uniquement dédié à la simulation fonctionnelle d'un circuit. Il permet également la détermination des valeurs à injecter sur les entrées primaires pour obtenir une configuration particulière des sorties. Le démonstrateur peut donc être utilisé aussi bien en tant que simulateur qu'en tant que générateur de vecteurs de test ([Gul85] indique à ce sujet la possibilité d'une implémentation du D-algorithme [RBS67] basée sur le démonstrateur).
La présence de ce double aspect (simulation "aval" et simulation "amont")[8] dans le même système bénéficie largement des possibilités intrinsèques du langage PROLOG [CIM85], notamment la bidirectionnalité de la propagation des contraintes et le non déterminisme. En revanche, ceci induit une efficacité du processus très variable, suivant que l'on effectue une simulation "amont" ou "aval" : si les notions de causalité et d'implication facilitent beaucoup la simulation "aval", le non déterminisme de la simulation "amont" peut rapidement devenir très coûteux à cause du nombre de retour arrière ("backtracking") au niveau des choix d'alternatives.

L'utilisation de stratégies heuristiques (critères de sélection et d'ordonnancement des buts à prouver) permet de réduire le coût de cette simulation "amont" en limitant le plus possible les opérations de retour arrière.
Toutes les stratégies utilisées dans le démonstrateur de GULLICHSEN sont orientées vers les buts ("goal-directed"), le terme "but" étant entendu au sens de PROLOG [CIM85]. Les heuristiques sont donc développées de façon à sélectionner, parmi toutes les alternatives présentes à une étape donnée du processus de simulation, le "meilleur" but à prouver, c'est-à-dire le but (ou sous-but) susceptible d'entraîner le moins de retour arrière possible.

---

8. On appellera simulation "aval" le processus de simulation des sorties en fonction des entrées, et simulation "amont" le processus de génération des entrées en fonction des sorties à obtenir.

Les heuristiques principalement utilisées sont les suivantes :

- Sélection des sous-buts (modules) à prouver :

  * *Simuler d'abord les modules ayant le moins le sorties libres.*
    Une sortie est liée (non libre) soit s'il s'agit d'une sortie primaire dont la valeur a été spécifiée avant le processus de simulation, soit s'il s'agit d'une sortie de module connectée à l'entrée d'un module déjà simulé.
    Cette stratégie permet de déduire les valeurs de simulation avec le plus de déterminisme possible, et évite donc des choix qu'il faudrait éventuellement reconsidérer ("backtracking").

  * *Parmi les modules ayant le moins de sorties libres, simuler d'abord ceux ayant également le moins d'entrées libres.*
    Cette stratégie permet d'ordonner le choix de modules classés de façon équivalente par la première heuristique. Son but est également de découvrir les contradictions (pouvant nécessiter un "backtracking") le plus tôt possible au cours du processus de simulation.

  * *Lorsqu'un ensemble critique d'entrées a été déduit de façon déterministe pour un module donné, choisir ce module comme prochain module à simuler.*
    Un ensemble critique d'entrées, pour un module donné, est l'ensemble minimal des valeurs d'entrée permettant de déterminer les sorties du module, quelles que soient les valeurs des autres entrées.
    Cette heuristique permet d'utiliser autant que possible le déterminisme de la simulation.

- Sélection des axiomes (portes logiques) à vérifier :
  Typiquement, cette tâche est non déterministe lorsque l'on doit attribuer des valeurs aux entrées non critiques d'une porte dont la valeur de sortie est connue.
  Ces choix étant effectués à un niveau *local*, la stratégie heuristique consiste à considérer d'abord ceux qui peuvent garantir la plus grande *cohérence globale*.
  Cette stratégie est basée sur des calculs de syndrômes de fonctions booléennes, suivant la technique développée par BRGLEZ [Brg83].

Les stratégies heuristiques employées dans le démonstrateur de GULLICHSEN permettent d'en améliorer les performances. [Gul85] utilise, pour les évaluer, une unité indépendante du langage d'implémentation (ici, PROLOG) : le nombre de GAMs ("Gate Axiom Matches", ou encore nombre d'axiomes correspondant à des portes logiques appariés). En effet, le nombre de GAMs effectués au cours d'une simulation permet de caractériser le nombre de "backtracking", et admet comme borne inférieure le nombre de portes composant le circuit simulé.
A titre d'exemple, [Gul85] présente les résultats de simulation de l'UAL/générateur de fonctions 74181. Ce circuit, à 14 entrées et 8 sorties, est composé de 60 portes logiques. La

borne inférieure du nombre de GAMs est donc évaluée à 60 à l'aide d'un programme PROLOG, préalablement à la simulation proprement dite.
L'expérimentation a consisté à retrouver les valeurs des entrées nécessaires à l'obtention d'un mot de sortie sélectionné aléatoirement, parmi tous les mots obtenus avec des valeurs d'entrées générées elles-mêmes de façon aléatoire. Les résultats sont donnés en figure IV.5.

Condition de simulation	# moyen de GAMs nécessaires
borne inférieure	60
(1) : utilisation de toutes les heuristiques	66.6
(2) : (1) - heuristique basée sur le critère de dépendance (non détaillée ici)	67.0
(3) : (2) - heuristique basée sur le calcul des syndrômes	92.1 (sur les 75% de simulations ayant pu être effectuées complètement)
(4) : (3) - heuristiques de sélection des sous-buts à prouver	Aucune simulation effectuée complètement (limitation des ressources du système au-delà de 377 GAMs)

Figure IV.5. Performances du simulateur de GULLICHSEN

Des résultats similaires ont été obtenus avec des circuits moins complexes. On peut remarquer que les différentes heuristiques n'ont pas la même efficacité. [Gul85] n'indique pas sur quelle type de machine ces expérimentations ont été conduites, mais on peut noter comme indication suffisante l'utilisation de la version 1.4 de l'interpréteur C-PROLOG développé à l'Université d'Edimbourg.

IV.3.3 Travaux de GHOSH

Les travaux de GHOSH [Gho86], à l'université de Stanford, mettent en oeuvre une approche à base de règles dans un système expert permettant d'unifier des méthodes de simulation fonctionnelle, de simulation de fautes, et de vérification de la synchronisation dans le circuit à tester.
Le langage ADA est utilisé pour la programmation du système lui-même, et fait également office de langage de description de circuits.

L'architecture générale du système à base de règles est présentée en figure IV.6 [Gho86].

Conceptuellement, ce système se divise en trois éléments :

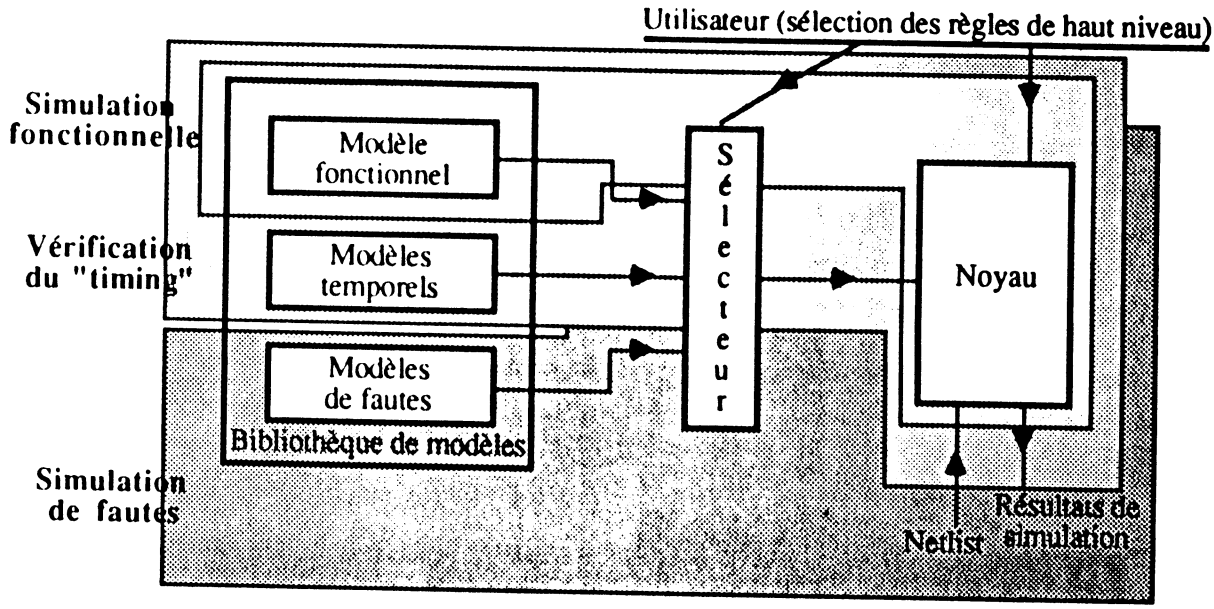


Figure IV.6. Architecture générale du système de GHOSH

Le "noyau" est une partie invariante, constituée de primitives d'ordonnancement des tâches, de routines permettant l'accès aux bases de données et de connaissances, et d'une banque de procédures ADA. Ce noyau assume donc la gestion globale du système.
Les flux de données à travers le circuit à simuler sont modélisés par des transferts de messages entre les différentes instances de tâches.[9]
Puisque le système est implémenté en langage ADA, la circulation des messages se fait par le biais d'appels de procédures et de passage de paramètres. Ces messages sont en outre fortement typés, permettant ainsi d'éviter tout message illégal.

L'ordonnancement des tâches, tout comme la gestion du flux de données, est entièrement distribué, puisque chaque tâche comporte dans sa propre description son état courant : actif ou suspendu (activable). On peut noter par ailleurs qu'une telle répartition de l'ordonnancement permet l'activation concurrente de plusieurs tâches, ce qui peut se traduire par une augmentation des performances du système de simulation dans un

9. Une tâche est liée à un modèle de composant élémentaire et à une fonction à exécuter sur ce modèle. A titre d'exemple, la simulation fonctionnelle d'une porte ET à deux entrées est définie comme une tâche, décrite par une procédure ADA, dont les paramètres prendront les valeurs des entrées et sortie de la porte ET, et dont le corps décrit la fonction d'évaluation de cette porte, conditionnée par les délais d'établissement des valeurs de sortie de la porte.

environnement multi-processeurs.
L'ordonnancement est réalisé comme suit : la première tâche activée lors du processus de simulation est une tâche spécifique, puisqu'elle est associée aux entrées primaires du circuit. Chacune des autres tâches reste à l'état "suspendu", jusqu'à ce que tous ses ports d'entrée reçoivent des messages. Lorsqu'une tâche - ou une instance de tâche - termine son exécution, elle transmet ses résultats, via ses ports de sortie, à toutes les tâches qui lui sont connectées. Ces messages sont émis dans un ordre indifférent.

La "bibliothèque" de modèles respectivement invoqués pour la simulation fonctionnelle, la simulation de fautes et la vérification des caractéristiques temporelles, est composée de règles de deux niveaux de priorité. Elles forment une base de règles organisée sous forme de paires "conditions/actions", associées à chaque modèle de composant élémentaire.
Pour chacun de ces modèles élémentaires, on distingue les règles de haut niveau associées au choix de la fonction à effectuer (simulation fonctionnelle, simulation de fautes ou vérification de la synchronisation), et les règles de bas niveau relatives à la mise en oeuvre d'une fonction particulière (par exemple, lien de causalité entre évènements et transitions d'états pour la simulation).
Ces règles utilisent la modélisation de chaque composant élémentaire ainsi que la modélisation des différentes altérations possibles du comportement de ces composants. La prise en compte d'un module non élémentaire ou du circuit dans son ensemble s'effectue par construction d'un modèle utilisant différentes instances des modèles élémentaires munies des paramètres adéquats.

La troisième partie du système (non représentée en figure IV.6) est une partie variable constituée de la compilation d'informations issues d'outils de CAO plus "classiques" (netlist, etc.). Cette partie représente la base de faits du système.

Les avantages d'une telle approche résident principalement dans son aspect totalement réparti, grâce à la notion d'intercommunication entre tâches par messages, un message envoyé représentant la propagation d'un signal d'un élément du circuit vers les éléments qui lui sont interconnectés. Notons par ailleurs que cette communication par messages permet de représenter plus fidèlement la propagation simultanée des signaux au niveau des "pieds de fan-out" vers toutes les branches de fan-out.
Cette approche a été mise en oeuvre dans le système RDV [Gho86] et validée sur des circuits digitaux. Elle reste néanmoins limitée aux circuits combinatoires dont la complexité se limite à l'échelle LSI.

IV.4 CONCLUSION

Différents systèmes à base de connaissances ont été présentés, que ce soit pour la simulation et la simulation de fautes, ou la génération automatique de vecteurs de test.

Concernant le processus de simulation, peu de méthodes faisant appel à des techniques d'intelligence artificielle ont pu être recensées, mais les travaux dans ce sens sont par contre bien plus avancés en génération de vecteurs de test.

Les principales techniques d'intelligence artificielle utilisées dans ce domaine sont l'utilisation d'heuristiques - notamment des fonctions d'évaluation de coût - pour guider la recherche dans l'espace des solutions, et des méthodes de réduction de problèmes - utilisation de graphes ET-OU -, alors que dans le domaine de la conception testable, il s'agit plutôt d'employer des techniques de planification (cf. chapitre III).
La notion de partitionnement du circuit et l'exploitation de la hiérarchie de conception sont toujours présentes, et constituent le seul moyen d'éviter l'écueil de la complexité des circuits VLSI.

V. SBC POUR LE DIAGNOSTIC

V.1 INTRODUCTION

Le diagnostic s'entreprend après la détection de fautes, et consiste à localiser les fautes, puis à les caractériser. Ces deux tâches - localisation et caractérisation - peuvent être effectuées d'un point de vue comportemental, fonctionnel, structurel, topologique ou physique, selon le niveau d'abstraction auquel on considère le système ou circuit sous test, et auquel la détection de fautes est réalisée.
D'autre part, le processus de diagnostic peut intervenir à toute phase de la vie d'un système ou circuit : en fin de conception, en fin de fabrication, ou durant son utilisation.

En fin de conception, le diagnostic permet la localisation et la caractérisation de fautes résiduelles de conception (résiduelles le plus souvent car on suppose que des outils d'aide à la conception ont été utilisés), notamment par utilisation d'extracteurs de schéma logique à partir du dessin des masques et par comparaison de ce schéma avec le schéma obtenu lors de la synthèse.
D'autre part, les prototypes réalisés doivent être validés et soumis à des simulations de conditions d'utilisation relatives à l'environnement défini dans le cahier des charges, ces simulations permettant d'évaluer d'une part les effets du vieillissement, et d'autre part les effets d'un environnement d'utilisation parfois particulièrement agressif (applications militaires ou spatiales, par exemple). Un diagnostic peut donc être également nécessaire lorsque ces tests ne sont pas subis avec succès par le prototype.

En fin de fabrication, les tests sont plus traditionnellement du type "go/nogo", et permettent d'effectuer un tri plutôt qu'un diagnostic, en vue d'éliminer les circuits non corrects relativement à ces tests.
Néanmoins, si le rendement de la chaîne de fabrication est très inférieur à la normale communément admise pour une technologie et une surface de circuit données, il peut y avoir remise en question du procédé de fabrication utilisé, et donc nécessité d'entamer un processus de diagnostic de ce procédé à cet effet.

Au cours de l'utilisation d'un circuit ou d'un système, dans un environnement réel et pour des applications réelles ou simulées, le diagnostic intervient à divers degrés, après constat d'une défaillance, selon qu'il s'agit d'une intervention de maintenance chez l'utilisateur (cas d'un système électronique, en général) ou d'une analyse de défaillances chez le fabricant (acheteur de circuits) de ce système (cas d'une carte électronique ou d'un circuit avant montage).
Lors d'une intervention de maintenance, le diagnostic doit s'effectuer le plus rapidement

possible, de façon à ne pas immobiliser le système trop longtemps. Il se restreint donc en général à une localisation d'unité la plus petite possible, *remplaçable sur site*, ayant causé - ou pu causer - la défaillance.
Lors de l'analyse de défaillances de cette unité remplacée, le diagnostic est utile pour localiser plus finement et surtout caractériser la faute - en général physique - ayant conduit à la défaillance : il peut s'agir d'effets non encore soupçonnés de conditions particulières ou marginales d'utilisation, ou encore de vieillissement. Les résultats du diagnostic en analyse de défaillances sont donc utiles au fabricant, en ce sens qu'ils sont porteurs d'informations permettant l'amélioration ultérieure des techniques de conception, de validation et de fabrication utilisées.

Dans toutes les phases du cycle de vie, outre le problème de la quantité d'informations possédées sur le système à tester, se pose le problème de l'observabilité de ce système. Souvent, en effet, les fautes ne sont détectées qu'au niveau des sorties du circuit ou système, et il s'agit donc d'une observabilité externe. Néanmoins, des possibilités réelles d'observation interne sont apparues avec le test sans contact, notamment par faisceau d'électrons.
Des applications pour l'analyse de défaillances sont en cours de développement [CCC89]. Quant à l'utilisation du test par faisceau d'électrons pour la mise au point de prototypes, elle fait l'objet de la deuxième partie de ce document.

Dans la suite du chapitre, des systèmes de diagnostic automatique à l'aide de techniques d'intelligence artificielle sont présentés. Ils ont été classés en deux catégories, selon qu'il s'agit de systèmes experts de première génération, ou de systèmes à base de connaissances profondes. Cette classification correspond à celle déjà présentée au chapitre II.

V.2 SYSTEMES A BASE DE CONNAISSANCES DE SURFACE

V.2.1 Généralités

Traditionnellement, le diagnostic de fautes au niveau logique utilise un dictionnaire de fautes, généré à l'issue du processus de simulation de fautes, à l'aide de vecteurs de test de diagnostic.[10]
Un dictionnaire de fautes est un ensemble de relations entre fautes possibles et combinaisons de sorties primaires qui les manifestent, pour un vecteur de test donné.[11]

---

10. Il s'agit de vecteurs différents de ceux générés pour la détection de fautes. Leur capacité à permettre la localisation de ces fautes est appelée résolution de diagnostic [Fuj85].

11. Nous nous contenterons ici de cette définition simple, mais ceci est détaillé dans la deuxième partie.

Bien qu'un dictionnaire de fautes puisse être utile au diagnostic lorsque l'on considère un circuit au niveau logique et que l'on ne dispose d'aucun moyen d'observabilité interne, il ne peut aider à la localisation à un autre niveau d'abstraction, ni à la localisation dans le cadre d'un système et non pas d'un circuit.
De plus, il ne permet pas de diagnostics partiels pouvant être remis en question, par son organisation même sous forme de table de correspondance, et son architecture fixe (cf. à ce sujet la deuxième partie, chapitre V, § V.4.1).

Les limites des dictionnaires de fautes, ajoutés aux résultats prometteurs des travaux menés sur les systèmes experts de diagnostic dans d'autres domaines (domaine médical, notamment), ont suscité l'intérêt de la communauté du test pour l'application de techniques d'intelligence artificielle au problème du diagnostic de fautes.
En effet, le diagnostic, lorsqu'il est vu comme un processus de classification ou d'étiquetage [BMC85], correspond tout à fait à la qualification et à l'identification de fautes expliquant les dysfonctionnements observés, parmi toutes les fautes possibles ayant été préalablement définies.
En ce sens, l'utilisation de systèmes experts de première génération pour le diagnostic de fautes procède du même principe que l'utilisation de dictionnaires de fautes, mais se révèle beaucoup plus puissante par les capacités qu'elle offre de gestion de bases de données importantes, ainsi que de raisonnement hypothétique (cf. chapitre II).

Deux systèmes experts sont présentés dans la suite (§ V.2.2 et § V.2.3). De nombreux autres exemples existent, comme le système CRIB [Har84], développé conjointement par la société ICL et l'Université de Brunel (Royaume Uni), et destiné originellement au diagnostic de fautes dans les systèmes logiques, mais qui a pu être réutilisé en tant que "coquille" ("shell"), une fois supprimée sa connaissance spécifique, pour une tâche de classification des espèces animales. Un autre exemple, proche du système MIND (cf. § V.2.3), est le système d'aide au diagnostic de défaillances pour les testeurs de VLSI de la gamme MegaOne, commercialisés par la société MegaTest [Mul84]. Enfin, pour avoir une idée générale de ce qui se fait actuellement dans le domaine en France, on pourra consulter les actes de la conférence spécialisée "Systèmes Experts et Maintenance", qui s'est tenue dans le cadre des "Huitièmes Journées Internationales d'Avignon" [Avi88].

V.2.2 Le système DART

Le système expert DART[12] [BeH81] est un système d'aide au diagnostic de fautes, développé par le Centre Scientifique IBM de Palo Alto et l'Université de Stanford. DART

12. A ne pas confondre avec le système de même nom, mettant en oeuvre les travaux de GENESERETH [Gen84], dont il est question au chapitre II, § II.4.3

est un système construit en utilisant EMYCIN (Essential MYCIN) [Van80], le moteur nu du système expert MYCIN [Sho76].

La vocation de DART est le diagnostic de fautes, logicielles ou matérielles, pouvant survenir dans un système informatique. L'expertise de DART lui permet, partant d'un point de vue systémique, d'identifier les composants défaillants.

Lors de la conception du premier prototype, les auteurs de DART ont choisi de s'intéresser aux problèmes de télécommunications pour les ordinateurs de la gamme IBM 370.

Nous présentons ci-après le processus d'inférence de DART [BeH81] :

- Rassembler les symptômes présentés et les informations sur la configuration de chaque sous-système.
- Inférer les sous-systèmes (CPU-E/S) suspects pour chaque problème.
- Identifier des chemins logiques pour chaque sous-système.
- Sélectionner les protocoles appropriés et les outils de diagnostic disponibles pour chaque chemin logique.
- Déterminer les protocoles violés et les composants susceptibles de défaillance pour chaque chemin physique.
- Indiquer les résultats (composants incriminés) et émettre des recommandations.

Notons que le processus d'inférence de diagnostic du système DART est exactement le même que celui du système MYCIN, notamment au niveau de l'évolution du raisonnement et au niveau de la "thérapeutique" conseillée.

L'analogie existe aussi dans la forme des règles d'inférence utilisées par DART, comme le montre l'exemple suivant, cité par [Dud83] :

IF

1) the class of the device that is failing is 3380, and
2) The problem is missing interrupt, and
3) A dynamic trace is available, and
4) The trace indicates major hardware, microcode or formatting errors, and
5) The interface is failing, and
6) The failing interface has a pack change interrupt outstanding

THEN

There is a strong evidence that the cause of the failure is the host (0.9). The missing interrupt is caused by the device being reserved without being released.

On notera tout particulièrement que la conclusion de la règle est assortie d'un degré de certitude compris entre 0 et 1.

La base de connaissances de DART est composée de 300 paramètres EMYCIN [Van80], et de 190 règles de production, obtenus à l'aide d'interviews de 5 spécialistes de différents aspects du processus de diagnostic.
A ce propos, la remarque suivante est formulée dans [BeH81] : bien que les experts aient pu exprimer leurs connaissances, il a été très difficile de modéliser celles-ci sous forme de règles, car chaque règle représente une déviation possible dans l'exécution d'un protocole, d'où notamment un problème de complétude.
La conclusion, tirée déjà en 1981,[13] était la nécessité d'orienter les travaux vers l'utilisation de représentations explicites des protocoles eux-mêmes, et de n'exprimer sous forme de règles de production que des méthodes générales de diagnostic, permettant d'inférer automatiquement des hypothèses sur les déviations ou dysfonctionnements.

V.2.3 Le système MIND

Le système MIND, développé par la société TERADYNE (Californie) [Wil84] est dévolu au diagnostic des testeurs de VLSI J941 commercialisés par cette société. L'impératif ayant conduit à la réalisation du système MIND est la réduction du temps moyen de réparation (MTTR : "Mean Time To Repair") des testeurs.
MIND répond à cet objectif d'une part en fournissant une aide au diagnostic (gain de temps et réduction de la mobilisation des experts), et d'autre part en limitant la précision de son diagnostic, puisque le but est d'identifier les défaillances au niveau des plus petites unités remplaçables sur site (LFRU : "Lowest Field Replaceable Unit") ; dans le cas des testeurs J941, les LFRUs sont des cartes électroniques.[14]

Le système MIND est construit autour du programme de test CHECK941 [WST84], utilisé auparavant par les ingénieurs de test de la société TERADYNE.
Les apports de MIND sont l'automatisation du processus de diagnostic, c'est-à-dire de la

---

13. Rappelons que les premiers travaux théoriques sur les systèmes à base de modèles profonds datent du début des années 80.

14. Notons que la notion de diagnostic au niveau LFRU se généralise à la totalité des applications dont le but est la diminution du MTTR (c'est en fait le cas de tout test au cours de la vie d'un système). Outre la réduction de l'espace de recherche parcouru par le système de diagnostic, cette notion permet d'assurer contrôlabilité et observabilité des composants suspects, accélérant ainsi l'obtention du diagnostic. Si un diagnostic plus fin est désiré, il peut se faire alors après remplacement de l'unité défaillante, sans immobiliser le système, et donc sans incidence sur le MTTR.

localisation proprement dite des unités défaillantes.
L'architecture du système MIND est donnée en figure V.1 [Wil84].

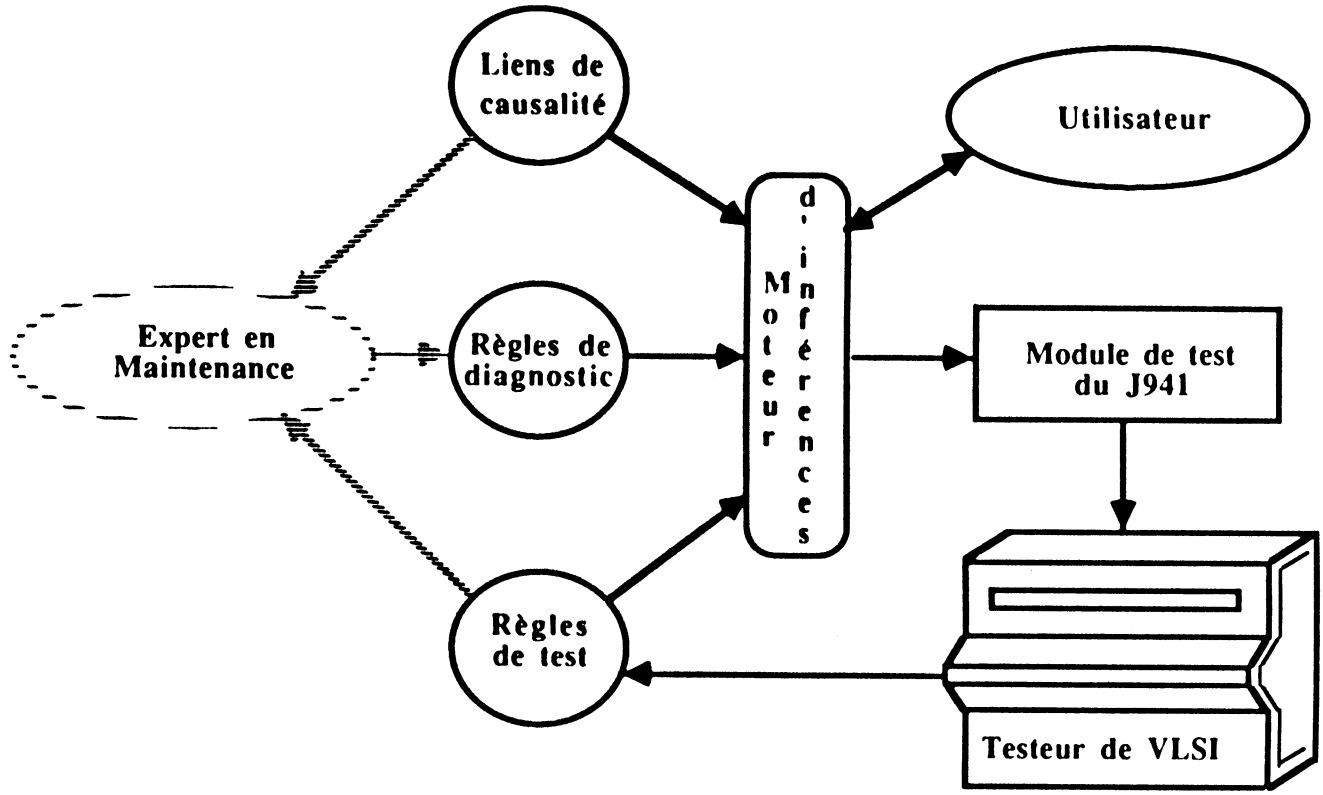


Figure V.1. Architecture du système MIND

Parmi les différents modules de la figure V.1, ceux dénommés "Module de test du J941", "Testeur de VLSI", et "Règles de test" existaient déjà dans CHECK941, la base de données contenant les symptômes, c'est-à-dire les couples stimulus/réponse appliqués et observés, et le système de test étant le testeur de la gamme J941. Le "Module de test du J941" est un module de détection de fautes spécifique au J941.

Les modules restants, ajoutés à ceux composant CHECK941, vont former le système MIND, et constituent les composantes habituelles d'un système expert de première génération (cf. chapitre II).
Le moteur d'inférences [GrW85] procède à la sélection et à l'application des règles contenues dans la base de connaissances ("Règles de diagnostic" [RyW85], en fonction des symptômes contenus dans la base de faits ("Règles de test"), et d'un ensemble de liens de causalité, permettant de mettre en relation les résultats de différents tests, ainsi que les connexions entre cartes composant le J941 (LFRUs).
Enfin, les modules notés respectivement "Utilisateur" et "Expert en maintenance" assurent l'interaction entre MIND et son utilisateur, et confèrent à MIND l'une de ses originalités,

puisque ce système ne se contente pas d'acquérir l'expérience d'un expert, il a aussi la capacité d'apprendre : en effet, le diagnostic fourni par MIND au cours d'une session peut être critiqué par un expert, ce qui permet au système de corriger et de mettre à jour ses connaissances, de façon interactive.

Le processus de diagnostic globalement suivi par MIND est le suivant [Wil84] :

- Collecte des informations
- Recherche, parmi les modules que contient MIND, de celui qui est susceptible de traiter le problème posé
- Mise en jeu d'une séquence de test pour détecter la défaillance
- Analyse des résultats de cette séquence de test
- Déclenchement d'une heuristique appropriée au diagnostic cherché
- Envoi du résultat à l'utilisateur
- Emission de recommandations pour la réparation

V.3 SYSTEMES A BASE DE CONNAISSANCES PROFONDES

V.3.1 Généralités

Les systèmes à base de connaissances profondes ont longuement été présentés au chapitre II, nous ne reviendrons pas ici sur leurs caractéristiques, sur ce qu'ils apportent de nouveau par rapport aux systèmes experts de première génération, sur les théories fondamentales qu'ils mettent en oeuvre, ni sur leurs limites.

Outre les travaux de DAVIS, de De KLEER, de GENESERETH et de REITER, déjà amplement décrits, nous présentons dans la suite deux autres systèmes à base de modèles profonds (cf. § V.3.2. et § V.3.3).

On pourra également avoir une vue d'ensemble des travaux effectués dans ce domaine en France en consultant les actes de la conférence spécialisée "Systèmes Experts de Seconde Génération", qui s'est tenue dans le cadre des "Neuvièmes Journées Internationales d'Avignon" [Avi89].

V.3.2 Le système FAULTFINDER

Le premier système examiné ici est FAULTFINDER [Esh82], un système de diagnostic de fautes dans les circuits logiques.

Le système FAULTFINDER se base uniquement sur le comportement du circuit vis-àvis de ses entrées/sorties, ce qui lui permet de ne pas avoir besoin d'accéder directement aux éléments constituants du circuit [Esh82].

Kave ESHGHI note que FAULTFINDER ne peut encore traiter que les circuits combinatoires (comme d'ailleurs la plupart des systèmes experts de ce genre), mais que le principe sur lequel ce système se base peut être appliqué aux circuits séquentiels, quoique, à notre avis, cette adaptation ne pourrait se faire que sous réserve de modifications importantes du système, notamment au niveau de la "théorie" du circuit sous test (cf. ci-après).

Le but poursuivi par le système FAULTFINDER au cours de son mécanisme inférentiel est d'extraire la "théorie" du circuit défaillant, en prenant comme présupposé que la "théorie" du circuit fonctionnant correctement est connue.

Pour extraire la "théorie" du circuit défaillant, FAULTFINDER opère une série d'expériences ou de tests sur ce circuit, et en compare le résultat avec la "théorie" du circuit correct.

Il convient à ce niveau de décrire un peu plus longuement ce que Kave ESHGHI entend par "théorie" [Esh82] : Une théorie T d'un circuit C est subdivisée en trois sous-théories :

- Description du circuit D
- Tables de vérité des portes logiques TB
- Lois générales sur les circuits logiques G

Chacune de ces trois sous-théories donne lieu à une série d'axiomes et de clauses du langage PROLOG [CIM85].
Ainsi, la description du circuit se traduit par un ensemble d'axiomes dont le prédicat est constitué par le type de porte, prenant comme arguments le nom affecté à l'instance de porte, et les entrées/sorties de cette instance.
Les tables de vérité de chaque porte logique sont représentées par des paquets d'axiomes (axiomes de même prédicat), de foncteur (au sens de PROLOG) le type de table (table ET, table OU, etc.), et d'arguments une combinaison valide des valeurs logiques des entrées/sorties.
Enfin, les "lois générales sur les circuits logiques", sont toutes exprimées à l'aide d'un prédicat principal à deux arguments - une liste d'entrées et une liste de sorties primaires du circuit, avec leurs états respectifs -, et permettent d'inférer les sorties correctes, à partir de la connaissances des entrées. Ces inférences se font "pas à pas", et utilisent la description du circuit et les tables de vérité. S'il s'avère, en fin de processus, que les sorties inférées ne correspondent pas aux sorties effectivement observées, le système essaie de dériver la théorie du circuit défaillant, par un procédé du type "hypothèse et test" [Esh82]. La trace de l'inférence est alors analysée, et la faute peut ainsi être localisée. Le problème est donc de générer les hypothèses permettant de mener au diagnostic, et ceci le plus rapidement possible, ce qui se traduit par un problème de détermination d'entrées "discrimantes", de façon à restreindre l'ensemble des hypothèses générées.

V.3.3 Travaux de THEARLING & IYER

Le système réalisé par Kurt THEARLING et Ravi IYER en 1987 à l'Université d'Urbana Champaign (Illinois) [Thl88] a pour objectif le diagnostic de fautes dans les systèmes digitaux, à l'aide d'une technique de raisonnement basé sur l'observation d'un comportement erroné du système étudié.

Son principe est le suivant : à partir de l'observation d'une erreur, de la connaissance du vecteur d'entrée exhibant cette erreur, et de la connaissance de la structure du système, il effectue son diagnostic, c'est-à-dire la détermination de la composante défaillante dans le système sous test.

Le système de THEARLING présente les caractéristiques suivantes :

- Il se restreint aux systèmes combinatoires
- Il ne considère que le cas de faute simple
- La structure du système à diagnostiquer est vue comme un ensemble de modules
- Le modèle de faute pris en compte est la constatation d'un nombre quelconque de valeurs erronées dans la table de vérité du comportement de ce module, *à condition que ceci ne résulte pas en une faute séquentielle* [Wad78].

Dans la modélisation adoptée, un module peut être soit un module simple (porte logique élémentaire), soit un module complexe (additionneur, multiplexeur, etc.). Une erreur peut prendre soit la valeur D (valeur attendue = 1, valeur obtenue = 0), soit la valeur $\overline{D}$ (valeur attendue = 0, valeur obtenue = 1).
La relation entre la structure du système et la propagation d'erreurs à travers ce système est modélisée à l'aide d'arbres ET-OU [Ric87], dont les deux noeuds génériques sont donnés en figure V.2.

Les noeuds génériques de la figure V.2 s'interprètent comme suit :

- *Noeud OU :*
  L'erreur "erreur1", observée en sortie du module C1 est due soit au fonctionnement défectueux du module C1, soit au fait que la valeur d'entrée "valeur1" est erronée OU au fait que la valeur d'entrée "valeur2" est erronée.
- *Noeud ET :*
  L'erreur "erreur2", observée en sortie du module C2 est due soit au fonctionnement défectueux du module C2, soit au fait que la valeur d'entrée "valeur3" est erronée ET au fait que la valeur d'entrée "valeur4" est erronée.

La figure V.3 présente les différents noeuds ET et noeuds OU que l'on peut obtenir pour des portes logiques élémentaires à deux entrées.

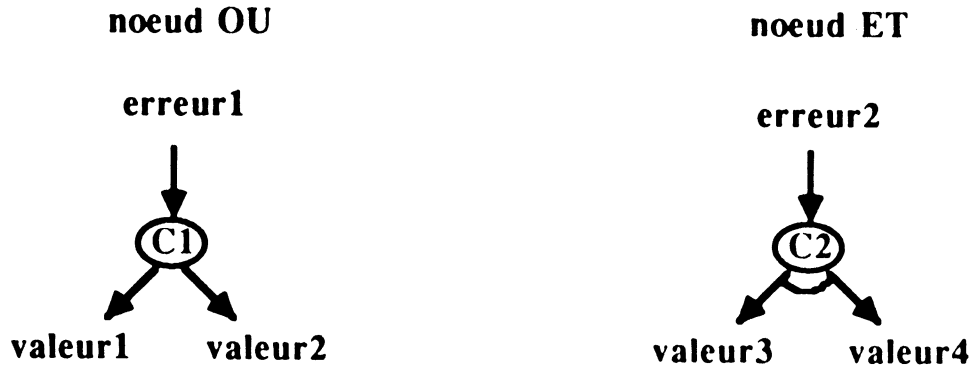


Figure V.2. Noeuds génériques de l'arbre ET-OU

Le principe de raisonnement pour l'obtention du diagnostic se ramène à un processus de démonstration de théorèmes. L'arbre ET-OU est transformé en une série de théorèmes (à raison d'un théorème par branche ET de l'arbre), décrits sous forme de clauses du langage PROLOG [CIM85] pour des facilités de manipulation.

La figure V.4 présente un exemple de circuit (a), l'arbre ET-OU obtenu (b), ainsi que les trois théorèmes qui en sont dérivés (c), en supposant observée l'erreur "Z = D" en sortie du circuit.

Un parcours de l'arbre correspond à une résolution progressive des théorèmes, sachant les valeurs des entrées primaires du système et l'erreur observée en sortie.
La résolution consiste, pour chaque vecteur de test, à éliminer au fur et à mesure les alternatives non réalisées sur les valeurs des entrées, jusqu'à obtention de théorèmes *activés*, c'est-à-dire ne comportant plus en prémisses que des hypothèses sur les modules défaillants. En fin de processus, il y a intersection de tous les ensembles contenant les modules apparaissant dans les théorèmes activés, obtenus pour chaque vecteur de test, de façon à réduire l'ensemble des suspects.

A partir des théorèmes de la figure V.4, et en supposant l'erreur observée "Z = D" en sortie du circuit, on peut obtenir les résolutions suivantes :

- *Cas des entrées primaires A = 1, B = 1, C = 1, D = 1 :*
  Ce vecteur de test permet d'éliminer toutes les alternatives sur les valeurs des entrées dans le théorème ThD3, puisque celles-ci ne sont pas réalisées. Pour ce qui concerne les théorèmes ThD1 et ThD2, seules certaines alternatives peuvent être éliminées.
  Il s'ensuit que le seul théorème activé est le théorème ThD3, et on obtient un premier ensemble de modules suspects, S1 = {O1,A1,A3}.

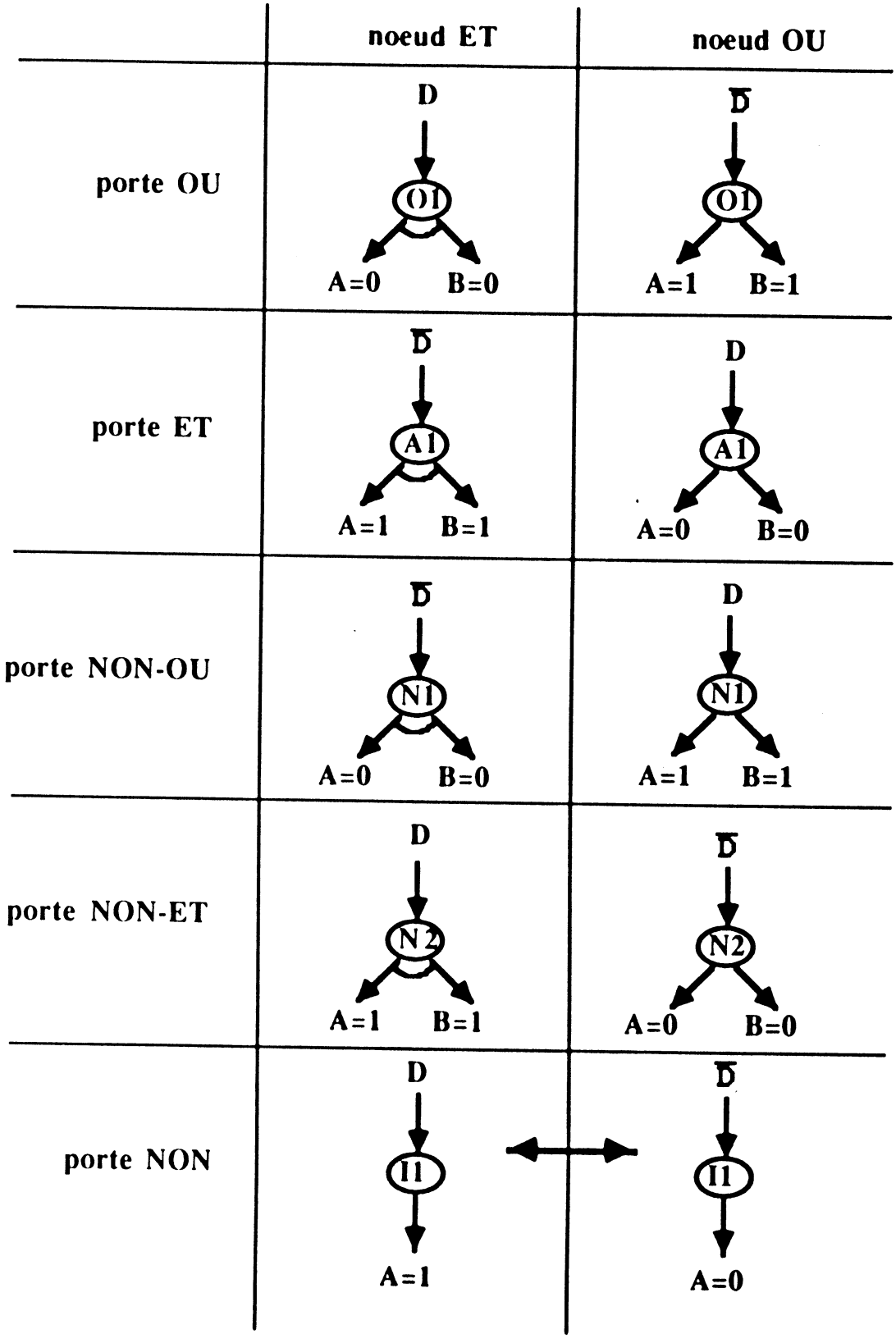
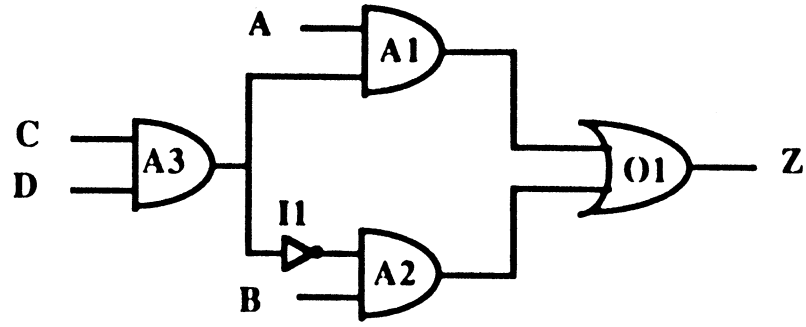
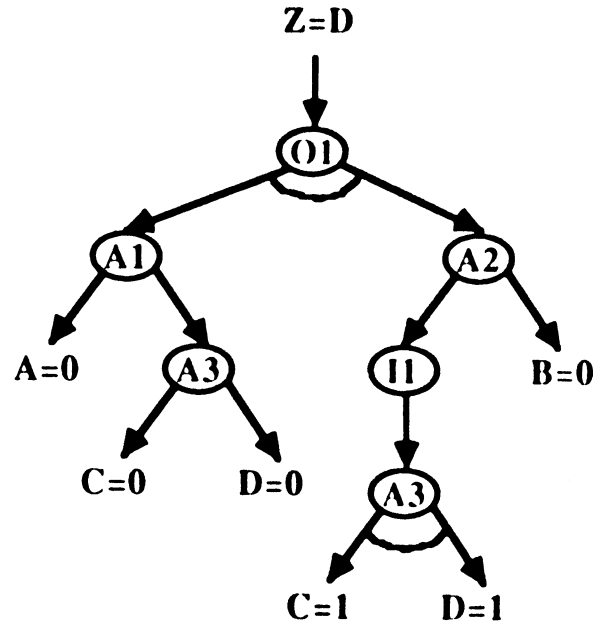


Figure V.3. Noeuds ET et noeuds OU pour des portes logiques élémentaires



(a) Circuit exemple



(b) Arbre ET-OU déduit

ThD1 : (O1' ou A2' ou I1' ou A3' ou B=0 ou D=1)

ET

ThD2 : (O1' ou A2' ou I1' ou A3' ou B=0 ou C=1)

ET

ThD3 : (O1' ou A1' ou A3' ou A=0 ou C=0 ou D=0)

(c) Théorèmes dérivés

Figure V.4. Arbre ET-OU et théorèmes dérivés d'un circuit exemple

- *Cas des entrées primaires A = 0, B = 1, C = 0, D = 0 :*
  Ce vecteur de test permet d'éliminer toutes les alternatives sur les valeurs des entrées dans les théorèmes ThD1 et ThD2, puisque celles-ci ne sont pas réalisées. Pour ce qui concerne le théorème ThD3, seules certaines alternatives peuvent être éliminées.
  Il s'ensuit que les théorèmes activés sont ThD1 et ThD2, et on obtient un deuxième ensemble de modules suspects, S2 = {O1,A2,I1,A3}.

