Compilation de programmes VHDL en vue de l'évaluation de testabilité d'équipements digitaux

Pierre Wodey

THESE

présentée par

Pierre WODEY

pour obtenir le grade de DOCTEUR

de L'Institut National Polytechnique de Grenoble

(Arrêté ministériel du 30 mars 1992)

Spécialité : Informatique

Compilation de Programmes VHDL en vue de L'Evaluation de Testabilité d'Equipements Digitaux.

Date de Soutenance : mercredi 3 novembre 1993

Composition du jury :	Chantal	ROBACH	directeur
	Alain	COSTES	rapporteur
	Christian	LANDRAULT	rapporteur
	Paul	JACQUET	examinateur
	Guy	MAZARE	président

Thèse préparée au sein du Laboratoire de Génie Informatique

PRESIDENT DE L'INSTITUT
Monsieur Maurice RENAUD

Année1993

PROFESSEURS DES UNIVERSITES

BARIBAUD	Michel	ENSERG
BARRAUD	Alain	ENSIEG
BARTHELEMY	Alain	ENSHMG
BAUDELET	Bernard	ENSPG
BAUDIN	Gérard	UFR PGP
BEAUFILS	Jean-Pierre	ENSIEG/ILL
BOIS	Philippe	ENSHMG
BOUVIER	Gérard	ENSERG
BRINI	Jean	ENSERG
BRUNET	Yves	CUEFA
CAVAIGNAC	Jean-François	ENSPG
CHARTIER	Germain	ENSPG
CHENEVIER	Pierre	ENSERG
CHERUY	Arlette	ENSIEG
CHOVET	Alain	ENSERG
COGNET	Gérard	ENSGI
COLINET	Catherine	ENSEEG
COMMAULT	Christian	ENSIEG
CORNUT	Bruno	ENSIEG
COULOMB	Jean-Louis	ENSIEG
COUTRIS	Nicole	ENSPG
CROWLEY	James	ENSIMAG
DALARD	Francis	ENSEEG
DARVE	Félix	ENSHMG
DELLA DORA	Jean	ENSIMAG
DEPEY	Maurice	ENSERG
DEPORTES	Jacques	ENSPG
DEROO	Daniel	ENSEEG
DESRE	Pierre	ENSEEG
DIARD	Jean-Paul	ENSEEG
DOLMAZON	Jean-Marc	ENSERG
DURAND	Francis	ENSEEG
DURAND	Jean-Louis	ENSPG
FAUTRELLE	Yves	ENSHMG
FOGGIA	Albert	ENSIEG
FORAY	Pierre	ENSHMG
FOULARD	Claude	ENSIEG
GALERIE	Alain	ENSEEG
GANDINI	Alessandro	UFR/PGP
GAUBERT	Claude	ENSPG
GENTIL	Pierre	ENSERG
GENTIL	Sylviane	ENSIEG
GUERIN	Bernard	ENSERG
GUYOT	Pierre	ENSEEG
IVANES	Marcel	ENSIEG
JACQUET	Paul	ENSIMAG
JALLUT	Christian	ENSEEG
JANOT	Marie-Thérese	ENSERG

JAULENT	Patrick	ENSGI
JAUSSAUD	Pierre	ENSIEG
JOST	Rémy	ENSPG
JOUBERT	Jean-Claude	ENSPG
JOURDAIN	Geneviéve	ENSIEG
KUENY	Jean-Louis	ENSHMG
LACHENAL	Dominique	UFR PGP
LACOUME	Jean-Louis	ENSIEG
LADET	Pierre	ENSIEG
LE NEST	Jean-François	UFR/PGP
LESIEUR	Marcel	ENSHMG
LESPINARD	Georges	ENSHMG
LIENARD	Joël	ENSIEG
LONGEQUEUE	Jean-Pierre	ENSPG
LORET	Benjamin	ENSHMG
LOUCHET	François	ENSEEG
LUCAZEAU	Guy	ENSEEG
LUX	Augustin	ENSIMAG
MASSE	Philippe	ENSPG
MASSELOT	Christian	ENSIEG
MAZARE	Guy	ENSIMAG
MICHEL	Gérard	ENSIMAG
MOHR	Roger	ENSIMAG
MOREAU	René	ENSHMG
MORET	Roger	ENSIEG
MOSSIERE	Jacques	ENSIMAG
OBLED	Charles	ENSHMG
OZIL	Patrick	ENSEEG
PANANAKAKIS	Georges	ENSERG
PAULEAU	Yves	ENSEEG
PERRET	Robert	ENSIEG
PERRIER	Pascal	ENSERG
PIAU	Jean-Michel	ENSHMG
PIC	Etienne	ENSERG
PLATEAU	Brigitte	ENSIMAG
POUPOT	Christian	ENSERG
RAMEAU	Jean-Jacques	ENSEEG
REINISCH	Raymond	ENSPG
RENAUD	Maurice	UFR/PGP
RIMET	Roger	ENSERG
ROBERT	François	ENSIMAG
ROGNON	Jean-Pierre	ENSIEG
ROSSIGNOL	Michel	ENSPG
ROYE	Daniel	ENSIEG
SABONNADIERE	Jean-Claude	ENSIEG
SAGUET	Pierre	ENSERG
SAUCIER	Gabriéle	ENSIMAG
SCHLENKER	Claire	ENSPG
SCHLENKER	Michel	ENSPG
SILVY	Jacques	UFR/PGP
SOHM	Jean-Claude	ENSEEG
SOLER	Jean-Louis	ENSIMAG
SOUQUET	Jean-Louis	ENSEEG
TICHKIEWITCH	Serge	ENSHMG
TROMPETTE	Philippe	ENSHMG
TRYSTRAM	Denis	ENSGI
VEILLON	Gérard	ENSIMAG
VERJUS	Jean-Pierre	ENSIMAG
VINCENT	Henri	ENSPG

SITUATION PARTICULIERE

PROFESSEURS D'UNIVERSITE

DETACHEMENT

BLOCH Daniel	ENSPG
BONNET Guy	ENSPG
BRECHET Yves	ENSEEG
CAILLERIE Denis	ENSHMG
GREVEN Héléne	CUEFA
LATOMBE Jean-Claude	ENSIMAG
PIERRARD Jean-Marie	ENSHMG

DIRECTEURS DE RECHERCHE CNRS

ABELLO	Louis
ALDEBERT	Pierre
ALEMANY	Antoine
ALLIBERT	Colette
ALLIBERT	Michel
ANSARA	Ibrahim
ARMAND	Michel
AUDIER	Marc
AUGOYARD	Jean-François
AVIGNON	Michel
BERNARD	Claude
BINDER	Gilbert
BLAISING	Jean-Jacques
BONNET	Roland
BORNARD	Guy
BOUCHERLE	Jean-Xavier
CAILLET	Marcel
CARRE	René
CHASSERY	Jean-Marc
CHATILLON	Christian
CIBERT	Joël
CLERMONT	Jean-Robert
COURTOIS	Bernard
CRIQUI	Patrick
CRISTOLOVEANU	Sorin
DAVID	René
DION	Jean-Michel
DOUSSIERE	Jacques
DRIOLE	Jean
DUCHET	Pierre
DUGARD	Luc
DURAND	Robert
ESCUDIER	Pierre
EUSTATHOPOULOS	Nicolas
FINON	Dominique
FRUCHARD	Robert
GARNIER	Marcel
GIROD	Jacques
GLANGEAUD	François
GUELIN	Pierre
HOPFINGER	Emil
JORRAND	Philippe
JOUD	Jean-Charles
KAMARINOS	Georges
KLEITZ	Michel
KOFMAN	Walter
LACROIX	Claudine
LANDAU	Ioan
LAULHERE	Jean-Pierre
LEGRAND	Michel
LEJEUNE	Gérard
LEPROVOST	Christian
MADAR	Roland
MARTIN	Jean-Marie
MERMET	Jean

MEUNIER	Gérard
MICHEL	Jean-Marie
NAYROLLES	Bernard
PASTUREL	Alain
PEUZIN	Jean-Claude
PHAM	Antoine
PIAU	Monique
PIQUE	Jean-Paul
POINSIGNON	Christiane
PREJEAN	Jean-Jacques
RENOUARD	Dominique
SENATEUR	Jean-Pierre
SIFAKIS	Joseph
SIMON	Jean-Paul
SUERY	Michel
TEODOSIU	Christian
VACHAUD	Georges
VAUCLIN	Michel
WACK	Bernard
YAVARI	Ali-Reza
YONNET	Jean-Paul

PERSONNES AYANT OBTENU LE DIPLOME

D'HABILITATION A DIRIGER DES RECHERCHES

BALESTRA	Francis
BALME	Louis
BECKER	Monique
BIGEON	Jean
BINDER	Zdeneck
BOE	Louis-Jean
BRECHET	Yves
CADOZ	Claude
CANUDAS DE WIT	Carlos
CHAMPENOIS	Gérard
CHOLLET	Jean-Pierre
COEY	Jean-Pierre
CORNUEJOLS	Gerard
COURNIL	Michel
CRASTES DE PAULET	Michel
DALLERY	Yves
DESCOTES-GENON	Bernard
DUGARD	Luc
DURAND	Madeleine
FERRIEUX	Jean-Paul
FEUILLET	René
FORAY	Pierre
FREIN	Yannick
GAUTHIER	Jean-Paul
GHIBAUDO	Gérard
GUILLEMOT	Nadine
GUYOT	Alain
HAMAR	Sylviane
HAMAR	Roger
HORAUD	Patrice
JACQUET	Paul
LATOMBE	Claudine
LE HUY	Hoang
LE GORREC	Bernard
LOZANO-LEAL	Rogelio
MACOVSCHI	Mihaïl
MAHEY	Philippe
METAIS	Olivier
MONMUSSON-PICQ	Georgette
MORY	Mathieu
MULLER	Jean
MULLER	Jean-Michel
NGUYEN TRONG	Bernadette
NIEZ	Jean-Jacques
PERRIER	Pascal
PLA	Fernand
RECHENMANN	François
ROGNON	Jean-Pierre
ROUGER	Jean
ROUX	Jean-Claude
SKOTNICKI	Tomasz
TCHUENT	Maurice
THOMAS	Olivier
VAHLAS	Constantin

PERSONNES AYANT OBTENU LE DIPLOME

DE DOCTEUR D'ETAT INPG

ABDEL-RAZEK	Adel
AKSAS	Haris
ALLA	Hassane
AMER	Ahmed
ANCELLE	Bernard
ANGENIEUX	Gilbert
ATMANI	Hamid
AYEDI	Hassine Feri
A.BADR	Osman
BACHIR	Aziz
BALANZAT	Emmanuel
BALTER	Roland
BARDEL	Robert
BARRAL	Gérard
BAUDON	Yves
BAUSSAND	Patrick
BEAUX	Jacques
BEGUINOT	Jean
BELLISSENT née FUNEZ	Marie-Claire
BELLON	Catherine
BEN RAIS	Abdejettah
BERGER-SABBATEL	Gilles
BERNACHE-ASSOLANT	Didier
BEROVAL	Abderrahmane
BERTHOD	Jacques
BILLARD	Dominique
BLANC épouse FOULETIER	Mireille
BOCHU	Bernard
BOJO	Gilles
BOKSENBAUM	Claude
BOLOPION	Alain
BONNARD	Bernard
BORRIONE	Dominique
BOUCHACOURT	Michel
BRION	Bernard
CAIRE	Jean-Pierre
CAMEL	Denis
CAPERAN	Philippe
CAPLAIN	Michel
CAPOLINO	Gérard
CASPI	Paul
CHAN-TUNG	Nam
CHASSANDE	Jean-Pierre
CHATAIN	Dominique
CHEHIKIAN	Alain
CHIRAMELLA	Yves
CHILO	Jean
CHUPIN	Jean-Claude
COLONNA	Jean-François
COMITI	Jacques
CORDET	Christian
COUDURIER	Lucien
COUTAZ	Jean-Louis

CREUTIN	Jean-Dominique
DAO	Trongtich
DARONDEAU	Philippe
DAVID	Bertrand
DE LA SEN	Manuel
DELACHAUME	Jean-Claude
DENAT	André
DESCHIZEAUX née CHERUY	Marie-Noëlle
DIJON	Jean
DOREMUS	Pierre
DUPEUX	Michel
EL ADHAM	Karim
EL OMAR	Fovaz
EL-HENNAWY	Adel
ETAY	Jacqueline
FABRE	Suzanne
FAURE-BONTE	Mireille
FAVIER	Denis
FAVIER	Jean-Jacques
FELIACHI	Movloud
FERYAL	Haj Hassan
FLANDRIN	Patrick
FOREST	Bernard
FORESTIER	Michel
FOSTER	Panayolis
FRANC	Jean-Pierre
GADELLE	Patrice
GARDAN	Yvon
GENIN	Jacques
GERVASON	Georges
GILORMINI	Pierre
GINOUX	Jean-Louis
GOUMIRI	Louis
GROC	Bernard
GROSJEAN	André
GUEDON	Jean-Yves
GUERIN	Jean-Claude
GUESSOUS	Anas
GUIBOUD-RIBAUD	Serge
HALBWACHS	Nicolas
HAMMOURI	Hassan
HEDEIROS SILIVEIR	Hamilton
HERAULT	Jeanny
HONER	Claude
HUECKEL	Tomasz
IGNAT	Michel
ILIADIS	Athananios
JANIN	Gérard
JERRAYA	Ahmed Amine
JUTTEN	Christian
KAHIL	Hassan
KHUONGQUANG	Dong
KILLIS	Andreas
KONE	Ali
LABEAU	Michel
LACAZE	Alain
LACROIX	Jean-Claude
LANG	Jean-Claude
LATHUILLERE	Chantal
LATY	Pierre

LAUGIER	Christian
LE CADRE	Jean-Pierre
LE GARDEVR	René
LE THIESSE	Jean-Claude
LEMAIGNAN	Clement
LEMUET	Daniel
LEVEQUE	Jean-Luc
LONDICHE	Henry
L'HERITIER	Philippe
MAGNIN	Thierry
MAISON	François
MAMWI	Abdullah
MANTEL épouse SIEBERT	Elisabeth
MARCON	Guy
MARTINEZ	Francis
MARTIN-GARIN	Lionel
MASSE	Dominique
MAZER	Emmanuel
MERCKEL	Gérard
MEUNIER	Jean
MILI	Ali
MOALLA	Mohamed
MODE	Jean-Michel
MONLLOR	Christian
MONTELLA	Claude
MORET	Frédéric
MRAYATI	Mohammed
M'SAAD	Mohammed
M'SIRDI	Kouider Nace
NEPOMIASTCHY	Pierre
NGUYEN	Trong Khoi
NGUYEN-XUAN-DANG	Michel
ORANIER	Bernard
ORTEGA MARTINEZ	Roméo
PAIDASSI	Serge
PASSERONE	Alberto
PEGON	Pierre
PIJOLAT	Christophe
POGGI	Yves
POIGNET	Jean-Claude
PONS	Michel
POU	Tong Eck
RAFINEJAD	Paiviz
RAGAIE	Harie Fikri
RAHAL	Salah
RAMA SEABRA SANTOS	Fernando
RAVAINE	Denis
RAZBAN-HAGHIGHI	Tchanguiz
RAZZOUK	Micham
REGAZZONI	Gilles
RIQUET	Jean-Pierre
ROBACH	Chantal
ROBERT	Yves
ROGEZ	Jacques
ROHMER	Jean
ROUSSEL	Claude
SAAD	Abdallah
SAAD	Youcef
SABRY	Mohamed Nabi
SALON	Marie-Christine

SAUBAT épouse MARCUS	Bernadette
SCHMITT	Jean-Hubert
SCHOELLKOPF	Jean-Pierre
SCHOLL	Michel
SCHOLL	Pierre-Claude
SCHOULER	Edmond
SCHWARTZ	Jean-Luc
SEGUIN	Jean
SIWY	Jacques
SKALLI	Abdellatif
SKALLI HOUSSEYNI	Abdelali
SOUCHON	Alain
SUETRY	Jean
TALLAJ	Nizar
TEDJAR	Farouk
TEDJINI	Smail
TEYSSANDIER	Francis
THEVENODFOSSE	Pascale
TMAR	Mohamed
TRIOLLIER	Michel
TUFFELIT	Denis
TZIRITAS	Georges
VALLIN	Didier
VELAZCO	Raoul
VERDILLON	André
VERMANDE	Alain
VIKTOROVITCH	Pierre
VITRANT	Guy
WEISS	François
YAZAMI	Rachid

A

Salomé et Jeanne.
Paul , Claire, Alain, Véronique et Christine.
Thierry, Daniel, Bernard, Brigitte...
Eric et Jean-luc.
Jeanne

Remerciements

Je tiens à faire part de mes vifs remerciements à Monsieur Guy MAZARÉ, Professeur à l'ENSIMAG, qui m'a fait l'honneur de présider le jury de ma thèse.

A Monsieur Christian LANDRAULT, Directeur de Recherche au CNRS, j'adresse des remerciements particuliers pour avoir accepté d'être rapporteur de mon mémoire de thèse. Son analyse approfondie du document et ses critiques constructives ont beaucoup aidé à en améliorer la qualité.

J'adresse mes remerciements sincères à Monsieur Alain COSTE, Professeur à l'INPT pour m'avoir accordé du temps en acceptant de rapporter mon mémoire, et pour le grand intérêt qu'il a manifesté pour ce travail.

Je suis également très reconnaissant envers Monsieur Paul JACQUET, Professeur à l'ENSIMAG, qui m'a honoré en acceptant le rôle d'examinateur ,et qui a ainsi apporté sa compétence en génie logiciel pour la critique de ce mémoire de thèse.

Je remercie également de nombreux membres du Laboratoire de Génie Informatique. Tout d'abord, Monsieur Yves CHIARAMELLA, le directeur, pour l'environnement scientifique et matériel qu'il a su instaurer et qui m'a beaucoup aidé pour mener à bien ce travail de recherche. Ensuite, les membres de l'unité du centre ville pour leur soutien moral et pour toutes les pauses café que nous avons partagées, et qui m'ont été d'un grand secours dans les périodes difficiles de ces trois années.

Enfin, je ne puis oublier de remercier tout spécialement Chantal ROBACH qui s'est avérée être un directeur de thèse remarquable malgré les lourdes tâches administratives qu'elle assure au sein du laboratoire. Tout au long de mon travail de thèse, elle a su doser son encadrement en me donnant une autonomie dans mes recherches tout en m'orientant dans les phases difficiles par ses connaissances. Elle a ainsi fait preuve de confiance dans mon travail et m'a toujours apporté son soutien, sans lesquels je n'aurais certainement pas abouti dans mon travail.

Introduction

La complexité des systèmes informatiques développés actuellement mène à une intégration de plus en plus importante de leurs fonctionnalités. Ceci est d'autant plus vrai si les-dits systèmes sont embarqués dans des avions ou des satellites. Ainsi, la complexité de ces systèmes se répercute à tous les niveaux et essentiellement au niveau des cartes et des circuits intégrés. Du point de vue de la conception de ces systèmes numériques, cette tendance a induit un nouveau type de circuits intégrés appelés ASIC (Circuits Intégrés à Application Spécifique). Le rôle de ces circuits est de réaliser des fonctions qui auparavant étaient, soit réalisées par des cartes (en utilisant des composants standards), soit réalisées par logiciel (autour d'un micro-processeur).

L'émergence de ces circuits, souvent fabriqués en très petites séries, a fait évoluer les méthodes de conception et a mis en évidence des problèmes spécifiques. En termes de méthodes, les concepteurs ont été amenés à appliquer des techniques d'analyse descendante proches de la conception du logiciel. Il s'agit donc de concevoir les circuits en appliquant plusieurs étapes d'affinement en partant des spécifications jusqu'à la réalisation physique. A chaque étape d'affinement sont associés des choix architecturaux qui induisent l'architecture finale du produit.

Chacune des étapes de conception est complétée par une phase de validation. La validation consiste à déterminer l'adéquation de la conception d'une fonction par rapport à sa spécification dans l'étape précédente. Pour garantir la validité, plusieurs approches sont possibles et souvent se complètent :

- la preuve formelle cherche à prouver l'adéquation d'une réalisation avec sa spécification ;
- le test de validation consiste à comparer le comportement de la spécification et de la réalisation sur un certain nombre de valeurs de test ;
- l'utilisation d'outils de synthèse permet de garantir la validité dans la mesure où les outils n'ont pas de défaut.

Bien que la preuve formelle semble théoriquement la plus sûre pour garantir la validité, l'utilisation des tests de validation est généralement incontournable. En effet, la preuve d'une conception est souvent exorbitante voir impossible à calculer. Bien que le test ne permette pas de garantir la validité d'une conception, il a l'avantage de garantir l'invalidité en cas d'échec.

Les étapes de conception et de validation mènent finalement à la définition du produit matériel sous forme de descriptions informatisées de la structure au niveau transistor ou électrique. C'est à partir de ce moment qu'est envisagée la phase de fabrication de circuits physiques matérialisant le circuit "théorique". Sur ces circuits physiques apparaît le problème du test de matériel. En effet, le circuit peut contenir des défauts qui le rendent différent du circuit théorique et peuvent donc induire des dysfonctionnement d'un système. Ces défauts peuvent provenir, soit de problèmes de fabrication, soit de problèmes de vieillissement du circuit.

Le test de matériel est donc un ensemble de données à appliquer à un circuit physique dans le but de déterminer s'il contient ou non des défauts. L'ensemble de données de test est appelé programme de test. Les problèmes cruciaux pour déterminer un programme de test se situent à différents niveaux :

- le programme couvrent-il ou non tous les défauts d'un circuit ?
- quel est le coût pour déterminer ce programme ?
- quel est le coût pour appliquer ce programme à un circuit physique ?
- quels moyens utiliser pour déterminer ce programme ?

Le programme de test doit avoir une bonne couverture des défauts pouvant apparaître. Or, cela est très difficile à mesurer car, d'une part, se pose le problème de la connaissance des défauts et de leurs manifestations et, d'autre part, il faut limiter le coût du programme de test qui ne peut être exhaustif. Dans le cas des circuits fabriqués en grandes séries sur plusieurs années, la couverture du programme de test peut être améliorée par analyse des défauts mis en évidence par les utilisateurs. Par contre, dans le cas des ASICs fabriqués en petite série, ce mécanisme d'amélioration n'est plus

envisageable, le programme de test se doit d'avoir une bonne couverture dès la commercialisation du produit.

Le coût de détermination du programme de test dépend de plusieurs paramètres, dont la place du test dans la conception du circuit et les outils éventuellement utilisés. Le coût est souvent élevé du fait de la prise en compte du test du matériel uniquement après la conception du circuit théorique. En effet, la structure d'un circuit (son architecture) peut induire des difficultés pour couvrir les défauts qui peuvent apparaître dans certaines zones du circuit. Parmi les outils utilisés pour la génération du programme de test, nous pouvons distinguer :

- les simulateurs de pannes qui permettent de connaître la couverture d'un programme de test vis-à-vis de certains modèles de pannes (les collages le plus souvent) ;
- les générateurs de programme de test qui à partir d'un circuit tendent à définir un programme de test ;
- les évaluateurs de testabilité qui permettent de connaître le coût du test et les zones du circuit plus ou moins facilement testables.

Les problèmes liés à la génération du test de matériel ont donc induit plusieurs types d'études qui peuvent être considérées comme complémentaires. Ainsi, des travaux s'intéressent aux défauts et à leurs manifestations (les modèles de collage et de court-circuit étant largement insuffisants), des études se sont orientées vers la prise en compte du test dans la conception (test intégré, scrutation de chemins), d'autres travaux se sont focalisés sur la génération du test (au niveau portes logiques puis au niveau comportemental) et enfin des travaux se sont concentrés sur le calcul de mesures de testabilité dans le but de prévoir l'effort de test et/ou de modifier certains choix de conception pour faciliter le test.

La complexité actuelle des circuits tend à faire remonter les outils de génération du test ou d'évaluation de testabilité dans les phases de conception pour deux raisons principales : tout d'abord, agir et connaître le plus tôt possible les caractéristiques de

test d'un produit en cours de conception et, ensuite, limiter le coût de ces traitements en travaillant sur des descriptions de haut niveau (comportement, hiérarchie), plus concises que les descriptions de niveaux portes logiques ou transistors.

Dans ce contexte, nous nous sommes penchés sur l'étude de l'évaluation de testabilité et l'aide à la génération de programmes de test à partir de descriptions en langage VHDL, descriptions de comportement et de hiérarchie. Le but étant d'obtenir un outil utilisable tout au long de la conception et ainsi de pouvoir :

- évaluer le coût de génération du test ;
- aider à la conception de produits facilement testables ;
- donner une ossature des programmes de test en fonction du contexte (fin de production, maintenance).

Ainsi, nous présentons dans le chapitre 1 un certains nombre de travaux caractéristiques de la génération du test à partir de descriptions du matériel basées sur des langages de haut niveau.

Dans le cadre de l'étude que nous présentons, nous nous sommes intéressés à un outil d'évaluation de testabilité a priori, c'est-à-dire un outil qui donne des appréciations sur la qualité de testabilité de circuits ou systèmes sans entreprendre la génération du test proprement dite. Dans cette optique le travail a consisté à définir comment utiliser un outil déjà développé au Laboratoire de Génie Informatique, l'outil SATAN (System Automatic Testability ANalysis), pour fournir une aide à la conception de circuits testables. L'outil SATAN est basé sur la représentation d'un équipement sous la forme d'un graphe de transfert d'information.

Le chapitre 2 présente les principes de base de cet outil. L'outil SATAN produit également une spécification fonctionnelle du programme de test d'un équipement. Ce résultat consiste en la définition d'une séquence de fonctions (d'entrée/sortie) à activer, l'ordonnancement des fonctions étant déterminé relativement à une stratégie de test. Une stratégie de test dépend du contexte de test, fin de production ou maintenance par

exemple. Elle constitue un critère d'ordonnancement qui garantit une localisation optimale tout en donnant un aperçu du coût de la génération du test.

La construction du graphe de transfert d'information que ne envisageons dans cette étude se base sur la possibilité de prendre en compte différents niveaux de description. Ceci nous a amené à concevoir deux générateurs de graphes de testabilité, l'un s'intéressant aux descriptions du niveau hiérarchique le plus bas (descriptions de comportement), et l'autre traitant des descriptions structurelles de niveaux de hiérarchie plus élevés.

Le lien entre les différents niveaux de hiérarchie est assuré, tout comme dans un environnement de conception assistée par ordinateur, par l'utilisation d'une bibliothèque spécifique à l'outil SATAN.

Le chapitre 3 présente une compilation de descriptions de comportement sous la forme de graphes de transfert d'information statiques. Nous y mettons en évidence des principes systématiques de détermination de la représentation pour le test à partir de descriptions de comportement. Dans un souci de limitation de la taille des graphes résultants, nous y introduisons également des étapes de simplification et de réduction du graphe.

Toutefois, si nous observons les données nécessaires au calcul des mesures de testabilité (la notion de capacité d'information), il s'avère que le principe de traduction montré n'est pas satisfaisant dans le cas de cycles. En effet, les cycles d'une description apparaissent également dans le graphe, et dans ce cas le flot d'information ne permet pas de représenter l'accumulation d'information ou l'émission d'une séquence d'informations. C'est le cas, par exemple, d'un compteur ou d'un registre à décalage.

Dans le chapitre 4, nous introduisons la notion de graphe de transfert d'information dynamique qui intègre une nouvelle définition de la capacité d'information. Les capacités d'information sont des valeurs nécessaires au calcul des mesures de testabilité. Nous présentons tout d'abord le moyen de déterminer ces valeurs

localement à partir des expressions VHDL, puis leur propagation à l'intérieur d'une fonction de test. La solution que nous avons mis en oeuvre ici ne considère qu'une classe restreinte des constructions que nous avons tenu à représenter correctement sous forme de graphe de transfert d'information.

Le chapitre 5 montre la compilation des descriptions structurelles qui correspondent aux niveaux de hiérarchie plus élevés que les descriptions de comportement. Cette compilation se base sur la présence, dans la bibliothèque de test, des graphes de transfert d'information des cellules instanciées, d'une part, et, d'autre part, sur le réseau d'interconnexion du circuit. Nous y montrons deux étapes de compilation. La première consiste à extraire de la bibliothèque les graphes nécessaires et éventuellement simplifier ceux-ci. La seconde étape consiste à analyser le réseau d'interconnexion de manière à concaténer les différents graphes entre eux.

Chapitre 1

Etat de l'art sur les méthodes de test à partir des langages de description de matériel

1. Introduction

Dans le processus de conception de circuits intégrés complexes, et particulièrement de circuits à la demande tels que les ASICs, les tâches de validation et de vérification peuvent prendre une importance considérable dans le coût de production et de maintenance d'un produit. Ce coût croissant est dû, entre autres, au degré d'intégration et à la complexité accrus des ASICs actuels, et ce d'autant plus que ces circuits sont souvent produits en séries limitées. Bien que l'environnement offert par les outils de CAO fournisse au concepteur un ensemble de moyens de conception automatisés, la génération des programmes de test n'est pas réellement intégrée dans cet environnement CAO, et exige une participation manuelle importante de la part du concepteur et/ou de l'ingénieur de test. Ceci explique l'effort constant dans la communauté scientifique pour tenter de développer de nouvelles méthodes de génération de test, capables de résoudre les nouveaux problèmes soulevés par l'évolution technologique.

Les méthodes et outils de génération de test ont été principalement développés à partir de descriptions au niveau portes logiques et ont été étendues pour des circuits décrits à partir de primitives matérielles plus complexes. La majorité de ces méthodes sont fondées sur le principe des chemins de sensibilisation [GOE81, FUJ83], dérivés du D-algorithme [ROT66]. Des améliorations dans le processus de génération de test ont été également apportées par les techniques telles que :

- les méthodes fondées sur la "scrutation de chemin" ou Scan-Path [WIL82, BEN84] qui orientent vers la conception de circuits facilement testables ;
- les évaluations numériques de testabilité [ARC85, ROB88] qui peuvent être utilisées comme heuristiques pour guider les Générateurs Automatiques de Programmes de Test (ATPGS) [BRG84, KRI85, IVA86] ;
- les techniques de partition [BEE86, MUR86] qui ont pour but de réduire l'effort de génération du test en partitionnant le circuit en sous-blocs matériels testables indépendamment.

Les progrès technologiques, conjugués au manque d'information sur la structure au niveau portes logiques, rendent les techniques de test classiques en partie inefficaces. Pour pallier les faiblesses des générateurs automatiques de stimuli de test face à une complexité qu'ils ne peuvent maîtriser, de nombreux travaux de recherche se sont orientés vers la définition de méthodes de test à partir d'une description fonctionnelle ou comportementale des circuits. Ces travaux sont essentiellement fondés sur des descriptions au niveau registre de transfert [SU81, LAI83, LIN85] ou des langages de description de haut niveau [LEV82, BAR86, ARM88]. Certains travaux cherchent à prouver que les fonctions du circuit sont correctes plutôt qu'à garantir que toutes les fautes structurelles sont détectées. Ces méthodes reposent sur la connaissance des fonctionnalités du circuit qui peuvent être données par un jeu d'instructions ou trouvées au moyens de techniques de "reverse engineering" à partir d'une description structurelle.

Ce chapitre a pour objectif de faire une synthèse des méthodes de génération de test à partir de langages de description de matériel. Ces méthodes peuvent être sommairement classées selon le niveau de description : le niveau *jeu d'instructions* utilisé pour les circuits programmables, le niveau *RTL* (Register Transfer Language) et le niveau *comportemental*. L'analyse des orientations pour le test de circuits complexes prendra en compte des critères tels que la représentation ou modélisation du circuit, les modèles de fautes considérés, le type de partitionnement (fonctionnel ou structurel) et la méthode de génération des données de test (extensions de la notion de chemin de sensibilisation).

Quatre travaux majeurs sont étudiés :

Lai et Siewiorek [LAI83] modélisent le comportement du circuit par un automate d'états fini, appelé *graphe de transformation d'état (STG)* qui peut être déduit d'une description RTL. Le graphe décrit le flot de données à travers le circuit ; les fautes sont considérées soit au niveau du graphe : fautes sur les arcs (chemins de données) ou sur les noeuds (opérateurs), soit au niveau des primitives du graphe. La génération du test est alors réalisée par des techniques de chemin sensible dans le graphe.

Lin et Su [SU81, LIN85] ont proposé une méthode de test fonctionnel fondée sur une description RTL : le comportement du circuit est décrit par un ensemble d'expressions RTL. Les fautes sont considérées à la fois au niveau RTL (transfert de données) et au niveau fonctionnel (erreurs de séquencement, par exemple). La génération du test est fondée sur l'exécution symbolique des machines erronées et de la machine juste ; les vecteurs de test sont alors obtenus en résolvant les inégalités entre les résultats de l'exécution symbolique.

Levendel et Menon [LEV82] ont proposé une extension des techniques de chemins sensibles, du niveau porte logique au niveau de primitives fonctionnelles complexes : la D-propagation est réalisée à travers les constructions élémentaires du langage décrivant les fonctions du circuit. Le modèle de fautes considère les fautes de collage sur les entrées des fonctions et un ensemble de *fautes fonctionnelles* qui altèrent le comportement des fonctions. Les *fautes fonctionnelles* sont soit des fautes de contrôle qui transforment la signification des expressions conditionnelles CHDL, soit des *fautes fonctionnelles générales* qui transforment une fonction CHDL en une autre fonction.

Armstrong et Barclay [BAR86, O'NE89] ont étudié le problème de la génération d'un test fonctionnel à partir de langages de description de haut niveau. Ils ont particulièrement étudié deux problèmes complémentaires : d'une part, la définition de fautes fonctionnelles de haut niveau ou fautes comportementales ; d'autre part, la génération de test proprement dite qui est fondée sur une technique *d'arbres d'objectifs*, travaillant sur une représentation PROLOG du circuit.

Ce chapitre a pour objectif de mettre en évidence les avantages et inconvénients d'un test fonctionnel à partir de descriptions de haut niveau, et d'indiquer les orientations actuelles du test fonctionnel.

2. Les Travaux de Lai et Siewiorek

2.1. Principe de base

L'outil de génération de test fonctionnel proposé par Lai et Siewiorek [Lai81, Lai83] est fondé sur une description du circuit à tester (microprocesseur) du type

transfert-registre, qui est transformée en un automate d'états fini. Cet automate est donné par un *graphe de transformation d'état (STG)* et un ensemble de variables d'état. La construction de l'automate est réalisée à l'aide d'éléments de base appelés *primitives.*

Les modèles de fautes, introduits par l'utilisateur, peuvent être définis :

- soit au niveau du graphe : les fautes concernent soit les arcs (transferts de données) soit les noeuds du graphe (opérateurs de transformation de données) ;
- soit au niveau des primitives du graphe.

Dans le premier cas, les fautes sont dites de niveau *macro* (modèle de fautes fonctionnelles) alors que dans le second cas, les fautes sont définies comme des *micro-fautes* (modèle de fautes structurelles).

La génération d'un programme de test fonctionnel est réalisée en trois étapes :

- *une analyse fonctionnelle* associe à chaque modèle de faute et à chaque primitive un test paramétré ; le test est déterminé par la propagation arrière des paramètres de test et la propagation avant des résultats de test. La construction du test est réalisée en définissant les contraintes de sensibilisation du chemin de test ;
- *une synthèse des cas de test* a pour rôle, pour chacun des tests paramétrés définis par l'analyse fonctionnelle, de substituer les paramètres formels par une suite de vecteurs de test fournis par une base de données de vecteurs associés à la faute ;
- *une synthèse du programme de test* construit un programme assembleur qui optimise le nombre de vecteurs de test et d'instructions ; cette phase n'est pas automatisée.

2.2. Représentation du système : graphe de transformation d'état

Le graphe de transformation d'état est une représentation fonctionnelle de l'équipement à tester sous forme d'un graphe orienté.

Les noeuds du graphe sont de deux types :

- les noeuds *liens* modélisent l'émission d'une donnée ou d'un contrôle, vers plusieurs destinations (notion de "fan out"). Ils sont principalement introduits pour une question de lisibilité ;
- les noeuds *acteurs* représentent les opérateurs de transformation des données ; les acteurs sont de sept types différents.

Il existe également deux noeuds spéciaux, le noeud *begin* qui ne comporte qu'un seul chemin sortant, et le noeud *end* qui ne comporte qu'un seul chemin entrant.

Les chemins du graphe représentent les transferts de contrôle et de données entre les noeuds. Ils peuvent correspondre à des transferts de plusieurs "bits" lorsqu'ils sont liés à des données (dans ce cas ils sont appelés *chemins de données*), ou à un seul "bit" qui peut éventuellement correspondre à un contrôle (*chemin de contrôle*).

Dans un graphe de transformation d'état, l'information est transférée par des *jetons*. Un chemin, à un instant donné, soit ne possède aucun jeton, soit en possède une et une seule. Le jeton représente la valeur présente sur le chemin à l'instant considéré : le jeton est soit un entier pour un chemin de n bits, soit une valeur binaire pour un chemin de un bit. Les jetons de données sont distingués des jetons de contrôle, ces derniers se référant soit à un *true-token* (signal vrai) soit à un *false-token* (signal faux) selon la valeur booléenne présente.

L'évolution des états du graphe est dirigée par un ensemble de *règles de mise à feu*. Ces règles spécifient l'ensemble des jetons requis sur les chemins d'entrée d'un noeud pour que ce noeud puisse être activé. L'activation d'un noeud va générer de nouveaux jetons sur ses chemins sortants.

2.3. Les modèles de fautes

Deux modèles de fautes ont été définis :

- les modèles de fautes au niveau du graphe ou macro-fautes,
- les modèles de fautes sur les primitives du graphe ou micro-fautes.

Modèles de fautes au niveau du graphe

Les modèles de fautes au niveau du graphe concernent soit les chemins soit les noeuds du graphe. Ce modèle associe un ensemble de données de test et un ensemble de résultats de test à chaque élément du graphe, c'est à dire à chaque chemin et à chaque noeud. Le modèle de fautes définit cette association au moyen de symboles représentant les paramètres de données de test et les résultats de test dans le graphe.

Lorsqu'on considère le test d'un chemin, le modèle de fautes détermine les paramètres de test et les résultats qui sont identiques et qui doivent être introduits sur le chemin. En ce qui concerne le test d'un noeud, le modèle de fautes détermine les paramètres de test sur les chemins d'entrée du noeud, et le résultat de test associé sur le chemin de sortie du noeud.

Les stimuli de test qui sont associés aux paramètres de test sont supposés prendre en compte toutes les fautes qui doivent être testées, aussi bien sur le chemin que sur le noeud.

L'analyse fonctionnelle ne détermine pas les stimuli de test, car ceux-ci sont prédéfinis par le modèle de fautes.

La génération de test est réalisée par le biais d'une propagation arrière des symboles représentant les paramètres de test vers un point d'entrée (noeud de lecture mémoire ou noeud *begin*), et d'une propagation avant des symboles représentant les résultats de test vers un point de sortie (noeud d'écriture en mémoire ou noeud *end*).

Les modèles de fautes au niveau du graphe peuvent considérer l'hypothèse de fautes multiples sur les chemins et/ou les noeuds : la limitation vient alors du nombre de tests qui seraient nécessaires pour prendre en compte cette hypothèse.

Modèles de fautes au niveau des primitives :

Ces modèles de fautes décrivent les fautes au niveau des primitives du graphe. Dans le cas général, il faut utiliser un test exhaustif ou aléatoire, si l'on ne connaît pas l'implémentation de la primitive. Selon la connaissance que l'on a de la primitive, le test peut être généré de trois manières différentes :

1. *inspection* : la primitive est vue comme une boîte noire ce qui induit un test exhaustif ou aléatoire. Cette technique s'applique à une primitive dont la structure n'est pas connue, seul ces entrées/sorties sont spécifiées ;
2. *méthodes traditionnelles* : le modèle de faute est fondé sur la réalisation spécifique de la primitive et des générateurs de test classiques peuvent être utilisés. Dans ce cas, la primitive doit être connue au niveau structurel ;
3. *génération récursive* : la primitive est décrite elle-même sous forme d'un graphe de transformation d'état dont les primitives sont de niveau inférieur ; ainsi il peut être fait un appel récursif à l'outil de génération.

2.4. Génération du test

Le processus de génération du test comporte trois phases. La première étape ou *analyse fonctionnelle* consiste en une propagation arrière des paramètres de test et une propagation avant des résultats de test, pour chaque cas de faute : cette phase permet la sensibilisation d'un chemin du graphe.

La seconde étape ou *synthèse des cas de test* associe des valeurs de test aux paramètres de test, en fonction du chemin sensible ; cette étape associe également les valeurs justes qui doivent être observées sur les paramètres de sortie.

La dernière étape ou *synthèse du programme de test* permet de générer le programme de test en langage assembleur.

Analyse fonctionnelle

L'analyse fonctionnelle du graphe se déroule, pour chaque modèle de faute, en quatre phases successives : introduction des paramètres de test, propagation arrière des paramètres de test, propagation avant des résultats de test, et justification de la solution obtenue.

- *Introduction des paramètres de test*

La résolution du test est basée sur l'introduction d'un ensemble de symboles qui représentent les données de test et les résultats de test dans le graphe. Le modèle de fautes contient la définition de ces paramètres qui représentent à la fois les données de

test de la faute, et les résultats du test de la faute. La génération du test va consister à propager ces paramètres, du lieu de manifestation de la faute vers les entrées et les sorties du graphe.

• *Propagations avant et arrière des paramètres de test*

La propagation arrière (resp. avant) consiste à propager un paramètre de test (resp. résultat de test) depuis l'endroit de manifestation de la faute vers une entrée (resp. sortie) du graphe. Une entrée du graphe est un noeud d'accès en lecture ou un noeud *begin* tandis qu'une sortie du graphe est un noeud d'accès en écriture ou un noeud *end*. La méthode de propagation est fondée sur la connaissance de la fonction qui transforme un paramètre de test (resp. résultat de test) en un paramètre d'entrée (resp. sortie). L'ensemble des chemins de données et des noeuds qui sont utilisés pour réaliser cette fonction de transformation, constitue un chemin de sensibilisation.

La propagation arrière se fait en choisissant un noeud amont qui réalise une fonction n-aire F et à l'aide d'un vecteur de propagation C tel que :

- $F(C_0, ...C_{i-1}, g(x), C_{i+1},...C_{n-1}) = x$ pour tout x appartenant au domaine de sortie de F
- $C = (C_0,C_1,...C_{i-1}, C_{i+1},...C_{n-1})$ est un vecteur constant
- g, telle que g^{-1} existe, est la fonction de transformation du vecteur.

La propagation avant se fait en choisissant un noeud aval qui réalise une fonction n-aire F et à l'aide d'un vecteur de propagation C, tel que :

- $F(C_0, ...C_{i-1}, x, C_{i+1},...C_{n-1}) = g(x)$ pour tout x appartenant au domaine d'entrée de F
- $C = (C_0,...C_{i-1}, C_{i+1},...C_{n-1})$ est un vecteur constant
- g, telle que g^{-1} existe, est la fonction de transformation du vecteur.

Les propagations avant et arrière fournissent une suite de vecteurs de propagation qui définissent les contraintes de sensibilisation d'un chemin de test. Quand une nouvelle contrainte est introduite, lors de la phase de propagation, il y a immédiatement une résolution de consistance de cette contrainte avec les contraintes existantes.

• *Justification*

La justification a pour rôle, après les phases de propagation des paramètres et de résolution de consistance, de déterminer les chemins dont les valeurs n'ont pas été spécifiées par ces phases. L'ensemble des chemins du graphe est spécifié de façon à ne pas interférer sur le chemin sensibilisé. L'analyse fonctionnelle fournit un test paramétré dont les contraintes permettent de sensibiliser un chemin de test ; les entrées du chemin de test sont les paramètres de test, et ses sorties sont les résultats de test.

Synthèse des cas de test

La synthèse des cas de test consiste à substituer, dans chacun des tests paramétrés, les paramètres par les données de test et les résultats de test réels. Les données de test sont stockées dans une base de données : les stimuli de test sont supposés avoir été développés à l'aide de modèles de fautes plus fins.

Synthèse du programme de test

La synthèse du programme de test consiste, en regroupant les instructions en classes pouvant être testées par des séquences de test similaires (même mode d'adressage, même nombre d'opérandes, etc...), à générer un programme de test assembleur qui optimise le nombre d'instructions et le nombre de vecteurs de test. Ceci s'applique seulement à des circuits programmables, tels que les microprocesseurs.

2.5. Conclusion

L'intérêt d'une telle méthode réside dans son utilisation possible dès les premières phases de la conception, alors même que la structure du circuit n'est pas entièrement connue. Cette méthode a été appliquée à l'unité centrale d'un calculateur PDP-8, en considérant les collages simples sur les chemins de données. Les résultats étaient prometteurs, puisque le programme de test généré ainsi fournissait un taux de détection supérieur à celui du programme du fabricant, et ceci avec un nombre d'instructions inférieur.

Le principal inconvénient de cette méthode provient de la représentation multiple d'un même bloc matériel ; ceci implique que, soit une faute donnée peut être testée

plusieurs fois, soit une faute structurelle va impliquer plusieurs fautes fonctionnelles. Cet inconvénient pourrait être minimisé en utilisant une meilleure représentation (difficile à réaliser) ou en optimisant le programme de test. Mais ces deux phases restent critiques dans la mesure où elles sont réalisées manuellement.

3. Les travaux de Lin et Su

3.1. Principe de base

Lin et Su [SU81, LIN84, LIN85] ont proposé une méthode de génération automatique du programme de test fonctionnel d'un équipement digital, à partir d'une description au niveau transfert registre (RTL) ; cette méthode est basée sur l'exécution symbolique du système à travers des chemins d'activation appelés *Function Submodules* (FS). La méthode consiste à comparer les résultats fournis par l'exécution symbolique d'une machine correcte et d'une machine dans laquelle est introduite une faute. Le type de description du système à tester montre que cette méthode s'applique principalement à la génération de test des systèmes microprogrammés. La méthode de génération du programme de test se décompose en cinq phases principales :

* partitionner la description RTL en un ensemble de fonctions d'activation appelées FS. Cet ensemble de fonctions doit couvrir tous les composants (instructions, variables,...) de la description RTL ;
* rechercher les chemins de couverture ou chemins d'exécution pour chaque FS, c'est à dire les différentes possibilités d'activation d'une FS ;
* faire l'exécution symbolique d'un chemin d'exécution et trouver les tests pour les fautes de transfert de données dans la FS ; il s'agit de déterminer les registres à charger pour obtenir une valeur donnée sur une instruction FS et les registres à lire pour observer le résultat de test ;
* injecter une faute dans une expression d'une FS, choisir un chemin couvrant cette expression et faire l'exécution symbolique du chemin en tenant compte des contraintes d'activation de ce chemin ;

* résoudre les inégalités symboliques pour déterminer les vecteurs de test qui permettent d'obtenir une inégalité entre les sorties observables de la machine correcte et celles de la machine erronée.

Lin et Su ont introduit neuf types de modèles de fautes au niveau d'une description RTL. Ces modèles concernent les registres, les labels de branchement, et les évaluations d'expressions conditionnelles.

3.2. Représentation du système

Le système à tester est représenté par un ensemble d'expressions RTL dont la syntaxe est la suivante :

$k : (t,c)\ R_d \leftarrow f(R_{S1},...,R_{Si}) \rightarrow n$

- k représente le label de l'instruction,
- t est un indicateur binaire de temps,
- c est une expression conditionnelle qui spécifie la condition d'exécution de l'instruction,
- R_d est le registre destination de l'instruction,
- R_{Si} est le ième registre source de l'instruction,
- f est une opération arithmétique ou logique,
- $\leftarrow$ représente le transfert de résultat vers le registre destination,
- $\rightarrow$n est le label de l'instruction suivante si t et c sont vrais ; sinon l'instruction suivante est celle de label (k+1).

Ces différents objets sont appelés composantes RTL. Ce type de description permet de décrire le système sous la forme d'un décodeur de micro-instructions au niveau transfert registre.

Une telle description peut aussi être représentée par un graphe orienté dont les noeuds représentent les expressions RTL ; un arc d'un noeud N_1 vers un noeud N_2 signifie que l'instruction associée au noeud N_2 sera exécutée après celle associée au noeud N_1. Un noeud lié à une instruction conditionnelle aura deux arc sortants : l'un

des arcs est relié au noeud successeur associé à la condition vraie, l'autre arc est relié au noeud successeur associé à la condition fausse.

La recherche des FS d'un système consiste à rechercher les chemins menant d'un noeud initial à un noeud final dans le graphe. Une FS est donc caractérisée par les noeuds qu'elle couvre, c'est à dire par les valeurs des conditions associées aux expressions conditionnelles. Un noeud conditionnel est couvert par au moins deux FS, l'une pour la condition vraie, l'autre pour la condition fausse.