Ces deux vecteurs de test appliqués permettent de réduire l'ensemble des suspects à S3 = S1∩S2 = {O1,A3}. On ne peut réduire plus cet ensemble sans appliquer d'autres vecteurs de test permettant d'observer la même erreur en sortie.

Les techniques employées dans ce système de diagnostic sont proches des travaux de DAVIS et al. présentés au chapitre II. Elles sont intéressantes, mais posent les problèmes suivants, dans l'état actuel du système : il est nécessaire d'obtenir un seul module suspect en fin de processus, à cause de l'hypothèse de faute simple, ce qui nécessite l'existence de vecteurs de test adéquats (problème de génération). D'autre part, l'application d'une telle méthode de diagnostic à des circuits ou systèmes de taille importante nécessite au préalable d'effectuer un partitionnement du système sous test, et de pouvoir utiliser la même méthode de diagnostic dans le cadre de résolutions partielles.

V.4 CONCLUSION

Ce chapitre sur les systèmes à base de connaissances pour le diagnostic de fautes dans les systèmes électroniques permet d'achever le tour d'horizon de l'emploi de techniques d'intelligence artificielle pour le test et la testabilité des circuits et systèmes électroniques.

Nous avons pu voir que le diagnostic est effectué à différents stades du cycle de vie d'un système, et qu'il intervient après la détection de fautes.
Cette détection de fautes va constituer l'unique modèle de fautes pris en compte par les systèmes à base de connaissances profondes : une divergence entre un comportement correct attendu et un comportement erroné observé.
Les systèmes à base de connaissances de surface, par contre, nécessitent l'identification préalable d'un certain nombre de dysfonctionnements possibles, et ramènent le diagnostic à un problème de classification.

Les remarques du chapitre II relatives aux spécificités de chacune des deux approches, ainsi que les conclusions qui en sont tirées, notamment sur l'intérêt d'une approche mixte (systèmes de seconde génération), sont tout particulièrement vraies pour le problème du diagnostic.

On remarquera que les systèmes experts de première génération concernent surtout le diagnostic de systèmes électroniques, effectué lors d'interventions de maintenance, alors que les systèmes à base de connaissances de seconde génération sont plutôt utilisés pour le diagnostic de circuits intégrés dans le cadre d'une analyse de défaillances, sans toutefois aboutir à une caractérisation physique du défaut, puisqu'ils se contentent en général de localiser le plus finement possible le siège de la faute physique ayant occasionné la défaillance.

Peu de - voire aucun - systèmes de diagnostic automatique lors des phases de fin de conception et de fin de fabrication existent, qu'il s'agisse de circuits intégrés ou de cartes électroniques.
Nous avons pu voir dans l'introduction de ce chapitre que le diagnostic en fin de fabrication concerne plus le procédé de fabrication que les circuits fabriqués, nous ne nous y intéressons donc pas dans le cadre de cette étude.
Par contre, la phase de validation de la conception et des prototypes de circuits souffre réellement du manque de systèmes de diagnostic automatique, manque qui ira d'ailleurs croissant avec l'intensité de l'utilisation de compilateurs de silicium.
Notre analyse de ce problème nous conduit à en reporter les causes sur l'absence d'utilisation d'une référence adéquate de bon fonctionnement. En effet, au stade actuel de l'état de l'art, la seule référence utilisée[15] est constituée par les résultats de simulation "fault-free" du circuit à valider, par conséquent certaines fautes de conception ne pourront jamais être diagnostiquées - ni même détectées -, puisqu'elles sont également contenues dans la description du circuit - supposée être correcte - en vue de sa simulation.
La solution à ce problème réside dans l'étude de méthodologies permettant non plus seulement de concevoir et fabriquer des circuits conformes à la description que l'on a pu en faire à l'aide des outils de CAO existants, mais également conformes à ce qu'on "voudrait qu'ils soient", c'est-à-dire concevoir et fabriquer des circuits capables de répondre à des spécifications de très haut niveau d'abstraction : un tel niveau d'abstraction peut correspondre, à notre avis, à une description comportementale constituant le cahier des charges d'un circuit.
Ces recherches pourront avantageusement tirer profit des travaux menés dans le cadre de la preuve de circuits [Mil86] et de programmes [Har80], ramenant ainsi le problème du diagnostic en phase de validation de prototypes à un problème de diagnostic de fautes de conception, intervenant donc beaucoup plus tôt dans le cycle de vie d'un circuit intégré.

---

15. Y compris par nous-même (cf. deuxième partie, chapitre IV).

VI. CONCLUSION DE LA PREMIERE PARTIE

Cette première partie nous a permis de passer en revue les différentes tâches intervenant dans la conception, la mise au point et la maintenance des circuits et systèmes électroniques. Plusieurs domaines d'étude ont donc été abordés, et il a été question de conception sûre (ou correcte), de conception en vue de la testabilité, de simulation et de simulation de fautes, de génération de vecteurs de test, et de diagnostic.

Considérées dans le cadre théorique de la *sûreté de fonctionnement* tel que défini par [LCG89], ces tâches ont pour objectif *l'évitement* des fautes ("comment empêcher, par construction, l'occurrence ou l'introduction de fautes"), ou à défaut *l'élimination* des fautes ("comment minimiser, par vérification, la présence de fautes").
L'évitement des fautes - autrement dit, l'objectif du "zéro-défaut" évoqué au chapitre III - est loin d'être complètement réalisé : il faudra attendre pour cela les nombreux progrès qu'il reste à faire en compilation de silicium.[16] C'est donc d'élimination des fautes - autrement dit, de vérification - qu'il faudra se préoccuper pendant longtemps encore, et par élimination, nous entendons surtout détection et localisation, préalables nécessaires à la correction.

Face à la complexité de plus en plus grande des circuits à haut degré d'intégration et des systèmes électroniques, les méthodes algorithmiques classiques ne sont plus en mesure d'assurer détection et localisation automatiques de fautes dans de bonnes conditions. Il est devenu nécessaire de fournir aux concepteurs et aux spécialistes du test des outils réellement capables de les aider à mener à bien les tâches qu'ils doivent accomplir.

Ces outils doivent d'abord permettre d'éviter la mise à plat du système étudié, par des méthodes de partitionnement à deux niveaux, hiérarchique d'une part, et modulaire d'autre part.
Le partitionnement hiérarchique permet de considérer le système à l'étude à différents niveaux d'abstraction, qu'il s'agisse du point de vue fonctionnel, du point de vue logique, du point de vue électrique ou du point de vue physique : c'est une décomposition

16. Ces progrès concernent non seulement la capacité des compilateurs de silicium à produire des circuits de façon entièrement automatique, mais également la confiance que l'on peut leur accorder : pour que les circuits produits soient corrects, encore faut-il que le compilateur de silicium qui les génère fonctionne lui-même correctement.

"verticale".
Le partitionnement modulaire permet de considérer le système, pour un même niveau d'abstraction structurel, à différents degrés de complexité, allant de la vue systémique jusqu'à la notion de "plus petites unités remplaçables" : c'est une décomposition "horizontale".
Ce partitionnement hiérarchique et modulaire doit néanmoins garantir la correspondance et la cohérence entre les différents niveaux d'abstraction et les différents degrés de décomposition.

Ces outils doivent ensuite permettre une modélisation adéquate du système étudié, de façon à favoriser la détection de ses dysfonctionnements, ainsi que la localisation et l'identification des fautes dont il est entâché.

Ces outils doivent enfin permettre de pallier les difficultés d'observabilité et de contrôlabilité internes, en autorisant un raisonnement hypothétique, à partir de connaissances parfois incertaines.

Nous espérons être parvenue à montrer dans cette partie, y compris par l'exemple, que des outils offrant de telles possibilités sont les systèmes à base de connaissances, mettant en oeuvre des techniques d'intelligence artificielle.

Il reste néanmoins très difficile d'évaluer ces systèmes et leurs apports, d'une part car ils sont souvent encore au stade de recherche - peu d'entre eux sont en tous cas utilisés intensivement dans l'industrie, bien que [Rau87] indique que la plupart des grandes sociétés d'équipement de test automatique sont concernées par ce marché : Daisy Systems, Fairchild, GenRad, MegaTest et Teradyne, comme il a été montré dans cette partie du document. -, d'autre part et surtout car peu de bancs d'essai ("benchmarks") existent.
En effet, seuls les systèmes à base de connaissances pour le diagnostic ont été étudiés de ce point de vue [Wil86], et l'unique benchmark dont nous ayons eu connaissance a été proposé très récemment [BBB89].

Concernant la dualité systèmes à base de connaissances de surface/systèmes à base de connaissances profondes, nous pouvons ajouter à leur comparaison que les premiers sont plus interactifs, mais ne peuvent fournir qu'une aide au diagnostic, alors que les seconds, par le fait même qu'ils raisonnent à partir des "principes premiers" du système étudié, sont capables d'effectuer le diagnostic complet sans intervention de l'utilisateur. En outre, les systèmes à base de connaissances de surface restent spécifiques à un type de circuit ou de système, alors que les systèmes à base de modèles profonds, utilisant des connaissances extrêmement générales sur l'électronique et la logique, peuvent être utilisables pour la plupart des applications.

Un rapide bilan des méthodes et systèmes présentés dans cette première partie nous permet de conclure que, si les approches à base de connaissances peuvent être efficacement utilisées pour les tâches de haut niveau que sont le partitionnement d'un circuit ou système

complexe - spécialement par sa taille et ses possibilités d'observabilité et de contrôlabilité -, et le choix de la méthode ou de l'algorithme qui se prête le mieux à une représentation d'un niveau d'abstraction donné, les algorithmes et techniques plus classiques restent utilisés en tant que tels pour des modules donnés, peu complexes, à des niveaux d'abstraction donnés.
L'emploi d'une approche à base de connaissances pour le test ou la testabilité d'un circuit ou système électronique n'a donc pas pour objectif de *remplacer* des méthodes qui ont amplement démontré leur efficacité et leurs performances, mais, étant donnée la complexité des dispositifs actuels, de *recréer* les conditions optimales d'utilisation de ces méthodes.

Bibliographie

[AbB85] M. S. Abadir et M. A. Breuer, **A knowledge-based system for designing testable VLSI chips**, IEEE Design and Test of Computers 4, 2 (Aout 1985), 56-68.

[AKM88] M. Abramovici, B. Krishnamurthy (moderator), R. Mathews, B. Rogers, M. Schulz, S. Seth et J. Waicukauski, **What is the path to fast fault simulation ? (Panel discussion)**, 19th IEEE International Test Conference, Washington, Sep. 1988, 183-192.

[Arm66] D. B. Armstrong, **On finding a nearly minimal set of fault detection tests for combinational logic nets**, IEEE Transactions on Electronic Computing ec-15, 1 (Fev. 1966), 66-73.

[Avi88] Avignon'88, **Actes de la Conférence spécialisée Systèmes Experts et Maintenance**, EC2, Avignon, Mai 1988.

[Avi89] Avignon'89, **Actes de la Conférence spécialisée Systèmes Experts de Seconde Génération**, EC2, Avignon, Mai 1989.

[BaF82] A. Barr et E. Feigenbaum, **The handbook of artificial intelligence**, Pitman, Londres, 1982.

[BBB89] M. Ben-Bassat, D. Ben-Arie, I. Beniaminy, J. Cheifetz et M. Klinger, **A proposed benchmark unit for evaluating troubleshooting expert systems**, 20th IEEE International Test Conference, Washington, Aout 1989, 78-86.

[BeH81] J. S. Bennet et C. R. Hollander, **Dart : an expert system for computer fault diagnosis**, International Joint Conference on Artificial Intelligence, Vancouver, 1981, 843-845.

[Bha88] S. Bhawmik, **An integrated CAD system for the design of testable VLSI circuits**, [Phd dissertation, Indian Institute of Technology], Kharagpur, Fev. 1988.

[BhP89a] S. Bhawmik et P. Pal Chaudhuri, **DFT EXPERT : designing testable VLSI circuits**, IEEE Design and Test of Computers 6, 5 (Oct. 1989), 8-19.

[BhP89b] S. Bhawmik et P. Pal Chaudhuri, **Selecting test methodologies for PLAs and random logic modules in VLSI circuits - an expert systems approach**, Integration, the VLSI journal 7, (1989), 267-281.

[BFR85] T. Blackman, J. Fox et C. Roseburgh, **The SILC silicon compiler: language and features**, 22nd ACM/IEEE Design Automation Conference, Las Vegas,

Juin 1985, 232-237.

[BFS88] G. Boy, B. Faller et J. Sallantin, **Acquisition et ratification de connaissances**, Actes des Journées Nationales du PRC-GRECO IA, Toulouse, Mars 1988, 321-356.

[Boy88] G. Boy, **Assistance à l'operateur : une approche de l'intelligence artificielle**, Teknea, Toulouse, 1988.

[BrF76] M. A. Breuer et A. D. Friedman, **Diagnosis and reliable design of digital systems**, Pitman, California, 1976.

[BrZ85] M. A. Breuer et X. Zhu, **A knowledge-based system for selecting a test methodology for a PLA**, 22nd ACM/IEEE Design Automation Conference, Las Vegas, Juin 1985, 259-265.

[BGG88] M. A. Breuer, R. Gupta, R. Gupta, K. J. Lee et J. C. Lien, **Knowledge-based systems for test and diagnosis**, IFIP workshop on knowledge-based systems for test and diagnosis, Grenoble, Sep. 1988, 4-28.

[Brg83] F. Brglez, **Testability in VLSI**, Canadian Conference on VLSI, Canada, 1983, 90-95.

[BMC85] T. Bylander, S. Mittal et B. Chandrasekaran, **CSRL : a language for expert system for diagnosis**, International Joint Conference on Artificial Intelligence, Los Angeles, 1985.

[CGG88] P. Camurati, P. Gianoglio, R. Gianoglio et P. Prinetto, **ESTA : an expert system for DFT rule verification**, IEEE Transactions on Computer-Aided Design 7, 11 (Nov. 1988), 1172-1180.

[ChM85] B. Chandrasekaran et R. Milne, **Special section on reasoning about structure, behavior and function - introduction**, SIGART Newsletter 1, 93 (Juil. 1985), 4-7.

[ClM85] W. F. Clocksin et C. S. Mellish, **Programmer en Prolog**, Eyrolles, Paris, 1985.

[CCC89] J. P. Collin, D. Conard, B. Courtois, P. Denis et D. Savart, **Failure analysis using e-beam**, 2nd European Conference on Electron and Optical Beam Testing of Integrated Circuits, Duisburg, Oct. 1989.

[Cou81] B. Courtois, **Test et LSI**, [Th. de Doctorat d'Etat, I.N.P.Grenoble], Grenoble, Juin 1981.

[DaT79] R. David et P. Thevenod-Fosse, **Panorama des méthodes de test non déterministes des circuits logiques**, RAIRO 13, 1 (1979), .

[Dav82] R. Davis, **Expert systems : where are we? And where do we go from?**, AAAI Magazine 3, 2 (1982), 01-22.

[DSH82] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley et S. Polit, **Diagnosis reasoning based on structure and function**, National Conference on Artificial Intelligence, Pittsburgh, Aout 1982, 137-142.

[DaS83] R. Davis et H. Shrobe, **Representing structure and behavior of digital hardware**, IEEE Computer Journal o-83, (Oct. 1983), 75-82.

[Dav84] R. Davis, **Diagnostic based on structure and behavior**, Artificial Intelligence 1, 24 (1984), 347-410.

[DaH88] R. Davis et W. Hamscher, **Model-based reasoning: troubleshooting**, in Exploring Artificial Intelligence - Survey talks from the National Conferences on A.I., H. Shrobe et AAAI (ed.), Morgan Kaufmann, Californie, 1988, 297-346.

[DeC85] A. J. De Geus et W. Cohen, **A rule-based system for optimizing combinational logic**, IEEE Design and Test of Computers 4, 2 (Aout 1985), 22-32.

[DeW86] J. De Kleer et B. C. Williams, **Reasoning about multiple faults**, National Conference on Artificial Intelligence, Philadelphia, 1986.

[DGL87] C. Delorme, N. Giambiasi, R. Lbath et P. Roux, **THESEE : un système expert d'aide à la génération de vecteurs de test**, in Systèmes experts en maintenance, M. Gabriel et J. C. Rault (ed.), Masson, Paris, 1987, 39-51.

[Dud83] R. O. Duda, **Applications of expert systems**, Automatic Test Program Generation Workshop, Californie, Mars 1983.

[EiW77] E. B. Eichelberger et T. W. Williams, **A logic design structure for LSI testing**, 14th ACM/IEEE Design Automation Conference, New Orleans, Juin 1977, 462-468.

[Esh82] K. Eshghi, **Application of meta-level programming to fault finding in logic circuits**, International Logic Programming Conference, Marseille, Sep. 1982, 240-246.

[FuS83] H. Fujiwara et T. Shimono, **On the acceleration of test generation algorithms**, IEEE Transactions on Computers c-32, 12 (Dec. 1983), 1137-1144.

[Fuj85] H. Fujiwara, **Logic testing and design for testability**, The MIT Press, Cambridge, 1985.

[FWA75] S. Funatsu, N. Wakatsuki et T. Arima, **Test generation systems in Japan**, 12th ACM/IEEE Design Automation Symposium, , Juin 1975, 114-122.

[FuH86] H. S. Fung et S. Hirschorn, **An automatic DFT system for the SILC silicon compiler**, IEEE Design and Test of Computers 1, 3 (Fev. 1986), 45-57.

[GCV80] J. Galiay, Y. Crouzet et M. Vergniault, **Physical versus logical fault models in MOS LSI circuits**, IEEE Transactions on Computers c-29, 6 (Juin 1980), 527-531.

[Gen84] M. Genesereth, **The use of design descriptions in automated diagnosis**, Artificial Intelligence 1, 24 (1984), 411-436.

[Gho86] S. Ghosh, **A rule-based approach to unifying functional and fault simulation and timing verification**, 23rd ACM/IEEE Design Automation Conference, Las Vegas, Juin 1986, 677-682.

[Goe81] P. Goel, **An implicit enumeration algorithm to generate tests for combinational logic circuits**, IEEE Transactions on Computers c-30, 3 (Mars 1981), 215-222.

[GrW85] O. Grillmeyer et A. J. Wilkinson, **The design and construction of a rule base and an inference engine for test system diagnosis**, 16th IEEE International Test Conference, Philadelphie, Nov. 1985, 857-867.

[Gul85] E. Gullichsen, **Heuristic circuit simulation using Prolog**, Integration, the VLSI journal 3, (1985), 283-318.

[Gup86] R. Gupta, **Test pattern generation for VLSI circuits in a Prolog environment**, International Conference on Logic Programming, Londres, Juil. 1986, 528-535.

[Har80] D. Harel, **Proving the correctness of regular deterministic programs : a unifying survey using dynamic logic**, Theoretical Computer Science 12, (1980), 61-81.

[Har84] R. T. Hartley, **CRIB : computer fault finding through knowledge engineering**, IEEE Computer Journal m-84, (Mars 1984), 76-83.

[Hoc87] J. M. Hoc, **Psychologie cognitive de la planification**, Presses Universitaires de Grenoble, Grenoble, 1987.

[Hor83] P. W. Horstmann, **Automation of the design for testability using logic programming**, [Phd dissertation, Syracuse University], Syracuse, Dec. 1983.

[Hor84] P. W. Horstmann, **A knowledge-based system using design for testability rules**, 14th IEEE Fault Tolerant Computing Symposium, Floride, Juin 1984,

278-284.

[Jer89] A. A. Jerraya, **Contribution à la compilation de silicium et au compilateur SYCO**, [Th. de Doctorat d'Etat, I.N.P.Grenoble], Grenoble, Dec. 1989.

[JoB87] N. A. Jones et K. Baker, **Knowledge-based system tool for high-level BIST design**, Microprocessors and Microsystems 1, 11 (Jan. 1987), 35-40.

[KRK88] A. Kara, R. Rastogi et K. Kawamura, **An expert system to automate timing design**, IEEE Design and Test of Computers 5, 5 (Oct. 1988), 28-40.

[KeS87] J. S. Keen et R. E. Seviora, **Expert systems for VLSI design : a survey and a perspective**, Journées Internationales d'Avignon sur les S.E. et leurs Applications , (Juin 1987), 1447-1458.

[Kel84] V. E. Kelly, **The CRITTER system - automated critiquing of digital circuit design**, 21st ACM/IEEE Design Automation Conference, Las Vegas, Juin 1984, 419-425.

[KiM88] T. Kirkland et M. R. Mercer, **Algorithms for automatic test pattern generation (Tutorial)**, IEEE Design and Test of Computers 5, 3 (Juin 1988), 43-55.

[KMZ79] B. Koenemann, J. Mucha et G. Zwiehoff, **Built-in logic block observation techniques**, 10th IEEE International Test Conference, Cherry Hill, Oct. 1979, 37-41.

[Kow84] T. J. Kowalski, **The VLSI Design Automation Assistant : a knowledge-based expert system**, [Phd dissertation, Carnegie-Mellon University], Pittsburgh, Avr. 1984.

[KGW85] T. J. Kowalski, D. J. Geiger, W. H. Wolf et W. Fichtner, **The VLSI Design Automation Assistant : from algorithms to silicon**, IEEE Design and Test of Computers 4, 2 (Aout 1985), 33-43.

[Kri87] B. Krishnamurthy, **Hierarchical test generation : can AI help ?**, 18th IEEE International Test Conference, Washington, Sep. 1987, 694-700.

[LaC86] D. O. Lahti et G. C. Chen-Ellis, **PROSPECT: a production system for partitioning and evaluating chip testability**, 17th IEEE International Test Conference, Washington, 1986, 360-367.

[LCG89] J. C. Laprie, B. Courtois, M. C. Gaudel et D. Powell, **Sûreté de fonctionnement des systèmes informatiques**, Dunod, Paris, 1989.

[Lau82a] J. L. Laurière, **Représentation et utilisation des connaissances (1)**, Techniques et Sciences Informatiques 1, 1 (1982), 25-42.

[Lau82b] J. L. Laurière, **Représentation et utilisation des connaissances (2)**, Techniques et Sciences Informatiques 1, 2 (1982), 109-132.

[LBK88] S. M. Lea, N. Brown, T. Katz et P. Collins, **Expert system for the functional test program generation of digital electronic circuit boards**, 19th IEEE International Test Conference, Washington, Sep. 1988, 209-220.

[Lux90] A. Lux, **Sur l'evolution de l'intelligence artificielle**, La Machine Perceptive-Journees francophones sur l'informatique, Grenoble, Jan. 1990, 09-21.

[MLC89] M. Marzouki, J. Laurent et B. Courtois, **A unified use of deep and shallow knowledge in an expert system for prototype validation of integrated circuits**, Journées Internationales d'Avignon sur les S.E. et leurs Applications, Avignon, Mai 1989, 55-69.

[Mau83] C. Maunder, **HITEST test generation system - interfaces**, 14th IEEE International Test Conference, Washington, Oct. 1983, 324-332.

[Mil86] G. Milne, **Towards verifiably correct VLSI design**, in Formal aspects of VLSI design, G. Milne et P. A. Subrahmanyam (ed.), North-Holland, 1986.

[Mul84] R. Mullis, **An expert system for VLSI tester diagnosis**, 15th IEEE International Test Conference, Philadelphie, Oct. 1984, 196-199.

[Nil80] N. J. Nilsson, **Principles of artificial intelligence**, Tioga, Palo Alto, 1980.

[Par88] P. S. Parry, **Artificial intelligence techniques in HITEST**, IFIP workshop on knowledge-based systems for test and diagnosis, Grenoble, Sep. 1988, 158-163.

[Rao89] O. Raoult, **Diagnostic de pannes des systèmes complexes**, [Th. de Docteur Ingenieur, I.N.P.Grenoble], Grenoble, Fev. 1989.

[Rau87] J. C. Rault, **Maintenance et système experts : besoins et marchés**, in Systèmes experts en maintenance, M. Gabriel et J. C. Rault (ed.), Masson, Paris, 1987, 22-31.

[Rei87] R. Reiter, **A theory of diagnosis from first principles**, Artificial Intelligence 87, 32 (1987), 57-95.

[Ric87] E. Rich, **Intelligence artificielle**, Masson, Paris, 1987.

[RLG89] C. Robach, D. Lutoff et N. Garcia, **Knowledge-based functional specification of test and maintenance programs**, IEEE Transactions on Computer-Aided Design 8, 11 (Nov. 1989), 1145-1156.

[Rob65] J. A. Robinson, **A machine-oriented logic based on the resolution principle**, Journal of the ACM 12, (1965), 23-41.

[Rob83] G. D. Robinson, **HITEST - intelligent test generation**, 14th IEEE International Test Conference, Washington, Oct. 1983, 311-323.

[RBS67] J. P. Roth, W. G. Bouricius et P. R. Schneider, **Programmed algorithm to compute tests to detect and distinguish faults in logic circuits**, IEEE Transactions on Electronic Computers ec-16, 5 (1967), 567-580.

[RyW85] P. M. Ryan et A. J. Wilkinson, **Knowledge acquisition for ATE diagnosis**, 16th IEEE International Test Conference, Philadelphie, Nov. 1985, 848-856.

[ShD83] M. Shirley et R. Davis, **Digital test generation from hierarchical models and symptom information**, International Conference on Computer Design, , Nov. 1983.

[Sho76] E. H. Shortliffe, **Computer-based medical consultations : MYCIN**, Elsevier, New York, 1976.

[Som88] L. Sombé (Collectif), **Inférences non classiques en intelligence artificielle**, Actes des Journées Nationales du PRC-GRECO IA, Toulouse, Mars 1988, 137-230.

[SWL86] N. C. E. Srinivas, A. S. Wojcik et Y. H. Levendel, **An artificial intelligence based implementation of the P-algorithm for test generation**, 17th IEEE International Test Conference, Washington, Sep. 1986, 732-739.

[StM84] L. I. Steinberg et T. M. Mitchell, **A knowledge-based approach to VLSI CAD. The REDESIGN system**, 21st ACM/IEEE Design Automation Conference, Las Vegas, Juin 1984, 412-418.

[ThA78] S. M. Thatte et J. A. Abraham, **A methodology for functional level testing of microprocessors**, 8th IEEE Fault Tolerant Computing Symposium, Toulouse, Juin 1978, 90-95.

[ThA79] S. M. Thatte et J. A. Abraham, **Test generation for general microprocessor architectures**, 9th IEEE Fault Tolerant Computing Symposium, Madison, Juin 1979, 203-210.

[ThI88] K. Thearling et R. Iyer, **Diagnostic reasoning in digital systems**, 18th IEEE Fault Tolerant Computing Symposium, Tokyo, Juin 1988, 286-291.

[Tur50] A. M. Turing, **Computing machinery and intelligence**, Mind 59, (Oct. 1950), 433-460.

[Van80] W. Van Melle, **A domain-independent system that aids in constructing knowledge-based consultation programs**, [PhD Dissertation, Stanford University], Stanford, Juin 1980.

[Wad78] R. L. Wadsack, **Fault modelling and logic simulation of CMOS and MOS integrated circuits**, The Bell System Technical Journal 57, 5 (Mai 1978), .

[Wei66] J. Weizenbaum, **ELIZA-a computer program for the study of natural language communication between man and machine**, Communications of the ACM 9, 1 (Jan. 1966), 36-44.

[Wha83] D. J. Wharton, **The HITEST test generation system - overview**, 14th IEEE International Test Conference, Washington, Oct. 1983, 302-310.

[Wil84] A. J. Wilkinson, **A method for test system diagnosis based on the principles of artificial intelligence**, 15th IEEE International Test Conference, Philadelphie, Oct. 1984, 188-195.

[WST84] A. J. Wilkinson, L. Stemke, J. Tucci, D. Millard, L. Deerr et J. Deerr, **CHECK941 program description**, Tech. Rep. 84, Teradyne Inc., Woodland Hills, 1984.

[Wil86] A. J. Wilkinson, **Benchmarking an expert system for electronic diagnosis**, 17th IEEE International Test Conference, Washington, Sep. 1986, 964-971.

[WiP82] T. W. Williams et K. P. Parker, **Design for Testability : a survey**, IEEE Transactions on Computers c-31, 1 (Jan. 1982), 2-15.

[Yau86] C. W. Yau, **Concurrent test generation using AI techniques**, 17th IEEE International Test Conference, Washington, Sep. 1986, 722-731.

DEUXIEME PARTIE
VALIDATION DE PROTOTYPES DANS LE CADRE D'UN TEST PAR FAISCEAU D'ÉLECTRONS

TABLE DES MATIERES

I. INTRODUCTION A LA DEUXIEME PARTIE 103

II. LE TEST DE CIRCUITS INTEGRES PAR FAISCEAU D'ELECTRONS 105

II.1 INTRODUCTION 105

II.2 PREMIERE GENERATION : T.F.E. 106

II.2.1 Présentation générale 106

II.2.2 T.F.E. et observabilité 107

II.2.2.1 Généralités 108

II.2.2.2 Le contraste de potentiel 108

II.2.2.3 Choix de la technique d'observation 110

II.2.3 T.F.E. et contrôlabilité 110

II.2.4 Sonde mécanique vs. M.E.B. 112

II.2.4.1 Possibilités d'observabilité 112

II.2.4.2 Effets secondaires 113

II.2.4.3 Performances 114

II.2.5 Problèmes non résolus par le M.E.B. 114

II.2.5.1 Règles à suivre pour l'observabilité 114

II.2.5.2 Règles à suivre pour la précision 115

II.2.5.3 Règles à suivre pour l'utilisation pratique 115

II.2.6 Applications possibles 115

II.3 DEUXIEME GENERATION : T.F.E. ETENDU 117

II.3.1 Système de conditionnement 118

II.3.2 Système de saisie et prétraitement de données 119

II.3.2.1 Saisie et prétraitement en mode image 119

II.3.2.2 Saisie et prétraitement en mode forme d'onde 120

II.3.3 Système de traitement des informations 120

II.3.3.1 Traitement des informations en mode image 121

II.3.3.2 Traitement des informations en mode forme d'onde 122

II.3.4 Quelques T.F.E. étendus existants 123

II.4 TROISIEME GENERATION : SYSTEME INTEGRE DE TEST 124

II.5 CONCLUSION 126

III. ADVICE : APPROCHE DICTIONNAIRE DE FAUTES 127

III.1 INTRODUCTION 127

III.2 DESCRIPTION GENERALE DU SYSTEME 128

III.2.1 Généralités 128
III.2.2 Utilisation des données en provenance d'outils de C.A.O. 129
III.2.3 Description du système en termes de flux d'opérations 129
III.3 STRATEGIES POUR LE TEST PAR FAISCEAU D'ELECTRONS 131
III.3.1 Conception en vue du test par faisceau d'électrons 131
III.3.1.1 Règles de conception logique 132
III.3.1.2 Règles topographiques 132
III.3.1.3 Règles de sélection de points de test 133
III.3.2 Simulation en vue du test par faisceau d'électrons 133
III.3.3 Génération de stimuli en vue du test par faisceau d'électrons 136
III.3.4 Dictionnaire de fautes en vue du test par faisceau d'électrons 138
III.4 LA STRUCTURE DE DONNEES DU SYSTEME ADVICE 141
III.4.1 Première phase : génération d'une structure intermédiaire 141
III.4.2 Deuxième phase : génération de la structure de données ADVICE 142
III.4.2.1 Description logique hiérarchique 142
III.4.2.2 Description électrique mise à plat 143
III.5 LA COUCHE METHODE DU SYSTEME ADVICE 143
III.5.1 Organisation générale de la couche méthode 144
III.5.1.1 Classe 1 : commandes de type SET 144
III.5.1.2 Classe 2 : commandes de type SHOW 145
III.5.1.3 Classe 3 : commandes de type FILE HANDLING 146
III.5.1.4 Classe 4 : commandes de type EXEC 147
III.5.1.5 Classe 5 : commandes de type SYSTEM 148
III.5.2 L'algorithme de guidage de la sonde 148
III.6 CONCLUSION 150

IV. PESTICIDE : APPROCHE BASEE SUR LA CONNAISSANCE 153
IV.1 INTRODUCTION 153
IV.2 ENVIRONNEMENT DE PESTICIDE 154
IV.2.1 Système d'Acquisition des Informations 154
IV.2.2 Système de Traitement des Informations 155
IV.2.3 Intégration des différents composants 156
IV.2.4 Limitation liée à l'environnement 158

IV.3 MODELISATION DES CONNAISSANCES 158
IV.3.1 Description des différents constituants d'un circuit 160
IV.3.1.1 Le bloc 160
IV.3.1.2 L'interface bloc 161
IV.3.1.3 La connexion 163
IV.3.2 Connaissances additionnelles 164
IV.3.2.1 Propriétés fonctionnelles 164
IV.3.2.2 Hypothèses 166
IV.4 ORGANISATION DES CONNAISSANCES 167
IV.4.1 Base de connaissances structurelles 167
IV.4.2 Base de connaissances fonctionnelles 168
IV.4.3 Base de connaissances comportementales 169
IV.5 VERIFICATION DE LA COHERENCE DES INFORMATIONS 170
IV.5.1 Incohérences intrinsèques 171
IV.5.2 Incohérences relatives au domaine 172
IV.5.3 Evaluation dans le cadre des recherches en vérification de la cohérence 173
IV.5.4 Mise en oeuvre du processus dans PESTICIDE 174
IV.6 DETECTION ET LOCALISATION DE FAUTES 176
IV.6.1 Evaluation de la validité d'un instant de mesure 176
IV.6.1.1 Examen de l'état d'un bloc 176
IV.6.1.2 Examen de l'état d'une connexion 178
IV.6.2 Détection de fautes sur les connexions 178
IV.6.3 Localisation de fautes à travers le circuit 179
IV.6.3.1 Cas de faute simple dans les circuits combinatoires 181
IV.6.3.2 Cas de faute multiple dans les circuits combinatoires 182
IV.6.3.3 Cas des circuits séquentiels 183
IV.7 IMPLEMENTATION ET FONCTIONNEMENT 184
IV.7.1 Cas de faute simple 185
IV.7.1.1 Faute manifestant des erreurs directes 185
IV.7.1.2 Faute manifestant des erreurs propagées 185
IV.7.2 Cas de faute multiple 188
IV.8 TRAITEMENTS SPECIFIQUES APRES LOCALISATION 188
IV.9 EXPLICABILITE DU FONCTIONNEMENT DU SYSTEME . . . 190
IV.9.1 Gestion d'une pile de raisonnement 191
IV.9.1.1 Méthode 191
IV.9.1.2 Problèmes posés et commentaires 192

IV.9.2 Analyse de la trace d'exécution 193
IV.9.2.1 Format de la trace 193
IV.9.2.2 Méthode 194
IV.10 CONCLUSION 194

V. EVALUATION ET COMPARAISON DES DEUX APPROCHES 199
V.1 INTRODUCTION 199
V.2 EVALUATION DU SYSTEME ADVICE 200
V.2.1 Evaluation du système dans sa globalité 200
V.2.2 Contribution du projet ADVICE au développement de l'état de l'art 201
V.2.3 Evaluation de la couche méthode 202
V.3 EVALUATION DU SYSTEME PESTICIDE 203
V.3.1 PESTICIDE et son environnement 204
V.3.2 Modélisation, organisation et utilisation des connaissances 205
V.3.3 PESTICIDE, système à base de connaissances de seconde génération 207
V.4 COMPARAISON DES DEUX APPROCHES DE DIAGNOSTIC 207
V.4.1 Dictionnaire de fautes vs. système à base de connaissances 208
V.4.1.1 Génération de la base de données 208
V.4.1.2 Consultation de la base de données 209
V.4.1.3 Performances des deux méthodes 209
V.4.2 ADVICE vs. PESTICIDE 210
V.5 CONCLUSION 212

VI. CONCLUSION DE LA DEUXIEME PARTIE 213

Bibliographie 215

LISTE DES FIGURES

Figure II.1. Représentation en couches d'un système intégré de test 106

Figure II.2. Paramètres pour le choix d'une technique d'observation 111

Figure II.3. Evaluation de méthodes de contrôlabilité interne par faisceau d'électrons 113

Figure II.4. Caractéristiques des T.F.E. étendus recensés 125

Figure III.1. Séquence d'opérations lors d'une session ADVICE 130

Figure III.2. Illustration des définitions sur un circuit exemple 135

Figure III.3. Expériences de génération de dictionnaire de fautes 140

Figure III.4. Déroulement de l'algorithme de guidage 149

Figure IV.1. Schéma fonctionnel du S.A.I. 155

Figure IV.2. Schéma fonctionnel du S.T.I. 156

Figure IV.3. Vue synoptique du système intégré de test 157

Figure IV.4. Décomposition hiérarchique de la description structurelle 159

Figure IV.5. Modélisation d'un circuit combinatoire 160

Figure IV.6. Modélisation d'un circuit séquentiel 160

Figure IV.7. Notion de cône de couverture 165

Figure IV.8. Utilité de la notion de cône de couverture 165

Figure IV.9. Mise en oeuvre de l'approche mixte dans PESTICIDE 175

Figure IV.10. Détection de faute sur une connexion 179

Figure IV.11. Localisation de faute : faute simple, circuit combinatoire 182

Figure IV.12. Localisation de faute : faute multiple, circuit combinatoire 183

Figure IV.13. Recherche d'instances valides d'interface bloc 184

Figure IV.14. Trace d'exécution de la clause DIRECTFAULT 186

Figure IV.15. Trace d'exécution de la clause PROPAGFAULT 187

Figure IV.16. Trace d'exécution de la clause POSSIBLEFAULT 189

Figure IV.17. Architecture globale du système PESTICIDE 195

Figure IV.18. Résultats obtenus en cas de Faute Simple Combinatoire et Faute Multiple Combinatoire (FSC et FMC en secondes) 195

Figure IV.19. Résultats obtenus en cas de Faute Simple Séquentielle et Faute Multiple Séquentielle (FSS et FMS en secondes) 196

Figure V.1. Schéma synoptique de l'interface 205

I. INTRODUCTION A LA DEUXIEME PARTIE

La deuxième partie de cette thèse est consacrée au diagnostic de circuits intégrés observés au microscope électronique à balayage. C'est en effet dans ce contexte que se sont plus précisément inscrits nos travaux.
Le projet de système intégré de test, actuellement en cours de développement au laboratoire TIM3/IMAG, s'est donné pour but d'allier un outil de test performant, le microscope électronique à balayage, à une nouvelle approche du problème de la localisation de fautes dans les circuits complexes, basée sur l'utilisation de méthodes de haut niveau de façon à entièrement automatiser le processus de diagnostic.

Ce projet, qui se situe dans la continuation du projet ACIME (Analyse de Circuits Intégrés par Microscopie Electronique) [Lau84] mené depuis janvier 1981, induit donc deux thèmes de recherche, qui sont : d'une part, **l'analyse de défaillances** pour des circuits à **structure non connue** que l'on compare à des circuits de référence, réputés bons [BBC83], [CCC89], et d'autre part, **la validation de prototypes** pour des circuits à **structure connue** et sur lesquels un grand nombre d'informations concernant le processus de conception est disponible (structure, résultats de simulation, etc.) [BBC83], [GMC87] ; c'est ce deuxième thème qui sera développé plus particulièrement dans la suite.

Dans les deux cas, le circuit sous test est soumis à une observation, réalisée par un système d'acquisition et de traitement d'images organisé autour du microscope électronique utilisé en mode contraste de potentiel. Cet outil, dont l'utilisation est justifiée par ses puissantes possibilités d'observabilité, pose justement un problème de gestion et d'exploitation de la masse de données qu'il peut fournir à propos du circuit en observation, ce qui conduit à la nécessité de développer des méthodes de traitement automatique des résultats d'observation.

Considérant le fait qu'à l'heure actuelle, dans le contexte scientifique international de la validation de prototypes de circuits intégrés par faisceau d'électrons, il n'existe pas d'outil de **diagnostic automatique,** nous nous sommes intéressée à l'étude de méthodes de haut niveau pour l'interprétation des résultats issus de l'observation au microscope électronique, qui seule permettent d'automatiser entièrement le processus de diagnostic.
Dans ce but, deux approches différentes ont fait l'objet de nos investigations, la première étant basée sur l'utilisation d'un dictionnaire de fautes, tandis que la deuxième s'appuie sur l'emploi d'un système à base de connaissances.

Cette seconde partie a donc pour objet de présenter et comparer ces deux approches, et se décompose comme suit :
Le chapitre II présente le contexte dans lequel nous avons travaillé, c'est-à-dire le test de circuits intégrés par faisceau d'électrons.
Le chapitre III décrit la première approche de diagnostic - celle basée sur un dictionnaire de fautes -, et est illustrée par la description du système ADVICE (Automatic Design Validation of Integrated Circuits using Electron-beam), réalisé dans le cadre du projet du même nom auquel nous avons participé (programme ESPRIT-CCE N. 271).
Le chapitre IV concerne la deuxième approche de diagnostic - celle basée sur un système à base de connaissances -, et est illustrée par la description du système PESTICIDE (a Prolog-written Expert System as a Tool for Integrated CIrcuits DEbugging), que nous avons développé dans le laboratoire.
Enfin, nous tenterons dans le chapitre V d'évaluer et de comparer les apports et lacunes des deux approches présentées, ainsi que des deux systèmes dans lesquels elles sont mises en oeuvre.

II. LE TEST DE CIRCUITS INTEGRES PAR FAISCEAU D'ELECTRONS

II.1 INTRODUCTION

Malgré l'existence de nombreuses méthodes de conception adaptée au test [WiP82], les outils de test basés sur des méthodes mécaniques, même lorsqu'ils permettent d'observer certains points internes du dispositif en examen, et même s'ils sont couplés à des méthodes de recherche de fautes très élaborées, ne permettent plus de faire face à l'extrême complexité des circuits tels qu'ils sont conçus actuellement, et tels qu'ils continueront de l'être.
La mise au point et l'usage d'outils permettant de repousser ces limites du test de circuits VLSI se sont donc tout naturellement imposés. Parmi ces outils, le Testeur à Faisceau d'Electrons (T.F.E.) est peut-être le plus performant à l'heure actuelle, en termes d'observabilité du dispositif à tester. Bien que d'une utilisation relativement récente dans le domaine du test de circuits intégrés, le T.F.E. fait déjà l'objet de toute l'attention de la communauté du test, tant dans les environnements universitaires qu'industriels, comme en témoignent les publications référencées au long de ce chapitre.

Le T.F.E. est basé sur un Microscope Electronique à Balayage (M.E.B.). Néanmoins, le M.E.B. n'étant pas intrinsèquement dédié à l'observation de circuits intégrés, il faut adapter certaines de ses fonctionnalités (par spécialisation et/ou optimisation) de façon à permettre son utilisation dans le cadre d'une telle application, et développer un certain nombre d'autres outils qui, lui étant couplés, pourront former un véritable équipement de test, ou testeur.
En outre, un testeur seul ne pouvant être que d'un intérêt limité pour l'utilisateur, il faut "habiller" le T.F.E. d'outils et/ou méthodes permettant de contrôler l'équipement de base, et de gérer les informations issues de l'observation. On obtient ainsi un "testeur étendu", capable de décharger l'utilisateur des tâches fastidieuses de gestion et mise en forme des données observées.
Enfin, pour arriver à assister l'utilisateur dans sa tâche de diagnostic, le stade ultime est de développer des méthodes automatiques d'interprétation des résultats observés, de façon à concevoir un système complet, et de préférence intégré, de mise au point de prototypes de circuits complexes.
On peut donc voir ce système intégré de diagnostic comme un "système en couches", dont le noyau serait le M.E.B., les deux couches intermédiaires formant respectivement le T.F.E. et le "T.F.E. étendu", et dont la couche supérieure serait formée des diverses méthodes de diagnostic (figure II.1). Cette taxonomie, que nous donnons ici en termes de "valeur ajoutée" par chaque couche supplémentaire, correspond tout à fait à la classification opérée

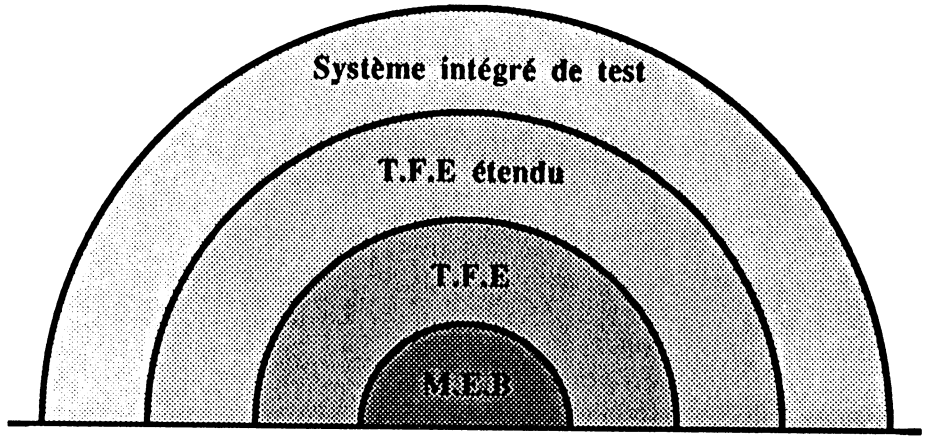


Figure II.1. Représentation en couches d'un système intégré de test

par [CCC89] en termes de "générations" de T.F.E. : COLLIN et al. y distinguent trois générations, dont la première n'offre à l'utilisateur que les outils de base lui permettant l'observation du circuit, la deuxième prenant en charge le traitement automatique des données observées, alors que la troisième génération, encore au stade de la recherche, devrait assurer également l'ensemble des tâches décisionnelles, c'est-à-dire le diagnostic proprement dit, et donc la localisation de la ou des fautes dans le circuit sous test.

Ce chapitre sera consacré au détail des deux couches les plus externes, et des interactions entre chaque niveau. Nous nous attacherons dans les paragraphes qui suivent à en décrire, lorsqu'il en existe, ou du moins lorsque nous en avons trouvé trace dans la littérature, les différentes applications et réalisations, qu'elles soient au stade de recherche ou de commercialisation, dans le but d'essayer de dresser un état de l'art dans le domaine.

II.2 PREMIERE GENERATION : T.F.E.

Bien que notre propos ne soit pas de décrire de façon précise et détaillée le principe de fonctionnement du T.F.E. de base, nous essaierons d'en donner une idée générale dans ce paragraphe, de manière à faciliter la compréhension de la suite de ce chapitre. Le lecteur intéressé pourra trouver plus de précisions dans [Lau84], [Mic88].

II.2.1 Présentation générale

Le principe est le suivant : le canon à électrons du M.E.B. émet un faisceau dirigé vers un point particulier du dispositif à observer. La rencontre électrons-matière peut se solder soit par un choc élastique, et dans ce cas les électrons incidents sont rétrodiffusés dans la chambre du M.E.B., soit par une recombinaison, avec émission d'électrons, parmi lesquels des électrons dits secondaires. Ce sont ces électrons qui, captés par le détecteur du M.E.B., serviront essentiellement à recomposer une image du dispositif observé.

Ceci s'explique par le fait que, en mode observation, les électrons incidents (composant le faisceau du M.E.B.), ont une énergie de l'ordre de 1000 électrons-Volts (eV),[1] ce qui se traduit par une tension d'accélération de 1000 Volts, alors que les plaques du détecteur sont portées à un potentiel positif de l'ordre de 200 à 300 Volts. Or, les électrons rétrodiffusés ne perdant que très peu d'énergie dans le choc élastique - la perte est de l'ordre de 1/10 -, ils ne sont donc pas fortement attirés par les plaques du détecteur, et se répartissent de manière égale dans toutes les zones de la chambre à vide du M.E.B.
Les électrons secondaires, par contre, ont une énergie très faible - de l'ordre de quelques eV -, et seront donc attirés très fortement par les plaques du détecteur ; ils vont donc recomposer, grâce au balayage du faisceau, l'image complète du dispositif observé.

Ce phénomène n'est bien sûr pas particulier au T.F.E., puisqu'il est utilisé depuis bien longtemps dans les laboratoires de métallurgie, biologie, physique des composants électroniques, etc.[2]
Les modifications à effectuer sur un M.E.B. pour en faire un T.F.E. concernent les caractéristiques suivantes :

- possibilité de fonctionnement en mode contraste de potentiel
- possibilité de contrôle du faisceau (positionnement et hachage)
- installation de détecteur spécial (observation en mode statique) et de spectromètre (observation en mode dynamique)

Il est d'autre part nécessaire d'effectuer l'adjonction d'un ou plusieurs outils ayant principalement en charge :

- le contrôle de la platine porte-objet du microscope
- le conditionnement du circuit (i.e. son alimentation)
- la numérisation du signal détecté

II.2.2 T.F.E. et observabilité

1. Cette valeur de 1 KeV (en réalité, 1 à 2 KeV) offre le meilleur compromis pour l'observation : plus importante, elle pourrait causer une dégradation du circuit sous test, mais plus faible, elle produirait une image trop bruitée.
2. Dans l'historique qu'il dresse du test par faisceau d'électrons [Luk87], LUKIANOFF situe aux alentours de 1942 les premiers M.E.B.s, alors qu'il faut attendre 1965 pour en voir les premières applications au test de circuits intégrés.

II.2.2.1 Généralités

Nous entendrons par observabilité tout moyen d'obtention d'informations sur la structure, le fonctionnement et/ou le comportement du circuit sous test, étant entendu que le circuit est alimenté : plus précisément, une observation pourra résulter aussi bien en une *image* qu'en une *valeur mesurée*.
Une image du circuit peut être soit une image en mode *"états stables"*, ou encore une image *stroboscopique*. Une valeur mesurée correspond soit à une mesure du *temps de propagation* d'un signal à travers le circuit, soit à une mesure de *l'évolution du potentiel dans le temps* en un point de ce circuit. Dans tous les cas, l'observation repose sur le principe du *contraste de potentiel*.

II.2.2.2 Le contraste de potentiel

La découverte, dans les années 50, du phénomène de contraste de potentiel a permis de considérer l'utilisation d'un M.E.B. en T.F.E. Cette découverte découle du fait que le nombre d'électrons secondaires émis par le dispositif subissant le faisceau incident du M.E.B. est une fonction du potentiel de surface de ce dispositif [Col83], [BaL82].
En effet, l'influence du potentiel de surface de l'échantillon est telle que le champ électrique d'attraction des électrons secondaires par le détecteur du M.E.B. est plus faible pour un point du circuit porté à un potentiel positif que pour un point porté à un potentiel nul. Cette modification de l'énergie des électrons secondaires en fonction du point d'impact du faisceau agit donc sur l'efficacité de leur collection.
Ce phénomène se visualise par une image du circuit en niveaux de gris, sur laquelle les zones portées à un potentiel positif seront d'autant plus sombres que ce potentiel est élevé : une image "idéale" montrerait en blanc les lignes portées à la masse et en noir celles portées à un potentiel correspondant au "1" logique.

Le phénomène de contraste de potentiel est donc à la base de tous les modes et techniques d'observation de circuits intégrés par microscopie électronique, moyennant l'utilisation d'équipements additionnels (stroboscopie de faisceau, obtention de la fréquence de balayage et sa synchronisation avec la fréquence de fonctionnement du circuit sous test, etc.), mais dont le fonctionnement physique et la réalisation électronique ne seront pas explicités ici.

Les différents modes et techniques d'observation peuvent se subdiviser en deux grandes catégories, qui sont :

- **Le mode image** : le résultat obtenu est une image en niveaux de gris, permettant de déduire, après traitement, une correspondance niveaux de gris/valeurs de potentiel. Le mode image se subdivise à son tour en deux modes de fonctionnement, suivant qu'il s'agit d'une observation statique ou dynamique.

* *Mode image statique* : les images acquises concernent soit des circuits combinatoires, soit des circuits séquentiels statiques ; on observe dans ce mode des états stables du circuit, et on peut détecter des fautes comme des coupures ou des courts-circuits.
  Si de plus on synchronise la fréquence du balayage du faisceau avec la fréquence (ou l'un de ses multiples ou sous-multiples) de fonctionnement du circuit, on peut mettre en oeuvre la technique dite de "voltage coding" [Col83], qui permet de traduire sur l'image une séquence temporelle d'un signal en une séquence spatiale, et ceci en tous points du circuit par lesquels est propagé ce signal. Pour citer un exemple "parlant", une telle technique permet de transposer le chronogramme d'un signal d'horloge en une ligne sur l'image, composée d'une alternance de bandes noires et blanches, correspondant respectivement aux états logiques hauts et bas du chronogramme.

* *Mode image dynamique* : ce mode utilise le principe de la stroboscopie pour moduler le faisceau, et permet de prendre en compte non plus seulement les dimensions spatiales, mais également la dimension temporelle dans l'observation du comportement du circuit sous test.
  Par modulation du faisceau incident en impulsions de très courtes durées, on peut obtenir une série d'"images instantanées" du circuit, dont chacune constitue un plan du "cube d'images". Ce cube est donc formé d'un certain nombre d'images spatiales, échantillonnées à différents instants.
  L'analyse de ces images instantanées permet d'observer l'évolution dans le temps des états du circuit, chaque image représentant l'état *au même instant* de tous les points de la zone observée.
  Une autre application du mode image dynamique est la technique dite de "logic state mapping".[3] Dans ce cas, le faisceau incident balaye toujours la même ligne [Lau84], avec un décalage de phase à chaque balayage. On obtient par cette méthode une carte ("mapping") de l'évolution dans le temps des états logiques de tous les points sur la ligne considérée.

- **Le mode forme d'onde** : le résultat obtenu est dans ce cas une valeur mesurée de l'amplitude d'un signal, destinée à être comparée - de façon automatique en général - à une valeur attendue, de même format [MRP87], [LyB87].
  Le mode forme d'onde permet la représentation graphique de l'évolution du potentiel

3. Cette méthode pourrait également s'appliquer en mode statique, sans utiliser la stroboscopie de faisceau, bien que citée dans le cadre des techniques d'observation en mode dynamique ; c'est néanmoins dans ce contexte qu'elle prend tout son sens.

dans le temps en un point particulier que l'on veut observer ; pour ce faire, le faisceau incident reste positionné sur ce point : il s'agit donc d'une mesure *locale* plutôt que globale.
Ce mode permet d'obtenir des mesures qualitatives (utilisation en analyseur logique) ou quantitatives (utilisation en oscilloscope) très précises.[4]

II.2.2.3 Choix de la technique d'observation

Le choix d'une technique d'observation se fait en fonction de plusieurs paramètres, qui sont :

- le type d'information que l'on veut obtenir : mesures qualitatives ou quantitatives, mesures en un point du circuit ou dans une zone plus vaste, mesures de délai ou de potentiel, mesures statiques ou dynamiques, etc.
- le type de circuit que l'on veut observer : circuits combinatoires, séquentiels statiques ou séquentiels dynamiques.
- le type de moyens que l'on peut mettre en oeuvre : T.F.E. lui-même, autres équipements, etc.

Tout ou partie de ces paramètres sont bien entendu corrélés. Ces corrélations, ainsi que la correspondance type d'observation/paramètre(s) considéré(s) sont résumées en figure II.2 (tableau établi à partir de [LaC83]).

II.2.3 T.F.E. et contrôlabilité

Les moyens d'observabilité offerts par les testeurs à faisceau d'électrons sont très puissants, certes, mais peuvent devenir très coûteux, en temps[5], et aussi en gestion des informations obtenues.
En effet, une observation n'est finalement "utile" que lorsqu'elle met en évidence les erreurs propagées à travers le circuit sous test, à partir du siège de la ou des fautes. Le seul moyen connu d'exhiber ces erreurs est d'alimenter le circuit par des vecteurs de test permettant de sensibiliser certains chemins sur lesquels les erreurs se manifesteront.
Cette méthode nécessite la mise en oeuvre de techniques de génération de vecteurs de test

---

4. MAY et al. font état dans [MSM84] de mesures effectuées avec une résolution de l'ordre de 10 millivolts.
5. Malgré des temps de traitement très peu élevés ([CCC89] annonce 40 millisecondes pour le traitement d'une image 512x512 pixels par un processeur spécialisé), il faut se rendre à l'évidence que, si l'on peut améliorer les temps pris par les actions de digitalisation et de stockage en mémoire de l'image, le temps d'acquisition de l'image lui-même ne peut guère être ramené au-dessous du temps nécessaire au balayage de l'image et à l'acquisition d'un signal vidéo.

technique d'observation	type de mesure	type de circuit	type et champ d'investigation	moyens : TFE + ···
image statique	statique	comb. seq. stat.	états stables d'une zone	--
image statique (V.C.)	synchro. fréquences	comb. seq. stat.	états stables d'un point	synchro. + div. de freq.
image dynamique (cube)	strobo. de faisceau	comb. seq. stat. séquentiel	propagation dans une zone	équipement strobo.
image dynamique (L.S.M.)	strobo. de faisceau	comb. seq. stat. séquentiel	propagation sur une ligne	équipement strobo. + décalage ϕ
forme d'onde (mes. qualit.)	analys. logique	comb. seq. stat. séquentiel	propagation en un point	équipement traitement signal
forme d'onde (mes. quant.)	oscil-loscope	comb. seq. stat. séquentiel	propagation en un point	équipement traitement signal

Figure II.2. Paramètres pour le choix d'une technique d'observation

[BrF76], [Fuj85], dont le principe général est le suivant :

- Choix de la faute f_k à tester, c'est-à-dire choix du modèle de faute p_i à appliquer en un point x_j du circuit - on a donc $f_k=(p_i,x_j)$.
- Propagation ("sensitizing") de la faute f_k jusqu'aux sorties primaires du circuit (phase de sensibilisation de chemin).
- Propagation arrière ("backtracing") à partir du siège de la faute f_k jusqu'à trouver les valeurs des entrées primaires nécessaires à la manifestation de cette faute (phase de génération proprement dite des vecteurs de test).

Ce principe général, sur lequel sont basées toutes les méthodes de génération de vecteurs de test, est le seul applicable pour ce qui est du test structurel, lorsque l'on ne peut agir sur des points du circuit autres que ses entrées primaires. Il pose de nombreux problèmes, comme la longueur des chemins à sensibiliser, et le caractère combinatoire du processus de propagation arrière. De plus, il faut choisir, dans le cas d'un test structurel, le niveau de description - niveau logique, niveau transistors ("switch-level"), etc. - auquel on veut tester le circuit, étant entendu qu'à chaque niveau de description correspond un ensemble de modèles de fautes, plus ou moins représentatifs des fautes réelles pouvant affecter le circuit.[6]

Par contre, ces problèmes peuvent être en grande partie résolus lorsque l'on a accès à des points internes du circuit, non seulement en observabilité, mais également en contrôlabilité.[7]

Etant donnés les résultats obtenus en observabilité par le test à faisceau d'électrons, des études ont été menées sur les possibilités d'utiliser la sonde électronique comme instrument de contrôle du circuit sous test [Mic88], de façon à toujours se prémunir contre les risques de dégradation engendrés par les sondes matérielles d'une part, et d'autre part à pouvoir utiliser le même type de sonde (faisceau d'électrons) en observabilité comme en contrôlabilité, puisqu'il est clair que le premier mode tirera parti du deuxième.
[Mic88] présente les différentes méthodes de contrôlabilité par faisceau d'électrons, que nous listons brièvement ci-après :

- Charge d'une équipotentielle
- Courant induit par faisceau d'électrons (EBIC : "Electron Beam Induced Current")
- Charge de grille flottante
- Modification des paramètres des transistors
- Utilisation de transistors à oxyde épais

Les conclusions de l'étude de [Mic88] sont résumées dans le tableau suivant (figure II.3).

II.2.4 Sonde mécanique vs. M.E.B.

Il est utile de rappeler les facilités et améliorations apportées par l'utilisation de la microscopie électronique à contraste de potentiel - sonde virtuelle -, par opposition à l'emploi de la sonde mécanique guidée (encore appelée "testeur à pointes") - sonde matérielle -.

II.2.4.1 Possibilités d'observabilité

L'un des aspects majeurs du T.F.E. réside dans ses possibilités d'observabilité du comportement interne des circuits. Ce point est d'autant plus important que le degré

---

6. Nous aurons l'occasion de discuter plus amplement ces problèmes de niveau de représentation du circuit, de modèles de fautes et de caractère combinatoire du processus de propagation arrière dans la suite de cette deuxième partie.
7. La contrôlabilité consiste à pouvoir imposer des valeurs en certains points du circuit. Elle est externe quand elle se fait à partir des entrées primaires.

† Ces techniques sont qualifiées de non utilisables dans la mesure où elles nécessitent la mise en place de dispositifs spéciaux, à prévoir lors de la conception du circuit sous test.

type de contrôl.	compatible observab.	phén. volatil	phén. réversible	cause de dommages	efficacité d'emploi
charge d'une équipot.	OUI	OUI	OUI	NON	peu exploitable (courant faible)
E.B.I.C.	NON	OUI	OUI	NON	théorique (tension d'accél. trop élevée)
charge de grille flot.	NON	NON	par irradiation	OUI	non utilisable†
modif. param. transistors	NON	NON	par irradiation	OUI	non utilisable†
transistors oxyde épais	NON	NON	par bombard. électr.	OUI	non utilisable†

Figure II.3. Evaluation de méthodes de contrôlabilité interne par faisceau d'électrons

d'intégration des circuits VLSI est en constante augmentation.

Le T.F.E. permet donc une observabilité beaucoup plus large du circuit sous test, grâce au balayage de l'ensemble du circuit par le faisceau d'électrons, alors que les instruments d'observation plus classiques ne permettent d'accéder qu'à un nombre restreint de points du circuits.

Enfin, en ce qui concerne le maniement de l'outil employé, l'utilisation de la sonde mécanique guidée, justement par son caractère mécanique non aisément contrôlable, entraîne souvent des problèmes de positionnement sur l'équipotentielle à observer, alors que la facilité de contrôle du faisceau d'électrons implique plus de rapidité et surtout plus de précision de l'observation, ce qui en augmente les possibilités.

II.2.4.2 Effets secondaires

L'utilisation d'une sonde mécanique peut endommager le circuit, étant donné qu'elle est en contact mécanique avec le dispositif observé.
D'autre part, l'emploi de "testeurs à pointes" peut introduire un effet capacitif important qui risque d'entraîner des perturbations du fonctionnement électrique du circuit [BCL82]. Ce phénomène provient en réalité non pas de la sonde elle-même, mais de ses dimensions relativement à celles du circuit à observer : en effet, le degré d'intégration est tel que les connexions à tester sont très proches les unes des autres, et les mesures du potentiel sur une connexion quelconque se trouvent perturbées par cette proximité.

Ces "effets secondaires", qui peuvent donc réellement endommager le circuit sous observation, ne peuvent survenir en aucun cas si l'on utilise le T.F.E., en raison de sa nature même (faisceau d'électrons, donc sonde "virtuelle"), et dans les conditions d'observation,

pour lesquelles une tension d'accélération relativement peu élevée suffit.

II.2.4.3 Performances

Les performances auxquelles nous nous intéressons ici sont relatives au mesures que l'on peut obtenir.
En effet, les mesures obtenues au moyen de la sonde mécanique sont peu précises pour des circuits VLSI, voire parfois carrément fausses, encore à cause des capacités parasites introduites par la proximité des connexions, alors que l'utilisation du T.F.E. en contraste de potentiel permet d'obtenir une échelle fine des mesures exactes du potentiel sur les connexions, par interprétation des niveaux de gris sur l'image du circuit obtenue, puisque les zones portées à un potentiel positif apparaissent d'autant plus sombres que le potentiel est élevé [BCL82].

II.2.5 Problèmes non résolus par le M.E.B.

Malgré les puissantes possibilités d'observabilité qu'elle peut offrir, l'observation de circuits intégrés par faisceau d'électrons a également ses limites. Celles-ci ont fait l'objet d'analyses, notamment de la part de sociétés qui commercialisent des testeurs par faisceau d'électrons [Lee89], mais aussi d'organismes de recherche [MCM87], pour aboutir à la définition d'un certain nombre de précautions/règles à suivre dans la conception de circuits destinés à un tel test.

Ces limitations, et les règles à respecter pour les dépasser, sont décrites dans la suite.
[Lee89] distingue trois niveaux de limitations, respectivement relatifs à l'observabilité, la précision et l'utilisation pratique d'un testeur par faisceau d'électrons.

II.2.5.1 Règles à suivre pour l'observabilité

Tout le problème dans ce cas vient du fait que l'observation par M.E.B. nécessite l'accès du faisceau d'électrons à la couche de matériau que l'on veut sonder, cet accès ne devant pas rencontrer d'obstacle, ou du moins, si une couche isolante est rencontrée, il faut assurer au conducteur enterré une voie d'accès à la surface du circuit.
Cet objectif peut être atteint de trois façons différentes [Lee89], selon que l'on assure un accès à la couche de surface, que l'on effectue des mesures directes ou indirectes, ou que l'on utilise des points de test.

Le maintien d'un accès à la couche de surface se fait en évitant d'"obscurcir" les points enterrés à sonder par des couches métalliques, ou, dans le cas de technologies pour lesquelles la couche métallique de surface constitue l'alimentation ou la masse, en pratiquant des trous dans cette couche, qui serviront de voies d'accès aux couches enterrées.

Les mesures effectuées sont qualifiées de directes ou indirectes, selon que le circuit est non-passivé ou passivé.
Dans le cas de circuits non-passivés, les mesures sont évidemment plus précises. Si le

circuit est déjà passivé, il est souvent préférable de se contenter de mesures indirectes plutôt que de le dépassiver, car ceci peut détruire le circuit, ou du moins en altérer le comportement. Toutefois, il vaut mieux que les couches de passivation et d'isolant soient les plus minces possibles.

Enfin, l'emploi de points de test métalliques, placés à la surface du circuit, et reliés aux noeuds à sonder, reste encore le meilleur moyen d'obtenir les mesures les plus précises.

II.2.5.2 Règles à suivre pour la précision

Il est surtout nécessaire de prendre ce type de précautions lorsque l'on veut effectuer des mesures en forme d'onde, notamment en mode oscilloscope (mesures quantitatives).

Ces règles ont d'abord pour but d'améliorer le rapport signal/bruit dans les mesures, par action sur la largeur et la topographie des conducteurs à sonder, et doivent d'autant plus être suivies que ces conducteurs sont profondément enterrés.

Le deuxième objectif à atteindre est de réduire les effets de proximité ("crosstalk") des lignes, qui peuvent être source de couplages capacitifs. Pour ce faire, il faut minimiser la distance à la surface (profondeur à laquelle est enterré le conducteur), au moins au niveau des points à sonder, et évidemment augmenter dans la mesure du possible la distance latérale entre conducteurs voisins pour en minimiser les influences réciproques.

Enfin, ces règles visent également à maintenir un environnement cohérent entre la sonde électronique et les points à observer. En effet, pour approcher le point à sonder, le faisceau doit traverser tous les champs électriques locaux créés par la puce à tester, qui viendront donc l'influencer. En outre, les électrons secondaires seront encore plus perturbés par ces mêmes champs puisque leur énergie est inférieure à celle des électrons du faisceau incident. Le champ global résultant est le plus fort lorsque plusieurs zones de la puce vont changer d'état de façon synchronisée. Pour éviter ce problème, il est recommandé de bien répartir ces blocs ou ces interconnections dont les transitions se font à l'unisson. Cette répartition doit donc accroître l'asymétrie du plan de masse [Lee89]. De plus, on peut également "stabiliser" l'effet des champs locaux en plaçant aux environs des points de test des lignes à la masse, qui ont un potentiel stable.

II.2.5.3 Règles à suivre pour l'utilisation pratique

Ces règles visent à "rentabiliser" l'observation par sonde électronique. En effet, celle-ci étant coûteuse, il s'agit d'obtenir les meilleures mesures en un temps le plus court possible. Ce temps dépend de plusieurs facteurs, dont ceux relatifs aux programmes de test utilisés. Ces règles seront détaillées au chapitre suivant, § III.3.1.

II.2.6 Applications possibles

L'utilisation d'un testeur à faisceau d'électrons, ou même simplement d'un microscope électronique à balayage peut se concevoir pour différentes applications. Ces applications, citées par [LaC83], sont :

- La caractérisation de paramètres physiques de la technologie utilisée pour réaliser un circuit
- L'inspection et le contrôle de qualité de circuits
- La recherche de fonctionnements limites des circuits
- La reconstitution du schéma électrique ou logique d'un circuit à partir de son dessin des masques ("reverse engineering")
- L'analyse de défaillances au cours de la vie du circuit
- La mise au point de circuits prototypes

Toutes ces applications sont explicitées dans [LaC83] et [Lau84]. Nous donnerons brièvement le principe de l'analyse de défaillances, quant à la mise au point de prototypes, elle fait l'objet du restant des chapitres de ce mémoire de thèse.

Dans le cas de l'analyse de défaillances, le principal problème provient du fait que le circuit observé est à structure non connue. En effet, il est simplement pré-supposé que, d'une part, un certain nombre de vecteurs de test manifestant la défaillance est connu (il s'agit d'un test fonctionnel), et que, d'autre part, l'on dispose d'un circuit réputé sans défaillance : le circuit de référence. Une telle situation de test se produit donc lorsqu'une défaillance est observée au cours de la vie utile d'un circuit.

La manifestation d'une défaillance se traduit donc par l'observation d'une différence entre une valeur attendue en un point observable du circuit et la réponse réellement obtenue. Parmi tous les points de différence, un seul indique le siège de la défaillance que l'on veut localiser, et d'autres n'en sont que l'une des manifestations correspondant au vecteur de test utilisé.

Le principe de base de la méthode de localisation est d'appliquer successivement un certain nombre de ces vecteurs de test, manifestant tous la même défaillance, sachant que le siège de la défaillance recherché figurera dans l'intersection de tous les ensembles de points de différence ainsi obtenus. Ainsi, en fin de processus, on imagine facilement que cette intersection se restreindra à un unique point, et la localisation sera alors effectuée, si l'on se restreint à l'hypothèse de faute simple, bien entendu. Si par contre l'hypothèse de faute multiple est formulée, on pourra tirer parti de la notion de fréquence d'apparition des différences, pour différents vecteurs de test : un vecteur donné pourra manifester une première défaillance, mais pas un second, tandis qu'un autre vecteur pourra inversement manifester la seconde défaillance, mais pas la première ; on peut donc raisonnablement supposer que les sièges respectifs des défaillances considérées seront localisés parmi les

points de différences apparaissant le plus fréquemment.

Le cas des circuits séquentiels pose des problèmes plus spécifiques, dus aux fait que des erreurs peuvent être mémorisées, induisant d'autres erreurs lors de l'application des vecteurs de test suivants, sans pour autant que ces vecteurs manifestent la défaillance. Pour pouvoir néanmoins localiser la défaillance, seule la première des images exhibant des points de différence devra être utilisée dans le processus diagnostic.
Le lecteur intéressé pourra trouver plus de détails à ce sujet dans [SaC87], [CoC89], et [CCC89].

II.3 DEUXIEME GENERATION : T.F.E. ETENDU

Comme il a été dit en introduction de ce chapitre, l'obtention d'un "T.F.E. étendu"[8] passe par l'adjonction au T.F.E. d'outils et de méthodes pour le contrôle de l'équipement de base, ainsi que pour la gestion et le formattage des données observées. En ce sens, ces outils et méthodes sont conçus et développés en fonction de l'application visée, puisque l'on n'aura pas forcément besoin des mêmes traitements en analyse de défaillances qu'en validation de prototypes, par exemple.
Dans la suite de ce paragraphe, nous nous restreindrons aux T.F.E. étendus en vue de la validation de prototypes de circuits intégrés.

Dans le cadre de la validation de prototypes, donc, les besoins, déjà identifiés par [Lau84], sont les suivants :

- Un système de conditionnement du circuit, qui puisse l'alimenter et lui injecter des vecteurs de test générés soit par le système lui-même, soit par un générateur de vecteurs de test externe
- Un système de saisie et prétraitement d'images observées ou de signaux obtenus
- Un système de traitement des informations, capable de mettre en correspondance les données obtenues avec des données de référence, et de gérer le tout
- Un système d'affichage et de dialogue avec l'utilisateur.

8. Cette terminologie montre, ne serait-ce que par son imprécision, à quel point cette notion est mal définie. Nous pourrons voir, dans la suite de ce paragraphe, que les extensions - en fait, les interfaçages avec l'environnement extérieur du T.F.E - peuvent être très variables suivant les réalisations qui ont été effectuées, et que la limite séparant un T.F.E. de base d'un T.F.E. étendu reste très floue.

II.3.1 Système de conditionnement

Un système de conditionnement doit avant toutes choses permettre le lien entre un générateur de vecteurs de test et un analyseur logique d'une part, et les broches du circuit sous test d'autre part. Ce lien va permettre l'alimentation du circuit, l'injection de vecteurs de test sur ses broches d'entrée, le recueil des résultats sur ses broches de sortie, et l'injection de valeurs sur les horloges externes.
Mais le système de conditionnement peut être plus qu'une simple interface : il peut être lui-même générateur de vecteurs de test, séquenceur, et/ou analyseur logique. Ou encore, il peut constituer un bon compromis entre ces deux extrêmes : interface banalisée, permettant l'utilisation de générateurs de vecteurs de test et d'analyseurs logiques quelconques, dotée d'une mémoire en lecture et d'une mémoire en écriture, et capable de séquencer la session de test suivant différents modes choisis par l'utilisateur.

Le système de conditionnement TESSIE [Bai84], [Bau85], par exemple, réalisé au laboratoire, présente de telles caractéristiques.
TESSIE est un outil de conditionnement de circuits, destinés à l'analyse logique, qu'elle soit externe (observation restreinte aux sorties primaires) ou interne (observation étendue à des points internes du circuit). TESSIE permet deux modes de fonctionnement : mode édition de vecteurs et mode test.

Le mode édition permet de charger des vecteurs de test préalablement générés dans l'une des mémoires internes (appelée RAMTEST) de TESSIE, et d'effectuer sur cette mémoire toutes les opérations d'édition désirées, à partir d'une console ou d'un micro-ordinateur, à l'aide de commandes spécifiquement définies.
Cette mémoire, - RAMTEST - sera exploitée par le mode test pour le conditionnement proprement dit du circuit. Le mode test offre quatre types de contrôle qui sont assurés par un microprocesseur interne à TESSIE [Bai84] :

- Fonctionnement en mode pas à pas
- Fonctionnement pour n cycles
- Fonctionnement jusqu'à un point d'arrêt
- Fonctionnement en continu

RAMTEST est donc une mémoire utilisée en lecture par le circuit. Pour permettre une analyse logique ultérieure, TESSIE possède une deuxième mémoire interne, nommée RAMANA, accessible en écriture au circuit, et dans laquelle seront stockées les réponses du circuit (mode test), destinées à être formattées et transférées (mode édition) vers un micro-ordinateur externe aux fins d'analyse.

TESSIE est un outil adapté au conditionnement de circuits à brochages différents, puisque ce brochage est banalisé par l'emploi d'une matrice réalisant l'allocation des

différentes lignes, en entrée en sortie, ou en entrée/sortie.
TESSIE limite actuellement le nombre de broches du circuit à 256 (4 modules de 64 broches). La longueur des séquences de test est restreinte à 2K mots de 32 bits (taille de chaque mémoire). Enfin, des travaux récents [BCD88] ont été effectués pour permettre l'utilisation distante de TESSIE, en étant simplement connecté à travers un réseau au micro-ordinateur utilisé.[9]
De plus, le développement, en cours [Zhu89], d'un traducteur du format EDIF ("Electronic Design Interchange Format") vers le format simplifié de données utilisé par TESSIE permettra une plus grande souplesse d'utilisation, et une meilleure standardisation de ce système de conditionnement.

II.3.2 Système de saisie et prétraitement de données

Le but de ce système est l'acquisition des données - qu'il s'agisse d'images ou de forme d'ondes - à partir de l'observation au microscope, et leur prétraitement (formattage, élimination de bruit et artéfacts, etc.), de façon à ce que ces données soient aisément accessibles aux programmes de traitement de l'information agissant en aval de ce système. La saisie et le prétraitement diffèrent évidemment selon le mode d'observation.

II.3.2.1 Saisie et prétraitement en mode image

La saisie a pour but de transformer le signal vidéo issu du microscope électronique en une image digitalisée. Rappelons que la finalité de ce processus est de reconnaître, à partir d'une image en niveaux de gris, le potentiel des connexions métalliques visualisées sur cette image.
Le prétraitement permet d'améliorer la qualité de l'image, de la binariser et d'en éliminer les artéfacts.
L'amélioration de la qualité de l'image peut déjà se faire avant même le prétraitement, par le procédé d'"intégration d'images à la saisie", consistant en une moyenne arithmétique de n images de la même zone d'observation. D'autres méthodes peuvent être utilisées à la suite de cette étape : elles consistent à effectuer une transformation de l'image dans le domaine spatial, soit par lissage (moyenne arithmétique des proches voisins de chaque pixel) ou par application d'un filtre médian.
Pour binariser l'image, on utilise des méthodes de seuillage, et on peut principalement effectuer une segmentation, soit par extraction de contours des motifs représentés, soit par segmentation en région homogène [Wes78].

---

9. En l'occurrence, il s'agit d'une station de travail IBM 6150, sous système d'exploitation UNIX.

Un exemple de système de saisie et prétraitement d'images est donné par ROMUALD [RJL83], utilisé au laboratoire, et réalisé dans le cadre d'une collaboration CICG-INSERM. D'un point de vue fonctionnel, ROMUALD peut se décomposer en quatre unités ayant en charge [Ber85] :

- La saisie de l'image (digitalisation d'un signal type vidéo) et son stockage en mémoire
- Son prétraitement (application d'algorithmes de lissage, seuillage, moyennage, recherche du gradient pour déterminer sa position, etc.)
- L'affichage de l'image traitée sur un écran de visualisation (reconversion en signal vidéo)
- Le contrôle et l'ordonnancement de ces tâches, ainsi que le dialogue à travers un terminal avec l'utilisateur ou un micro-ordinateur hôte.

D'un point de vue architectural, ROMUALD est organisé autour d'un mégabus, auquel sont reliées l'unité d'entrées/sorties, munie de convertisseurs analogique-digital et digital-analogique, l'unité de commande globale, avec sa mémoire interne, et l'unité de traitement. Celle-ci se décompose en huit processeurs locaux, effectuant les mêmes traitements en parallèle, sur chacune des huit zones de l'image ainsi partitionnée.
Une image numérisée par ROMUALD est représentée comme une matrice de 512 lignes, chaque ligne étant composée de 512 pixels, dont le niveau de gris est codé sur 8 bits. Une image saisie correspond à un zoom sur une zone particulière du circuit à observer, avec un grossissement suffisant pour que cette zone soit analysable dans de bonnes conditions [Gui85].

II.3.2.2 Saisie et prétraitement en mode forme d'onde

Ce mode d'observation utilise la mesure d'un courant, fonction de la variation de la quantité d'électrons secondaires détectés, dépendant elle-même de la variation du potentiel au point d'observation sur lequel est positionné le faisceau [UrF89]. La saisie consiste donc à effectuer la transformation de cette variation de courant en une variation de potentiel, de façon à obtenir la courbe - forme d'onde - retraçant cette variation [FFW81].
Des problèmes se posent pour l'extraction des valeurs logiques de potentiel à partir de telles courbes. Il faut en effet éliminer le bruit dont le signal est entaché, et on utilise pour cela des méthodes d'intégration et de moyennage du signal détecté. Un système présentant de telles caractéristiques est décrit dans [MRP87].

II.3.3 Système de traitement des informations

C'est au niveau du système de traitement des informations que va se réaliser le lien entre le T.F.E. et les outils et bases de données de C.A.O.
En effet, toujours dans le cadre de la validation de prototypes de circuits intégrés, les données observées, ayant préalablement été saisies et prétraitées, doivent subir des

traitements plus élaborés en vue d'être directement exploitables pour le diagnostic.

Ces traitements ont pour finalité de déterminer si les points observés sur le circuit sont en erreur ou non, c'est-à-dire si l'on a mesuré en ces points une valeur logique de potentiel différente de la valeur attendue, qui représente la référence de bon fonctionnement du circuit sous test.
Pour ce faire, il faut être en mesure de mettre en correspondance les zones observées avec leur description en termes de dessin des masques du circuit, de déduire les valeurs logiques à partir des niveaux de gris de l'image ou des valeurs de potentiel de la forme d'onde, et d'effectuer la comparaison proprement dite entre valeur logique observée et valeur logique de référence.
Le traitement des informations, tout comme la saisie et le prétraitement des données, va dépendre du mode d'observation utilisé.

II.3.3.1 Traitement des informations en mode image

Le processus de traitement des informations en mode image sera illustré par la description du travail réalisé au laboratoire dans cet objectif [Gui85], [GML86], [GMC87].
La correspondance et la comparaison entre l'image observée et la description du circuit en termes de dessin des masques[10] est réalisée à travers les étapes suivantes [GML86] :

- *Calcul des coordonnées des points d'observation dans la description des masques :* à partir du fichier CALMA, les données décrivant les masques sont organisées en plusieurs fichiers, et les coordonnées de chaque point à observer sont déduites de la hiérarchie de description des cellules CALMA. Des programmes d'affichage ont également été développés, qui permettent de désigner directement sur l'écran, à l'aide d'un réticule, le point que l'on veut observer, et dont les coordonnées seront déduites. Ces coordonnées sont appelées "coordonnées théoriques" des points.

- *Calcul des adresses des points d'observation dans les images M.E.B. :* cette étape a pour but de réaliser la correspondance entre des points d'observation sur l'image M.E.B. et les mêmes points dans la description du circuit en termes de dessin des masques. Il s'agira donc de déterminer les "coordonnées pratiques" de ces points, correspondant à leurs "coordonnées théoriques".
  Une première méthode, présentée dans [Gui85], utilise trois "points de référence", dont les coordonnées pratiques et théoriques sont données dans une phase d'initialisation. Ces trois points, qui représentent respectivement le point origine, un déplacement horizontal Δx et un déplacement vertical Δy par rapport à cette origine, sont censés

10. Dans cette réalisation, le format de description de masques adopté est le "stream format" de CALMA.

permettre d'effectuer l'association coordonnées théoriques/coordonnées pratiques pour n'importe quel point, à l'aide d'une fonction de correspondance décrite dans [Gui85]. Mais, après expérimentations, cette méthode s'est avérée beaucoup trop imprécise dans la localisation, notamment à cause des distorsions constatées. La solution, présentée dans [GML86], consiste à compenser ces distorsions, et à considérer les contours des polygônes étudiés. La compensation des distorsions se fait par une superposition exacte de l'image M.E.B. et d'une "image théorique" générée à partir de la description des masques. De plus, les points de référence sont remplacés par des fenêtres de corrélation. Cette méthode de superposition et d'extraction de contours est détaillée dans [GML86] et [GMC87].

- *Détermination du niveau logique d'un point d'observation en fonction de son niveau de gris :* le niveau de gris du point est codé, rappelons-le, dans la mémoire d'images sur 8 bits, ce qui autorise 256 niveaux de gris. La détermination du niveau logique correspondant nécessite donc la comparaison de ce niveau de gris à un ou deux seuils [GML86]. Si le niveau de gris est supérieur au seuil le plus élevé, le point a un niveau logique bas, s'il est inférieur au seuil le plus bas, le niveau logique du point est haut. Entre ces deux seuils, on considère que le point est dans un état intermédiaire.

- *Comparaison entre niveau logique observé et niveau logique de référence, obtenu par simulation :* cette étape n'est pas encore réalisée. Elle ne posera de toutes façons pas de problèmes majeurs, dès lors que l'on sera en mesure d'effectuer la correspondance entre une description en termes de dessins des masques et une description d'un niveau d'abstraction plus élevé.[11]

II.3.3.2 Traitement des informations en mode forme d'onde

Ce processus sera illustré par la description du travail réalisé par [BoH87] dans le cadre du projet ADVICE (cf. chapitre suivant). Ce travail concerne la réalisation d'un comparateur de formes d'onde, qui effectue les tâches suivantes :

- Traduction des données échantillonnées (forme d'onde) en une description logique de format adapté à la comparaison

- Etant donné le nom du point considéré, extraction des données de référence obtenues par simulation à partir des fichiers créés par le simulateur,[12] et traduction de ces données dans le format d'entrée du comparateur

---

11. Il existe des outils sur le marché qui le permettent [BGV88].

12. Le simulateur choisi est celui du système HILO3 [Gen85].

- Comparaison proprement dite.

En sortie, le comparateur est en mesure de fournir :

- Un fichier contenant le résultat de la comparaison
- Une valeur booléenne indiquant s'il y a différence ou non au point considéré, i.e. indiquant si ce point est en erreur ou non
- S'il y a eu des différences, les instants auxquels elles ont été constatées.

Plus de détails sur les fonctions effectuées par le comparateur peuvent être trouvés dans [BoH87].

II.3.4 Quelques T.F.E. étendus existants

Le niveau "T.F.E. étendu" constitue, à l'heure actuelle, le seuil maximal de développement des systèmes de test par faisceau d'électrons existants dans le monde industriel, qu'ils soient déjà commercialisés ou en développement. Dans la suite de ce paragraphe, nous essaierons de présenter ces systèmes.

Les T.F.E. étendus que nous avons pu recenser[13] sont les suivants :

- "Integrated Diagnostic System" (IDS 5000), développé par la société SENTRY-SCHLUMBERGER, FRANCE-USA
- "Integrated Electron-beam Measurement System" (IEMS), développé par la société SIEMENS, RFA
- "CAD-linked Electron-beam Tester System", développé par la société TOSHIBA, JAPON
- "Automated Electron-beam Tester with CAD Interface" (FINDER), développé par les laboratoires du centre national de télécommunications japonais (NTT), JAPON
- "CAD System Interface for a stand-alone Electron-beam Tester" (CAD I & CAD II), développé par la société LINTECH, GB

La figure II.4 présente les caractéristiques principales des T.F.E. étendus mentionnés plus haut. Cette synthèse n'a été réalisée qu'à titre indicatif. Plus de précisions peuvent être trouvées dans les articles référencés, encore que, [CoL87] mis à part, ces articles sont assez pauvres de détails sur les méthodologies utilisées, et révèlent souvent des fins publicitaires.

---

13. D'autres existent peut-être. Nous n'en avons pas trouvé trace dans la littérature.

Parmi les systèmes décrits, ceux de SENTRY-SCHLUMBERGER, NTT et LINTECH paraissent les plus achevés, bien que le système NTT ne soit pas - pas encore ? - commercialisé, ce qui explique sans doute que l'adaptation à des formats de CAO standards n'aie pas semblé faire l'objet d'une préoccupation majeure de ses concepteurs, de même d'ailleurs que l'interfaçage avec l'utilisateur.

II.4 TROISIEME GENERATION : SYSTEME INTEGRE DE TEST

Il a été dit au paragraphe précédent que les T.F.E. étendus de SENTRY-SCHLUMBERGER, NTT et LINTECH sont les plus achevés, et par conséquent les plus aptes à "endosser" une couche supplémentaire, formée d'outils et méthodes de haut niveau pour l'automatisation du diagnostic, c'est-à-dire de la localisation de fautes. Il est raisonnable de penser - en l'absence d'information publique supplémentaire - que l'on "y travaille", ou du moins que l'on y réfléchit sérieusement chez SENTRY-SCHLUMBERGER comme chez LINTECH.

Les caractéristiques à ajouter à ces T.F.E. étendus concernent en priorité la comparaison entre les valeurs observées et les valeurs attendues. En effet, il est nécessaire que cette étape soit automatisée, de façon à obtenir une liste des lieux où des erreurs ont été exhibées.
Ensuite, et ceci concerne surtout le T.F.E. étendu proposé par SENTRY-SCHLUMBERGER, il faut pouvoir disposer de la description logique du circuit sous test, d'une part pour augmenter la simplicité d'utilisation du dispositif, et d'autre part en vue d'une automatisation du processus de localisation de fautes : ce processus, très coûteux dans le cas de circuits VLSIs, ne serait même pas envisageable s'il devait se limiter à un raisonnement au seul niveau du dessin des masques.
Enfin, l'automatisation de la localisation de fautes reste entièrement à concevoir et à réaliser, pour les systèmes SENTRY-SCHLUMBERGER et LINTECH. Par contre, pour ce qui concerne l'état d'avancement du système NTT, [YaO87] présente déjà une méthode de diagnostic assisté, à l'aide d'un dictionnaire de fautes. Cette méthode, qui vise à organiser les vecteurs de test par le taux de couverture de fautes qu'ils assurent, et donc à appliquer en premier au circuit sous test ceux qui permettent de détecter le plus grand nombre de fautes, permet effectivement de réduire le nombre d'observations et de mesures, mais laisse quand même l'entière responsabilité de la localisation à l'utilisateur.

Nous aurons l'occasion de revenir, dans les chapitres III et V, sur l'utilisation de dictionnaires de fautes. Nous nous contenterons de dire ici qu'il existe deux approches du diagnostic : l'une, basée sur un dictionnaire de fautes pour un diagnostic assisté, qui sera illustrée à travers la présentation du système ADVICE au chapitre suivant, et l'autre, utilisant une approche à base de connaissances pour un diagnostic automatique, qui sera présentée à travers le système PESTICIDE au chapitre IV. Ces deux systèmes, avec la méthode NTT, sont les seules études connues à ce jour pour dépasser le seuil du T.F.E. étendu dans le cadre de la validation de prototypes de circuits intégrés.

Caract.	SENTRY	SIEMENS	TOSHIBA	NTT	LINTECH
Contrôle du T.F.E	auto-matique	auto-matique	auto-matique	auto-matique	auto-matique
Position-nement fin	"DC motors"	auto-matique (points de ref.)	auto-matique (points de ref.)	auto-matique (superp. d'images)	auto-matique (points de ref.)
Modes de mesure	f. d'onde	f. d'onde	f. d'onde	f. d'onde + image	f. d'onde + image
Formats layout acceptés	APPLICON CALMA CIF ···	non précisé	non précisé	propre NTT + CALMA	APPLICON CALMA CIF ···
Formats netlist acceptés	SPICE TEGAS EDIF ···	non précisé	non précisé	propre NTT	SPICE TEGAS EDIF HILO3
Corresp. layout-netlist	auto-matique	manuelle	auto-matique	manuelle	auto-matique
Compar.	manuelle	manuelle	manuelle	manuelle	manuelle
Descrip. logique	non	oui	non	oui	prévue
Interface utilisateur	ergono-mique	rudimen-taire	rudimen-taire	rudimen-taire	ergono-mique
Etat	commercial.	proto.	proto.	proto.	commercial.
Ref.	[CLL86] [CoR87b] [CoL87] [CoR87a] [Lee87] [LLP87] [Ric87] [CRP88]	[GHK87] [GHK88]	[KMS86] [KMS87]	[TaK86] [KuT86]	[HKP87]

Figure II.4. Caractéristiques des T.F.E. étendus recensés

II.5 CONCLUSION

Notre objectif dans ce chapitre était de présenter brièvement le cadre dans lequel nous avons travaillé, et dans lequel se placent les deux systèmes dont le détail fait l'objet des chapitres III et IV, de façon à prendre la mesure des enjeux du test par faisceau d'électrons et de sa nécessité, mais également des difficultés qu'il présente.

En effet, en repoussant les limites d'observabilité des circuits à haut degré d'intégration, le test sans contact - ici, le test par faisceau d'électrons - est reconnu comme seule méthode viable de test interne.
Néanmoins, seule une observation de la couche métallique de surface du dispositif sous test est possible en l'absence de possibilités d'accès aux couches enterrées, à prévoir dès la conception du circuit, et nous avons pu voir d'autre part qu'assurer la contrôlabilité du circuit à l'aide du même outil demande également des dispositifs supplémentaires, et reste par conséquent difficile à mettre en oeuvre.