- *Remarque* :

Dans le but d'avoir une correspondance étroite entre une FS et le décodage d'une micro-instruction, Lin et Su ont introduit deux types de conditions, donc deux types de noeuds conditionnels :

- les conditions sur le code opération, qui permettent de déterminer l'instruction à décoder,
- les autres conditions, liées aux données, ne définissent pas deux FS distinctes, mais deux chemins d'activation possibles dans une même FS.

La recherche des FS prend en compte les conditions du premier type. Les FS ainsi déterminées représentent les fonctions réalisées par le système. La génération du programme de test va consister à les traiter dans un certain ordre relativement à un ensemble de critères de caractérisation et une stratégie de test par rapport à ces critères.

Les critères de caractérisation d'une FS sont :

- les instructions qui réalisent des opérations de transfert,
- le nombre d'instructions qui forment la FS,
- le nombre de registres manipulés par la FS.

La stratégie de test utilisée pour ordonner les FS est du type *boule de neige (Start-Small)* [DEN68] : les premières FS activées sont celles relatives à de simples opérations de transfert ; les autres FS sont ensuite considérées, d'abord par rapport au nombre d'instructions puis par rapport au nombre de registres.

Lorsqu'une FS comporte des noeuds conditionnels, il existe donc plusieurs manières de l'activer, caractérisées par les différents chemins à travers le sous-graphe de la FS. Ces chemins sont appelés chemins de couverture.

La détermination de ces chemins est basée sur le même principe que la détermination des FS dans la description du système.

3.3. Les modèles de fautes

La description du système au niveau RTL permet de dériver neuf types de fautes sur les différents constituants d'une description, soit au niveau de leur fonctionnement, soit au niveau de leur sélection. A chaque type de faute sont associées les différentes possibilités qu'elle a de se manifester. Les modèles de faute sont les suivants :

* k/k' : faute sur le label de sélection d'une instruction ; le label erroné k' peut correspondre soit à un label existant ou inexistant dans la description, soit à une absence totale de label ;
* →/→' : faute au niveau du label de branchement de l'instruction suivante ; le branchement erroné →' peut correspondre soit à un label existant ou inexistant dans la description, soit à une absence totale de label ;
* t/t' : faute sur le signal de synchronisation avec une horloge ;
* c/c' : faute sur le résultat de l'évaluation d'une condition ;
* (R)/(R') : faute au niveau du contenu d'un registre ; les modèles de faute associés sont d'une part, le collage simple sur les registres constants, et d'autre part, les collages multiples et les courts-circuits entre les lignes de données pour les autres registres ;
* ←/←' : faute lors du transfert de données vers un registre destination ; les modèles de faute envisagés sont les mêmes que pour les registres non constants du type précédent ;
* R/R' : faute de décodage d'un registre ; le registre sélectionné R' est soit un autre registre du système, soit un registre inexistant ;

* f/f' : faute de décodage d'un opérateur ; l'opérateur sélectionné f' est soit un autre opérateur du système, soit un opérateur inexistant ;
* (f)/(f)' : faute au niveau de l'opération effectuée par un opérateur ; la modélisation de ce type de faute n'est pas évidente au niveau RTL.

Une analyse d'équivalence est effectuée sur ces modèles de fautes, de manière analogue à l'analyse faite au niveau porte logique ; cette analyse permet de déterminer les fautes équivalentes et de réduire ainsi le nombre de fautes à traiter aux cinq types suivants : R/R', f/f', →n/→n', c/c' et ←/←'.

Le processus de génération du test consiste alors à introduire chaque type de faute dans les expressions RTL, sachant que les fautes du type (f/f') ne sont pas considérés par Lin & Su. En effet, ce type de faute est très difficile à modéliser : une solution à ce problème a été proposée par Armstrong & Barclay par la proposition de perturbations de micro-opérations.

Parmi ces fautes, les fautes de transfert ←/←' sont dites non énumérables car elles sont liées à des valeurs alors que les autres fautes sont dites énumérables car elles sont liées aux constructions du langage RTL. Les traitements liés à ces deux classes de fautes seront différents : une faute énumérable sera traitée par l'exécution symbolique, alors qu'une faute non-énumérable sera traitée par une heuristique sur les valeurs.

3.4. Génération du test

Technique d'exécution symbolique

L'exécution symbolique d'un programme écrit en langage de haut niveau est une technique déjà utilisée dans le domaine de la preuve de programme. Dans le cadre de la génération du test, le problème est simplifié par la simplicité du langage de description RTL en comparaison d'un langage de haut niveau. Le principe de l'exécution symbolique consiste à exécuter une fonction en associant, non pas des valeurs effectives, mais des expressions symboliques aux différentes variables. Le résultat de l'exécution symbolique d'une FS est une expression formée de constantes et de valeurs symboliques représentant les entrées de l'expression. A une telle expression

est associé un arbre binaire appelé *arbre d'exécution symbolique (SET*: Symbolic Execution Tree) ; cet arbre binaire met en évidence tous les flots possibles d'information à travers le programme.

Dans le cadre du test, l'exécution symbolique est utilisée pour construire le *SET* associé à une machine correcte et le *SET* associé à une machine défectueuse (pour une faute énumérable donnée dans une FS donnée). La seconde étape consiste à résoudre l'inégalité symbolique des deux expressions ainsi construites : dans le but de détecter la faute introduite, il faut obtenir un résultat différent entre la machine correcte et la machine défectueuse. La résolution des inégalités symboliques détermine un ensemble de valeurs sur les variables symboliques d'entrée de la FS qui fournissent des résultats différents selon qu'il y ait présence de la faute ou non. Si de telles valeurs sont trouvées, elles seront retenues pour former des vecteurs de test.

Algorithme de génération du test

L'algorithme global de génération du test est décomposé en trois phases principales :

- *le pre-process* qui consiste à analyser la syntaxe de la description RT, partitionner le système en un ensemble de FS et à déterminer les chemins de couverture ;
- *le S-algorithme* qui détermine les vecteurs de test de chaque FS ;
- *le post-process* qui vérifie que toutes les FS, toutes les instructions et toutes les fautes ont été traitées et renvoie les résultats.

3.5. Conclusion

L'utilisation de techniques d'exécution symbolique est un outil puissant qui permet la génération de test à partir de descriptions de haut niveau. D'autre part, la détermination de chemins d'activation (Sous modules de fonction : FS) et l'utilisation d'une stratégie de test assure la qualité et une bonne structuration du programme de test.

La principale limitation de cette technique vient du fait qu'elle est applicable uniquement aux systèmes programmables, représentés par un jeu d'instructions. Un autre inconvénient est le coût pour obtenir les arbres d'exécution symbolique, pour

tous les cas de test possibles, et pour résoudre les inégalités symboliques. Son utilisation est donc limitée par la complexité des circuits que l'on veut traiter.

4. Les travaux de Levendel et Menon

4.1. Principe de base

Les travaux sur la génération de test fonctionnel, proposée par Levendel et Menon, se sont basés, d'une part, sur l'extension du D-algorithme aux constructions des langages de description de haut niveau et, d'autre part, sur le *-algorithme [LEV83] ; les auteurs ont proposé une extension de ces algorithmes, originellement appliqués au niveau portes logiques, à des constructions de langages de description de haut niveau (CHDL : Computer Hardware Description Language).

Le D-algorithme est basé sur la sensibilisation de chemins : il s'agit de propager l'effet d'une faute (D ou $\overline{D}$) d'un point du système vers une sortie observable et de justifier logiquement les conditions de propagation. La propagation des fautes est réalisée en utilisant des D-cubes qui représentent la propagation de la valeur D pour une entité. Le *-algorithme a introduit la notion de chemins critiques : un chemin critique décrit, par rapport à une sortie donnée, les combinaisons d'entrée permettant de contrôler cette sortie, et il détermine le sous-ensemble de ces entrées, dont le changement de valeur entraîne un changement de valeur de la sortie. La propagation de fautes est alors décrite par un ensemble de cubes critiques appelés *-cubes ; un *-cube est associé à chaque primitive du langage ou entité élémentaire liée au niveau de description. Dans le cas de langages de description de haut niveau, une entité est une construction du langage.

Le premier point résolu par Levendel et Menon a été de déterminer les D-cubes pour le D-algorithme, ou les *-cubes pour le *-algorithme, associés aux différentes constructions rencontrées dans un langage de description CHDL. Selon le type de construction CHDL : expression algébrique à résultat booléen, expression algébrique complexe (addition, décalage, comptage,...) ou structure de contrôle, les méthodes d'obtention des cubes seront différentes. Les cubes sont obtenus soit à partir

d'équations de base pour les opérateurs ET, OU, NON (appelées D-équations), soit par combinaison de D-cubes d'opérateurs de base pour les opérateurs tels qu'additionneur, multiplexeur, codeur, décodeur. Les cubes critiques sont obtenus soit à partir de leur définition par rapport aux D-cubes, soit à partir d'un système d'équations critiques appelés *-équations pour les opérateurs ET, OU et NON .

Les modèles de faute considérés sont les collages sur les entrées ou les sorties des blocs ou structures de base du langage CHDL, les fautes de contrôle qui modifient la signification des expressions conditionnelles (if ... then ... else, case ... of ..), et les *fautes fonctionnelles générales* qui transforment une fonction f en une autre fonction f_α.

La génération du programme de test se décompose en trois phases : propagation (c'est à dire construction d'un chaînage de cubes critiques), implication et justification.

4.2. La propagation à travers les constructions CHDL

Les D-cubes

Les D-cubes doivent décrire tous les chemins sensibles, c'est à dire les chemins permettant de propager une valeur D à travers une construction du langage de description CHDL. A chaque construction seront associés tous les cubes de propagation, un cube correspondant à un chemin. La méthode d'obtention des D-cubes dépend du type de construction CHDL considéré.

Expressions algébriques booléennes

Pour une expression algébrique dont le résultat est booléen, les D-cubes sont obtenus à l'aide d'un système d'équations, appelées D-équations, qui décrit les D-cubes des opérateurs booléens de base : ET, OU et NON. Les D-équations pour la fonction C=ab (opérateur ET) sont les suivantes :

$$C^0 = (ab)^0 = a^0 + b^0 + a^D b^{\overline{D}}$$

$$C^1 = (ab)^1 = a^1 b^1$$

$$C^D = (ab)^D = a^D b^1 + a^1 b^D + a^D b^D$$

$$C^{\overline{D}} = (ab)^{\overline{D}} = a^{\overline{D}} b^1 + a^1 b^{\overline{D}} + a^{\overline{D}} b^{\overline{D}}$$

La valeur en exposant est la valeur donnée à la variable. Les D-cubes s'obtiennent en calculant C^D qui fournit l'expression des entrées et les valeurs associées permettant d'obtenir une valeur D sur la sortie C. L'expression écrite sous forme de somme de monômes, décrit les différentes possibilités sur les entrées donnant une valeur D en sortie ; chaque monôme correspond donc à un vecteur de génération d'une valeur D sur la sortie. Un tel monôme est appelé D-cube et il correspond à la propagation d'une valeur D d'une entrée ayant pour exposant D vers la sortie, c'est à dire un chemin de propagation de la valeur D.

Pour une expression algébrique booléenne générale $Q = f(e_1, e_2,..., e_n)$, le calcul des D-cubes se fait par le développement de l'expression $Q^D = (f(e_1, e_2,...,e_n))^D$, à l'aide des D-équations des opérateurs logiques de base.

Expressions algébriques non booléennes

Dans le cas des expressions algébriques non booléennes, notamment les unités arithmétiques et logiques, la fonction est exprimée en termes d'opérateurs *types* dont les D-cubes sont connus. Ces opérateurs *types*, qui permettent de décrire l'ensemble des expressions, sont l'additionneur, l'incrémenteur, l'opérateur de décalage, le décodeur, l'encodeur et le multiplieur. Dans le cas des opérateurs de ce type, la propagation d'une valeur D sur une sortie est envisagée pour chacune des positions binaires de la sortie. La description des D-cubes devient complexe dans le cas d'opérateurs, tels l'additionneur ou l'incrémenteur, dont la valeur d'une position binaire de sortie dépend des valeurs des positions de poids plus faible (propagation de retenue).

Structures de contrôle

La définition des cubes de propagation à travers les structures de contrôle fait intervenir les deux possibilités de propagation d'une valeur D : soit à travers les expressions de transfert de données, soit à travers les expressions de contrôle. La méthode d'obtention des D-cubes consiste, pour chaque variable observable qui apparaît dans la structure de contrôle, à écrire son expression algébrique.

Pour les expressions cause-effet *(if...then...else, case)*, une variable V, qui est sensible à l'instruction, est choisie comme variable de propagation. Si $Expr_1$ est affectée à V

lorsque la condition V est vraie, sinon $Expr_2$ est affectée à V, alors l'expression résultante de V est :

$$V=C.Expr_1+\overline{C}.Expr_2.$$

La propagation d'une valeur D à travers les données est donnée par :

$$V^D = C^1.(Expr_1)^D + C^0.(Expr_2)^D$$

alors que la propagation au travers du contrôle est donnée par :

$$V^D = C^D.(Expr_1)^1.(Expr_2)^0 + C^{\overline{D}}.(Expr_1)^0.(Expr_2)^1$$

Pour les expressions conditionnelles imbriquées, la génération des cubes de propagation est basée sur des arbres de contrôle où les noeuds représentent les tests, les arcs représentent les résultats de test et les noeuds terminaux représentent les transferts de données. Le problème de D-propagation est alors ramené à un problème de parcours de graphe.

Blocs CHDL

Dans le cas de circuits comportant des blocs CHDL, la propagation d'une valeur D se fait à travers un ensemble d'instructions observables, c'est à dire un ensemble de variables accessibles à l'extérieur du bloc. La méthode de propagation dépend du type de langage, procédural ou non-procédural :

- dans un langage de type non-procédural, les instructions d'un bloc sont interprétées en parallèle ; la propagation d'une valeur consiste à choisir une variable de sortie et à déterminer les valeurs d'entrée correspondant à la sensibilisation d'un chemin vers cette variable. Cette procédure est possible dans le cas d'un langage de type non-procédural car les valeurs en partie droite d'une affectation ne changent pas lors d'une évaluation, car il n'y a pas séquencement des instructions ;
- dans le cas d'un langage de type procédural, les instructions du bloc sont évaluées en séquence ; il faut donc tenir compte, dans la sensibilisation d'un chemin vers une variable de sortie, du fait qu'une variable peut être modifiée puis utilisée dans une expression ultérieure. Par exemple si l'on envisage la

séquence "$C = Expr_1 ; F = f(C, x)$", la propagation d'une valeur vers F n'est pas simplement déterminée par $f(C,x)$, mais également par $Expr_1$.

Il faut donc faire intervenir dans l'expression d'une variable, pour un chemin de contrôle donné dans le bloc, toutes les opérations précédentes qui influencent les variables en partie droite de l'affectation de la variable. Par conséquent, la nécessité de considérer des séquences d'expressions complique la procédure.

Les cubes critiques ou *-cubes

L'algorithme de génération de test à partir des cubes critiques, le *-algorithme, est analogue au D-algorithme. Les chemins critiques sont construits à partir des sorties jusqu'aux entrées par le chaînage de cubes critiques. La notion de chemin critique étant moins précise que la notion de D-propagation, il en résulte un nombre moins important de cubes critiques que de D-cubes, donc une diminution de la complexité de génération du test.

Définition d'un cube critique

Si l'on considère une fonction n-aire $z = f(x_1,...,x_n)$, on dit qu'une entrée x_i est critique par rapport à la sortie z si un changement de valeur de x_i provoque un changement de valeur de z. Le cube critique d'une fonction est une combinaison des valeurs d'entrée qui permettent de définir un ensemble maximal de variables critiques par rapport à la sortie z. Il en découle que les cubes critiques dépendent de la valeur de la sortie.

- *Notation* : dans l'expression d'un cube critique, y^i signifie $y = i$ ($i \in \{0,1\}$), et y^{i*} signifie que y^i est critique (le passage de $y = i$ à $y = i'$ entraîne la modification de la sortie liée au cube critique).

Pour l'obtention des cubes critiques, Levendel et Menon proposent trois algorithmes :

* la première procédure, utilise la notion de D-cubes maximaux ; les cubes critiques sont obtenus en remplaçant D et $\overline{D}$ dans chaque cube maximal par 1* et 0* ;
* les deuxième et troisième procédures sont basées sur un ensemble d'équations critiques des opérateurs ET, OU, et NON, et les cubes critiques sont obtenus de

façon analogue aux cubes critiques pour les expression algébriques. Les deux procédures diffèrent par le système d'équations choisi. La procédure 2 fournit un ensemble de cubes critiques moins complet que celui de la procédure 3.

4.3. Les modèles de fautes

Un circuit étant totalement défini par la connaissance de la valeur des entrées et des variables d'état, un modèle de fautes évident est le collage à 0 ou à 1 des entrées, des sorties et des variables d'état.

De plus, Levendel et Menon considèrent deux autres catégories d'erreurs fonctionnelles :

- *les fautes de contrôle* correspondent à une variable de contrôle qui a une valeur erronée, et affectent les transferts conditionnels dans les constructions CHDL. Bien qu'il n'y ait pas de correspondance directe entre une faute de contrôle et les défauts physiques, la détection de ces fautes de contrôle améliore le taux de couverture en activant des branches qui ne seraient pas activées par ailleurs ;
- *les fautes fonctionnelles générales* sont des fautes dont les effets sont connus mais ne peuvent être décrits par des modèles de collage ou d'erreurs de contrôle. Ce modèle de faute prend en compte le changement d'une expression E en une expression fausse E_α, α représentant le défaut.

4.4. Génération du test

Les étapes du processus de génération de test pour des circuits comportant des blocs CHDL sont pratiquement identiques à celles du processus de génération pour des circuits décrits comme un réseau de portes logiques : ce sont l'injection de faute, la D-propagation ou *-propagation et la justification.

4.5. Conclusion

Le principal intérêt de cette méthode repose sur l'adaptation du D-algorithme ou du *-algorithme à des descriptions matérielles de haut niveau. Cette méthode peut être appliquée à tout type de circuit, à contrôle centralisé ou distribué.

Par contre, l'inconvénient majeur de cette méthode est lié à la construction des D-cubes ou des cubes critiques : cette construction nécessite soit des manipulations algébriques complexes ou l'utilisation de tables complexes. En outre, cette technique trouve sa limitation dans la complexité des circuits qui peuvent être traités : un nombre élevé de cas de test entraîne un coût de génération du test élevé.

5. Les Travaux du Virginia Polytechnic Institute

5.1. Principe de base

La méthode proposée par Armstrong et Barclay [Bar86, Arm88, Cha88, O'ne89] est fondée sur le D-algorithme : une liste de fautes est établie, chaque faute est sensibilisée et l'effet de la faute est propagé vers une sortie observable. Les auteurs ont proposé un outil appelé *Générateur de Test Comportemental (BTG)* qui génère un test comportemental à partir d'une description du circuit en langage VHDL.

Les points clés de cette méthode sont la définition de modèles de fautes fonctionnelles et l'utilisation de techniques d'Intelligence Artificielle dans l'algorithme de génération.

La modélisation de fautes est basée sur la notion de *perturbation de modèle* qui permet de prendre en compte tout type de faute : les fautes de collages classiques, les fautes de contrôle et les fautes impliquant la perturbation des opérateurs logiques et arithmétiques (transformation de la fonction exécutée).

En vue du test, la description comportementale VHDL du circuit est transformée en un ensemble de prédicats PROLOG. Cette description est analysée pour obtenir une base de faits qui décrit toutes les informations et les mécanismes nécessaires pour sensibiliser un chemin, et pour construire la liste des fautes comportementales à tester. Le processus de génération est fondé sur la technique d'arbre de buts et de résolution des buts. Trois buts de haut niveau sont définis : justification, exécution et propagation. La base de faits permet de diviser récursivement les buts à atteindre en sous-buts jusqu'à obtenir des buts primitifs tels que la spécification d'entrées ou l'observation de sorties. Une recherche en profondeur est utilisée pour satisfaire les sous-buts, guidée

par des mesures de contrôlabilité et d'observabilité des signaux. Ces mesures de testabilité sont calculées en analysant les différents cheminements des signaux et sont stockées dans la base de faits.

5.2. Représentation du système

La description comportementale VHDL du circuit est traduite en prédicats PROLOG, qui sont ensuite analysés pour construire la base de faits. Armstrong et Barclay travaillent sur des descriptions *data flow* qui comportent seulement des expressions d'affectation concurrentes et qui sont donc proches du comportement réel du circuit. Les outils qui ont été développés utilisent un sous-ensemble de VHDL pour la description du modèle, mais ce sous-ensemble est suffisant pour décrire n'importe quel circuit : la limitation porte seulement sur le niveau d'abstraction. La représentation PROLOG est alors analysée pour extraire les informations de base telles que : les types d'expressions, les noms des objets, les clauses relatives à chaque expression de contrôle, ... Cette information est utilisée pour former les prédicats PROLOG qui représentent la connaissance sur le circuit.

Des mesures de contrôlabilité et d'observabilité sont calculées pour chaque signal. La contrôlabilité d'un signal est déterminée par la position de l'expression source dans le modèle, alors que l'observabilité est mesurée par le nombre d'expressions qui doivent être exécutées afin de pouvoir observer le signal. Le calcul de ces mesures prend en compte un poids associé aux expressions, qui est fondé sur une estimation de la difficulté à exécuter une expression. Si un signal appartient à plusieurs chemins, la mesure de testabilité choisie est la meilleure. Ces mesures de testabilité sont utilisées comme heuristique de sélection des chemins dans les phases de justification et de propagation.

Une liste de fautes comportementales est également établie, en fonction du modèle de fautes et de la description du circuit, et elle est stockée dans la base de faits.

5.3. Le modèle de fautes

Armstrong a proposé une alternative aux modèles de fautes classiques par l'introduction de la notion générale de *perturbation de modèle,* et la liste de ces perturbations constitue la liste des fautes. Une *perturbation de modèle* décrit la transformation d'une construction VHDL en une autre. Au niveau des signaux, le modèle de collage est considéré comme une forme de perturbation de modèle.

Les perturbations de modèle sont divisées en deux classes : les fautes de contrôle qui affectent le séquencement des micro-opérations, et les fautes de micro-opérations qui affectent les opérateurs.

Les *fautes de contrôle* sont modélisées par la perturbation des constructions de contrôle et sont donc dépendantes de l'expression de contrôle :

- si une expression n'appartient pas à un bloc conditionnel, la perturbation est exprimée par la non exécution de cette expression ;
- une expression IF-THEN-ELSE peut être erronée par collage à THEN, par collage à ELSE ou par inversion de la condition ;
- les expressions CASE erronées peuvent s'exprimer par la non exécution de la clause sélectionnée.

Les perturbations de micro-opérations constituent la partie la plus originale du modèle de fautes fonctionnelles. Ce type de fautes a été introduit de manière informelle dans des travaux de recherche antérieurs (fautes fonctionnelles générales de Levendel & Menon) sans avoir été définies de manière précise. La perturbation considère qu'une micro-opération peut être transformée en une ou plusieurs autres micro-opérations.

Les travaux d'Armstrong ont eu pour objectif de prendre en compte diverses perturbations des opérateurs, de déterminer les tests pour chacune des perturbations envisagées et d'évaluer le taux de couverture par rapport aux fautes de collage de chaque ensemble de test, en utilisant le D-algorithme. Cette étude a abouti sur la conclusion qu'il fallait faire une distinction entre opérations logiques et opérations arithmétiques pour la définition de la fonction de perturbation.

De plus, des heuristiques ont été utilisées pour améliorer le taux de couverture. Ces améliorations ont été apportées dans deux domaines :

- *l'affectation des valeurs indéterminées* : les expériences ont montré que l'affectation alternée de 1 et de 0 dans une même position binaire de vecteurs de test successifs, améliore nettement le taux de couverture des collages sur une représentation équivalente de niveau porte logique ;
- *la perturbation sur les opérateurs arithmétiques* : l'analyse de micro-opérations complexes a montré que le taux de couverture atteint avec la perturbation de modèle est peu élevé. Le problème consiste à trouver le meilleur ensemble de valeurs logiques pour les bits indéterminés de chaque vecteur de test. L'heuristique qui a été choisie affecte des valeurs logiques à ces bits de telle sorte qu'il y ait le maximum de transitions 1-0 et 0-1 sur chacun de ces bits.

5.4. La génération de test

L'algorithme de génération est équivalent au D-algorithme. Etant donné une perturbation de modèle, il s'agit de mettre en évidence l'opération erronée et de propager l'effet de la faute vers une expression de transfert sur une sortie.

Le *Générateur de test comportemental* développé est intéressant parce qu'il utilise le concept d'arbre de buts pour organiser la structure de l'algorithme. L'algorithme travaille à partir de trois buts de haut niveau :

- *le but justification* : pour une expression donnée et une valeur donnée V, ce but a pour objet de déterminer l'ensemble des valeurs des signaux pour les objets de l'expression, de manière à mettre la valeur V dans l'expression ;
- *le but exécution* : à un moment donné et pour une expression donnée, ce but place les valeurs nécessaires dans les expressions de contrôle ;
- *le but propagation* choisit un chemin vers une sortie primaire et spécifie toutes les entrées nécessaires pour la propagation de la manifestation de la faute le long du chemin.

Le processus global de sensibilisation utilise les buts de justification, exécution et propagation de manière récursive, jusqu'à ce que la faute soit propagée sur une sortie primaire. Pour chaque but de haut niveau, des conditions de test plus simples sont définies : ce sont des buts de niveau inférieur ou sous-buts ; ces sous-buts sont organisés sous la forme d'un arbre ET-OU, où chaque noeud représente une étape du développement du vecteur de test. Le processus se termine lorsque les buts successifs sont éclatés en *buts primaires* qui sont le chargement d'une entrée ou l'observation d'une sortie. Les sous-buts sont satisfaits par une recherche en profondeur. Un but est résolu soit lorsque c'est un but primaire soit en le subdivisant en un ensemble de sous-`buts qui doivent être à leur tour résolus. La façon dont chaque but peut être satisfait dépend du programme source VHDL. Quand plusieurs possibilités s'offrent pour résoudre un but, des heuristiques sont utilisées pour essayer les choix les plus probables.

L'algorithme de sensibilisation de fautes fournit alors une suite d'informations d'entrée (valeur, date) et une suite de résultats de sortie (valeur, date).

5.5. Conclusion

Le *Générateur de test comportemental* est intéressant pour deux raisons principales. Tout d'abord, il est fondé sur une recherche approfondie de la signification et de la définition de fautes fonctionnelles, particulièrement en ce qui concerne les *fautes fonctionnelles générales*. Les perturbations de modèle proposées ont été validées en estimant leur taux de couverture par rapport au modèle de collage sur une représentation équivalente en portes logiques. D'autre part, l'utilisation des techniques d'arbres de but et résolution de buts semble prometteuse.

Néanmoins, cette technique n'a pas été pour l'instant appliquée sur des circuits complexes tels que les ASICS.

6. Conclusion

Cette synthèse sur les méthodes de test fonctionnel à partir de langages de description de matériel met en évidence le fait que les techniques diffèrent selon le type de système considéré : systèmes à contrôle centralisé et système à contrôle distribué. Ce

dernier type de conception asynchrone constitue une technique de plus en plus employée pour la conception de circuits ASIC en raison des bonnes caractéristiques de rapidité notamment.

Dans tous les cas, il semble évident que l'utilisation de méthodes de test fonctionnel soit une voie d'avenir pour traiter des circuits et systèmes de plus en plus complexes et de moins et moins accessibles et observables. Ce type de test est particulièrement intéressant si on travaille à partir de langages de description déclaratifs de haut niveau, tel que VHDL. Néanmoins, la conception actuelle des circuits devrait mener vers des méthodes de test prenant en compte et même profitant de la hiérarchie des descriptions : la génération du test global se faisant à partir d'une description comportementale ou data flow, les spécifications de test local (au niveau des primitives) étant réalisées à partir d'une description de type transfert registre ou porte logique.

Chapitre 2

Analyse de testabilité et aide à la génération du test : L'outil SATAN

1. Introduction

Dans un processus de conception, des règles de spécification, de conception et de mission sont définies ; pour être utilisables, ces règles vont s'appuyer sur des abstractions (de comportement, de fonctionnement, ...) et comportent donc un certain degré d'imprécision. En dépit de ces règles et en raison de leur imprécision, des fautes seront commises qui nécessitent donc une vérification (validation, qualification, test selon le niveau de conception). La vérification étant elle-même imparfaite, elle doit être associée à une évaluation de son efficacité pour obtenir un certain niveau de confiance. Vu la complexité croissante des équipements électroniques numériques, trois points sont particulièrement critiques :

- la nécessité d'intégrer le paramètre du test dans la chaîne de conception, afin d'automatiser la production des différents programmes de test et de les rendre homogènes ;
- la nécessité d'avoir des programmes de test assurant une bonne localisation selon son contexte d'application : fin de production ou maintenance sur le site ;
- la nécessité d'introduire des mesures de testabilité prédictives (qualitatives et/ou quantitatives) pour obtenir de bonnes performances du programme de génération du test.

La notion générale de test doit intégrer plusieurs notions :

- la *testabilité* qui doit permettre d'évaluer le coût et la difficulté de génération du programme de test et éventuellement guider le concepteur dans la définition d'un équipement facilement testable ;
- la *diagnosabilité* qui exprime le degré de localisation de défauts et qui est une caractéristique complémentaire de la précédente ;
- la *spécification fonctionnelle du programme de test* qui permet de structurer le test relativement à une stratégie adaptée au contexte de test (test de fin de production ou test de maintenance préventive ou curative).

Pour répondre à ces besoins essentiels, le projet SATAN (System Automatic Testability ANalysis) a été développé au sein du Laboratoire de Génie Informatique de

Grenoble. Cet outil se base sur un graphe de transfert d'information de l'équipement à tester. L'utilisation de la notion de quantité d'information permet de limiter les coûts de traitement par l'analyse de la propagation d'une grandeur sans chercher à générer le programme de test comme cela est le cas des outils d'évaluation de testabilité basés sur le D-algorithme.

L'importance du problème d'évaluation de testabilité des équipements actuels, nécessite l'intégration d'un tel outil dans l'environnement de Conception Assistée par Ordinateur (C.A.O.). Et, pour qu'un tel outil soit opérationnel, il faut qu'il soit utilisable sans demander un surcroît de travail de la part des concepteurs ; les mécanismes et les données nécessaires à l'outil devant être transparents pour l'utilisateur, ces données doivent donc être déduites automatiquement à partir des descriptions déjà existantes dans l'environnement de conception. L'évaluation de testabilité devient ainsi une fonctionnalité de l'outil de conception.

Dans ce Chapitre, nous allons introduire les concepts sous-jacents à l'outil SATAN, ainsi que l'organisation générale de l'outil. En effet, cette présentation est nécessaire à la compréhension des Chapitres ultérieurs qui présentent certains traitement intégrés dans l'outil et qui dépendent donc des concepts de bases aussi bien que des objectifs en termes d'analyse de testabilité et d'aide à la génération du test.

2. L'outil d'évaluation de testabilité SATAN

2.1 Concepts

L'outil d'évaluation de testabilité SATAN est fondé sur une représentation de l'équipement à tester sous forme de graphe de transfert d'information (GTI). Le graphe de transfert d'information est défini sur la base d'un graphe biparti orienté (voir l'annexe A) formé de sommets appelés *places*, de sommets appelés *transitions* et d'arcs. Parmi les places, nous distinguons les sources (entrées) génératrices d'information dans le graphe, les puits (sorties) récepteurs d'information et les modules qui correspondent aux fonctions élémentaires du circuit ou de la cellule.

L'utilisation d'une telle représentation de l'équipement dont on cherche à effectuer l'analyse de testabilité induit un certains nombre d'avantages qui ont donné la caractéristique d'évolutivité de l'outil SATAN. En effet, le graphe de transfert d'information est une abstraction du système et peut donc être défini aussi bien à partir d'une description de bas niveau (portes logiques par exemple) que pour une description de niveau plus élevé (transfert de registre, algorithmique) et, de plus, le domaine d'application de l'outil s'en trouve élargi : application aux cartes, circuits ou systèmes de cartes. Dans ce rapport, nous considérerons une hiérarchie simple de la forme cellule/circuit, sachant que, d'une part, des niveaux de hiérarchie multiples peuvent être traités et que, d'autre part, la description peut correspondre à des cartes ou des systèmes.

Ces propriétés donnent donc à l'outil une certaine pérennité vis à vis des évolutions technologiques et méthodologiques dans la conception de systèmes numériques. Ainsi, les mêmes principes appliqués initialement sur des circuits de petite taille décrits au niveau portes logiques sont applicables sur des équipements plus complexes décrits à des niveaux d'abstraction plus grands.

L'outil SATAN est envisagé en liaison avec un environnement de conception de telle sorte que aussi bien l'analyse de testabilité que l'aide à la génération du test puissent être considérées comme des fonctions complémentaires de celles déjà présentes dans un environnement de C.A.O.. Nous distinguons donc trois parties :

- les types et niveaux de description de C.A.O. considérés pour déterminer le GTI d'un circuit ou d'une cellule ;
- la compilation de telles descriptions sous forme de GTI ;
- l'aide à la génération du test et l'analyse de testabilité ;

L'outil SATAN se base sur deux types de descriptions de simulation du circuit : la cellule, décrite par son comportement explicite, et le circuit correspondant à une interconnexion de cellules. Nous distinguons donc deux compilateurs de descriptions sous forme de GTI : le compilateur pour les descriptions de comportement et le compilateur pour les descriptions structurelles. Pour prendre en compte la possibilité

de compiler des descriptions hiérarchiques tout en limitant les coûts de compilation, nous avons défini une bibliothèque d'objets appelés *Type_SATAN*. Le type_SATAN est composé du GTI, appelé *graphe générique*, d'une cellule ainsi que des données relatives aux entrées/sorties de la cellule. Ainsi, la compilation d'une description structurelle consistera à extraire de cette bibliothèque les type_SATAN des cellules utilisées dans le circuit puis à concaténer les GTI à la vue des entrées/sorties et du réseau d'interconnexion.

A partir d'un GTI, l'analyse de testabilité consiste tout d'abord à rechercher les activations fonctionnelles du circuit depuis ses entrées vers ses sorties. Une activation fonctionnelle est appelée chemin d'information, elle correspond à un chemin de test partiel du circuit. Le circuit se trouve ainsi décomposé en un ensemble d'écoulements, chacun recouvrant un certain nombre de fonctions élémentaires du circuit, qui vont constituer la base pour la spécification fonctionnelle du programme de test. Le programme de test sera défini comme une séquence d'écoulements à activer. Par ailleurs, des mesures de testabilité prédictives peuvent être calculées pour chaque fonction élémentaire à l'intérieur d'un écoulement. De plus, l'ensemble des écoulements permet également de déterminer le degré de localisation des fonctions élémentaires, sachant que deux fonctions élémentaires sont discernables si elles peuvent être dissociées à l'aide des écoulements.

La spécification fonctionnelle du programme de test consiste à choisir et ordonner les écoulements pour définir une séquence de test répondant à une stratégie de base qui dépend du contexte de test. Le choix des écoulements prend en compte les mesures de testabilité des fonctions élémentaires.

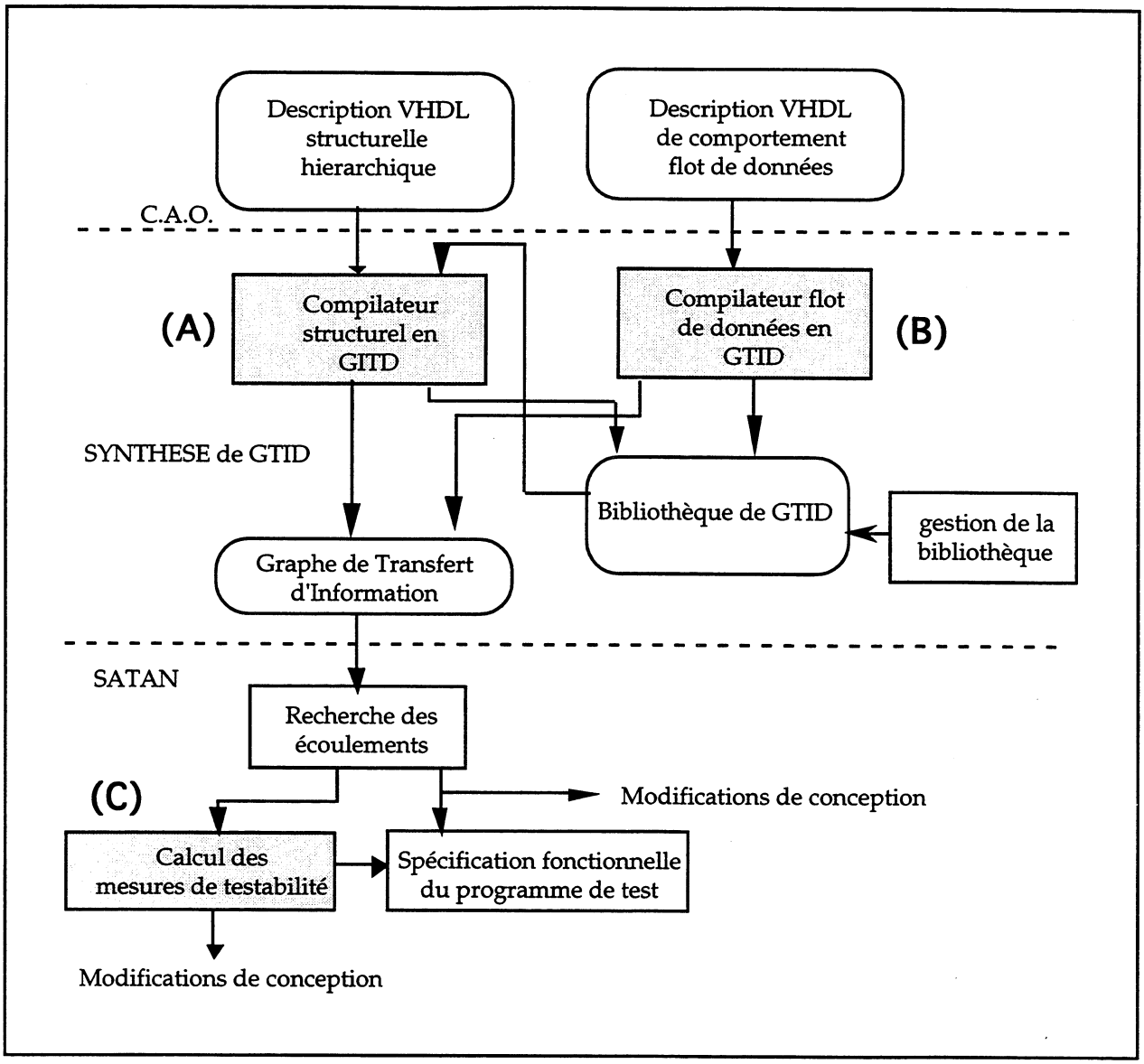

Figure 2.1. L'outil d'analyse de testabilité SATAN.

La partie Synthèse Satan de la figure 2.1 met en évidence la structuration en deux niveaux de la construction du graphe de transfert d'information.

2.2 Modélisation du transfert d'information

La compilation de description de C.A.O. consiste à construire un GTI à partir, soit d'une description de comportement, soit une description structurelle. Le GTI est complété par la définition des entrées/sorties fonctionnelles qui font le lien entre le graphe et le port d'entrée/sortie de l'entité considérée. La forme du graphe, ainsi que

les compilateurs correspondant sont présentés dans les chapitres suivants, nous ne rappelons ici que succinctement les principes de base.

1. Compilation d'une description de comportement

Le GTI d'une cellule a pour objet de représenter les fonctionnalités de la cellule au sein du circuit par le biais d'une description des transferts d'information entre les entrées et sorties fonctionnelles de la cellule.

Le graphe générique est composé de :

- l'ensemble des entrées fonctionnelles ;
- l'ensemble des sorties fonctionnelles ;
- l'ensemble des fonctions internes à la cellule.

Le graphe générique est alors représenté par un graphe biparti orienté dont les noeuds sont des places et des transitions.

Les places sont de trois types :

- les sources qui sont associées aux entrées fonctionnelles de la cellule ;
- les puits qui sont associés aux sorties fonctionnelles de la cellule ;
- les modules qui sont associés aux fonctions internes de la cellule.

Lors de cette étape de compilation, nous avons introduit des phases de simplification qui permettent de diminuer la complexité du graphe résultant en vue de limiter les coûts de traitement ultérieurs. Ces simplifications sont présentées dans le chapitre 3. Les données nécessaires au calcul des mesures de testabilité sont déterminées par des capacités d'information dynamiques qui sont définies dans le chapitre 4.

2. Compilation d'une description structurelle

La modélisation du transfert d'information à travers le circuit est automatiquement réalisée à partir de la description structurelle du circuit et d'une bibliothèque des types_SATAN des cellules du circuit. Cette compilation est présentée dans le chapitre 5.

La construction du graphe de testabilité d'un circuit se déroule en plusieurs étapes :

- la connexion de signaux d'entrée à des valeurs constantes peut mener à la réduction du fonctionnement d'une cellule. Une première étape consiste donc à analyser ces signaux constants et à simplifier les graphes génériques. le résultat de cette simplification est appelé *graphe spécifique partiel (GSP)* ;
- déterminer les transferts d'information entre cellules et avec les connecteurs externes, par une analyse du réseau de connexion, relativement aux entrées et sorties fonctionnelles de chaque cellule. Cette phase donne lieu à la création d'un *graphe spécifique final (GSF)* pour chaque cellule du circuit ;
- établir les liens entre les GSF de manière à obtenir le graphe de testabilité global du circuit.

La seconde phase permet d'établir les liaisons entre les graphes spécifiques partiels, en analysant les lignes de données et de contrôle du réseau d'interconnexion. Plus précisément, cette phase a pour objet de générer, pour chaque GSP, les origines des entrées fonctionnelles et des destinations des sorties fonctionnelles.

Cette étape de construction du graphe spécifique final de chaque cellule est fondée sur :

- une analyse des entrées fonctionnelles,
- une analyse des sorties fonctionnelles.

L'information nécessaire sur l'entrée fonctionnelle d'une cellule peut provenir soit de cellules adjacentes soit des entrées primaires du circuit. L'objectif est de spécifier l'information requise sur chaque entrée fonctionnelle par rapport aux différentes origines. Le processus d'analyse des entrées fonctionnelles consiste à déterminer de quelle manière cette information peut être générée et à distinguer les origines possibles de cette information.

En ce qui concerne l'analyse des sorties fonctionnelles, le processus est dual du précédent. L'information issue d'une sortie fonctionnelle peut être émise soit vers des cellules adjacentes soit vers des sorties primaires du circuit. L'objectif est alors de spécifier cette information de sortie par rapport aux différentes destinations. Ce processus fait intervenir les notions de *distribution* ou de *sélection* d'information et la définition de sous-ensembles des sorties fonctionnelles appelés *groupes de sortie*.

La génération du graphe de testabilité global du circuit se résume ensuite à une concaténation des graphes spécifiques finaux de toutes les cellules.

2.3 Analyse de testabilité et aide à la génération du test

L'analyse de testabilité et la spécification fonctionnelle du programme de test d'un circuit sont réalisées automatiquement, sur le graphe de testabilité du circuit, par le logiciel SATAN. Cette phase comporte deux étapes essentielles :

- la recherche des activations fonctionnelles élémentaires du circuit (appelées écoulements) et l'évaluation de testabilité des différentes cellules ;
- la structuration du programme de test sous la forme d'une séquence d'écoulements, par le biais d'une stratégie de test, et l'évaluation de la qualité du diagnostic.

1. Recherche des écoulements

L'objectif est de rechercher, à partir des fonctions locales exécutables par chacune des cellules, l'ensemble des activations fonctionnelles au niveau du circuit global. Le graphe de testabilité du circuit, qui modélise les transferts d'information entre entrées et sorties primaires, permet de déterminer les différents chemins d'information activables dans le circuit : ces chemins d'information sont appelés *écoulements*.

Un écoulement, qui caractérise une activation fonctionnelle particulière du circuit, définit les entrées du circuit à activer pour sensibiliser le chemin d'information spécifique de cette activation, les sorties du circuit qui émettent le résultat de l'exécution de cette fonction et les cellules activées lors de l'exécution de la fonction. L'algorithme de recherche des écoulements dans un graphe biparti orienté est donné en détail dans [ROB 79].

L'ensemble des écoulements spécifiant les fonctionnements élémentaires, un programme de test pourra donc être spécifié en termes d'écoulements.

2. Stratégie de test

L'objectif est de rechercher, parmi l'ensemble des activations fonctionnelles possibles, le sous-ensemble d'écoulements nécessaires et suffisants pour assurer le test complet du circuit. Le programme de test est alors spécifié comme une séquence d'écoulements.

Ce choix d'écoulements est réalisé par le biais d'une *stratégie de test*, qui prend en compte le degré de diagnostic désiré et le contexte de vérification : fabrication, production ou maintenance.

L'utilisation d'une stratégie permet une structuration fonctionnelle du programme de test qui facilite, d'une part, l'écriture proprement dite du programme et, d'autre part, le suivi lors de modifications de conception. Elle implique également une réduction du coût de génération du programme de test puisqu'elle permet de prendre en compte la décomposition du circuit en fonctions. Ces fonctions peuvent être plus aisément manipulées, soit par des simulateurs de fautes, soit par des ATPGs.

Les stratégies implantées dans SATAN sont soit du type Start-Small pour les tests de conception et de production, soit du type Multiple-Clue (test par recoupements) pour les tests de maintenance [DEN 68].

3. Analyse de testabilité

Deux types de renseignements peuvent être obtenus à partir de la connaissance des écoulements du circuit :

- une évaluation de la testabilité des différentes cellules du circuit selon l'écoulement considéré,
- une pré-évaluation de la diagnosabilité des fautes à la cellule près.

L'évaluation de testabilité est fondée sur une estimation de la contrôlabilité et de l'observabilité des cellules dans les différents écoulements. Ces évaluations sont données, respectivement, en termes de quantité d'information accessible sur l'entrée fonctionnelle d'une cellule à partir des entrées primaires de l'écoulement, et de quantité d'information observable sur les sorties primaires de l'écoulement à partir de la sortie

fonctionnelle de la cellule. Les principes de la méthode utilisée sont détaillés en [ROB86, DAM85].

La pré-évaluation de diagnosabilité permet de définir le degré de résolution du diagnostic, c'est-à-dire d'évaluer la capacité à localiser une faute au niveau d'une cellule. Cette évaluation introduit la notion de modules indiscernables par rapport au diagnostic. Pour un écoulement nous pouvons considérer deux paramètres vis-à-vis des modules recouverts et de sa position dans la spécification fonctionnelle du programme de test. A savoir :

- les modules activés par l'écoulement ;
- les modules activés qui ne sont pas recouverts par les écoulements prédécesseurs dans la spécification du programme de test.

Le premier paramètre indique une notion de complexité globale de l'écoulement alors que le second paramètre indique une complexité en termes d'effort de test de l'écoulement. En effet, les vecteurs de test choisis pour un écoulement doivent permettre en premier lieu de tester les modules qui n'ont pas encore été testés dans les pas précédents du programme de test.

Nous pouvons donc, pour une spécification fonctionnelle du test, définir la notion de granularité appelée aussi degré de localisation relatif. Ces mesures associent aux différents pas d'un programme de test la taille en modules à tester. Les modules qui doivent être testés dans un même pas du programme de test sont dits indiscernables entre eux car deux écoulements ne permettent pas de les dissocier. Ainsi, plus le nombre de modules indiscernables est grand plus l'effort de génération du test et de localisation de défaut sera important.

Les mesures de testabilité et de diagnosabilité fournies sont intéressantes à plus d'un titre :

- elles permettent d'envisager des modifications sélectives de la conception, par l'adjonction de points d'accès ou d'observation, soit au niveau des cellules dont la contrôlabilité ou l'observabilité sont mauvaises, soit à l'intérieur d'un bloc

d'indiscernabilité. Après simulation des modifications, le logiciel permet d'évaluer les gains de testabilité et/ou de diagnosabilité ;

- elles peuvent être utilisées comme paramètres dans le choix des écoulements qui doivent être utilisés pour tester le circuit, et plus particulièrement, orienter vers le choix de fonctions multiples pour le test d'une cellule donnée (chaque fonction assurant un test partiel).

3. Conclusion

Les concepts de cet outil ont été appliqués sur deux études de cas. Une première étude [WOD90] concernait un ASIC, réalisant une file d'attente en mode FIFO, développé par la société THOMSON TMS. La complexité du circuit était d'environ deux mille portes et bascules. La seconde étude de cas effectuée pour la société SEXTANT-AVIONIQUE concernait un ASIC de communication ARINC d'une complexité d'environ quatre mille portes et bascules. Ces études de cas se sont avérées probantes quant à l'aide que l'outil SATAN peut apporter à un concepteur tant au niveau de l'évaluation de testabilité qu'au niveau de la génération du programme de test.