Mettre l'outil de test par faisceau d'électrons à la disposition d'un public formé de spécialistes de conception et de test de circuits intégrés VLSI a nécessité des efforts de recherche et de développement pour en faire ce que nous avons dénommé un "T.F.E. étendu".
Bien qu'importants, ces efforts ne résultent encore actuellement qu'en un interfaçage - certes très convivial - du T.F.E. avec l'utilisateur : on remarquera notamment que la comparaison entre les valeurs de potentiel mesurées et les résultats de simulation est manuelle pour tous les systèmes présentés en figure II.4, qu'ils soient commercialisés ou à l'état de prototype. Quant au diagnostic lui-même (la localisation de fautes), son automatisation n'est même pas évoquée, et nous paraît de toutes façons difficile à envisager lorsque le seul mode de mesure choisi est le mode forme d'ondes (SENTRY-SCHLUMBERGER, SIEMENS, TOSHIBA), et lorsque la prise en compte d'une description au niveau logique du circuit n'est pas prévue (SENTRY-SCHLUMBERGER, TOSHIBA) : effectuer une recherche manuelle - pratiquement "en aveugle" dans ces conditions - à travers un circuit intégré VLSI représenté au seul niveau électrique sans tenir compte de sa hiérarchie de conception est peu réaliste.

C'est pourquoi, dans le cadre du projet d'analyse de circuits intégrés par microscopie électronique mené au laboratoire TIM3 [Lau84], la nécessité de développer non seulement des extensions entièrement automatiques au T.F.E., mais aussi des méthodes de diagnostic également automatisées, est apparue très tôt dans la définition du projet, aussi bien pour l'analyse de défaillances que pour la mise au point de prototypes de circuits.
Les chapitres suivants sont consacrés à l'automatisation du diagnostic pour la mise au point de prototypes.

III. ADVICE : APPROCHE DICTIONNAIRE DE FAUTES

III.1 INTRODUCTION

Le projet ADVICE (acronyme de "Automatic Design Validation of Integrated Circuits using Electron-beam") avait pour objectif le développement d'un système de test par faisceau d'électrons, complet, intégré, et automatique.
Ce projet [CoM87] se situe dans le cadre du programme ESPRIT de la CCE, et réunit trois partenaires industriels qui sont :

- BTRL ("British Telecom Research Laboratories"), pour le Royaume-Uni
- CNET (Centre National d'Etudes des Télécommunications), pour la France
- CSELT ("Centro Studi E Laboratori Telecommunicazioni"), pour l'Italie

et deux partenaires universitaires, qui sont :

- TIM3/IMAG (Laboratoire TIM3 - Groupe d'Architecture des Ordinateurs - Institut d'Informatique et de Mathématiques Appliquées de Grenoble), pour la France
- TCDU ("Trinity College DUblin"), pour l'Irlande.

Le projet ADVICE, d'une durée de cinq années, a débuté en Novembre 1984, et a bénéficié de moyens humains évalués à 50 hommes/années.
Le projet ADVICE se proposait de résoudre les problèmes suivants, liés à l'utilisation d'un microscope électronique à balayage pour le test de circuits intégrés :

- Positionnement du faisceau et d'acquisition des mesures
- Techniques de localisation d'erreurs de conception du circuit sous test
- Stratégie de test, de façon à minimiser le nombre de vecteurs de test à générer, ainsi que le nombre de mesures à effectuer

En ce sens, le projet ADVICE intervient à tous les niveaux du système en couches présenté au chapitre II.
Le projet s'est subdivisé en deux périodes distinctes. La première (Novembre 1984 à Novembre 1986) a été consacrée à la définition de l'équipement nécessaire à la réalisation du testeur, ainsi qu'au développement du logiciel de base pour automatiser les procédures d'acquisition de mesures. Elle a permis de réaliser un testeur à faisceau d'électrons. La deuxième période (Novembre 1986 à Novembre 1989) a été dévolue à l'intégration du testeur à un environnement de C.A.O., ainsi qu'au développement de l'interface utilisateur

et à la mise au point de stratégies de localisation de fautes. Cette période visait à obtenir non seulement un "T.F.E. étendu", mais un système intégré de test.

La suite de ce chapitre donne une description du système dans sa globalité, néanmoins, nous nous consacrerons surtout à présenter la couche "méthode", c'est-à-dire les stratégies de localisation de fautes, puisqu'elles nous intéressent plus particulièrement.

III.2 DESCRIPTION GENERALE DU SYSTEME

III.2.1 Généralités

Le système ADVICE, développé sur machine DEC VaxstationII/GPX, offre à l'utilisateur un environnement graphique interactif pour la validation de prototypes de circuits observés à l'aide d'un microscope électronique à balayage utilisé en contraste de potentiel.
Vu par l'utilisateur, ce système est organisé comme un environnement multi-fenêtres, géré par un ensemble de menus. Il a pour principale caractéristique un interfaçage convivial avec l'utilisateur (concepteur du circuit prototype à valider) lui permettant l'accès aux outils d'observation et aux informations en provenance des bases de données de C.A.O.
L'écran de la station de travail utilisée se divise en plusieurs fenêtres, correspondant à divers modules du système. Ces modules peuvent être :

- Un module de dialogue
- Un module de contrôle du microscope
- Un module d'affichage du dessin des masques du circuit sous test
- Un module de contrôle de l'équipement automatique de test (conditionneur de circuit, analyseur logique)
- Un module de contrôle du processeur de traitement d'images
- Un module de contrôle de l'affichage des formes d'ondes logiques et analogiques
- Un module de diagnostic

Un processus du système d'exploitation hôte (VMS) est associé à chaque module, donc à chaque fenêtre. Le contrôle de ces processus, qui communiquent entre eux à l'aide de messages (mécanisme de boîte à lettres), est assuré par un processus superviseur, nommé "ADVICE SHELL". Lorsque l'un des processus a besoin d'un service, il adresse sa requête au superviseur sous forme de "macro-message". Le shell génère alors les "micro-messages" correspondants, puis les transmet aux processus concernés.
Un mécanisme de sauvegarde/restauration de sessions de diagnostic permet d'effectuer en plusieurs temps l'analyse de circuits très complexes.

III.2.2 Utilisation des données en provenance d'outils de C.A.O.

Ces données sont nécessaires, puisqu'il s'agit de validation de prototypes, à la comparaison avec les données observées et les mesures qui en sont issues.

Les choix faits pour le système ADVICE concernent, pour la description au niveau logique du circuit, le système HILO3 [Gen85], et pour la description en termes de dessin des masques, des formats tels que CIF ou GDSII. Le système HILO3 a été choisi car il offre un environnement de simulation complet, permettant la description logique du circuit, la simulation ("fault-free simulation"), la simulation de fautes, la génération de vecteurs de test, et la production d'un dictionnaire de fautes. Les données de référence, exprimées et/ou obtenues à l'aide du système HILO3, seront comparées aux données observées, dans le but de fournir un diagnostic sur le circuit sous test.

L'utilisation d'informations en provenance d'outils de C.A.O. va permettre d'automatiser le positionnement du faisceau et l'obtention de mesures, et ceci par la mise en place, après initialisation du M.E.B., d'une procedure comprenant les trois étapes suivantes :

- Identification des points à observer sur le dessin des masques du circuit
- Positionnement "grossier" par déplacement de la platine du microscope
- Vérification de l'exactitude du positionnement et ajustement, si nécessaire, par un positionnement plus "fin", en utilisant des techniques de traitement d'images.

Les deux premières étapes de ce processus utilisent la donnée des coordonnées des points à observer sur le dessin des masques, et les transforment en signaux de commandes de déplacement de la platine du microscope. La troisième étape utilise la mise en correspondance de l'image observée du circuit et de son dessin des masques [GML86], [StT87].
Une fois la mesure effectuée, on obtient les formes d'ondes logiques et/ou électriques qui seront comparées aux résultats de la simulation du fonctionnement correct du circuit.

III.2.3 Description du système en termes de flux d'opérations

Une session complète de validation de prototype à l'aide du système ADVICE est présentée, de façon synoptique, en figure III.1 [MBG88]. Cette figure décrit la session en termes de séquence d'opérations à effectuer, et se comprend comme suit :

Les deux premières étapes (génération de séquences de test et simulation logique, traduction et chargement des résultats dans la structure de données interne) sont des pré-processus, effectués en dehors d'une session.

Les séquences de test sont ensuite transmises au conditionneur de circuits pour effectuer un premier test, sans utiliser le microscope. Ce test se fait au niveau des entrées/sorties primaires du circuit, et le testeur fonctionne alors en mode analyseur logique.

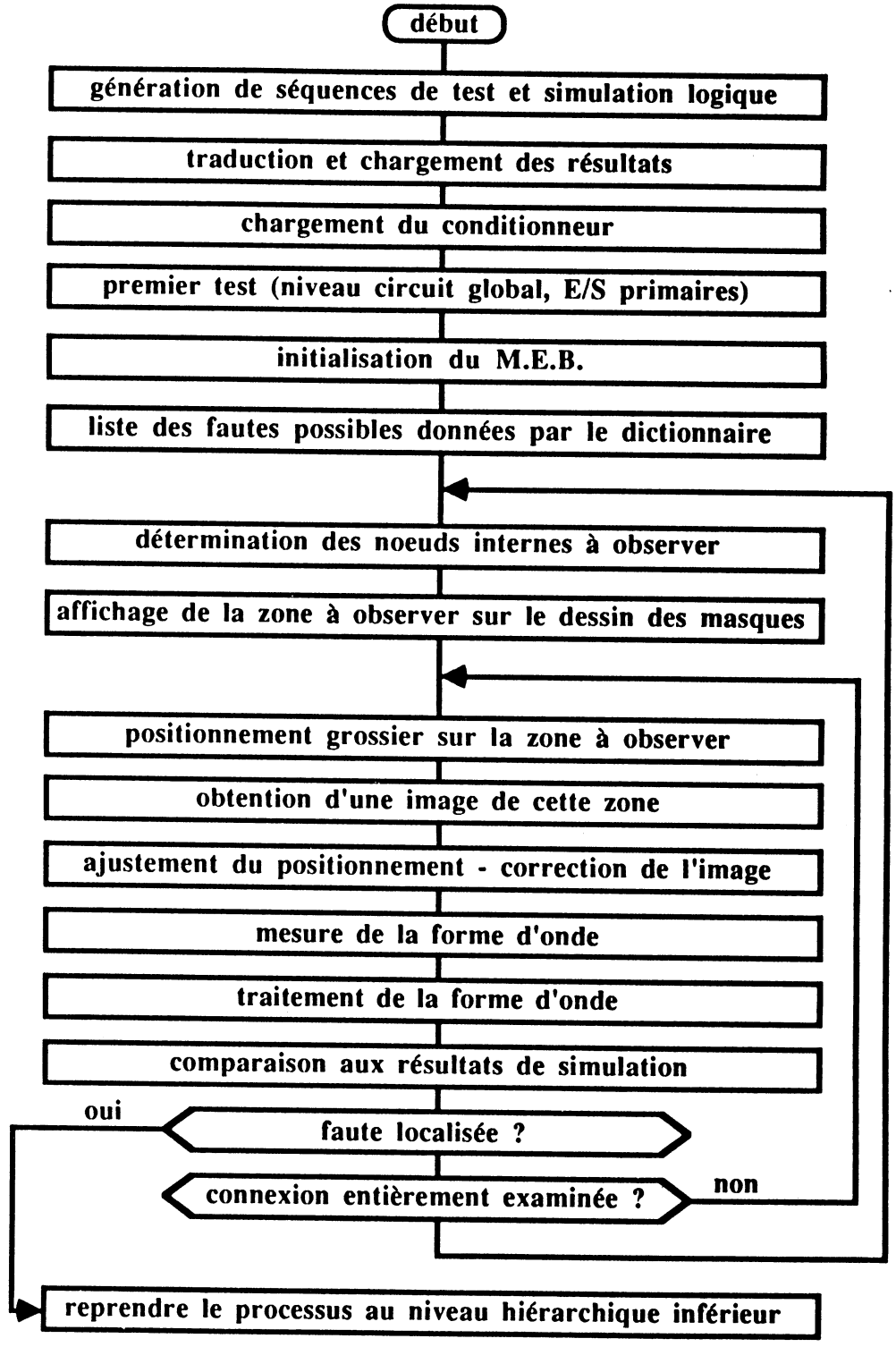


Figure III.1. Séquence d'opérations lors d'une session ADVICE

Puis le microscope est initialisé, et le dictionnaire de fautes est utilisé pour une première tentative de localisation de fautes, censée déterminer les noeuds à observer à l'intérieur du circuit.
Les phases d'observation et d'acquisition de mesures peuvent alors être entamées, fournissant des formes d'ondes qui sont traitées (élimination de parasites, etc.) et comparées aux réponses attendues, conformément aux résultats de simulation.

Si une faute est localisée, alors on essaie de raffiner le diagnostic, en reprenant les mêmes étapes depuis le début du processus, à un niveau hiérarchique inférieur.
Si aucune faute n'est encore localisée, on essaie d'observer d'autres noeuds internes au circuit, de même niveau hiérarchique.

Le graphe de la figure III.1 ne représenterait qu'un "T.F.E. étendu", si l'opération de détermination des noeuds à observer était manuelle.[14] L'originalité du système ADVICE réside justement dans les recherches conduisant à une automatisation - qui peut être partielle ou totale, comme nous le verrons dans la suite - de cette opération, par le développement d'un algorithme de guidage de l'observation ("probing algorithm" [MBG88]). Cet algorithme, ainsi que toutes les informations qu'il utilise, fait l'objet de la suite de ce chapitre.

III.3 STRATEGIES POUR LE TEST PAR FAISCEAU D'ELECTRONS

Nous ne reviendrons pas sur les avantages apportés par le test par faisceau d'électrons à la validation de prototypes de circuits intégrés. Néanmoins, nous avons pu voir au chapitre II que ceci n'allait pas sans imposer certaines contraintes au niveau de la conception des circuits, et que certaines règles de conception adaptée au test par faisceau d'électrons devaient être définies.

En collaboration avec d'autres partenaires dans le projet, nous avons mené une reflexion sur ces règles de conception [MCM87], ainsi que sur les stratégies de simulation à adopter pour ce type de test [MaM87]. Les principaux résultats de cette réflexion sont rapportés dans la suite de ce paragraphe.

III.3.1 Conception en vue du test par faisceau d'électrons

Certaines recommandations pour la conception de circuits destinés à un test par faisceau d'électrons ont été élaborées. Ces recommandations se divisent en trois ensembles de règles, qui sont :

---

14. C'est ce qui se fait dans les systèmes actuellement commercialisés, cités au chapitre II.

- des règles de conception logique
- des règles topographiques
- des règles de sélection de points de test

III.3.1.1 Règles de conception logique

Elles sont destinées à résoudre des problèmes n'apparaissant que pour les circuits complexes (LSI et VLSI), relatifs à la plus ou moins grande difficulté de générer des séquences de test capables de stimuler ces circuits. Ces problèmes sont dûs à la complexité des circuits, mais aussi au fort degré de séquentialité qu'ils présentent, requerrant par là-même de longues séquences de stimuli. De plus, ces stimuli devant être répétés plusieurs fois pour permettre les mesures en mode stroboscopie, les séquences générées doivent être périodiques.

Le type de test pratiqué à l'aide du système ADVICE est un test hors-ligne ("off-line test"), et par conséquent des modes de test spécifiques doivent être utilisés.
Dans chaque séquence périodique à appliquer, on doit distinguer une partie d'initialisation ("homing sequence"), servant à mettre le circuit dans un état connu, et une partie constituant le stimulus proprement dit. Une bonne façon de réduire la longueur de la séquence d'initialisation est d'utiliser des méthodes classiques de D.F.T. [WiP82], qui peuvent se diviser en trois catégories [CMN82] :

- Méthodes de conception augmentant la testabilité (notamment utilisation de structures faciles à partitionner, et insertion de points de test)
- Méthodes de conception permettant d'éliminer la génération de vecteurs de test (utilisation de "universal test sets", de cellules prétestées, etc.)
- Méthodes de conception de circuits autotestables (utilisation de BILBOs ("Built-In Logic Block Observer"), de LFSRs ("Linear Feedback Shift Register" ou de NLFSRs ("Non Linear Feedback Shift Register"), etc.)

Mais ces méthodes, même si elles permettent de réduire les besoins en observabilité et contrôlabilité des circuits, ne résolvent pas tous les problèmes, notamment pour ce qui est du diagnostic, qui nécessite non seulement de détecter les fautes, mais également de les localiser. C'est pourquoi des recherches en contrôlabilité ont été menées [Mic88] dans le cadre du projet (cf. chapitre II).

III.3.1.2 Règles topographiques

Ces règles ont été définies pour s'affranchir des problèmes dûs au microscope qui ont été présentés au chapitre II. Elles reprennent en substance les recommandations déjà citées et, puisqu'elles ne sont pas caractéristiques du projet ADVICE, nous ne les rappellerons pas ici.

III.3.1.3 Règles de sélection de points de test

Ces règles se subdivisent à leur tour en trois catégories :

- Règles liées au diagnostic
- Règles liées à la reconnaissance de formes
- Règles liées à la sonde électronique

Les règles relatives à la première sous-classe dépendent également des stratégies de simulation en vue du test par faisceau d'électrons (cf. § III.3.2).
D'une façon générale, l'ensemble des points de test doit comprendre au moins les ports d'entrée/sortie de chaque cellule, à chaque niveau hiérarchique de la description logique. Cet ensemble doit également contenir les branches de lignes à sortance multiple ("fan-out"), lorsqu'elles existent.

Les règles de la deuxième sous-classe ont pour but principal d'optimiser le temps nécessaire pour retrouver les correspondances entre les coordonnées des points sur l'image du circuit et sur son dessin des masques. Elles visent à éviter autant que possible la répétition de structures régulières dont l'intervalle de séparation est trop petit,[15] et dont la largeur est supérieure à la largeur normale d'une fenêtre d'observation.

Enfin, les règles de la troisième catégorie permettent d'obtenir des formes d'ondes de meilleure qualité. Elles visent par exemple à sonder le plus possible de points métalliques, et à éviter le parasitage d'un signal par un autre qui lui serait trop proche.

Ces règles de "conception adaptée au test par faisceau d'électrons", bien que minimales, peuvent, si elles sont suivies, grandement faciliter et améliorer une session de test. Le "dernier mot" revient néanmoins au concepteur, qui seul peut décider du degré de testabilité de son circuit.

III.3.2 Simulation en vue du test par faisceau d'électrons

Etant donné que l'on doit assurer la corrélation entre le dessin des masques du circuit et sa description logique - notamment en ce qui concerne les noms logiques des différents composants du circuit -, certaines lignes de conduite ont été suggérées pour résoudre les problèmes de la description du circuit et de sa simulation. Elles tiennent compte de la notion de hiérarchie de décomposition.

---

15. Par "trop petit" on entend "inférieur à la moitié de l'erreur causée par le déplacement de la platine du microscope".

La description du circuit se fait, dans la mesure du possible, en tenant compte de la hiérarchie, de façon à mettre en évidence le partitionnement en blocs fonctionnels. Elle est traduite et stockée, avec les autres informations disponibles, en une structure de données hiérarchique interne.

Pour éviter la redondance d'informations, notamment entre les différents niveaux hiérarchiques, lors de la mise en correspondance de la description logique avec la description en termes de dessin des masques, une liste de "synonymes" est construite, dans laquelle chaque ensemble de noeuds (connexions) équivalents au niveau logique pointe sur un seul élément de la liste de synonymes, associé à un élément de la liste des noeuds observables.
Dans la suite, nous reprenons quelques définitions utiles à la compréhension des stratégies de simulation en vue du test par faisceau d'électrons [MBG88].[16]

Définition 1 :
Deux noeuds sont dits "logiquement équivalents" s'ils constituent une connexion "simple" ("fan-out free") entre une cellule et la frontière de la cellule englobante de niveau hiérarchique immédiatement supérieur.□

Si la connexion n'est pas "simple", c'est-à-dire si elle constitue une branche de "fan-out", les deux noeuds ne sont plus équivalents, puisqu'une coupure au niveau logique peut changer le comportement du circuit.
D'autre part, pour chaque ensemble de noeuds logiquement équivalents, est conservée une liste chaînée de pointeurs sur un ensemble de noeuds "équipotentiellement connectés".

Définition 2 :
Deux noeuds sont dits "équipotentiellement connectés" s'ils sont directement liés par une connexion équipotentielle.□

Des procédures de navigation dans la description logique du circuit ("netlist") doivent déterminer la directionalité des ports d'entrée/sortie de chaque cellule : entrée, sortie, ou bidirectionnel. Cette directionalité n'est que potentielle puisqu'elle ne tient pas compte de l'effet de signaux pouvant la contrôler.

Définition 3 :
Le port d'une cellule est dit "potentiellement bidirectionnel" s'il peut être traversé par des

16. Ces définitions ont été traduites de l'anglais, à partir de l'article cité. Elles ne concernent pas des notions nouvelles, mais précisent simplement la terminologie employée dans la suite.

données en provenance ou à destination de cette cellule.□

De plus, pour chaque port de cellule, une liste de "dépendances logiques" est construite.

Définition 4 :
Un port d'entrée et un port de sortie, appartenant à la même cellule, sont dits "logiquement dépendants" s'il existe un chemin logique à l'intérieur de la cellule qui les relie.□

Ces quatre définitions sont illustrées en figure III.2 [MBG88]. Sur cette figure, on peut voir un réseau de cellules décrit hiérarchiquement.
Lorsque l'entrée b pénètre dans la cellule B, elle se transforme en la ligne f : b et f sont logiquement équivalents (définition 1).
dans la cellule B, la connexion f a deux branches de "fan-out", dirigées respectivement vers les portes logiques 1 et 2 : le pied de "fan-out" ("stem") et les deux branches sont équipotentiellement connectés, mais non logiquement équivalents (définition 2).
La connexion l est logiquement dépendante de e, ainsi que de f (définition 4). On dit également que f *contrôle* les lignes l et m.

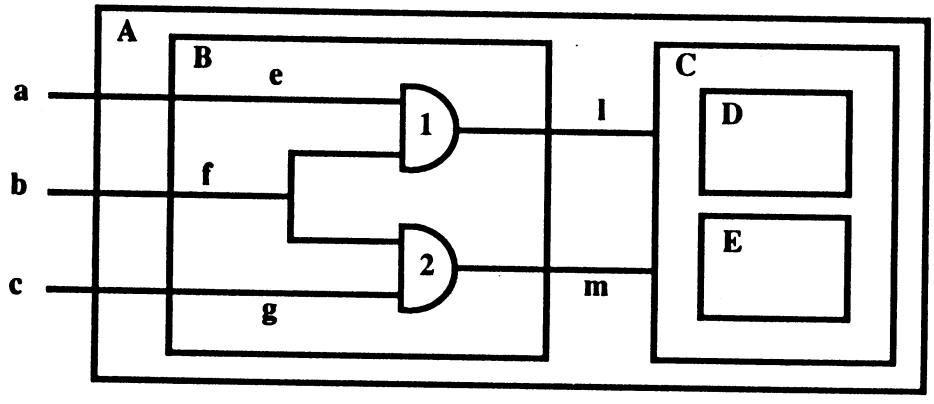


Figure III.2. Illustration des définitions sur un circuit exemple

Tout ceci permet de simuler un nombre restreint de points, et d'éviter ainsi des pertes de temps et de performances à cause des redondances, le but étant de minimiser le coût de la session de test par faisceau d'électrons en restreignant le nombre de points à sonder d'une part, et d'autre part en remplaçant un point non-observable par la sonde électronique par son point d'observation "le plus proche".

Définition 5 :
Un point j est dit point d'observation "le plus proche" d'un point i si j est observable, s'il est topologiquement le plus proche de i, et s'il est **contrôlé** *par i.*
Les points i et j peuvent être situés à des niveaux hiérarchiques différents.□

III.3.3 Génération de stimuli en vue du test par faisceau d'électrons

Dans le cadre du système ADVICE, la connaissance de stimuli est nécessaire à trois objectifs : simulation du fonctionnement correct du circuit, simulation de fautes et conditionnement du circuit sous test.

Notre idée première était de doter le système ADVICE de méthodes de génération automatique de séquences de test. Une étude a donc été menée sur les méthodes existantes, essentiellement la différence booléennne, le D-algorithme, PODEM ("Path-Oriented DEcision Making") et FAN ("FAN-out oriented test generation algorithm"),[17] dans le but d'en adapter les meilleurs aspects aux spécificités du test par faisceau d'électrons dans le système ADVICE.

Une méthode de génération de vecteurs de test ad hoc aurait pu tenir fortement compte de la hiérarchie de décomposition du circuit en sous-circuits moins complexes, et de la définition de relations de dépendance entre les ports de sortie et les ports d'entrée de chaque sous-circuit.
Les grandes lignes d'une telle méthode sont, étant donné un port de sortie de sous-circuit au niveau duquel on veut exhiber une faute :

- Déduire les ports d'entrée du même sous-circuit contrôlant (au sens de la définition 4, § III.3.2) ce port de sortie
- Construire autant de listes contenant le triplet :
  (nom du port de sortie, nom du sous-circuit, nom du port d'entrée)
  qu'il y a de ports d'entrée contrôlant la sortie, puis concaténer ces listes aux listes obtenues aux étapes précédentes
- Itérer ce processus en considérant chacun de ces ports d'entrée comme une sortie de sous-circuit.

A la fin de ce processus, tous les chemins de données liant chaque noeud à observer aux entrées primaires du circuit qui les contrôlent seront identifiés, et seuls les sous-circuits figurant sur ces chemins devront être "mis à plat", après leur avoir appliqué la même méthode en les décomposant à leur tour, jusqu'à atteindre une description en termes de portes logiques, de façon à pouvoir alors mettre en oeuvre des méthodes déjà existantes de génération de vecteurs de test, méthodes qui pourraient d'ailleurs varier en fonction des spécificités de chaque type de sous-circuit.

---

17. Cette étude est présentée dans [Mar87b], mais on trouvera le détail de tous ces algorithmes, accompagné d'exemples, dans [Fuj85].

L'intérêt d'une telle méthode ad hoc, outre son utilisation intensive du partitionnement et de la hiérarchie, aurait été justement l'effet synergique créé par son intégration au reste du système ADVICE : trop de méthodes de génération de vecteurs de test sont appliquées par des ingénieurs de test isolés par rapport aux ingénieurs de conception, et par conséquent obligés de se contenter du schéma logique du circuit à tester comme seule information.
Mais il a été décidé, pour diverses raisons qu'il ne nous appartient pas de présenter ici, de limiter les ambitions du projet - en termes de ressources humaines - en ce qui concerne la génération de stimuli, et nous avons dû interrompre les recherches dans ce sens, en dépit de leur intérêt scientifique actuel, dont témoignent des publications récentes [MuH90], [SMB90].

Le choix fait par les membres du projet a donc été de définir des stratégies de *sélection* de vecteurs de test et non pas de *génération*, en tenant pour acquis que ces vecteurs ont été générés au préalable par l'utilisateur du système ADVICE, par des méthodes de sa convenance.
En effet, générer en temps réel, c'est-à-dire au cours d'une session de travail du système ADVICE, les vecteurs de test nécessaires au bon déroulement du processus de localisation de fautes à l'aide d'un générateur de vecteurs de test commercial aurait augmenté de façon très sensible le temps de la session de travail, sans pour autant que cette tâche soit interactivement liée à la localisation de fautes proprement dite. D'autre part, la cohérence des choix effectués pour le système ADVICE, notamment en ce qui concerne les outils de simulation utilisés, imposait que, s'il devait y avoir sélection d'un générateur de vecteurs de test, ce soit celui du système HILO3 [Gen85], basé sur la méthode du "chemin critique", or celui-ci a été déclaré inutilisable, pour des questions de temps d'exécution, par les membres du projet ayant l'expérience de ce système.

Chaque séquence de vecteurs, générée par l'utilisateur, devra être composée de sous-séquences, chacune normalement composée de :

- Une partie d'initialisation ("reset part")
- Une partie de positionnement du circuit dans un état connu ("homing part")
- Les données de test elles-mêmes

Ces séquences de vecteurs de test sont décrites par un fichier spécial du système, nommé "user history file", car il retrace effectivement l'historique des séquences de test appliquées pour la simulation du fonctionnement correct du circuit, la correspondance entre ces vecteurs et l'ensemble de points qu'ils sensibilisent, ainsi que la correspondance entre les sous-séquences et les instants de début et de fin d'une période de simulation.
Ce fichier contient des informations textuelles, qui seront compilées puis chargées dans la structure de données interne du système. De plus amples détails sur le format de ce fichier et la façon dont il est interprété et utilisé se trouvent dans [GMM88].

III.3.4 Dictionnaire de fautes en vue du test par faisceau d'électrons

La génération d'un dictionnaire de fautes est une étape déterminante pour le système ADVICE, dont l'approche du diagnostic, rappelons-le, est basée sur l'utilisation d'un tel dictionnaire. Cette génération se fait à l'aide du système HILO3, qui offre cette possibilité à l'issue du processus de simulation de fautes.
L'étude de méthodes de génération et de stratégies d'utilisation d'un dictionnaire de fautes dans le cadre du système ADVICE figure dans [Dow87].

La génération du dictionnaire se base sur des *modèles de fautes* préalablement établis, représentant des fautes pouvant survenir à certains endroits du circuit, et se manifestant sur ses sorties primaires. Pour une simulation de fautes au niveau logique, telle que celle pouvant être effectuée à l'aide du système HILO3, ces modèles de fautes sont :

- S0 (collage à 0 ou "stuck-at-0") représentant un court-circuit entre une ligne quelconque et la masse
- S1 (collage à 1 ou "stuck-at-1") représentant un court-circuit entre une ligne quelconque et l'alimentation
- O0 (coupure (0) ou "open (0)") représentant une coupure sur une connexion, qui pérennise sur la ligne une valeur de potentiel égale à 0
- O1 (coupure (1) ou "open (1)") représentant une coupure sur une connexion, qui pérennise sur la ligne une valeur de potentiel égale à 1
- Oz (coupure (z) ou "open (z)") représentant une coupure sur une connexion, qui pérennise sur la ligne un état de haute impédance
- D0 ("drive (0)") représentant une sortie de porte logique conduisant toujours la valeur 0
- D1 ("drive (1)") représentant une sortie de porte logique conduisant toujours la valeur 1
- IN ("inhibit") représentant une faute fonctionnelle qui inhibe l'occurrence d'un évènement dans un sous-circuit modélisé au niveau fonctionnel
- DL ("delay") représentant un circuit ou sous-circuit partiellement ouvert, dont l'effet est de retarder, d'un délai spécifique, la propagation de toutes les valeurs
- S ("short") représentant une forme généralisée de collage dans laquelle deux connexions quelconques peuvent être accidentellement court-circuitées. Le type de court-circuit est alors déterminé en fonction des forces logiques ("logic strengths") des valeurs affectées aux connexions concernées

Tout ou partie de ces modèles peut être utilisé pour la simulation de fautes. Le dictionnaire de fautes résultant sera formé des informations suivantes :

- type de la faute
- nom de la connexion sur laquelle la faute est manifestée
- date (en cycles d'horloge) à laquelle la faute est observée
- valeur correcte de la connexion
- valeur erronée
- type de détection (sûre, potentielle, etc.)
- autres informations

D'autre part, le dictionnaire de fautes fournit également une évaluation du taux de couverture de fautes permis par les vecteurs de test appliqués.

Tout ceci est valable pour une séquence de test donnée, car il faut générer un dictionnaire par séquence de vecteurs appliquée.
Ces quelques informations données, le lecteur peut déjà se rendre compte (l'ingénieur de test, lui, le sachant bien) de la complexité d'un dictionnaire de fautes, du temps nécessaire à sa génération, et surtout de la quantité d'informations fournies, dont une grande partie n'est finalement pas utilisable [MBG88].
Tous ces inconvénients ont rendu nécessaire la définition d'un certain nombre de stratégies rendant utilisable un dictionnaire de fautes dans le cadre du projet ADVICE.

Etant donné le contexte de travail - observation interne du circuit par microscopie électronique -, les fautes à considérer sont soit celles dont le siège est une connexion observable, soit celles dont le siège n'est pas directement observable, mais pour lesquelles une relation biunivoque permet d'associer, à l'aide d'une séquence de test connue, une séquence de valeurs mesurées en un point d'observation à une faute donnée (cette liste d'associations constitue typiquement un dictionnaire de fautes).
D'autre part, on considèrera les points internes aux "cellules feuilles", dans la décomposition hiérarchique du circuit sous test, comme non observables.
L'ensemble des modèles de fautes possibles est donc restreint aux fautes sur les connexions, c'est-à-dire les collages à 0 ou à 1 ("stuck-at-0/1") et les coupures de lignes ("opens"), elles-mêmes modélisées à leur tour comme des collages.
Les notions d'équivalence et de dominance de fautes [BrF76] sont prises en compte par le système HILO3 pour la réduction du nombre de fautes à simuler ("fault collapsing") : il suffit d'établir qu'une faute est détectable par la séquence de test appliquée pour en déduire que toutes les fautes qui lui sont équivalentes, ou qui la dominent, sont également détectables par cette même séquence de test.
Enfin, une faute est éliminée de l'ensemble des modèles après sa première détection (approche SOFE : "Stop On First Error" de la simulation de fautes).

En effet, une série d'expérimentations [Mel87b], dont la description suit, a montré que les tailles des dictionnaires de fautes générés sont de toutes façons inacceptables - du moins dans le cadre du système ADVICE - si cette approche n'est pas adoptée.
Les expériences de génération de dictionnaires de fautes ont été conduites pour un circuit du CSELT. Ce circuit, une unité arithmétique et logique d'un synthétiseur de parole, a une taille évaluable à 1859 portes logiques regroupées en 226 sous-circuits. Les conditions et résultats de ces expériences sont résumées en figure III.3.

EXP.	CONDITIONS			RESULTATS		
	fautes injectées	**réponses**	**# de détections**	**taille dict. (MO)**	**temps CPU**	**temps total**
1	toutes	sorties primaires	1 (SOFE)	806.5	76 mn	4 h
2	toutes	sorties primaires + noeuds internes	255 (max)	-- (trop grande)	276 mn	16h30
3	collages 0/1	sorties primaires	1 (SOFE)	258	23 mn	2 h
4	collages 0/1	sorties primaires + noeuds internes	255 (max)	7192.5	109 mn	7h15

Figure III.3. Expériences de génération de dictionnaire de fautes

La stratégie globale de génération de dictionnaire est la suivante [MBG88] :
Pour chaque sous-séquence de test (telle que définie en 3.3), deux niveaux de dictionnaire de fautes peuvent être générés, dont le premier, qui prend en compte toutes les fautes possibles, ne considère comme points d'observation que les sorties primaires du circuit, et le second niveau, qui ne prend en compte que le bloc (sous-circuit) que l'on veut tester, considère comme points d'observation les sorties de ce bloc, donc des points internes au circuit testé.
Le premier niveau de dictionnaire servira à une *détection externe* des erreurs (fonctionnement du testeur en mode analyseur logique, cf. § III.2.3), alors que le second niveau de dictionnaire servira à une *détection interne* des erreurs, en vue d'une localisation de fautes (fonctionnement en mode testeur par faisceau d'électrons, par guidage des noeuds à sonder).
Ces deux niveaux de dictionnaire sont générés suivant l'approche "SOFE", pour chaque sous-séquence de test, de façon à exploiter la structure séquentielle inhérente à chaque sous-séquence (cf. § III.3.3).

Ces dictionnaires de fautes représenteront les informations dont le système dispose sur le comportement incorrect du circuit sous test. Ils sont censés réduire le nombre d'observations et de mesures, ainsi que le temps de calcul nécessaires à la détermination du diagnostic. Comme la simulation et la génération de vecteurs de test, la simulation de fautes et la production des dictionnaires de fautes sont des processus "hors-ligne", c'est-à-dire effectués en dehors d'une session ADVICE.

III.4 LA STRUCTURE DE DONNEES DU SYSTEME ADVICE

La couche "méthode" du système ADVICE est essentiellement composée de procédures pour le diagnostic. Ces procédures doivent utiliser toutes les données dont on dispose sur le circuit sous test, sur son fonctionnement correct et sur son fonctionnement effectif lorsque des séquences de test lui sont appliquées. Pour ce faire, ces informations sont regroupées et organisées en une structure de données logique interne, dans laquelle les procédures de diagnostic doivent "naviguer".[18]

Cette structure de données, hiérarchique, reflète la description logique du circuit, ainsi que les informations relatives aux coordonnées physiques (sur le dessin des masques) des points à sonder, aux résultats de simulation, et aux erreurs observées.
La description du circuit, sous forme de schéma logique ("netlist"), est effectuée à l'aide du système HILO3. Un compilateur a été développé pour la traduction et le chargement de cette description dans la structure de données. La compilation se fait en deux phases, la première permettant de générer une structure de données intermédiaire ("unexpanded data structure"), qui sera le squelette de la structure de données finale, et la deuxième phase consistant à "étoffer" cette structure intermédiaire, pour obtenir la structure de données qui sera effectivement utilisée ("expanded data structure").

III.4.1 Première phase : génération d'une structure intermédiaire

Cette première étape permet de compiler une description sous forme de "netlist" au format HILO3 en une forme intermédiaire retraçant tous les modèles de cellules définis par le concepteur. La hiérarchie de description, ainsi que les connexions entre cellules y sont également reflétées. Cette phase de précompilation a été développée en collaboration avec des partenaires extérieurs au projet ADVICE.[19] Le précompilateur a été généré à l'aide des

---

18. Le terme de "navigation" est employé pour désigner le parcours de la structure de données logique interne par les procédures qui vont utiliser ces données. Ces procédures doivent notamment permettre le passage d'un niveau de représentation hiérarchique à un autre, par l'intermédiaire de pointeurs prévus à cet effet dans la définition de la structure de données.

19. Essentiellement Roberto Manione, du CSELT, qui participe au développement du compilateur de silicium ACCORDO [ABM] dans le cadre du projet ESPRIT N° 802 : "CAD for VLSI systems".

outils LEX [LeS75] et YACC [Joh75] du système UNIX.
Ni le précompilateur, ni la structure de données intermédiaire ne seront plus amplement décrits ici. Nous nous limiterons à la structure de données finale.

III.4.2 Deuxième phase : génération de la structure de données ADVICE

A partir du squelette de la description HILO3 obtenu précédemment, la structure de données ADVICE peut être construite, dans le but de satisfaire aux objectifs suivants :

- Charger la description du circuit, comprenant non seulement les modèles de cellules, mais aussi les instances d'appel de ces cellules
- Ajouter les champs relatifs aux procédures de navigation
- Créer des données supplémentaires reflétant les relations entre les points à sonder et leurs coordonnées physiques d'une part, et les relations entre les résultats de mesures et de comparaison d'autre part
- Créer des fichiers de commandes pour la simulation, la simulation de fautes, la génération des dictionnaires de fautes, et la sélection de séquences de vecteurs de test.

L'un des problèmes majeurs dans la construction de cette structure de données provient du fait que, d'une part, il est important de conserver la hiérarchie de décomposition du circuit, et que d'autre part cette hiérarchie n'est qu'une abstraction de la réalité physique, qui est la mise à plat du circuit.
Pour résoudre ce problème, deux structures ont été créées, l'une relative à une description logique ("netlist") hiérarchique, et l'autre à une description électrique mise à plat, la correspondance entre ces deux structures étant assurée par un système de pointeurs (cf. § III.3.2). Ainsi, la description logique hiérarchique sera le support de la navigation dans la décomposition en cellules du circuit, alors que la description électrique mise à plat contiendra les informations relatives aux connexions (points à sonder, résultats de mesure, erreurs détectées).

III.4.2.1 Description logique hiérarchique

La hiérarchie de décomposition du circuit est représentée par une structure d'arbre, dont les noeuds sont les cellules décrites par le concepteur, et les liens les connexions entre ces cellules à chaque niveau hiérarchique.

Une cellule comporte un nom, un pointeur sur sa vue topologique, une liste d'entrées/sorties permettant l'interfaçage avec les niveaux hiérarchiques supérieurs, et un pointeur sur la cellule suivante de même niveau hiérarchique.

La vue topologique d'une cellule comporte le nom du modèle dont cette cellule est une instance, un pointeur sur la cellule appelante, un pointeur sur son contenu (autres cellules de niveau hiérarchique inférieur), et divers indicateurs utiles au diagnostic, comme l'état de ce

bloc, le fait qu'il a déjà été visité, etc.

Les connexions entre les cellules sont décrites par une "netlist" globale, pour chaque niveau hiérarchique. Chaque élément de cette "netlist" est une équipotentielle définie par son nom, la liste des entrées/sorties de blocs connectés à cette équipotentielle, un pointeur sur la ligne correspondante au niveau électrique, et un pointeur sur l'équipotentielle suivante dans le même bloc, donc de même niveau hiérarchique.

III.4.2.2 Description électrique mise à plat

Le terme "électrique" est employé ici de façon impropre, car il ne fait pas référence à une véritable "netlist" électrique, mais plutôt à la représentation de connexions logiques qui soit sont observables par microscopie électronique, soit sont identifiables sur le dessin des masques du circuit. Cette description est dite "mise à plat" car elle ne comprend qu'un seul élément par équipotentielle, ignorant les différents noms logiques qui pourraient lui être associés.

Une équipotentielle est donc représentée à ce niveau par un numéro qui lui est affecté, une référence à son nom logique de niveau hiérarchique le plus haut dans le circuit (utile à la simulation), une référence à son nom logique utilisé pour la correspondance avec les coordonnées sur le dessin des masques, un pointeur sur la liste des points à sonder sur cette équipotentielle, un pointeur sur le point d'observation le plus proche (au sens de la définition 5, § III.3.2) de cette équipotentielle, un pointeur sur l'équipotentielle suivante dans le circuit, et diverses informations relatives aux mesures effectuées sur cette équipotentielle.

Cette structure de données ADVICE est très complexe, et très lourde à gérer. Nous n'avons pas pu la décrire dans le détail, pour des raisons de place, mais aussi et surtout de confidentialité imposée par le projet. Néanmoins, nous pourrons voir comment elle est utilisée dans la suite, grâce à la description de la couche "méthode" et de l'algorithme de guidage de la sonde électronique.

III.5 LA COUCHE METHODE DU SYSTEME ADVICE

Puisque la couche "méthode", associée au processus chargé des tâches de diagnostic, agira comme la couche supérieure du système ADVICE, y compris du point de vue de l'utilisateur, il a été décidé de l'organiser comme un interpréteur de commandes.
Cet interpréteur de commandes sera l'interface, d'une part entre les procédures définies dans le processus "méthode" lui-même (pour l'exploitation des dictionnaires de fautes, la sélection de séquences de test, etc.), et d'autre part entre le processus "méthode" et les autres processus de même niveau, supervisés par le "shell" ADVICE (comme le module de contrôle du microscope, le module de contrôle de l'équipement automatique de test, etc. cf. § III.2.1).

On notera que, dans cette définition de l'interpréteur de commandes, l'utilisateur du système est vu comme un processus à part entière du "shell" ADVICE ; de cette façon, trois modes de diagnostic sont permis : manuel, assisté, ou automatique.

III.5.1 Organisation générale de la couche méthode

En tant qu'interface entre différents processus, le processus "méthode" agit comme un "sous-shell" du "shell" ADVICE, à travers lequel les autres processus vont communiquer. Le rôle du "sous-shell" sera donc d'être à l'écoute des processus, et, lorsqu'une commande est formulée, de lire cette commande, déterminer sa validité, l'interpréter, pour finalement l'exécuter, en utilisant les facilités offertes par le "shell" ADVICE.

Considérant le grand nombre de commandes possibles et leur diversité d'une part, et considérant d'autre part l'aspect convivial que l'interpréteur de commandes doit présenter à l'utilisateur, il a été décidé d'organiser l'ensemble des commandes en cinq classes, chacune d'entre elles correspondant à un type de requête particulier, ainsi qu'à un type d'exécution particulier.
A chacune de ces commandes correspond une procédure, ou un ensemble de procédures plus ou moins complexes, qui l'exécute. Le détail de ces commandes figure dans [GMM88].

III.5.1.1 Classe 1 : commandes de type SET

La classe 1 contient les commandes de type "SET". Elles ont pour but d'assigner à un paramètre du système une valeur donnée.
Cette classe se subdivise en deux catégories, suivant que le paramètre auquel on veut assigner une valeur est un simple indicateur ("flag") ou une entité.

Les indicateurs se voient assigner une valeur pour guider la tâche de diagnostic, ou pour signifier qu'une information donnée est disponible ou non.

Exemple :

SET OUTPUTNODE <indicateur>
Cette commande permet de fixer les points dont la valeur logique est à mesurer.
<indicateur> peut prendre l'une des valeurs suivantes :
POONLY : seules les sorties primaires seront considérées
POINTERNAL : les sorties primaires et les noeuds internes observables seront considérés
INTERNALONLY : seuls les noeuds internes observables seront considérés (en cas de simulations partielles de sous-circuits).

Les entités se voient assigner une valeur pour guider la "navigation" dans la structure de données ADVICE, et pour indiquer laquelle d'entre elles (cellules, connexions, etc.) est considérée à un instant donné.

Exemple :

SET LISTRESPONSE [<sens> <valeur>]+
Alors que la commande précédente permet de préciser le cadre général de travail lors d'une session ADVICE, celle-ci fixe la liste de noeuds courants à considérer.
<valeur> est donc une liste de noeuds. Il était plus intéressant de décider que cette commande pouvait mettre à jour la liste courante plutôt que la remplacer : le paramètre <sens> sert à préciser ce type de mise à jour. Si <sens> prend la valeur "+", il s'agit d'ajouter des éléments à la liste courante, et si <sens> prend la valeur "-", il s'agit d'en retirer.
Supposons que la liste courante soit [rw1, rw2, rw4]. Après la commande :
SET LISTRESPONSE + rw3 rw6 - rw2 + rw5
la liste courante devient [rw1, rw3, rw4, rw5, rw6].