De par les mesures de testabilité, cette approche a permis de mettre en évidence les parties des circuits qui demandèrent un effort important pour la génération du test. De par la structuration du programme de test en séquences d'écoulements l'outil SATAN constitue une aide appréciable pour la détermination d'un programme de test par le traitement de sous blocs des circuits. Sans ces aides, le concepteur se trouvait confronté à un problème difficile à gérer par l'utilisation de simulateurs de pannes qui ne donnent pas une vision globale du circuit.

Ce projet a de plus été développé partiellement en collaboration avec l'AEROSPATIALE et avec le CNES. Leur intérêt pour un tel outil réside dans l'intégration de évaluation de testabilité dans la conception, ceci par le biais de traducteurs adéquats.

Chapitre 3

Graphe de Transfert d'Information Statique

1. Introduction

La construction du graphe de transfert d'information d'une description permet de prendre en compte les possibilités de décomposition hiérarchique. Dans ce but, les descriptions, une fois compilées, sont introduites dans une bibliothèque spécifique : la bibliothèque SATAN. L'organisation générale de l'outil SATAN est décrite dans le Chapitre 2 de ce document.

Les éléments de la bibliothèque, appelés types SATAN, sont extraits lors de l'instantiation d'un composant dans une description structurelle. Dans le cadre de la compilation de descriptions VHDL sous forme de graphes de transfert d'information, ce chapitre fait l'objet de la compilation des descriptions de comportement pour les cellules de niveau de hiérarchie le plus bas. Il s'agit donc de construire le Type SATAN d'une description de comportement.

Nous rappelons que le type SATAN d'une cellule est défini par un triplet composé des champs suivants :

- le nom du type SATAN. Il constitue le moyen d'accès à l'élément dans la bibliothèque. Le nom est le même que celui de l'entité VHDL ;
- les entrées et les sorties fonctionnelles sont les déclarations qui définissent les relations entre les identificateurs de signaux externes et les identificateurs places génératrices ou réceptrices d'information du graphe de transfert d'information. Ces déclarations sont nécessaires à la compilation de descriptions structurelles ;
- le graphe de transfert d'information Statique.

Une description VHDL est constituée de deux objets, l'un est nommé entité, l'autre architecture. La définition de l'objet entité d'une description VHDL est la déclaration des entrées/sorties, de leur sens de transfert et de leur type. La définition d'une architecture est, pour le compilateur que nous présentons dans ce chapitre, constituée par la définition des signaux locaux et de leur type ainsi que par la description des affectations concurrentes modélisant le comportement.

Relativement à un type SATAN, les deux premiers champs, nom du type et définition des entrées/sorties fonctionnelles sont déduites de la déclaration d'entité. Ce traitement relativement simple est décrit dans le paragraphe 3 de ce chapitre et il est commun avec le chapitre 4.

Pour le corps d'architecture nous présentons ici uniquement la compilation des instructions d'affectation concurrentes, les déclaration locales donnant un environnement implicite que nous ne préciserons pas dans cette thèse.

Le corps d'architecture est supposé être décrit par le sous-ensemble du langage VHDL qui est présenté dans le paragraphe 4. L'utilisation d'un analyseur VHDL nous a amené à définir une première syntaxe abstraite <VHDL1> sur laquelle nous effectuons la compilation. Pour la réalisation nous nous sommes basés sur le compilateur VHDL en format intermédiaire développé par la société CLSI. Ce logiciel effectue une analyse lexicale, syntaxique et sémantique d'une description VHDL, puis insère l'arbre correspondant dans une bibliothèque. Des fonctions d'accès et de parcours de la description compilée permettent de développer un outil tel notre compilateur.

La figure 3.1 montre l'organisation interne du traducteur de descriptions VHDL concurrentes.

Dans ce chapitre, nous nous intéresserons à la construction de Graphes de Transfert d'Information Statiques (GTIS). Nous définissons donc tout d'abord le GTIS pour ensuite présenter les différentes étapes de traduction d'une description de comportement. Les étapes de traduction mises en évidence dans la figure 3.1 concernent aussi bien des propriétés que doit posséder un GTIS que la volonté de limiter la taille du graphe résultant.

En effet, les modules du graphes seront associés à des expressions, comme nous l'avons introduit dans le chapitre 2, il s'agit donc ici de spécifier la forme de ces expressions de manière à bien mettre en évidence les problèmes de testabilité ou de spécification fonctionnelle du programme de test. Nous avons donc défini une autre syntaxe abstraite <VHDL 2> ainsi que le traducteur à partir de la syntaxe abstraite initiale <VHDL 1>.

De plus, les traitements nécessaires à l'évaluation de testabilité, la recherche des écoulements notamment, ont un coût important vis-à-vis de la taille du graphe. Par conséquent, nous avons défini une phase de simplification des expressions puis une phase de recherche des expressions communes dans la description.

Dans ce chapitre, nous définissons tout d'abord le graphe de transfert d'information statique, puis nous détaillons le traitement associé à la déclaration d'entité qui constitue la définition du nom de la description et de son port d'entrée/sortie.

La partie suivante montre la traduction des expressions d'une description de comportement, elle précise comment construire le graphe ainsi que les simplifications que nous mettons en oeuvre.

Enfin, nous montrons comment optimiser la taille du graphe en recherchant les expressions ayant plusieurs occurrences dans la description VHDL.

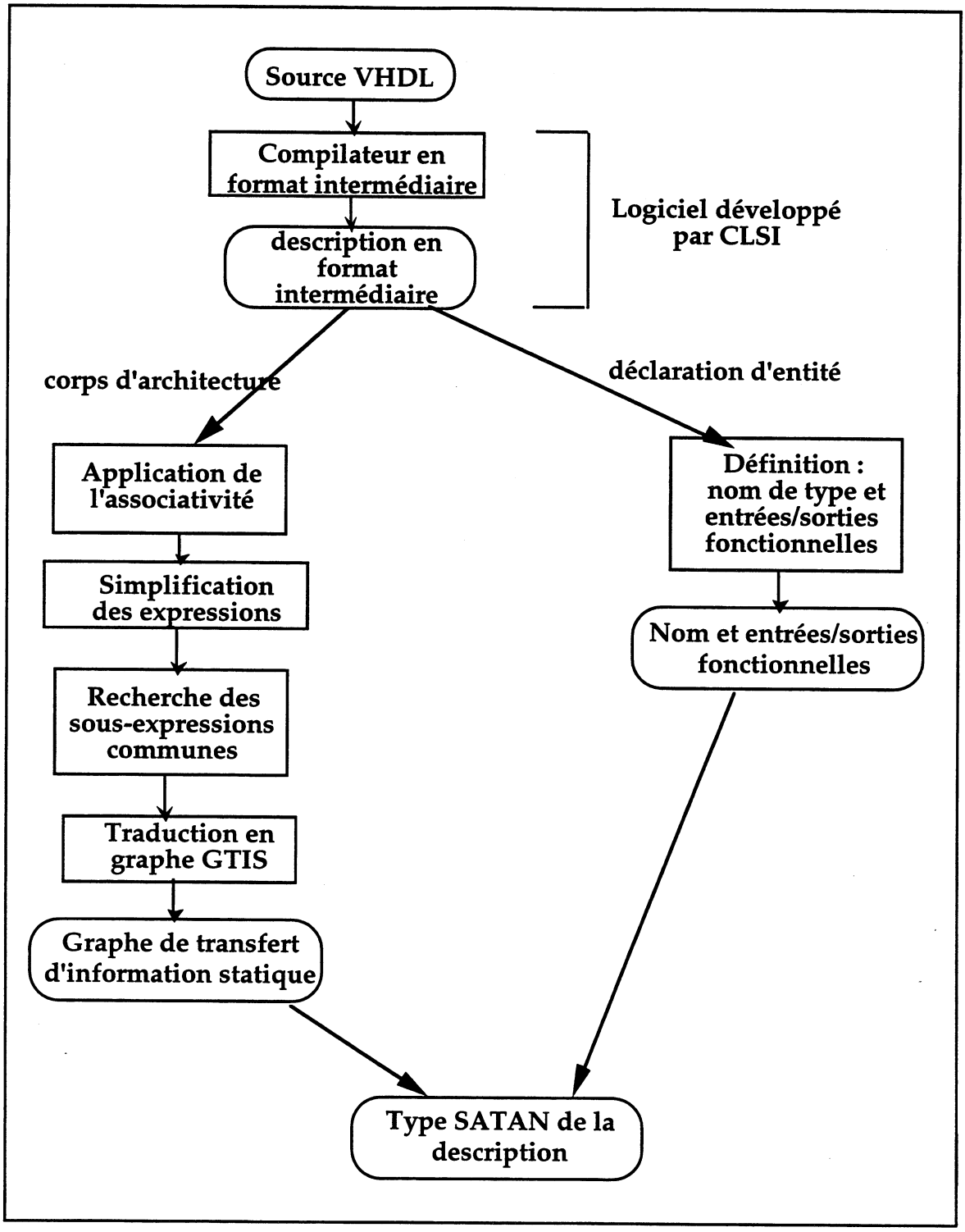

Figure 3.1. Compilation d'une description VHDL concurrente.

2. Graphe de transfert d'information statique

Comme nous l'avons spécifié précédemment, le type SATAN d'une cellule comporte une composante, définie par le graphe de transfert d'information, décrivant les transferts d'information à travers une cellule. Dans cette partie, nous définissons

formellement le Graphe de transfert d'information statique (GTIS) qui constitue la représentation de base (les capacités d'information ne sont pas déterminées).

2.1. Définitions

Les graphes de transfert d'information statiques sont définis à partir des graphes bipartis orientés qui sont eux-mêmes définis en Annexe A. Un GTIS est un graphe biparti donc les places sont associées à des expressions par une fonction. Ces expressions sont déduites à partir des descriptions VHDL.

Nous notons $\mathcal{E}$ l'ensemble des expressions. Les places et transitions sont considérées comme nommées, nous définissons ainsi *ID* l'ensemble de tous les identificateurs possibles.

Définition 3.1. Un graphe de transfert d'information statique est un quintuplet $G = (P, T, \alpha, \beta, Expr)$ définissant un graphe biparti orienté dont les places sont reliées à des expressions. Les composantes du quintuplet sont :

- (P,T,α,β), un graphe biparti orienté (cf. Annexe A) ;
- *Expr*, la fonction qui, à un identificateur de place, associe l'expression de transformation de données ou d'information. *Expr* est une fonction de *ID* vers $\mathcal{E}$.

Les propriétés sont :

- $\forall t \in T, \Gamma^{+}_{t,G}(t) \neq \varnothing \quad et \quad \Gamma^{-}_{t,G}(t) \neq \varnothing$, toute transition a au moins une place prédécesseur et au moins une place successeur ;
- $\forall p \in P, \Gamma^{+}_{p,G}(p) \neq \varnothing \quad ou \quad \Gamma^{-}_{p,G}(p) \neq \varnothing$, toute place a au moins une transition prédécesseur ou successeur ;
- l'ensemble des sources du graphe, défini par $S(G) = \{p \in P / \Gamma^{-}_{p,G}(p) = \varnothing\}$, est non vide ;
- l'ensemble des puits du graphe, défini par $R(G) = \{p \in P / \Gamma^{+}_{p,G}(p) = \varnothing\}$,est non vide.

Définition 3.2. Une place p d'un GTIS G est un **module**, si et seulement si, elle a au moins une transition successeur et au moins une transition prédécesseur. On notera m

un module et $M(G)$ l'ensemble des modules du graphe G. $M(G)$ est défini par : $M(G) = \{p \in P / \Gamma^{+}_{p,G}(p) \neq \varnothing \quad et \quad \Gamma^{-}_{p,G}(p) \neq \varnothing\}$. Les modules sont associés aux opérateurs de transformation d'information du graphe G.

Définition 3.3. *Sémantique des éléments d'un graphe.* Les constituants d'un graphe de transfert d'information ont une sémantique qui est la suivante :

- une source est considérée comme un générateur d'information dans le graphe : une source sera donc associée à une entrée du système numérique représenté ;
- un puits est un récepteur d'information du graphe : un puits sera donc associé à une sortie du système représenté ;
- un module est un opérateur de transformation d'information : un module sera donc associé à une fonction interne du graphe ;
- les transitions représentent les modes de transfert d'information entre les places.

3. Compilation de la déclaration d'entité

3.1. Syntaxe VHDL

Dans la déclaration d'entité, nous considérons uniquement la définition du nom de l'entité et la définition du port d'entrée/sortie. Ci-dessous nous présentons la syntaxe VHDL correspondante.

```
EntDecl ::= entity nome is port (ListeDeclExt) end [nome];

ListeDeclExt ::= DeclExt | DeclExt ; ListeDeclExt;

DeclExt ::= sig : sens Type

sens ::= in | out | inout | buffer | link

Type ::= integer | boolean | real | scalar_type | array_type
```

3.2. Détermination des entrées/sorties fonctionnelles

Dans ce paragraphe nous montrons la détermination des places sources et puits d'un graphe de transfert d'information. Ces places sont associées aux signaux externes

d'entrée (sources) ou à des signaux externes de sortie (puits). Comme nous l'avons présenté dans le chapitre 2, les entrées/sorties fonctionnelles sont exploitées lors de la compilation de descriptions structurelles. Au cours de cette étape, il faudra être capable de concaténer des graphes de transfert d'information par la concaténation de leurs sources et de leurs puits en fonction du réseau d'interconnexion entre les composants. Pour pouvoir effectuer cette opération, il est donc nécessaire de connaître le nom de la place associée à chacun des signaux externes.

Or le lien ne peut être simplement effectué en nommant une source ou un puits avec le même identificateur que le signal externe associé. En effet, les signaux externes peuvent être bidirectionnels, et dans ce cas ils seront liés à la fois à une source et à un puits qui ne peuvent posséder le même nom pour des raisons d'unicité. De plus, dans le cas de signaux de type tableau, le réseau d'interconnexion permet de décrire des liens partiels dans lesquels chaque composante du tableau peut être reliée à une origine ou à une destination distincte. Les entrées et les sorties fonctionnelles décrivent explicitement la liste de signaux, chacun correspondant à une composante du tableau, associés à une source et/ou un puits.

Une entrée (respectivement une sortie) fonctionnelle définit donc le lien entre un identificateur de source (resp. puits) et une liste non vide de signaux d'entrée (resp. de sortie).

Ces définitions sont obtenues par l'analyse de l'interface de la déclaration d'entité d'une cellule. Considérons donc la déclaration d'un signal externe sous la forme générique définie par la règle de grammaire "DeclExt".

Pour déterminer l'entrée et/ou la sortie fonctionnelle à partir d'une déclaration, il faut tout d'abord déterminer le(s) nom(s) de la source et/ou du puits généré(s), et ensuite l'association entre ce(s) nom(s) et une liste de signaux élémentaires.

3.2.1. Nom des sources et des puits

L'identificateur d'une source et/ou d'un puits sera déterminé par le nom et le mode de transfert du signal de la façon suivante :

- si le sens est *in* (resp. *out*), le nom de la source (resp. puits) sera le même que le nom du signal ;
- si le sens est *inout, buffer* ou *link*, le signal sera associé à une source et à un puits ayant pour nom le nom du signal préfixé par *S* pour la source et *P* pour le puits. De façon à générer un nouvel identificateur, une séquence de *S* ou de *P* peut être préfixée à l'identificateur du signal.

3.2.2. Lien entre places et signaux de type construit

Soit *p* l'identificateur d'une place (source ou puits) associée à un signal externe *sig*.

La liste de signaux associés à *p* dépend du type du signal :

- si le type est d'un type simple, l'association est *(p,(sig))* ;
- si le type du signal est un vecteur avec comme intervalle d'indices *[a, b]*, alors *p* sera associé à la liste $l = (s(a), s(a+1), \ldots, s(b)) = S(i)_{a \leq i \leq b}$.

4. Compilation des expressions décrites par l'architecture

Nous allons ici décrire les différentes étapes de traduction d'une description du corps d'architecture. Nous rappelons tout d'abord la syntaxe VHDL des descriptions. nous avons limité notre étude au sous-ensemble des expressions d'affectation concurrentes. Ainsi seuls des signaux sont manipulés et le seul le temps de simulation intervient.

4.1. Syntaxe VHDL

Nous présentons ci-dessous la syntaxe du sous-ensemble d'expressions concurrentes que nous avons considéré dans ce projet de thèse. Nous ne rappelons toutefois pas les déclarations des signaux locaux.

```
ArchDecl ::= architecture noma of nome is
                    ListeDeclLoc
                    begin Block end [noma];
Block ::= Statement | Statement ; Block
```

```
Statement ::=    ConcAssStatement | CondAssStatement |
                 SelectAssStatement | BlockStatement
ConcAssStatement ::= s <= [guarded] Waveform;
CondAssStatement ::= s <= [guarded] Waveform when Expression
else
                                 {Waveform when Expression else}
                                 Waveform;
SelectAssStatement ::= with Expression select
                        s <= Waveform when ChoiceList
                                 {,Waveform when ChoiceList};
BlockStatement ::= Label : block (Expression)
                        ListeDeclLoc
                        begin
                        Block
                        end block ;
Waveform ::=     Expression after TimeExpression
Expression ::=   constante | s | s'ATTRIBUT |
                 Expression bop Expression | uop Expression
TimeExpression ::= Expression unite
ChoiceList := others | ListeValue
ListeValue ::= const | const ListeValue1
Label ::= Nom
```

La syntaxe abstraite ci-dessous appelée <VHDL1> est celle qui correspond à la structure d'arbre obtenue par l'analyseur (compilateur en format intermédiaire) développé par la société CLSI, avec l'introduction d'opérateurs : *lgarde, seln,* et les clauses $\in$ et $\notin$.

```
<arch> ::=       (◁ affectations)
<affectations> ::= (<= s expr) {<= s expr}
```

<expr> ::= s | const | (<uop> <expr>) | (<bop> <expr$_1$> <expr$_2$>) |
(<nop> <expr$_1$> ... <expr$_n$>) | (<top> <expr$_1$> <expr$_2$> <expr$_3$>)

où :

<uop> ::= AttOp | not | opposé | inverse | complément | event

<bop> ::= + | - | * | / | div | mod | and | or | xor | xand | = |
compare | garde

<top> ::= cond | $\in$ | $\notin$

<nop> ::= seln | l_garde

La plupart des opérateurs utilisés dans cette grammaire sont directement associés aux opérateurs du langage VHDL. Toutefois, nous précisons la signification des opérateurs "l_garde" et "seln" avec $\in$ et $\notin$.

L'opérateur "l_garde" prend pour arguments les différentes sources, dans le cas où il y en a plusieurs, d'un signal. "l_garde" est donc toujours associé à une affectation d'un signal. Dans le cas de l'opérateur "l_garde", nous supposons que les différentes sources sont exclusives par le biais d'affectations gardées.

a) Exemple

La représentation, dans la syntaxe <VHDL1>, des deux expressions VHDL ci-dessous est donnée par l'arbre de la figure 3.2.

```
B1 : (x)
     begin
          S <= guarded A;
     end block ;
B2 : (not x)
     begin
          S <= guarded B;
     end block;
```

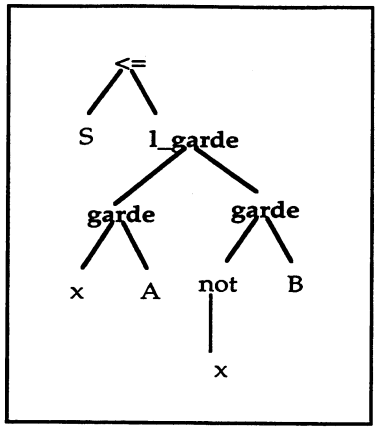

Figure 3.2. Expression avec opérateur l_garde.

A partir d'une expression "**with** expr **select**" et d'une affectation avec une liste de clauses, on génère une expression "seln" qui a pour arguments autant d'expressions qu'il y a de clauses, ces expressions étant soit de la forme (∈ expr expr1 liste) ou de la forme (∉ expr expr1 liste) si la clause est de la forme "**others**". La signification est que cette expression renvoie la valeur résultante de "expr1" si la condition "expr ∈ liste" ou "expr ∉ liste" est vraie.

b) Exemple

L'expression avec sélecteur de champ ci-dessous conduit à l'arbre de la figure 3.3.

```
with x select
        S <=   A when 0, 1, 2, 3 else
               B when 4, 5, 6, 7 else
               C when others;
```

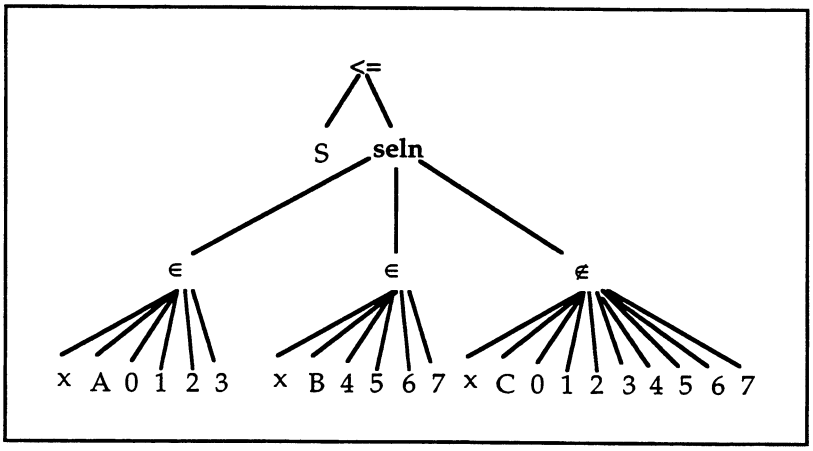

Figure 3.3. Expression avec opérateur sélection.

4.2. Définitions

Définition 3.4. *Opérateurs avec loi de groupe.* Nous représentons les opérateurs associatifs par le symbole +. Selon la théorie de l'opérateur : l'élément neutre est noté *0*, l'inverse - et nous notons *anti+* l'opération binaire correspondant à $anti+(x,y)=x+(-y)$. Dans le cas où l'opérateur possède également un élément absorbant, et dans le cas où nous devons le différencier, nous noterons : * l'opérateur, 1 l'élément neutre, 0 l'élément absorbant, √ l'inverse et *anti** l'opérateur binaire définit de façon similaire à *anti+*.

Définition 3.5. *Symbole terminal.* Une expression de la forme <s> (identificateur de signal) ou <const> (référence à une valeur) sera appelée **symbole terminal**. Pour une expression φ, nous appelons $Ter(\varphi)$ l'ensemble des symboles terminaux et $Sig(\varphi)$ le sous-ensemble de $Ter(\varphi)$ des symboles terminaux dénotant des identificateurs de signaux.

Définition 3.6. *Opérateur d'une expression.* Soit φ une expression, l'opérateur associé à φ est dénoté par $Op(\varphi)$. Si φ est un symbole terminal alors l'opérateur est ce symbole : soit un identificateur de signal, soit une valeur constante.

Définition 3.7. *Arguments d'une expression.* Soit φ une expression, $Arg(\varphi)$ est la liste des arguments de l'expression φ. C'est aussi l'ensemble des sous-expressions directes. Si φ est un symbole terminal alors $Arg(\varphi)$ dénote la liste vide. .

Définition 3.8. *Profondeur d'une expression.* Soit φ une expression, $Prof(\varphi)$ dénote sa profondeur et $|\varphi|$ dénote sa longueur, c'est à dire le nombre de symboles (opérateurs, identificateurs et constantes) rencontrés lors du parcours de φ. La profondeur d'une expression ψ est inductivement définie de la manière suivante :

(i) si $\varphi \in Ter(\psi)$ alors $Prof(\varphi) = 0$

(ii) sinon $Prof(\varphi) = 1 + \underset{\varphi' \in Arg(\varphi)}{Max} (Prof(\varphi'))$

Définition 3.9. *Familles d'expressions.* Soit ψ une expression :

- la famille des sous-expressions de ψ est dénoté $SE(\psi)$,

- la famille des sous-expressions strictes, c'est-à-dire différentes de ψ elle-même, est dénoté $SE^{-}(\psi)$,
- la famille des sous-expressions de ψ de profondeur supérieure ou égale à 1 est dénoté $SE^{+}(\psi)$,
- la famille des sous-expressions de profondeur égale à 1 est dénoté $SE^{1}(\psi)$. Une telle expression sera appelée *terminale*. Nous définissons également la fonction *Terminale* définie sur l'ensemble $\mathcal{E}$ qui renvoie vrai si l'argument est une expression terminale.

Ces définitions sont données en termes de famille ce qui signifie que si une expression φ a plusieurs occurrences dans une expression ψ, elle sera plusieurs fois représentée dans la famille.

Notations. Dans la suite de ce chapitre, nous présentons certaines transformations sur les expressions à l'aide d'un système de règles d'inférence. Les règles d'inférence représentent les transformations élémentaires et une expression ψ se traduisant en une expression φ se note $\psi \rightarrow \varphi$. Les règles d'inférences sont représentées par : $\frac{\alpha}{\psi \rightarrow \psi'}$, qui signifie que l'expression de la forme ψ se traduit en l'expression ψ' si la condition α est vérifiée. Une règle d'inférence sans condition d'activation est appelée un axiome et notée comme suit : $\overline{\psi \rightarrow \psi'}$.

Pour les systèmes d'inférence que nous présentons nous considérons la présence de la règle d'inférence fondamentale ci-dessous :

$$\frac{\varphi_i \rightarrow \xi}{(\omega\ \varphi_1 \ldots \varphi_i \ldots \varphi_n) \rightarrow (\omega\ \varphi_1 \ldots \varphi_{i-1}\ \xi\ \varphi_{i+1} \ldots \varphi_n)} \qquad (0)$$

4.3. Transformation des expressions

4.3.1. Application de la propriété d'associativité

Les expressions de calcul (arithmétiques et logiques) sont associées à certaines places. Ici se pose le choix du nombre de places qui vont être générées pour une expression donnée : soit toute l'expression est représentée par une seule place, soit

chaque noeud de l'arbre, décrit par la syntaxe <VHDL 1>, donne lieu à la création d'une place. La première possibilité possède l'avantage d'une certaine compaction du graphe, peu de places générées pour des expressions même complexes. En contre partie, ce choix induirait, d'une part, une hétérogénéité dans la complexité des places et, d'autre part, un manque de précision sur les opérations internes à une place (toute une description pourrait se résumer à une seule place !).

Par contre, une représentation sous forme d'un graphe où à chaque noeud induit la création d'une place conduit à un graphe dont le nombre de places est maximal. Nous avons donc opté pour une solution intermédiaire dans laquelle l'expression liée à une place ne comporte qu'un seul opérateur. Mais, si l'opérateur est associatif, cette propriété est appliquée de manière à ce qu'aucun argument ne soit une expression de même opérateur. Ceci conduit à considérer les opérateurs associatifs comme des opérateurs d'arité variable. La syntaxe <VHDL 2> décrit la nouvelle forme des expressions. Nous appelons *maximalité* la propriété des expressions ainsi transformées.

a) Exemple

Considérons l'expression $(x + y) + (z + u)$, dont l'arbre sous la forme syntaxique <VHDL 1> est donné dans la figure 3.2 (a). La forme minimale dans la syntaxe <VHDL 2> est donnée par la figure 3.2 (b), ce qui correspond à l'expression préfixée : $(+\ x\ y\ z\ u)$.

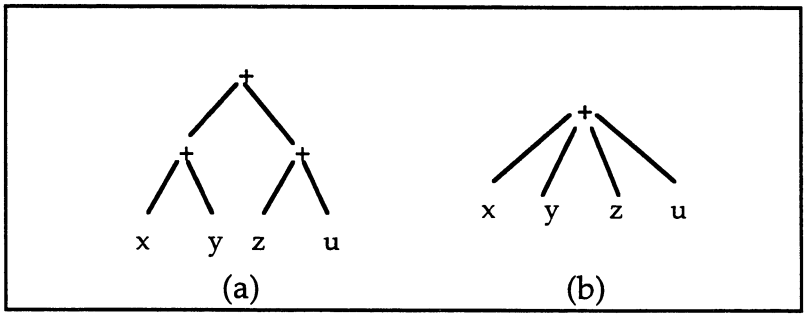

Figure 3.4. Application de l'associativité.

b) Syntaxe abstraite VHDL2

```
<arch> ::=      (◁ affectations)

<affectations> ::= (<= s expr) {<= s expr}
```

<expr> ::= s | <const> | (<uop> <expr>) | (<bop> expr$_1$> <expr$_2$ >) |

(<nop> <expr$_1$> … <expr$_n$>) | (<top> <expr$_1$> <expr$_2$> <expr$_3$>)

où :

<uop> ::= AttOp | not | opposé | inverse | complément

<bop> ::= - | / | compare | garde | = | ≠ | < | ≤ | > | ≥

<top> ::= cond

<nop> ::= seln | l_garde | * | + | and | or | xor | xand

Pour passer d'une représentation initiale de la syntaxe <VHDL 1> à une représentation dans laquelle les expressions d'opérateur associatif sont maximalisées, nous définissons la fonction par la *règle d'inférence* suivante :

$$\frac{\varphi_i = (+\ \xi_1 \dots \xi_k)}{(+\ \varphi_1 \dots \varphi_i \dots \varphi_n) \rightarrow (+\ \varphi_1 \dots \varphi_{i-1}\ \xi_1 \dots \xi_k\ \varphi_{i+1} \dots \varphi_n)}$$

4.3.2. Complexité en temps

Définition 3.11. Nous notons Ω_A l'ensemble des opérateurs associatifs et nous définissons la famille des expressions dont l'opérateur est associatif par : $SE^A(\psi) = \{\varphi \in SE(\psi) / Op(\varphi) \in \Omega_A\}$

Théorème. Le temps de conversion d'une expression ψ en une expression maximalisée ψ' est $T_{max} \leq |\psi| + |\Omega_A|.|SE^+(\psi)| + 2.|SE^A(\psi)| + \frac{(|\psi| - |\psi^1|)(|\psi| - |\psi^1| + 1)}{2}$.

Démonstration.

Soit ψ une expression de la forme syntaxique initiale et ψ' l'expression résultante de la traduction de ψ en expression maximalisée.

Le temps maximum de comparaison de l'appartenance de l'opérateur d'une expression φ non atomique est $|\Omega_A|$. Le nombre de tests effectués est égal au nombre d'expressions de ψ qui ne sont pas des symboles terminaux, soit $|SE^+(\psi)|$. Le coût de détermination de l'atomicité ou non d'une expression est de 1. Il en résulte que $|\psi|$ unités de temps sont nécessaires pour le parcours de ψ.

Dans le cas où une expression est non atomique et d'opérateur associatif, il faut tester les deux sous-expressions directes pour déterminer la nécessité d'une opération de concaténation ou non. Ceci conduit à un coût de $2\left|SE^A(\psi)\right|$. Le nombre de concaténations d'arguments effectuées est égal à $|\psi|-|\psi'|$, ce qui correspond aussi au nombre d'opérateurs supprimés dans ψ pour obtenir ψ'. Le coût d'une concaténation de deux listes d'arguments de longueur l_1 et l_2 est égal à l_1. Donc, globalement le coût en opérations de concaténation est majoré par $\sum_{i=1}^{|\psi|-|\psi'|} i = \frac{(|\psi|-|\psi'|)(|\psi|-|\psi'|+1)}{2}$.

4.4. Simplification des expressions

La transformation des expressions initiales en expressions maximales conduit à définir les opérateurs associatifs avec une arité variable. Après avoir transformé les expressions sous une forme maximale, nous présentons ici l'étape suivante qui consiste à simplifier les expressions au vu des constantes apparaissant en argument. Nous prenons en compte, lors de cette simplification, les propriétés liées aux différents opérateurs, à savoir la commutativité, l'associativité, l'idempotence et les éventuels éléments neutres ou absorbants.

Le tableau 3.1 rappelle les propriétés à prendre en compte pour chacun des opérateurs que nous avons défini.

opérateur	associatif	Commutatif	élément neutre	élément absorbant
and	x	x	t	f
or	x	x	f	t
$\oplus$	x	x	f	
$\otimes$	x	x	t	
+	x	x	0	
anti+			0 à droite	
*	x	x	1	0
anti*			1 à droite	0 à gauche
cond				
seln				
garde			t à droite	f à gauche
l_garde	oui	oui		

Table 3.1.

La simplification des expressions est décrite par un système de règles d'inférence. Pour décrire cette réduction nous allons tout d'abord introduire les notations utilisées :

Notations. Pour un opérateur ω, nous définissons $\omega_{\mathcal{D}}$ comme l'opérateur correspondant sur le domaine de valeurs sémantique : par exemple $+_N$ dénote l'addition dans N si + est l'addition sur les valeurs de type *integer*. A une constante c de l'ensemble des expressions, nous associons **<c>** la valeur de cette constante dans $\mathcal{D}$, le domaine sémantique du type de la constante. Inversement, à une valeur **<c>** du domaine sémantique $\mathcal{D}$ nous associons $\lceil < c > \rceil$, sa représentation dans une expression. Dans ce cas, le type de la valeur retournée est cohérent avec le type de l'expression dans laquelle la valeur est utilisée.

Définition des règles d'inférence

$$\overline{(\omega\,(c_i)_{i\in I}) \rightarrow \lceil (\omega_{\mathcal{D}}\,(< c_i >)_{i\in I})\rceil} \qquad (1)$$

$$\frac{\varphi_i = 0}{(+\,\varphi_1 \ldots \varphi_n) \rightarrow (+\,\varphi_1 \ldots \varphi_{i-1}\,\varphi_{i+1} \ldots \varphi_n)} \qquad (2)$$

$$\overline{(+\,\varphi) \rightarrow \varphi} \qquad (3)$$

$$\frac{\varphi_i = c_i,\ \varphi_j = c_j,\ i < j}{(+\,\varphi_1 \ldots \varphi_n) \rightarrow (+\,\lceil (+_{\mathcal{D}} < c_i >< c_j >)\rceil \varphi_1 \ldots \varphi_{i-1}\,\varphi_{i+1} \ldots \varphi_{j-1}\,\varphi_{j+1} \ldots \varphi_n)} \qquad (4)$$

$$\frac{\varphi_i = 1}{(*\,\varphi_1 \ldots \varphi_n) \rightarrow (*\,\varphi_1 \ldots \varphi_{i-1}\,\varphi_{i+1} \ldots \varphi_n)} \qquad (5)$$

$$\frac{\varphi_i = 0}{(*\,\varphi_1 \ldots \varphi_n) \rightarrow 0} \qquad (6)$$

$$\overline{(-(-\,\varphi)) \rightarrow \varphi} \qquad (7)$$

$$\overline{(anti + 0\,\varphi) \rightarrow (-\,\varphi)} \qquad (8)$$

$$\overline{(anti + \varphi\,0) \rightarrow \varphi} \qquad (9)$$

$$\overline{(anti * 1\,\varphi) \rightarrow (\sqrt{}\,\varphi)} \qquad (10)$$

$$\frac{}{(anti*\ \varphi\ 1) \rightarrow \varphi} \quad (11)$$

$$\frac{}{(anti*\ 0\ \varphi) \rightarrow 0} \quad (12)$$

$$\frac{\varphi_i = t}{(\oplus\ \varphi_1 \ldots \varphi_n) \rightarrow (\oplus\ (\neg\varphi_1) \ldots \varphi_{i-1}\ \varphi_{i+1} \ldots \varphi_n)} \quad (13)$$

$$\frac{\varphi_i = f}{(\otimes\ \varphi_1 \ldots \varphi_n) \rightarrow (\otimes\ (\neg\varphi_1) \ldots \varphi_{i-1}\ \varphi_{i+1} \ldots \varphi_n)} \quad (14)$$

$$\frac{}{(cond\ t\ \varphi_2\ \varphi_3) \rightarrow \varphi_2} \quad (15)$$

$$\frac{}{(cond\ f\ \varphi_2\ \varphi_3) \rightarrow \varphi_3} \quad (16)$$

$$\frac{}{(cond\ \varphi\ c\ c) \rightarrow c} \quad (17)$$

$$\frac{< c > \in_D \{< c_1 >, \ldots, < c_n >\}}{(\in c\ \varphi_2\ c_1 \ldots c_n) \rightarrow \varphi_2} \quad (18)$$

$$\frac{< c > \notin_D \{< c_1 >, \ldots, < c_n >\}}{(\in c\ \varphi_2\ c_1 \ldots c_n) \rightarrow \bot} \quad (19)$$

$$\frac{< c > \notin_D \{< c_1 >, \ldots, < c_n >\}}{(\notin c\ \varphi_2\ c_1 \ldots c_n) \rightarrow \varphi_2} \quad (20)$$

$$\frac{< c > \in_D \{< c_1 >, \ldots, < c_n >\}}{(\notin c\ \varphi_2\ c_1 \ldots c_n) \rightarrow \bot} \quad (21)$$

$$\frac{}{(garde\ t\ \varphi) \rightarrow \varphi} \quad (22)$$

$$\frac{}{(garde\ f\ \varphi) \rightarrow u} \quad (23$$

Théorème. Ce système d'inférence, appliqué à une expression de $\mathcal{E}$, termine et renvoie une expression de $\mathcal{E}$.

Démonstration.

Nous définissons et notons $|\varphi|_\omega$ le cumul sur l'ensemble des sous-expressions de φ du nombre d'arguments de l'opérateur ω.

Soit $\prec$ la relation d'ordre définie sur les formules comme suit :

$$\varphi_1 \prec \varphi_2 \Leftrightarrow |\varphi_1| + |\varphi_1|_{\oplus} + |\varphi_1|_{\otimes} <_{\mathcal{N}} |\varphi_2| + |\varphi_2|_{\oplus} + |\varphi_2|_{\otimes}$$

Cette relation d'ordre définit un ordre $(\mathcal{E}, \prec)$ bien fondé (il n'existe pas de suite infinie strictement décroissante), car $(N, <)$ est un ordre bien fondé.

Il est évident que toutes les règles d'inférence définies sont strictement décroissantes sur $(\mathcal{E}, \prec)$.

4.5. Règles d'équivalence expressions/graphe

Lors de la construction du graphe de transfert d'information, nous devons définir les différentes composantes du quintuplet. Pour pouvoir représenter le graphe, il nous faut bien évidemment définir des identificateurs pour les places et les transitions créées, par contre, pour les arcs nous définissons des couples non identifiés explicitement. Les identificateurs de places et de transitions doivent être uniques, nous définissons pour cela une fonction qui pour une expression renvoie un nom unique pour une place. Pour certaines places, non directement associées à une expression, nous notons un nouvel identificateur par *Np*. De même, les identificateurs de transitions crées sont notés *Nt*.

Définition 3.12. A chaque expression ou sous-expression φ d'une expression ψ, la fonction *Nom* associe un nom unique. Cette fonction est définie comme suit :

- $\forall \varphi \in SE^+(\psi) \cap SE^-(\psi), Nom(\varphi)$ est un identificateur unique généré au premier appel de la fonction avec l'expression pour argument ;
- $\forall \varphi \in Ter(\psi), Nom(\varphi) = \varphi$. Pour un identificateur de signal externe de sens *inout*, le nom de la place dépend du contexte d'apparition de l'identificateur, destination ou source d'une expression.

Définition 3.13. Soit $\perp$ l'expression vide et $Expr_0$ la fonction qui a tout identificateur associe l'expression vide : $\forall x \in ID, Expr_0(x) = \perp$.

Soit *Expr* une fonction de *ID* dans $\mathcal{E}$, on note $Expr[x / \varphi]$ la fonction définie par :

$$(\forall y \in ID, y \neq x \Rightarrow Expr[x / \varphi](y) = Expr(y)) \wedge (Expr[x / \varphi](x) = \varphi)$$

Cette opération est étendue aux fonctions de la manière suivante :

$$\forall x \in ID, (Expr_2(x) = \perp \Rightarrow Expr_1[Expr_2](x) = Expr_1(x)) \wedge (Expr_2(x) \neq \perp \Rightarrow Expr_1[Expr_2](x) = Expr_2(x))$$

Théorème. Soient $Expr_1$ et $Expr_2$ deux fonctions de ID dans $\mathcal{E}$:

$$(\forall x \in ID, (Expr_1(x) = \perp) \vee (Expr_2(x) = \perp)) \Rightarrow (Expr_1[Expr_2] = Expr_2[Expr_1])$$

Démonstration. Si les supports de chacune deux fonctions $Expr_1$ et $Expr_2$ sont disjoints, alors aucune des deux fonctions ne modifie l'autre.

Ce théorème montre que l'opération *[]* est commutative pour des fonctions qui ont des domaines de définition disjoints. Cette propriété est utilisées dans la traduction ci-dessous par l'utilisation de l'opérateur généralisé sachant que l'ordre de construction de la fonction est indifférent.

Définition 3.14. Pour ce type de fonction nous définissons l'opérateur généralisé $\underset{i=1}{\overset{i=n}{[]}} Expr_i = Expr_1[Expr_2]...[Expr_i]...[Expr_n]$.

Soit ψ une expression décrivant une architecture d'une entité. La traduction de l'expression en un graphe de transfert d'information statique est définie par la fonction *Trad* construite inductivement comme suit :

Traduction de identificateurs de signaux

- $(\varphi = s)$

$$(s \in Sent) \Rightarrow Trad(\varphi) = (\{s\}, \varnothing, \varnothing, \varnothing, Expr_0[s/s])$$

Traduction d'une valeur constante

Le graphe représentant les transferts d'information, nous ne générons pas de places dont l'expression associée est constante. En effet, la quantité d'information associée à une constante étant nulle, nous ne désirons pas lui associer une place. Ceci consisterait à lui accorder la notion de flot d'information.

- $(\varphi = const) \Rightarrow Trad(\varphi) = (\varnothing, \varnothing, \varnothing, \varnothing, Expr_0)$

Traduction de l'opérateur cond

Dans le cas d'expressions conditionnelles définissant deux ou plusieurs sources pour une destination, nous décidons d'introduire un graphe qui représente autant de chemins d'information qu'il y a de sources (non constantes) possibles. Ceci conduit a la définition d'une expression avec le l'opérateur *"sélection"* qui transmet en sortie l'unique source active en entrée. Chacune des sources envoie soit une information (lorsqu'elle est active) soit aucune information, et ceci avec une exclusion mutuelle. Dans le cas d'expressions conditionnelles dont l'une des expressions de donnée au moins est une constante, le graphe comporte un seul chemin d'information.

- $(\varphi = (cond\ \varphi_1\ \varphi_2\ \varphi_3)) \wedge (\varphi_2 \neq const) \wedge (\varphi_3 \neq const) \Rightarrow$

$$
\begin{aligned}
Trad(\varphi) = (&P_1 \cup P_2 \cup P_3 \cup \{Nom(\varphi), Np_1, Np_2\},\\
&T_1 \cup T_2 \cup T_3 \cup \{Nt, Nt_1, Nt_2, Nt_3, Nt_4\},\\
&\alpha_1 \cup \alpha_2 \cup \alpha_3 \cup \{(Nom(\varphi_1), Nt), (Nom(\varphi_2), Nt), (Nom(\varphi_2), Nt_1), (Nom(\varphi_3), Nt_2)\},\\
&\beta_1 \cup \beta_2 \cup \beta_3 \cup \{(Nt_1, Np_1), (Nt_2, Np_2), (Nt_3, Nom(\varphi)), (Nt_4, Nom(\varphi))\}\\
&Expr_1[Expr_2][Expr_3][Np_1 / (si\ Nom(\varphi_1)\ alors\ Nom(\varphi_2) sinon\ \varepsilon)]\\
&\qquad [Np_2 / (si\ Nom(\varphi_1)\ alors\ \varepsilon\ sinon\ Nom(\varphi_3))]\\
&\qquad [Nom(\varphi) / (selection\ Np_1\ Np_2)])
\end{aligned}
$$

où $Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i), i = 1, 2, 3$

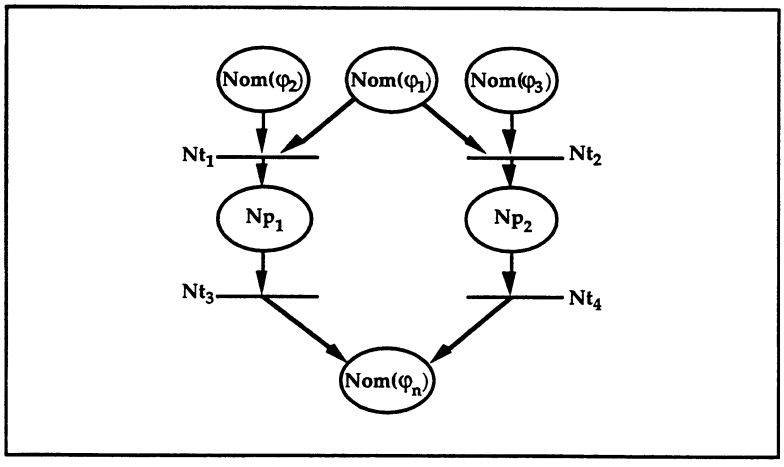

Figure 3.5. GTIS d'une expression conditionnelle sans constante.

- $(\varphi = (cond\ \varphi_1\ c_2\ c_3)) \Rightarrow$

$Trad(\varphi) = (P_1 \cup \{Nom(\varphi)\},$
$T_1 \cup \{Nt\},$
$\alpha_1 \cup \{(Nom(\varphi_1), Nt)\},$
$\beta_1 \cup \{(Nt, Nom(\varphi))\},$
$Expr_1\ [Nom(\varphi) / (si\ Nom(\varphi_1)\ alors \cup c_2\ sinon\ c_3)])$

où $Trad(\varphi_1) = (P_1, T_1, \alpha_1, \beta_1, Expr_1)$

- $(\varphi = (cond\ \varphi_1\ \varphi_2\ c_3)) \Rightarrow$

$Trad(\varphi) = (P_1 \cup P_2 \cup \{Nom(\varphi)\},$
$T_1 \cup T_2 \cup \{Nt\},$
$\alpha_1 \cup \alpha_2 \cup \{(Nom(\varphi_1), Nt), (Nom(\varphi_2), Nt)\},$
$\beta_1 \cup \beta_2 \cup \{(Nt, Nom(\varphi))\},$
$Expr_1[Expr_2]\ [Nom(\varphi) / (si\ Nom(\varphi_1)\ alors \cup \varphi_2\ sinon\ c_3)])$

où $Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i), i = 1,2$

- $(\varphi = (cond\ \varphi_1\ c_2\ \varphi_3)) \Rightarrow$

$Trad(\varphi) = (P_1 \cup P_3 \cup \{Nom(\varphi)\},$
$T_1 \cup T_3 \cup \{Nt\},$
$\alpha_1 \cup \alpha_3 \cup \{(Nom(\varphi_1), Nt), (Nom(\varphi_2), Nt)\},$
$\beta_1 \cup \beta_3 \cup \{(Nt, Nom(\varphi))\},$
$Expr_1[Expr_3]\ [Nom(\varphi) / (si\ Nom(\varphi_1)\ alors \cup c_2\ sinon\ \varphi_3)])$

où $Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i), i = 1,2$

Traduction de l'opérateur seln et des clauses $\in$ et $\notin$

- $(\varphi = (seln\ \varphi_1 \ldots \varphi_i \ldots \varphi_n)) \Rightarrow$

$$Trad(\varphi) = (\{Nom(\varphi)\} \cup (\bigcup_{i=1}^{i=n} P_i), \{Nt_i\}_{1 \le i \le n} \cup (\bigcup_{i=1}^{i=n} T_i), \{(Nom(\varphi_i), Nt_i)\}_{1 \le i \le n} \cup (\bigcup_{i=1}^{i=n} \alpha_i),$$
$$\{(Nt_i, Nom(\varphi))\}_{1 \le i \le n} \cup (\bigcup_{i=1}^{i=n} \beta_i), (\bigcup_{i=1}^{i=n} Expr_i)[Nom(\varphi) / (selection\ \varphi_1 \ldots \varphi_i \ldots \varphi_n)])$$

où $\forall i \in \{1, \ldots, n\}, Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i)$

- $(\varphi = (\in \varphi_1\ \varphi_2\ c_1 \ldots c_i \ldots c_n)), \forall i, c_i$ *est une constante* $\Rightarrow$

$Trad(\varphi) = (P_1 \cup P_2 \cup \{Nom(\varphi)\}, T_1 \cup T_2 \cup \{Nt\},$
$\alpha_1 \cup \alpha_2 \cup \{(Nom(\varphi_1), Nt), (Nom(\varphi_2), Nt)\}, \beta_1 \cup \beta_2 \cup \{(Nt, Nom(\varphi))\},$
$Expr_1[Expr_2][Nom(\varphi) / (si\ Nom(\varphi_1) \in \{c_1 \ldots c_i \ldots c_n\} alors\ Nom(\varphi_2) sinon\ \varepsilon)])$

où $Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i), i = 1,2$

- $(\varphi = (\notin \varphi_1\ \varphi_2\ c_1 \ldots c_i \ldots c_n))$, $\forall i, c_i$ *est une constante* $\Rightarrow$

$Trad(\varphi) = (P_1 \cup P_2 \cup \{Nom(\varphi)\}, T_1 \cup T_2 \cup \{Nt\},$
$\alpha_1 \cup \alpha_2 \cup \{(Nom(\varphi_1), Nt), (Nom(\varphi_2), Nt)\}, \beta_1 \cup \beta_2 \cup \{(Nt, Nom(\varphi))\},$
$Expr_1[Expr_2][Nom(\varphi) / (si\ Nom(\varphi_1) \in \{c_1 \ldots c_i \ldots c_n\} alors\ \varepsilon\ sinon\ Nom(\varphi_2))])$

où $Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i), i = 1,2$

Nous montrons la traduction d'une expression VHDL du type sélecteur de clauses par l'exemple ci-dessous. Le graphe correspondant est donné dans la figure 3.6.

```
with x select
        S <=  A when 0, 1, 2, 3 else
              B when 4, 5, 6, 7 else
              C when others;
```

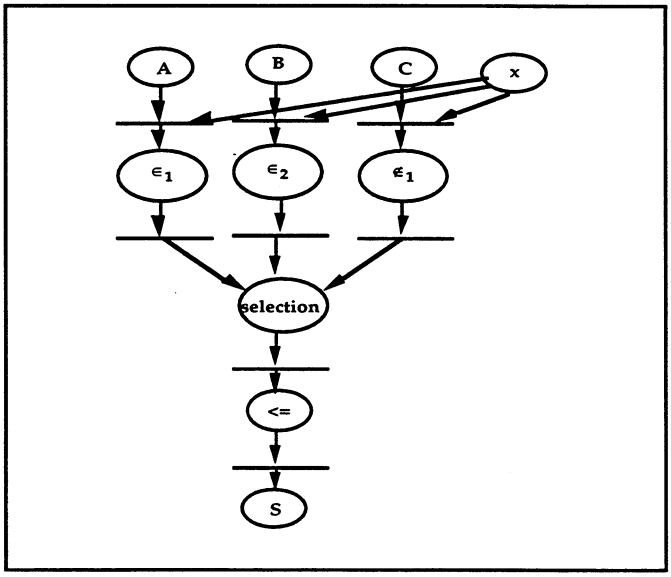

Figure 3.6. GTIS pour une expression sélection.