III.5.1.2 Classe 2 : commandes de type SHOW

La classe 2 regroupe les commandes de type "SHOW". Elles sont destinées à l'affichage des paramètres et de leurs valeurs. Comme les commandes de type "SET", ces commandes se subdivisent en deux catégories, suivant qu'elles concernent des indicateurs ou des entités.

Exemple :

SHOW BLOCK
Cette commande permet de visualiser le bloc courant (par défaut, celui de plus haut niveau hiérarchique).
Ainsi, soit la description suivante d'un circuit simple de nom "PP" (description en langage HDL du système HILO3 [Gen85]) :

```
cct a (a0,a1,a2)
not n1(a0,a2)
   n2(a1,a2);
input a2;
wire a0,a1;
cct b (b0,b1,b2)
a  n1(b0,b1,b2)
   n2(b0,b1,b2);
input b2;
wire b0,b1;
cct pp(aa,e,f,l)
a aaa(k,l,aa);
a bbb(k,l,aa);
b ccc(e,f,k);
```

```
wire k,l,e,f;
input aa;
```

Après chargement de ce circuit dans la structure de données ADVICE, l'exécution de la commande SHOW BLOCK (en mode "verbeux") donne les informations suivantes :

```
Block PP        of type PP information :
        It is NOT a LEAF cell
        It contains 8 elementary primitives
        in the 4 levels of hierarchy down to the leaves
        The BLOCK is the TOP LEVEL BLOCK
        The block has no PREDECESSOR
        The block has no SUCCESSOR
        The block calls :
                AAA   of type : A
                BBB   of type : A
                CCC   of type : B
        Pin number : 1 (type :  INPUT)
                No external connection wire
                Internal connection wire : AA
        Pin number : 2 (type :  OUTPUT)
                No external connection wire
                Internal connection wire : E
        Pin number : 3 (type :  OUTPUT)
                No external connection wire
                Internal connection wire : F
        Pin number : 4 (type :  OUTPUT)
                No external connection wire
                Internal connection wire : L
        It has NOT been VISITED yet with current pattern
        It was NOT found FAULTY with previous pattern
```

III.5.1.3 Classe 3 : commandes de type FILE HANDLING

La classe 3 contient les commandes de type "FILE HANDLING" (gestion et manipulation de fichiers). Elles servent aux opérations de création, chargement, sauvegarde et restauration. Les fichiers concernés sont soit de simples fichiers ASCII, soit des fichiers de structures.

Les commandes de chargement de fichiers dans la structure de données ADVICE sont parmi les plus complexes : chacune d'entre elles est exécutée par un compilateur. On distingue le chargement de la description du circuit ("LOAD CIRCUIT"), le chargement du

fichier des stimuli ("LOAD HISTORY"), le chargement de la correspondance entre les points d'observation et leurs coordonnées dans le dessin des masques, générée par l'outil commercial DRACULA [BGV88] ("LOAD DRACULA"), et le chargement des dictionnaires de fautes ("LOAD DICTIONARY").

Les commandes de création de fichiers vont servir à générer l'exécution de processus "hors-ligne". Elles concernent la simulation ("CREATE SIMULATION"), la simulation de fautes ("CREATE FSIMULATION"), et la génération de la correspondance entre les points d'observation et leurs coordonnées ("CREATE DRACULA").

Les commandes de sauvegarde et restauration ("SAVE SESSION", "RESTORE SESSION") d'une session ADVICE sont également très complexes, car elles correspondent à la sauvegarde et à la restauration de *toutes les informations* présentes dans la structure de données ADVICE, y compris les résultats de mesure et de comparaison.

III.5.1.4 Classe 4 : commandes de type EXEC

La classe 4 contient les commandes de type "EXEC". Elles permettent l'exécution de tout ou partie des processus définis, qu'ils soient internes ou externes au processus "méthode". Dans la suite, les principales commandes de ce type sont définies.

La sélection de séquences de test se fait par exécution du processus TPS ("Test Pattern Selector"). Cette commande ("EXEC TPS") admet comme paramètre le mot-clé "NEXT" (séquence qui suit chronologiquement la séquence courante), un nom de cellule (séquence qui sensibilise cette cellule), une faute donnée (séquence qui exhibe cette faute), ou un instant en cycles d'horloge (séquence dont les instants de début et de fin englobent cet instant).

L'utilisation du dictionnaire de fautes se fait à travers l'exécution du processus FDN ("Fault Dictionary Navigator"). Cette commande ("EXEC FDN") admet comme paramètre soit le mot-clé "SEARCH" (recherche d'une différence entre une valeur attendue et une valeur mesurée, c'est-à-dire recherche d'une erreur manifestée), soit le mot-clé "PRUNE" (élagage des fautes non observables et/ou équivalentes).

La mesure et la comparaison d'états logiques font appel au processus CURVES. Cette commande ("EXEC CURVES") s'exécute pour une connexion, mesurée et simulée entre deux instants donnés.

Le processus EXERCIZE, lié à la commande "EXEC EXERCIZE", a le même rôle que le processus précédent, sauf qu'il se restreint à la mesure et la comparaison d'états logiques des sorties primaires du circuit.

Le processus LAYOUT est appelé par la commande "EXEC LAYOUT". Son rôle est soit de mettre en évidence un point d'observation sur la fenêtre d'affichage du dessin des masques lorsque le paramètre donné est le mot-clé "DISPLAY", soit d'ajuster le

positionnement "fin" du faisceau sur le point à observer, lorsque le paramètre donné est le mot-clé "PLACE".

On peut faire appel à l'algorithme de guidage de la sonde, pour décider du prochain point à observer, par la commande "EXEC PROBALG". Si aucun paramètre n'est donné, c'est que le diagnostic doit se faire en mode complètement automatique. Si par contre un paramètre est donné (cellule, connexion, ou point d'observation), il s'agit du mode assisté, et on veut savoir quelle est la prochaine cellule, ou la prochaine connexion, ou quel est le prochain point d'observation, à examiner.

III.5.1.5 Classe 5 : commandes de type SYSTEM

La classe 5 regroupe les commandes de type "SYSTEM". Il s'agit de commandes d'ordre général qui permettent de faire appel à un manuel en-ligne, de changer l'entrée et/ou la sortie standard, de retourner au système d'exploitation hôte (VMS) pour des besoins spécifiques, et ceci sans quitter une session ADVICE, et de quitter le sous-shell "méthode".

III.5.2 L'algorithme de guidage de la sonde

Nous avons pu voir (§ III.2.3) que le système ADVICE pouvait être adaptable, du point de vue du degré d'automatisation du diagnostic. Ce degré d'automatisation est paramétré par la façon dont on réalise la tâche de détermination des noeuds à observer.
Lorsque l'utilisateur du système décide seul, en fonction de la comparaison entre résultats de mesure et résultats de simulation, de s'intéresser à un point particulier du circuit, le système est utilisé en mode *manuel*.
Lorsque l'utilisateur effectue son choix en fonction d'informations qui l'aident dans sa décision (comme une pré-localisation de fautes par un dictionnaire de fautes, par exemple), le système est utilisé en mode *assisté*.
Lorsque la localisation de fautes se fait sans intervention de l'utilisateur, le système est utilisé en mode *automatique*.

Ainsi, les tâches de diagnostic dans le système ADVICE vont se subdiviser en tâches exécutives, et en tâches analytiques. Les premières sont lancées à travers les commandes du "sous-shell méthode" présentées en III.5.1, nous n'y reviendrons pas. Les secondes auront en charge la sélection effective des prochains points à observer ou, de façon plus globale, la prochaine cellule à examiner.

La figure III.4 [MBG88] retrace le déroulement de l'algorithme de guidage de la sonde. Elle sera commentée dans la suite de ce paragraphe.

L'algorithme démarre à partir de la description du circuit au niveau hiérarchique le plus haut, pour identifier une cellule (bloc) non encore visitée, sélectionner une séquence de test permettant de la sensibiliser, et effectuer un premier test au niveau des sorties primaires.
En cas de réponse erronée, le dictionnaire de fautes est consulté, dans le but de vérifier si une faute déjà simulée a pu provoquer le même comportement incorrect.

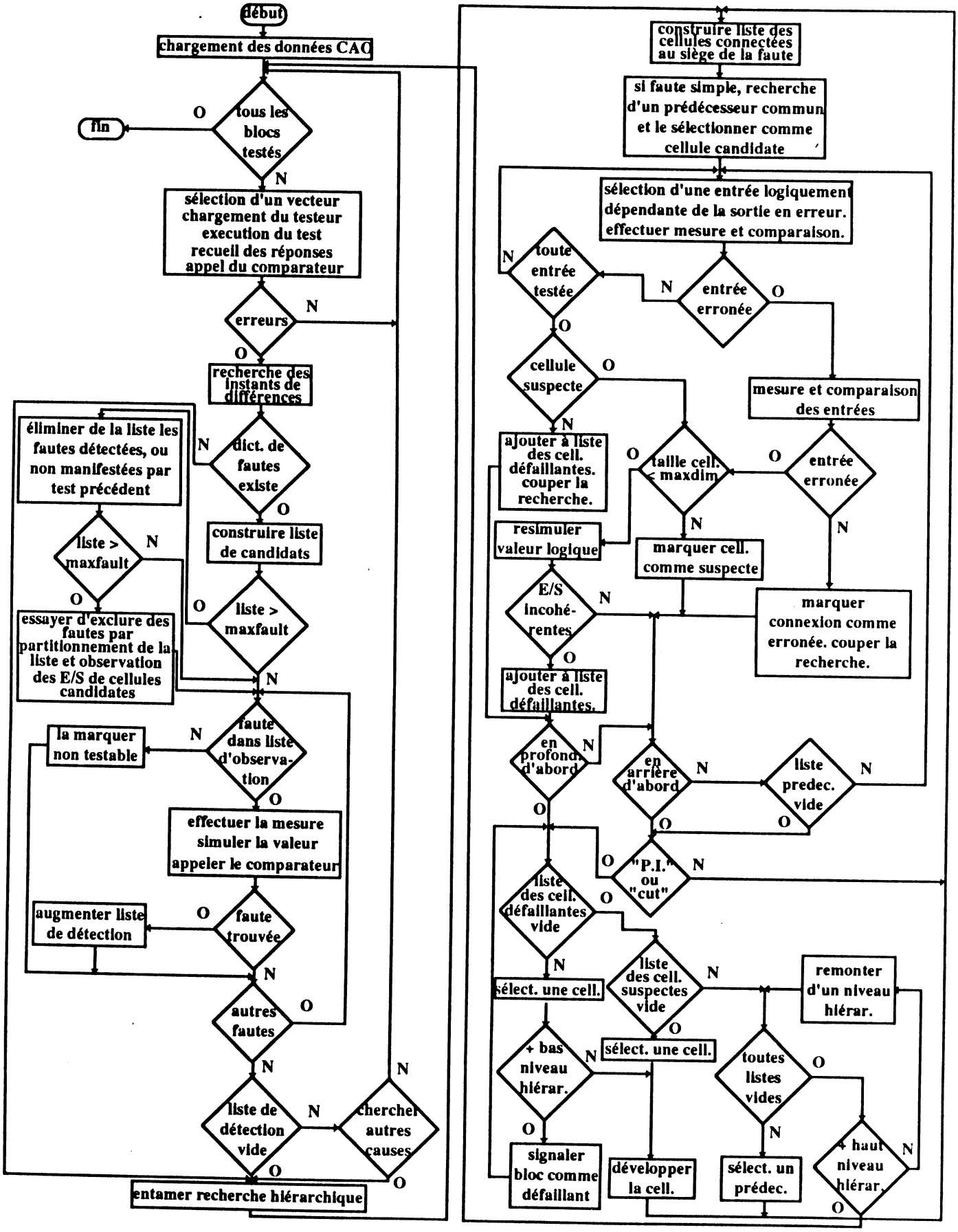


Figure III.4. Déroulement de l'algorithme de guidage

Si la liste de fautes répondant à ces caractéristiques est plus longue que le maximum fixé (un paramètre de l'algorithme), alors il y a tentative de raccourcissement de cette liste, excluant les fautes déjà détectées précédemment, et on effectue des mesures à l'aide de la sonde électronique sur le site de ces fautes possibles : en ce sens, le dictionnaire de fautes aura permis une pré-localisation de fautes.

Si par contre on ne trouve dans le dictionnaire de fautes aucune faute répondant aux caractéristiques définies plus haut, ou si, à la suite des observations effectuées, on montre que les fautes qui étaient candidates ne sont pas en réalité celles que l'on recherchait, l'algorithme entame une analyse de la structure du circuit, à partir des sorties primaires erronées, remontant vers les entrées primaires jusqu'à trouver une cellule qui serait susceptible de contenir la faute. Lorsque cette cellule est identifiée, son contenu est examiné, de façon à obtenir un diagnostic plus précis, à un niveau hiérarchique inférieur.
L'algorithme stoppe la recherche lorsqu'une connexion, ou une cellule primitive (au sens du système HILO3), est trouvée défaillante.

Cette tâche de diagnostic essaie donc de minimiser le temps et le coût de la localisation de fautes. Les procédures employées peuvent être contrôlées par l'utilisateur, à divers degrés, puisque des paramètres de l'algorithme peuvent être ajustés à cet effet. On pourra trouver tous les détails à ce sujet dans [GMM88].

III.6 CONCLUSION

Le projet ADVICE est actuellement achevé. Tout n'est pas réalisé, notamment au niveau de la couche méthode, comme nous le verrons au chapitre V, mais il a représenté un cadre de réflexion intéressant, dans le sens où les recherches et les développements effectués ont permis d'aller de l'avant par rapport aux systèmes commerciaux existants.

Les caractéristiques du système réalisé - niveaux T.F.E. et T.F.E. étendu - figurent au chapitre V, alors que le présent chapitre a été centré sur la méthode de diagnostic et sur les stratégies, développées dans le cadre du projet, pour le test par faisceau d'électrons.

Le système ADVICE est encore à l'état de prototype, et plusieurs des choix effectués sont, à notre avis, à revoir, en particulier le choix d'utiliser un dictionnaire de fautes pour la localisation, et de débuter le processus d'analyse par une simulation logique du circuit tout entier, ce qui est peu réaliste dans le cas des circuits VLSI.
D'autre part, utiliser les formes d'onde comme unique moyen de mesurer les valeurs de potentiel en un grand nombre de points du circuit n'est pas raisonnablement envisageable (cf. conclusion du chapitre II) : puisque le mode image est disponible dans le système, il aurait pu être également utilisé pour observer simultanément plusieurs points, et non pas

seulement pour ajuster le déplacement de la platine du microscope.
Tous ces choix seront discutés et évalués au chapitre V, et la méthode de localisation de fautes du système ADVICE fera l'objet d'une comparaison avec une autre méthode de diagnostic, développée au laboratoire, dont la présentation est effectuée au chapitre suivant.

IV. PESTICIDE : APPROCHE BASEE SUR LA CONNAISSANCE

IV.1 INTRODUCTION

Le système PESTICIDE (acronyme de "a Prolog-written Expert System as a Tool for Integrated CIrcuits DEbugging") constitue la mise en oeuvre d'un deuxième type de méthode de diagnostic, c'est-à-dire de localisation de fautes, pour des circuits soumis à un test par faisceau d'électrons.
Nous avons développé ce système au sein du laboratoire,[20] dans le cadre du projet de validation de prototypes de circuits intégrés par microscopie électronique. Ce système avait pour objet de montrer la faisabilité d'une approche basée sur la connaissance pour la résolution du problème de validation de prototypes.

Le système PESTICIDE a été pensé et conçu en deux temps : la première étape, qui a été consacrée à l'étude du domaine (test de circuits VLSI, et particularités de l'observation au microscope électronique), s'est soldée par une premiere maquette, pour laquelle nous n'avons considéré que l'étude des circuits combinatoires. A partir de cette première ébauche, nous avons pu mettre au point de nouvelles méthodes et stratégies de diagnostic [Mar87a].
En effet, une évaluation critique de ce noyau a été conduite, à partir des problèmes rencontrés, qui a abouti aux considérations suivantes :

- Tout d'abord, il fallait envisager le cas des circuits séquentiels, étant donné que seuls les circuits combinatoires avaient été pris en compte jusqu'alors. Le principal problème qu'il fallait résoudre était lié à la notion de mémorisation, et par conséquent de retard dans la transmission de l'information à travers le circuit.

- Ensuite, et ceci était aussi valable pour les circuits séquentiels que pour les circuits combinatoires, il fallait étudier d'autres données que les propriétés structurelles des circuits, à savoir leurs propriétés logiques et fonctionnelles, et voir dans quelle mesure ces propriétés intervenaient dans la localisation des défaillances.

---

20. Ce travail a bénéficié du soutien du GCIS-CNRS (Groupement Circuits Intégrés sur Silicium), axe "outils de C.A.O. et test", pendant l'année 1989. Il est également issu d'une phase d'analyse ayant partiellement été soutenue par le programme ESPRIT de la CCE (projet N° 271 : ADVICE), pendant la première période allant de 1986 à 1987.

- De plus, il fallait prendre en compte l'existence de *fautes* sur les connexions, puisque, jusqu'alors, seules les *erreurs* observées sur les connexions avaient été considérées, ces erreurs étant la manifestation de fautes survenant au niveau des blocs.
- Enfin, un autre problème concernait le type des connexions, et l'on avait pu voir alors que le cas de connexions bidirectionnelles pouvait se présenter. La solution que nous avions adoptée dans la première ébauche était de formuler l'hypothèse d'unidirectionnalité de toutes les connexions. Il s'agissait donc de différencier les connexions, et de traiter chacune d'entre elles selon son type.

A ces quelques remarques préliminaires, il nous faut ajouter l'étude, que nous avons menée sur les systèmes à base de connaissances de seconde génération (cf. première partie), dont certains résultats se retrouvent en substance dans [MLC89], et qui nous a montré que le problème que nous avions à considérer entre tout à fait dans le cadre du raisonnement à base de modèles profonds ("deep knowledge"), du moins pour la tâche de diagnostic proprement dite.

Cette assertion, ainsi que tous les choix majeurs que nous avons pu faire, seront plus largement argumentés dans le chapitre suivant, au cours de la comparaison entre la méthode de diagnostic utilisée par le système ADVICE et celle implémentée dans PESTICIDE. Pour des raisons évidentes de clarté, nous nous contenterons dans ce chapitre de décrire les différentes caractéristiques de PESTICIDE, après en avoir brièvement présenté l'environnement.

IV.2 ENVIRONNEMENT DE PESTICIDE

PESTICIDE, en tant que méthode de diagnostic, peut être vu comme la couche la plus externe d'un système intégré de diagnostic par faisceau d'électrons, suivant la classification opérée au chapitre II. A ce titre, il interagit avec les autres couches, qui lui fournissent les informations dont il a besoin, et qui prennent en compte ses résultats et/ou demandes d'informations complémentaires. D'autre part, en tant qu'application de niveau d'abstraction le plus élevé dans l'ensemble du système de diagnostic, PESTICIDE doit également assurer l'interaction avec les utilisateurs de ce système.

On peut alors voir le système intégré de test comme un ensemble formé de deux composantes principales, l'une dédiée à l'acquisition de l'information, et l'autre au traitement de cette information.
Dans la suite, nous nommerons la première le système d'acquisition des informations (S.A.I.) et la seconde le système de traitement des informations (S.T.I.).

IV.2.1 Système d'Acquisition des Informations

Il correspond à la réunion des deux couches de plus bas niveau, c'est-à-dire au testeur par faisceau d'électrons. Ce système d'acquisition des informations est fonctionnellement divisé en trois composants, qui sont [BaL82] :

- Un conditionneur ("exercizer") qui assure la contrôlabilité externe du circuit sous test, par simulation de son environnement extérieur.
- Un microscope électronique à balayage, utilisé en mode contraste de potentiel, et muni d'un sous-système de commande.
- Un processeur de traitement d'images permettant leur numérisation et leur stockage en mémoire.

Le schéma fonctionnel du système d'acquisition des informations est donné en figure IV.1.

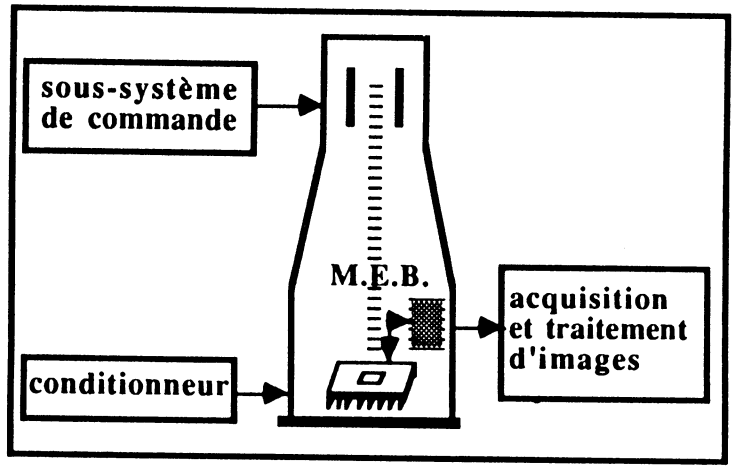


Figure IV.1. Schéma fonctionnel du S.A.I.

IV.2.2 Système de Traitement des Informations

Il constitue la réunion des deux couches supérieures du système intégré de diagnostic. Ce que nous appelons "T.F.E. étendu" (cf. chapitre II) va jouer en quelque sorte le rôle de pré-processeur vis-à-vis de PESTICIDE, interprétant les résultats de l'observation du circuit sous test par le T.F.E., puis comparant ces résultats à des données de référence, en l'occurrence celles fournies par une simulation du fonctionnement correct du circuit ("fault-free simulation") et celles portant sur la description structurelle de ce circuit. Toutes ces données sont issues d'outils de C.A.O. externes au système intégré de test, et constituent la référence de circuit correct ("golden device") à laquelle sera comparé le circuit sous test.
Les tâches assumées par le "T.F.E. étendu" sont alors :

- L'interprétation des images en niveaux de gris et/ou des valeurs mesurées (cas du mode forme d'onde, cf. chapitre II), de façon à obtenir les valeurs de potentiel, puis les

valeurs logiques sur les points internes du circuit qui ont été observés.

- La comparaison des valeurs ainsi obtenues aux résultats de simulation, pour déduire finalement l'état (erroné ou non) de chaque point observé.

D'autre part, une troisième tâche doit être effectuée ; elle n'entre pas forcément dans les attributions du "T.F.E. étendu", mais fait néanmoins partie du traitement global des informations. Cette tâche consiste dans le partitionnement de la structure du circuit qui, rappelons-le, est connue puisqu'il s'agit de validation de prototypes, donc le diagnostic se fait en collaboration avec le concepteur du circuit. Ce partitionnement doit fournir une représentation sous forme de blocs logiques interconnectés.

Toutes ces informations issues du "T.F.E. étendu" vont alimenter PESTICIDE [MLC89] pour l'établissement du diagnostic, comme le montre la figure IV.2.

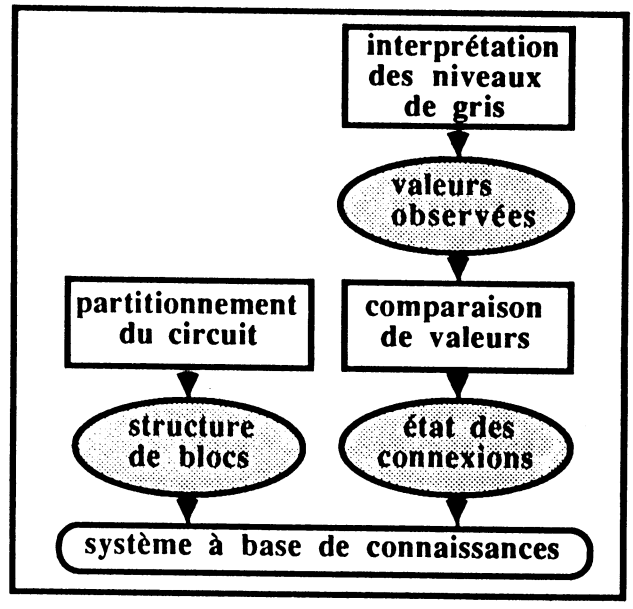


Figure IV.2. Schéma fonctionnel du S.T.I.

IV.2.3 Intégration des différents composants

La communication entre le système d'acquisition des informations, le système de traitement des informations, et l'environnement extérieur est formée de la circulation de diverses données, qui, dans le cadre d'un système de validation de prototypes automatique et intégré, doivent être [MaC89] :

- Des images et/ou valeurs mesurées, du T.F.E. vers le "T.F.E. étendu".
- La description de la structure du circuit, et les résultats de simulation, de l'environnement extérieur (outils de C.A.O.) vers le "T.F.E. étendu".

- Certaines hypothèses sur le type de test que l'on veut pratiquer, de l'environnement extérieur (utilisateur du système) vers PESTICIDE.[21]
- Une liste des noeuds ou zones à observer pour complément d'information, de PESTICIDE vers le T.F.E.
- Une liste de vecteurs - ou de séquences de vecteurs - de test, à appliquer au circuit pour vérification de conjectures, de PESTICIDE vers le T.F.E.
- Le diagnostic final - ou même partiel, si l'on veut le système fortement interactif -, de PESTICIDE vers l'environnement extérieur (utilisateur du système).

La figure IV.3 [MaC89] décrit, dans une vue synoptique du système intégré de test, la circulation des données à travers ce système.

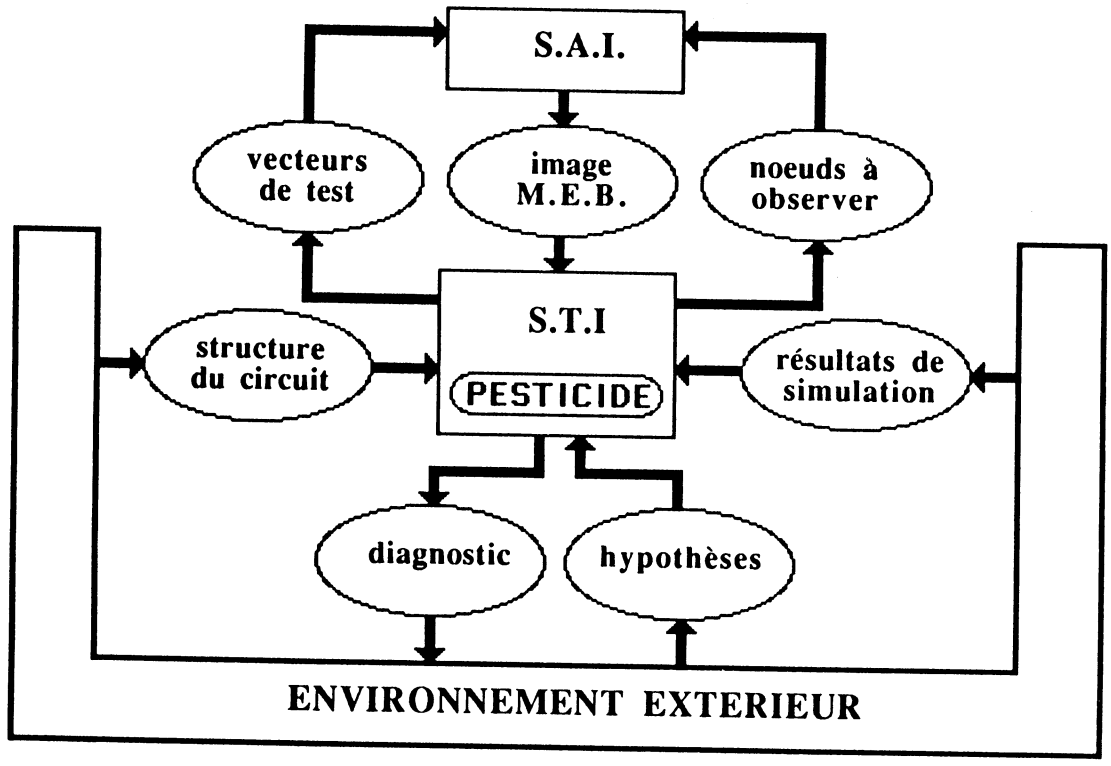


Figure IV.3. Vue synoptique du système intégré de test

21. Nous verrons dans la suite que ces hypothèses peuvent être absentes, et qu'elles ne servent qu'à accélérer le processus de diagnostic.

IV.2.4 Limitation liée à l'environnement

L'environnement du système PESTICIDE, et au-delà de ce système, de toute méthodologie de test par faisceau d'électrons, impose une limitation de la gamme des circuits traités.
Les circuits considérés seront soit combinatoires, soit séquentiels synchrones. En effet, l'utilisation du testeur par faisceau d'électrons ne permet d'effectuer des mesures que pour les circuits synchrones,[22] du moins en l'état actuel de l'observation à l'aide de la stroboscopie de faisceau (problèmes de synchronisation).

D'autre part, on supposera que les plots du circuit constituant respectivement l'alimentation (VDD), la masse et les horloges ont préalablement été contrôlés et sont dans un état correct.
Ceci n'est pas à proprement parler une limitation, puisque l'application d'une méthodologie de localisation de fautes pourrait être envisagée en-dehors d'une telle supposition. Néanmoins, il serait peu intéressant d'utiliser un outil comme le testeur par faisceau d'électrons et de mettre en oeuvre une méthodologie de haut niveau, pour localiser une faute somme toute identifiable par des moyens bien moins coûteux. De toutes façons, pour que le circuit puisse être observé à l'aide du testeur par faisceau d'électrons, son fonctionnement ne doit pas être complètement bloqué, comme par un défaut d'alimentation extérieure par exemple. Cette remarque n'est d'ailleurs pas spécifique à la validation de prototypes de circuits intégrés, puisqu'elle est également formulée dans le cas de l'analyse de défaillances [Sav90].

IV.3 MODELISATION DES CONNAISSANCES

Dans la modélisation utilisée, le circuit sous test est hiérarchiquement décomposé en blocs interconnectés, de façon à ce que chaque niveau hiérarchique corresponde à un niveau de complexité dans la description (figure IV.4 [MLC89]) ; ainsi, le niveau 0 (niveau d'abstraction le plus élevé) représente le circuit dans son ensemble (vu comme un seul bloc), alors que le niveau le plus profond peut être une description en termes de portes logiques, de transistors, ou même sous forme de dessin des masques du circuit, selon le test que l'on veut effectuer, et le degré de précision auquel on veut localiser le(s) faute(s).

Nous considèrerons un circuit séquentiel comme effectivement de ce type lorsque l'image que nous en avons à une étape donnée du test (c'est-à-dire l'image correspondant à

22. Il s'agit des circuits séquentiels. Rappelons que ce problème ne se pose pas pour les circuits combinatoires complètement asynchrones, puisque dans ce cas on n'observe que des états *stables*.

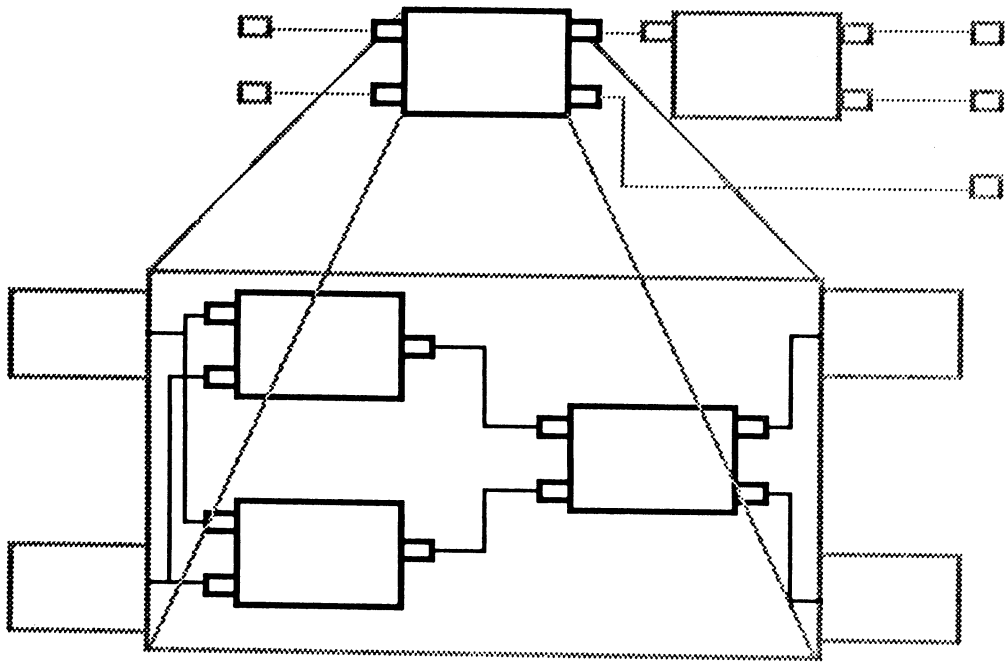


Figure IV.4. Décomposition hiérarchique de la description structurelle

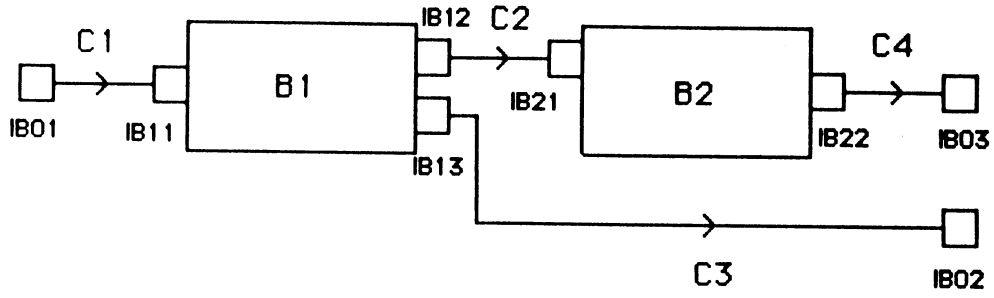
une décomposition structurelle donnée) est formée de blocs dont l'interconnexion fait apparaître des "boucles" dans le circuit, qui devient par là un système séquentiel de blocs. Autrement dit, la séquentialité "interne" d'un bloc (non visible dans la décomposition adoptée) sera ignorée pour cette décomposition.

Le modèle utilisé permet de décrire un circuit à l'aide de trois éléments de base, qui sont :

- Le bloc
- La connexion
- L'interface bloc

Ces trois composants de base, étant donné le pré-supposé formulé ci-dessus, vont permettre de modéliser aussi bien des circuits séquentiels que combinatoires, moyennant l'association de paramètres pertinents à chacun des composants.
Ainsi, l'exemple présenté en figure IV.5 représente la modélisation d'un circuit combinatoire, même si le bloc B1 est lui-même séquentiel, alors que celui de la figure IV.6 modélise un circuit séquentiel, lorsque les blocs B1 et B3 sont eux-mêmes des blocs séquentiels.

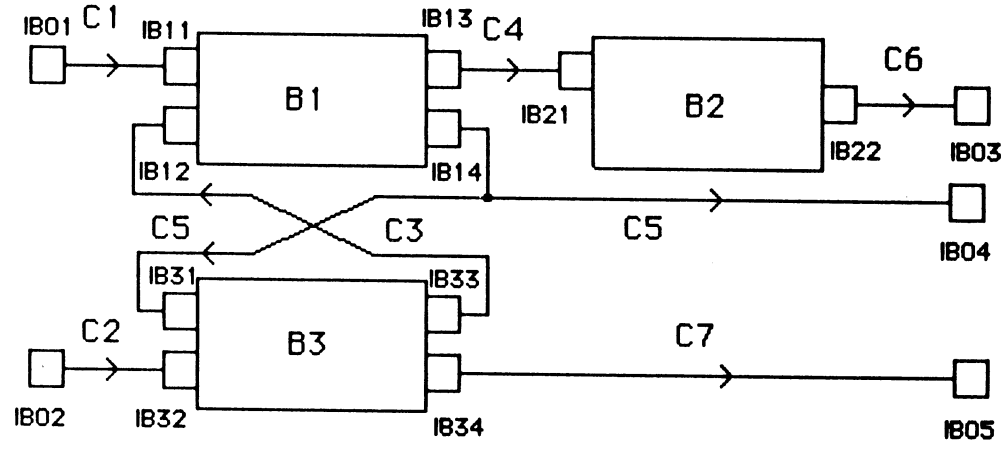
Nous verrons dans la suite de ce chapitre pourquoi, et comment, cette modélisation va permettre de représenter aussi bien les circuits séquentiels que combinatoires.



Bi : blocs ; Ck : connexions

IBij : interfaces bloc du bloc i

Figure IV.5. Modélisation d'un circuit combinatoire



IB12 - IB14 - IB31 - IB33 : boucle de rétroaction potentielle

Figure IV.6. Modélisation d'un circuit séquentiel

IV.3.1 Description des différents constituants d'un circuit

Les trois composants de base identifiés sont décrits dans ce paragraphe.

IV.3.1.1 Le bloc

Aucune hypothèse n'est faite a priori sur le bloc (séquentiel ou combinatoire), sauf que le partitionnement du circuit est fait et donne lieu à une certain nombre de ces blocs (le problème du partitionnement automatique est traité au chapitre V).
Les paramètres associés à un bloc sont définis comme suit :

- *Le nom :*
  Il s'agit d'un nom symbolique (ou d'un numéro) dont l'affectation au bloc concerné est effectuée par l'outil de partitionnement du circuit.

- *la liste des interfaces :*
  Il s'agit des interfaces (entrées, sorties, bidirectionnelles) reliant le bloc à son environnement extérieur. Chaque élément de cette liste est du type interface bloc, décrit au paragraphe suivant.

IV.3.1.2 L'interface bloc

L'interface bloc est donc l'élément qui permet de relier un bloc à son environnement extérieur, c'est-à-dire aux autres blocs et aux plots du circuit sous test. La raison majeure de l'introduction de cet élément en tant qu'entité constituante du circuit est d'une part de permettre la détection de fautes sur les connexions (et non plus seulement d'erreurs), et d'autre part, par la prise en compte d'arguments comme le délai (voir plus loin le détail des paramètres associés à une interface bloc), d'élargir la gamme des circuits testables aux circuits séquentiels, car le délai permet également d'exprimer le séquencement des états logiques du circuit sous test.

Bien qu'assez difficile à localiser précisément en termes de coordonnées dans le dessin des masques du circuit sous test (la notion de blocs étant une "vue de l'esprit", comment décider, sur une ligne de métal, où s'arrête l'interface bloc et où commence la connexion proprement dite?), cette notion d'interface bloc peut être d'une grande utilité si la conception du circuit s'est faite à partir de cellules (prétestées ou non) de bibliothèque.
Les différents paramètres associés à une interface bloc sont les suivants :

- *le nom :*
  Pour l'affectation du nom, cf. nom d'un bloc.

- *l'état :*
  L'état d'une interface bloc est lié au test en cours : il s'agit donc d'une donnée dynamique du système. L'état d'une interface bloc peut avoir l'une des trois valeurs suivantes : "erroné", "non erroné", "inconnu", où "inconnu" est l'état initial de chaque interface bloc, et "erroné" ou "non erroné" est l'état résultant de la comparaison entre la valeur observée au M.E.B. et la valeur attendue.

- *la valeur :*
  Il s'agit de la valeur logique du potentiel sur cette interface bloc. Il est intéressant de connaître la valeur et non pas seulement l'état pour la détection de fautes sur les connexions (pour plus de détails, voir plus loin la méthode de détection de ces fautes).

- *l'instant :*
  Il s'agit de l'instant de mesure de cette valeur, qui s'exprime en nombre de tops d'horloge écoulés depuis le début du test en cours. Notons que cet instant est aussi

valable pour l'état de l'interface bloc (dans ce cas, il est bien sûr fait abstraction du temps mis pour comparer la valeur observée à la valeur attendue).

- *le type :*
  Le type ou sens de l'interface bloc peut prendre l'une des trois valeurs suivantes : unidirectionnel d'entrée, unidirectionnel de sortie ou bidirectionnel. Ce paramètre, nécessaire si l'on veut pouvoir vérifier la cohérence des données sur les entrées et sorties de blocs, servira aussi à élargir la gamme des circuits testables par le système expert, et permettra également une classification des interfaces bloc et des connexions.

- *le délai :*
  Le délai d'une interface bloc représente le temps au bout duquel la valeur d'une interface bloc peut être prise en compte. Ce temps est également exprimé en nombre de tops d'horloge. Ce nombre est évalué comme suit :

  * le type de l'interface bloc est "unidirectionnel d'entrée" : le délai est le temps de propagation sur la connexion menant à cette interface bloc.

  * le type de l'interface bloc est "unidirectionnel de sortie" : le délai représente le temps de fonctionnement du bloc nécessaire à la stabilisation de la sortie.

  * le type de l'interface bloc est "bidirectionnel" : le délai est évalué comme dans le premier cas ou le deuxième, la détermination du cas pertinent se faisant sur une condition particulière.

    Exemple de condition :

    une interface bloc de type "unidirectionnel d'entrée", représentant une entrée "enable" et ayant une valeur de potentiel égale à "1" implique que l'interface bloc de type "bidirectionnel" considérée devient "unidirectionnelle de sortie" (deuxième cas). Cette entrée "enable" sert donc d'interrupteur pour passer du premier cas au second).

Remarque :
Pour l'un quelconque des trois cas précédents, le délai peut être nul. En effet, cette notion de délai a été introduite pour la prise en compte de circuits séquentiels, et si l'on a une partie de circuit uniquement combinatoire, on peut considérer les délais comme nuls pour cette partie (seuls les états stables sont observés).
Il est donc bien clair que, dans notre définition, le délai ne représente pas les caractéristiques électriques des transistors réalisant le circuit, mais plutôt le séquencement interne des états logiques de ce circuit.

- *le temps de maintien :*
  Le temps de maintien d'une valeur logique sur une interface bloc est le temps (en nombre de tops d'horloge) au bout duquel cette valeur logique va changer, sous l'effet de la propagation d'un signal à travers le circuit. Cette donnée sera considérée comme fournie par le concepteur du circuit.

IV.3.1.3 La connexion

Une connexion est une ligne physique reliant deux blocs à travers leurs interfaces bloc respectives, ou reliant un bloc aux plots du circuit, représentés par des interfaces bloc particulières.
Notons que le même problème que pour les interfaces bloc se pose pour la localisation en vue de l'observation d'une connexion : sur le dessin des masques, une connexion commence là où se termine une interface bloc, et se termine là où commence une autre interface bloc.

Le fait de considérer les connexions comme des entités permet de :

- Détecter des fautes sur les lignes physiques comme les coupures ou les courts-circuits.
- Vérifier la cohérence de certaines informations comme la directionnalité des connexions et des interfaces bloc, mais cette vérification se paie par une redondance de l'information (cf. § IV.5).
- Eviter de supposer que toutes les connexions sont de même type : unidirectionnel ou bidirectionnel.

Les différents paramètres associés à une connexion sont les suivants :

- *le nom :*
  Pour l'affectation du nom, cf. nom d'un bloc.
- *la liste des interfaces :*
  Il s'agit de la liste des interfaces bloc se trouvant en amont et en aval de la connexion. Chaque élément de cette liste est du type interface bloc détaillé ci-dessus, et comporte par conséquent tous les paramètres nécessaires.
  Il est à ce propos important de noter le détail suivant : le type d'une interface bloc est défini comme étant unidirectionnel d'entrée ou unidirectionnel de sortie, mais il est bien entendu qu'il s'agit d'entrée et de sortie de bloc. Une interface bloc, si elle est de type unidirectionnel d'entrée, est donc en aval d'une connexion, et inversement, si elle est unidirectionnelle de sortie, elle est en amont d'une connexion. Les interfaces bloc de type bidirectionnel seront considérées de la même façon, une fois les conditions nécessaires à leur entière détermination élaborées (c'est-à-dire les conditions qui permettent de les considérer tour à tour comme des entrées ou des sorties).

IV.3.2 Connaissances additionnelles

IV.3.2.1 Propriétés fonctionnelles

Les connaissances décrites jusqu'à présent sont essentiellement des connaissances sur la structure du circuit. Elles sont bien sûr nécessaires, mais ne suffisent pas au processus de localisation de fautes. Il faut donc ajouter à ces connaissances structurelles la donnée de connaissances fonctionnelles.
Il ne s'agit pas ici de connaître précisément le fonctionnement du circuit, mais de disposer d'un minimum d'information.

- *Notion de cône de couverture :*
  Il est important de savoir, pour chaque bloc défini par l'opération de partitionnement (qu'elle soit effectuée manuellement ou automatiquement), quelle est la relation entre les entrées et les sorties (fonction de transfert) de ce bloc.
  Par relation entre les entrées et les sorties, nous entendons une sorte de "lien de cause à effet" entre ces différents éléments. En d'autres termes, ceci permet de savoir, pour chaque sortie, quelles sont les entrées qui agissent sur (ou qui activent) cette sortie.
  En effet, supposons que l'on ne possède pas cette information, et que l'on observe une erreur sur l'une des sorties et sur l'une des entrées d'un même bloc, une règle de propagation peut être alors activable, et son application mener à une conclusion erronée si l'entrée considérée n'agit pas sur la sortie en erreur.
  Pour éviter ce genre de problèmes, on prévoit un élément de connaissance dénommé "cône de couverture" pour chaque interface bloc de sortie, constitué de la liste des interfaces bloc d'entrée qui agissent sur cette sortie, et permettant d'exprimer ces relations de dépendance fonctionnelle [Mac84].

  Bien que non absolument nécessaire, cette notion de cône de couverture permet d'améliorer le fonctionnement du système de diagnostic par la réduction du nombre d'alternatives (chemins de données ou "datapaths") à explorer [MLC89].

  La figure IV.7 illustre cette notion. Les interfaces bloc d'entrée qui constituent le cône de couverture de l'interface bloc de sortie O_2 sont I_2, I_3 et I_4, puisque les valeurs que O_2 peut prendre dépendent (au moins potentiellement, d'après la topologie du circuit) des valeurs prises par I_2, I_3 et I_4, ainsi que de la fonctionnalité du sous-bloc SB_2. La figure IV.8 illustre quant à elle l'utilité de cette notion de cône de couverture : supposons que des erreurs sur la sortie O_2 et sur les entrées I_3 et I_5 sont observées. Si toutes les entrées du bloc représenté étaient non différenciables, alors des règles de propagation arrière pourraient être activées sur les chemins O_2 - I_3, et O_2 - I_5, alors que l'entrée I_5 n'est pas fonctionnellement liée à la sortie O_2. La notion de cône de couverture, en différenciant les entrées d'un bloc, permet d'éviter ce genre d'activations parasites car inutiles, et contribue en cela à l'augmentation des performances du système.