Traduction des autres opérateurs

- $(\varphi = (\omega\ \varphi_1 \ldots \varphi_i \ldots \varphi_n)) \Rightarrow$

$$Trad(\varphi) = (\{Nom(\varphi)\} \cup (\bigcup_{i=1}^{i=n} P_i), \{Nt\} \cup (\bigcup_{i=1}^{i=n} T_i), \{(Nom(\varphi_i), Nt)\}_{1 \le i \le n} \cup (\bigcup_{i=1}^{i=n} \alpha_i),$$
$$\{(Nt, Nom(\varphi))\} \cup (\bigcup_{i=1}^{i=n} \beta_i), (\prod_{i=1}^{i=n} Expr_i)[Nom(\varphi) / (\omega\ \varphi_1 \ldots \varphi_i \ldots \varphi_n)])$$

où $\forall i \in \{1, \ldots, n\}, Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i)$

Cette règle est montrée dans la figure 3.7.

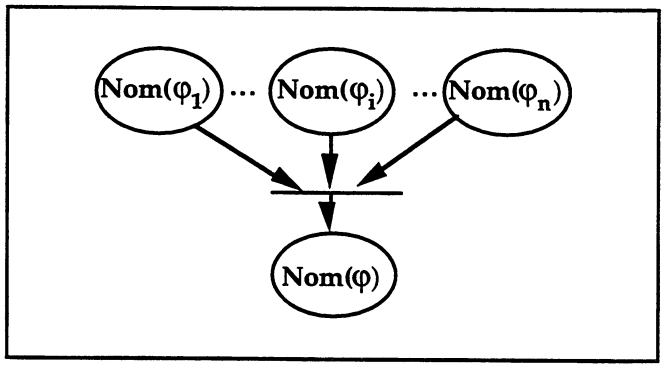

Figure 3.7.Traduction d'un opérateur.

Traduction des affectations de la description

$$(\varphi = (\triangleright\ \varphi_1 \ldots \varphi_i \ldots \varphi_n)) \Rightarrow$$
$$Trad(\varphi) = ((\bigcup_{i=1}^{i=n} P_i),(\bigcup_{i=1}^{i=n} T_i),(\bigcup_{i=1}^{i=n} \alpha_i),(\bigcup_{i=1}^{i=n} \beta_i),([]_{i=1}^{i=n} Expr_i))$$
$$où \quad \forall i \in \{1,\ldots,n\}, Trad(\varphi_i) = (P_i, T_i, \alpha_i, \beta_i, Expr_i)$$

Dans la définition de la fonction *Trad* ci-dessus, l'opération $\cup$ dénote l'union sur les ensembles. Nous allons maintenant exprimer la taille du graphe résultant d'une expression donnée en termes de places.

Définition 3.15. Soit ψ une expression, nous définissons $Ncond(\psi)$ la fonction qui donne le nombre de sous-expressions φ de ψ de la forme $\varphi = (cond\ \varphi_1\ \varphi_2\ \varphi_3)$ *telle que* $\varphi_2 \neq const$ *et* $\varphi_3 \neq const$.

Théorème. Soit ψ une expression, la taille de Trad(ψ) en nombre de places est :

$$Np(\psi) = 2 \cdot Ncond(\psi) + \left|SE^{+}(\psi)\right| + \left|Sig(\psi)\right| - 1$$

Démonstration. Dans les règles de traduction ci-dessus, toutes les expressions de profondeur supérieure à 1 se traduisent par la définition d'une seule place, sauf pour le cas de l'opérateur conditionnel n'ayant aucun argument constant. En effet, dans ce dernier cas, il y a génération de trois places. De plus, un identificateur de signal conduit

à la définition d'une seule place, quel que soit le nombre de ses occurrences dans la description.

Ce théorème de complexité du graphe résultant d'une expression donnée permet d'envisager l'optimisation du graphe par la réduction la taille de l'expression. Cette réduction est caractérisées par la représentation unique des expressions ayant plusieurs occurrences.

5. Recherche des expressions à occurrences multiples

Comme l'indique le théorème précédent, la taille du graphe en nombre de places est proportionnelle à la taille de la famille des sous-expressions strictes de profondeur supérieure à 1. Il en résulte que si l'on réduit la taille de cette famille, on réduit également la taille du graphe résultant

Pour réduire la taille de la famille, il serait possible de rechercher les expressions communes de manière à ce que chaque expression n'ait plus qu'une occurrence. Mais, rechercher les sous-expressions communes de manière à minimiser globalement la taille de la famille est prohibitif en termes de complexité algorithmique. En effet, pour cela il faudrait prendre en compte les propriétés liées à des théories sur plusieurs opérateurs comme la distributivité et les inverses (soustraction et opposé vis à vis de l'addition et division et inverse vis à vis de la multiplication).

Pour cela, nous nous limiterons à la reconnaissance d'expressions communes, si et seulement si, les occurrences ne diffèrent que par l'associativité, la commutativité et/ou l'idempotence pour les opérateurs logiques.

5.1. Principe d'identification

L'expression VHDL résultant des étapes précédentes peut encore être réduite. Au vu de la définition du graphe de testabilité, la présence de plusieurs occurrences d'une même sous-expression conduit à la construction de sous-graphes identiques, à la différence près du nom des modules et transitions.

Définition 3.16. Deux graphes de transfert d'information $G_1 = (P_1, T_1, \alpha_1, \beta_1, Expr_1)$ et $G_2 = (P_2, T_2, \alpha_2, \beta_2, Expr_2)$ sont identiques, si et seulement si :

$$(S(G_1) = S(G_2)) \wedge (R(G_1) = R(G_2))$$

$\exists$ *une bijection* $f : P_1 \cup T_1 \rightarrow P_2 \cup T_2$

$$p \in P_1 \mapsto f(p) \in P_2$$

$$t \in T_1 \mapsto f(t) \in T_2$$

telle que:

$$\forall t \in T_1, f(\Gamma^-_{t,G_1}(t)) = \Gamma^-_{t,G_2}(f(t))$$

$$\forall t \in T_1, f(\Gamma^+_{t,G_1}(t)) = \Gamma^+_{t,G_2}(f(t))$$

$$\forall p \in P_1, f(\Gamma^-_{p,G_1}(p)) = \Gamma^-_{p,G_2}(f(p))$$

$$\forall p \in P_1, f(\Gamma^+_{p,G_1}(p)) = \Gamma^+_{p,G_2}(f(p))$$

$$\forall p \in P_1, Exp_2(f(p)) = Exp_1(p) \textit{ dans laquelle :}$$

$$\forall p_i \in P_1, p_i \textit{ a été substitué par } f(p_i).$$

Deux expressions identiques conduisent donc à deux sous-graphes identiques dans le cas où elles ne diffèrent que modulo l'associativité, la commutativité et l'idempotence.

Pour réduire la taille d'une description ψ par rapport à une sous-expression φ ayant plusieurs occurrences, nous utilisons une technique de renommage qui associe à un nouvel identificateur de signal *Nidf* l'expression φ et substitue les occurrences de φ par des occurrences de l'identificateur *Nidf*. Le nouvel identificateur introduit n'est qu'un identificateur de travail, ce qui signifie qu'il n'est utilisé que dans l'expression et qu'il n'aura pas à apparaître dans le graphe résultant. Ainsi, lors de la construction du graphe, les références à l'identificateur *Nidf* seront substituées par des références au nom de la place associée à l'expression φ.

Définition 3.17. Soit φ une sous-expression stricte de ψ ; on note $Rnm(\varphi, \psi)$, l'expression obtenue à partir de ψ dans laquelle :

- l'expression φ a été renommée,
- ses occurrences substituées par des occurrences d'un identificateur de travail,
- l'identificateur est défini par l'expression φ.

Théorème. Soit φ une sous-expression stricte d'une expression ψ, soient T(φ) (resp. T(ψ)) la taille du graphe associé à l'expression φ (resp. ψ) en nombre de places. Si l'expression φ a n occurrences dans ψ, alors la taille du graphe correspondant à $Rnm(\varphi,\psi)$ est $T(Rnm(\varphi,\psi)) = T(\psi) - (n-1)T(\varphi)$.

5.2. Stratégie de reconnaissance des expressions à occurrences multiples

Nous avons vu que la reconnaissance des différentes occurrences d'une expression n'est pas envisageable de façon réaliste si les occurrences diffèrent relativement aux propriétés de distributivité. Il serait éventuellement envisageable de définir un système de réécriture qui mène à une forme normale unique, mais les résultats correspondants ne garantiraient nullement l'optimalité du graphe en termes de places. Nous considérons donc les expressions telles qu'elles ont été obtenues par les étapes de traduction précédentes. Par contre, nous allons tirer partie des propriétés propres à chaque opérateur : l'associativité et la commutativité. L'associativité étant déjà traitée par l'application de la transformation dans la syntaxe <VHDL 2>, seule la commutativité est donc à prendre en compte.

L'associativité pourrait être utilisée de façon plus fine dans cette étape de compaction. En effet, il est possible de rechercher des sous-expressions communes à des expressions d'opérateur associatif. Mais là encore le coût de traitement serait trop important pour déterminer un résultat optimal.

Compte tenu de ces restrictions, les différentes occurrences d'une expression sont telles qu'il existe un parcours syntaxique de chacune d'elles qui permette d'afficher la même séquence de sortie. Les parcours ne différeraient que par le choix de l'ordre sur les sous-expressions directes d'une expression dont l'opérateur est commutatif.

Définition 3.18. Condition d'identification des occurrences d'une expression. Soit $\varphi \prec \psi$ une expression ayant n occurrences identifiables $\varphi_1, \varphi_1, \ldots, \varphi_n$, alors :

$\forall \xi \prec \varphi, \xi$ *a au moins* n *occurrences identifiables* $\xi_1, \xi_2, \ldots, \xi_n$ *telles que :*
$\forall i = 1,2,\ldots,n,\ \xi_i \prec \varphi_i$

Cette définition de caractérisation met en évidence le problème de la reconnaissance des occurrences d'une expression par le biais d'une méthode descendante. En effet, la prise en compte de la commutativité induirait un coût important pour rechercher les occurrences des sous-expressions ξ_i dans les occurrences de φ. Par contre une stratégie ascendante qui consiste, de façon constructive, à appliquer le renommage sur des expressions de profondeur égales à 1 (expressions dites terminales), permet de minimiser l'espace de recherche à chaque pas. Une telle stratégie est basée sur l'écriture contraposée de la définition ci-dessus.

Corollaire. Soit $\varphi \prec \psi$ une sous-expression.

$$(\exists\, \xi \prec \varphi \text{ et } \xi \text{ n'a qu'une occurrence identifiable dans } \psi) \Rightarrow$$
$$(\varphi \text{ n'a qu'une seule occurrence identifiable dans } \psi)$$

Dans la stratégie ascendante, la recherche des occurrences d'une expression n'est envisagée que si toutes les sous-expressions ont plusieurs occurrences.

De plus, la reconnaissance des différentes occurrences d'une expression terminale est simple. Donc, si le renommage est effectué directement après cette reconnaissance, les expressions initialement non terminales, qui ont plusieurs occurrences identifiables, se ramènent à des expressions terminales par renommages successifs de ses sous-expressions. Ce principe permet donc de construire un algorithme relativement simple qui ne prend en compte que l'identification d'expressions terminales. Initialement, les occurrences des identificateurs de signaux ou de constantes sont des occurrences multiples identifiées d'expressions à occurrences multiples.

Définition 3.19. Soit ω un opérateur du langage ; nous définissons $=_\omega$ l'égalité de deux listes d'arguments relativement à la théorie de l'opérateur ω. Ceci signifie que si ω est Associatif/Commutatif/Idempotent, alors $=_\omega$ signifie l'égalité de deux ensembles, si l'opérateur est Associatif/Commutatif, $=_\omega$ signifie l'égalité de deux familles sinon $=_\omega$ correspond à l'égalité de deux listes.

Définition 3.20. *Identification d'expressions terminales.* Soient φ_1 *et* φ_2 deux expressions terminales ; φ_1 est identique à φ_2 si et seulement si elles ont même opérateur et des

listes d'arguments identiques vis à vis de la théorie associée à leur opérateur. Nous noterons $\varphi_1 \equiv \varphi_2$. Cette définition s'exprime comme suit :

$$(\varphi_1 \equiv \varphi_2) \Leftrightarrow (Op(\varphi_1) = Op(\varphi_2) = \omega) \text{ et } (Arg(\varphi_1) =_\omega Arg(\varphi_2))$$

Définitions 3.21. *Famille des sous-expressions terminales.* Soit ψ une expression de profondeur supérieure ou égale à 1. Nous définissons la famille des sous-expressions terminales de ψ par $SE^1(\psi) = \{\varphi \in SE(\psi) / Prof(\varphi) = 1\}$. A tout opérateur ω nous associons la famille des expressions terminales de ψ qui ont pour opérateur ω : $SE^1_\omega(\psi) = \{\varphi \in SE^1(\psi) / Op(\varphi) = \omega\}$. Nous définissons ainsi la fonction *Terminale* définie sur $\mathcal{E}$ qui vaut vrai si une expression est terminale et faux sinon.

L'identification d'expressions terminales a lieu en tenant compte de l'opérateur et de la liste d'arguments. La définition des ensembles $SE^1_\omega(\psi)$ correspond à partitionner l'ensemble des expressions terminales en classes d'expressions ayant même opérateur. Pour un opérateur donné, nous décrivons la construction de classes d'équivalence d'expressions dont les listes d'arguments sont égales au vu de la théorie associée à l'opérateur. Il est évident que par construction, $=_\omega$ est une relation d'équivalence car elle est basée sur l'égalité. L'égalité de familles est une égalité de liste obtenue par tri lexical des atomes des familles.

Pour définir l'algorithme, nous définissons :

- l'occurrence d'une expression dans la description. Cette définition permet de retrouver la racine d'une expression qui doit être renommée par un identificateur ;
- les structures de données utilisées tout au long de l'algorithme ;
- les opérateurs portant sur ces structures de données.

Nous présentons ensuite l'algorithme.

5.2.1. Occurrence d'une expression

Nous utilisons la notion d'occurrence classiquement utilisées pour identifier les différents noeuds des arbres [LAL 90]. La représentation consiste à prendre en compte

la liste des numéros des arcs, les arcs sortants d'un noeud étant numérotés à partir de 1 de gauche à droite. Nous précisons ici les structures de données que nous utilisons.

Définition 3.23. *Chemin d'occurrence* . Soit ψ une expression et φ une sous-expression de ψ, $\varphi \prec \psi$. On appelle *chemin d'occurrence* la liste des occurrences des expressions se touvant sur le chemin de la racine ψ à l'expressionφ. Cette liste est notée $CO(\psi, \varphi)$.

Définition 3.24. *Rang*. Soit ψ une expression telle que $Prof(\psi) \geq 1$ et φ une sous-expression directe, c'est-à-dire telle que $d(\psi, \varphi) = 1)$, on appelle rang de la sous-expression l'indice d'apparition de φ dans $Arg(\psi)$, le premier indice étant égal à 1. Cette valeur est notée$Rang(\psi)$.

Nous pouvons maintenant définir l'occurrence d'une expression au sein de la description de l'architecture.

Définition 3.25. *Occurrence*. Soit ψ l'expression associée à la description d'architecture d'une entité VHDL. L'occurrence d'une sous-expression φ est définie comme la liste des rangs des expressions du chemin d'occurrence. L'ordre de définition des rangs étant celui du chemin d'occurrence. Nous notons $Occ(\varphi)$ cette occurrence. Plus formellement : $CO(\psi, \varphi) = (\xi_i)_{1 \leq i \leq n} \Rightarrow Occ(\varphi) = (Rang(\xi_i)_{1 \leq i \leq n}, Rang(\varphi))$

5.2.2. Structures de données

Pour tout ensemble E on note l'ensemble des séquences non vides possibles d'éléments de cet ensemble par $E^+ = \bigcup_{i=1}^{\infty} E^i$

Soit Θ l'ensemble des occurrences possibles, $\Theta = \left(N^*\right)^+$ on note θ une occurrence et $\perp_\Theta$ l'occurrence vide et O un ensemble d'occurrences.

Soit A l'ensemble des listes d'arguments possibles, $A = \mathcal{E}^+$, on note π une liste d'arguments.

Soit C l'ensemble des classes d'équivalence possibles, $C = A \times \mathcal{P}(\Theta)$, on note χ une classe d'équivalence.

Soit $H = \mathcal{P}(C)$ l'ensemble des ensembles de classes d'équivalence possibles, on note η un élément de cet ensemble.

L'ensemble des $|\Omega|$-uplets possibles qui associent à tout opérateur ω un ensemble de classes d'équivalence est dénoté par $\mathcal{L}$, $\mathcal{L} = H^{|\Omega|}$ et on note λ un élément de cet ensemble. Un $|\Omega|$-uplets sera interprété comme une application, $\lambda : \Omega \to H$. On note $\lambda_{\varnothing}, \forall \omega \in \Omega, \lambda_{\varnothing}(\omega) = \varnothing$.

Soit $\lambda \in \mathcal{L}$, on note $\lambda(\omega) = \eta_{\omega,\lambda} \in H$, l'ensemble des classes d'équivalence de l'opérateur ω présent dans λ. Une classe d'équivalence de l'opérateur ω est dénotée par χ^{ω}.

5.2.3. Opérateurs sur les structures de données

Soit $(\chi_1^{\omega}, \chi_2^{\omega}) \in C^2$, nous définissons l'*union des deux classes* d'équivalence par l'opérateur $\cup_C$ de la manière suivante :

$$(\chi_1^{\omega} = (\pi_1, O_1)) \wedge (\chi_2^{\omega} = (\pi_2, O_2)) \Rightarrow si\ (\pi_1 =_{\omega} \pi_2 = \pi) alors\ \chi_1^{\omega} \cup_C \chi_2^{\omega} = (\pi, O_1 \cup O_2))$$
$$sinon\ \chi_1^{\omega} \cup_C \chi_2^{\omega}\ n'est\ pas\ défini$$

Définition. *Réunion sur l'ensemble H.* Soient η_1 et η_2 deux ensembles de classes, leur réunion est dénotée par l'opérateur $\cup_H$ et définie par :

$$\begin{aligned}
&(\chi = (\pi, O) \in \eta_1 \cup_H \eta_2) \Leftrightarrow \\
&\quad ((\exists (\chi_1, \chi_2) \in \eta_1 \times \eta_2, (\chi_1 = (\pi_1, O_1)), (\chi_2 = (\pi_2, O_2)), \\
&\qquad (\pi_1 =_{\omega} \pi) \wedge (\pi_1 =_{\omega} \pi)) \wedge (\chi = \chi_1 \cup_C \chi_2)) \vee \\
&\quad ((\exists (i, j) \in \{1,2\}^2, i \neq j, \\
&\quad (\exists \chi_i \in \eta_i, (\chi_i = (\pi_i, O_i)), (\pi_i =_{\omega} \pi)) \wedge \\
&\qquad (\forall \chi_k \in \eta_j, (\chi_k = (\pi_k, O_k)), (\pi_k \neq_{\omega} \pi)))) \wedge (\chi = \chi_i))
\end{aligned}$$

Notation. On note $\lambda[\omega_i / \eta_{\omega_i}]$ l'application obtenue à partir de λ et telle que :

$$\forall \omega \in \Omega, ((\omega \neq \omega_i) \Rightarrow \lambda[\omega_i / \eta_{\omega_i}](\omega) = \lambda(\omega)) \wedge (\lambda[\omega_i / \eta_{\omega_i}](\omega_i) = \eta_{\omega_i})$$

Définition. *Concaténation de deux applications.* Soit $(\lambda_1, \lambda_2) \in \mathcal{L}^2$, la concaténation de ces deux applications est l'application notée $\lambda_1 \nabla \lambda_2$ et définie comme suit :

$$\forall \omega \in \Omega, (\lambda_1 \nabla \lambda_2)(\omega) = \lambda_1(\omega) \cup_H \lambda_2(\omega)$$

La loi ∇ est associative et commutative. L'opération de concaténation généralisée $\overset{i=n}{\underset{i=1}{\nabla}} \lambda_i$ correspond à la concaténation des n applications $\lambda_i, 1 \leq i \leq n$. Dans ce cas : $\overset{i=1}{\underset{i=1}{\nabla}} \lambda_i = \lambda_1$.

5.2.4. Algorithme d'identification

L'identification se compose de deux étapes. Une prtemière étape d'initialisation permet de déterminer par un parcours d'une expression la fonction qui associe à chaque opérateur son ensemble de classes d'expressions identiques. La seconde étape consiste à itérer un processus de renommage et de détermination d'une nouvelle fonction λ, ceci jusqu'à ce que la fonction résultante soit égale à $\lambda_\varnothing$ ce qui correspond à la non reconnaissance d'expressions terminales identiques et donc terminaison du processus.

Une fonction de tri lexicographique de listes de symboles notée *Sort* est utilisée au sein de cet algorithme.

a) Initialisation

Initialisation. La fonction d'initialisation appelée *Init* est définie comme suit :

$$Init: \mathcal{E} \rightarrow \mathcal{L}$$
$$\psi \mapsto \lambda = Init1(\psi, \perp_\Theta)$$

La fonction *Init1* parcourt une expression et construit pour chacune des sous-expressions une occurrence, si cette sous-expression est terminale, alors la classe correspondante est créée et introduite dans le résultat. Nous définissons *Init1* comme suit :

$$Init1 : \mathcal{E} \times \Theta \rightarrow \mathcal{L}$$
$$(\varphi, \theta) \mapsto Init1(\varphi, \theta) = \lambda$$

Cette fonction est décrite sous forme du système d'inférence suivant :

$$\frac{Atom(\varphi)}{(\varphi, \theta) \rightarrow \lambda_\varnothing}$$

$$\frac{Terminale((\omega\ \varphi_1 \ldots \varphi_n))}{((\omega\ \varphi_1 \ldots \varphi_n), \theta) \rightarrow \lambda_\varnothing[\omega / \{(Sort(\omega, (\varphi_1 \ldots \varphi_n)), \{\theta\})\}]}$$

$$\frac{\neg Terminale((\omega\ \varphi_1 \ldots \varphi_n))}{((\omega\ \varphi_1 \ldots \varphi_n),\theta) \rightarrow \bigvee_{i=1}^{i=n} Init1(\varphi_i, \theta.i)}$$

b) Identification

Greffe. Nous définissons la substitution de la i^{ème} sous-expression directe, d'une expression ψ non atomique, dont l'occurrence est θ par une expression ξ de la manière suivante :

$$\sigma: \Theta \times N \times \mathcal{E} \rightarrow \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\theta, i, \xi) \mapsto \sigma(\theta, i, \xi)$$

où

$$\sigma(\theta, i, \xi): \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\psi, \lambda) \mapsto \sigma(i, \xi)(\psi, \lambda) = (\psi', \lambda')$$

(ψ', λ') est défini comme suit :

soit $\psi = (\omega\ \varphi_1 \ldots \varphi_i \ldots \varphi_n)$

alors $\begin{cases} \psi' = (\omega\ \varphi_1 \ldots \varphi_{i-1}\ \xi\ \varphi_{i+1} \ldots \varphi_n) \\ \text{si } Terminale(\psi') \text{ alors } \lambda' = \lambda \nabla \lambda_{\varnothing}\left[\omega / \{((\varphi_1 \ldots \varphi_{i-1}\ \xi\ \varphi_{i+1} \ldots \varphi_n), \{\theta\})\}\right] \\ \text{sinon } \lambda' = \lambda \end{cases}$

Geffe d'une occurrence. Nous définissons la fonction σ_{Occ} qui greffe une expression à la place de celle dont l'occurrence est donnée en paramètre. Si la greffe induit la création d'une nouvelle expression terminale, alors celle-ci est introduite dans l'environnement des expressions terminales résultantes.

Cette fonction est définie comme suit :

$$\sigma_{Occ}: \Theta \times \Theta \times \mathcal{E} \rightarrow \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\theta_1, \theta_2, \xi) \mapsto \sigma_{Occ}(\theta_1, \theta_2, \xi)$$

où

$$\sigma_{Occ}(\theta_1, \theta_2, \xi): \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\psi, \lambda) \mapsto \sigma_{Occ}(\theta_1, \theta_2, \xi)(\psi, \lambda) = (\psi', \lambda')$$

définis comme suit :

soit $\psi = (\omega\ \varphi_1 \ldots \varphi_j \ldots \varphi_n)$ *et* $\theta_2 = \theta'.i$

si $\theta' = \perp_{\Theta}$ *alors* $(\psi', \lambda') = \sigma(\theta_1, i, \xi)(\psi, \lambda)$

sinon $(*\theta' = j.\theta''*)$ *et alors* $(\psi', \lambda') = \sigma_{Occ}(\theta_1, \theta'', \xi)(\varphi_j, \lambda)$

Greffes multiples d'un ensemble d'occurrences. Nous définissons la fonction qui greffe une expressiona la place d' n ensemble d'expressions. Chaque expression est identifiée par son occurrence. Cette fonction est définie comme suit :

$$\sigma_{LOcc}: C \times \mathcal{E} \rightarrow \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\chi, \xi) \mapsto \sigma_{LOcc}(\chi, \xi)$$

où

$$\sigma_{LOcc}(\chi, \xi): \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\psi, \lambda) \mapsto (\psi', \lambda') \textit{ tel que}$$
$$\textit{soit } \chi = (\pi, \{\theta_i\}_{1 \leq i \leq n})$$
$$(\psi', \lambda') = (\underset{1 \leq i \leq n}{\circ} \sigma_{Occ}(\theta_i, \theta_i, \xi))(\psi, \lambda)$$

Ajout. Nous définissons la fonction qui à une expression donnée, ajoute en argument (dernier argument) une expression non atomique. Cette fonction, notée κ, est définie comme suit :

$$\kappa: \mathcal{E} \rightarrow \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$\xi \mapsto \kappa(\xi)$$

où

$$\kappa(\xi): \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$((\omega\ \varphi_1 \ldots \varphi_n), \lambda) \mapsto \kappa(\xi)(\psi, \lambda) = ((\omega\ \varphi_1 \ldots \varphi_n\ \xi), \lambda)$$

Traitement d'un ensemble de classes. Nous définissons la fonction qui associe à un opérateur les substitutions à effectuer au vu de l'ensemble des classes d'expressions identiques. Cette fonction est définie comme suit :

$$\tau: \Omega \times H \rightarrow \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\omega, \eta) \mapsto \tau(\omega, \eta)$$

où

$$\tau(\omega, \eta): \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\psi, \lambda) \mapsto \tau(\omega, \eta)(\psi, \lambda) \textit{ comme suit}$$
$$\tau(\omega, \eta)(\psi, \lambda) = (\underset{\chi=(\pi, \{\theta_i\}_{1 \leq i \leq n}), n>1}{\circ} (\kappa((<= id\ (\omega\ \pi))) \circ \sigma_{LOcc}(\chi, id)))(\psi, \lambda)$$

id est un nouvel identificateur de signal.

Traitement d'une fonction λ. Nous définissons ici la fonction recherche, pour tous les opérateur, les identifications d'expressions terminales possibles. Cette fonction est définie comme suit :

$$Ident: \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$\lambda \mapsto Ident(\lambda)$$
où
$$Ident(\lambda): \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\psi, \lambda_1) \mapsto Ident(\lambda)(\psi, \lambda_1) \text{ comme suit}$$
$$Ident(\lambda)(\psi, \lambda_1) = (\underset{\omega \in \Omega}{\circ} \tau(\omega, \lambda(\omega)))(\psi, \lambda_1)$$

Recherche de toutes les identifications. Nous définissons la fonction qui applique les pas d'identification jusqu'à ce que la fonction résultante soit égale à $\lambda_\varnothing$. Cette fonction est définie comme suit :

$$Itere: \mathcal{E} \times \mathcal{L} \rightarrow \mathcal{E} \times \mathcal{L}$$
$$(\psi, \lambda) \mapsto Itere(\psi, \lambda) \text{ comme suit}$$
$$si\ (\lambda = \lambda_\varnothing) alors\ Itere(\psi, \lambda) = (\psi, \lambda)$$
$$sinon\ Itere(\psi, \lambda) = (Itere \circ Ident(\lambda))(\psi, \lambda_\varnothing)$$

Algorithme. A partir des fonctions précédemment définies, l'algorithme se résume à :

$$Identification: \mathcal{E} \rightarrow \mathcal{E}$$
$$\psi \mapsto Itere(\psi, Init(\psi))$$

5.3. Coût en temps de l'identification

Pour déterminer le coût en temps de la phase d'identification des sous-expressions communes, nous considérons le cas d'une expression φ ayant n occurrences dans la description, en supposant que toute sous-expression de φ a également n occurrences dans la description. Nous donnons un calcul approximatif du coût en temps, l'objectif étant de montrer la complexité polynomiale.

Lemme 3.2. Le coût de la fonction *Init*, pour une description ψ, est de l'ordre de : $T(Init(\psi)) = O(\frac{1}{2}|\psi| * (|SE^+(\psi)| - Prof(\psi)))$.

Démonstration

Il est évident que le nombre d'appels de la fonction $Init_1$ est directement lié à la taille de la description ψ. Le coût en est modulé par le coût d'insertion d'une expression terminale dans l'ensemble des classes d'expressions terminales d'un opérateur donné. Ce coût est maximum si le nombre d'expressions terminales de ψ est maximum et si toutes ces expressions terminales ont même opérateur. Or, le nombre maximum d'expressions terminales, relativement à la profondeur de ψ et à la taille de la famille des sous-expressions non atomiques, est obtenue lorsque toutes les expressions de profondeur supérieure ou égale à 2 sont unaires. Ceci conduit à $|SE^1(\psi)| \leq |SE^+(\psi)| - Prof(\psi)$. Le nombre moyen d'arguments des expressions terminales est, dans ce cas, majoré par $\frac{|\psi|}{|SE^1(\psi)|}$. Ce qui conduit à un coût en comparaison de listes d'arguments majoré par :

$$(1+2+\ldots+(|SE^1(\psi)|-1))*\frac{|\psi|}{|SE^1(\psi|} = \frac{1}{2}|\psi|*(|SE^1(\psi)|-1).$$

En conclusion le coût de l'initialisation est donc inférieur à la somme de la taille de ψ et du coût en comparaisons. Le majorant est alors donné par la valeur : $T(Init(\psi)) = \frac{1}{2}|\psi|*(|SE^1(\psi)|+1) = \frac{1}{2}|\psi|*(|SE^+(\psi)| - Prof(\psi))$.

Pour déterminer une approximation du coût des autres pas d'identification, nous considérerons que seule l'expression φ a plusieurs occurrences dans la description ψ.

Lemme 3.3. Le nombre d'identificateurs créés est égal à $|SE^+(\varphi)|$.

Démonstration. Globalement, toutes les sous-expressions de φ, à savoir $|SE^+(\varphi)|$ expressions ont été reconnues comme ayant n occurrences. Ceci signifie qu'il y a eu définition de $|SE^+(\varphi)|$ nouveaux identificateurs dont l'affectation a été ajoutée dans l'expression de la description.

Lemme 3.4. Le coût de création de la définition des $|SE^+(\varphi)|$ identificateurs est égal à $T(\kappa(\psi),\varphi) = |SE^+(\varphi)|*(|Arg(\psi)|+1)$.

Démonstration. Les expressions définissant les identificateurs sont toutes ajoutées en tant qu'arguments directs de l'expression initiale. Comme il ne faut pas modifier l'ordre des arguments de l'expression initiale car les occurrences doivent être conservées, les expressions définissant les identificateurs peuvent être au mieux insérées en tant que $|Arg(\psi)|$-ième argument de ψ. D'où un coût d'accès égal à $|Arg(\psi)|+1$ pour l'insertion de chacun des identificateurs.

Le nombre de pas d'identification, c'est à dire le nombre d'appels de la fonction *Itere* est égal à la profondeur de l'expression φ. A chaque appel de la fonction *Itere,* la fonction *Ident* est appelée. Globalement, toutes les sous-expressions de φ, à savoir $|SE^+(\varphi)|$ expressions ont été reconnues comme ayant n occurrences. Ceci signifie qu'il y a eu définition de $|SE^+(\varphi)|$ nouveaux identificateurs dont l'affectation a été ajoutée dans l'expression de la description.

Nous notons $\varphi_1,\ldots,\varphi_n$ les n occurrences de l'expression φ.

Lemme 3.5. Le temps de substitution des occurrences des expressions communes par les nouveaux identificateurs peut être majoré par $|SE^+(\psi)|*(|\psi|-n*|\varphi|)+n*\frac{|\varphi|*(|\varphi|-1)}{2}$.

Démonstration. Pour chaque sous-expression non atomique de φ il faut, d'une part accéder à l'expression racine φ_i, puis à l'expression proprement dite. Donc il y a, pour chaque représentant φ_i, $|SE^+(\varphi)|$ accès à φ_i. Cette valeur correspond à $|SE^+(\varphi)|*(\sum_{i=1}^{i=n} d(\psi,\varphi_i))$. Le cumul des distances à la racine des expressions φ_i peut être majoré par la valeur $|\psi|-n*|\varphi|$, d'où un majorant : $|SE^+(\varphi)|*(|\psi|-n*|\varphi|)$

Enfin, après avoir acéder à la racine d'une occurrence de φ, il faut encore accéder à chacune de ses sous-expressions. Ces derniers accès sont majorés par la valeur $\frac{|\varphi|*(|\varphi|-1)}{2} = 1+2+\ldots+(|\varphi|-1)$, cas où chaque accès est de coût maximal.

Un majorant pour les substitutions des expressions par des identificateurs peut donc s'exprimer par : $|SE^+(\psi)|*(|\psi|-n*|\varphi|)+n*\frac{|\varphi|*(|\varphi|-1)}{2}$

Lemme 3.6. Le temps d'insertion des expressions terminales dans les classes d'équivalence est de l'ordre de $n*|SE^+(\varphi)|*(ln((|\varphi|+1)^{\frac{1}{(Prof(\varphi)+1)}})+\frac{ln(|\varphi|+1)}{(Prof(\varphi)+1)})$.

Démonstration. Le temps d'insertion d'une expression terminale après identification de ces sous-expressions dépend du nombre d'expressions dont les occurrences ont été identifiées au même pas, soit du nombre de classes d'équivalence générées à ce pas. Ce temps dépend également du nombre d'arguments de chacune des classes ainsi que des coûts éventuels de tri des arguments si l'opérateur est commutatif.

A un pas donné, le coût est maximal si toutes les expressions identifiées ont même opérateur et que cet opérateur est associatif. Pour donner une idée de ce coût, nous considérerons que toutes les expressions ont même arité, cette arité correspondant à l'arité moyenne déterminée par $a_{moy}=(|\varphi|+1)^{\frac{1}{(Prof(\varphi)+1)}}$. Cette expression est déduite en déterminant une base qui permet de représenter $|\varphi|+1$ nombres sur $Prof(\varphi)+1$ chiffres. L'arbre de l'expression est considéré bien balancé. Après l'initialisation, il y a donc $a_{moy}{}^{Prof(\varphi)}$ classes d'expressions ayant a_{moy} arguments. Puis pour les pas suivants, ceux qui nous concernent, nous avons :

pour tout $p=1,\ldots,Prof(\varphi)$, $a_{moy}{}^{(Prof(\varphi)-p)}$ classes de n expressions identiques.

Le coût du tri des listes d'arguments est donc de $|SE^+(\varphi)|*ln(a_{moy})=|SE^+(\varphi)|*ln((|\varphi|+1)^{\frac{1}{(Prof(\varphi)+1)}})=\frac{|SE^+(\varphi)|*ln(|\varphi|+1)}{(Prof(\varphi)+1)}$ qu'il faut multiplier par le nombre d'occurrences.

Les substitutions d'expressions identiques par un nouvel identificateur sont effectuées successivement. Les classes sont donc aussi créées successivement, car deux expressions terminales ne sont identiques que si elles ont le même nouvel identificateur en argument. Le coût en comparaison peut donc se ramener sur une seule classe, soit $|SE^+(\varphi)|*n*a_{moy}=|SE^+(\varphi)|*n*(|\varphi|+1)^{\frac{1}{(Prof(\varphi)+1)}}$. Cette valeur est conditionnée par une stratégie qui consisterait à insérer en tête l'expression terminale associée à la première substitution.

Théorème. Le coût de l'identification des sous-expressions communes est polynomial en la taille de la description, en la taille des sous-expressions communes et en nombre d'occurrences de ces sous-expressions.

Démonstration. Etant donné les résultats de complexité polynomiale donnés ci-dessus dans les lemmes 3.2, 3.3, 3.4, 3.5 et 3.6, leur somme est également polynomiale.

6. Exemple

Considérons la description VHDL donnée par la figure 3.8.

```
entity exemple is
   port ( E1, E2, E3, E4, E5 : in integer ;
          S1, S2, S3, S4, S5, S6 : out integer);
end exemple ;
architecture arch of exemple is
begin
   S1 <= (E1 + E2 + E3) when (E4 = E5) else (E1 * E2) ;
   S2 <= (E2 * E1) when (E4 = E6) else (E2 + E3) ;
   S3 <= (E2 + E1 + E3) * E2 * E1 ;
   S4 <= E1 *E2*(E3 + E2 + E1) ;
   S5 <= E1 + E2 + 5 ;
   S6 <= 4 when (E1 = E2) else 6 ;
end arch ;
```

Figure 3.8. Description VHDL de l'exemple.

Cette description conduit à l'expression décrite sous forme d'un arbre par la figure 3.9.

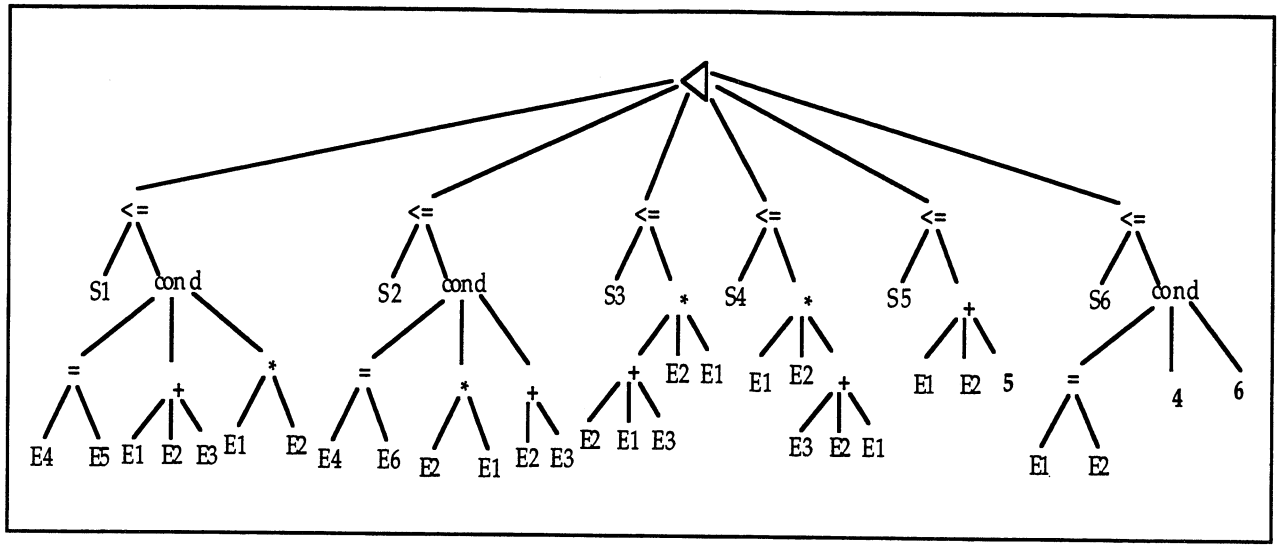

Figure 3.9. Arbre des expressions.

Le graphe de transfert d'information statique sans identification est donné par la figure 3.10 (a) et après identification par la figure 3.10 (b). Dans cette figure, les entrées et sorties sont nommées avec les mêmes noms que les signaux et les places correspondantes sont dupliquées pour une meilleure lisibilité. Dans ces figures nous ne montrons pas les sous-graphes correspondant aux signaux *S5* et *S6*, ceux-ci ne différant pas selon que l'identification est appliquée ou non.

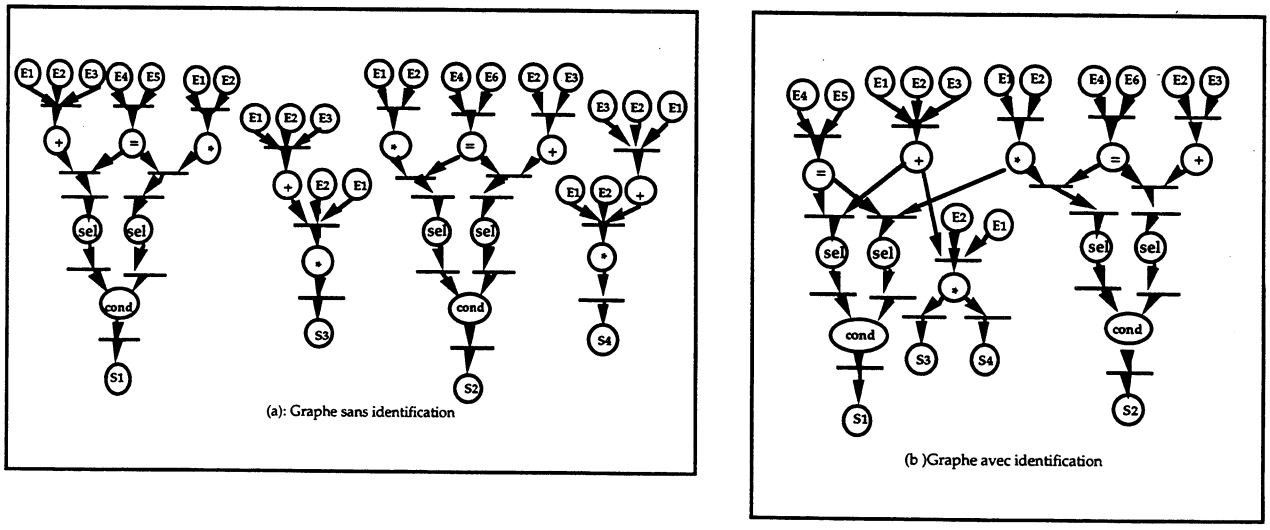

Figure 3.10 GTIS sans et après identification.

Nous détaillons ci-dessous les différentes étapes d'identification appliquées sur cet exemple.

Le résultat de l'appel de la fonction *Init* renvoie l'expression inchangée et l'application λ, pour les opérateurs qui sont significatifs, décrite par les trois tableaux ci-dessous. Chaque ligne d'un tableau représente une classe d'équivalence d'expressions terminales. Les occurrences sont notées par une liste de valeurs. Chaque valeur est le numéro d'ordre de l'argument en considérant la numérotation incrémentale à partir de 1 et de la gauche vers la droite.

opérateur "="

Arguments	Occurrences
{E4, E5}	{(1.2.1)}
{E4, E6}	{(2.2.1)}
{E1, E2}	{(6.2.1)}

opérateur "+"

Arguments	Occurrences
{E1, E2, E3}	{(1.2.2), (3.2.1), (4.2.3)}
{E2, E3}	{(2.2.3)}
{E1, E2, 5}	{(2.2)}

opérateur "*"

Arguments	Occurrences
{E1, E2}	{{(1.2.3), (2.2.2)}

L'analyse de ces classes d'équivalence par la fonction *Ident* conduit à l'expression de la description décrite par la figure 3.11 et à une nouvelle application λ décrite par le tableau ci-dessous. Après initialisation de l'environnement, deux nouveaux identificateurs de signaux sont introduits, *NS1* et *NS2*.

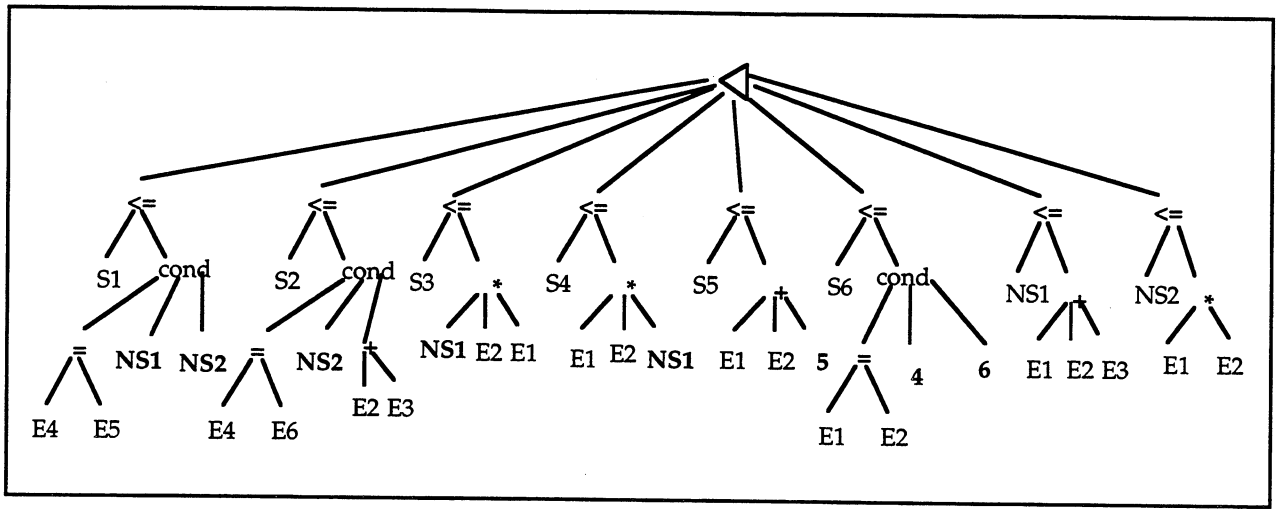

Figure 3.11. Arbre après le premier pas d'identification.

opérateur "*"

Arguments	Occurrences
{E1, E2, NS1}	{(3.2), (4.2)}

Ici encore, la seule classe existante contient plus d'un élément. Il en résulte une nouvelle étape d'identification qui conduit à l'expression donnée par la figure 3.12, la fonction λ étant égale à $\lambda_{\varnothing}$. L'identificateur *NS3* est ajouté à la description.

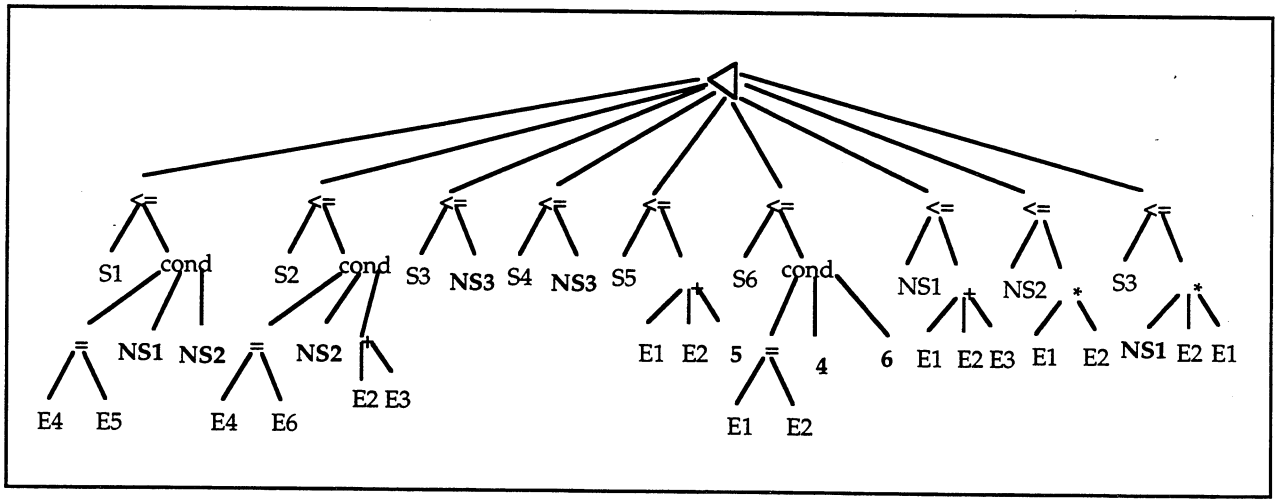

Figure 3.12. Arbre après le dernier pas d'identification.

7. Conclusion

Dans ce chapitre, nous avons montré le processus de compilation d'une description VHDL flot de donnée sous forme d'un graphe de transfert d'information statique. Un tel graphe représente les fonctionnalités de la description.

La compilation prend en compte les deux composantes d'une description, à savoir la déclaration d'entité et la déclaration d'architecture. Ces deux composantes permettent de construire un élément de la bibliothèque appelé type-SATAN. Alors que la déclaration d'entité conduit à la définition du nom du type et des entrées/sorties fonctionnelles, la déclaration de l'architecture interne conduit à la définition du graphe de transfert d'information.

Bien que dans l'analyse des signaux externes de la déclaration d'entité nous prenions en compte des signaux de type tableau, ces derniers ne sont pas pris en compte de façon satisfaisante dans le corps de l'architecture. Ceci est dû au fait que le traitement des signaux composites dans une description structurelle (cf. chapitre 5) est clairement défini alors qu'au niveau d'une description de comportement il faudrait envisager un traitement plus poussé et plus complexe qui permettrait d'obtenir un résultat représentatif de l'utilisation des signaux composites.

Après avoir défini le sous-ensemble du langage VHDL que nous prenons en compte pour la compilation, nous avons donné la définition du GTIS ainsi que sa construction avec des étapes de simplification et de réduction de la taille du graphe. Les algorithmes spécifiés sont de complexité polynomiale par rapport à la taille lexicographique de la description VHDL.

A partir d'une description dans cette syntaxe, les règles d'équivalence entre les expressions et le graphe de transfert d'information montrent que la simplification des expressions et la reconnaissance des expressions communes aux différentes instructions amènent à une réduction du graphe en terme du nombre de place et de transitions. Or, les traitements liés à l'analyse de testabilité, la recherche des écoulements d'information notamment, sont d'un coût important vis à vis de la taille du graphe, et réduire cette taille peut apporter un gain en temps non négligeable. Ceci d'autant plus que la

compilation d'une description sous forme d'un type SATAN ne se fait qu'une fois, les utilisations ultérieures dans des descriptions structurelles consistant à extraire le type SATAN de la bibliothèque.

Les traitement présentés dans ce chapitre ont été implémentés sous la forme d'un prototype écrit en langage LISP et en langage C.

Chapitre 4

Graphe de Transfert d'Information Dynamique

1. Introduction

La première partie de ce document présentait la détermination d'un graphe de testabilité (places, transitions et arcs) uniquement décoré par la fonction des différentes places, ce graphe étant déduit d'une description VHDL. Dans le but de calculer les mesures de testabilité liées à la quantité d'information, nous cherchons à déduire également les paramètres nécessaires à ce calcul. Le principe de calcul des mesures de testabilité est basé sur la notion de flot d'information. Ainsi, une mesure de contrôlabilité et une mesure d'observabilité sont associées aux modules pour chacun des écoulements. La contrôlabilité mesure une approximation de la quantité d'information disponible à l'entrée d'un module à partir des sources de l'écoulement. La mesure d'observabilité est une approximation de la quantité d'information, en sortie d'un module, observable aux puits d'un écoulement.

L'approximation des quantités d'information qui peuvent transiter à travers le graphe est spécifiée localement par la notion de capacité d'information associée aux arcs, aux entrées et aux sorties des modules. La capacité d'information exprime la quantité d'information maximale pouvant traverser un élément.

Ces valeurs sont facilement déterminées lorsqu'il n'y a pas de cycles dans le graphe. Mais, dans le cas de cycles cette tâche est difficile à réaliser, car il est nécessaire de représenter une capacité d'information associée à plusieurs activations de fonctions cycliques.

Pour résoudre ce problème, la notion de débit d'information permet de prendre en compte les aspects dynamiques. Dans notre étude, il s'agit tout d'abord de détecter certaines structures séquentielles cycliques, de formaliser les différents éléments qui la composent et de donner les limitations sur le type de descriptions que nous prendrons en compte.

Ensuite, nous étudions la possibilité de maximiser le flot d'information à travers une telle structure en déterminant les aspects temporels qui permettent de l'atteindre. Ces aspects temporels ne sont pas explicités en termes de dates mais

uniquement en termes du nombre d'activations de certaines expressions. Ces nombres d'activations permettent de connaître des capacités dynamiques qui conduisent à des mesures de testabilité plus réalistes. Pour déterminer les valeurs nécessaires au calcul des mesures, nous présentons aussi la propagation à travers un écoulement des activations multiples de certaines fonctions.

2. Mesures de testabilité dans SATAN

Les mesures de testabilité basées sur la théorie du transfert de l'information consistent à déterminer une mesure de contrôlabilité et une mesure d'observabilité en calculant une borne supérieure du flot d'information depuis les entrées vers un module donné et d'un module vers les sorties. Le flot d'information est déterminé par un algorithme de coupe minimale/flot maximal et prend en compte deux caractéristiques dans le graphe : la capacité d'une ligne (ou d'un arc) et la capacité d'un module (fonction).

2.1. Rappels sur la théorie de l'information

Définitions. Une information X est une variable aléatoire sur le domaine de valeurs $x_1, x_2, \ldots, x_n$ avec comme distribution de probabilités : $P_{x_i} = p(X = x_i), 1 \leq i \leq n$. La quantité d'information associée à une telle variable est définie par :

$$H(X) = -\sum_{i=1}^{i=n} p_{x_i} log_2 p_{x_i}$$

A partirde cette définition, la quantité d'information associée à une source est maximale lorsque la distribution de probabilités est uniforme, et la valeur maximale est $H(X) = \log_2 n$.

Définition. Un canal d'information est soit un élément de support d'information soit une fonction de transformation de l'information. Un canal d'information est constitué d'une information d'entrée X et d'une information de sortie Y. Y est la

variable aléatoire résultant de la variable aléatoire X à travers le canal. Un canal d'information est dénoté par $c\,(X\,;\,Y)$.

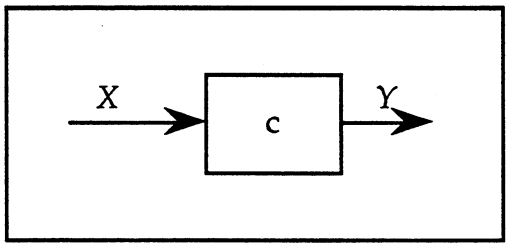

Figure 4.1. Canal d'information.

La capacité d'information d'un canal $c\,(X\,;\,Y)$ est définie comme la quantité d'information maximale disponible en sortie du canal à la vue des distributions possibles sur l'entrée :

$$C(c(X;Y)) = \underset{\substack{distributions \\ sur\ X}}{Max}\ H(Y)$$

Dans cette étude nous considérerons la capacité d'un canal comme étant la capacité de la source de sortie soit :

$$C(c(X;Y)) = \underset{\substack{distributions \\ sur\ X}}{Max}\ H(X) = log_2\, n$$

où n est le nombre de valeurs du domaine de la variable aléatoire X.

Dans notre étude nous cherchons à déterminer la valeur des capacités des différentes lignes et modules à partir de descriptions VHDL. Dans le cas général, la capacité d'une ligne peut être déterminée facilement car une ligne correspond toujours à un signal, explicitement déclaré ou implicite, dont le type est connu, et donc également le nombre de valeurs distinctes possibles. La capacité d'information est donc le logarithme du cardinal du domaine de valeurs.

Les modules correspondent à des fonctions, nous leur associons deux capacités : l'une associée au domaine d'entrée notée C_i et l'autre associée au domaine de sortie notéeC_o.

2.2. Mesures de testabilité

Les mesures de testabilité, contrôlabilité et observabilité, sont calculées pour chaque module dans un écoulement d'information donné. La mesure de contrôlabilité exprime la facilité d'amener des valeurs de test en entrée d'un module à partir des entrées de l'écoulement. La mesure d'observabilité exprime la facilité d'observation de la sortie d'un module à partir de l'observation des puits de l'écoulement.

Les valeurs de capacité des arcs et des modules de l'écoulement permettent de mesurer la quantité d'information maximale pouvant transiter au travers de l'arc ou du module. Ainsi, la mesure de contrôlabilité est déterminée, d'une part, par la quantité d'information maximale qui peut transiter à travers le sous-graphe menant des sources vers l'entrée d'un module et, d'autre part, par la capacité d'information en entrée du module. Similairement, la mesure d'observabilité est déterminée, d'une part, par la quantité d'information transitant à travers le sous-graphe menant de la sortie du module vers les puits et, d'autre part, par la capacité d'information en sortie du module.

La mesure de la quantité d'information qui peut traverser un sous-graphe est déterminée par la notion de transinformation. Cette valeur est pratiquement calculée par un algorithme de coupe minimale et flot maximal. En effet, la quantité d'information pouvant traverser un sous-graphe est majorée par la plus petite capacité d'information qui forme une coupe (i.e. un passage obligatoire). De plus, un canal peut toutjours faire transiter une quantité d'information inférieure à la valeur de sa capacité. La transinformation entre deux places p et q est notée $T(p\ ;\ q)$.

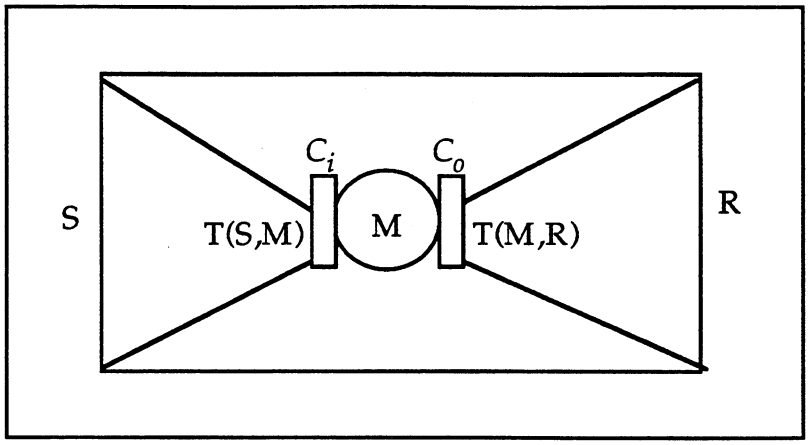

Figure 4.2. Mesures de testabilité.

Le problème que nous soulevons dans ce chapitre se situe dans la détermination de la capacité des arcs et des places. La capacité d'information peut être déterminée directement à partir du type des signaux qui induit la cardinalité des valeurs pouvant transiter à travers un arc ou une place. Cette capacité est appelée capacité d'information statique car elle décrit la quantité d'information qui peut transiter à un instant donné.

Mais certaines configurations, que nous appellerons fonctions ou structures séquentielles cycliques, ne peuvent être prises en compte directement sous cette forme en termes de capacité d'information. Par exemple, un registre à décalage, avec entrée parallèle et sortie série, doit être considéré dans le sens où l'on active plusieurs fois la sortie et le décalage pour connaître la valeur initialement chargée en entrée ; or la capacité de la sortie étant de 1 alors que la capacité d'entrée est supérieure à 1. La représentation de la capacité d'information statique indiquerait un problème d'observabilité.

Le paragraphe suivant a pour objectif d'illustrer le problème de la représentation du graphe des capacités d'information pour les structures séquentielles cycliques. Pour illustrer ce problème, nous donnons les exemples d'un registre à décalage et d'un compteur.

2.3. Exemples

La détermination des capacités d'information est directe à partir de la taille des chemins d'information lorsqu'il n'y a pas de cycle de dépendance dans la description VHDL. Mais, pour les descriptions incluant des cycles de dépendance, il ne suffit plus uniquement de prendre en compte la taille du chemin. En effet, pour ce type de descriptions il faut envisager éventuellement plusieurs activations successives pour décrire correctement le flot d'information. Les exemples présentés dans ce paragraphe montrent, sans présenter de solution, l'inadéquation d'une représentation par des capacités d'information statiques.

2.3.1. Registre à décalage

Considérons un registre à décalage ayant une entrée parallèle ou une entrée série et une sortie série. La description VHDL de cette cellule est la suivante :

```
entity shift is
    port (E : in positive range 0 to 255 ;
          ES : in positive range 0 to 1 ;
          ck, PS : in boolean ;
          S : out positive range 0 to 1)
end shift ;
architecture flot of shift is
    signal M : positive range 0 to 255 ;
begin
    SH : (ck and not ck'stable)
         begin
              M <= guarded E when PS else M div 2 + 128 * ES ;
         end block ;
    S <= M mod 2 ;
end ;
```

ck	PS	E	ES	M	S
↑	false	V	X	V	M mod 2
↑	true	X	v	décalage	M mod 2

Tableau 4.1. Diagramme fonctionnel du registre à décalage.

Le graphe de transfert d'information statique est montré par la figure 4.3. Pour les places, les capacités d'information statiques en entrée (Ci) et en sortie (Co) sont données dans le tableau 4.2. Les mesures de contrôlabilité et observabilité, pour l'écoulement associé au chargement de l'entrée parallèle (indiqué en gras sur le graphe) sont données dans le tableau 4.3.

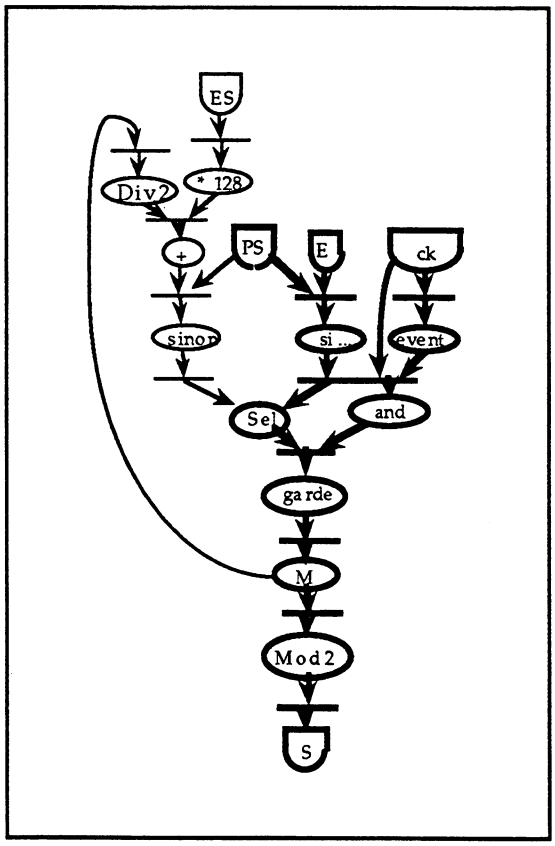

Figure 4.3. Graphe du registre à décalage.

places	capacité d'entrée	capacité de sortie
E	8	8

M	8	8
S	1	1

Tableau 4.2. Capacités d'information.

	E	ck	si	sel	garde	M	mod 2	S	event	and
CO	1	1	1	1	1	1	1	1	1	1
OB	0.125	1	0.125	0.125	1	0.125	1	1	1	1

Tableau 4.3. Mesures de testabilité.

A la vue de ces résultats, il ressort un problème d'observabilité de la valeur chargée dans le point de mémorisation M. Ce problème mis en évidence par le modèle de testabilité n'est pas un problème réel car il est évident que pour une séquence d'activation de la fonction de décalage nous obtenons une observabilité parfaite de la valeur contenue dans le registre.

2.3.2. Compteur

Considérons un compteur ayant une entrée de type horloge et une fonction de mise à zéro. La description VHDL de cette cellule est la suivante :

```
entity count is
    port(ck, RAZ : in boolean ;
        S : out positive range 0 to 255)
end shift ;
architecture flot of count is
    signal M : positive range 0 to 255 ;
begin
    SH : (RAZ or ck and not ck'stable)
        begin
            M <= guarded 0 when RAZ else (M + 1) mod 256;
        end block ;
    S <= M ;
end ;
```

Le graphe de transfert d'information statique est donné par la figure 4.4. Les valeurs de capacité d'entrée (Ci) et de sortie (Co) des places sont données dans le tableau 4.4. Les mesures de contrôlabilité et d'observabilité, pour l'unique écoulement, sont données dans le tableau 4.5.

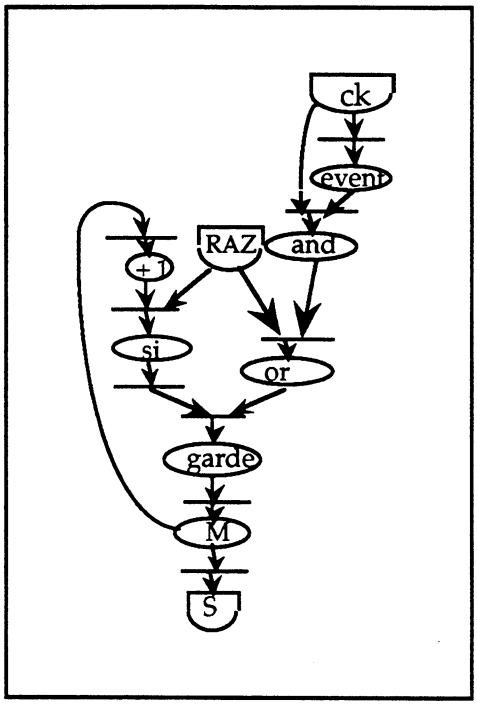

Figure 4.4. Graphe du compteur.

places	capacité d'entrée	capacité de sortie
ck	1	1
M	8	8
S	8	8

Tableau 4.4. Capacités d'information.

	ck	si	garde	M	S	event	and
CO	1	0.125	1	0.125	0.125	1	1
OB	1	1	1	1	1	1	1

Tableau 4.5. Mesures de testabilité.

A la vue de ces résultats, il ressort un problème de contrôlabilité de la valeur chargée dans le point de mémorisation. Ce problème mis en évidence par le modèle de testabilité n'est pas un problème réel car il est évident que pour une séquence d'activation de la fonction de comptage nous obtenons une observabilité parfaite de la valeur contenue dans le point de mémorisation.

2.3.3. Conclusion

Ces deux exemples mettent en évidence la faiblesse de l'interprétation directe de certaines constructions du langage. En effet, cette interprétation fait apparaître des problèmes d'observabilité et de contrôlabilité, qui sont réels si l'on considère le point de vue statique. Toutefois, il est possible, dans ces deux exemples, de contrôler et d'observer complètement les différents modules si l'on considère le point de vue dynamique. C'est à dire la quantité d'information qui peut traverser cette construction avec plusieurs activations de certaines fonctions.

Nous avons donc été amenés à définir le graphe de transfert d'information dynamique qui prend en compte des capacités d'information statiques et des activations multiples. Ces deux caractéristiques sont dissociées dans un premier temps car elles sont déterminées tout d'abord localement au niveau d'une construction du langage VHDL. Puis, avant de calculer les mesures de testabilité, il s'agit de propager les activations multiples à travers un écoulement.

2.4. Définition du graphe de transfert d'information dynamique

Un graphe de transfert d'information dynamique est défini à partir d'un graphe de transfert d'information statique. Pour pouvoir représenter les capacités dynamiques, nous allons redéfinir le graphe de transfert d'information en complétant la définition du Graphe de Transfert d'Information Statique donnée dans le chapitre 3. Les attributs supplémentaires concernent, d'une part, les capacités d'information statiques des arcs et places et, d'autre part, le nombre d'activations d'un arc ou d'une place. Concernant les places, nous distinguons :

- la capacité de l'entrée,

- la capacité de la sortie,
- le nombre d'activations du flot d'entrée,
- le nombre d'activations du flot de sortie,
- le nombre d'activations globales de la place.

Définition. Un Graphe de Transfert d'Information Dynamique (GTID) est un septuplet $G = (P, T, \alpha, \beta, Expr, \gamma_e, \gamma_p)$, où :

- $(P, T, \alpha, \beta, Expr)$ est un graphe de transfert d'information statique ;
- γ_e est l'application qui à un arc associe le couple formé de la capacité statique de l'arc et le nombre d'activations de l'arc :

$$\gamma_e : \alpha \cup \beta \rightarrow R \times (N \cup \{x\})$$

$e \mapsto (ce, ne)$ où ce est la capacité statique et ne le nombre d'activations de l'arc. La valeur x signifie que le nombre d'activations est tiré aléatoirement entre 1 et 2^{ce}, l'information est liée à la longueur de la séquence ;

- γ_p est l'application qui à une place associe le quintuplet définissant sa capacité de sortie, la longueur de la séquence, le nombre d'activations d'entrée et de sortie :

$$\gamma_p : P \rightarrow R \times R \times (N \cup \{x\}) \times (N \cup \{x\}) \times (N \cup \{x\})$$

$p \mapsto (ci, co, ni, no, np)$, où ci est la capacité statique du port d'entrée, co la capacité statique du port de sortie, ni le nombre d'activations de l'entrée, no le nombre d'activations de la sortie et np le nombre d'activations de l'opérateur.

Les deux valeurs ni et no sont liées, elles définissent le rapport entre le nombre d'activations nécessaires en entrée pour une activation en sortie ou, inversement, le nombre d'activations nécessaires en sortie pour une activation d'entrée. Ces deux valeurs sont donc telles que : $ni = 1$ ou $no = 1$, en supposant que la fonction associée à la place n'est activée qu'une seule fois. Ces caractéristiques intrinsèques à l'opérateur sont déterminées lors de la compilation d'une description VHDL. Par

contre, np est déterminé à l'intérieur d'un chemin d'information, cette valeur est calculée par la propagation d'activations multiples d'autres places.

Pour simplifier les notations ultérieures, nous définissons les fonctions Ce, Ci, Co, Ne, Ni, No, Np qui correspondent à la projection des fonctions γ_e et γ_p sur les champs ce, ci, co, ni, no et np.

3. Construction du GTID

3.1. Détermination des capacités d'information statiques

La capacité d'information statique d'un arc ou d'une place est directement liée au type du signal ou de l'expression VHDL qui a engendré l'élément du graphe. La capacité est égale au logarithme base 2 du nombre de valeurs que définit le type. Il est donc important, dans la description VHDL, de bien définir les types des signaux manipulés pour obtenir une bonne précision dans le calcul des mesures de testabilité. Nous distinguons deux types de places dans un graphe : celles qui sont associées à un signal déclaré dans l'entité ou dans l'architecture, et celles qui sont liées à un opérateur.

Définition. Pour toute valeur réelle x nous noton $\lceil x \rceil$ (resp. $\lfloor x \rfloor$) sa partie entière supérieure (resp. sa partie entière inférieure).

3.1.1. Capacité statique d'une place liée à un signal

Soit s un signal et τ son type. Les capacités d'entrée et de sortie de la place associée à ce signal sont respectivement Ci et Co toutes deux égales à $log_2(|\tau|)$.

3.1.2. Capacité d'information d'un arc.

Soit t une transition. Soit $\Gamma^-_{t,G}(t) = \{p_1,\ldots,p_n\}$ l'ensemble des places prédécesseurs de t. Nous définissons la capacité des arcs prédécesseurs et successeurs, voir figure 4.5, de la manière suivante :

- $\forall e_i = (p_i,t) \in A^-_t(t), \gamma_e(e_i) = (C_o(p_i),1)$

- soit $\{e^+\} = A_t^+(t), \gamma_e(e^+) = (\sum_{e_i \in A_t^-(t)} C_e(e_i), 1)$

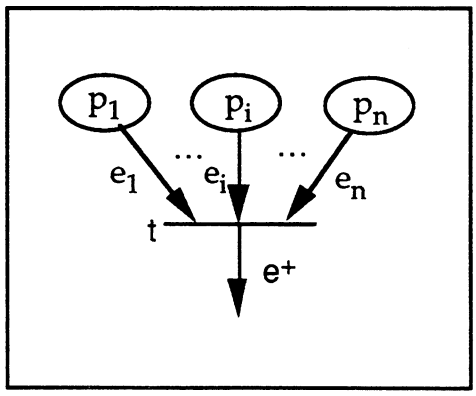

Figure 4.5. Capacité des arcs.

3.1.3. Capacité d'information statique d'une place liée à un opérateur.

Soit p une place dont l'opération associée est $(\omega\, p_1 \ldots p_n)$. Cette place n'a qu'une seule transition prédécesseur car l'opérateur n'est pas du type *sélection* ou *lgarde*. Dans ce cas, la capacité d'entrée de la place est définie par la capacité de l'arc entrant (sauf dans le cas des fonctions internes) et la capacité de sortie dépend de l'opérateur et de la capacité de sortie des places prédécesseurs $p_1 \ldots p_n$. Ci-dessous nous étudions les différents opérateurs :

a) Opérateurs logiques et relationnels

Pour les opérateurs logiques ou relationnels, le résultat étant toujours booléen, la capacité de sortie du module est toujours égale à 1.

b) Opérateurs arithmétiques sur des types intervalles d'entiers

b.1) Opérateur additif (addition et soustraction)

Dans le cas d'un opérateur additif, nous déterminons la capacité du signal implicite de sortie par le domaine de valeurs possibles en supposant que nous ayons des intervalles de nombres positifs. Nous présentons le résultat pour

l'addition, la soustraction se ramenant à une addition par le biais de l'utilisation de l'opposé.

Chaque place prédécesseur a une capacité de sortie $C_o(p_i)$ que nous notons $C_{o,i}$, le domaine de valeur de chacune de ces sorties ce ramène donc à l'intervalle $[0, 2^{C_{o,i}} - 1]$, l'intervalle de valeur de la somme est donc $[0, \sum_{i=1}^{i=n}(2^{C_{o,i}} - 1)]$.

Pour majorer la capacité correspondante, nous approximons l'intervalle par l'intervalle $[0, (\sum_{i=1}^{i=n} 2^{C_{o,i}}) - 1]$, la capacité étant alors $log_2(\sum_{i=1}^{i=n} 2^{C_{o,i}})$ dont nous prendrons la partie entière supérieure.

b2) Multiplication

La multiplication de nombres ne donne pas toutes les valeurs de l'intervalle résultant et la détermination du nombre de valeurs atteintes n'est pas immédiate. Ainsi, plutôt que de tenter de définir une valeur exacte de la capacité, nous optons pour une capacité maximale déterminée par les bornes de l'intervalle résultant. Ce choix est justifié par le fait que la taille du chemin d'information sera réduite lors de l'affectation du résultat sur un signal d'un type donné. La capacité de sortie de l'opérateur est donc la somme des capacités des places prédécesseurs, soit aussi la capacité d'entrée de la place : $C_o(p) = C_i(p)$.

b.3) Opérateur modulo

Pour l'opérateur modulo, nous distinguons plusieurs cas selon qu'un des deux arguments est une valeur constante ou non.

Si les deux arguments sont non constants, le domaine de valeurs maximum en sortie est $[0, min(2^{C_{o,1}} - 1, 2^{C_{o,2}} - 1)]$, et la capacité de sortie de la place est $min(C_{o,1}, C_{o,2})$.

Si l'un des arguments est constant, le domaine de valeurs du résultat dépend de cette constante et de la capacité de la place argument non constante. Soit v une valeur constante et soit l'expression d'une place p de la forme $(mod\ p_1\ v)$; la capacité de sortie de la place p est alors $C_o(p) = min(log_2 v, C_o(p_1))$, cette valeur étant le logarithme du nombre de valeurs résultantes possibles. Dans le cas d'une

expression de la forme $(mod\ v\ p_1)$, l'intervalle des valeurs résultantes possibles est $[0,\lceil v/2 \rceil]$, ce qui conduit à une capacité statique en sortie de la place p égale à $C_o(p) = log_2(1+\lceil v/2 \rceil)$.

b.4) Opérateur diviseur

Soit une place p dont l'expression est $(div\ p_1\ p_2)$; le nombre de valeurs possibles en résultat est au plus le nombre de valeurs de p_1. La capacité de la place p est donc $C_o(p) = C_o(p_1)$. Par contre, si l'un des deux arguments est constant, il faut envisager les cas suivants :

- si l'expression est de la forme $(div\ p_1\ v)$, l'intervalle des valeurs possibles est $[0,E^-(\frac{2^{C_o(p_1)}}{v})]$, ce qui conduit à une capacité en sortie de la place p égale à $C_o(p) = log_2(1+E^-(\frac{2^{C_o(p_1)}}{v}))$;
- si l'expression est de la forme $(div\ v\ p_1)$, le nombre de valeurs possibles est $2*\sqrt{v}$, ce qui conduit à une capacité en sortie de la place p égale à $C_o(p) = 1+\frac{1}{2}log_2(v)$;

c) Opérateurs conditionnels

Le tableau 4.6 montre la capacité de sortie d'une place p en fonction de l'opération qu'elle représente.

expression de la place p	Capacité de sortie
$(si\ p_1\ alors\ p_2\ sinon\ \varepsilon)$ ou $(si\ p_1\ alors\ \varepsilon\ sinon\ p_2)$	$C_o(p) = C_o(p_2).$
$(si\ p_1\ alors\ p_2\ sinon\ v)$ ou $(si\ p_1\ alors\ v\ sinon\ p_2)$	$C_o(p) = C_o(p_2).$
$(si\ p_1\ alors\ v_1\ sinon\ v_2)$	$C_o(p) = 1.$
$(si\ p_1 \in \{v_1,\ldots,v_n\}\ alors\ p_2\ sinon\ \varepsilon)$ ou $(si\ p_1 \in \{v_1,\ldots,v_n\}\ alors\ \varepsilon\ sinon\ p_2)$	$C_o(p) = Max(C_o(p_2),1).$
$(selection\ p_1 \ldots p_n)$	$C_o(p) = Max(\overset{i=n}{\underset{i=1}{Max}} C_o(p_i), log_2\lvert\{p_i, 1 \le i \le n / C_o(p_i) = 1\}\rvert)$
$(garde\ p_1\ p_2)$	$C_o(p) = Max(C_o(p_2),1).$
$(lgarde\ p_1 \ldots p_n)$	$C_o(p) = Max(\overset{i=n}{\underset{i=1}{Max}} C_o(p_i), log_2\lvert\{p_i, 1 \le i \le n / C_o(p_i) = 1\}\rvert)$
$(<= s\ p_2)$	$C_o(p) = \lvert\tau\rvert$, où τ est le type du signal s.

Tableau 4.6. Capacité de sortie des opérateurs conditionnels.

3.2. Structures séquentielles cycliques

Certaines configurations nécessitent d'introduire la notion de débit d'information. Le problème posé consistera alors à déterminer les contraintes temporelles qui permettront de maximiser le flot d'information à travers une telle structure.

Dans cette partie nous allons formaliser la notion de structure séquentielle cyclique, puis définir ses composantes par l'analyse des expressions gardées du langage VHDL.

Une structure séquentielle n'est considérée que si elle est définie par une expression gardée ; elle détermine la valeur d'un signal M qui est un point de

mémorisation. La forme générale de telles structures peut être représentée par l'expression VHDL sous la forme déclarative suivante :

```
block (Garde)
begin
    M <= guarded      W1 when C1 else
                      W2 when C2 else
                      ...
                      Wn-1 when Cn-1 else
                      Wn;
end block
```

Cette expression signifie que le signal M n'est modifié que si la condition de garde (Garde) est vraie. La nouvelle valeur de M est déterminée par l'expression W_i correspondant à C_i,, où C_i, est la première condition évaluée à vrai des expressions $C_1, C_2, \ldots, C_{n-1}, C_n$. Les expressions W_i sont appelées chronogrammes, elle constituent l'ensemble des sources de M en termes de données. Pour expliciter la solution que nous proposons au problème du transfert d'information à travers une telle structure, nous allons en étudier plus précisément les différentes composantes.

Définition. *Fonction de sortie.* Une fonction de sortie d'une structure séquentielle cyclique définissant la valeur d'un point de mémorisation M est une expression dans laquelle M est référencé et dont M ne dépend pas. Nous notons F_{sortie} une telle expression.

3.2.1. Forme générale des expressions de garde

Une expression ou condition de garde fait apparaître deux principales composantes : une composante asynchrone (elle fait référence à des commandes)

que nous noterons ψ^A et une composante synchrone (de type événementiel) que nous noterons ψ^S.

Une condition de garde s'écrit donc ψ^A *or* ψ^S, où ψ^A fait référence à des valeurs de signaux (niveaux) et ψ^S à des valeurs et des événements (fronts) sur des signaux.

La composante asynchrone est donc une expression booléenne qui ne fait intervenir aucun aspect temporel.

La composante synchrone ne peut, à un instant donné, être valide que sur un seul événement : en effet, par définition, deux événements ne peuvent avoir lieu en même temps. Par conséquent, une telle expression est une disjonction d'expressions synchrones élémentaires φ_i^S qui sont elles-mêmes la conjonction entre une expression asynchrone et le test d'un front sur un signal (horloge) :

$$\psi^S = \underset{i=1}{\overset{i=k}{or}}\, \varphi_i^S$$

$$\varphi_i^S = \varphi_i^A \; and \; event(s_i)$$

Les k signaux s_i forment l'ensemble des horloges qui déclenchent l'affectation de M de façon synchrone.

3.2.2. Forme générale des expressions W_i (chronogrammes)

Pour une validation de la condition de garde à une date donnée et pour un environnement de valeurs des signaux à cette même date, la nouvelle valeur affectée à M est donnée par l'une des expressions W_i (chronogramme). Une telle expression est soit constante, soit dépendante d'un certain nombre de signaux. L'ensemble des expressions $W_i, 1 \leq i \leq n$ peut être partitionné en trois classes :

- la classe des expressions dont la valeur ne dépend pas de la valeur courante du point de mémorisation M, mais uniquement d'entrées de type donnée. Nous notons F_{init} une telle expression ;
- la classe des expressions dont la valeur dépend de la valeur courante du point de mémorisation M et de la valeur de certaines entrées de type donnée. Nous notons F_{evol} une telle expression ;

- la classe des expressions dont la valeur dépend uniquement de la valeur courante du point de mémorisation M. Nous notons F_{inter} une telle expression.

Les chronogrammes de la première classe peuvent être interprétés comme des expressions ou fonctions d'initialisation du signal M (sa valeur est déterminée par l'extérieur) ; les chronogrammes de la seconde classe peuvent être considérés comme des expressions ou fonctions d'évolution du signal s (sa valeur future dépend de l'extérieur et de sa valeur courante) alors que les chronogrammes de la dernière classe seront considérés comme des fonctions internes de la structure. Nous appellerons W_{init} l'ensemble des expressions de la première classe, W_{evol} l'ensemble des expressions de la seconde classe et W_{inter} l'ensemble des expressions de la dernière classe.

3.2.3. Propriétés conjointes entre l'expression de garde et les chronogrammes

A chacune des deux composantes (asynchrone et synchrone) de l'expression de garde, on peut associer l'ensemble des expressions chronogrammes qui peuvent définir la nouvelle valeur du point de mémorisation M. Soit W_A l'ensemble des expressions calculées lorsque ψ^A est active, et W_S l'ensemble des expressions calculées lorsque ψ^S est active. Les propriétés qui lient ces deux ensembles sont :

- Une expression peut être liée à la composante synchrone, à la composante asynchrone ou aux deux composantes à la fois :

  $W_A \cap W_A \subset \{W_1, \ldots, W_n\}$;

Et plus précisément :

- une expression d'initialisation peut être liée à n'importe quelle composante de la garde ou aux deux composantes :

  $W_{init} \subset (W_A \cup W_S)$;

- une expression liée à la composante asynchrone ne peut être qu'une expression d'initialisation :

  $W_A \subset W_{init}$;

- ou inversement une expression d'évolution ne peut être que liée à la composante synchrone :

  $(W_{evol} \cup W_{inter}) \subset W_S$ et $(W_{evol} \cup W_{inter}) \cap W_A = \varnothing$.

3.2.4. Hypothèses restrictives de cette étude.

Dans le cadre de cette étude, nous n'envisagerons pas toutes les possibilités en termes d'expressions séquentielles cycliques. La restriction que nous donnons concerne le nombre de fonctions d'évolution ou de fonctions internes que nous limitons à un. Le problème dans le cas contraire réside dans la difficulté d'associer la bonne condition de garde à une fonction cyclique, ainsi que la bonne fonction cyclique à une fonction de sortie. L'extension de cette étude préliminaire peut être envisagée soit par le biais d'une interaction avec l'utilisateur (ce qui diminue les aspects 'presse-bouton' de la fonction testabilité) soit par le biais de calculs symboliques poussés aussi bien au niveau des expressions de contrôle que des expressions cycliques.

a) Condition de garde :

Pour la condition de garde nous considérons les deux formes suivantes :

- s_A *or* $(c_s$ *and ck'front)*
- $(c_s$ *and ck'front)*

dans lesquelles, s_A est un identificateur de signal spécifiant la composante asynchrone, c_s un identificateur de signal spécifiant la condition asynchrone de la composante synchrone, et *ck* l'identificateur de signal dont un événement peut déclencher la composante synchrone.

Cette limitation n'est en fait qu'une limitation en termes de représentation, car compte tenu du fait qu'une seule fonction cyclique est admise, on peut toujours décrire une structure séquentielle cyclique sous cette forme là.

b) Fonctions d'initialisation

Dans l'affectation du point de mémorisation, nous considérons au maximum deux fonctions d'initialisation, et, dans le cas où il y en a deux, l'une doit être activée par la composante asynchrone de la garde et l'autre par la composante synchrone. Chacune de ces fonctions doit être définie par un identificateur de signal ce qui permet de représenter tous les types de fonctions d'initialisation. De plus, dans le cas d'une fonction d'initialisation activée par la composante synchrone de l'expression de garde, une condition supplémentaire doit permettre de dissocier l'activation de la fonction d'initialisation de la fonction d'évolution ou interne. Cette condition devra elle aussi être définie par un identificateur de signal.

Nous notons :

- E_A et E_S les identificateurs de signaux correspondant aux entrées d'initialisation associées respectivement à la composante asynchrone et à la composante synchrone de la garde ;
- C_{init} l'identificateur de signal correspondant à la condition de transfert de la valeur E_S dans le point de mémorisation. Il différencie l'activation de la fonction d'initialisation synchrone de la fonction cyclique.

La forme générale de l'affectation d'une valeur initiale devient donc :

M <= **guarded** E_A **when** s_A **else**
E_S **when** C_{init} **else**
... -- fonction cyclique

c) Fonctions cycliques

Les expressions définissant les fonctions séquentielles cycliques sont aussi soumises à des contraintes. Alors que dans le cas des fonctions internes nous

n'introduisons aucune contrainte, une fonction d'évolution doit dépendre du point de mémorisation M et d'un seul identificateur de signal X.

Cette limitation est introduite par le fait que nous cherchons à connaître la quantité d'information apportée de l'extérieur dans le point de mémorisation pour une activation de la fonction d'évolution. Nous considérons que cette valeur est égale à la quantité d'information que l'activation de la fonction d'évolution a détruite de l'information initialement présente dans M.

Par exemple, si la fonction d'évolution est un décalage d'une position binaire, son activation induit une perte d'information correspondant à une valeur binaire ce qui correspond aussi à la quantité d'information apportée de l'extérieur.

d) Fonctions de sortie

Nous pouvons considérer un nombre quelconque de fonctions de sortie, mais nous introduisons la contrainte que chaque fonction de sortie est définie par un identificateur de signal dont l'expression ne dépend que de M. La quantité d'information observée pour une activation d'une fonction de sortie sera considérée égale à la capacité du signal de sortie affecté par une fonction de sortie.

3.2.5. Exemples

Nous considérons les exemples du registre à décalage et du compteur donnés dans le paragraphe 2.2 de ce chapitre. Pour ces exemples, nous montrons les différents constituants que nous avons isolé précédemment.

a) registre à décalage avec entrée parallèle ou entrée série / sortie série

Pour le registre à décalage, les constituants sont :

S_A et E_A n'existent pas
ψ_S = *Clock and Clock'EVENT*
C_s= *Clock*
E_S=*E*
C_{init} = *PS*
F_{evol} *(M, ES) = M div 2 + 128*IS*
F_{sortie}*(M) = M mod 2*
WA = ø
*WS = {(E, M div 2 + 128*IS)}*

b) Compteur avec initialisation à 0

Pour le compteur, les constituants sont :

S_A= *RAZ*
ψ_S = *Clock and Clock'EVENT*
C_s= *Clock*
E_A=*0*
C_{init} *n'existe pas*
F_{inter}*(M) = (M + 1) mod 256*
F_{sortie}*(M) = M*
WA = ø
WS = {((M + 1)mod 256)}

3.3. Transfert d'information à travers une structure séquentielle cyclique

L'étude du transfert d'information à travers une structure séquentielle cyclique consiste à déterminer, à partir des fonctions F_{init}, F_{evol} ou F_{inter} et F_{sortie}, le nombre et les instants d'activation de chacune de ces fonctions de manière à maximiser le flot d'information.

Pour la présentation de l'étude, nous considérons un point de mémorisation M dont la capacité d'information (définie à partir du type du signal M) est C_M.

Dans la suite, nous distinguerons le cas d'une structure cyclique avec une fonction d'évolution, d'une structure cyclique avec une fonction interne. Toutefois,

nous pouvons présenter de façon commune la solution proposée pour une valeur d'initialisation synchrone ou asynchrone.

3.3.1. Flot d'information en entrée pour une fonction d'initialisation

Considérons une fonction d'initialisation définie par un signal E_A ou E_S.

Si une telle fonction est activée, la valeur qu'elle détermine est introduite dans le point de mémorisation en remplacement de sa valeur précédente. Il n'y a donc, dans ce cas, aucune possibilité de cumuler de l'information dans le point de mémorisation M. Il s'ensuit qu'une seule activation de la valeur d'initialisation peut être envisagée.

La capacité de l'arc reliant la fonction d'initialisation à M est égale à la capacité C_M. En effet, l'arc pris isolément doit modifier tout le codage de la valeur de M, à savoir les C_M positions binaires nécessaires. La Figure 4.3 montre, pour l'activation d'une fonction d'initialisation, la capacité d'information qui peut être chargée dans la mémoire M.

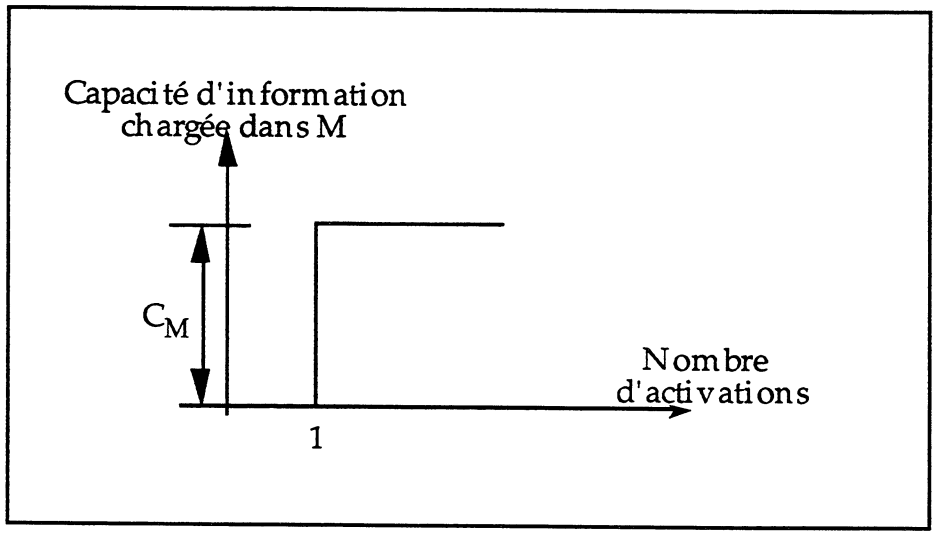

Figure 4.6. Représentation du flot d'entrée avec Finit

3.3.2. Flot d'information en entrée pour une fonction d'évolution

Dans le cas d'une structure cyclique associée à une fonction d'évolution, nous considérerons que maximiser le flot d'information consistera à maximiser l'information cumulée dans le point de mémorisation à l'aide des fonctions *Finit*

ou F_{evol}, puis à maximiser le flot en sortie de l'information stockée. Nous distinguons donc deux phases : l'une traite du flot d'information en entrée du point de mémorisation et la seconde traite de l'émission de l'information depuis le point de mémorisation vers l'extérieur. Dans la première phase, nous considérons un instant initial auquel le point de mémorisation ne contient pas d'information : la valeur initiale, connue ou non, ne sera pas prise en compte. Ensuite, pour une activation de la fonction d'initialisation ou de la fonction d'évolution, il faudra considérer uniquement la quantité d'information nouvellement introduite dans le point de mémorisation. Si cette quantité d'information est égale à la quantité maximale que le point de mémorisation peut porter, le flot maximal en entrée est atteint. Dans le cas contraire, il faut continuer à introduire de l'information dans le point de mémorisation par un certain nombre d'activations de la fonction d'évolution avec l'entrée externe considérée comme aléatoire, cela jusqu'à ce que la quantité d'information cumulée soit maximale (égale à la capacité d'information du point de mémorisation).

Il faut donc connaître, pour une activation de la fonction d'évolution, la quantité d'information détruite dans la mémoire M, cette valeur est considérée comme égale à la quantité d'information introduite. Cette propriété peut être vue comme un état stationnaire de la quantité d'information présente dans le point de mémorisation pour un nombre quelconque d'activations de la fonction d'évolution.

Pour déterminer cette quantité d'information de façon exacte, il faudrait définir des règles correctes sur la combinaison de capacités d'information pour les différents opérateurs. Nous restreignons la capacité d'information introduite par la capacité d'information du signal X, donné en paramètre à la fonction d'évolution. Ce choix donne une valeur correcte dans le cas d'opérations de décalage par exemple, dans les autres cas, elle donne toutefois une borne supérieure du flot d'information en entrée.

Soit C_X la capacité du signal X, le nombre d'activations de la fonction d'évolution est égal à $\frac{C_M}{C_X}$.

Pour représenter ce traitement, nous introduisons l'opérateur *Répéter1* dont les arguments sont :

- M, l'identificateur du point de mémorisation ;
- X, l'identificateur du signal externe de la fonction d'évolution ;
- l'expression définissant la fonction d'évolution ;
- n, le nombre d'activations de la fonction d'évolution ;

Soit $\varphi = (Répéter1\ M\ X\ \xi\ n)$, nous définissons le graphe de transfert d'information dynamique associé à φ par le graphe $G = (P, T, \alpha, \beta, Expr, \gamma_e, \gamma_p)$ où :

- $P = \{Nom(\varphi)\}$, l'ensemble des places ;
- $T = \{t_1\}$;
- $\alpha = \{e_1 = (X, t_1)\}$;
- $\beta = \{e_2 = (t_1, Nom(\varphi))\}$;
- $Exp = Expr_0[Nom(\varphi) / (Répéter1\ M\ X\ \xi n)]$;
- $\gamma_e = \{(e_1, (C_X, 1)), (e_2, (C_X, 1))\}$;
- $\gamma_p = \{(Nom(\varphi), (C_X, C_M, \frac{C_M}{C_X}, 1, 1)\}$

La Figure 4.7 montre les différentes capacités d'information cumulées dans la mémoire M pour les différents nombres d'activations de la fonction d'évolution. En considérant la perte d'information égale au gain d'information, cette capacité atteint l'état stationnaire avec la valeur C_M.