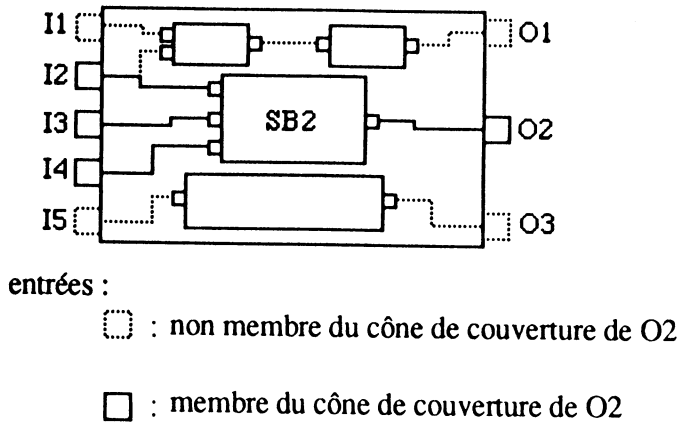


Figure IV.7. Notion de cône de couverture

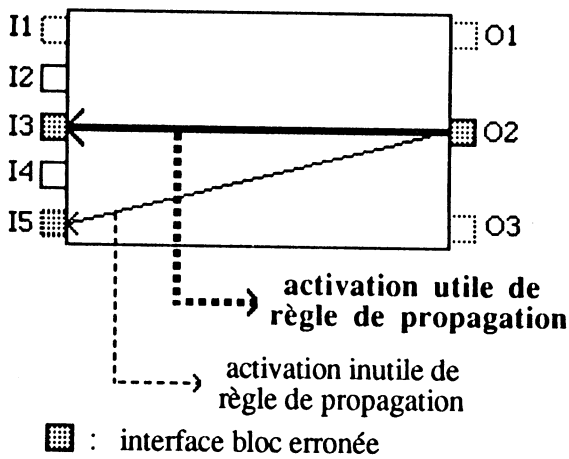


Figure IV.8. Utilité de la notion de cône de couverture

- *Notion de condition :*
  Nous avons pu voir, lorsque les différents types d'interfaces bloc ont été décrits, que le système admettait des interfaces bloc bidirectionnelles, qui, lors du fonctionnement du circuit, devenaient entrées ou sorties de bloc suivant certaines conditions.

  Ces conditions seront pour le système des éléments de connaissances (à raison d'une condition par interface bloc bidirectionnelle), indiquant suivant une liste de valeurs d'interfaces bloc, le sens effectif (entrée ou sortie) de l'interface bloc bidirectionnelle en question.

IV.3.2.2 Hypothèses

Quatre hypothèses peuvent être retenues, dont chacune va induire une stratégie de recherche donnée. Ces hypothèses sont les suivantes :

- *faute simple combinatoire :*
  Il s'agit de l'hypothèse classique de faute simple, restreinte aux circuits ou parties de circuits entièrement combinatoires.

- *faute multiple combinatoire :*
  Cette hypothèse permet l'existence de plus d'une faute dans le circuit, mais comme pour le cas de faute simple, elle est restreinte aux circuits ou parties de circuits entièrement combinatoires.

- *faute cachée ("latente") :*
  Il s'agit de fautes apparaissant dans les circuits séquentiels en tant que réponse à un stimulus appliqué un certain nombre de tops d'horloge plus tôt : la faute a donc été "cachée" (non observée) pendant l'écoulement de ce temps.
  Cette terminologie ("fault latency") est donc utilisée ici pour définir une extension aux circuits séquentiels de la notion de "latence d'erreur" ("error latency"), désignant le temps s'écoulant entre l'occurrence d'une faute et sa détection, dans un circuit combinatoire [ShM75]. On trouvera également une généralisation de la notion d'erreur latente aux systèmes informatiques logiciels et matériels dans [Lap85].

- *faute sur les lignes :*
  Si cette hypothèse est émise, cela signifie que l'on considère les fautes sur les lignes physiques (connexions).
  Il faut examiner si cette hypothèse a lieu d'exister, ou si l'on doit toujours considérer qu'il peut y avoir des fautes sur les lignes (il s'agit uniquement d'un problème de temps de réponse du système, qui serait certainement bien augmenté si l'on devait examiner systématiquement toutes les connexions).

Remarque concernant les trois premières hypothèses :

L'intérêt de considérer trois hypothèses distinctes (faute simple ou multiple pour circuits combinatoires, et faute cachée pour les circuits séquentiels, avec les deux sous-hypothèses possibles, faute cachée simple et faute cachée multiple) est surtout de limiter la recherche des solutions dans certains cas, ce qui se traduira par un temps de réponse diminué.
En réalité il faudrait choisir parmi les différentes formules suivantes :

- Trois hypothèses distinctes, et pour chacune une stratégie particulière.

- Ne considérer que le cas de faute multiple (le cas faute simple apparaissant de lui-même si on ne détecte qu'une seule faute), mais distinguer deux stratégies, l'une pour

les parties combinatoires, l'autre pour les parties séquentielles. On aura donc deux hypothèses : fautes multiples cachées, fautes multiples non cachées.

- Ne considérer que des fautes cachées (les fautes non cachées étant considérées comme cachées pendant un délai nul), mais distinguer deux stratégies, l'une pour le cas faute simple, et l'autre pour le cas faute multiple. On aura donc également deux hypothèses : faute cachée simple, fautes cachées multiples.

- Ne pas émettre du tout d'hypothèse, le seul cas considéré étant : fautes multiples cachées (regroupement des formules précédentes). Une stratégie adéquate pourra alors détecter n'importe laquelle des situations possibles : faute simple non cachée, faute simple cachée, faute multiple non cachée, faute multiple cachée. Cette dernière formule est plus proche de ce qui se passe en pratique : en effet, la notion de faute simple n'est qu'une hypothèse émise pour la modélisation des fautes, leur classification et la définition de méthodes pour leur détection. De plus, il est rare de trouver des circuits uniquement combinatoires, et il est difficile d'isoler, par une méthode de partitionnement, des parties entièrement combinatoires dans des circuits quelconques. Néanmoins, nous verrons en conclusion de ce chapitre (cf. § IV.10) que les résultats obtenus à la suite des expérimentations menées justifient - en termes de temps d'exécution - le choix d'hypothèses distinctes.

IV.4 ORGANISATION DES CONNAISSANCES

Les modèles, propriétés et hypothèses décrits au § IV.3 vont former l'essentiel des bases de connaissances de PESTICIDE.
Ce terme "bases de connaissances" est hérité des systèmes de première génération (plus particulièrement des systèmes à règles de production). Néanmoins, dans le cas de PESTICIDE, une base de connaissances n'est pas une collection de connaissances associatives de la forme "symptômes observés → déductions possibles" [Lau82a], [Lau82b], mais plutôt un ensemble de spécifications de la structure du circuit, de ses fonctionnalités, et de son comportement.
Dans le but d'opérer une distinction claire entre ces différentes sortes de spécifications, trois bases de connaissances ont été définies : une base de connaissances structurelles, une base de connaissances fonctionnelles, et une base de connaissances comportementales.

Ce paragraphe se veut donc une récapitulation et une organisation de ce qui a été présenté précédemment. Les éléments de connaissances décrits dans la suite seront sous forme de clauses du langage PROLOG [CIM85], tels qu'ils ont été implémentés.

IV.4.1 Base de connaissances structurelles

La base de connaissances structurelles contient toutes les informations disponibles sur le circuit sous test. Ces données sont les suivantes :

- *Les blocs :*
  BLOCK(NAME, LIST_OF_BLOCK_INTERFACES).
  NAME : nom du bloc.
  LIST_OF_BLOCK_INTERFACES : liste des noms d'interfaces bloc (entrée, sortie, bidirectionnelle) qui connectent le bloc à son environnement.

- *Les interfaces bloc :*
  BLOCKINT(NAME, TYPE, DELAY, HOLD_TIME).
  NAME : nom de l'interface bloc.
  TYPE : entrée, sortie ou bidirectionnel.
  DELAY : délai, évalué comme décrit au paragraphe IV.3.1.2.
  HOLD_TIME : temps de maintien.

  Remarque :

  On notera que seuls les arguments "structurels" ou "fonctionnels" qualifiant l'interface bloc sont présents dans la base de connaissances. Les autres arguments (d'ordre comportemental) seront détaillés au paragraphe IV.4.3.

- *Les connexions :*
  WIRE(NAME, LIST_OF_WIRE_INTERFACES).
  NAME : nom de la connexion.
  LIST_OF_WIRE_INTERFACES : liste des noms d'interfaces bloc amont ou aval.

IV.4.2 Base de connaissances fonctionnelles

La base de connaissances fonctionnelles contient des connaissances additionnelles sur les interfaces bloc.[23] Ces données sont les suivantes :

- *Les conditions :*
  CONDITION(BINT, WAY, LIST_OF_CONDITIONS).
  BINT : nom de l'interface bloc sur laquelle porte la condition.
  WAY : sens (entrée ou sortie) de BINT si les conditions sont vérifiées.
  LIST_OF_CONDITIONS : liste des conditions.

- *Les cônes de couverture :*
  COVERAGE(BINT, LIST_OF_MEMBERS).

---

23. On pourra légitimement s'étonner du fait que cette base de connaissances ne décrive pas également la *fonction* des blocs composants le circuit. Des réponses à cette question seront apportées au chapitre V.

BINT : nom de la sortie sur laquelle agissent les entrées formant son cône de couverture.
LIST_OF_MEMBERS : liste des interfaces bloc d'entrée membres du cône de couverture.

IV.4.3 Base de connaissances comportementales

La base de connaissances comportementales contient les données disponibles sur le test en cours. Ces données sont les suivantes :

- *Les instances d'interfaces bloc :*
  INSTBI(NAME, STATE, VALUE, INSTANT, WAY).
  NAME : nom de l'interface bloc instanciée.
  STATE : état (erroné ou non erroné) de cette interface bloc.
  VALUE : valeur logique mesurée.
  INSTANT : instant de mesure.
  WAY : sens (entrée ou sortie) de l'interface bloc.

  Remarque :

  Comme nous l'avons dit précédemment, on retrouve dans la base de connaissances comportementales une partie des arguments des interfaces bloc. Ces arguments sont regroupés en instances des différentes interfaces bloc, à raison d'une instance par mesure effectuée (à des instants différents).

- *Les hypothèses :*
  HYPOTHESIS(HYP).
  HYP : hypothèse formulée, qui peut prendre l'une des valeurs suivantes :
  SIMPCOMB : faute simple "combinatoire".
  MULTCOMB : faute multiple "combinatoire".
  SIMPSEQ : faute simple "séquentielle".
  MULTSEQ : faute multiple "séquentielle".

  Remarque :
  Ainsi que mentionné aux paragraphes IV.2.3 et IV.3.2.2, cette notion d'hypothèse sur le type de test que l'on veut mener n'est pas une restriction du caractère automatique du diagnostic, mais est uniquement destinée à simplifier et accélérer le processus d'établissement de ce diagnostic. En l'absence d'hypothèse formulée, PESTICIDE prendra en compte le cas le plus général, c'est-à-dire le cas de faute multiple dans les circuits séquentiels.

IV.5 VERIFICATION DE LA COHERENCE DES INFORMATIONS

Sachant que PESTICIDE est alimenté par une très large masse d'informations (qu'il s'agisse d'informations structurelles et fonctionnelles, issues des spécifications, ou d'informations comportementales, issues de l'observation), et sachant également que ces informations sont très souvent corrélées, il est important de vérifier leur cohérence globale, comme préalable au processus de diagnostic, de façon à éviter la déduction de résultats erronés, obtenus à partir d'informations elles-mêmes erronées.
Pour ce faire, différents types d'incohérences doivent être définis, et chaque fois que les bases de connaissances sont modifiées (par ajout ou suppression d'information), il faut vérifier s'il y a occurrence d'une telle incohérence.

Cette notion de cohérence des informations nécessite d'être précisée, car elle peut être considérée sous deux formes, la première étant ce que nous appellerons "cohérence intrinsèque" (i.e. absence d'assertions contradictoires, non coexistence de valeurs différentes pour un attribut mono-valué, etc. [ACP88]), et la seconde forme regroupant ce que nous nommerons "cohérence relative au domaine".
Cette seconde forme de cohérence est spécialement intéressante car elle permet de s'affranchir de l'un des inconvénients des techniques de "raisonnement basé sur la structure" ("schematic-based reasoning") [Dav84], qui est de pré-supposer que le modèle du dispositif à étudier (ce modèle constituant la référence) est exact, alors qu'il peut en réalité comporter des erreurs.

La seule manière de vérifier l'exactitude de la description du circuit est d'identifier et prévoir les inexactitudes qui pourraient survenir, et de les exprimer sous la forme de connaissances associatives du type "si une inexactitude préalablement identifiée est décelée, alors inférer l'erreur correspondante dans la description" : il s'agit donc bien ici de règles de production.
Ceci, ajouté au fait que d'une part, la vérification de la cohérence des informations fait intervenir non seulement les données contenues dans des bases de connaissances distinctes, mais aussi les interactions entre ces différentes données, et que d'autre part, cette tâche de vérification/maintien de la cohérence n'est pas vraiment un processus de *raisonnement*, mais plutôt un problème de *détection*, permet de conclure que le module de vérification de la cohérence utilise des connaissances dites "de surface" ("shallow knowledge"), et souffre par conséquent d'inconvénients bien connus (cf. première partie), l'un d'entre eux étant l'incomplétude de l'ensemble identifié des inexactitudes pouvant survenir.

Malgré cette incomplétude,[24] somme toute inhérente aux connaissances associatives, un

---

24. On peut remarquer tout de même que la détection de ce que nous avons appelé les "incohérences relatives au domaine" n'est qu'une facette mineure de PESTICIDE. En effet, les incohérences de ce type sont le plus souvent détectées lorsque des langages de description de circuits (HDL : "Hardware Description Language") évolués comme VHDL [LSU89] ou le HDL du système HILO [Gen85] sont utilisés, leurs

certain nombre d'incohérences ont pu être identifiées, et sont listées dans la suite.

IV.5.1 Incohérences intrinsèques

- *Existence d'instances d'interfaces bloc déclarées erronées et non erronées au même instant de mesure :*
  Le traitement de ce cas repose donc sur un simple examen de toutes les instances d'interfaces bloc figurant dans la base de connaissances comportementales.

- *Existence de clauses "CONDITION" contradictoires :*
  Ce cas apparaît lorsque, pour une même interface bloc bidirectionnelle, la même liste de conditions vérifiées peut résulter en deux directionnalités différentes. Il est traité par examen exhaustif des clauses "CONDITION".

- *Déclaration d'hypothèses mutuellement exclusives :*
  Le traitement consiste, pour chaque hypothèse formulée par l'utilisateur, à générer automatiquement les nouvelles clauses qui traduisent le rejet des hypothèses contredisant l'hypothèse formulée.
  Pour que le système puisse effectuer ce traitement, il doit connaître, pour chaque hypothèse, la liste des hypothèses qui la contredisent.
  Ces informations sont fournies au système à l'aide du prédicat suivant, dans la base de connaissances comportementales :

  LIST(HYPOTHESIS, LIST_OF_HYPOTHESES).

  "HYPOTHESIS" est une hypothèse quelconque, "LIST_OF_HYPOTHESES" la liste des hypothèses dont la formulation est exclue.

  Exemple :

  Pour indiquer au système que l'hypothèse de faute simple combinatoire exclut les autres hypothèses, la clause atomique suivante doit être connue :

  LIST(SIMPCOMB, [SIMPSEQ, MULTCOMB,MULTSEQ]).

  A l'aide de cet axiome, le système peut générer, toujours dans la base de connaissances

---

compilateurs permettant des analyses syntaxiques et sémantiques assez poussées, notamment au niveau des contrôles de types.

comportementales, les clauses de rejet des autres hypothèses, par l'exécution du prédicat suivant :

NEWCLAUSE(NOTHYPOTHESIS(HYPOTHESIS)).

où "NOTHYPOTHESIS" est le prédicat de rejet à générer, et "HYPOTHESIS" prend la valeur de chacun des éléments de la liste d'hypothèses exclues.
Ce prétraitement effectué, le traitement d'un tel type d'incohérence consiste à détecter une incohérence chaque fois qu'une hypothèse "H" figure à la fois comme argument d'un axiome "HYPOTHESIS" et d'un axiome "NOTHYPOTHESIS".

IV.5.2 Incohérences relatives au domaine

- *Connexion déclarée comme admettant plus d'une interface bloc en amont :*
  La façon la plus efficace dans ce cas de distinguer les cas d'occurrence d'une fonction logique câblée (c'est-à-dire sans utiliser la porte logique de base correspondante, mais en court-circuitant simplement les connexions comme pour un "ET" logique, par exemple) d'un court-circuit réalisé indépendamment de la volonté du concepteur du circuit sous test reste de faire appel de façon interactive à l'utilisateur en lui posant directement la question.
  Le traitement consiste donc à examiner chaque connexion pour relever la liste de ses interfaces bloc, et de vérifier que dans cette liste ne figure qu'un seul élément de type unidirectionnel de sortie (on rappelle que le type d'une interface bloc est établi par rapport à un bloc, et qu'une sortie de bloc est forcément en amont d'une connexion).

- *Déclaration d'un bloc sans sortie :*
  Le traitement revient à examiner, pour chaque bloc déclaré, la liste de ses interfaces d'entrée/sortie, et à vérifier que cette liste contient au moins une interface bloc de type unidirectionnel de sortie, ou de type bidirectionnel, et dans ce dernier cas, à vérifier également qu'il existe au moins une condition pour laquelle cette interface bloc devient une sortie.

- *Incohérences au niveau de la directionnalité des interfaces bloc par rapport aux connexions :*
  Dans ce cas, il faut vérifier que, dans la liste des interfaces bloc amont et aval d'une connexion, il existe toujours au moins un élément amont et un élément aval.
  Le traitement de ce type d'incohérence relève du même principe que celui du précédent.

- *Déclaration d'une interface bloc non reliée à au moins une connexion :*
  Pour relever les occurrences de ce type d'incohérence, le système doit vérifier, pour chaque interface bloc déclarée dans la base de connaissances structurelles, que cette interface bloc est membre de la liste d'interfaces d'au moins une connexion.

- *Déclaration d'une interface bloc non reliée à au moins un bloc :*
  Ce type d'incohérence est exactement le dual du type précédent, et est donc traité de la même façon.

- *Déclaration d'une sortie de bloc de cône de couverture non existant :*
  Le système doit donc vérifier qu'il existe bien une clause "COVERAGE" pour chaque sortie, admettant au moins un membre du cône de couverture.

- *Déclaration d'une entrée de bloc n'agissant sur aucune sortie :*
  Ce type d'incohérence est le dual du précédent, et est traité de la même façon.

- *Déclaration d'une interface bloc bidirectionnelle non munie d'une condition sur le sens effectif qu'elle peut prendre :*
  Ce cas est traité par vérification de la présence dans la base de connaissances fonctionnelles d'une clause "CONDITION" pour chacune de ces interfaces bloc.

IV.5.3 Evaluation dans le cadre des recherches en vérification de la cohérence

La détection des incohérences - qu'elles soient intrinsèques ou relatives au domaine - est mise en oeuvre au moyen d'un paquet de clauses (ensemble de clauses ayant le même littéral de tête, et dont le corps correspond, pour chaque élément du paquet, à la recherche d'une incohérence particulière parmi celles citées précédemment).
La recherche de l'occurrence de chaque type d'incohérence revient à l'implémentation d'une règle de production en langage PROLOG, et le processus de vérification de la cohérence des informations est considéré comme achevé lorsqu'il y a saturation de la base de règles ainsi définie.

On notera qu'il s'agit uniquement d'assurer la cohérence *statique* des informations (suivant la terminologie définie par [Lau88]), puisque seules les informations d'ordre factuel (les axiomes) sont vérifiées dans notre cas.
D'autre part, comme nous l'avons déjà précisé, identifier des types d'incohérence pouvant survenir et les détecter dans les informations signifie uniquement que ces informations seront cohérentes *par rapport* aux types identifiés.

Ces considérations rejoignent celles présentées par [ACP88] au cours d'une synthèse sur les travaux concernant l'étude de la cohérence dans les bases de connaissances, menés dans le cadre du Programme de Recherche Concertée en Intelligence Artificielle (PRC-IA). En effet, les problèmes fondamentaux identifiés dans [ACP88] sont formalisés comme suit :[25]

---

25. Les systèmes étudiés dans le cadre du PRC-IA sont des systèmes experts "classiques", composés d'une base de faits formée de couples (attribut, valeur) ou de triplets (objet, attribut, valeur), d'une base de règles de production et d'un moteur d'inférences.

Les contraintes à respecter pour que la base de faits (BF) soit cohérente sont supposées être synthétisables sous la forme d'un unique prédicat CF. Ainsi, une base de faits BF est dite CF-cohérente si et seulement si CF(BF).
La base de faits BF est dite validée si un expert juge cette base réaliste. Si BF est validée, alors on a CF(BF) [ACP88], mais la réciproque n'est pas forcément vraie, ou alors CF serait une définition de la validité, et serait complet.

Ceci étant posé, [ACP88] énonce les problèmes à résoudre pour vérifier la cohérence *dynamique* des systèmes à base de connaissances, c'est-à-dire vérifier que tous les enchaînements possibles des règles existant dans la base de règles du système ne génèrent pas d'anomalies dans la base de faits. Autrement dit [ACP88] :

Soient BR la base de règles et BR.BF la base de faits déduite de BF par application de BR, il s'agit de montrer :

Quelque soit BF, si CF(BF) alors CF(BR.BF)

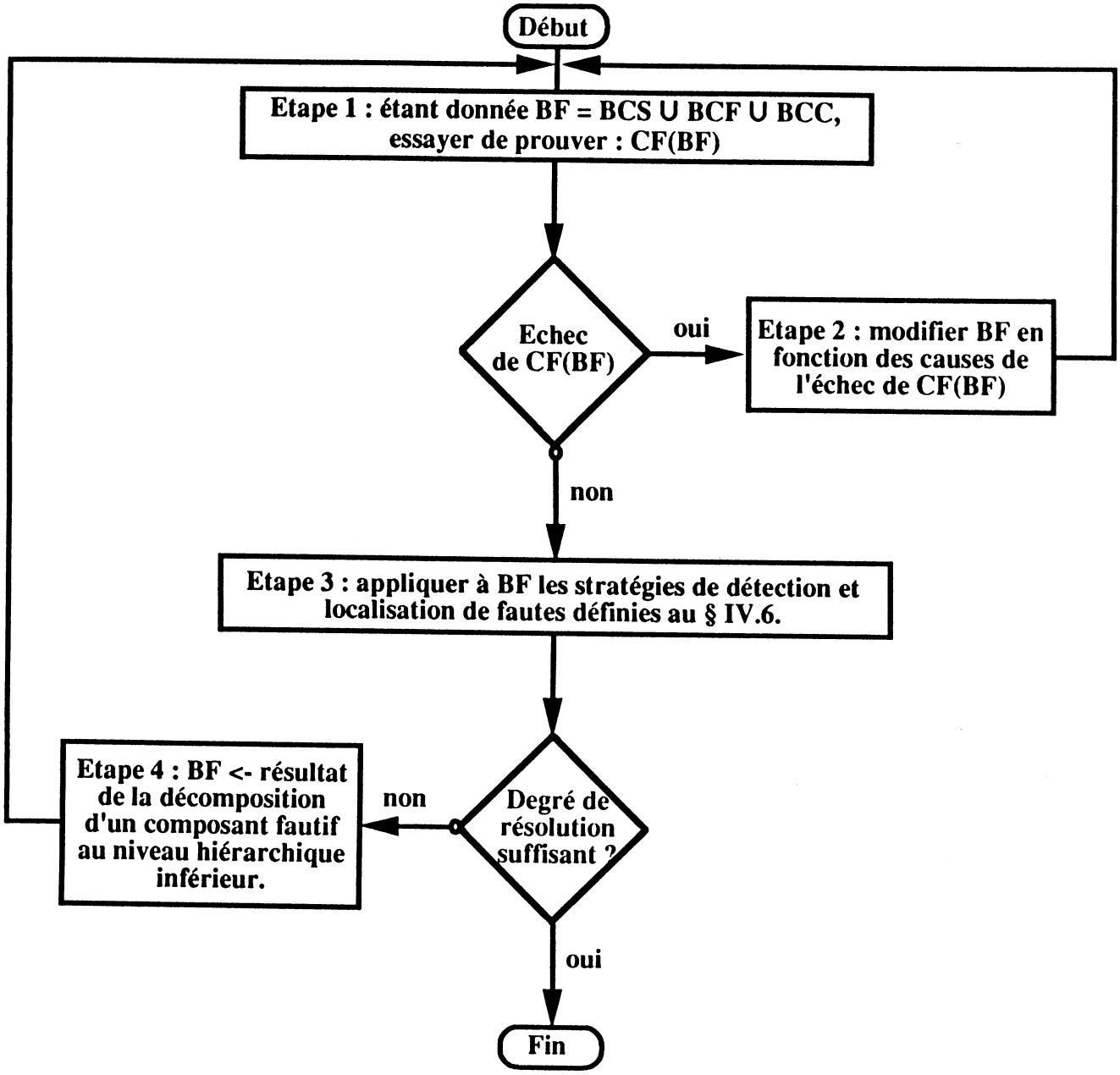
Ce qui revient à montrer que, pour toute BF CF-cohérente, BR est également CF-cohérente.

Ce qui nous intéresse pour notre part, dans le cadre d'un système à base de connaissances de seconde génération, est de vérifier CF(BF), c'est-à-dire de vérifier le modèle représentant le dispositif à étudier.
Le prédicat CF dans notre cas est représenté par le littéral de tête du paquet de clauses permettant la détection des incohérences intrinsèques ou relatives au domaine.

Dans l'approche de [ACP88], la CF-cohérence est conditionnée par la validation d'un expert n'ayant détecté aucune anomalie dans la BF ; dans notre propre approche, la propriété de CF-cohérence est conditionnée par la vérification automatique qu'un ensemble d'anomalies (pouvant être complété à tout moment si nécessaire) n'est pas présent dans la BF.
Il s'agit dans les deux cas d'une approche expérimentale, dont la complétude et la validité formelles ne peuvent être prouvées dans l'état actuel des recherches [ACP88].

IV.5.4 Mise en oeuvre du processus dans PESTICIDE

Le processus de vérification de la cohérence des informations contenues dans les bases de connaissances est mis en oeuvre dans PESTICIDE comme une étape de l'approche mixte décrite dans la première partie du document (chapitre II, § II.5). Cette mise en oeuvre est illustrée en figure IV.9.



Avec :

BF : Base de Faits

BCS : Base de Connaissances Structurelles

BCF : Base de Connaissances Fonctionnelles

BCC : Base de Connaissances Comportementales

CF : Littéral de tête du paquet de clauses permettant la vérification de la cohérence

Figure IV.9. Mise en œuvre de l'approche mixte dans PESTICIDE

IV.6 DETECTION ET LOCALISATION DE FAUTES

Les processus de détection et de localisation de fautes à travers le circuit sous test constituent les deux tâches principales du système en vue de l'établissement de son diagnostic. On distingue les cas de recherche de *fautes sur les connexions* et de *fautes dans les blocs* car, comme il sera montré dans la suite, il s'agit dans le premier cas d'un problème de *détection* de fautes, alors que dans le deuxième cas on s'intéresse à un problème de *localisation* de fautes.
Avant de présenter le détail des stratégies adoptées dans ces deux cas, nous décrirons la méthode mise en place pour évaluer la validité d'un instant de mesure d'une interface bloc quelconque, et ceci dans le but de faciliter la compréhension de la suite de ce paragraphe.

IV.6.1 Evaluation de la validité d'un instant de mesure

Comme il en a déjà été fait mention, la déclaration d'une instance d'interface bloc comprend comme paramètre l'instant de mesure de la valeur logique au niveau de cette interface bloc, et la déclaration de l'interface bloc elle-même contient le délai et le temps de maintien de la valeur de cette interface bloc. Rappelons également que ce délai a différentes significations suivant le type de l'interface bloc considérée. Cet instant de mesure ne peut évidemment être pris en compte que s'il s'agit du "bon" instant.[26]
Pour déterminer la validité de cet instant, le système doit donc l'évaluer et utilise pour cela la stratégie décrite ci-après :

IV.6.1.1 Examen de l'état d'un bloc

Il s'agit donc ici de la relation entre entrées et sorties d'un même bloc.
Soient :
T(S) l'instant de mesure de la sortie S considérée
T(E) l'instant de mesure de l'entrée E considérée
D(S) le délai au niveau de la sortie S
HT(E) le temps de maintien ("hold time") au niveau de la sortie S
C(S) le cône de couverture de la sortie S.
Pour que l'instant T(S) soit valide, les inéquations suivantes doivent être vérifiées :

---

26. Nous appelons "bon" instant l'instant à partir duquel (pour les circuits combinatoires) ou le laps de temps pendant lequel (pour les circuits séquentiels) l'on dispose de l'état logique que l'on veut mesurer ou observer.

$$(I_1) : T(S) \geq T(E) + D(S)$$

$$(I_2) : T(S) \leq T(E) + D(S) + HT(E)$$

Ceci est valable lorsque C(S) = {E}.
Si maintenant C(S) = $\{E_i\}$, pour $i_0 \leq i \leq i_n$, T(S) doit vérifier les inéquations suivantes :

$$(I_1') : T(S) \geq MAX_{i_0 \leq i \leq i_n}\ T(E_i) + D(S)$$

$$(I_2') : T(S) \leq MAX_{i_0 \leq i \leq i_n}\ T(E_i) + D(S) + MIN_{i_0 \leq i \leq i_n}\ HT(E_i)$$

Au niveau de l'implémentation en PROLOG, l'évaluation de la validité d'un instant de mesure revient à prouver la clause suivante :

VALID(LISTIN, LISTINST, BINTOUT, INSTOUT).

où LISTIN est la liste des membres du cône de couverture de la sortie
LISTINST est la liste des instants de mesure de ces entrées
BINTOUT est l'interface bloc de sortie du bloc
INSTOUT est l'instant de mesure de BINTOUT.

Pour prouver cette clause, il faut avoir au préalable collecté les éléments de LISTIN et LISTINST.
Le corps de la clause consiste à chercher dans la base de connaissances structurelles le délai de la sortie, à collecter le temps de maintien de chaque entrée membre de son cône de couverture, et à appliquer la formule présentée ci-dessus.

En PROLOG, cette clause s'écrit donc :

```
VALID(LISTIN, LISTINST, BINTOUT, INSTOUT):-
                BLOCKINT(BINTOUT,_,DELAIOUT,_),
                COLLECT(LISTIN,[],LISTTEMPS),
                MAXIMUM(MAX,LISTINST),
                MINIMUM(MIN,LISTTEMPS),
                BORNINF is MAX + DELAIOUT,
                BORNSUP is BORNINF + MIN,
                INSTOUT >= BORNINF,
                INSTOUT =< BORNSUP.
```

COLLECT, MAXIMUM et MINIMUM sont des prédicats définis ailleurs dans le programme.
LISTTEMPS est la liste des temps de maintien des entrées.
MAX et MIN sont des variables "muettes".
BORNINF et BORNSUP sont les membres droits respectifs des deux inégalités précédemment mentionnées.

Cette clause réussit si l'instant est valide, échoue sinon.

IV.6.1.2 Examen de l'état d'une connexion

Il s'agit dans ce cas de la relation entre interfaces bloc en amont et en aval d'une connexion, et on doit avoir les mêmes inégalités qu'en IV.6.1.1 (I_1 et I_2, avec un seul membre du cône de couverture), en remplaçant toutefois : "entrée" par "interface bloc amont", et "sortie" par "interface bloc aval", le délai au niveau de la sortie devenant implicitement le temps d'établissement du signal sur la connexion concernée.
Au niveau de l'implémentation en PROLOG, l'évaluation de la validité d'un instant de mesure revient à prouver la même clause que précédemment, à la différence près que dans ce cas, on n'a pas une liste de membres du cône de couverture, mais un seul élément de référence, l'interface bloc amont de la connexion, tandis que l'interface bloc aval joue le même rôle que l'interface bloc de sortie pour la clause "VALID".

IV.6.2 Détection de fautes sur les connexions

Outre la détection de court-circuits, permise au cours de la vérification de la cohérence des informations par le fait qu'une connexion a plus d'une interface bloc en amont, sans que cela soit voulu par le concepteur du circuit sous test, le système doit pouvoir effectuer la détection d'autres fautes sur les connexions, principalement les coupures (cut), ou l'occurrence d'un signal insuffisamment amplifié.
Pour ce faire, la stratégie utilisée est la suivante :

Si toutes les interfaces bloc reliées à la connexion examinée n'ont pas la même valeur

logique, alors il y a faute sur cette connexion.

Remarque :

Lorsqu'il est fait mention de valeur logique, celle-ci s'entend comme la "bonne" valeur pour les interfaces bloc en aval de la connexion incriminée, c'est-à-dire la valeur mesurée au bon instant, à savoir après écoulement du délai d'établissement du signal à travers la connexion.

Ce processus de détection de faute sur une connexion est illustré dans la suite sur l'exemple présenté en figure IV.10 [MaC88].

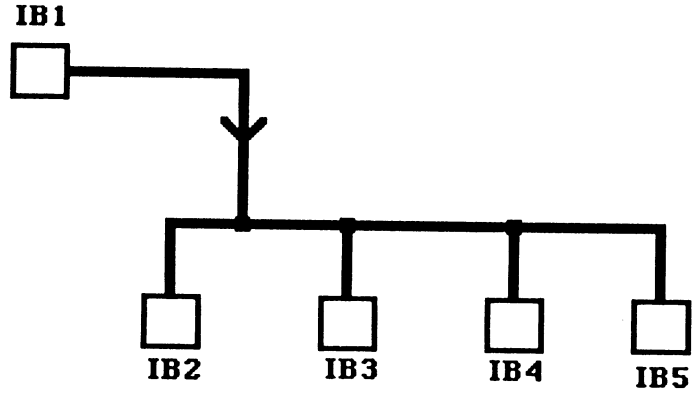


Figure IV.10. Détection de faute sur une connexion

Dans cet exemple, l'interface bloc IB_1 est connectée en amont de la connexion, alors que les interfaces bloc IB_2 à IB_5 lui sont connectées en aval. Si D(IB) est le délai de l'interface bloc IB, et si l'expression VAL(IB / T) représente la valeur logique de l'interface bloc IB à l'instant T, la connexion de la figure IV.10 est correcte si et seulement si les égalités suivantes (E_{12}) à (E_{15}) sont vérifiées :

$$(E_{12}) : VAL(IB_1 / T_1) = VAL(IB_2 / T_1 + D(IB_2))$$

...

$$(E_{15}) : VAL(IB_1 / T_1) = VAL(IB_5 / T_1 + D(IB_5))$$

IV.6.3 Localisation de fautes à travers le circuit

Nous nous en tiendrons, pour ce module de localisation, au choix fait au paragraphe IV.3.2.2, c'est-à-dire à la définition de quatre stratégies distinctes : faute simple "combinatoire", faute multiple "combinatoire", faute simple "séquentielle", et faute multiple

"séquentielle".
Avant de présenter ces stratégies, il est nécessaire de formuler certaines remarques :

Remarque 1 :

Lorsqu'il est question de faute (simple ou multiple) "séquentielle", il s'agit bien de faute dans un circuit séquentiel, et non pas de faute telle qu'un circuit combinatoire, lorsqu'il en est entâché, présente un comportement séquentiel (mémorisation indésirable d'état logique) [Wad78].

Remarque 2 :

La détection de faute nécessitant de faire fonctionner le circuit sous test (même s'il s'agit d'un test "hors-ligne" dans notre cas, c'est-à-dire que le circuit ne fonctionne pas dans son environnement réel), il ne sera plus fait mention d'interfaces bloc bidirectionnelles dans ce paragraphe. En effet, toutes les interfaces bloc prendront une valeur au cours du fonctionnement du circuit, et en particulier les interfaces bloc faisant office de condition pour décider de la directionnalité d'autres interfaces bloc.
Dans la suite de ce paragraphe, on considèrera donc les instances d'interfaces bloc plutôt que les interfaces bloc elles-mêmes, étant entendu que l'argument "WAY" (cf. paragraphe IV.4.3) d'une instance d'interface bloc, c'est-à-dire le paramètre indiquant s'il s'agit d'une entrée ou d'une sortie, aura pris sa valeur effective par application des clauses "CONDITION" (cf. paragraphe IV.4.2).

Remarque 3 :

On aura pu s'étonner d'apprendre que les circuits séquentiels sont considérés, sans pour autant qu'il soit fait mention du traitement des boucles de rétroaction ("feedback loops").
Ceci s'explique par l'existence des différentes instances d'interfaces bloc dans la base de connaissances comportementales, qui vont permettre de tester le circuit dans un fonctionnement en mode "pas à pas". On dispose donc à tout moment, grâce à l'utilisation de la stroboscopie de faisceau pour l'observation, d'un état instantané du circuit, et il est par conséquent inutile de différencier les entrées d'un bloc séquentiel en entrées externes d'une part, et en états internes d'autre part [BeH82].

Remarque 4 :

Lorsqu'il n'est question que de circuits combinatoires, on ne tient pas compte des données suivantes :

- Délai et temps de maintien, dans la déclaration des interfaces bloc.
- Instant de mesure, dans la déclaration des instances d'interfaces bloc.

En effet, on considèrera dans ce cas que le délai est nul, que le temps de maintien est infini, et donc que l'instant de mesure importe peu (rappelons que seuls les états stables sont observés dans le cas des circuits combinatoires).

Ces remarques étant faites, examinons à présent les différentes stratégies possibles de localisation des blocs défaillants.

IV.6.3.1 Cas de faute simple dans les circuits combinatoires

Il s'agit de la stratégie suivie par le système lorsque l'hypothèse "SIMPCOMB" est formulée dans la base de connaissances comportementales.
Cette stratégie se résume par les étapes suivantes :

☞ Collecter toutes les interfaces bloc dont l'état est déclaré erroné dans la base de connaissances comportementales.

Puis, pour chacune de ces interfaces :

☞ Examiner le bloc dont elle est une sortie.

☞ Si aucun des membres du cône de couverture de cette interface bloc n'est erroné alors :

  ☜ Ce bloc est défaillant et manifeste sur la sortie concernée une erreur directe.

☞ Si l'un au moins des membres du cône de couverture de cette interface bloc est erroné alors :

  ☜ Remonter la chaîne des blocs en amont à travers les connexions, jusqu'aux plots du circuit si nécessaire, pour trouver le bloc défaillant manifestant une erreur propagée sur l'interface bloc en cours de traitement.

L'application de cette stratégie résulte, en fin de processus, en l'assertion d'autant de résultats qu'il y a d'interfaces bloc erronées. Ceci ne signifie pas que plus d'une faute a été localisée, mais que le système fournit une raison à *chaque manifestation observée de cette faute* : cette raison est le chemin de propagation du siège de la faute vers chaque erreur.

On peut voir un exemple d'application de cette stratégie en figure IV.11. Dans cet exemple, les interfaces bloc en erreur sont IB_{23}, IB_{32}, IB_{33} et IB_{03} (en noir sur le schéma).

Le siège de la faute est le bloc B_2, les erreurs manifestées sont de type direct pour IB_{23} et de type propagé pour IB_{33}, la propagation s'étant faite à travers le bloc B_3. Les deux autres interfaces bloc (IB_{32} et IB_{03}) sont également en erreur car les valeurs erronées se sont propagées à travers les connexions W_4 (reliant IB_{23} à IB_{32}) et W_5 (reliant IB_{33} à IB_{03}, sortie primaire du circuit exemple).

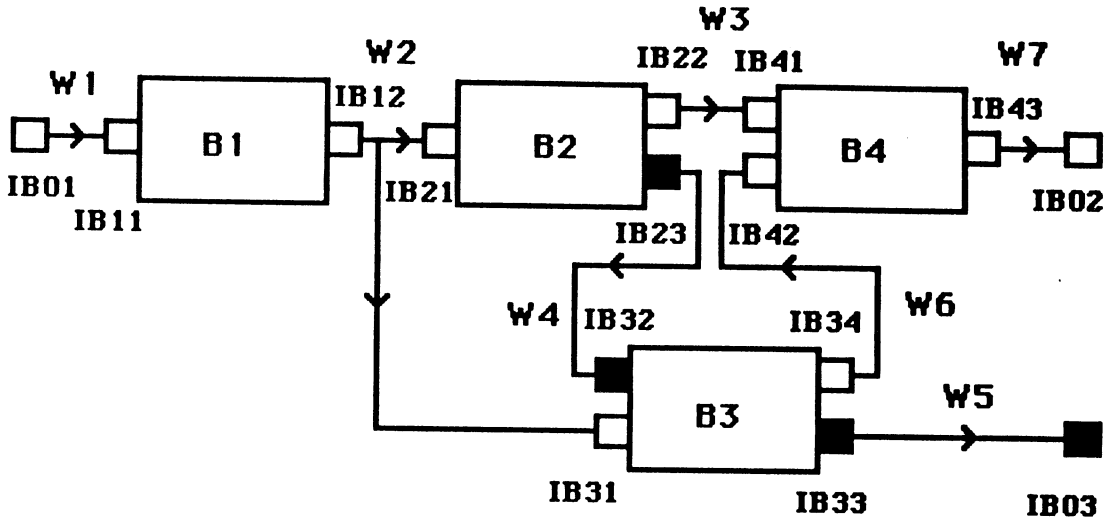


Figure IV.11. Localisation de faute : faute simple, circuit combinatoire

IV.6.3.2 Cas de faute multiple dans les circuits combinatoires

Cette stratégie est sollicitée par la formulation de l'hypothèse "MULTCOMB" dans la base de connaissances comportementales.
Elle consiste à appliquer les étapes suivantes :

- Recherche de tous les blocs défaillants (par application de la stratégie "faute simple combinatoire").

- Pour chacun de ces blocs, collecter les interfaces bloc de sortie dont l'état est erroné.

Pour chacune de ces interfaces bloc :

- Considérer les interfaces bloc qui lui sont reliées, celles-ci constituant des entrées de blocs en aval.

Pour chacun de ces blocs :

- S'il n'est pas déjà classé comme étant défaillant, alors :

  - Déclarer ce bloc comme présumé défaillant.

- Parcourir à partir de ce bloc la chaîne des blocs en aval en effectuant le même traitement.

Certains blocs sont déclarés "présumés défaillants" car, à cette étape du processus de localisation, le système n'a pas encore les informations suffisantes pour trancher : ces blocs, ayant tous une ou plusieurs entrées erronées, peuvent être effectivement eux-mêmes défaillants, comme ils peuvent être tout à fait corrects, mais propager une défaillance. A ce

stade, le système se contente donc de conjectures, et on verra au paragraphe IV.8 par quels moyens ces conjectures pourront être confirmées ou infirmées.

La figure IV.12 donne un exemple d'application de cette stratégie. Dans cet exemple, les interfaces bloc en erreur sont IB_{23}, IB_{32}, IB_{34} et IB_{41} (en noir sur le schéma). La recherche de tous les blocs défaillants par application de la stratégie "faute simple combinatoire" donne comme unique résultat le bloc B_2. Une première propagation aval à partir de l'interface bloc erronée en sortie de B_2 permet de générer B_3 comme candidat à la défaillance ("fault candidate"). Dans une seconde étape (propagation à partir de l'interface bloc erronée IB_{34} en sortie de B_3), B_4 sera également présumé défaillant. La recherche des candidats à la défaillance s'arrête alors puisque d'une part aucune sortie de B_4 n'est en erreur, et d'autre part toutes les sorties erronées de B_2 et B_3 ont été considérées.

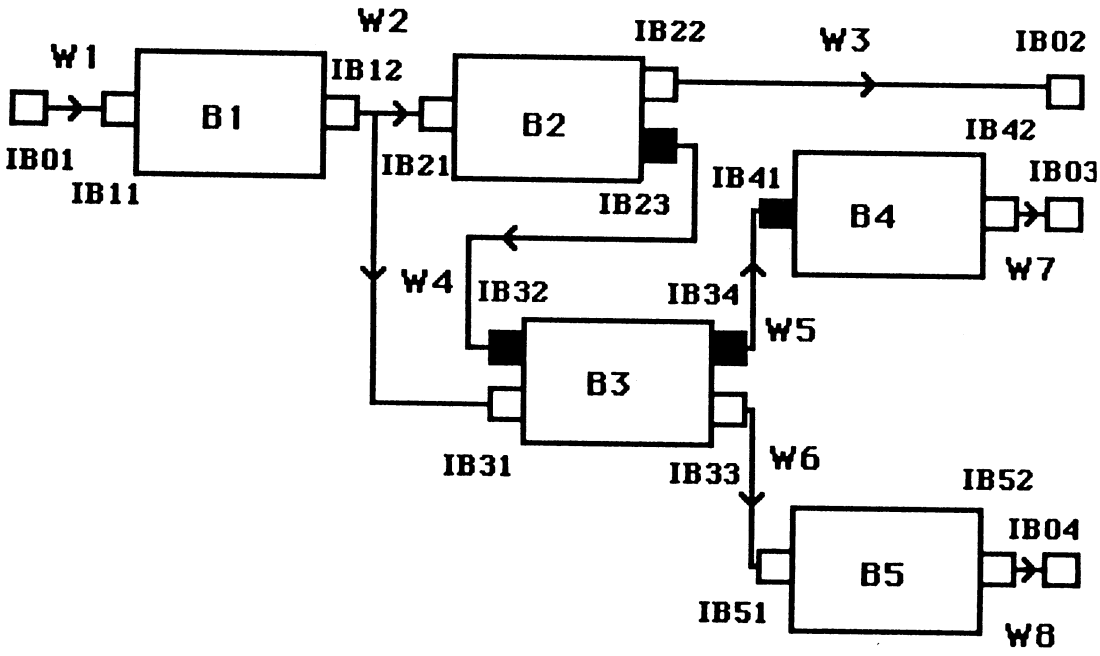


Figure IV.12. Localisation de faute : faute multiple, circuit combinatoire

IV.6.3.3 Cas des circuits séquentiels

Dans le cas des circuits séquentiels, l'hypothèse formulée dans la base de connaissances comportementales est soit "SIMPSEQ" (faute simple), soit "MULTSEQ" (faute multiple). La stratégie suivie alors par le programme de contrôle est exactement la même que dans le cas des circuits combinatoires, que ce soit pour des fautes simples ou des fautes multiples, à la différence que, chaque fois qu'il y a examen de l'état d'une interface bloc, il s'agit de l'état d'une *instance* d'interface bloc, observé au "bon" instant. Les clauses d'évaluation de la validité d'un instant de mesure (cf. § IV.6.1) sont donc utilisées à chaque étape de la stratégie.