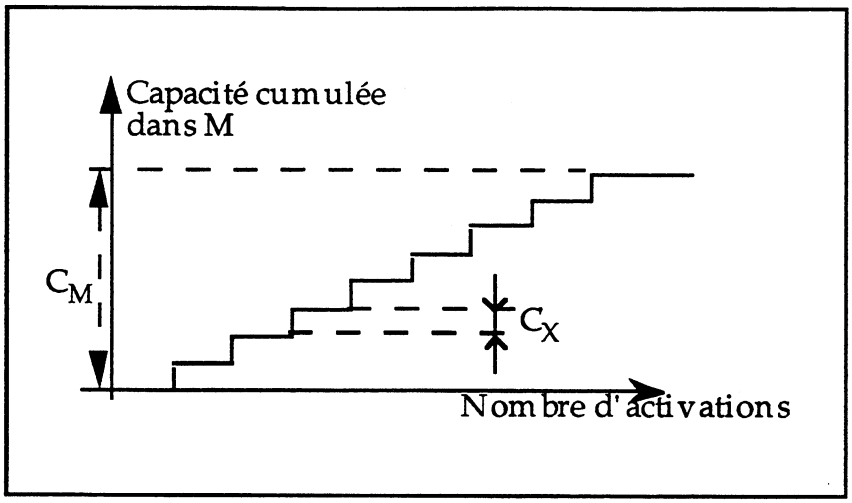

Figure 4.7. Représentation du flot d'entrée avec Fevol

Exemple

Considérons le registre à décalage décrit dans le paragraphe 2.2. La fonction d'évolution est le décalage à droite de la mémoire *M* synchronisé sur le front de l'horloge *ck*. La capacité du point de mémorisation est égale à 8 et la capacité de la donnée série *ES* est égale à 1. Il faut donc générer une place de la forme *Répéter1* avec un nombre d'activations égal à 8. Cette expression conduit donc à la définition du sous-graphe donné dans la figure 4.8.

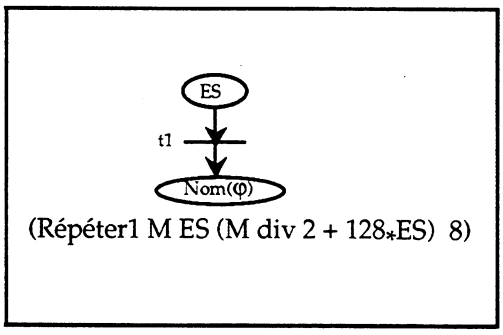

Figure 4.8. Graphe pour une fonction d'évolution.

3.3.3. Flot d'information en entrée pour une fonction interne

Une fonction interne ne dépend que du signal *M*. Une telle fonction est déterministe, elle associe à une valeur donnée de la mémoire une valeur suivante. A partir d'une certaine valeur initiale, une séquence d'activation d'une fonction interne associe une séquence de valeurs de la mémoire. Pour pouvoir introduire

une certaine quantité d'information dans la mémoire, nous devons donc envisager plusieurs cas selon que la fonction interne fait perdre de l'information ou quelle permet d'obtenir toutes les valeurs possibles de M à partir de n'importe quelle valeur initiale. Le premier cas correspond à une fonction de rotation ou de décalage avec introduction d'une valeur constante, le second cas correspond à une fonction d'incrémentation par exemple.

Pour distinguer les deux cas de figure, il nous faut calculer le domaine de valeurs possibles pour un nombre quelconque d'activations de la fonction interne. Ce calcul sera considéré sur l'intervalle $[0,2^{C_M}]$.

Lorsque toutes les valeurs du domaine de définition du point de mémorisation peuvent être atteintes, la fonction interne associe une valeur à la longueur de la séquence de contrôle (nombre d'événements). C'est une transformation d'une séquence d'événements en une valeur. Dans ce cas, il suffit de générer aléatoirement la longueur de la séquence pour obtenir une valeur aléatoire dans le point de mémorisation à partir d'une valeur initiale quelconque. Cette propriété est représentée par le symbole x du nombre d'activations de la fonction, ce qui revient à dire que la condition d'activation définie une information par le biais du nombre d'activation de la fonction interne.

Dans le cas contraire, la quantité d'information résiduelle de M après l'activation de la fonction interne est inférieure à la capacité de M, ce qui correspond à une perte d'information.

La fonction interne peut, à partir d'une valeur initiale dans la mémoire, être vue comme un constructeur d'ensemble, c'est-à-dire l'ensemble des valeurs accessibles pour les différents nombres d'activations de la fonction. Nous ne prétendons pas ici déterminer cet ensemble, et nous considérerons donc un point de vue macroscopique qui consiste à dire, que s'il n'y a pas de perte d'information dans M, alors toutes les valeurs peuvent être accédées à partir d'une valeur initiale quelconque. Donc le nombre d'activations de la fonction interne est aléatoirement choisi sur l'intervalle $[0,2^{C_M}]$. S'il y a perte d'information, soit C_r la capacité

d'information résiduelle. Le nombre de valeurs accessibles est alors défini par $\frac{C_M}{C_M - C_r}$, ce qui conduit à un nombre d'activations de la fonction interne aléatoirement choisi dans l'intervalle $[0, \frac{C_M}{C_M - C_r}]$.

Nous définissons pour une telle fonction, l'expression *Répéter2* qui prend pour arguments :

- l'expression associée à la fonction interne ;
- l'identificateur de l'horloge de la composante synchrone de la condition de garde ;
- l'identificateur du signal associé au point de mémorisation ;
- la taille de l'intervalle dans lequel choisir le nombre d'activations de la fonction interne.

Soit $\varphi = (Répéter2\ \xi\ ck\ M\ n)$, nous définissons le graphe de transfert d'information dynamique associé à φ par le graphe $G = (P, T, \alpha, \beta, Exp, \gamma_e, \gamma_p)$ où :

- $P = \{Nom(\varphi)\}$, l'ensemble des places ;
- $T = \{t_1\}$;
- $\alpha = \{e_1 = (Nom(event\ ck), t_1)\}$;
- $\beta = \{e_2 = (t_1, Nom(\varphi))\}$;
- $Exp = Expr_0[Nom(\varphi) / (Répéter2\ \xi\ Nom(event\ ck)\ n)]$;
- $\gamma_e = \{(e_1, (1,1)), (e_2, (1,1))\}$;
- $\gamma_p = \{(Nom(\varphi), (\log_2 n, C_M, x, 1, 1)\}$

Exemple

Considérons le compteur décrit dans le paragraphe 2.2. La fonction interne est l'incrémentation de la mémoire *M* synchronisée sur le front de l'horloge *ck*. La capacité du point de mémorisation est égale à 8. La fonction interne est donc la conversion du nombre d'événements successifs (sans initialisation) vers une valeurs codée sur 8 bits. Cette expression conduit donc à la définition du sous-graphe donné dans la figure 4.9.

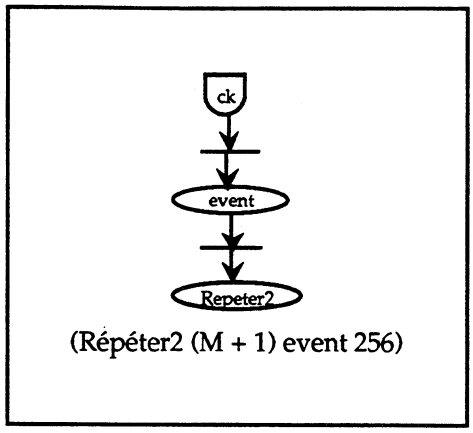

Figure 4.9. Graphe pour une fonction interne.

3.3.4. Flot d'information en sortie

Soit $F_{sortie}(M)$ une fonction de sortie, où M est le point de mémorisation. L'analyse en sortie doit permettre de déterminer le nombre d'activations de la fonction de sortie de manière à émettre le plus d'information possible du point de mémorisation vers l'extérieur de la structure séquentielle cyclique. Cette valeur doit prendre en compte la quantité d'information émise depuis M pour une activation de la fonction de sortie. Comme chaque fonction de sortie est l'affectation d'un signal par une expression qui ne dépend que de M, nous considérons que la quantité d'information émise de M via la fonction de sortie est égale à la capacité du signal de sortie S.

De plus, nous supposons que soit la fonction interne, soit la fonction d'évolution associée à la structure séquentielle cyclique permet par son activation entre deux activations de la fonction de sortie, d'émettre de l'information non encore émise par les activations précédentes. Il en résulte que, pour une fonction de sortie sur un signal S dont la capacité d'information est C_S, toute l'information présente dans la mémoire M peut être émise par $\frac{C_M}{C_S}$ de la fonction de sortie et de la fonction cyclique.

Nous définissons pour une telle fonction, l'expression *Répéter3* qui prend pour arguments :

- l'expression associée à la fonction de sortie ;

- l'identificateur du signal associé au point de mémorisation ;
- l'identificateur du signal associé à la destination de la fonction de sortie ;
- le nombre d'activations de la fonction de sortie.

Soit $\varphi = (Répéter3\ \xi\ M\ S\ n)$, nous définissons le graphe de transfert d'information dynamique associé à φ par le graphe $G = (P, T, \alpha, \beta, Exp, \gamma_e, \gamma_p)$ où :

- $P = \{Nom(\varphi), S\}$, l'ensemble des places ;
- $T = \{t_1, t_2\}$;
- $\alpha = \{e_1 = (M, t_1), e_2 = (Nom(\varphi), t_2)\}$;
- $\beta = \{e_3 = (t_1, Nom(\varphi)), e_4 = (t_2, S)\}$;
- $Exp = Expr_0[Nom(\varphi) / (Répéter3\ \xi\ M\ n)][S / (<= Nom(\varphi))]$;
- $\gamma_e = \{(e_1, (C_M, 1)), (e_2, (C_S, 1)), (e_3, (C_M, 1)), (e_4, (C_S, 1))\}$;
- $\gamma_p = \{(Nom(\varphi), (C_M, C_S, 1, \frac{C_M}{C_S}, 1), (S, (C_S, C_S, 1, 1, 1))\}$

La figure 4.10 montre, pour les différentes activations successives de la fonction de sortie et de la fonction cyclique, d'une part, la capacité d'information cumulée en sortie et, d'autre part, la capacité d'information résiduelle de l'information présente dans M avant la séquence de sortie.

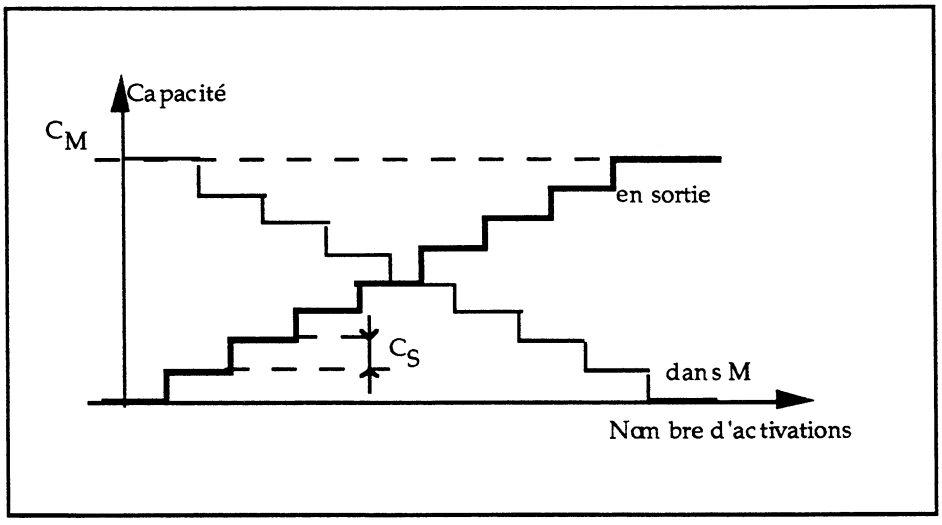

Figure 4.10. Représentation du flot de sortie

Exemple

Considérons le registre à décalage décrit dans le paragraphe 2.2. La fonction de sortie est décrite par l'expression *M mod* 2. La capacité du point de mémorisation est 8 alors que la capacité de la sortie est de 1. La répétition de l'activation de la fonction de sortie et de la fonction de décalage, permet d'émettre toute l'information de *M* en sortie. Alors que l'activation de la fonction de décalage induit une perte d'information dans *M* de capacité 1, l'activation de la fonction de sortie induit un cumul de l'information émise incrémenté de 1.

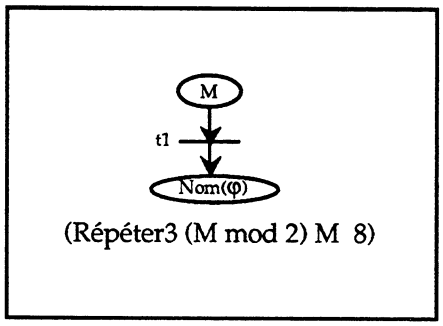

Figure 4.11. Graphe pour une fonction de sortie

3.3.5. Conclusion sur les flots d'information

Les deux phases, flot d'information en entrée et flot d'information en sortie, sont donc prises en compte de façon séparée. En entrée, nous considérons soit une fonction d'initialisation, soit une fonction d'évolution ou interne et nous déterminons le nombre de fois qu'il faut activer cette fonction. Par contre, en sortie nous considérons une fonction de sortie mais également une fonction d'évolution ou interne qui doit, éventuellement, être activée une ou plusieurs fois. Sans préciser quelle fonction d'évolution ou interne est associée à une fonction de sortie, nous considérons toujours qu'elle existe. Cette propriété nous permettrait de généraliser notre étude aux descriptions séquentielles cycliques composées de plusieurs fonctions cycliques, comme le décalage à droite et à gauche par exemple. La seule restriction est que dans le cas de fonctions internes, une seule composante

synchrone doit exister dans l'expression de garde, car, dans le cas contraire, il faudrait résoudre le problème de l'association d'une condition à une fonction.

3.3.6. Règles de simplification associées aux opérateurs *"Répéter"*:

Nous avons, dans le but de prendre en compte certaines descriptions cycliques, introduit trois nouveaux opérateurs. Ainsi, il faut compléter les règles de simplification des expressions définies dans le chapitre 3. Les règles que nous présentons ici sont donc à ajouter au système d'inférence du chapitre 3, les propriétés de terminaison étant conservées de façon évidente vu qu'il y a réduction de la taille des expressions.

Le signal extérieur d'une fonction d'évolution est une constante :

$$\overline{(Répéter1\ M\ c\ \xi\ n) \rightarrow c} \qquad (24)$$

Le signal d'horloge pour une fonction interne est une constante :

$$\overline{(Répéter2\ \xi\ c\ M\ n) \rightarrow \varepsilon} \qquad (25)$$

3.4. Exemples

3.4.1. Registre à décalage

Registre à décalage : entrée parallèle EP ou série ES, sortie série S.

```
block (H and H'EVENT)
      begin
            M <= guarded  EP when PS else
                          M div 2 + 128 * ES;
      end block
S <= M mod 2;
```

$F_{init}(EP) = EP$: c'est la fonction identité

F_{evol} *(M, ES) = M div 2 + 128 * ES*

F_{sortie} *(M) = M div 2*

Les capacités des signaux ES et S étant égales à 1, il faut donc considérer des expressions d'opérateur *répéter1* et *répéter3* avec 8 activations respectivement pour la fonction d'évolution et la fonction de sortie. La figure 4.12 montre les places, transitions et arcs du graphe de transfert d'information. Le tableau 4.7 indique les valeur des capacités d'entrée et de sortie des différentes places ainsi que les nombres d'activation d'entrée et de sortie. Initialement le nombre d'activations global d'une place est toujours égal à 1.

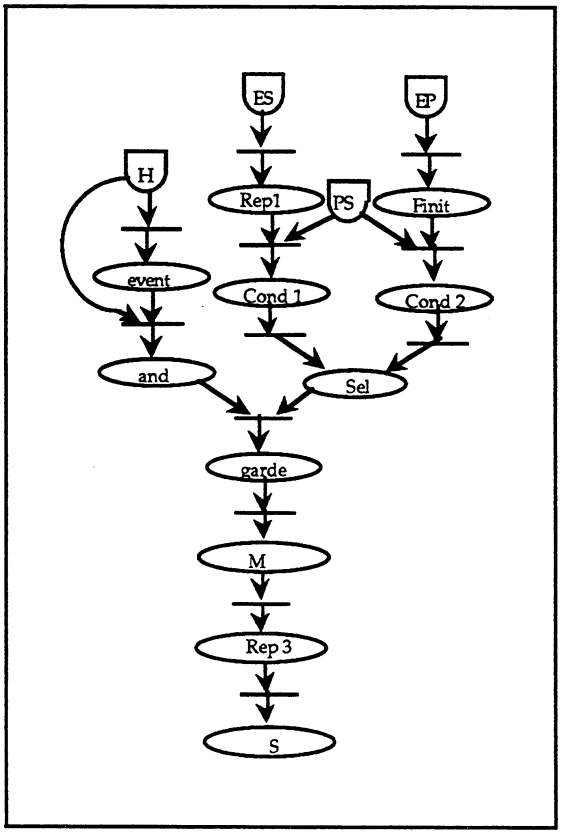

Figure 4.12. Modèle de transfert d'information

	H	ES	EP	PS	event	and	Cond1	Cond2	Rep1	Finit	Sel	garde	M	Rep3	S
C_i	1	1	8	1	1	2	9	9	1	8	8	9	8	8	1
C_o	1	1	8	1	1	1	8	8	8	8	8	8	8	1	1
n_i	1	1	1	1	1	1	1	1	8	1	1	1	1	1	1

n_0	1	1	1	1	1	1	1	1	1	1	1	1	1	8	1

Tableau 4.7. Fonction γ_p pour le registre à décalage.

3.4.2. Registre universel

Registre universel : entrée parallèle ou série à droite ou à gauche, sortie série à droite ou à gauche ou parallèle, décalage à droite ou à gauche

M : point de mémorisation sur 8 bits (0..255)

PS : booléen fonctionnement parallèle ou série

DG : booléen fonctionnement à droite ou à gauche

ESD, ESG : entrées séries droite et gauche

```
block (H and H'EVENT)
      begin
            M <= guarded        EP when PS else
                                M div 2 + 128 * ESD when DG else
                                (M*2)mod 256 + ESG;
      end block
S1 <= M mod 2;
S2 <= M div 128;
S3 <= M;
```

Description des différentes fonctions

$F_{init}(EP) = EP$

$F_{evol1}\ (M, ESD) = M\ div\ 2 + 128 * ESD$

$F_{evol2}\ (M, ESG) = (M*2) mod\ 256 + ESG$

$F_{sortie1}\ (M) = M\ div\ 2$

$F_{sortie2}\ (M) = M\ div\ 128$

$F_{sortie3}(M) = M$

Pour chacune des fonctions d'évolution et de sortie, il faut envisager 8 activations en entrée et respectivement en sortie, excepté pour la fonction $F_{sortie3}$ qui n'a à être activée qu'une seule fois. Pour cet exemple nous ne présentons que les places, transitions et arcs du graphe par la Figure 4.13.

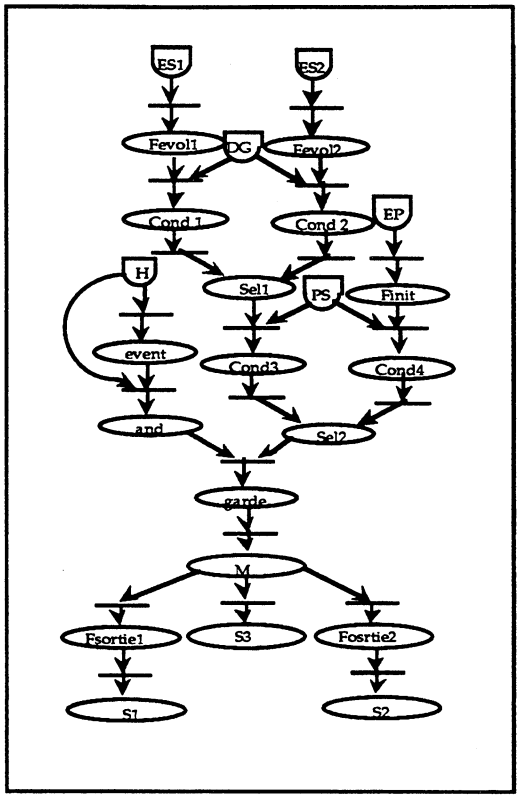

Figure 4.13. Modèle de transfert d'information

3.4.3. Compteur

Compteur avec chargement d'une valeur initiale EP et observation de toute la valeur.

```
block (H and H'EVENT)
      begin
            M <= guarded EP when LD else
                                 M+1;
      end block
S <= M;
```

Fonction associées à cette description

$F_{init}(EP) = EP$

$F_{interne}(M) = M + 1$

$F_{sortie}(M) = M$

Pour le cas d'une fonction interne nous définissons donc une place Rep2 qui est associée à l'opérateur *"Répéter2"*.

La figure 4.14 montre le graphe de transfert d'information correspondant à cette description. Le tableau 4.8 montre les capacités d'information et nombre d'activations des différentes places. La valeur 'x' indique un nombre d'activation qui est déterminé aléatoirement (séquence d'évennements).

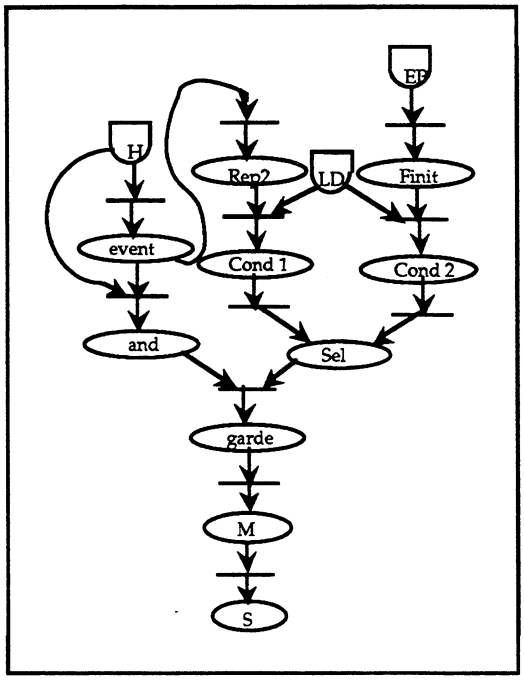

Figure 4.14. Modèle de transfert d'information

	H	LD	EP	event	and	Cond1	Cond2	Rep2	Finit	Sel	garde	M	S
C_i	1	1	8	1	2	9	9	8	8	8	9	8	8
C_o	1	1	8	1	1	8	8	8	8	8	8	8	8
n_i	1	1	1	1	1	1	1	x	1	1	1	1	1
n_o	1	1	1	1	1	1	1	1	1	1	1	1	1

Tableau 4.8. Fonction γ_p pour le compteur.

3.5. Conclusion

Dans la première partie de ce chapitre nous avons mis en évidence le problème posé par les constructions cycliques puis donné une solution pour déterminer localement des capacités d'information dynamiques. Pour pouvoir calculer les

mesures de testabilité il faut encore, pour un écoulement donné, propager les activations multiples en aval et en amont des places correspondant à des opérations *"Répéter"*.

4. Propagation des capacités dynamiques

Les capacités d'information dynamiques calculées précédemment sont des données localement définies sur les places. La propagation des éventuelles activations multiples dépend de l'écoulement que l'on considère et ne doit donc être effectuée qu'après avoir déterminé les écoulements. Dans cette partie, nous présentons tout d'abord l'algorithme de propagation avec ses restrictions puis nous allons montrer son application sur les exemples du registre à décalage et du compteur.

4.1. Algorithme de propagation

Un écoulement est la restriction du graphe de transfert d'information, il constitue également un GTID. Lors du calcul des mesures de testabilité, l faut considérer un chemin d'information. Si le chemin d'information comporte des opérateurs *Répéter*, il faut propager les activations multiples depuis cees opérateurs vers les sources et vers les puits. En effet :

- pour une place dont l'opérateur est *Répéter1* activer *ni* fois tout le sous-graphe amont,
- pour une place dont l'opérateur est *Répéter2*, il faut propager en amont le symbole x (longueur de séquene aléatoire),
- pour une place Répeter3, il faut propager en aval *ni* le nombre d'activations de la sortie.

En résultat, tous les arc et totes les places de l'écoulement ont un nombre d'activation qui permet un transfert d'information cohérent.

Nous décrivons ci-dessous la propagation des activations multiples à l'intérieur d'un écoulement à l'aide d'un système de règles d'inférence. Ces règles d'inférence

seront appliquées avec une stratégie et un certain nombre d'hypothèses que nous décrirons ultérieurement. Nous considérons initialement le graphe de l'écoulement avec les fonctions γ_e et γ_p définies localement. Nous définissons ensuite les fonctions X_e et X_p, initialement égales à γ_e et γ_p, qui servent à déterminer les valeurs en cours de propagation. Ces deux fonction contiennent finalement les capacités dynamiques résultantes.

4.1.1. Règles d'inférence

a) Prologue

Soit $G=(P,T,\alpha,\beta,Expr,\gamma_e,\gamma_p)$ le GTID associé à un écoulement. Construire $X_e=\gamma_e$ et $X_p=\gamma_p$.

b) Règles initiales

Pour une expression *"Répéter1"* :

$$\frac{\exists p\in P,\gamma_p(p)=(C_i,C_o,n_i,1,1),n_i>1,A_p^-(p)=\{e\},X_e(e)=(C_i,1)}{G,X_e\rightarrow X_e[e/(C_i,n_i)]} \quad (1)$$

Pour une expression *"Répéter2"* :

$$\frac{\exists p\in P,\gamma_p(p)=(C_i,C_o,1,n_o,1),n_o>1,A_p^+(p)=\{e\},X_e(e)=(C_o,1)}{G,X_e\rightarrow X_e[e/(C_o,n_o)]} \quad (2)$$

Pour une expression *"Répéter3"* :

$$\frac{\exists p\in P,\gamma_p(p)=(C_i,C_o,x,1,1),A_p^-(p)=\{e\},X_e(e)=(1,1)}{G,X_e\rightarrow X_e[e/(C_i,x)]} \quad (3)$$

c) Propagation amont pour les séquences d'événements

D'une place vers un arc prédécesseur :

$$\frac{\exists p\in P,X_p(p)=(C_i,C_o,1,1,x),\exists e\in A_p^-(p),X_e(e)=(C_e(e),1)}{G,X_e\rightarrow X_e[e/(C_i,x)]} \quad (4)$$

D'un arc successeur d'une transition vers ses arcs prédécesseurs :

$$\frac{\exists e^* = (t,p) \in \beta, X_e(e^*) = (C_{e^*}, x), \gamma_e(e^*) = (C_e(e^*), 1), \exists e \in A_t^-(t), \gamma_e(e) = (C_e(e), 1)}{G, X_e \rightarrow X_e[e / (\frac{C_{e^*}}{C_e(e^*)} * C_e(e), x)]} \quad (5)$$

D'un arc vers une place prédécesseur dont l'opérateur n'est pas *"Répéter"* :

$$\frac{\exists e^* = (p,t) \in \alpha, X_e(e^*) = (C_{e^*}, x), \gamma_e(e^*) = (C_e(e^*), 1), X_p(p) = (C_i, C_o, 1, 1, 1)}{G, X_e \rightarrow X_p[p / (\frac{C_{e^*}}{C_e(e^*)} * C_i, \frac{C_{e^*}}{C_e(e^*)} * C_o, 1, 1, x)]} \quad (6)$$

D'un arc vers une place prédécesseur dont l'opérateur est *"Répéter3"*, seul cas envisagé :

$$\frac{\exists e^* = (p,t) \in \alpha, X_e(e^*) = (C_{e^*}, x), \gamma_e(e^*) = (C_e(e^*), 1), X_p(p) = (C_i, C_o, 1, n_o, 1)}{G, X_e \rightarrow X_p[p / (C_i, C_o, 1, n_o, \frac{C_{e^*}}{C_i})]} \quad (7)$$

d) Propagation amont pour des activations multiples

D'une place vers un arc prédécesseur :

$$\frac{\exists p \in P, X_p(p) = (C_i, C_o, n_i, 1, n), n_i * n > 1, \exists e \in A_p^-(p), X_e(e) = (C_i, 1)}{G, X_e \rightarrow X_e[e / (C_i, n * n_i)]} \quad (8)$$

D'un arc successeur d'une transition vers ses arcs prédécesseurs :

$$\frac{\exists e^* = (t,p) \in \beta, X_e(e^*) = (C_{e^*}, n^*), \exists e \in A_t^-(t), \gamma_e(e) = (C_e(e), 1)}{G, X_e \rightarrow X_e[e / (C_e(e), n^*)]} \quad (9)$$

D'un arc vers une place prédécesseur de d'opérateur différent de *"Répéter3"* :

$$\frac{\exists e^* = (p,t) \in \alpha, X_e(e^*) = (C_{e^*}, n^*), X_p(p) = (C_i, C_o, n_i, n_o, 1), n_o < n^*}{G, X_e \rightarrow X_p[p / (C_i, C_o, n_i, n_o, \frac{n^*}{n_o})]} \quad (10)$$

e) Propagation aval pour des activations multiples

D'une place vers un arc successeur :

$$\frac{\exists p \in P, X_p(p) = (C_i, C_o, n_i, n_o, n), n * n_o > 1, \exists e \in A_p^+(p), X_e(e) = (C_o, 1)}{G, X_e \rightarrow X_e[e / (C_o, n * n_o)]} \quad (11)$$

D'un arc prédécesseur d'une transition vers ses autres arcs prédécesseurs et vers son arc successeur :

$$\frac{\exists e^* = (p,t) \in \alpha, X_e(e^*) = (C_{e^*}, n^*), \exists e \in (A_t^-(t) \cup A_t^-(t)), \gamma_e(e) = (C_e(e), 1)}{G, X_e \rightarrow X_e[e / (C_e(e), n^*)]} \quad (12)$$

D'un arc successeur d'une transition la place successeur de l'arc, place dont l'opérateur n'est pas *"Répéter3"* :

$$\frac{\exists e^* = (t,p) \in \beta, X_e(e^*) = (C_{e^*}, n^*), X_p(p) = (C_i, C_o, n_i, n_o, 1), n_i < n^*}{G, X_e \rightarrow X_p[p / (C_i, C_o, n_i, n_o, \frac{n^*}{n_i})]} \quad (13)$$

f) Epilogue

En épilogue de l'algorithme, les fonctions γ_e et γ_p du graphe G sont remplacées par les fonctions X_e et X_p. A partir des capacités dynamiques ainsi calculées, les données introduites pour le calcul des mesures de testabilité sont des capacités d'information que nous appellerons *"pseudo-statiques"*, et dont la valeur est calculée pour les arcs et les transitions de la façon suivante :

- pour un arc dont la capacité est C et le nombre d'activations $n \neq x$, la capacité pseudo-statique est $C_{ps} = C * n$;
- pour un arc dont la capacité est C et le nombre d'activations x, la capacité pseudo-statique est $C_{ps} = C$;
- pour une place dont aucune valeur n'est égale à x les capacités pseudo-statique d'entrée et de sortie de la place sont respectivement $C_{psi} = C_i * n_i * n$ et $C_{pso} = C_o * n_o * n$;

- pour une place dont le nombre d'activations global est x, les pseudo-capacités d'entrée et de sortie sont respectivement $C_{psi} = C_i$ et $C_{pso} = C_o$;

4.1.2. Contrôle et stratégie d'application des règles

Dans la propagation des activations multiples, se pose le problème des collisions, c'est à dire la propagation de deux nombres d'activations différents sur une même partie du graphe, mais à partir d'origines initiales différentes. Ceci impose de considérer deux problèmes :

- la cohérence du nombre d'activations de places dans un sous-graphe ;
- le choix d'un nombre d'activations d'un sous-graphe.

Pour résoudre le premier problème, nous proposons d'effectuer, à partir d'une origine donnée, toute la propagation du nombre d'activations en amont et en aval. Ceci induit une priorité minimale des règles correspondant à l'initialisation d'une propagation (règles 1, 2, 3) par rapport aux autres règles.

Le second problème, nécessite de mettre en place soit une stratégie de choix d'une première origine qui impose les valeur au sous-graphe qu'elle aurait en commun avec d'autres origines, soit des règles de modification des valeurs selon une certaine stratégie. Le but que nous cherchons à atteindre est que les nombres d'activations et capacités statiques ainsi définis correspondent à la valeur maximale de la capacité dynamique utilisée. Toutefois, il faut également tenir compte de l'éventuelle présence de cycles dans le graphe qui pourraient entraîner la non terminaison du processus de propagation dans le cas d'une stratégie où l'on donne toujours la possibilité d'augmenter la valeur de la capacité dynamique d'un élément du graphe. Les règles présentées ci-dessus sont associées à une stratégie où l'on commence par une valeur initiale correspondant à la plus grande capacité dynamique. Seuls les éléments dont le nombre d'activations est égal à 1 sont traités. Ainsi, nous garantissons qu'un élément ne sera modifié qu'une seule fois par la propagation des activations multiples.

Ainsi la stratégie consiste à choisir une règle initiale telle que le produit de la capacité par le nombre d'activations soit maximal, en donnant priorité aux places dont l'opérateur est du type *"Répéter3"*. Si plusieurs places remplissent la condition de maximalité et s'il existe une relation de dépendance entre elles, nous choisissons celle qui est le plus en aval. Si aucune dépendance n'existe, nous pouvons en choisir une quelconque.

Cet ensemble de règles d'inférence définit la propagation des activations multiples dans un écoulement. Le résultat est un graphe de transfert d'information dynamique dont les arcs et les places ont des capacités d'information que nous pouvons définir statiquement comme étant le produit du nombre d'activations par la capacité statique considérée. Ainsi, ces valeurs constituent des données qui, d'une part, peuvent être prises en compte pour le calcul des mesures de testabilité avec la méthode classique et, d'autre part, décrivent des flots d'information dynamiques.

Nous supposons de plus dans cette étude que la propagation d'activations de la forme *"x"* ne se produit que sur des places dont la capacité de l'arc sortant est de 1. Ceci induit que, soit l'horloge n'est déterminée que par une expression booléenne sur des signaux booléens, soit la propagation sous la forme *"x"* se change en une propagation avec un nombre d'activations fixé.

4.2. Exemples

4.2.1. Exemple du registre à décalage

Considérons l'exemple du registre à décalage donné dans le paragraphe 3 de ce chapitre. Le graphe de transfert d'information comporte deux écoulements représentés dans les figures 4.15 et 4.16.

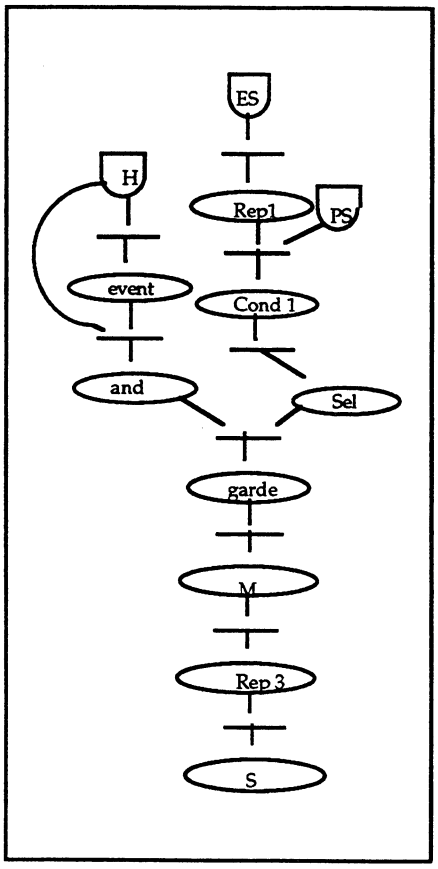

Figure 4.15. Ecoulement ECL1.

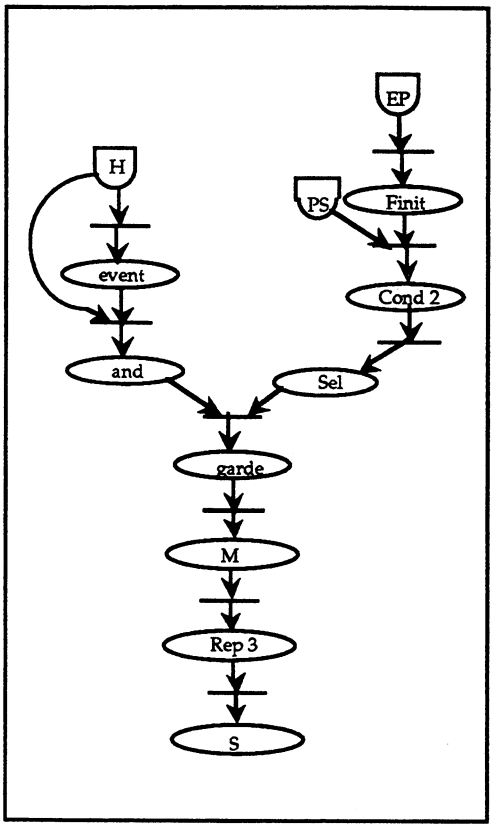

Figure 4.16. Ecoulement ECL2

	H	ES	PS	event	and	Cond1	Rep1	Sel	garde	M	Rep3	S
C_i	1	1	1	1	2	9	1	8	9	8	8	1
C_o	1	1	1	1	1	8	8	8	8	8	1	1
n_i	1	1	1	1	1	1	8	1	1	1	1	1
n_o	1	1	1	1	1	1	1	1	1	1	8	1
n	1	8	1	1	1	1	1	1	1	1	1	8
C_{psi}	1	8	1	1	2	9	8	8	9	8	8	8
C_{psi}	1	8	1	1	1	8	8	8	8	8	8	8

Tableau 4.9. Fonction X_p l'écoulement ECL1

	H	EP	PS	event	and	Cond2	Finit	Sel	garde	M	Rep3	S
C_i	1	8	1	1	2	9	8	8	9	8	8	1
C_o	1	8	1	1	1	8	8	8	8	8	1	1
n_i	1	1	1	1	1	1	1	1	1	1	1	1
n_o	1	1	1	1	1	1	1	1	1	1	8	1
n	1	1	1	1	1	1	1	1	1	1	1	8
C_{psi}	1	8	1	1	2	9	8	8	9	8	8	8
C_{psi}	1	8	1	1	1	8	8	8	8	8	8	8

Tableau 4.10. Fonction X_p de l'écoulement ECL2.

4.2.2. Exemple du compteur

Considérons l'exemple du compteur donné dans le paragraphe 3 de ce chapitre. Le graphe de transfert d'information comporte deux écoulements représentés dans les figures 4.17 et 4.18.

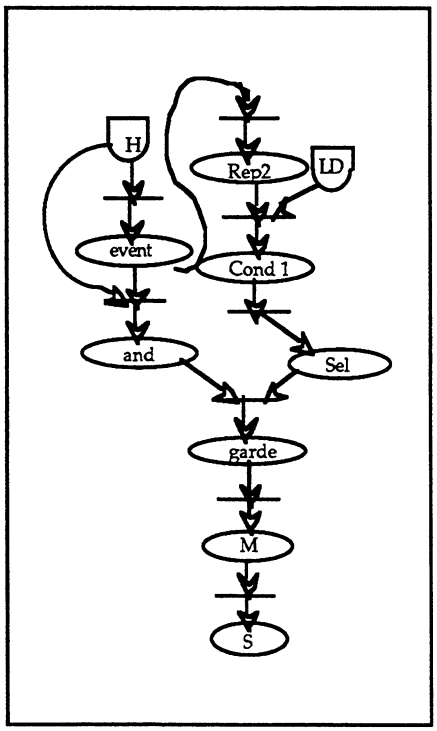

Figure 4.17. Ecoulement ECL1.

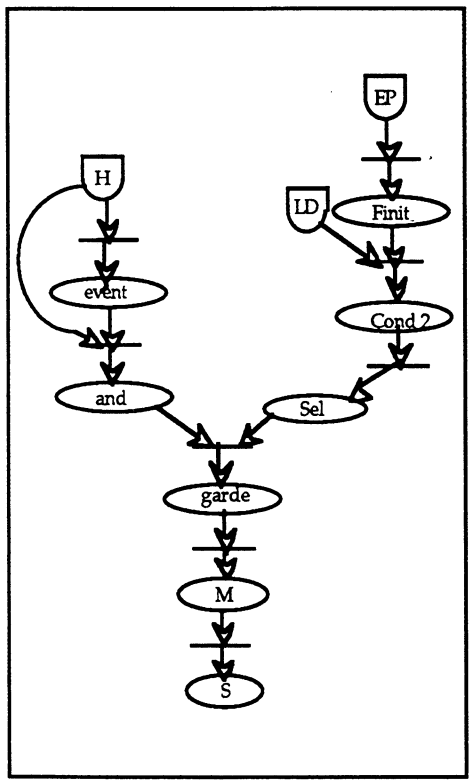

Figure 4.18. Ecoulement ECL2

	H	LD	event	and	Cond1	Rep2	Sel	garde	M	S
C_i	8	1	8	2	9	8	8	9	8	8
C_o	8	1	8	1	8	8	8	8	8	8
n_i	1	1	1	1	1	x	1	1	1	1
n_o	1	1	1	1	1	1	1	1	1	1
n	x	1	x	1	1	1	1	1	1	1

Tableau 4.11. Fonction X_p pour l'écoulement ECL1 du compteur.

5. Conclusion

Dans ce chapitre nous avons présenté tout d'abord la méthode de calcul des mesures de testabilité basées sur la quantité d'information. Pour pouvoir calculer ces mesures, il est nécessaire de définir certains paramètres associés aux places et arcs du graphe de transfert d'information, à savoir les capacités d'information. Alors qu'une capacité d'information statique est directement déduite de la description VHDL en prenant en compte le type des signaux et des différentes expressions, il s'avère nécessaire d'effectuer un traitement particulier pour certaines constructions qui font intervenir des cycles.

C'est pourquoi nous avons été amené à définir une nouvelle notion de débit d'information, ce qui signifie que plusieurs activations d'un même arc permettent de cumuler des capacités d'information statiques. Ainsi, nous avons présenté l'obtention de ces paramètres localement sur les places, avec l'introduction de nouveaux opérateurs ; cette étude a été effectuée sur un domaine restreint de structures séquentielles cycliques.

Nous présentons ensuite comment prendre en compte les capacités dynamiques à l'intérieur d'un écoulement par la propagation des activations multiples de certaines fonctions. Cette propagation est restreinte à une classe d'écoulements qui n'introduisent pas de conflits de propagation entre différentes fonctions cycliques.

Chapitre 5

Compilation d'une Description VHDL Structurelle

1. Introduction

Nous envisageons, dans ce chapitre, la compilation de descriptions hiérarchiques. La hiérarchie du circuit est définie par une description structurelle qui correspond à un réseau d'interconnexion de cellules. Chaque cellule est elle-même décrite par un programme VHDL, soit structurel lui aussi, soit de type comportemental.

Pour la compilation de descriptions VHDL, nous considérons que toutes les cellules ont été compilées au préalable et leur graphe de transfert d'information chargé dans la bibliothèque. La tâche de compilation consiste donc à étudier le réseau d'interconnexion et ainsi d'en déduire aussi bien les graphes de transfert d'information des différentes cellules et leur concaténation afin d'obtenir le graphe de transfert d'information global du circuit.

Comme nous l'avons montré précédemment, la bibliothèque SATAN contient les graphes de transfert d'information des descriptions comportementales ou structurelles d'entités VHDL. Les descriptions de la bibliothèque sont composées de deux éléments :

- la définition des entrées et sorties fonctionnelles, à savoir l'association entre les sources (respectivement les puits) et un ensemble de signaux d'entrée (respectivement de sortie).
- la définition du graphe de transfert d'information proprement dit sous la forme d'un graphe dynamique tel qu'il a été défini dans le chapitre 4.

Considérons une description structurelle d'une entité VHDL correspondant donc à une description hiérarchique. Une telle description est un réseau d'interconnexion entre des composants, les signaux étant nommés, soit dans la déclaration d'entité si ils sont d'entrée/sortie, soit dans la définition de l'architecture si ces signaux sont locaux. Nous distinguons, pour chaque composant constituant une entité, d'une part, le nom de l'entité qui définit la forme générique du composant, et, d'autre part, le nom du composant qui permet d'identifier les instantiations des entités.

Le nom de l'entité d'un composant est, par construction, le même que le nom de l'élément de la bibliothèque de graphe de testabilité.

La compilation d'une description VHDL structurelle va se décomposer en plusieurs étapes :

- pour chaque instantiation d'un composant, extraire de la bibliothèque le type SATAN du composant. Il en résulte autant de couples (entrées/sorties fonctionnelles, graphe de transfert d'information) qu'il y a d'instantiations de composants.
- simplifier éventuellement les graphes de transfert d'information vis-à-vis des signaux constants passés en paramètre. Cette étape reprend une adaptation aux graphes de transfert d'information du système de réécriture présenté dans le chapitre 3. Il en résulte, pour chaque instantiation, un graphe de transfert d'information qui, d'une part, respecte la contrainte qui interdit de décrire des places ne fournissant pas d'information et, d'autre part, ne décrit que les fonctionnalités réellement utilisées d'un composant. Nous appelons cette étape procédé de réduction et le graphe résultant le graphe spécifique partiel.
- analyser le réseau d'interconnexion, décrit par les signaux, entre composants et entrées/sorties. Cette étape consiste à déterminer comment l'information peut être amenée sur une entrée fonctionnelle à partir des entrées primaires ou des sorties fonctionnelles d'autres composants. Elle implique donc une modification du graphe au niveau des sources et des puits des graphes spécifiques partiels. Finalement un bon nommage des places va définir la concaténation entre les différents graphes spécifiques partiels. Le résultat est le graphe de transfert d'information global de la description structurelle.

La figure 5.1 montre l'organisation de la compilation de descriptions structurelles.

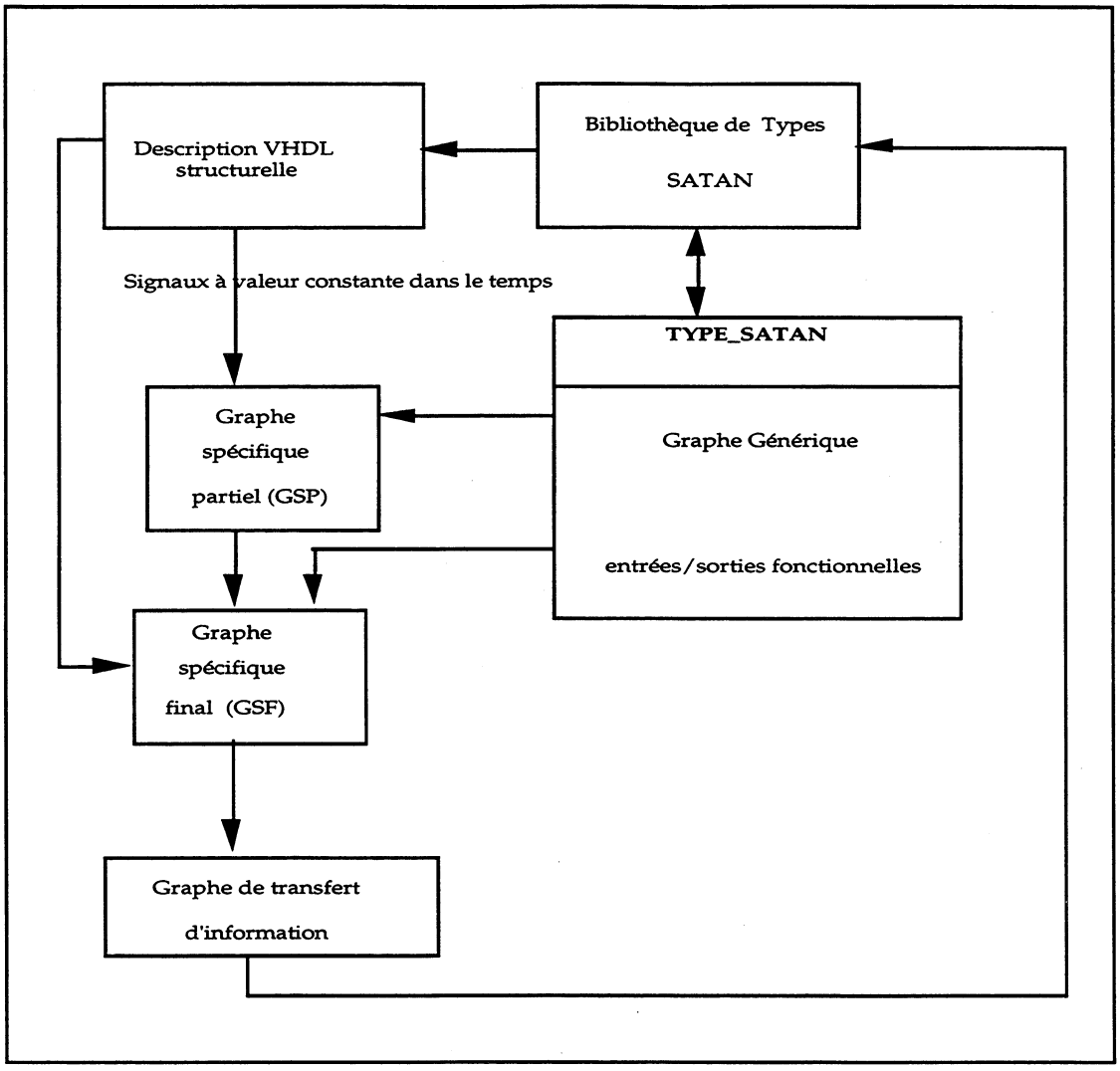

Figure 5.1 : Schéma de construction du graphe de testabilité

Dans une première partie, nous présentons le procédé de réduction qui consititue le moyen mis en oeuvre pour simplifier les graphes par rapport aux signaux constants. Puis nous montrons l'obtention du graphe spécifique final à partir des graphes simplifiés et du réseau d'interconnexion entre les différentes cellules. Nous y analysons le flot d'information sur une entrée fonctionnelle et, ensuite, le flot d'information sur une sortie fonctionnelle. Finalement, un exemple simple illustre la génération du graphe spécifique final.