Pour ce faire, il s'agit de déterminer, pour chaque instance d'interface bloc de sortie d'un bloc donné, les instances d'interface bloc, membre de son cône de couverture, qui valident cette instance de sortie.
Ce processus se traduit sur l'exemple de la figure IV.13 comme suit :

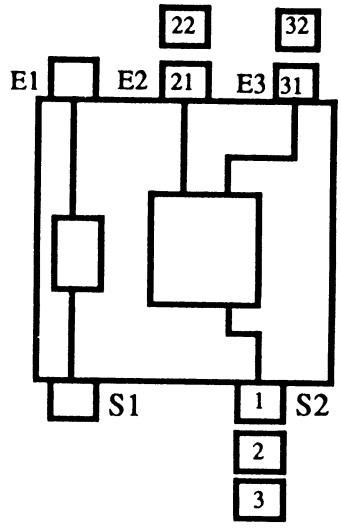


Figure IV.13. Recherche d'instances valides d'interface bloc

Supposons que l'on veuille examiner l'interface bloc de sortie S2, dont trois instances (1, 2 et 3) ont été observées, parmi lesquelles on veut valider l'instance 2.
Les membres du cône de couverture de S2 sont les entrées E2 (instances 21 et 22) et E3 (instances 31 et 32).
On commence par établir la liste des combinaisons d'instances possibles ((21,31), (21,32), (22,31), (22,32)).
Puis, pour chaque combinaison d'instances, on essaie de vérifier les clauses d'évaluation des instants de mesure (correspondant, dans le cas de la figure IV.13, aux inéquations I_1' et I_2' définies au § IV.6.1.1), relativement à l'instance de sortie 2.
En fin de processus, on ne retient que les combinaisons valides d'instances d'entrée, membres du cône de couverture de la sortie S2. S'il n'existe aucune combinaison valide, PESTICIDE demande d'effectuer une observation supplémentaire, en fournissant l'instant auquel elle doit avoir lieu.
Des opérations similaires sont effectuées pour les instances d'interfaces bloc amont et aval des connexions du circuit.

IV.7 IMPLEMENTATION ET FONCTIONNEMENT

Dans ce paragraphe, nous essaierons d'expliciter les stratégies présentées ci-dessus. Nous nous contenterons des stratégies en cas de faute simple et faute multiple pour les

circuits combinatoires. Les stratégies suivies lorsque les hypothèses de faute simple et de faute multiple dans les circuits séquentiels sont formulées ont été également implémentées. On en trouvera le détail et des exemples d'exécution dans [Ben90].
Etant donné qu'il s'agit de présenter ici la façon dont nous avons implémenté ces stratégies, on trouvera plusieurs clauses PROLOG dans ce paragraphe. Toutefois, dans un souci de clarté, ces clauses seront données sous une forme "semi-exécutable", i.e. la syntaxe du langage ne sera pas toujours respectée à la lettre.
Dans la suite, l'utilisation du caractère "souligné" ("_") à la place d'un argument d'un sous-but donné signifiera que cet argument peut être unifié à une valeur quelconque, ou même qu'il peut ne pas être unifié du tout (notion de variable libre).
De plus, afin de distinguer les variables des valeurs qu'elles peuvent prendre, les caractères minuscules seront utilisés pour les variables, alors que les majuscules représenteront les valeurs.

IV.7.1 Cas de faute simple

On distinguera les cas d'erreurs directes et propagées.

IV.7.1.1 Faute manifestant des erreurs directes

Ce type de faute est localisé à l'aide de la clause suivante :

```
DIRECTFAULT(b,ib):-
        HYPOTHESIS(SINGCOMB),
        INSTBI(ib,ERRONEOUS,_,_,OUTPUT),
        BLOCK(b,[ · · · ,ib, · · · ]),
        NOTERRORIN(b).
```

Dans cette clause, ib est l'interface bloc erronée courante, et constitue l'une des sorties du bloc b. Ceci est vérifié par les trois premiers sous-buts.
Ensuite, le prédicat nommé NOTERRORIN vérifie qu'aucun des membres du cône de couverture de la sortie ib n'est erroné.
Le but DIRECTFAULT réussit si le but NOTERRORIN réussit.

L'exemple de la figure IV.11 donne, pour la clause DIRECTFAULT, la trace d'exécution synthétisée en figure IV.14. Le résultat de cette exécution est donc l'instanciation DIRECTFAULT(b_2,ib_{23}), donnant B_2 comme bloc défaillant et une erreur de type direct sur IB_{23}.

IV.7.1.2 Faute manifestant des erreurs propagées

Ce type de faute est localisé à l'aide de la clause suivante :

prédicat	réponse	variable	valeur
DIRECTFAULT(b,ib)	?	b	?
		ib	?
HYPOTHESIS(SINGCOMB)	YES	--	--
INSTBI(ib,ERRONEOUS,_,_,OUTPUT)	YES	ib	IB_{23}
BLOCK(b,[· · · ,ib, · · ·])	YES	b	B_2
		ib	IB_{23}
NOTERRORIN(b)	YES	b	B_2

Figure IV.14. Trace d'exécution de la clause DIRECTFAULT

PROPAGFAULT(b,ib_1):-
HYPOTHESIS(SINGCOMB),
INSTBI(ib_1,ERRONEOUS,_,_,OUTPUT),
BLOCK(b,[· · · ,ib_1, · · ·]),
ERRORIN(b),
CONNECTED(upstream,b,ib_2),
SUITPROPAG(upstream,ib_2,ib_1).

Les trois premiers sous-buts sont identiques à ceux de la clause DIRECTFAULT, et jouent le même rôle. Si ces prédicats réussissent, alors le sous-but nommé ERRORIN vérifie qu'il existe au moins une interface bloc erronée parmi les membres du cône de couverture de ib_1. Ensuite, le prédicat CONNECTED cherche le bloc nommé upstream, connecté à b directement en amont, dont la sortie concernée (i.e. par laquelle se fait la connexion à b) est IB_2.
Après cela, l'interpréteur PROLOG essaie de prouver le prédicat SUITPROPAG, défini récursivement par le paquet de clauses suivant :

SUITPROPAG(upstream,ib_2,ib_1):-
INSTBI(ib_2,ERRONEOUS,_,_,INPUT),
NOTERRORIN(upstream),
!.

SUITPROPAG(upstream,ib_2,ib_1):-
CONNECTED($upstream_2$,upstream,ib),
SUITPROPAG($upstream_2$,ib,ib_1).

La première clause du paquet est une condition d'arrêt, vraie si le bloc upstream n'a pas d'entrée erronée. Dans ce cas, upstream sera le bloc défaillant, responsable de l'erreur propagée jusqu'à l'interface bloc ib_1.
Si cette condition d'arrêt n'est pas vérifiée, et uniquement dans ce cas - ceci est assuré par la coupure ("cut" ou "!") de PROLOG -, l'interpréteur PROLOG recherche le bloc $upstream_2$, connecté directement en amont de upstream par l'interface bloc ib, et la clause SUITPROPAG s'exécute récursivement sur $upstream_2$, comme elle s'est exécutée sur upstream.

L'exemple de la figure IV.11 donne, pour la clause PROPAGFAULT, la trace d'exécution synthétisée en figure IV.15.

prédicat	réponse	variable	valeur
PROPAGFAULT(b,ib_1)	?	b	?
		ib_1	?
HYPOTHESIS(SINGCOMB)	YES	--	--
INSTBI(ib_1,ERRONEOUS,_,_,OUTPUT)	YES	ib_1	IB_{33}
BLOCK(b,[$\cdots$,ib_1, $\cdots$])	YES	b	B_3
		ib_1	IB_{33}
ERRORIN(b)	YES	--	--
CONNECTED(upstream,b,ib_2)	YES	upstream	B_2
		b	B_3
		ib_2	IB_{32}
SUITPROPAG(upstream,ib_2,ib_1)	YES	upstream	B_2
		ib_2	IB_{32}
		ib_1	IB_{33}
INSTBI(ib_2,ERRONEOUS,_,_,INPUT)	YES	ib_2	IB_{32}
NOTERRORIN(upstream)	YES	upstream	B_2
!	STOP	--	--

Figure IV.15. Trace d'exécution de la clause PROPAGFAULT

Le résultat de cette exécution donne les instanciations suivantes :

SUITPROPAG(B_2,IB_{32},IB_{33}) et PROPAGFAULT(B_3,IB_{33}), ce qui permet de conclure que B_2 est défaillant, et manifeste l'erreur sur IB_{33}, propagée à travers B_3.

IV.7.2 Cas de faute multiple

Les clauses PROLOG suivantes donnent quelques indications sur l'implémentation de cette stratégie :
D'abord, une clause DIRECTFAULT permet de détecter le ou les blocs défaillants. Par rapport à la clause de même nom définie en IV.7.1.1, seule l'hypothèse de travail (MULTCOMB au lieu de SINGCOMB) diffère.
Puis l'interpréteur PROLOG essaie de vérifier la clause POSSIBLEFAULT pour chacun des blocs défaillants b trouvé, et pour chacune de ses sorties erronées ib_1 trouvée :

```
POSSIBLEFAULT(b,ib1):-
        CONNECTED(b,downstream,ib1),
        QUALIFY(downstream),
        INSTBI(ib2,ERRONEOUS,_,_,OUTPUT),
        BLOCK(downstream,[ ··· ,ib2, ··· ]),
        POSSIBLEFAULT(downstream,ib2).
```

b est donc le bloc défaillant courant et ib_1 sa sortie erronée en cours d'examen. Le prédicat CONNECTED cherche le bloc downstream, connecté directement en aval de b via la sortie ib_1.
Le rôle du prédicat QUALIFY est de vérifier que downstream n'est pas déjà déclaré défaillant, et de le déclarer comme présumé défaillant.
Ensuite, l'interpréteur PROLOG recherche une sortie erronée de downstream, et, s'il en existe, réapplique à ce bloc la clause POSSIBLEFAULT, de façon récursive.
Une propagation aval donnée est stoppée lorsque le bloc présumé défaillant courant n'a plus de sortie erronée.

L'exemple de la figure IV.12 donne, pour la clause POSSIBLEFAULT, la trace d'exécution synthétisée en figure IV.16. Le résultat de cette exécution donne les instanciations suivantes : POSSIBLEFAULT(B_2,IB_{23}), QUALIFY(B_3) et QUALIFY(B_4). L'exécution s'arrête alors car le bloc B_4 n'a aucune sortie erronée. La conclusion est que B_2 est défaillant, alors que B_3 et B_4 sont des candidats à la défaillance.

IV.8 TRAITEMENTS SPECIFIQUES APRES LOCALISATION

Cette étape se place après l'étape de détection et localisation de fautes et la première tâche qu'elle effectue est une taxonomie des blocs. En effet, on n'appliquera pas les mêmes traitements suivant qu'un bloc est défaillant, présumé défaillant, ou encore présumé correct.

prédicat	réponse	variable	valeur
POSSIBLEFAULT(b,ib_1)	?	b	B_2
(première tentative)		ib_1	IB_{23}
CONNECTED(b,downstream,ib_1)	YES	b	B_2
		downstream	B_3
		ib_1	IB_{23}
QUALIFY(downtream)	YES (B_3 prés. déf.)	downstream	B_3
INSTBI(ib_2,ERRONEOUS,_,_,OUTPUT)	YES	ib_2	IB_{34}
BLOCK(downstream,[· · · ,ib_2, · · ·])	YES	downstream	B_3
		ib_2	IB_{34}
POSSIBLEFAULT(b,ib_1)	?	b	downstream ≡ B_3
(deuxième tentative)		ib_1	$ib_2 \equiv IB_{34}$
CONNECTED(b,downstream,ib_1)	YES	b	B_3
		downstream	B_4
		ib_1	IB_{34}
QUALIFY(downtream)	YES (B_4 prés. déf.)	downstream	B_4
INSTBI(ib_2,ERRONEOUS,_,_,OUTPUT)	NO	--	--
BLOCK(downstream,[· · · ,ib_2, · · ·])	NO	downstream	B_4

Figure IV.16. Trace d'exécution de la clause POSSIBLEFAULT

La qualification de "présumé correct" pour un bloc peut paraître incongrue, mais il est essentiel de garder à l'esprit que les déductions faites par le système lors d'une session de travail donnée ne sont obtenues que relativement à l'application d'une séquence de test particulière, et il n'est pas concevable d'appliquer un test exhaustif (c'est-à-dire toutes les combinaisons possibles de vecteurs de test, dans le cas d'un test structurel), ce qui serait

pourtant l'idéal dans notre cas, car le système ne prend pas en compte des hypothèses classiques de fautes telles que les collages logiques.

Bien que non encore implémentés, ces traitements, qui font appel à des connaissances très vastes concernant le test, ont fait l'objet de quelques réflexions, livrées ici :

- Un bloc déclaré défaillant doit faire l'objet lui-même, et tant que la chose est possible, d'un partitionnement, de façon à lui appliquer les mêmes stratégies qu'au circuit tout entier.

On retrouve donc ici une méthodologie de type "start big approach" [Den68], basée sur une description hiérarchique à plusieurs niveaux d'un circuit, et qui vise à partir d'un ensemble de blocs qui pourraient être à l'origine d'une faute, puis à restreindre cet ensemble au fur et à mesure que l'on descend dans la hiérarchie de décomposition, de façon à avoir l'assurance de converger vers le résultat voulu.

- Un bloc déclaré présumé défaillant pourra être simulé (au niveau fonctionnel) en lui appliquant les mêmes entrées que celles mesurées lors de son observation au microscope, et, dans le cas où les résultats de simulation montrent que les sorties obtenues sont identiques aux sorties observées, on pourra déduire que le bloc n'était pas vraiment défaillant, mais qu'il propageait simplement une erreur.

- Pour les autres blocs, et d'une façon générale, il est intéressant d'étudier les différentes méthodes de génération de vecteurs de test, afin de décider si des méthodes connues, comme le D-algorithme ou PODEM, détaillées dans [Fuj85], peuvent être utilisées, ou s'il est nécessaire de définir de nouvelles techniques de génération de vecteurs de test, adaptées aux possibilités d'observabilité du microscope électronique, et surtout à l'approche par système à base de connaissances.
  Néanmoins, si la taille de chaque bloc a de plus pu être suffisamment réduite, il est même envisageable d'appliquer à ces blocs (ayant chacun un petit nombre d'entrées) un test exhaustif, puisque la notion de cône de couverture permet de restreindre dans ce cas la longueur de la séquence de test nécessaire. Il s'agira alors d'un test pseudo-exhaustif [Mac84].

IV.9 EXPLICABILITE DU FONCTIONNEMENT DU SYSTEME

La faculté d'auto-explication, ou du moins d'auto-explicabilité de son raisonnement est une caractéristique essentielle d'un système à base de connaissances.[27]

---

27. Certains chercheurs en intelligence artificielle vont même plus loin, en définissant l'I.A. comme la "science des explications" [Kod86].

En effet, que ce soit dans un but d'apprentissage et de perfectibilité du système, ou d'ergonomie de son utilisation, l'explicabilité du fonctionnement du système à base de connaissances fera toute la différence entre le type d'approche qu'il met en oeuvre et une programmation entièrement algorithmique, souvent efficace, certes,[28] mais dont le fonctionnement reste une "boîte noire" pour l'utilisateur.
En outre, et ceci est particulier au domaine auquel nous nous intéressons (la validation de prototypes de circuits complexes), l'intérêt du diagnostic n'est pas tant de pouvoir intervenir rapidement et efficacement en maintenance, mais plutôt, pour le concepteur du circuit, d'identifier avec exactitude en quoi, et à quelle étape de sa démarche, une erreur a pu intervenir dans la conception de son circuit.
Pour toutes ces raisons, nous nous sommes intéressée à l'explicabilité du raisonnement du système présenté.

Parmi les solutions examinées, nous en avons retenu deux, qui sont la gestion d'une "pile de raisonnement" et l'analyse de la trace d'exécution fournie par l'interpréteur PROLOG.

IV.9.1 Gestion d'une pile de raisonnement

Pour pouvoir expliquer le raisonnement du système, il faut être en mesure de mémoriser toutes les clauses qu'il a tenté de vérifier, avec les résultats obtenus (succès ou échec). Cette mémorisation utilise une structure de données adaptée au raisonnement des systèmes à base de connaissances en général : la pile ou arbre de raisonnement. En effet, la structure d'arbre, que l'on peut aussi représenter par une pile dans ce cas précis, est particulièrement aisée à manipuler lorsque l'on utilise la récursivité, propriété essentielle du raisonnement des systèmes à base de connaissances.

La méthode que nous allons expliciter dans la suite de ce paragraphe revient en fait à simuler le comportement de l'interpréteur PROLOG, de façon à mémoriser toutes les étapes du raisonnement.

IV.9.1.1 Méthode

Le principe général est assez simple : il suffit d'empiler, à l'entrée de chaque clause, la "tentative de prouver" cette clause, c'est-à-dire le fait que le littéral de tête de cette clause ait unifié avec un appel précédemment émis. De même, à la sortie de chaque clause, il faut empiler la "réussite de la preuve" de cette clause, c'est-à-dire le fait que tous les sous-buts

---

28. Il est vrai qu'en I.A. l'explicabilité prend souvent le pas, et en tous cas peut se faire au détriment de l'efficacité [Kod86]. Mais il n'en reste pas moins vrai également que, pour le traitement de problèmes combinatoires, par exemple, les méthodes entièrement algorithmiques deviennent inutilisables, parce que rédhibitoires.

constituant le corps de cette clause ont pu être vérifiés. Le littéral consistant à empiler la "réussite de la preuve" de la clause étant le dernier littéral de queue de cette clause, il va sans dire que le seul fait d'y parvenir implique que tous les sous-buts de la clause en question ont été prouvés.

Cette méthode permet d'obtenir, à la fin de l'exécution du littéral d'appel (la question), la "pile de raisonnement", composée uniquement de deux sortes de littéraux, celui de la "tentative de preuve" et celui de la "réussite de la preuve", avec comme arguments les différentes clauses que le système a pu appeler au cours d'une résolution.
Une variante de cette méthode serait de constituer non pas une "pile de raisonnement" unique, mais deux listes, celle des "tentatives de preuve" et celle des "réussites de preuve".
Une fois la pile constituée, il s'agit de la parcourir en vue d'expliquer le raisonnement suivi. Le principe de ce parcours est le suivant : pour chaque "tentative de preuve" d'une clause rencontrée dans la pile, si l'on trouve la "réussite de preuve" associée, c'est que cette clause a pu être vérifiée, sinon il y a échec. Le lecteur intéressé pourra trouver un exemple d'application de cette méthode dans [Mar87a].

IV.9.1.2 Problèmes posés et commentaires

Cette méthode, basée sur la gestion d'une "pile de raisonnement" parait simple à première vue, mais pose un certain nombre de problèmes.
En effet, cette méthode nécessite l'ajout des littéraux d'empilement de la "tentative de preuve" et de la "réussite de preuve" à chaque clause du programme de contrôle, que ces clauses soient atomiques ou non, ce qui implique les conséquences suivantes :

- Le programme de contrôle ne comportera plus aucun axiome, puisque les anciens axiomes se verront augmentés d'une queue de clause, composée des deux littéraux d'empilement.

- Tous les prédicats évaluables utilisés par l'interpréteur PROLOG devront être réécrits, pour y ajouter, comme aux clauses du programme, les littéraux d'empilement.

- La reconnaissance de la coupure ("cut") sera très complexe à gérer, puisque la présence du "cut" peut entraîner plusieurs empilements de "réussites de preuve" sans que les "tentatives de preuve" correspondantes aient été empilées.

- Le problème inverse du précédent sera aussi posé en cas d'échec de l'unification pour un littéral quelconque parmi les littéraux de queue d'une clause, on trouvera dans la pile la "tentative de preuve" de cette clause sans trouver la "réussites de preuve" correspondante. Ceci constitue un problème seulement dans le cas où la clause en question comporte un appel récursif, et il sera extrêmement difficile dans ce cas d'établir une correspondance entre les "tentatives de preuve" et les "réussite de preuve" figurant dans la pile.

- Enfin, si l'on veut non seulement expliquer le "pourquoi" d'un choix fait ou d'une solution trouvée, mais aussi le "pourquoi pas" d'une solution rejetée, cette méthode est insuffisante, car il faudrait retenir également tous les "échecs de preuve".

IV.9.2 Analyse de la trace d'exécution

La deuxième méthode d'explication du raisonnement examinée fait intervenir la trace d'exécution fournie par l'interpréteur PROLOG lui-même. Cette trace, prévue par les concepteurs du langage PROLOG et mise à la disposition des utilisateurs pour leur permettre de corriger leurs programmes sans trop de peine, revient en fait à l'exécution en mode "pas à pas" des programmes.
Avant d'exposer la méthode d'explication proprement dite, il convient, pour plus de clarté de présenter le principe de "debugging" utilisé ainsi que le format de la trace.

IV.9.2.1 Format de la trace

Le "debugging" des programmes PROLOG est basé sur le modèle de "boîtes de contrôle" ("procedure box control flow model"). Ces boîtes de contrôle représentent les procédures du programme, et on s'intéresse au contrôle de flux à travers quatre ports :

- Port d'appel initial (CALL)
- Port de retour (BACKTO)
- Port de sortie avec succès (EXIT)
- Port de sortie avec échec (FAIL)

Lorsque la trace est demandée par l'utilisateur, avant une question, l'interprétation de cette question sera effectuée en mode pas à pas par PROLOG, suivant le format suivant :

$$(x)\ y\ \text{nom\_du\_port} : \text{clause}(arg_1, \cdots, arg_n)$$

Les différents paramètres de ce format ont la signification suivante :
(x) : Ce numéro entre parenthèses correspond au numéro (unique) d'identification de la clause appelée. Il est incrémenté à chaque entrée d'une procédure par le port dénommé CALL.
y : Ce numéro correspond au degré de profondeur dans la résolution en cours. Il permet de connaître le nombre de procédures qui sont les ascendants de la clause actuellement examinée.
nom_du_port : Il s'agit de l'un des quatre ports d'entrée/sortie de la procédure (CALL, BACKT EXIT ou FAIL).
clause($arg_1, \cdots, arg_n$) : Il s'agit de la clause en cours d'exécution, accompagnée de ses arguments.

IV.9.2.2 Méthode

Cette deuxième méthode de réalisation du processus d'explication du raisonnement utilise donc la trace de l'exécution d'un programme, fournie par l'interpréteur PROLOG. Cette trace ne peut servir d'explication en elle-même, étant donné son format, et spécialement dans le cas où le destinataire de l'explication est un utilisateur non-informaticien.
La méthode consiste donc, d'une part à récupérer la trace dans un fichier, et d'autre part à procéder à l'analyse syntaxique et sémantique du contenu de ce fichier, en fonction du modèle de "boîtes de contrôle", tout en gardant à l'esprit que les arguments figurant dans les clauses, et non encore instanciés, sont représentés par l'interpréteur PROLOG à l'aide de numéros de variables.
On trouvera également un exemple d'exécution de cette méthode dans [Mar87a].

Notons que, malgré la réflexion dont elles ont fait l'objet, aucune méthode d'explication n'a encore été implémentée, d'une part car il est encore trop tôt pour le faire, car cela entrera dans le cadre du processus d'interfaçage de PESTICIDE avec son environnement extérieur - y compris l'utilisateur du système - (voir chapitre suivant), et d'autre part car la génération d'explications constitue en elle-même un thème de recherche complexe, lié aux domaines de l'apprentissage et de l'ergonomie [Kod86].
On retiendra toutefois que, s'il n'est pas encore expliqué, le fonctionnement de PESTICIDE est explicable, par sa conception même.

IV.10 CONCLUSION

Il n'est pas encore temps d'évaluer le système présenté dans ce chapitre, ni même de proposer des perspectives pour son développement, car ceci sera fait après analyse comparative dans le chapitre suivant.
Nous nous contenterons ici, en guise de récapitulation, de synthétiser les principales composantes déjà implémentées de PESTICIDE, et de présenter les résultats de premières expériences utilisant ce système.

Trois bases de connaissances sont utilisées dans PESTICIDE. Elles contiennent des connaissances structurelles et fonctionnelles sur le circuit sous test, et des connaissances comportementales sur les résultats du test en cours.
Trois modules de contrôle exploitent ces différentes connaissances : un module de vérification assure le maintien de la cohérence dans les bases de connaissances, un module de détection identifie les fautes sur les connexions, et un module de localisation recherche les fautes dans les blocs.
Ces différents éléments sont gérés par l'interpréteur PROLOG, autour duquel est bâti le système. L'architecture globale de ce système est schématisée en figure IV.17.

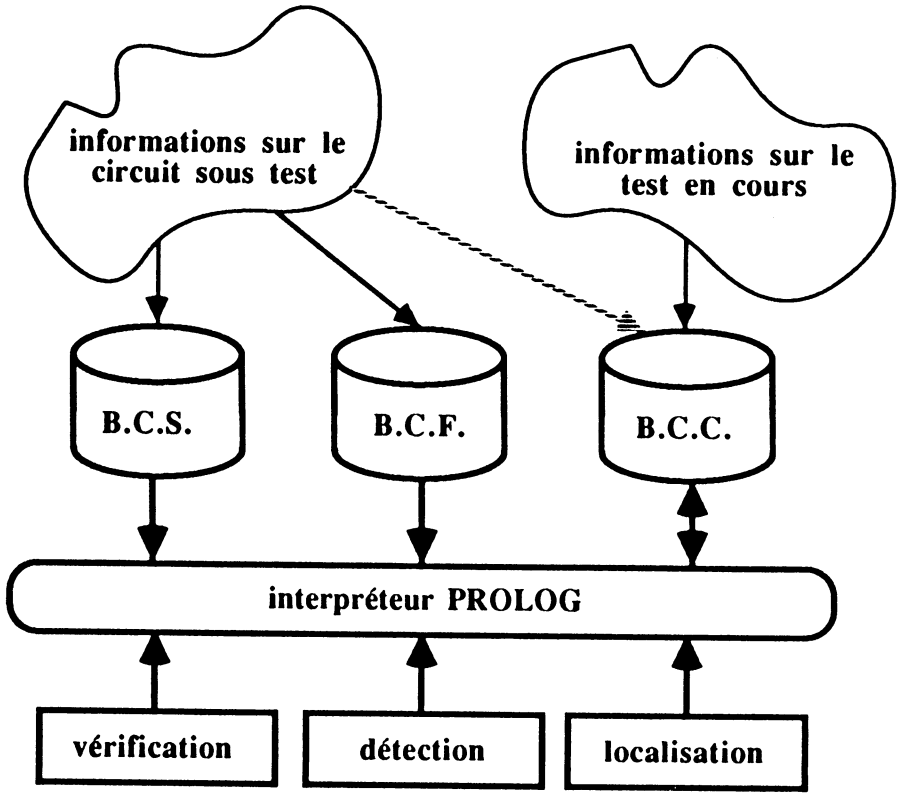


Figure IV.17. Architecture globale du système PESTICIDE

Des expériences ont été réalisées à l'aide de PESTICIDE, dans le double but de valider les stratégies de localisation de fautes définies pour différentes hypothèses et d'effectuer quelques mesures de paramètres intervenant dans le déroulement du processus de diagnostic. Ces expériences n'ont pas été effectuées sur un circuit réel, car le lien entre PESTICIDE et son environnement n'est pas encore complètement réalisé, bien que des travaux à cet effet soient en cours (cf. chapitre suivant). Néanmoins, des résultats montrant la faisabilité de l'approche définie et de l'implémentation réalisée ont pu être obtenus et sont présentés en figures IV.18 et IV.19.

Exemple	# blocs.	# int. bloc	# erreurs	FSC	FMC
Add. 2-b.	2	10	3	0.08	0.14
Add. 4-b.	4	20	5	0.16	0.20
Add. 8-b.	8	40	7	0.43	0.61
Add. 16-b.	16	80	10	1.02	1.27

Figure IV.18. Résultats obtenus en cas de Faute Simple Combinatoire et Faute Multiple Combinatoire (FSC et FMC en secondes)

Exemple	# blocs.	# int. bloc	# instances	# erreurs	FSS	FMS
Add. 2-b.	2	10	10	3	0.50	0.41
Add. 4-b.	4	20	20	5	1.16	0.67
Add. 8-b.	8	40	40	7	2.80	1.38
Add. 16-b.	16	80	80	10	7.10	3.30
Bascule D	6	18	72	15	10.70	2.20

Figure IV.19. Résultats obtenus en cas de Faute Simple Séquentielle et Faute Multiple Séquentielle (FSS et FMS en secondes)

La figure IV.18 indique le temps CPU mesuré en secondes pour l'exécution des stratégies de localisation de fautes sur une série d'additionneurs (2-bits, 4-bits, 8-bits et 16-bits), sous les hypothèses de "Faute Simple Combinatoire" (FSC) et "Faute Multiple Combinatoire" (FMC), et dans lesquels un nombre variable d'interfaces bloc erronées a été simulé. Dans tous les cas, chaque additionneur élémentaire est modélisé par un bloc.
La figure IV.19 indique le temps CPU mesuré en secondes pour l'exécution des stratégies de localisation de fautes sous les hypothèses de "Faute Simple Séquentielle" (FSS) et "Faute Multiple Séquentielle" (FMS). Les expériences ayant conduit à ces résultats ont été réalisées sur la série d'additionneurs évoquée précédemment, ainsi que sur une bascule D maître/esclave, pour laquelle chaque porte NAND a été modélisée comme un bloc.
Les conditions d'expérimentation sont fournies dans les deux cas (figures IV.18 et IV.19). Ces conditions sont le nombre de blocs considérés, le nombre total d'interfaces bloc, ainsi que le nombre d'interfaces bloc erronées. Dans le cas des "stratégies séquentielles", le nombre d'instances d'interfaces bloc prises en compte est également fourni.
Les temps CPU en secondes ont été obtenus sur un ordinateur BULL DPX5000, en utilisant la version 1.5 de l'interpréteur C-PROLOG sous système d'exploitation UNIX System V.

La première remarque qui s'impose au vu de ces résultats concerne les deux types de circuits - combinatoires ou séquentiels - traités : les cas particuliers choisis pour les besoins de l'expérimentation sont très simples, mais ce choix est moins restrictif qu'il n'y paraît puisqu'à son stade actuel de développement, PESTICIDE n'utilise, à un niveau hiérarchique donné, que des règles de propagation topologique ; par conséquent, les paramètres importants sont le nombre de blocs considérés, leur degré d'interconnexion et le nombre d'interfaces bloc, beaucoup plus que la complexité interne des blocs.

La deuxième remarque ne concerne que les circuits combinatoires : la table de la figure IV.18, comparée aux quatre premières lignes de celle de la figure IV.19, permet d'établir les ratios suivants :

$$FSS/FSC = 6.7$$
$$FMS/FMC = 2.7$$

FSC/FMC = 0.7
FSS/FMS = 1.7

L'interprétation de ces ratios conduit aux conclusions suivantes :

- Que ce soit dans le cas de faute simple ou de faute multiple, le temps CPU obtenu avec les "stratégies combinatoires" est bien inférieur à celui obtenu dans les mêmes conditions, mais avec les "stratégies séquentielles". Ceci s'explique par le fait que, dans ce dernier cas, les formules concernant la validité d'un instant de mesure doivent être évaluées pour chaque combinaison d'instances d'interfaces bloc. Par conséquent, l'intérêt du choix de quatre différentes stratégies, définies pour quatre hypothèses distinctes, est confirmé par ces résultats.

- Néanmoins, on peut noter que le ratio FSS/FSC est environ 2.5 fois plus grand que le ratio FMS/FMC (ce qui correspond au fait que FSS/FMS est 2.5 fois plus grand que FSC/FMC). En d'autres termes, le temps CPU nécessaire en cas de "stratégie séquentielle", rapporté au temps requis en cas de "stratégie combinatoire", a subi une augmentation plus importante pour le processus de localisation de faute simple que pour le processus de localisation de faute multiple. Ceci s'explique par les tailles des espaces de recherche d'instances d'interfaces bloc à explorer dans chacun des cas : l'espace de recherche est beaucoup plus important pour la propagation arrière effectuée en vue du diagnostic d'erreurs propagées (faute simple) que pour l'identification de blocs présumés défaillants (faute multiple). En effet, dans le premier cas l'espace de recherche est un arbre, alors que dans le deuxième cas il est simplement constitué de quelques chemins à parcourir directement (sans nécessité de "backtracking").

V. EVALUATION ET COMPARAISON DES DEUX APPROCHES

V.1 INTRODUCTION

Le but de ce chapitre est de dégager les apports et lacunes de chacun des deux systèmes présentés respectivement aux chapitres III et IV, en fonction bien sûr des possibilités du test par faisceau d'électrons, mais également de ses impératifs et limites esquissés au chapitre II. Il s'agit également pour nous ici d'essayer de montrer quelle est, de l'utilisation d'un dictionnaire de fautes ou d'une approche basée sur la connaissance, la méthode la plus viable et la mieux adaptée pour la validation de prototypes de circuits intégrés.

Mais il faut comparer des entités comparables : le système ADVICE est un système intégré de test, à considérer dans sa globalité, alors que le système PESTICIDE n'est qu'une partie d'un système intégré de test, chargée des tâches décisionnelles - le diagnostic. De même, les moyens engagés sur chacun des deux projets ne peuvent être comparés.

C'est pourquoi, dans la première partie de ce chapitre, nous évaluerons séparément chacun des deux projets présentés, sans nous préoccuper de la pertinence du choix de la méthode de diagnostic utilisée dans chaque cas. C'est seulement ensuite que, sans tenir compte de leurs degrés de réalisation matérielle et d'intégration dans le système global de test par faisceau d'électrons, nous effectuerons une comparaison des deux méthodes :

- diagnostic à l'aide d'un système à base de connaissances - PESTICIDE - que nous avons conçu et réalisé
- diagnostic à l'aide d'un dictionnaire de fautes - ADVICE - auquel nous avons contribué pendant les trois dernières années (1987-1989) du projet.[29]

---

29. Notre participation au projet ADVICE intervient au niveau de la couche "méthode" [Mel87a]. Elle concerne principalement, en collaboration avec le CSELT, la conception globale de cette couche, la définition de l'interpréteur de commandes, la définition de stratégies de conception adaptée au test par faisceau d'électrons [MCM87], de simulation [MaM87], et de génération de vecteurs de test [Mar87b]. Au niveau réalisation, elle peut se chiffrer à 10.000 lignes de programme (identification et sélection des vecteurs de test).
En ce sens, nous ne sommes pas le porte-parole du projet ADVICE, et tout ce qui sera dit dans la suite concernant l'évaluation du système, particulièrement les critiques que nous pourrons formuler, n'engage que notre propre responsabilité, et en aucune façon celle des autres membres du projet.

V.2 EVALUATION DU SYSTEME ADVICE

V.2.1 Evaluation du système dans sa globalité

En tant que système intégré de test, il est clair que le système ADVICE est, parmi tous les outils de test par faisceau d'électrons, le plus avancé à l'heure actuelle, bien qu'encore au stade de système prototype.
En effet, si l'on se réfère aux différents critères pris en compte pour l'établissement du tableau comparatif du chapitre II (figure II.4, § II.3.4), le système ADVICE présente les caractéristiques suivantes :

- contrôle du T.F.E. : automatique
- positionnement fin : automatique, par des techniques de reconnaissance de formes ("pattern matching")
- modes de mesure : forme d'onde et image
- formats layout acceptés : CIF et CALMA GDSII
- formats netlist acceptés : HILO (textuel)
- correspondance layout-netlist : automatique (utilisation de DRACULA)
- comparaison : automatique
- description logique : oui (HILO)
- interface utilisateur : ergonomique.

Outre ces caractéristiques, le système ADVICE offre une assistance à l'utilisateur pour la sélection des points à observer et la localisation de fautes, ainsi que des facilités pour conduire les processus de simulation et de simulation de fautes, et de sélection de programmes de test.

Néanmoins, l'un des inconvénients majeurs du système ADVICE reste sa portabilité faible. En effet, ce système, conçu pour être implanté sur un MicroVax (VaxStation II GPX), est fortement dépendant du système d'exploitation hôte, VMS.
Ceci pourrait être compréhensible pour les procédures de bas niveau (contrôle du T.F.E., gestion du graphisme, etc.), mais devient peu pratique dès lors qu'il s'agit de l'interface utilisateur, de la communication entre processus, et de certaines commandes de la couche "méthode", notamment pour ce qui concerne la gestion de fichiers, et les entrées/sorties de manière générale.
On peut également déplorer, malgré une utilisation intensive du langage de programmation C, la présence de certains modules écrits en PASCAL, et le recours à des caractéristiques non-standards de C.
Une autre limitation du système, encore compréhensible pour un prototype, réside dans son

incapacité actuelle à prendre en compte d'autres formats, standards, de CAO.

Les limitations imposées par les choix effectués pour le système ADVICE ont été perçues par les partenaires tout au long du développement du projet, et ces choix seront remis en question pour la poursuite de ce projet (ADVICE II - "Advanced E-beam based validation system for ULSI devices" - Projet N. 5043 du programme ESPRIT II - CCE).
Ainsi, le choix d'une station de travail mieux adaptée, et par conséquent l'abandon vraisemblable du système VMS pour le système d'exploitation UNIX constitueront les premières tâches du projet ADVICE II.
De même, l'absence actuelle de la possibilité d'afficher le schéma logique du circuit sous test peut constituer un inconfort dans l'utilisation du système pour un concepteur, habitué à des représentations graphiques.
Enfin, il est également prévu, dans le cadre du projet ADVICE II, d'examiner d'autres systèmes de CAO existants, et de prendre en considération des formats standards de description, notamment le format EDIF.

V.2.2 Contribution du projet ADVICE au développement de l'état de l'art

L'évaluation précédente concerne le système ADVICE du point de vue de son utilisation et de sa maintenance : il y était donc surtout question des choix effectués du point de vue matériel et logiciel utilisés.
Il est également important d'évaluer le projet du point de vue des apports scientifiques auxquels il a contribué, et il s'agit notamment ici des stratégies développées pour le test de circuits intégrés par faisceau d'électrons (cf. chapitre III, § III.3).

Il est dès à présent établi que les recherches effectuées dans le cadre du projet ADVICE ont fait progresser l'état de l'art dans le domaine de la validation de prototypes de circuits intégrés par faisceau d'électrons.

Ces recherches concernent d'abord le développement de stratégies de conception en vue du test par faisceau d'électrons.
En effet, pratiquer un examen interne, comme celui permis par la sonde électronique, peut permettre de pallier un certain nombre d'inconvénients relatifs aux examens externes, mais pose des problèmes spécifiques (cf. chapitres II et III), et de même que des méthodes de conception adaptée au test (D.F.T.) ont été développées depuis une dizaine d'années, il a fallu déterminer des règles de conception adaptée au test par faisceau d'électrons, ce qui n'avait pas été fait jusqu'alors.
Outre les règles de conception logique et les règles topographiques, les règles de sélection de points de test, et particulièrement celles liées au diagnostic (chapitre III, § III.3.1.3), constituent, avec les stratégies de simulation en vue du test par faisceau d'électrons, l'un des apports majeurs du projet à la recherche dans le domaine.

Ces stratégies de simulation (cf. chapitre III, § III.3.2) reposent sur la définition, ou l'adaptation au test par faisceau d'électrons, de concepts déterminants pour une localisation de fautes efficace, qui puisse minimiser le coût de l'examen interne du circuit, tout en tenant compte de la hiérarchie de décomposition du dispositif à tester.
En ce sens, et de même que les stratégies de conception évoquées ci-dessus si on les compare aux méthodes de D.F.T., ces stratégies de simulation jettent les bases d'une méthodologie de simulation pour le test interne, constituant l'équivalent des méthodologies développées jusqu'à présent pour le test externe de circuits intégrés.

Mais autant les stratégies évoquées précédemment peuvent constituer un apport réel à la recherche dans le domaine, autant les choix concernant la génération de stimuli et les dictionnaires de fautes peuvent être déplorés.
Nous ne reviendrons pas sur la génération de stimuli, qui se restreint en réalité à une sélection de stimuli préalablement générés, ceci est exposé au chapitre III, § III.3.3. Si nous nous contentons de reformuler ici nos griefs contre cette décision, c'est bien parce que les choix effectués ne permettent pas de tirer parti des concepts développés dans le cadre des stratégies de simulation, et par conséquent ces choix résultent en un appauvrissement du système de ce point de vue, puisque les spécificités du test par faisceau d'électrons n'y sont pas du tout exploitées.
Le choix d'une approche basée sur l'utilisation de dictionnaires de fautes est l'un des reproches majeurs que nous faisons au projet ADVICE. Il sera commenté dans la suite (§ V.4).

V.2.3 Evaluation de la couche méthode

Grâce à cette couche supplémentaire, le système ADVICE est plus qu'un "T.F.E. étendu", et se classe parmi les systèmes intégrés de test par faisceau d'électrons.

Cette couche "méthode" a pour vocation de réaliser ou d'aider à l'accomplissement des tâches de diagnostic.
Pour ce faire, elle a la charge de gérer toutes les informations accumulées au cours d'une session de test donnée. Ces informations sont organisées en une structure de données unique, qui s'en trouve donc extrêmement complexe et d'une exploitation très pesante.

Cette structure de données reflète en effet non seulement la description du circuit, mais également des informations sur la session de test depuis son commencement, y compris les résultats des préprocessus (simulation, simulation de fautes, stimuli appliqués, réponses obtenues, etc.).

Si la nécessité de cette quantité d'informations ne peut être remise en question dans le cadre de l'approche de diagnostic choisie, les choix relatifs à la façon de l'organiser auraient, quant à eux, pu être différents : le système aurait pu gagner en simplicité et en efficacité si plusieurs niveaux d'organisation de données avaient été pris en compte, correspondant à différents degrés de détail, et à différents types d'informations, pour

différents degrés d'exploitation, c'est-à-dire différentes phases du diagnostic.
Outre le gain en simplicité et en efficacité, une telle organisation des données aurait permis d'obtenir un système beaucoup moins sensible aux problèmes d'espace mémoire, puisque seules les informations nécessaires à une étape donnée du diagnostic auraient résidé en mémoire centrale : lorsqu'il s'agit de circuits réels et non plus d'exemples d'école, ces contingences peuvent prendre des proportions rédhibitoires.

La couche "méthode" est vue de l'extérieur, c'est-à-dire par l'utilisateur ou par les autres modules constituant le système, comme un interpréteur de commandes. Ces commandes, décrites au chapitre III (§ III.5.1), permettent de consulter, mettre à jour, et utiliser la structure de données.
En raison de la complexité, déjà évoquée, de cette structure de données, l'utilisation de ces commandes requiert une connaissance approfondie du système. En effet, il existe près de 70 commandes possibles et, malgré l'existence d'un manuel en ligne, leur utilisation à bon escient ne peut être que le résultat d'un entraînement intensif.

C'est pourquoi il a été décidé que le mode de fonctionnement du système ADVICE serait paramétrable : on peut en effet utiliser ce système selon l'un des trois modes mis en place, le mode manuel, le mode assisté, ou le mode automatique. Ces trois modes de fonctionnement sont explicités au chapitre III, § III.5.2.

Si l'on revient au tableau comparatif du chapitre II, § II.3.4, on peut dire que le système ADVICE, utilisé en mode manuel, correspond à un "T.F.E. étendu", avec comparaison automatique entre résultats attendus et résultats observés.
Le système ADVICE, utilisé en mode assisté, pourrait être comparé au système NTT, doté de la méthode présentée par [YaO87] de diagnostic assisté à l'aide d'un dictionnaire de fautes.
Mais, utilisé en mode automatique, le système ADVICE, malgré tous les inconvénients et faiblesses identifiés dans ce chapitre, reste encore le plus achevé et le plus puissant à l'heure actuelle : en dépit de son état de prototype, il n'a pas d'équivalent.

V.3 EVALUATION DU SYSTEME PESTICIDE

Avant toute évaluation, il est utile de rappeler que le système PESTICIDE a uniquement en charge les tâches décisionnelles du processus de test par faisceau d'électrons : il ne s'agit donc pas d'un système intégré de test, et c'est la raison pour laquelle les choix effectués pour le système d'acquisition des informations (cf. chapitre IV, § IV.2.1) ne seront pas commentés ici.
Néanmoins, dans un premier temps, il est intéressant de situer PESTICIDE par rapport au reste du système de traitement des informations, ainsi que par rapport à son environnement extérieur, utilisateur du système compris. C'est seulement ensuite que nous commenterons les choix effectués pour la conception du système PESTICIDE lui-même.

V.3.1 PESTICIDE et son environnement

A l'heure actuelle, PESTICIDE n'est pas encore lié à son environnement extérieur. Il s'agit là d'un inconvénient majeur, ne serait-ce que pour la validation du système.

En effet, certaines tâches du système de traitement des informations ne sont pas encore réalisées, notamment la comparaison automatique entre les valeurs de potentiel sur les connexions, observées au microscope électronique, et les résultats de simulation.

Par contre, l'interfaçage entre la description du circuit et PESTICIDE a déjà fait l'objet d'investigations [Ben89], et est en cours d'achèvement [Ben90].
Dans son état d'avancement actuel, l'interface permet de générer automatiquement la base de connaissances structurelles (exception faite des paramètres "délai" et "temps de maintien" des interfaces bloc) et les clauses "cône de couverture" de la base de connaissances fonctionnelles.
Cette première version de l'interface utilise comme langage source le langage de description de circuits du système HILO3 [Gen85]. Des versions futures, établissant l'interfaçage de PESTICIDE avec d'autres langages, sont envisageables, en fonction de l'évolution des outils standards en matière de CAO.

Dans le schéma synoptique de l'interface (figure V.1), les analyseurs lexical et syntaxique d'un programme HILO3 ont été respectivement générés à partir des outils LEX [LeS75] et YACC [Joh75] disponibles sous système UNIX.

Si aucune erreur ne figure dans la description du circuit, le générateur de forme intermédiaire [Ben89] permet d'établir une table des modèles utilisés (modèles définis par le concepteur ou modèles prédéfinis du langage HDL de HILO3), ainsi que l'arbre des instances d'appel de ces modèles, la racine de l'arbre correspondant à l'instance relative au modèle principal (description du circuit au plus haut niveau hiérarchique).
A partir de cet arbre d'instances, le générateur de connaissances [Ben90] construit les bases de connaissances structurelles et fonctionnelles directement exploitables par PESTICIDE. La base de connaissances structurelles est déduite de l'arbre des instances de façon quasi-immédiate, tandis que la base de connaissances fonctionnelles (clauses concernant les cônes de couverture) requiert un traitement plus élaboré, puisqu'il s'agit de déterminer des informations non présentes explicitement dans le fichier source.

En ce qui concerne les liens de PESTICIDE vers le reste du système de traitement des informations, ou même vers le système d'acquisition des informations, ils sont relatifs à des processus de génération de vecteurs de test et de commande de déplacement de la platine du microscope (cf. chapitre IV, § IV.2.3).
Ces liens demandent également à être réalisés, et si les commandes permettant d'agir sur le microscope restent en dehors de nos objectifs, il est envisagé de développer des méthodes de génération - ad hoc - de vecteurs de test. Ces méthodes tiendront certainement compte

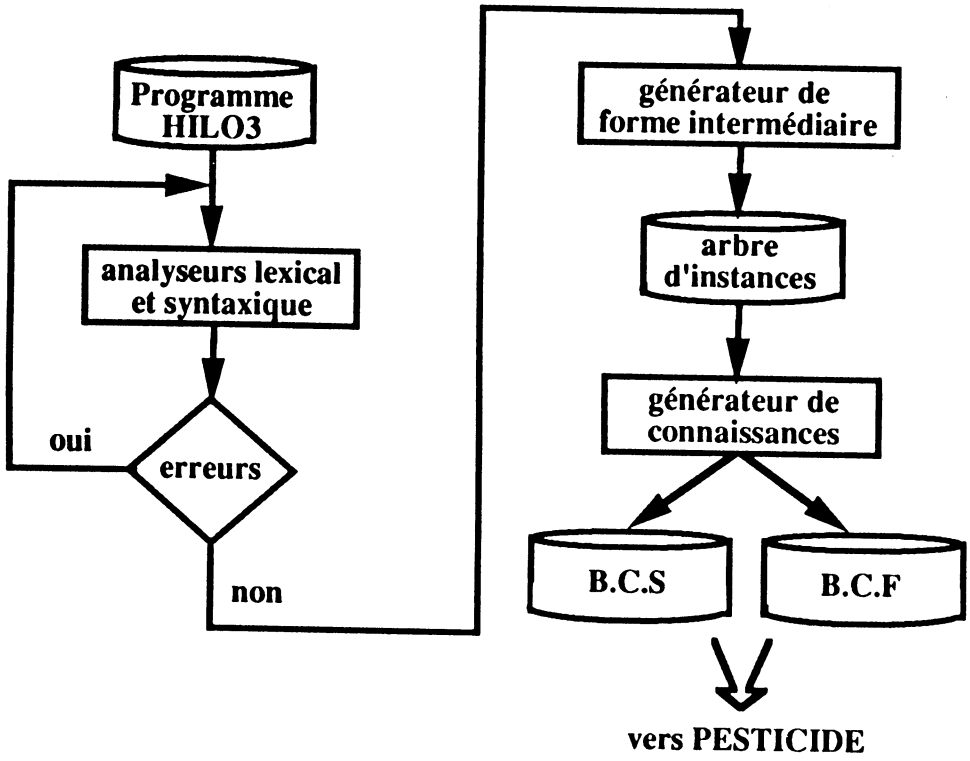


Figure V.1. Schéma synoptique de l'interface

des points exposés au chapitre III, § III.3.3.

Le système PESTICIDE a donc été conçu en formulant le présupposé que ses données d'entrée étaient disponibles sous le format désiré (il s'agit du format explicité au chapitre IV, § IV.4), sans préjuger de la façon d'obtenir ces données.

V.3.2 Modélisation, organisation et utilisation des connaissances

Considérée globalement, la façon de modéliser et d'organiser les connaissances dans PESTICIDE constitue le point fort de ce système.

Point fort par sa généralité, tout d'abord, puisque cette modélisation est adaptée à tout type de circuit, qu'il soit séquentiel ou combinatoire, et peut même être utilisée pour des cartes et/ou systèmes électroniques.
En effet, les composants et leurs paramètres pris en compte dans la modélisation ne sont pas relatifs à un type de composant particulier, et peuvent donc représenter, dans une vue structurelle, toute description hiérarchique de haut niveau.
En outre, cette généralité n'est pas pénalisante, car si l'on veut décrire de façon plus spécifique le dispositif sous test, les connaissances fonctionnelles permettent d'exprimer ces particularités. Si les éléments de connaissances fonctionnelles actuellement introduits dans PESTICIDE restent encore relatifs à une représentation plutôt générique de plusieurs types de circuits, une modélisation de niveau moins élevé pourrait sans peine augmenter la base

de connaissances fonctionnelles déjà existante de primitives décrivant la fonction de chacun des blocs logiques représentés. Ainsi, il est tout à fait aisé de prévoir des primitives de description non seulement des portes logiques couramment utilisées dans la synthèse de circuits, mais également de cellules et modèles plus complexes, servant à la conception, structurée, de circuits de taille plus importante.

Point fort par sa simplicité, ensuite, tant dans les concepts modélisant le dispositif sous test, que dans leur utilisation, puisque la modélisation et l'organisation choisies peuvent être induites tout à fait naturellement d'une démarche de conception à laquelle sont habitués les architectes de circuits.
De plus, les notions de partitionnement et de hiérarchie, toujours présentes dans une démarche de conception de circuits complexes, qu'il s'agisse de circuits spécifiques à la demande (ASICs, "Application-Specific Integrated Circuits"), ou de circuits fabriqués en grande série (standards), sont largement exploitées dans cette modélisation, de telle sorte qu'il est possible d'associer, à chaque niveau hiérarchique de description, une base de connaissances structurelles et une base de connaissances fonctionnelles, sur lesquelles vont opérer les mêmes stratégies de détection et de localisation de fautes.
Cet aspect du système PESTICIDE permet de suivre une méthodologie "start-big" réelle (cf. chapitre IV, § IV.8), ne nécessitant pas l'existence en mémoire de toutes les données à chaque phase du diagnostic.
Le fait de séparer les données correspondant à différents niveaux de description, et à différents éléments composant le dispositif sous test présente enfin l'avantage de tenir compte des processus de conception de circuits tels qu'ils se déroulent actuellement à cause de la complexité des circuits visés, à savoir l'utilisation de cellules de bibliothèque, la réalisation de parties de circuit par des personnes différentes travaillant indépendamment les unes des autres, et même la génération automatique (compilation de silicium [Jer89]) de certains modules composant le circuit final.

Point fort par sa facilité d'exploitation, enfin, qu'il s'agisse de vérifier la cohérence des informations contenues dans les bases de connaissances, de détecter les fautes sur les connexions, ou de localiser les fautes dans les composants du circuit sous test.
Ces trois tâches reposent en effet sur l'examen et la mise en correspondance de paramètres pertinents des objets contenus dans chacune des trois bases de connaissances.
On peut noter à ce sujet que la possibilité de vérifier la cohérence des informations fournies au système de diagnostic n'est pas offerte par une structuration de données plus classique, comme celle utilisée dans le système ADVICE par exemple.
Quant au processus de localisation de fautes, il est rendu très simple encore une fois par la modélisation choisie et les éléments de connaissance introduits d'une part, et d'autre part par le seul modèle de fautes pris en compte, c'est-à-dire la divergence du comportement observé par rapport au comportement attendu. Ceci permet d'éviter de formuler des hypothèses restrictives sur le type de fautes ayant pu survenir, sachant ces modèles (collages, principalement) peu représentatifs de la réalité.

Les stratégies de localisation de fautes utilisées sont donc celles naturellement suivies par un analyste humain, et correspondent aux règles de propagation de fautes dans les circuits digitaux.
Pour finir, on peut signaler à l'actif d'une telle modélisation des connaissances la possibilité de traiter sensiblement de la même manière aussi bien les circuits séquentiels que combinatoires, par l'intermédiaire de l'établissement des formules d'évaluation de la validité d'un instant de mesure, et par là-même de tirer parti du type d'observation pratiqué à l'aide du test par faisceau d'électrons, qui permet d'obtenir des "vues instantanées" du dispositif en examen.

V.3.3 PESTICIDE, système à base de connaissances de seconde génération

PESTICIDE est un système à base de connaissances de seconde génération.
Système à base de connaissances, d'abord : ce terme nous paraît en effet plus approprié que celui de "système expert", puisqu'il ne s'agit pas ici d'utiliser une connaissance empirique pour détecter l'occurrence d'une situation de dysfonctionnement préalablement rencontrée et répertoriée, et d'en inférer la cause correspondante. Il s'agit au contraire de raisonner, à partir de la connaissance du dispositif à étudier, en utilisant une connaissance, dite profonde, des lois générales du domaine, pour être en mesure d'expliquer les causes d'un dysfonctionnement observé.
Cette approche, qui se base sur une claire distinction entre la structure, le fonctionnement attendu, et le comportement en situation du dispositif étudié, ainsi que sur une utilisation intensive des notions de partitionnement et de hiérarchie (cf. première partie) est utilisée pour le diagnostic proprement dit, c'est-à-dire la localisation de fautes.

Mais, comme il a été montré au chapitre IV, § IV.5, cette approche présente l'inconvénient de pré-supposer que la description de la structure et du fonctionnement du dispositif étudié est exacte, puisque cette description représente la référence à laquelle le comportement observé sera confronté. Or cette description peut elle-même comporter des erreurs. Pour s'affranchir de cette limitation, une vérification de la cohérence des informations a été mise en place et, puisque cette vérification s'effectue par la détection d'incohérences pouvant potentiellement survenir parmi diverses formes d'incohérences préalablement répertoriées, il s'agit bien là d'utiliser une connaissance empirique, telle que celle possédée par un expert de ce domaine. Le fait que les deux approches - la première utilisant des connaissances "profondes", et la deuxième utilisant des connaissances "de surface" - coexistent dans le même système font de PESTICIDE ce qu'il est convenu d'appeler maintenant un système à base de connaissances de seconde génération.

V.4 COMPARAISON DES DEUX APPROCHES DE DIAGNOSTIC

Il s'agit là des deux approches respectivement mises en oeuvre dans les systèmes ADVICE et PESTICIDE.

V.4.1 Dictionnaire de fautes vs. système à base de connaissances

Traditionnellement, les dictionnaires de fautes sont utilisés pour pallier les faibles possibilités d'observabilité des méthodes classiques comme la sonde mécanique guidée, par exemple (cf. chapitre II, § II.2.4). Mais, l'outil de microscopie électronique faisant l'unanimité quant aux performances accrues qu'il offre, il s'agit à présent d'adapter ou de définir les méthodes de test adéquates pour utiliser le mieux possible les résultats qu'il peut fournir.
Dans la suite de ce paragraphe, les deux méthodes seront comparées. Ces deux méthodes reposant sur le même principe de base, à savoir la consultation d'une base de données dans le but d'identifier les fautes, la comparaison se fera sur les trois aspects suivants : génération, consultation et performances de la base de données.

V.4.1.1 Génération de la base de données

Des programmes de simulation du fonctionnement du circuit sous test seront utilisés par les deux méthodes à l'étape de génération de la base de données. La différence réside dans le fait que, dans le cas du dictionnaire, tout le processus de génération sera entièrement basé sur les résultats de la simulation [RiB85], alors que dans le cas du système à base de connaissances, seule une partie des données sera fonction de ces résultats, cette partie constituant la base de connaissances comportementales du système, c'est-à-dire les données concernant la session de test en cours.
Ceci implique que le processus de génération sera entrepris pour chaque circuit à tester, et, par conséquent, qu'à chaque circuit ou type de circuit correspondra un dictionnaire, tandis que seule une partie des bases de connaissances sera générée à chaque test.

L'exemple du dictionnaire de fautes décrit par V. RATFORD et P. KEATING [RaK86] est à ce titre édifiant : le dictionnaire qu'ils décrivent est basé sur l'interprétation des différentes combinaisons de sorties primaires du circuit à tester, de façon à créer des relations entre les différentes fautes existant dans l'univers de fautes considéré et les signatures de ces fautes, constituées par les combinaisons de sorties primaires regroupées en ensembles. Ces relations, créées par un programme de génération, vont former le dictionnaire.
L'établissement des relations sus-citées est un processus très coûteux, puisqu'il implique la consultation et l'analyse de toutes les combinaisons de sorties primaires obtenues par simulation, pour vérifier si elles correspondent aux valeurs obtenues pour un circuit correct. Il faut en outre regrouper les fautes de même signature en ensembles.
Enfin, il faut signaler que dans le cas du dictionnaire, la simulation est une simulation des différentes fautes pouvant survenir, effectuée sur la base de modèles de fautes préalablement établis, alors que dans le cas du système à base de connaissances, il s'agit de simuler le fonctionnement correct du circuit, ce qui est moins coûteux en temps.

Pour ce qui est du système à base de connaissances, la génération des données comportementales sera effectuée simplement par deux interfaces de traduction des données provenant d'une part du simulateur, et d'autre part des résultats observés sur le circuit réel, ces traductions étant nécessaires pour adapter les différents formats de données au format accepté par le système à base de connaissances.
De plus l'un des avantages les plus importants des systèmes à base de connaissances est qu'ils peuvent fonctionner avec des connaissances partielles ou incomplètes, quitte à compléter la base de connaissances plus tard, alors que le dictionnaire de fautes doit être généré complètement avant sa première utilisation.

V.4.1.2 Consultation de la base de données

La consultation du dictionnaire est assez particulière, dans le sens où une faute ne peut être utilisée en tant que référence, puisque le but est précisément la détermination de la faute. Il est donc hors de question d'utiliser les fonctions d'accès classiques (dichotomie, hash-code, etc.) permettant d'accélérer la recherche. Il s'agit, au contraire, d'analyser toutes les relations composant le dictionnaire et de déterminer celles qui correspondent aux résultats observés sur le circuit réel. Le temps de consultation est alors proportionnel à la taille du dictionnaire, et il faut rappeler ici que celui-ci doit être complet, c'est-à-dire qu'il doit comporter toutes les relations possibles, pour pouvoir être utilisé.
Si l'on utilise un système à base de connaissances, par contre, et quelle que soit la taille de celui-ci, il disposera nécessairement d'heuristiques permettant d'accélérer l'obtention du diagnostic.
Ceci implique donc qu'un dictionnaire doit avoir la taille la plus restreinte possible, ce qui n'est pas facilement compatible avec l'exigence de complétude de la base de données, alors qu'un système à base de connaissances ne sera pas limité au niveau de la taille de sa base de données.

V.4.1.3 Performances des deux méthodes

Les performances des deux méthodes seront comparées relativement aux trois critères suivants : les résultats obtenus, le coût et la maintenabilité.

En ce qui concerne les résultats obtenus, et dans le cas du dictionnaire de fautes, ceux-ci sont directement fonction de la structure du dictionnaire, qui est bien sûr figée. Si l'on prend l'exemple du dictionnaire décrit dans [RaK86], les connaissances sont basées uniquement sur les valeurs des sorties primaires, ce qui interdit de prendre en compte des informations sur d'autres points observables, comme celles fournies par le microscope électronique par exemple. Bien entendu, il aurait été possible de tenir compte de ceci, mais uniquement à l'étape de conception de l'architecture du dictionnaire.
Par contre, et ceci est une particularité inhérente aux systèmes à base de connaissances, les résultats obtenus ne sont pas fonction de la structure du système, mais des données qui lui sont fournies. L'exactitude des résultats obtenus dépendra donc de la qualité des données et

non de la méthode de déduction, comme pour le cas du dictionnaire.
Enfin, pour terminer avec ce point, il est intéressant de remarquer qu'avec le dictionnaire, le diagnostic est soit exact, soit impossible à fournir, tandis que le système à base de connaissances, s'il n'a pas assez de connaissances pour fournir un diagnostic exact, peut, néanmoins et dans certains cas, fournir un diagnostic approximatif à titre indicatif.

Le deuxième point de comparaison est le coût des deux méthodes, ou plutôt le "rapport qualité/prix". Si les systèmes à base de connaissances coûtent cher, pour leur définition et leur réalisation, les résultats qu'ils permettent d'obtenir amortissent pleinement cet investissement, surtout pour ce qui concerne les systèmes à base de connaissances d'aide au diagnostic qui ont déjà fait leurs preuves depuis près d'une vingtaine d'années.
Quant à la méthode utilisant un dictionnaire de fautes, les chiffres suivants, fournis par [RaK86] sont suffisamment édifiants : Les séries de tests qui ont été conduits indiquent que le coût de la seule génération du dictionnaire varie avec le nombre de fautes simulées. Ainsi, la génération d'un dictionnaire permettant d'identifier les erreurs de fabrication et d'assemblage des circuits augmente le temps CPU total de 15%, tandis que celle d'un dictionnaire permettant d'identifier des erreurs de conception augmente ce temps de 700%! (Rappelons qu'il faut exécuter cette génération pour chaque type de circuit à tester).
On peut également dire à ce propos qu'un même système à base de connaissances peut servir aussi bien à identifier des erreurs de conception que de fabrication ou d'assemblage des circuits, car on peut y prendre en compte différents modèles de fautes, voire des modèles peu ou mal définis, ou même des fautes non modélisées, alors qu'un dictionnaire est basé sur un modèle unique, choisi définitivement au moment de sa conception, ce modèle étant généralement le collage à une valeur logique donnée ("1" ou "0") des connexions du circuit [RiB85].

Le troisième et dernier critère de comparaison est la maintenabilité. Il est maintenant pratiquement inutile de s'appesantir sur les grandes facultés de maintenabilité et d'adaptabilité des systèmes à base de connaissances, celles-ci ayant fait l'objet de nombreux articles et autres études. Il suffit d'avoir à l'esprit que la maintenance d'un système à base de connaissances s'effectue simplement par ajout ou suppression de données qui constituent la base de connaissances, ce qui est très facilement faisable étant donnée la convivialité des systèmes à base de connaissances.
La maintenance d'un dictionnaire de fautes est par contre rendue très difficile par sa structure figée. Il convient de dire, enfin, que l'on peut augmenter les performances d'un système à base de connaissances par ajout d'heuristiques, qui constituent la métaconnaissance du système.

V.4.2 ADVICE vs. PESTICIDE

La comparaison entre l'utilisation d'un dictionnaire de fautes et l'utilisation d'un système à base de connaissances se fera plus spécifiquement dans ce paragraphe, puisque nous prendrons les exemples des systèmes ADVICE et PESTICIDE.

Dans la littérature traitant de ce sujet particulier, tous les auteurs [NoR72], [BrF76], [KLM81], [Fuj85], [RiB85], [RaK86], [KSK89], [WaL89], s'accordent pour remarquer qu'une simulation de *toutes* les fautes (même lorsque seul le modèle du collage à une valeur logique est pris en compte) possibles est non seulement absolument impraticable, mais également inefficace, car, dans la réalité, seul un nombre limité de fautes (mais de type imprévisible) est effectivement observé dans l'analyse d'un circuit.

Par conséquent, un certain nombre de techniques et méthodes sont mises en oeuvre pour éviter cette simulation de fautes complète, et générer des dictionnaires raisonnablement utilisables.

Dans le cas du système ADVICE, les partenaires chargés de la définition de cette tâche particulière ont choisi d'adopter les démarches suivantes :

- *Restriction des modèles de fautes aux collages, coupures, et court-circuits,* or il est largement reconnu (et ceci a encore été entériné récemment par [KSK89] et [WaL89]) que l'efficacité et la précision du diagnostic dépendent des modèles de fautes utilisés, de telle sorte que l'on ne peut en aucune façon obtenir une localisation correcte de la faute dans le circuit sous test si cette faute n'entre pas dans la catégorie des modèles pris en compte. Il n'est donc possible de garantir la *robustesse* du diagnostic qu'en ne formulant aucun présupposé quant aux fautes pouvant survenir.
  PESTICIDE, en considérant toute faute comme une divergence entre une valeur correcte attendue et une valeur erronée mesurée, assure un diagnostic robuste.

- *Réduction du nombre de fautes prises en compte ("fault collapsing"),* de façon à ne simuler, à l'aide des notions d'équivalence et de "dominance", qu'une faute par classe d'équivalence et qu'une faute par classe de "dominance". Mais le problème du diagnostic revient à une localisation *topologique* de la faute, alors que l'équivalence et la dominance de fautes sont des notions *fonctionnelles.* En d'autres termes, si la consultation du dictionnaire de fautes permet de déduire que la faute recherchée est la faute f_0, cela revient en réalité à déduire que la faute recherchée est contenue dans l'ensemble $\{f_0, \cdots, f_n\}$, où les fautes f_i, $1 \leq i \leq n$ soit sont équivalentes à f_0, soit dominent f_0, ce qui ne résout pas le problème de localisation.
  PESTICIDE, en ne se basant que sur une simulation du fonctionnement correct du circuit, et non sur une simulation de fautes, ne connait pas ce problème.

- *Adoption d'une approche SOFE ("Stop On First Error"),* or ce type de méthode de diagnostic utilisant un dictionnaire de fautes généré par approche SOFE reste limité [WaL89], car ce processus ne permet pas d'isoler une faute particulière. En effet, plusieurs fautes (qui ne sont pas forcément équivalentes, ni liées par une relation de "dominance") peuvent expliquer des divergences par rapport aux résultats de simulation attendus, et par conséquent il faut limiter cet ensemble de fautes possibles en appliquant d'autres vecteurs de test aux entrées primaires du circuit, pour restreindre

les suspects à l'intersection des différents ensembles ainsi obtenus. Un autre inconvénient de l'approche SOFE est qu'il n'est pas toujours possible de suffisamment confiner la zone dans laquelle apparaît la faute. Enfin, cette approche ne permet pas un diagnostic efficace de circuits autres que les collages logiques, et reste d'une utilité restreinte pour le diagnostic de fautes multiples.
PESTICIDE, encore une fois, évite de tels inconvénients puisque ce système n'effectue aucune simulation de fautes.

V.5 CONCLUSION

Notre objectif principal, dans ce chapitre, était de montrer que l'utilisation d'un dictionnaire de fautes pour le diagnostic de fautes dans les circuits intégrés est une méthode qui, si elle était viable pour les circuits et l'état de l'art des années 70, présente à l'heure actuelle trop d'inconvénients pour être applicable.
Lorsque cette méthode est de plus couplée à un outil de test sans contact présentant les capacités d'observabilité interne offertes par un testeur par faisceau d'électrons, à ces inconvénients s'ajoutent une inefficacité totale.
Par contre, des méthodes basées sur la connaissance peuvent permettre un diagnostic de circuits complexes non seulement réalisable, car ne requérant qu'un nombre de ressources somme toute raisonnable, mais également robuste.

Quoique anecdotique, la meilleure preuve en est que les tâches de génération et d'exploitation de dictionnaires de fautes dans le cadre d'ADVICE ne sont et ne seront pas réalisées, elles ont par contre été remplacées, dans le mode automatique, par des procédures de guidage de la sonde électronique mettant en oeuvre certains concepts définis dans PESTICIDE.

VI. CONCLUSION DE LA DEUXIEME PARTIE

Dans cette deuxième partie, les intérêts, limites et problèmes de la validation de prototypes de circuits intégrés observés au microscope électronique à balayage utilisé en mode contraste de potentiel ont été présentés.
Nous avons pu étudier deux approches de diagnostic dans le cadre d'un tel test, et ce travail s'est soldé d'une part par la conception du système PESTICIDE, et d'autre part par la contribution que nous avons apportée à la réalisation du système ADVICE.

Notre investissement à la fois dans l'un et l'autre de ces deux projets, s'il a parfois demandé beaucoup de compromis, n'en demeure pas moins enrichissant, peut-être justement à cause de la dualité qu'il impliquait.
En effet, la double investigation qui a été menée a, par la force des choses, dû l'être suffisamment profondément pour nous permettre d'évaluer avec le moins d'a priori possible les apports et lacunes des deux systèmes réalisés.

Nous restons tout à fait consciente du fait que ni l'un ni l'autre des deux systèmes n'a encore pu bénéficier d'une validation sérieuse et complète.
Il s'agit là d'un problème qui dépasse le cadre de ce travail, puisqu'il demande d'une part que toutes les composantes des deux systèmes soient réalisées et intégrées, ce qui n'est pas encore le cas, et d'autre part que de gros efforts et moyens soient fournis non seulement pour la description, la simulation, la simulation de fautes (pour ADVICE) et la génération de stimuli nécessaires à l'émulation du circuit sous test, mais aussi pour mettre en oeuvre les bonnes conditions d'observation et de mesure d'un circuit réellement complexe : fournir ces efforts et remplir ces conditions n'est pas toujours facile dans des environnements universitaires.

Au-delà de ces problèmes, il faut retenir la contribution des deux projets à l'avancement de l'état de l'art dans le domaine de la validation de prototypes de circuits intégrés observés au microscope électronique à balayage.
Nous avons pu voir que des "T.F.E. étendus" existent déjà sur le marché, et fonctionnent dans plusieurs centres industriels de test. D'ici peu, ces centres pourront certainement se doter de systèmes intégrés de test.

De plus en plus, les circuits complexes sont conçus à l'aide d'outils de CAO puissants et robustes : peu d'erreurs de conception peuvent persister.
De plus en plus, ces circuits sont testés à l'aide de méthodes et procédures sophistiquées : peu de fautes de fabrication peuvent passer au travers de ces cribles.
Mais de plus en plus, les exigences de qualité se font grandes dans des domaines sensibles,

sinon critiques, puisqu'ils mettent en jeu la fiabilité et la sécurité des circuits et des machines, donc des personnes qui les utiliseront.

Le seul moyen de maîtriser la complexité des circuits intégrés actuels et à venir, dans le but d'assurer ces objectifs de qualité se résume dans les voies de recherche suivantes :

- Utiliser des spécifications de très haut niveau comme référentiel
- Développer des méthodes de haut niveau pour le diagnostic, introduisant des raisonnements qualitatifs
- Dépasser les modèles de fautes actuels, peu représentatifs de fautes résiduelles imprévisibles
- Concevoir des environnements de travail qui, des premières étapes de la conception jusqu'à la certification finale, puissent être cohérents et harmonieux dans les concepts qu'ils utilisent.

Les approches à base de connaissances, et notamment les systèmes à base de connaissances de seconde génération, peuvent aider à la réalisation de ces objectifs.

Cette conclusion rejoint l'analyse présentée en première partie du document, et nous permet de constater l'évolution parallèle des outils et des méthodes de test.
Des possibilités d'observabilité interne comme celles offertes par les testeurs à faisceau d'électrons peuvent limiter en grande partie les problèmes d'observation de circuits à haut degré d'intégration, mais présentent l'inconvénient de générer une masse d'information trop importante pour pouvoir être gérée manuellement par l'utilisateur.
L'interprétation automatique des données observées, complexes par leur quantité et leur nature, requiert donc des méthodologies adéquates. Montrer que ces méthodologies doivent - et peuvent - être suffisamment souples pour pouvoir prendre en compte différents niveaux d'abstraction, de spécification et de représentation a constitué l'objet de notre thèse.

Bibliographie

[ABM] G. Arato, G. Bussolino et R. Manione, **ACCORDO : second generation floor planning**, Rapport de Recherche No , CSELT, Turin.

[ACP88] M. Ayel, M. Chein, E. Pipard et M. C. Rousset, **De la cohérence dans les bases de connaissances**, Actes des Journées Nationales du PRC-GRECO IA, Toulouse, Mars 1988, 357-384.

[BCL82] G. Baille, B. Courtois et J. Laurent, **Analyse de défaillances de VLSI par microscopie électronique**, Rapport de Recherche No 304, TIM3-IMAG, Grenoble, Juin 1982.

[BaL82] G. Baille et J. Laurent, **Etude du comportement interne des circuits VLSI par microscopie électronique**, Congrès AFCET : architecture des machines et systèmes informatiques, Lille, Nov. 1982.

[BBC83] G. Baille, L. Bergher, B. Courtois, J. Laurent et C. Rubat Du Mérac, **Testing for failure analysis : new tools and new test methods**, 13th IEEE Fault Tolerant Computing Symposium, Milan, Juin 1983, 266-269.

[Bai84] G. Baille, **Testeur logique de circuits intégrés**, Tech. Rep. 84, TIM3-IMAG, Grenoble, 1984.

[BCD88] G. Baille, F. Conquet, B. Devautour et T. Gouyon, **Utilisation à distance d'un testeur à faisceau d'électrons**, Rapport de Recherche No 760-I, TIM3-IMAG, Grenoble, Oct. 1988.

[BGV88] M. Battu, P. Garino, Y. J. Vernay et J. Dowe, **Extraction of coordinates of nodes to be measured from CAD data (ADVICE C.I.R.)**, Tech. Rep. 7, ESPRIT I-C.C.E., Bruxelles, Mai 1988.

[Bau85] F. Baudrand, **Présentation de TESSIE : testeur de circuits intégrés**, Rapport de Recherche No 650 729, TIM3-IMAG, Grenoble, Fev. 1985.

[Ben89] M. Ben Ali, **Interface PESTICIDE-HILO3**, [Mem. d'Ingenieur, F.S.Tunis], Grenoble, Juin 1989.

[Ben90] R. L. Ben Dakhlia, **Stratégies de test et génération de connaissances pour un système expert**, [Mem. d'Ingenieur, F.S.Tunis], Grenoble, Juil. 1990.

[Ber85] L. Bergher, **Analyse de défaillances de circuits VLSI par microscopie électronique à balayage**, [Th. de Docteur Ingenieur, I.N.P.Grenoble], Grenoble, Juin 1985.

[BeH82] J. M. Bernard et J. Hugon, **De la logique câblée aux microprocesseurs - T. 3 : méthodes de conception de systèmes**, Eyrolles, Paris, 1982.

[BoH87] F. M. Boland et K. Hundertpfund, **Automatic logic waveform comparison (ADVICE C.I.R.)**, Tech. Rep. 6, ESPRIT I-C.C.E., Bruxelles, Dec. 1987.

[BrF76] M. A. Breuer et A. D. Friedman, **Diagnosis and reliable design of digital systems**, Pitman, California, 1976.

[ClM85] W. F. Clocksin et C. S. Mellish, **Programmer en Prolog**, Eyrolles, Paris, 1985.

[CoM87] M. Cocito et M. Melgara, **ADVICE : a european effort towards automatic e-beam testing**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 235-241.

[Col83] J. P. Collin, **Une alternative économique au contraste de potentiel stroboscopique : le traitement du signal d'électrons secondaires d'un microscope à balayage**, Journées d'Electronique - EPFL , (Oct. 1983), 283-297.

[CCC89] J. P. Collin, D. Conard, B. Courtois, P. Denis et D. Savart, **Failure analysis using e-beam**, 2nd European Conference on Electron and Optical Beam Testing of Integrated Circuits, Duisburg, Oct. 1989.

[CoC89] D. Conard et B. Courtois, **Failure analysis using e-beam : a fully automatic process**, Colloque ADESO : Qualité des composants électroniques, Bordeaux, Avr. 1989.

[CLL86] S. Concina, G. Liu, L. Lattanzi, S. Reyfman et N. Richardson, **Software integration in a workstation-based e-beam tester**, 17th IEEE International Test Conference, Washington, Sep. 1986, 644-649.

[CoR87a] S. Concina et N. Richardson, **Workstation-driven e-beam prober**, 18th IEEE International Test Conference, Washington, Sep. 1987, 554-560.

[CoR87b] S. Concina et N. Richardson, **IDS 5000 : an integrated diagnostic system for VLSI**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 339-342.

[CoL87] S. Concina et G. Liu, **Integrating design information for IC diagnosis**, 24th ACM/IEEE Design Automation Conference, Floride, Juin 1987, 251-257.

[CRP88] S. Concina, N. Richardson, J. L. Pelissier et X. Larduinat, **The SCHLUMBERGER ATE IDS 5000 for VLSI debug and validation applications**, IFIP workshop on knowledge-based systems for test and diagnosis, Grenoble, Sep. 1988.

[CMN82] B. Courtois, P. Marchal et M. Nicolaidis, **Design of testable microprocessors**, Test Technology Newsletter j-82, (Juil. 1982), .

[Dav84] R. Davis, **Diagnostic based on structure and behavior**, Artificial Intelligence 1, 24 (1984), 347-410.

[Den68] J. J. Dent, **Diagnostic engineering requirements**, AFIPS, , 1968, 503-507.

[Dow87] J. Dowe, **Generating a fault dictionary for the ADVICE project (ADVICE C.I.R.)**, Tech. Rep. 6, ESPRIT I-C.C.E., Bruxelles, Dec. 1987.

[FFW81] P. Fazekas, H. P. Feuerbaum et E. Wolfgang, **Scanning electron beam probes VLSI chips**, Electronics 54, 14 (Juil. 1981), 105-112.

[Fuj85] H. Fujiwara, **Logic testing and design for testability**, The MIT Press, Cambridge, 1985.

[Gen85] GenRad, **Hilo-3 User Manual**, Tech. Rep. 85, GenRad Inc., Milpitas, 1985.

[GHK87] S. Gorlich, H. Harbeck, P. Kebler, E. Wolfgang et K. Zibert, **Integration of CAD, CAT and electron-beam testing for IC-internal logic verification**, 18th IEEE International Test Conference, Washington, Sep. 1987, 566-574.

[GHK88] S. Gorlich, H. Harbeck, P. Kebler, E. Wolfgang et K. Zibert, **Speeding IC design verification**, Test & measurement World s-88, (Sep. 1988), 91-101.

[Gui85] I. Guiguet, **Liaison d'un outil de description des circuits intégrés à un microscope électronique à balayage**, [Mem. d'Ingenieur, CNAM], Grenoble, Oct. 1985.

[GML86] I. Guiguet, D. Micollet, J. Laurent et B. Courtois, **Electron-beam observability and controlability for the debugging of integrated circuits**, European Solid-State CIRcuits Conference, Delft, Sep. 1986.

[GMC87] I. Guiguet, M. Marzouki et B. Courtois, **An integrated debugging system based on e-beam test**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical beam Testing of Integrated Circuits) 7, 2-4 (1987), 275-282.

[GMM88] I. Guiguet, R. Manione, M. Marzouki et M. Melgara, **Process method description (ADVICE C.I.R.)**, Tech. Rep. 7, ESPRIT I-C.C.E., Bruxelles, Mai 1988.

[HKP87] S. S. Henning, W. R. Knowles et G. S. Plows, **CAD system interface for a stand-alone e-beam tester**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 317-325.

[Jer89] A. A. Jerraya, **Contribution à la compilation de silicium et au compilateur SYCO**, [Th. de Doctorat d'Etat, I.N.P.Grenoble], Grenoble, Dec. 1989.

[Joh75] S. C. Johnson, **YACC : Yet Another Compiler Compiler**, Tech. Rep. 31, Bell Laboratories, Murray Hill, 1975.

[KSK89] J. Kato, T. Shimono et M. Kawai, **Fault diagnosis based on post-test fault dictionary**, 20th IEEE International Test Conference, Washington, Aout 1989, 940.

[KLM81] S. Kochan, N. Landis et D. Monson, **Computer-guided probing techniques**, 12th IEEE International Test Conference, Philadelphie, Oct. 1981, 253-268.

[Kod86] Y. Kodratoff, **Is A. I. a sub-field of computer science? or A. I. is the science of explanations**, Rapport de Recherche No 312, LRI, Orsay, Nov. 1986.

[KMS86] F. Komatsu, M. Miyoshi, T. Sano, K. Sekiwa et K. Okumura, **Electron-beam tester with a CAD pattern data**, International Congress on Electron Microscopy, Kyoto, 1986.

[KMS87] F. Komatsu, M. Miyoshi, T. Sano et K. Okumura, **An electron beam test system linked with a CAD database**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 267-274.

[KuT86] N. Kuji et T. Tamama, **An automated e-beam tester with CAD interface : FINDER**, 17th IEEE International Test Conference, Washington, Sep. 1986, 857-863.

[Lap85] J. C. Laprie, **Dependable computing and fault tolerance : concepts and terminology**, 15th IEEE Fault Tolerant Computing Symposium, Michigan, Juin 1985, 2-11.

[LaC83] J. Laurent et B. Courtois, **Définition et utilisation d'un outil de test de VLSI par faisceau d'électrons**, Rapport de Recherche No 374, TIM3-IMAG, Grenoble, Mai 1983.

[Lau84] J. Laurent, **Projet ACIME : Analyse des Circuits Intégrés par Microscopie Electronique**, [Th. de Doctorat, I.N.P.Grenoble], Grenoble, Oct. 1984.

[Lau88] J. P. Laurent et al., **Schéma pour la description et l'évaluation de systèmes experts et d'outils de développement de systèmes experts**, Actes des Journées Nationales du PRC-GRECO IA, Toulouse, Mars 1988, 385-434.

[Lau82a] J. L. Laurière, **Représentation et utilisation des connaissances (1)**, Techniques et Sciences Informatiques 1, 1 (1982), 25-42.

[Lau82b] J. L. Laurière, **Représentation et utilisation des connaissances (2)**, Techniques et Sciences Informatiques 1, 2 (1982), 109-132.

[Lee87] W. Lee, **A context-sensitive help system based on hypertext**, 24th ACM/IEEE Design Automation Conference, Floride, Juin 1987, 429-435.

[LLP87] W. Lee, G. Liu et K. Peterson, **TED : a graphical technology description editor**, 24th ACM/IEEE Design Automation Conference, Floride, Juin 1987, 423-428.

[Lee89] W. T. Lee, **Engineering a device for electron-beam probing**, IEEE Design and Test of Computers 6, 3 (Juin 1989), 36-49.

[LeS75] M. E. Lesk et S. Schmidt, **LEX : a lexical analyser generator**, Tech. Rep. 32, Bell Laboratories, Murray Hill, 1975.

[LSU89] R. Lipsett, C. Schaefer et G. Ussery, **VHDL: Hardware Description and Design**, Kluwer Academic Publishers, Massachussetts, 1989.

[Luk87] G. V. Lukianoff, **History of scanning electron beam testing development**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 115-129.

[LyB87] E. R. Lynch et F. M. Boland, **Waveform parameter extraction in e-beam testing**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 195-199.

[Mac84] E. J. Mac Cluskey, **Verification testing - a pseudo-exhaustive test technique**, IEEE Transactions on Computers c-33, 6 (Juin 1984), 541-546.

[MRP87] D. J. Machin, D. W. Ranasinghe et G. Proctor, **A high speed signal averager for electron beam test systems**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 201-207.

[MaM87] M. Marzouki et M. Melgara, **Assessment of the simulation problem (ADVICE C.I.R.)**, Tech. Rep. 5, ESPRIT I-C.C.E., Bruxelles, Mai 1987.

[Mar87b] M. Marzouki, **Assessment of the TPG problem (ADVICE C.I.R.)**, Tech. Rep. 6, ESPRIT I-C.C.E., Bruxelles, Dec. 1987.

[Mar87a] M. Marzouki, **Définition et réalisation d'un système expert de localisation d'erreurs de conception des circuits intégrés**, [DEA Informatique, I.N.P.Grenoble], Grenoble, Juin 1987.

[MaC88] M. Marzouki et B. Courtois, **PESTICIDE : a Prolog-written Expert System as a Tool for Integrated CIrcuits DEbugging**, Rapport de Recherche No

691-I, TIM3-IMAG, Grenoble, Jan. 1988.

[MLC89] M. Marzouki, J. Laurent et B. Courtois, **A unified use of deep and shallow knowledge in an expert system for prototype validation of integrated circuits**, , Avignon, Mai 1989.

[MaC89] M. Marzouki et B. Courtois, **Debugging integrated circuits : A. I. can help**, European Test Conference, IEEE Computer Society Press, Paris, Avr. 1989, 184-191.

[MSM84] T. C. May, G. L. Scott, E. S. Meieran, P. Winer et V. R. Rao, **Dynamic fault imaging of VLSI random logic devices**, International Reliability Physics Conference, , 1984.

[MCM87] M. Melgara, B. Courtois, M. Marzouki, D. Micollet et Y. J. Vernay, **Design for electron-beam debugging (ADVICE C.I.R.)**, Tech. Rep. 5, ESPRIT I-C.C.E., Bruxelles, Mai 1987.

[Mel87a] M. Melgara, **Task C : work program definition (ADVICE C.I.R.)**, Tech. Rep. 5, ESPRIT I-C.C.E., Bruxelles, Mai 1987.

[Mel87b] M. Melgara, **General consideration of fault insertion - simulation problems**, Tech. Rep. 6, ESPRIT I-C.C.E., Bruxelles, Dec. 1987.

[MBG88] M. Melgara, M. Battu, P. Garino, F. Boland, J. Dowe, M. Marzouki et Y. J. Vernay, **Automatic location of IC design errors using an electron beam system**, 19th IEEE International Test Conference, Washington, Sep. 1988, 898-907.

[Mic88] D. Micollet, **Etude de la contrôlabilité des circuits intégrés par faisceaux d'électrons**, [Th. de Doctorat, I.N.P.Grenoble], Grenoble, Sep. 1988.

[MuH90] B. T. Murray et J. P. Hayes, **Hierarchical test generation using precomputed tests for modules**, IEEE Transactions on Computer Aided Design 9, 6 (Juin 1990), 594-602.

[NoR72] K. Nozawa et K. Ritani, **FACOM 230-60 diagnostic program**, 2nd IEEE Fault Tolerant Computing Symposium, Newton, Juin 1972, 68-72.

[RaK86] V. Ratford et P. Keating, **Integrating guided probe and fault dictionary : an enhanced approach**, 17th IEEE International Test Conference, Washington, 1986, 304-311.

[Ric87] N. Richardson, **E-beam probing for VLSI circuit debug**, VLSI Systems Design 8, 9 (Aout 1987), 24-29.

[RiB85] J. Richman et K. R. Bowden, **The modern fault dictionary**, 16th IEEE International Test Conference, Philadelphie, Nov. 1985, 696-702.

[RJL83] C. Rubat Du Mérac, P. Jutier, J. Laurent et B. Courtois, **A new domain for image analysis : VLSI circuits testing, with ROMUALD, specialized in parallel image processing**, British Pattern Recognition Association, Oxford, Sep. 1983.

[SMB90] D. G. Saab, R. B. Mueller-Thuns, D. Blaauw, J. T. Rahmeh et J. A. Abraham, **Hierarchical multi-level fault simulation of large systems**, Journal of Electronic Testing : Theory and Applications 1, 2 (Mai 1990), 139-149.

[SaC87] D. Savart et B. Courtois, **Automatic failure analysis of VLSI circuits using e-beam**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 259-266.

[Sav90] D. Savart, **Analyse de défaillances de circuits intégrés VLSI par testeur à faisceau d'électrons**, [Th. de Doctorat, I.N.P.Grenoble], Grenoble, Juin 1990.

[ShM75] J. J. Shedletsky et E. J. Mac Cluskey, **The error latency of a fault in a combinational digital circuit**, 5th IEEE Fault Tolerant Computing Symposium, , Juin 1975, 210-214.

[StT87] F. W. M. Stentiford et T. J. Twell, **Automatic registration of scanning electron microscope images**, The Microelectronic Engineering Journal (Special Issue on Electron and Optical Beam Testing of Integrated Circuits) 7, 2-4 (1987), 215-221.

[TaK86] T. Tamama et N. Kuji, **Fully automated electron beam tester combined with CAD database**, International Congress on Electronic Microscopy, Kyoto, 1986.

[UrF89] K. Ura et H. Fujioka, **Electron beam testing**, Advances in Electronics and Electron Physics 73, (1989), 233-317.

[Wad78] R. L. Wadsack, **Fault modelling and logic simulation of CMOS and MOS integrated circuits**, The Bell System Technical Journal 57, 5 (Mai 1978), .

[WaL89] J. A. Waicukauski et E. Lindbloom, **Failure diagnosis of structured VLSI**, IEEE Design and Test of Computers 6, 4 (Aout 1989), 49-60.

[Wes78] J. S. Weszka, **A survey of threshold selection techniques**, Computer vision, graphics and image processing 7, 1 (1978), 259-265.

[WiP82] T. W. Williams et K. P. Parker, **Design for Testability : a survey**, IEEE Transactions on Computers c-31, 1 (Jan. 1982), 2-15.

[YaO87] T. Yano et H. Okamoto, **Fast fault diagnostic method using fault dictionary for electron beam tester**, 18th IEEE International Test Conference, Washington, Sep. 1987, 561-565.

[Zhu89] C. Zhu, **CTEDIF : compilateur et traducteur de fichiers EDIF pour le testeur TESSIE**, [Rapport de stage, TIM3-IMAG], Grenoble, Sep. 1989.

'NSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Grenoble, le 03 Décembre 1990

)ÉPARTEMENT DES ÉTUDES DOCTORALES

ffaire suivie par
él : 76.57.

/Réf. :

)bjet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

- Monsieur COSTES
- Monsieur ABRAHAM

Madame MARZOUKI Meryem

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 : "Informatique"

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