2. Procédé de réduction

Lors de l'intégration d'une entité dans une description structurelle, certaines des fonctionnalités intrinsèques de la cellule peuvent éventuellement être inutilisables. Le

procédé de réduction est destiné à modifier le graphe de transfert d'information d'une cellule de telle sorte que seules les fonctionnalités réellement utilisées soient décrites. De plus, le procédé de réduction doit permettre de définir un GTI qui soit conforme à la notion de quantité d'information, les places constantes devant être supprimées.

Que ce soit au niveau de la réduction des fonctionnalités d'une cellule ou au niveau de données constantes, celles-ci sont provoquées par la connexion d'une entrée de la cellule à un signal de valeur constante dans le temps.

Le procédé de réduction va donc effectuer des transformations sur le graphe et ceci à deux niveaux : au niveau des expressions associées aux places et au niveau du graphe proprement dit. Les transformations seront décrites par un système d'inférence.

2.1. Transformations sur le Graphe.

Les transformations sur le graphe concernent la suppression des places dont l'expression est constante ainsi que la fermeture transitive des places et transitions supprimées dans le graphe. Les trois règles suivantes décrivent ces transformations.

Une place dont l'expression est constante, doit être supprimée :

$$\frac{\exists p \in P, Exp(p) = c}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\},T,\alpha \setminus \{(p,t)\},\beta \setminus \{(t,p)\}, Exp[p/\varepsilon][p'/subst(p,c,Exp_G(p'))]_{p' \in P_G^+(p)})} \quad (A)$$

Une place qui n'est pas un puits et qui n'a pas d'arc successeur est supprimée :

$$\frac{\exists p \in P, (\Gamma^+_{p,G}(p) = \varnothing) \wedge (p \notin R(G))}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\},T,\alpha,\beta \setminus \{(t,p)\},Exp[p/\varepsilon])} \quad (B)$$

Une transition qui n'a pas de place prédécesseur ou successeur est supprimée :

$$\begin{cases} \dfrac{\exists t \in T, (\Gamma^+_{t,G}(t) = \varnothing)}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P,T \setminus \{t\},\alpha \setminus \{(p,t)\},\beta,Exp)} \\ \dfrac{\exists t \in T, (\Gamma^-_{t,G}(t) = \varnothing)}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P,T \setminus \{t\},\alpha,\beta \setminus \{(t,p)\},Exp)} \end{cases} \quad (C)$$

2.2. Transformation sur les expressions du graphe.

Concernant les expressions du graphe, il suffit d'extraire du système de simplification, présenté dans le chapitre 3, les règles pouvant s'appliquer. Nous ne considérons ni les règles 15, 16 et 17 car elles concernent l'opérateur *cond* qui n'est pas directement en bijection avec une place, ni les règles 13 et 14 car il y a introduction d'un *"non"* et donc augmentation de la taille du graphe résultant.

Les règles du chapitre 3 utilisées dans le procédé de réduction sont modifiées de manière à prendre en compte la structure de graphe. Ainsi, les règles du chapitre 3 sont transformées en nouvelles règles en suivant le principe ci-dessous :

$$\frac{Q(\psi)}{\psi \rightarrow f(\psi)} \quad est \quad transformée \quad en \quad \frac{\exists p \in P, Q(Exp(p))}{(P,T,\alpha,\beta,Exp) \rightarrow (P,T,\alpha,\beta,Exp[p/f(Exp(p))])}$$

2.3. Prise en compte de l'opérateur *cond*

Une expression d'opérateur *cond* conduit, par traduction, à des places dont les opérateurs peuvent être *"si alors sinon"* ou *"sélection"*. Nous allons donc présenter les règles pour ces opérateurs. Ces règles conduisent à une réduction du graphe comme suit : si la condition est valeur constante, alors la place qui correspond à la valeur toujours transférée remplace le sous-graphe généré pour l'expression conditionnelle initiale.

- La condition d'une expression conditionnelle devient une constante :

$$\frac{\exists p \in P, (Expr(p) = si\ vrai\ alors\ p_1\ sinon\ \varepsilon), \Gamma^-_{p,G}(p) = t^-, \Gamma^+_{p,G}(p) = t^+, P^+_G(p) = q}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T \setminus \{t^-\}, (\alpha \cup \{(p_1,t^+)\}) \setminus \{(p_1,t^-),(p,t^+)\}, \beta, Expr[p/\perp][q/subst(p,p1,Expr(q))])} \quad (26)$$

$$\frac{\exists p \in P, (Expr(p) = si\ vrai\ alors\ \varepsilon\ sinon\ p_1), \Gamma^+_{p,G}(p) = t^+, P^+_G(p) = q}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T \setminus \{t^-\}, (\alpha \setminus \{(p,t^+)\}, \beta, Expr[p/\perp][q/subst(p,\varepsilon,Expr(q))])} \quad (27)$$

$$\frac{\exists p \in P, (Expr(p) = si\ faux\ alors\ \varepsilon\ sinon\ p_1), \Gamma^-_{p,G}(p) = t^-, \Gamma^+_{p,G}(p) = t^+, P^+_G(p) = q}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T \setminus \{t^-\}, (\alpha \cup \{(p_1,t^+)\}) \setminus \{(p_1,t^-),(p,t^+)\}, \beta, Expr[p/\perp][q/subst(p,p1,Expr(q))])} \quad (28)$$

$$\frac{\exists p \in P, (Expr(p) = si\ faux\ alors\ p_1\ sinon\ \varepsilon), \Gamma^+_{p,G}(p) = t^+, P^+_G(p) = q}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T, \alpha \setminus \{(p,t^+)\}, \beta, Expr[p/\perp][q/subst(p,\varepsilon,Expr(q))])} \quad (29)$$

- L'un des arguments de l'opérateur *sélection* n'est jamais actif :

$$\frac{\exists p \in P, (Expr(p) = selection\ \varepsilon\ p_1) \vee (Expr(p) = selection\ p_1\ \varepsilon), \Gamma^+_{p,G}(p_1) = t_1^+}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T, (\alpha \cup \{(p_1,t^+)\}_{t^+ \in \Gamma^+_{p,G}(p)}) \setminus \{(p,t^+)_{t^+ \in \Gamma^+_{p,G}(p)}, (p_1,t_1^+)\}, \beta, Exp[p/\perp]([q/subst(p,p_1,Expr(q))])_{q \in P^+_G(p)})} \quad (30)$$

- Une expression de donnée d'un opérateur conditionnel devient constante : :

$$\frac{\exists p \in P, (Expr(p) = si\ p_1\ alors\ c\ sinon\ c)}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P,T,\alpha,\beta,Expr[p/c])} \quad (31)$$

$$\frac{\exists p \in P, (Expr(p) = si\ p_1\ alors\ c_1\ sinon\ \varepsilon), (\{t^+\} = \Gamma^+_{p,G}(p)), (\{p^+\} = P^+_G(p)), (Expr(p^+) = selection\ p\ c_2)}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T, \alpha \cup \{(p_1,t^+)\} \setminus \{(p,t^+)\}, \beta, Expr[p/\perp][p^+/si\ p_1\ alors\ c_1\ sinon\ c_2])} \quad (32)$$

$$\frac{\exists p \in P, (Expr(p) = si\ p_1\ alors\ \varepsilon\ sinon\ c)}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P,T,\alpha,\beta,Expr[p/c])} \quad (33)$$

$$\frac{\exists p \in P, (Expr(p) = si\ p_1\ alors\ \varepsilon\ sinon\ c_2), (\{t^+\} = \Gamma^+_{p,G}(p)), (\{p^+\} = P^+_G(p)), (Expr(p^+) = selection\ c_1\ p)}{G = (P,T,\alpha,\beta,Expr) \rightarrow (P \setminus \{p\}, T, \alpha \cup \{(p_1,t^+)\} \setminus \{(p,t^+)\}, \beta, Expr[p/\perp][p^+/si\ p_1\ alors\ c_1\ sinon\ c_2])} \quad (34)$$

2.4. Exemple

Considérons la description VHDL d'un multiplexeur suivante :

```
entity mux is
        port ( X, Y : in integer ;
                C : in boolean ;
                Z : out integer);
end mux;
architecture DataFlow of mux is
begin
        Z <= X when C else Y;
end;
```

Cette description conduit au graphe de transfert d'information générique donnée dans la figure 5.2. Si, lors de l'utilisation de ce multiplexeur dans une architecture structurelle d'un circuit, l'entrée de contrôle *C* est reliée à la valeur constante *'true'*, nous obtenons le graphe initial donné dans la figure 5.3 (*C devient une place constante).*

L'application des règles du procédé de réduction conduit aux différentes réductions :

- application de la règle *(A)* pour la place *C* qui est constante. Le résultat de cette première étape de simplification donne le graphe de la figure 5.4 ;
- ensuite, il est possible d'appliquer deux règles d'inférence : la règle *(26)* sur la place NP_1, ou la règle *(27)* sur la place NP_2. L'application de la règle *(26)* conduit au graphe de la figure 5.5, puis l'application de la règle *(27)* au graphe de la figure 5.6 ;
- Finalement, seule la règle *(30)* est applicable et donne le graphe spécifique partiel du multiplexeur de la figure 5.7.

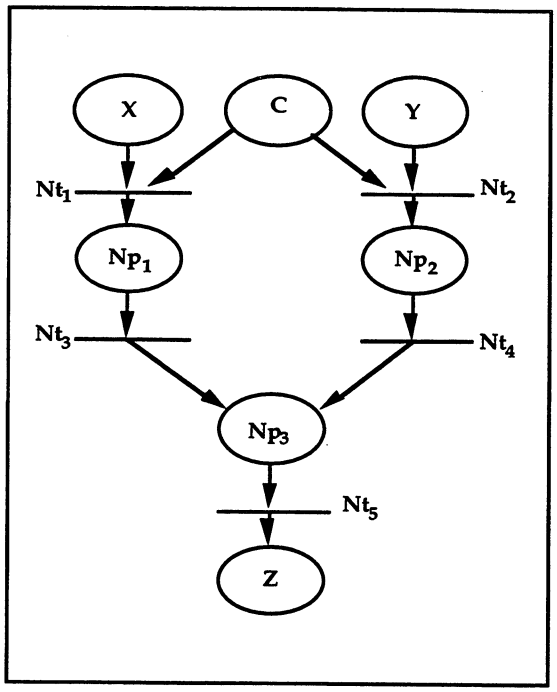

Figure 5.2. Graphe générique du multiplexeur.

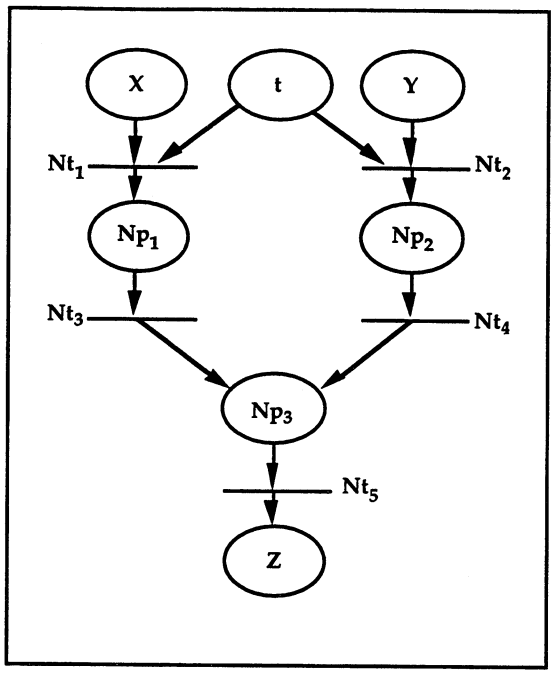

Figure 5.3. Graphe initial du multiplexeur.

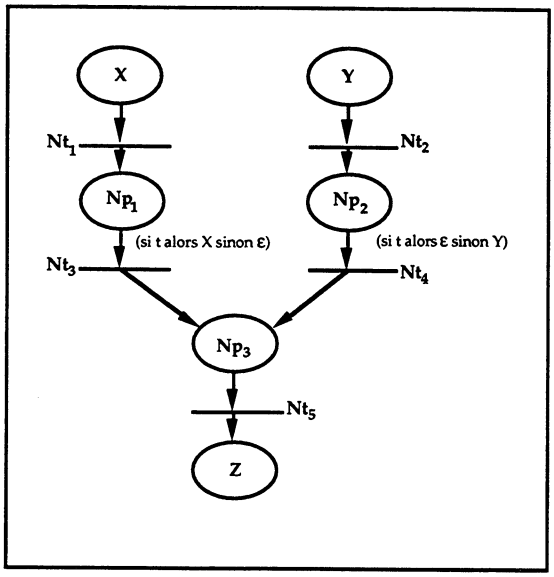

Figure 5.4. Après règle (A).

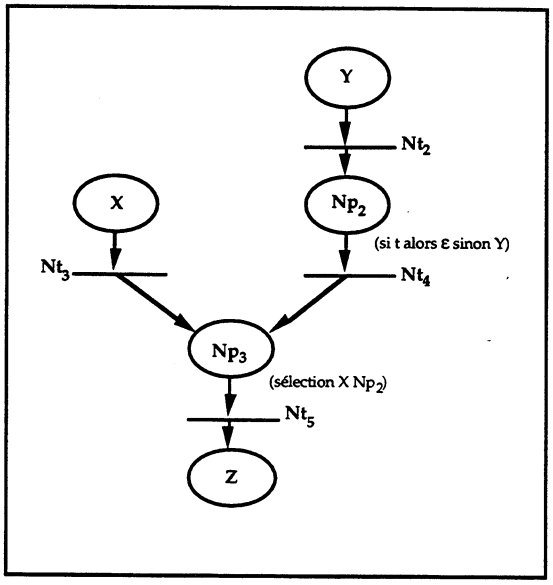

Figure 5.5. Après règle (26).

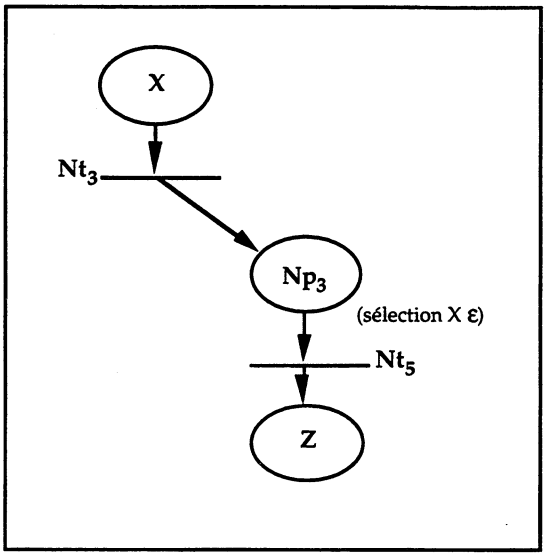

Figure 5.6. Après règle (27).

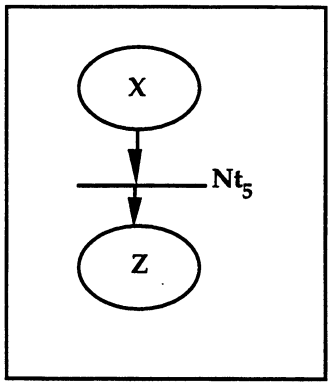

Figure 5.7. Après règle (30).

3. Génération du graphe spécifique final

L'analyse du réseau d'interconnexion d'une description va permettre d'établir les liens entre les graphes spécifiques partiels ; ceci se traduira par l'obtention d'un graphe, appelé graphe spécifique final, pour chacun des composants ; la concaténation de l'ensemble de ces graphes fournit alors le graphe de transfert d'information global de la description structurelle.

La prise en compte de l'intégration d'un composant dans le réseau permet de mettre en évidence :

- les diverses origines d'information sur une entrée fonctionnelle,
- les diverses destinations de l'information issue d'une sortie fonctionnelle.

3.1. Analyse des entrées fonctionnelles

3.1.1. Introduction et exemple

L'information générée sur une entrée fonctionnelle est une information, partielle ou non, en provenance soit d'autres composants, soit des signaux externes d'entrée de la description à compiler.

L'objectif est de spécifier l'information générée sur chaque entrée fonctionnelle d'un composant relativement aux origines de cette information.

Le traitement de la multiplicité des origines possibles de l'information sur une entrée fonctionnelle est surtout abordé dans le cas où l'entrée fonctionnelle décrit une source associée à plusieurs signaux d'entrée. En effet, dans ce cas, il peut être nécessaire de déterminer des combinaisons d'origines de l'information. Pour une entrée fonctionnelle formée de plusieurs signaux, il s'agit de déterminer toutes les combinaisons d'origines (éventuellement partielles) qui permettent d'amener de une source d'information et une seule à chacune des composantes de l'entrée fonctionnelle.

Exemple introductif

L'exemple de la figure 5.8 met en évidence les problèmes à résoudre. Nous y introduisons certaines notions définies dans la suite de cette section.

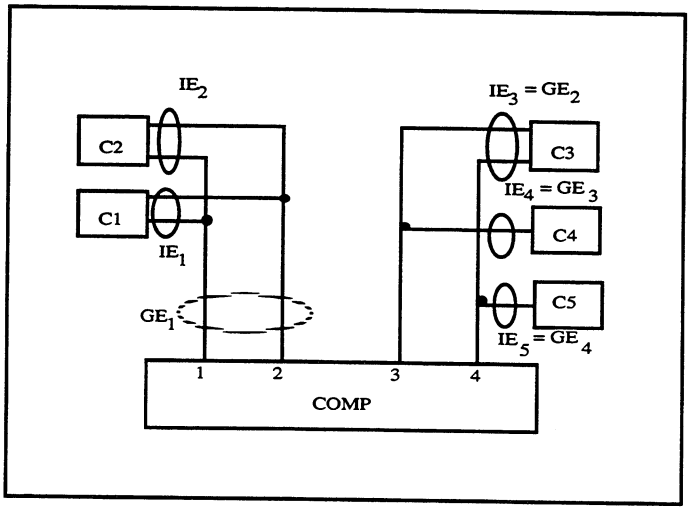

Figure 5.8. Schéma sur une entrée fonctionnelle

L'objectif est d'analyser comment on peut générer une information sur les signaux d'entrée *{1,2,3,4}* du composant *COMP* à partir des composants origines *{C1,...C5}*.

Il s'agit d'abord de reconnaître les lignes d'information $Ci \rightarrow COMP$, qu'on appellera informations élémentaires IE_i : l'information d'entrée sera une combinaison de ces informations élémentaires $IE_1,...IE_5$.

Une analyse plus fine permet d'observer que les informations élémentaires IE_1 et IE_2 sont associées aux mêmes signaux d'entrée *{1,2}* du composant *COMP* ; il s'ensuit que

cette information partielle *{1,2}* peut être générée indifféremment à partir de *C1* ou de *C2*. Cette propriété sera caractérisée par la notion de groupe d'entrée.

Pour minimiser la recherche des combinaisons possibles, l'étape suivante consiste à trouver une partition des signaux d'entrée du composant *COMP* impliquant une partition de l'ensemble des informations élémentaires associées. Cette partition conduit à définir des jonctions d'information.

La recherche des combinaisons n'a plus alors à être effectuée qu'à l'intérieur de ces classes. Dans cet exemple, ceci revient à dire que l'information *{3,4}* peut être générée :

- soit à partir du composant *C3*,
- soit à partir des composants *C4* et *C5* conjointement.

L'information globale peut donc être obtenue :

- pour les signaux *1* et *2*, soit à partir de *C1*, soit à partir de *C2* ;
- pour les signaux *3* et *4*, soit à partir de *C3*, soit à partir de *C4* et C5.

3.1.2. Modes de transfert de l'information sur une entrée

a) Le mode jonction

Il y a jonction lorsqu'une information *I* est la concaténation de deux informations I_i et I_j issues d'origines différentes : I_i et I_j sont deux informations de supports disjoints, et l'union des supports est égale au support de *I*.

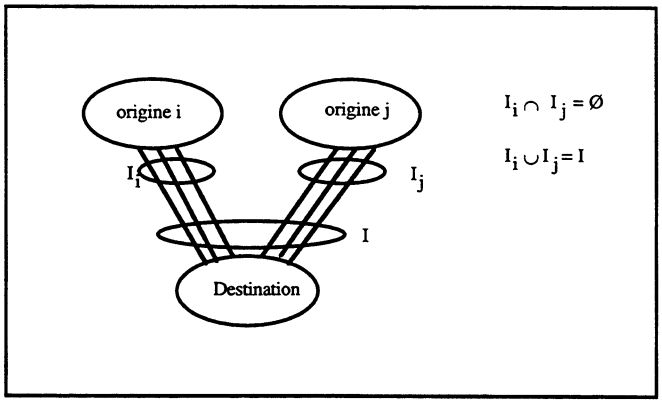

Figure 5.9. Concaténation d'information

Le mode jonction se représente comme suit :

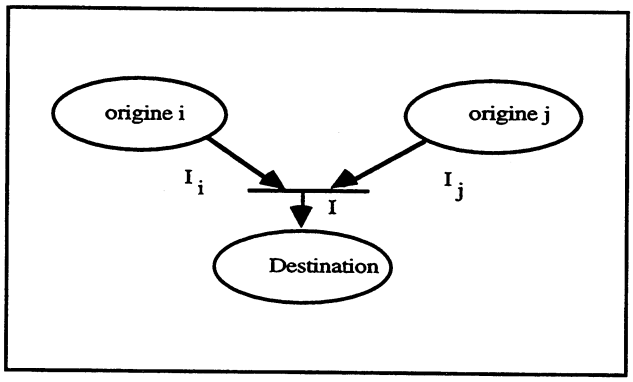

Figure 5.10. La jonction d'information

On notera l'information résultante : $I = I_i \bullet I_j$.

b) Le mode attribution

Il y a attribution lorsqu'une information I est issue, dans sa totalité, soit d'une origine $ORIGINE_i$, soit d'une origine $ORIGINE_j$: il y a sélection d'une des informations I_i ou I_j.

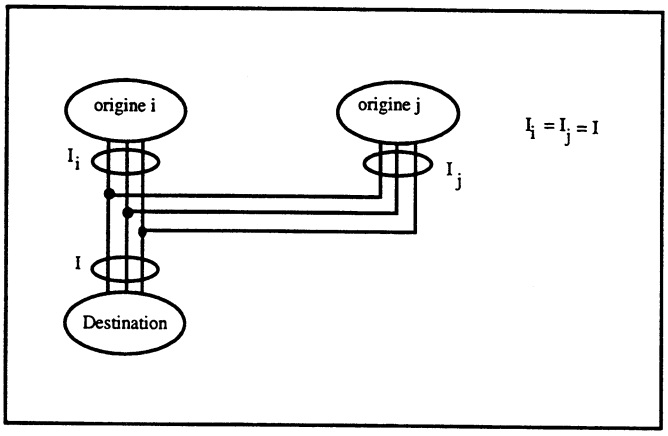

Figure 5.11. Choix de chemins d'information.

Le mode attribution se représente comme suit :

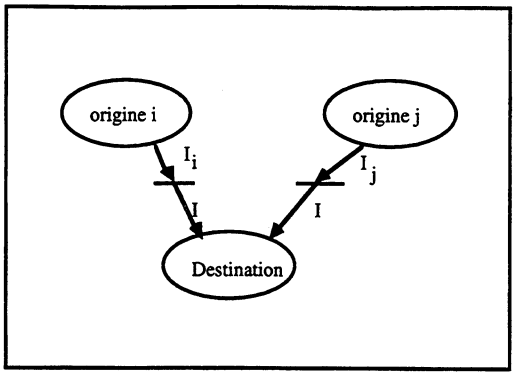

Figure 5.12. Le mode attribution.

On notera l'information résultante : $I = I_i + I_j$

3.1.3. Information élémentaire d'entrée

Définition 5.1. On définit une *origine* d'information par un couple formé du nom d'un composant et du nom d'une sortie fonctionnelle. Une origine sera notée *ORIGINE = (SF, COMP).*

Définition 5.2. Une *information élémentaire* IE d'une entrée fonctionnelle *EF* comme un couple, d'une part, formé par un sous-ensemble de signaux de *EF* ayant même origine et, d'autre part, formé par l'identification de cette origine. Des signaux ont même origine, si leurs valeurs proviennent :

- soit du connecteur externe
- soit de la même sortie fonctionnelle d'un composant adjacent de *EF* .

Une information élémentaire est alors définie par un couple :

IE = (INFO ; ORIGINE)

où :

- *INFO* = $\{E_1,...E_p\}$ est l'ensemble des signaux d'entrée de *EF* constituant *IE.*
- *ORIGINE* = $(SF_j, COMP_j)$ est l'origine de l'information (sortie fonctionnelle SF_j du composant $COMP_j$).

a) Exemple 1 :

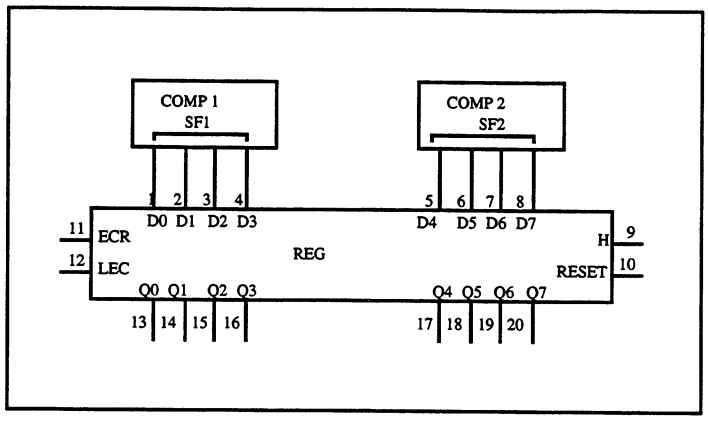

Figure 5.13. Cas de jonction.

Considérons l'entrée fonctionnelle *EF* du composant *REG* :

$EF = D0\text{-}7 = \{1,2,3,4,5,6,7,8\}$

Les informations élémentaires associées à *EF* sont :

$IE_1 = (\{1,2,3,4\} ; (SF_1, COMP_1))$

$IE_2 = (\{5,6,7,8\} ; (SF_2, COMP_2))$

Cet exemple illustre le mode jonction : l'entrée fonctionnelle *EF* est la concaténation des informations élémentaires IE_1 et IE_2. On remarque en effet que :

$INFO(IE_1) \cap INFO(IE_2) = \emptyset$

$INFO(IE_1) \cup INFO(IE_2) = EF$

On peut donc noter que $EF = IE_1 \bullet IE_2$

b) Exemple 2 :

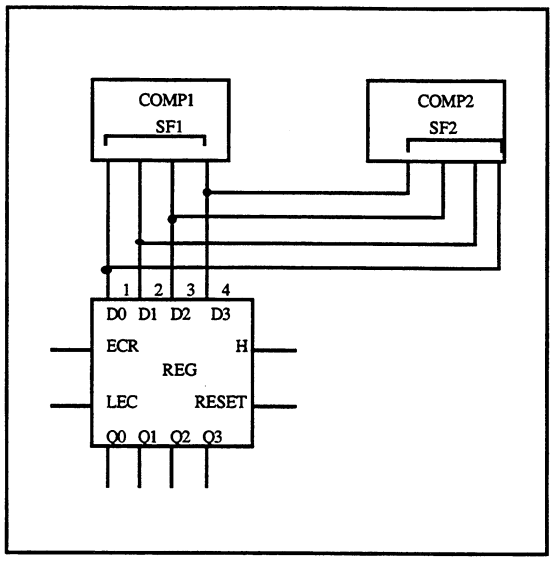

Figure 5.14. Cas d'attribution.

Considérons l'entrée fonctionnelle *EF = D0-3 = {1,2,3,4}* du composant REG. Les informations élémentaires associées à *EF* sont :

$IE_1 = (\{1,2,3,4\} ; (SF_1, COMP_1))$

$IE_2 = (\{1,2,3,4\} ; (SF_2, COMP_2))$

Cet exemple illustre le mode attribution : l'entrée fonctionnelle *EF* est une sélection des informations élémentaires IE_1 et IE_2. On remarque en effet que :

$INFO\ (IE_1) = INFO\ (IE_2) = EF$

3.1.4. Groupe d'entrée d'un composant

Lorsque plusieurs informations élémentaires d'entrée recouvrent les mêmes signaux d'une entrée fonctionnelle, il faudra, pour ces signaux, définir autant de chemins d'information qu'il y a d'informations élémentaires d'entrée. Ce cas de figure correspond à l'attribution d'information; Pour représenter cette propriété, nous avons introduit la notion de *groupe d'entrée* qui permet de partitionner l'ensemble des informations élémentaires d'entrée en classe par rapport aux signaux recouverts.

Définition 5.3. *Equivalence entre informations élémentaires d'entrée.* Soient $\{IE_k\}$, $1 \leq k \leq n$, les informations élémentaires associées à une entrée fonctionnelle :

$IE_k = (INFO_k, ORIGINE_k)$. Deux informations sont dites équivalentes, si et seulement si, elles ont même ensemble.

$$\forall (i,j) \in [1..k]^2, IE_i \text{ équivalente à } IE_j \Leftrightarrow (INFO(IE_i) = INFO(IE_j)$$

Nous notons $IE_i \equiv IE_j$ l'équivalence entre les informations élémentaires.

Définition 5.4. *Groupe d'entrée.* Soient $\{IE_k\}$, $1 \leq k \leq n$, les informations élémentaires associées à une entrée fonctionnelle : $IE_k = (INFO_k, ORIGINE_k)$.. Les groupes d'entrée sont construits à partir des informations élémentaires d'entrée, à l'aide de la relation d'équivalence de la définition 5.3.

A chacune des classes d'équivalence nous associons un groupe d'entrée défini par un couple $GE = (S; \{IE\})$ où :

- S est l'ensemble des signaux d'entrée,
- $\{IE\}$ est l'ensemble des informations élémentaires d'entrée telles que $INFO(IE) = S$

Par abus de notation, nous ne noterons que les origines de l'information élémentaire d'entrée dans la seconde composante d'un groupe d'entrée car les informations élémentaires d'un groupe d'entrée ont même support que le groupe d'entrée lui-même.

Exemple :

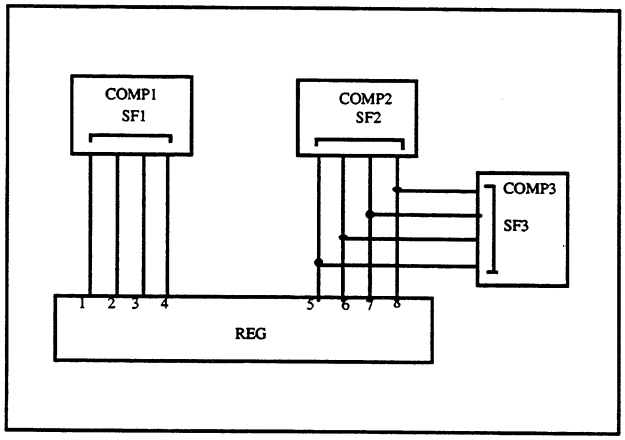

Figure 5.15. Groupes d'entrée.

Le registre *REG* comporte une entrée fonctionnelle $EF = \{1,2,3,4,5,6,7,8\}$ et trois informations élémentaires :

$IE_1 = (\{1,2,3,4\} ; (SF_1, COMP_1))$

$IE_2 = (\{5,6,7,8\} ; (SF_2, COMP_2))$

$IE_3 = (\{5,6,7,8\} ; (SF_3, COMP_3))$

On peut déterminer deux groupes d'entrée à partir de ces informations élémentaires :

$GE_1 = (\{1,2,3,4\} ; \{(SF_1, COMP_1)\})$

$GE_2 = (\{5,6,7,8\} ; \{(SF_2, COMP_2), (SF_3, COMP_3)\})$

Un groupe d'entrée est une attribution d'informations élémentaires ; on notera :

$GE_k = IE_i + ... + IE_j$

3.1.5. Hypergroupe d'entrée d'un composant

Définition 5.5. *Hypergroupe d'entrée.* Soit une entrée fonctionnelle *EF* et les groupes d'entrée associés : $GE_k = (S_k ; \{IE\}_k)$.

Un hypergroupe d'entrée *HGE* est construit à partir des groupes d'entrée $\{GE_k\}$sur la base d'un partitionnement de l'ensemble des groupes d'entrée à l'aide de la relation d'équivalence suivante :

$GE_i \, \Re \, GE_j \Leftrightarrow \exists \{GE_{k,1}, \ldots, GE_{k,n}\}$ *tel que :*

$$\begin{cases} GE_{k,1} = GE_i \\ GE_{k,n} = GE_j \\ \forall\, p \in [1..n-1], GE_{k,p} \cap GE_{k,p+1} \neq \varnothing \end{cases}$$

Un hypergroupe d'entrée est défini par un couple dont l'une des composantes est l'ensemble des groupes d'entrée associés à la classe d'équivalence définissant l'hypergroupe d'entrée, et l'autre composante l'union sur ces groupes de leur information.

$HGE = (INFO ; [GE])$

où

INFO est l'ensemble des signaux d'entrée des groupes GE_k dont *HGE* est issu :

$INFO\ (HGE) = \cup_k\ INFO(GE_k)$

ORIGINES est l'union des origines des groupes d'entrée GE_k dont *HGE* est issu.

Par définition, les ensembles de signaux d'entrée associées aux hypergroupes sont disjoints deux à deux :

$$\forall i, \forall j, i \neq j, INFO(HGE_i) \cap INFO(HGE_j) = \emptyset$$

Exemple :

Soit le réseau d'interconnexion donné dans la figure 5.16.

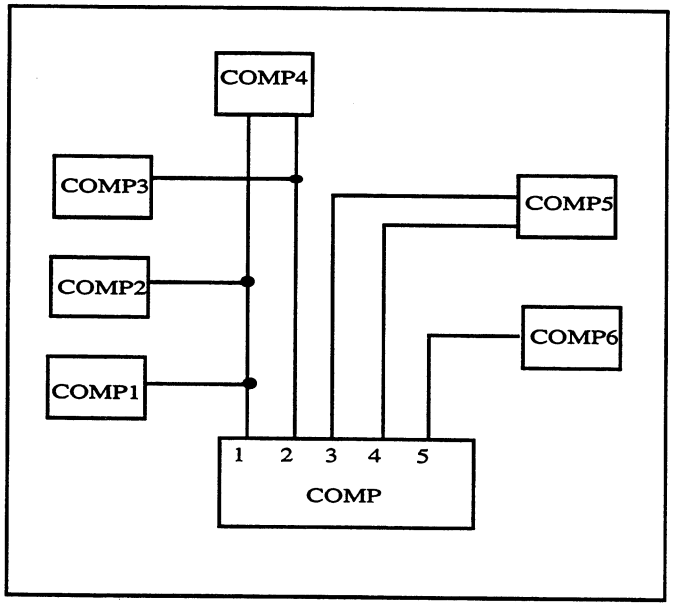

Figure 5.16. Réseau d'interconnexion.

Les informations élémentaires de l'entrée fonctionnelle $EF = \{1,2,3,4,5\}$ du composant *COMP* sont :

IE_1	{1}	(SF_1, $COMP_1$)
IE_2	{1}	(SF_2, $COMP_2$)
IE_3	{2}	(SF_3, $COMP_3$)
IE_4	{1,2}	(SF_4, $COMP_4$)
IE_5	{3,4}	(SF_5, $COMP_5$)
IE_6	{5}	(SF_6, $COMP_6$)

Il y a 5 groupes d'entrée :

GE_1	{1}	$(SF_1, COMP_1), (SF_2, COMP_2)$	$IE_1 + IE_2$
GE_2	{2}	$(SF_3, COMP_3)$	IE_3
GE_3	{1,2}	$(SF_4, COMP_4)$	IE_4
GE_4	{3,4}	$(SF_5, COMP_5)$	IE_5
GE_5	{5}	$(SF_6, COMP_6)$	IE_6

Il y a trois hypergroupes d'entrée :

HGE_1	GE_1, GE_2, GE_3	{1,2}	$(SF_1, COMP_1), (SF_2, COMP_2), (SF_3, COMP_3), (SF_4, COMP_4)$
HGE_2	GE_4}	{3,4}	$(SF_5, COMP_5)$
HGE_3	GE_5	{5}	$(SF_6, COMP_6)$

Il faut noter que l'information nécessaire sur une entrée fonctionnelle EF sera une jonction des hypergroupes d'entrée. On note

$$EF = HGE_i \bullet ..... \bullet HGE_j$$

On a vu qu'un hypergroupe d'entrée est issu d'un ensemble de groupes d'entrée ; il s'agit donc, pour chaque hypergroupe d'entrée *HGE*, de définir les combinaisons possibles des groupes d'entrée sont nécessaires et suffisantes pour générer l'information partielle associée à *HGE*.

Dans l'exemple précédent, considérons l'hypergroupe d'entrée HGE_1 générant l'information partielle *{1,2}*. On voit que cette information peut être générée, à partir de :

- soit GE_3,
- soit GE_1 et GE_2

toutes les autres combinaisons étant invalides.

3.1.6. Supergroupe d'entrée d'un composant

Considérons un hypergroupe d'entrée $HGE = \{GE_1, ... GE_p\}$ couvrant les signaux d'entrée $\{E_1, ... E_n\}$.

On définit un supergroupe d'entrée *SGE* d'un hypergroupe d'entrée comme un sous-ensemble de *HGE* tel que :

- les informations des groupes d'entrée de *SGE* sont disjointes deux à deux,
- l'union des informations des groupes d'entrée de *SGE* est égale à l'information de *HGE*.

Soit $HGE = \{GE_1, \ldots GE_p\}$ et $SGE = \{GE_1, \ldots GE_m\}$

$\forall i \in [1,m], \forall j \in [1,m], i \neq j \quad INFO(GE_i) \cap INFO(GE_j) = \emptyset$

$\forall i \in [1,m], \cup_i INFO(GE_i) = INFO(HGE)$

Parmi l'ensemble des combinaisons possibles des groupes d'entrée de *HGE*, on recherche les combinaisons telles que :

- une entrée E_i appartenant à *INFO(HGE)* n'appartient qu'à un seul groupe d'entrée,
- l'union des informations des groupes d'entrée est égale à l'information de *HGE*.

3.1.7. Récapitulation des règles sur une entrée fonctionnelle

L'ensemble de ces définitions permet donc de définir de manière très précise le mode de génération d'une entrée fonctionnelle d'un composant, à partir des informations élémentaires :

- une entrée fonctionnelle *EF* est une jonction d'hypergroupes d'entrée,
- un hypergroupe d'entrée est une attribution de supergroupes d'entrée,
- un supergroupe d'entrée est une jonction de groupes d'entrée,
- un groupe d'entrée est une attribution d'informations élémentaires.

D'après les définitions et les représentations graphiques des modes jonction et attribution, on peut donc représenter chaque étape de construction d'une entrée fonctionnelle à partir des informations élémentaires (figure 5.17) :

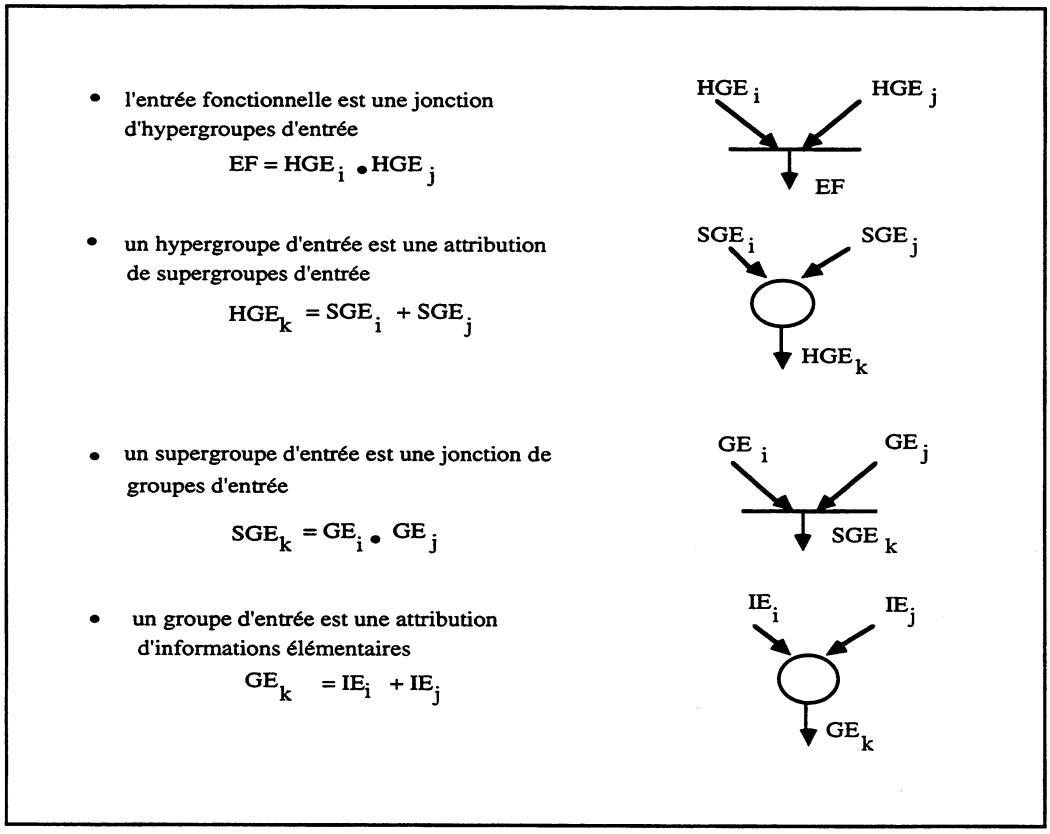

Figure 5.17. Récapitulation sur les entrées fonctionnelles.

3.1.8. Exemple de synthèse

On considère un registre 8 bits connecté aux composants $COMP_1,...,COMP_{10}$ comme suit :

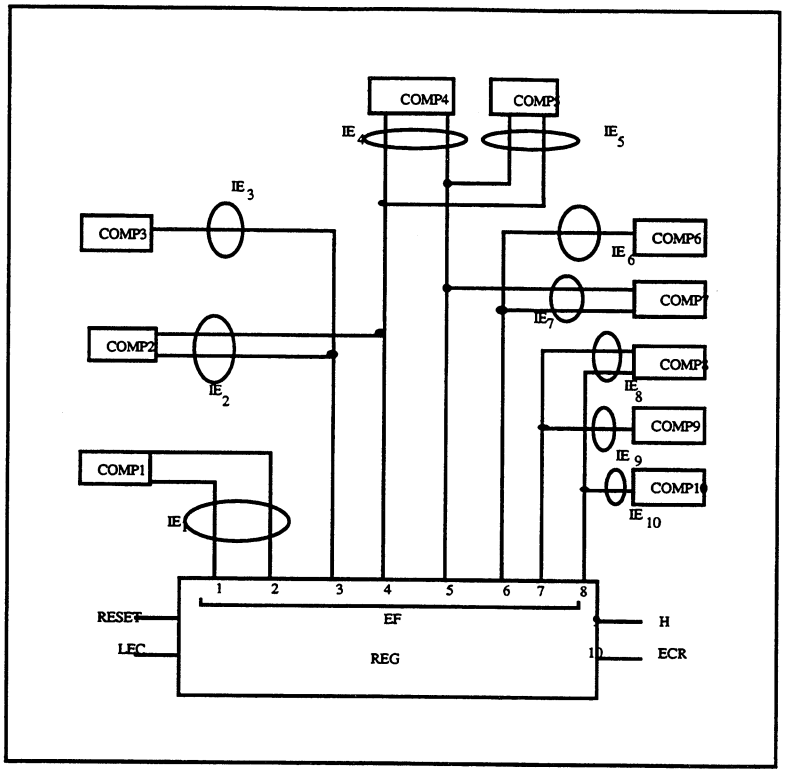

Figure 5.18. Exemple récapitulatif en entrée.

Pour des raisons de simplification, on représentera l'origine *(SF_i, $COMP_i$)* par le seul nom du composant (*$COMP_i$*).

Le composant *REG* a une entrée fonctionnelle :

EF = D0-7 = {1,2,3,4,5,6,7,8}

Les informations élémentaires de EF sont :

IE_1	IE_2	IE_3	IE_4	IE_5	IE_6	IE_7	IE_8	IE_9	IE_{10}
{1,2}	{3,4}	{3}	{4,5}	{4,5}	{6}	{5,6}	{7,8}	{7}	{8}
$COMP_1$	$COMP_2$	$COMP_3$	$COMP_4$	$COMP_5$	$COMP_6$	$COMP_7$	$COMP_8$	$COMP_9$	$COMP_{10}$

Les groupes d'entrée sont les suivants :

GE_1	IE_1	{1,2}	$COMP_1$
GE_2	IE_2	{3,4}	$COMP_2$
GE_3	IE_3	{3}	$COMP_3$
GE_4	$IE_4 + IE_5$	{4,5}	$(COMP_4), (COMP_5)$
GE_5	IE_6	{6}	$COMP_6$
GE_6	IE_7	{5,6}	{$COMP_7$
GE_7	IE_8	{7,8}	{$COMP_8$
GE_8	IE_9	{7}	$COMP_9$
GE_9	IE_{10}	{8}	$COMP_{10}$

On définit 3 hypergroupes d'entrée :

HGE_1	GE_1	{1,2}
HGE_2	$GE_2, GE_3, GE_4, GE_5, GE_6$	{3,4,5,6}
HGE_3	GE_7, GE_8, GE_9	{7,8}

Calcul des supergroupes d'entrée :

• HGE_1 comportant un seul élément, le supergroupe d'entrée associé est égal à lui même :

$$SGE_1 = HGE_1 = GE_1$$

• HGE_2 donne lieu à deux supergroupes d'entrée :

$$SGE_2 = GE_2 \bullet GE_6$$

$$SGE_3 = GE_3 \bullet GE_4 \bullet GE_5$$

$$HGE_2 = SGE_2 + SGE_3$$

• HGE_3 donne lieu à deux supergroupes d'entrée :

$$SGE_4 = GE_7$$

$$SGE_5 = GE_8 \bullet GE_9$$

$$et\ HGE_3 = SGE_4 + SGE_5$$

On peut donc écrire l'entrée fonctionnelle *EF* comme suit :

$$EF = HGE_1 \bullet HGE_2 \bullet HGE_3$$

$$soit\ EF = GE_1 \bullet (SGE_2 + SGE_3) \bullet (SGE_4 + SGE_5)$$

$$soit\ EF = GE_1 \bullet (GE_2 \bullet GE_6 + GE_3 \bullet GE_4 \bullet GE_5) \bullet (GE_7 + GE_8 \bullet GE_9)$$

$$soit\ EF = IE_1 \bullet [IE_2 \bullet IE_7 + IE_3 \bullet (IE_4 + IE_5) \bullet IE_6] \bullet (IE_8 + IE_9 \bullet IE_{10})$$

On peut donc représenter l'équation générale d'une entrée fonctionnelle par une combinaison des graphes précédents, telle que :

- chaque jonction $A = B \bullet C$ est représentée par une transition ayant *B* et *C* comme entrée, *A* comme sortie ;
- chaque attribution $A = B + C$ est représentée par un module ayant *B* et *C* comme entrées, *A* comme sortie.

Exemple :

Si nous considérons l'exemple de synthèse (§1.7) dont l'équation de l'entrée fonctionnelle est la suivante :

$$EF = D0\text{-}7 = IE_1 \bullet [(IE_2 \bullet IE_7 + IE_3 \bullet (IE_4 + IE_5) \bullet IE_6] + (IE_8 + IE_9 \bullet IE_{10})$$

le graphe associé est le suivant :

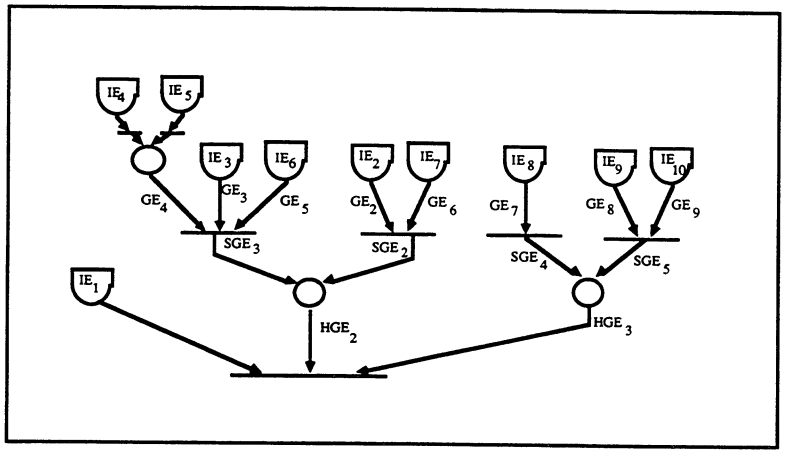

Figure 5.19. Graphe spécifique final.

Le graphe spécifique final après analyse des entrées fonctionnelles est obtenu en remplaçant, pour chaque transition successeur de l'entrée fonctionnelle *EF* , la source

EF et son arc de sortie par le graphe associé à l'analyse des entrées. Les autres entrées de la transition restent inchangées.

3.2. Analyse des sorties fonctionnelles

L'information utile issue d'une sortie fonctionnelle (à savoir les signaux de sortie connectés au réseau) est à destination, de manière partielle ou globale, d'autres composants de la description.

L'objectif est de spécifier l'information issue de chaque sortie fonctionnelle d'un composant relativement aux destinations de cette information.

Après analyse des entrées fonctionnelles, il s'agit donc uniquement de retrouver les informations élémentaires qui sont nécessaires en entrée des graphes génériques des composants adjacents. La démarche d'analyse et sa justification sont similaires à celles données pour l'analyse des entrées fonctionnelles.

3.2.1. Information élémentaire de sortie

Définition. Une *destination* est un couple formé du nom d'un composant, d'une part, et du nom d'une sortie fonctionnelle de ce composant d'autre part.

Définition. Une *information élémentaire IE* d'une sortie fonctionnelle *SF* est définie comme un sous-ensemble des signaux de *SF* ayant même destination, à savoir la même entrée fonctionnelle d'un composant destination de *SF* :

Une information élémentaire est alors définie par un couple :

$IE = (INFO ; DESTINATION)$

où

- $INFO = \{S_1, \ldots S_p\}$ est l'ensemble des signaux de sortie de *SF* constituant *IE*,
- $DESTINATION = (EF_j, COMP_j)$ est la destination de cette information, à savoir l'entrée fonctionnelle EF_j du composant $COMP_j$.

3.2.2. Groupe de sortie d'un composant

Soit une sortie fonctionnelle *SF* et les informations élémentaires IE_k associées à cette sortie fonctionnelle :

$IE_k = (INFO_k ; DESTINATION_k)$

A toute information élémentaire IE_i on associe un groupe de sortie GS_i si et seulement si :

$\forall i, \forall j, i \neq j, INFO(IE_i) \neq INFO(IE_j)$

S'il existe deux informations élémentaires comportant les mêmes signaux de sortie : $INFO(IE_i) = INFO(IE_j)$, on associe un même groupe de sortie à ces deux informations élémentaires.

Un groupe de sortie est alors défini par un couple :

$GS = (INFO ; \{DESTINATIONS\})$

où

• *INFO* est l'ensemble des signaux d'entrée de l'information élémentaire dont *GS* est issu,

• *DESTINATIONS* est la ou les destinations des informations élémentaires dont *GS* est issu.

Exemple :

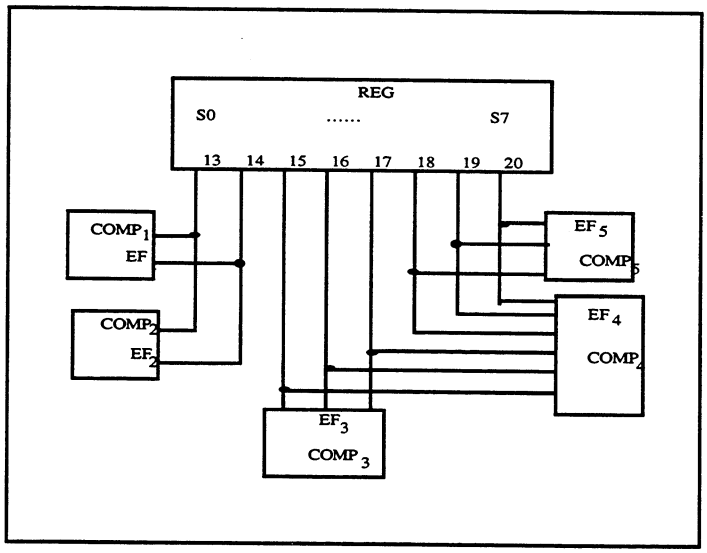

Figure 5.19. Réseau d'interconnexion en sortie.

Le registre REG comporte une sortie fonctionnelle

$SF = S0\text{-}7 = \{13, 14, 15, 16, 17, 18, 19, 20\}$.

Cette sortie fonctionnelle comporte 5 informations élémentaires :

IE_1	{13, 14}	(EF_1, $COMP_1$)
IE_2	{13, 14}	(EF_2, $COMP_2$)
IE_3	{15, 16, 17}	(EF_3, $COMP_3$)
IE_4	{15, 16, 17, 18, 19, 20}	(EF_4, $COMP_4$)
IE_5	{18, 19, 20}	(EF_5, $COMP_5$)

On peut déterminer quatre groupes de sortie à partir de ces informations élémentaires :

GS_1	$E_1 + IE_2$	{13, 14}	(EF_1, $COMP_1$), (EF_2, $COMP_2$)
GS_2	IE_3	{15, 16, 17}	(EF_3, $COMP_3$)
GS_3	IE_4	{15, 16, 17, 18, 19, 20}	(EF_4, $COMP_4$)
GS_4	IE_5	{18, 19, 20}	(EF_5, $COMP_5$)

• Modifications du graphe générique induites par les groupes de sortie (figures 5.20 et 5.21)

Considérons une sortie fonctionnelle SF comportant p groupes de sortie $GS_1, ..., GS_p$: chaque groupe de sortie correspond à une information (éventuellement partielle) de la sortie fonctionnelle *SF*.

La modification résultante sur le graphe générique consiste à remplacer la transition Ti, en amont de *SF*, par un ensemble de transitions T_{ik}, $k \in [1,p]$, telles que :

• T_{ik} a comme antécédent le module prédécesseur de T_i,

• T_{ik} a comme successeur le groupe de sortie GS_k,

où GS_k est :

- soit une information élémentaire,
- soit le représentant de deux informations élémentaires IE_i et IE_j telles que $INFO(IE_i) = INFO(IE_j)$.

$INFO(IE_i) = INFO(IE_j)$.

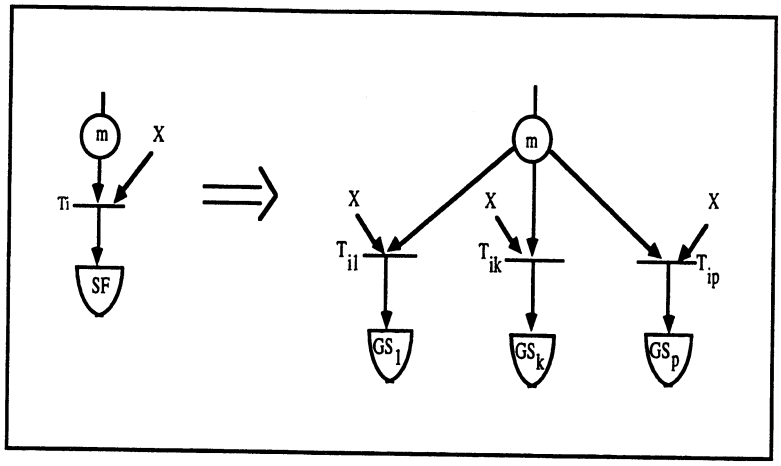

Figure 5.20. Modifications sur la sortie.

S'il existe plusieurs transitions en amont de la sortie fonctionnelle *SF*, ce processus est appliqué pour chaque transition.

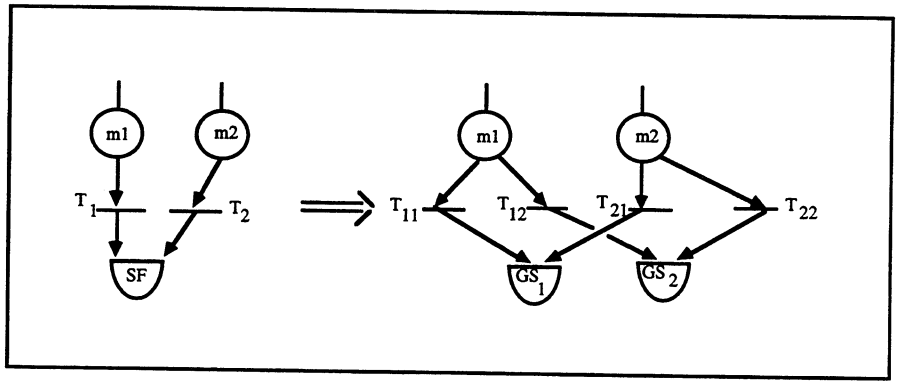

Figure 5.21. Modifications avec transitions multiples.

3.3. Le Graphe des testabilité d'une description structurelle

L'analyse du réseau d'interconnexions permet de définir un graphe spécifique final pour tous les composants de la carte étudiée, à partir des graphes génériques des composants. La concaténation de l'ensemble de ces graphes définit le graphe de testabilité de la carte. L'obtention de ce graphe se fait en confondant les groupes de sortie *GSi* d'un composant *COMP* et les informations élémentaires *IEj* d'un composant aval *COMP**, tels que :

$INFO(GS_i) = INFO(IE_j)$

$DESTINATION(GS_i) = COMP^*$

$ORIGINE(IEj) = COMP$

4. Exemple

Nous considérons une carte hypothétique dont le schéma logique est donné ci-après. La carte comporte :

- deux registres 4-bits : 4076-1 et 4076-2, notés *REG1* et *REG2* ;
- un registre 8-bits noté *REG* ;
- un circuit d'incrémentation (*INCR*) et un circuit de décrémentation (*DECR*) opérant sur des données de 4 bits.

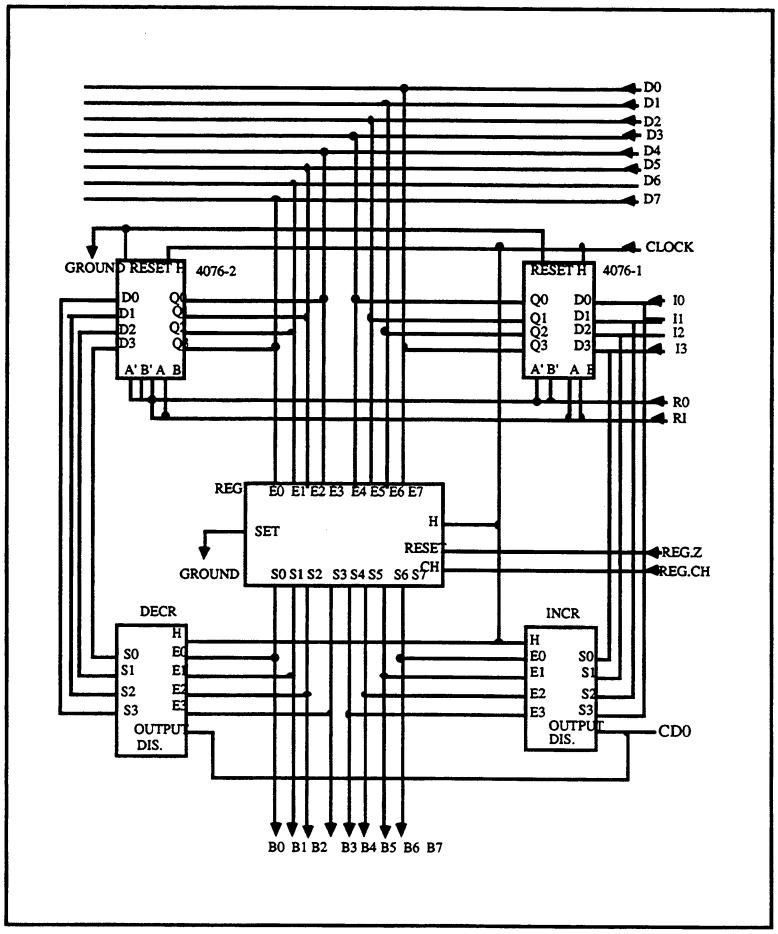

Figure 5.22. Schéma de la description structurelle.

4.1. Le type Satan des différents composants

4.1.1. Graphes génériques simplifiés des composants de la carte

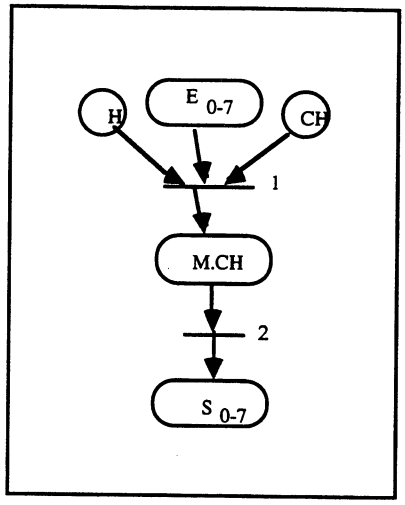

Figure 5.23. Graphe générique du registre 8 bits.

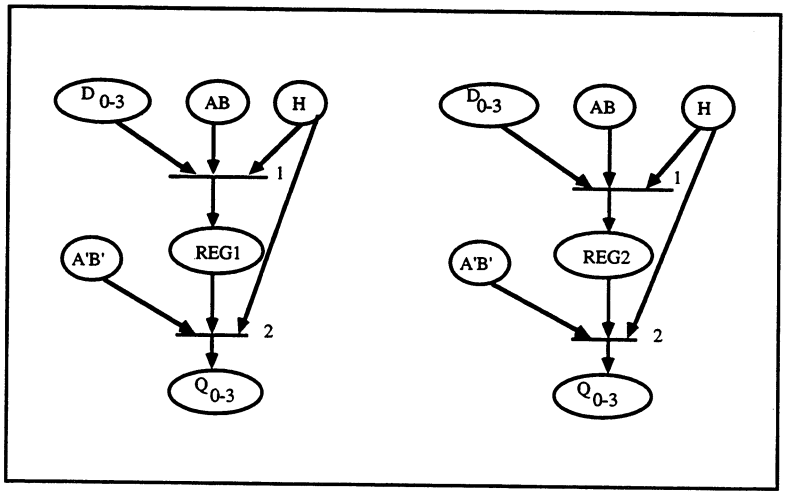

Figure 5.24. Graphes génériques des registresREG1 et REG2.

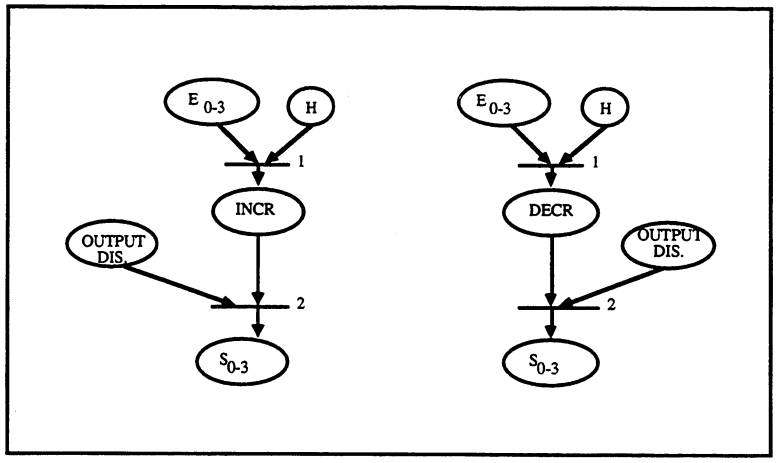

Figure 5.25. Graphes génériques des circuits d'incrémentation et de décrémentation.

4.1.2. Les entrées/sorties fonctionnelles des composants

• Registre 8 bits *REG* :

Le registre comporte une entrée fonctionnelle de données :

$EF = E_{0\text{-}7} = \{E_0, \ldots E_7\}$,

et une sortie fonctionnelle :

$SF = S_{0\text{-}7} = \{S_0, \ldots, S_7\}$

• Registres 4 bits *REG1* et *REG2* :

Les registres 4 bits *REG1* et *REG1* comportent une entrée fonctionnelle de données :

$EF1 = D_{0\text{-}3} = \{D_0, \ldots D_3\}$

et une sortie fonctionnelle :

$SF_1 = Q_{0\text{-}3} = \{Q_0, \ldots Q_3\}$

• Circuits d'incrémentation et de décrémentation

Les circuits comportent une entrée fonctionnelle

$E_{0\text{-}3} = \{E_0, \ldots, E_3\}$

et une sortie fonctionnelle

$S_{0\text{-}3} = \{S_0, \ldots S_3\}$

• Concernant le connecteur externe, on notera :

$EXT1 = \{D0, \ldots D7\}$

$EXT2 = \{I0, \ldots I3\}$

$EXT3 = \{B0, \ldots B7\}$

4.2. Le graphe spécifique final du registre *REG*

• Analyse des entrées fonctionnelles :

Le composant a une entrée fonctionnelle de données :

$EF = E_{0\text{-}7} = \{E_0, E_1, \ldots E_7\}$

Les informations élémentaires sont :

IE_1	$\{E_0, \ldots, E_7\}$	EXT1)
IE_2	$\{E_0, \ldots, E_3\}$	(CS_2, REG_2)
IE_3	$\{E_4, \ldots, E_7\}$	(CS_1, REG_1)

Ces informations élémentaires sont des groupes d'entrée :

GE_1	IE_1
GE_2	IE_2
GE_3	IE_3

Il existe un hypergroupe d'entrée :

$HGE_1 = \{E_0, \ldots, E_7\} = \{GE_1, GE_2, GE_3\}$

Cet hypergroupe donne lieu à deux supergroupes d'entrée :

$SGE_1 = GE_1$

$SGE_2 = GE_2 \bullet GE_3$

On peut donc écrire l'entrée fonctionnelle de données :

$EF = E_{0\text{-}7} = HGE_1 = SGE_1 + SGE_2 = GE_1 + GE_2 \bullet GE_3 = IE_1 + IE_2 \bullet IE_3$

• Analyse des sorties fonctionnelles

Le registre *REG* comporte une sortie fonctionnelle :

$SF = S_{0\text{-}7} = \{S_0, S_1, \ldots, S_7\}$

Cette sortie fonctionnelle comporte 3 informations élémentaires :

IE_1	$\{S_0,..., S_7\}$	EXT3)
IE_2	$(\{S_0,..., S_3\}$	$(CE_D$, DECR)
IE_3	$\{S_4,..., S_7\}$	$(CE_I$, INCR)

Les groupes de sortie sont donc :

GS_1	IE_1
GS_2	IE_2
GS_3	IE_3

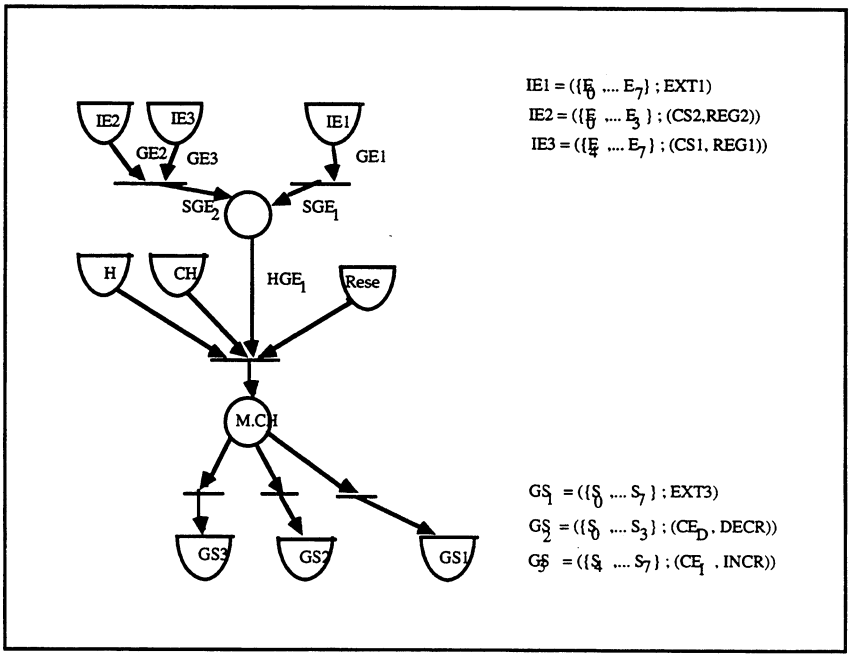

Figure 5.26. Graphe spécifique final du registre REG.

4.3. Le graphe spécifique final du registre *REG1*

REG1 comporte une entrée fonctionnelle :

$EF_1 = \{D_0,... D_3\}$

Il y a deux informations élémentaires

$IE_1 = (\{D_0,...D_3\} ; EXT2)$

$IE_2 = (\{D_0,... D_3\} ; (CS_I, INCR))$

Il y a donc un groupe d'entrée :

$GE_1 = (\{D_0,...D_3\} ; (EXT2), (CS_I, INCR))$

et par conséquent un seul hypergroupe d'entrée

$HGE_1 = GE_1 = IE_1 + IE_2$

cet hypergroupe comportant un seul élément

$SGE_1 = HGE_1 = GE_1$

La sortie fonctionnelle est constituée d'une seule information élémentaire

$IE_1 = (\{Q_0,... Q_3\} ; (CE, REG).$

Il y a donc un groupe de sortie unique $GS_1 = IE_1$ qui est donc groupe primaire de recouvrement de sortie.

Le graphe spécifique final du composant *REG1* est donc :

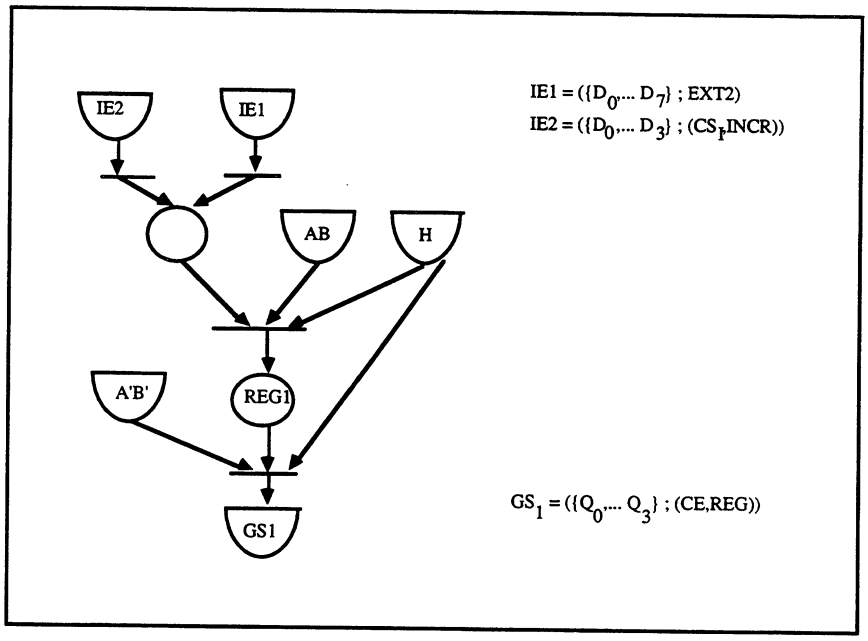

Figure 5.27. GSF du registre REG1.

4.4. Le graphe spécifique final du registre *REG2*

L'entrée fonctionnelle EF_1 ne comporte qu'une information élémentaire

$IE_1 = (\{D_0, ...D_3\} ; (CS_D, DECR)).$

La sortie fonctionnelle SF_1 ne comporte également qu'une information élémentaire :

$IE_1 = (\{Q_0,...Q_3\} ; (CE, REG))$

Le graphe spécifique final est donc le graphe générique :

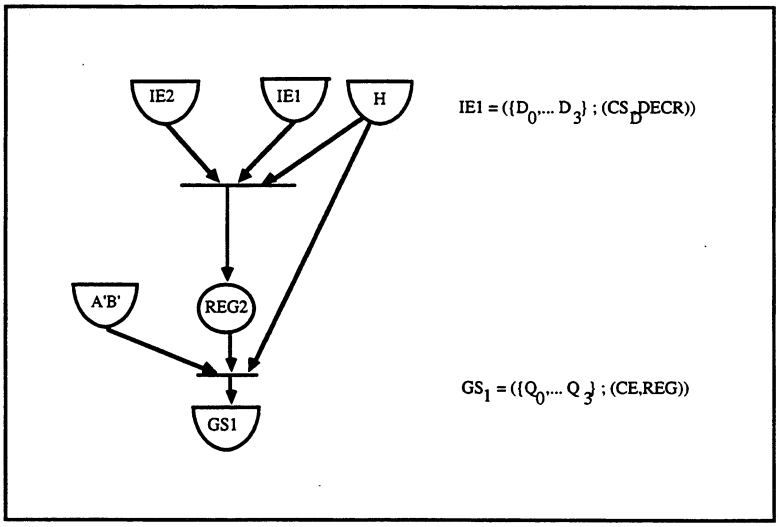

Figure 5.28. GSF du registre REG 2.

4.5. Le graphe spécifique final des composants *INCR* et *DECR*

L'entrée fonctionnelle et la sortie fonctionnelle de ces composants comportent une seule information élémentaire. Le graphe spécifique final est donc inchangé.

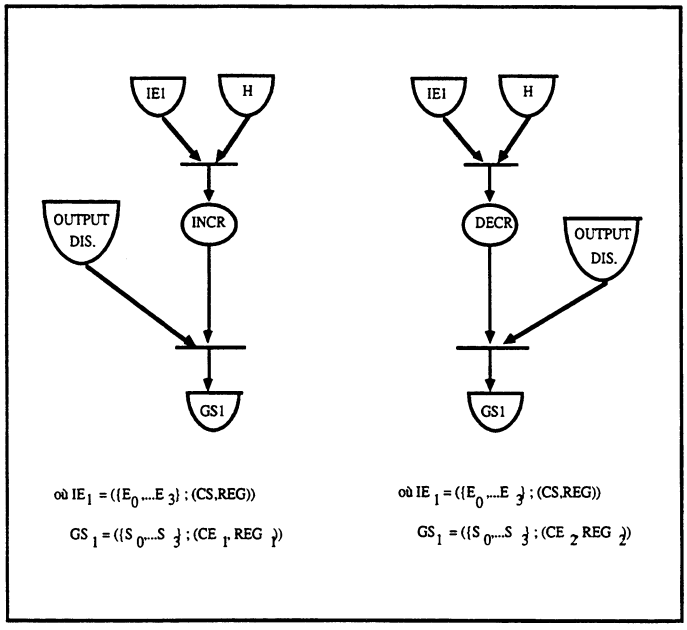

Figure 5.29. GSF de INCR et DECR.

4.6. Le graphe de testabilité de la carte

Le graphe de testabilité de la carte est obtenu par concaténation des graphes spécifiques finaux de tous les composants.

La concaténation se fait entre :

GS_3 de REG	IE_1 de INCR
GS_2 de REG	IE_1 de DECR
IE_2 de REG	GS_1 de REG_2
IE_3 de REG	GS_1 de REG_1
IE_2 de REG_1	GS_1 de INCR
IE_1 de REG_2	GS_1 de DECR

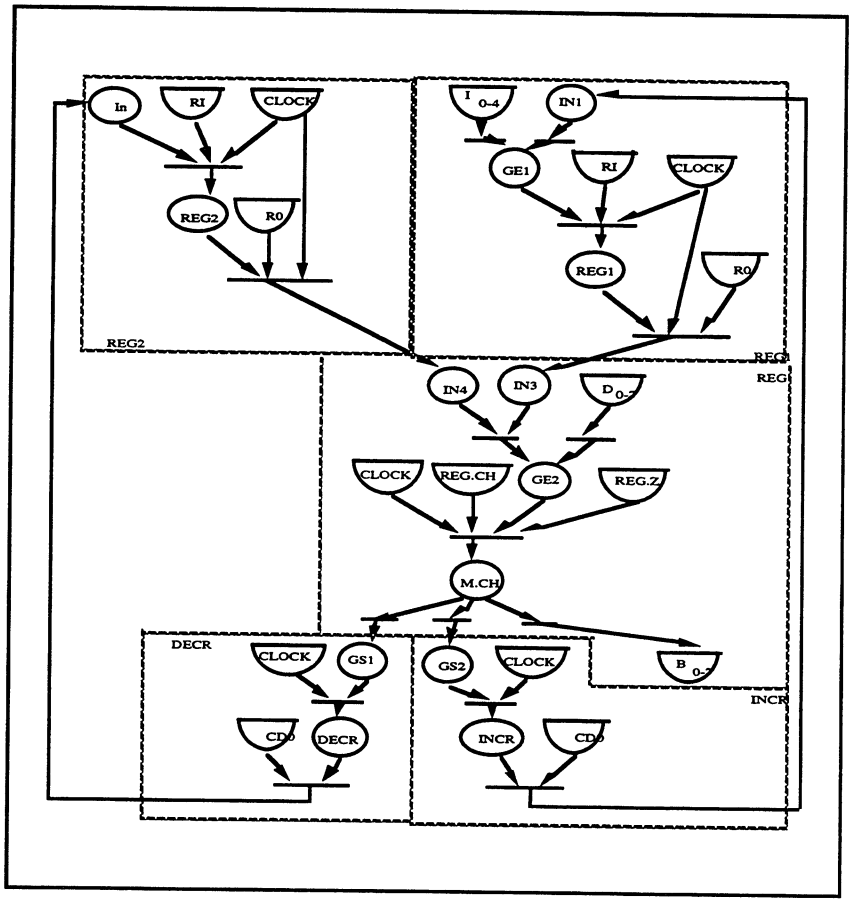

Figure 5.30. Graphe de transfert d 'information du circuit.

5. Analyse de testabilité de l'exemple

Considérons l'exemple précédent, dont la construction du graphe de transfert d'information est donnée dans le paragraphe 4. Nous nous intéressons à l'analyse de testabilité et à la spécification fonctionnelle du programme de test de ce circuit.

5.1. Ecoulements

Le GTI du circuit donné en exemple comporte trois écoulements d'information : ECL1, ECL2 et ECL3. Les figures 5.31, 5.32 et 5.33 montrent chacun de ces écoulements.

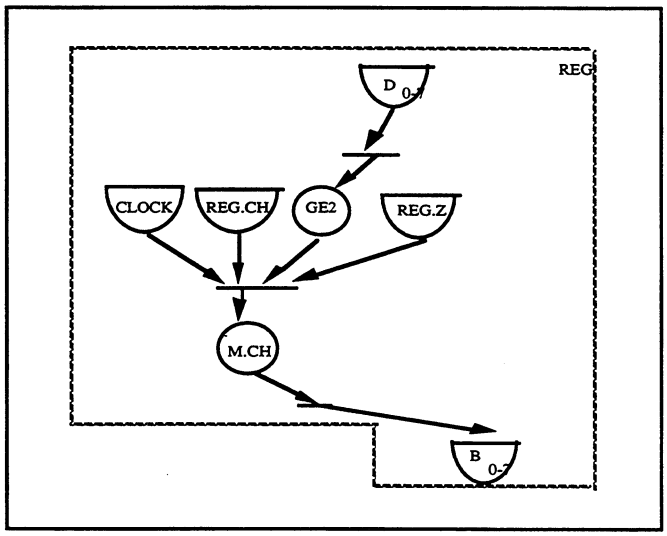

Figure 5.31. Ecoulement ECL1.

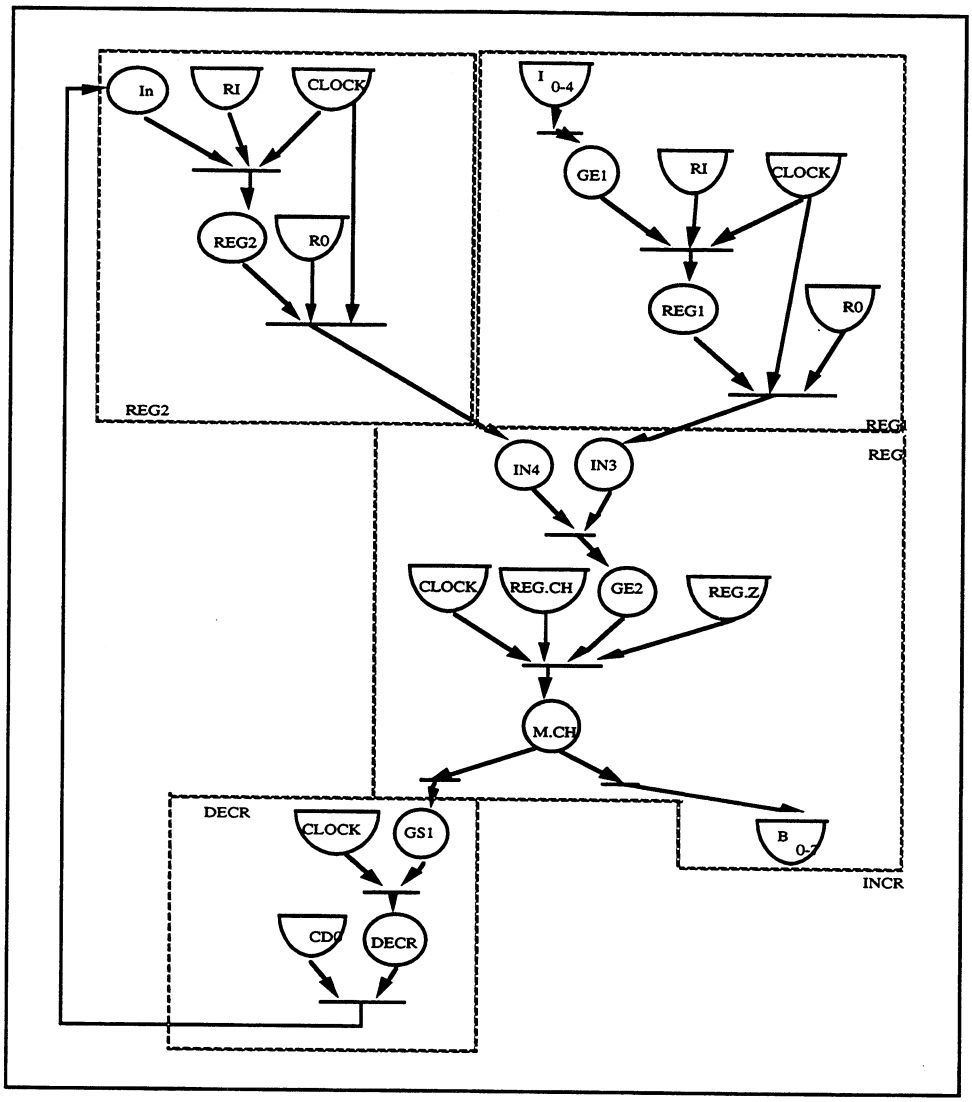

Figure 5.32. Ecoulement ECL2.

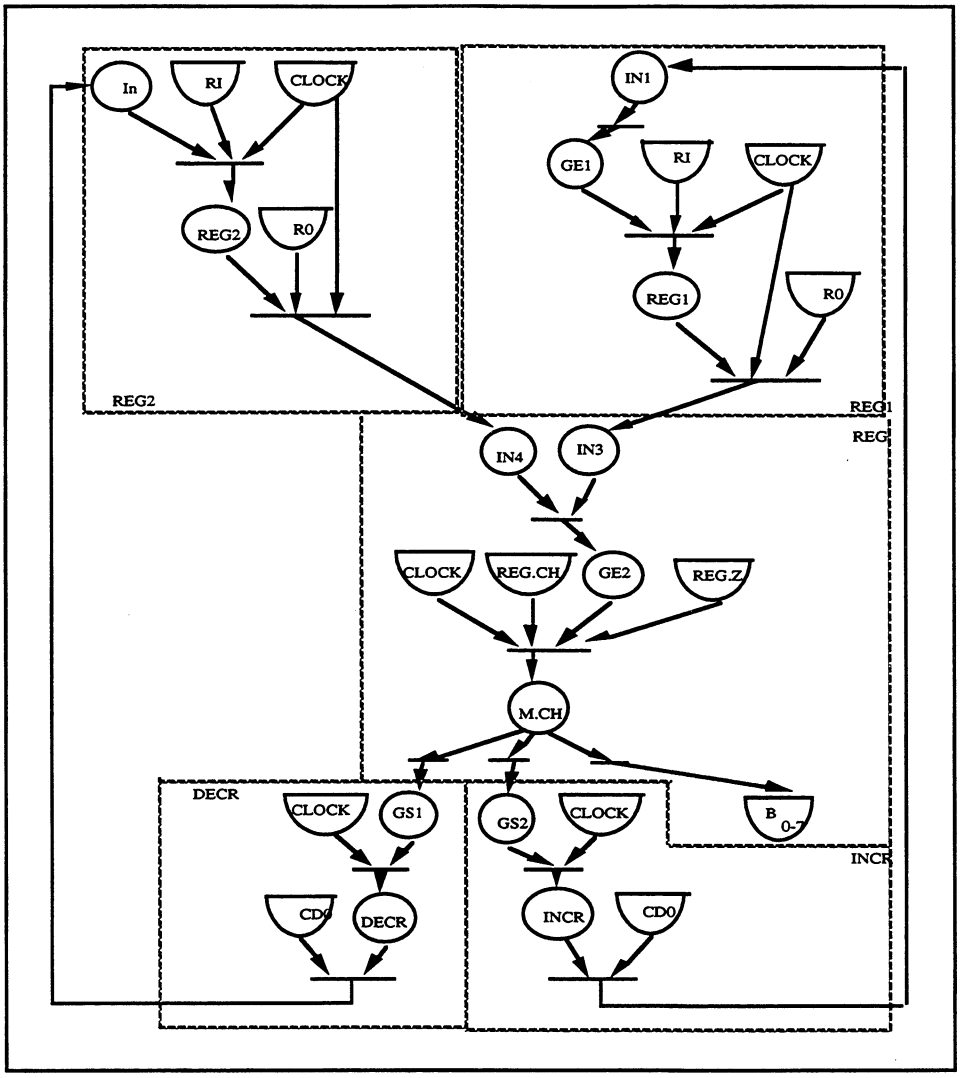

Figure 5.33. Ecoulement ECL3.

5.2. Spécification fonctionnelle du programme de test

Chacun des trois écoulements du circuit recouvre un certain nombre de modules. Pour déterminer la spécification fonctionnelle du programme de test, il est possible d'appliquer une stratégie de test qui prend en compte le critère de couverture. Par exemple, une stratégie de type *"boule de neige"* consiste à ordonner les écoulements à activer de manière croissante en fonction du nombre de modules qu'ils recouvrent.

Une telle stratégie induirait la structure du programme de test définie par l'activation dans l'ordre des écoulements *ECL1*, *ECL2* et *ECL3*.

Modules recouverts par chacun des écoulements :

ECL1

sources : D0..7, CLOCK, REG.CH, REG.Z

puits : B0..7

modules : GE2, M.CH

ECL2

sources : I0..4, CLOCK, REG.CH, REG.Z, RI, R0, CDO

puits : B0..7

modules : GE2, M.CH, REG2, GE1, REG1, IN3, IN4, GS1, DECR, In

ECL3

sources : CLOCK, REG.CH, REG.Z, RI, R0, CDO

puits : B0..7

modules : GE2, M.CH, REG2, GE1, REG1, IN3, IN4, GS1, DECR, In, IN1, GS2, INCR

Le tableau ci-dessous indique, pour chacun des modules du graphe, le pas du programme de test dans lequel le module est testé.

module	GE2	M.CH	REG2	GE1	REG1	N3	IN4	GS1	DECR	IN	IN1	GS2	INCR
pas	1	1	2	2	2	2	2	2	2	2	3	3	3

Le second pas du programme de test indique une indiscernabilité assez importante car tous les modules testés dans ce pas sont indiscernables entre-eux. Ceci induit donc un effort de test important car il faudra déterminer des valeurs de test qui activent correctement tous ces modules. Du point de vue de la localisation, il sera également difficile, lors de la détection d'un défaut en sortie, de déterminer le module fautif. pour cela il est nécessaire de choisir des valeurs de test qui permettent la discernabilité.

6. Conclusion

Dans ce chapitre nous avons présenté la compilation de descriptions CHDL structurelles sous forme de graphe de transfert d'information dynamique. Le principe

repose sur l'extraction de la bibliothèque du graphe de transfert d'information des différents composants référencés dans la description structurelle. Les graphes extraits sont ensuite simplifiés à la vue des éventuelles valeurs constantes associées à des signaux d'entrée des composants. Puis le réseau d'interconnexion est analysé pour déterminer des transformations sur les sources et les puits des différents graphes de transfert d'information. Le graphe de transfert global est obtenu, d'une part, par le nommage correct des sources et puits des graphes des différents composants et, d'autre part, par le préfixage des noms de modules avec le nom du composant. Ceci permet d'isoler les différentes références d'un même type de composant dans une description structurelle.

L'étude présentée dans ce chapitre a donné lieu à la réalisation d'un logiciel qui, d'une part, inclue la gestion de la bibliothèque de descriptions de graphes génériques et, d'autre part, réalise la compilation des descriptions VHDL structurelles.

Conclusion

Etant donné l'intérêt des utilisations possibles d'un graphe de transfert d'information pour l'évaluation de la testabilité et l'aide à la génération du test, nous avons envisagé dans cette étude l'intégration de l'outil SATAN dans le processus de conception au travers de la compilation de descriptions de simulation sous forme de graphe de transfert d'information.

La compilation de descriptions de simulation s'est limitée, dans cette étude, d'une part, aux descriptions de type flot de données ou concurrentes et, d'autre part, structurelles du langage VHDL. Le travail d'étude réalisé dans cette thèse a ainsi recouvert la compilation de descriptions comportementales, la compilation de descriptions structurelles et la définition de capacités d'information dynamiques.

Les capacités d'information dynamiques permettent, pour une classe restreinte de descriptions, de prendre en compte de façon satisfaisante les structures séquentielles cycliques.

La difficulté de cette étude a résidé dans différents points liés à l'évolution d'un outil dont les données étaient jusqu'à présent générées manuellement par un utilisateur, vers un outil dont les données sont synthétisées automatiquement. Ainsi, de nombreux paramètres qui, liés à l'expertise de l'utilisateur, étaient déduits par l'utilisateur, devaient par cette étude être formalisés et les moyens de leur détermination spécifiés.

La formalisation de base, décrite dans le chapitre 3, introduit la notion de graphe de transfert d'information statique qui donne une compilation rigoureuse d'une description VHDL sous forme de graphe.

Les choix effectués et la formalisation introduite ont permis, d'une part, de bien cerner la signification d'un graphe de transfert d'information et, d'autre part, de mettre en évidence les possibilités de simplification et de réduction du graphe. Ces dernières tâches sont essentielles dans un but de diminution du coût des tâches ultérieures à la compilation.

Par contre, à la du calcul des mesures de testabilité tel qu'il est défini dans l'outil, et vu l'absence de différenciation entre flot de contrôle et flot de données, nous avons été amenés à introduire la notion de capacité d'information dynamique avec des opérateurs non directement définis dans le langage VHDL.

Ces opérateurs détruisent donc partiellement le formalisme de base, mais permettent une représentation correcte du flot d'information au travers de constructions séquentielles contenant des cycles. A partir de la description ainsi obtenue, il ne semble plus possible de déterminer le fonctionnement d'un écoulement au niveau de son contrôle et de ses données comme cela aurait été possible avec une description dans le formalisme de base.

Toutefois, une étude plus approfondie pourrait éventuellement permettre aussi bien d'étendre la classe des structures séquentielles cycliques correctement traduisibles que de clarifier, dans un but ultérieur d'étude de l'activation d'un écoulement, la représentation du graphe vis à vis de sa fonctionnalité en termes de commandes et de temps.

Le chapitre 5 présente la prise en compte des descriptions structurelles VHDL. Les différentes étapes de compilation se décomposent sous la forme du traitement local au graphe de transfert d'information d'un composant de la description et de la concaténation des graphes de transfert d'information. Localement, une étape de simplification permet de déterminer le graphe de transfert d'information ne comportant que les fonctionnements spécifiquement utilisés du composant. Cette étape est analogue à la simplification d'une expression concurrente, des règles de simplification au niveau du graphe y étant ajouté. Elle constitue une simplification au vu des constantes connectées en entrée du composant. Ensuite, l'analyse du réseau d'interconnexion permet de définir les transferts d'information sur les entrées et sur les sorties fonctionnelles de manière à correctement relier les graphes de transfert d'information entre eux.

En termes d'implémentation, nous avons défini et mis en place un prototype écrit en langage C et LISP du traducteur de descriptions comportementales concurrentes vers le type SATAN dont le graphe est un graphe de transfert d'information statique. L'outil de gestion de la bibliothèque de types et le compilateur de descriptions VHDL structurelles n'ont pas encore été complètement remaniés par rapport au produit développé dans le cadre du projet avec l'Aérospatiale. En effet, alors que le développement du compilateur de descriptions concurrentes a pu être effectué sans remettre en cause ou retoucher des logiciels existants, les autres phases demandent de modifier des éléments logiciels dont la complexité induirait un temps très important de réalisation.

Cette étude peut se poursuivre sur différents aspects qui concernent aussi bien le niveau de la réalisation logicielle que des études théoriques. En termes d'étude théorique il semblerait intéressant de compléter voir de redéfinir la solution proposée ici pour les structures séquentielles cycliques afin de pouvoir prendre en compte des représentations structurelles de ces structures et d'affiner le calcul des capacités dynamiques aussi bien localement que globalement sur un écoulement. Une telle étude consisterait donc à élargir les limitations que nous avons donné dans ce document.

L'utilisation du graphe de transfert d'information constitue une abstraction qui devrait permettre d'envisager des utilisations multiples de l'évaluation de testabilité. Dans cette orientation, une étude a été entamée au cours de cette année sur la possibilité d'appliquer les méthodes d'analyse de testabilité, avec le point de vue du projet SATAN, à la testabilité du logiciel.

Bibliographie

[AHO 89] A. AHO, R. SETHI, J. ULLMAN, "Compilateurs, principes et outils", Interéditions, 1989

[ARC 85] E. ARCHAMBEAU, "Testability analysis techniques : a critical survey", VLSI Systems Design, Vol.6, n°12, Decembre 1985, pp.46-52.

[ARM 88] J. R. ARMSTRONG, "Chip-level modeling with HDL's", IEEE Design & Test of Computers, Vol.5, Février 1988, pp.8-18.

[BAR 86] D.S. BARKLAY, J.R. ARMSTRONG, "A heuristic chip-level test generation algorithm", Design Automation Conference, Las Vegas, Juin 1986, pp. 257-262.

[BEE 86] F. P. BEENKER et al., "Macro-testing : unifying IC and board test", IEEE Design & Test of Computers, Vol.3, Décembre 1986, pp.26-32.

[BEN 84] R. G. BENNETS, "Design of testable Logic Circuits", Reading, MA : Addison-Wesley, 1984.

[BRG 84] F. BRGLEZ, P. POWNALL, R. HUM, "Applications of testability analysis from ATPG to critical delay path tracing", International Test Conference, Philadelphia, Octobre 1984, pp. 705-712.

[CHA 88] C. H. CHAO, F.G. GRAY, "Micro-operation perturbations in chip-level fault modeling", Design Automation Conference, Anaheim, Juin 1988, pp. 579-582.

[CHA 91] C. H. CHAO, J. R. ARMSTRONG,"VHDL Semantics for Behavioural Test Generation", International Symposium on Computer Hardware Description Languages and their Applications - CHDL'91 (Marseille), 22-24 April 1991, pp. 395-412.

[DAM 85] A. DAMMAK, "Etude de mesures de testabilité de systèmes logiques", Thèse de Docteur de l'université Paris-Sud (Orsay), juillet 1985.

[DEN 68] J. J. DENT, "Diagnostic engineering requirements", Proc. AFIPS, SJCC, 1968, pp.503-507.

[DUS 78] J. A. DUSSAULT, "A Testability measure", Proc. Proc. Digital semiconductor test symposium, Cherry Hill (N J), otobre 1978, pp.113-116.

[FUJ 83] H. FUJIWARA, T. SHIMONO, "On the acceleration of test generation algorithms", EEE Transactions on Computers, vol. C-32, n°12, Decembre 1983, pp. 1137-1144.

[GOE 81] P. GOEL, B.C. ROSALES, "PODEM-X : An Automatic Test Generation System for VLSI Logic Structures", Design Automation Conference, Nashville, Juillet 1981, pp. 260-268.

[HUE 80] G. HUET, "Confluent Reductions : Abstract Properties ans Applications to Term Rewriting Systems", Journal of the ACM, vol. 27, n°4, Octobre 1980, pp. 797-821.

[IEEE 87] IEEE Standard VHDL, Language Reference Manual

[IVA 86] A. IVANOV, V.K. AGARWAL, "Testability measures - what do they do for ATPG ?", International Test conference, Washington, Septembre 1986, pp. 129-138.

[KRI 85] B. KRISHNAMURTHY, R.L.C. SIENG, "A new approach to the use of testability analysis in test generation", IEEE International Test Conference, Philadelphia, Novembre 1985, pp. 769-778.

[LAI 81] K.W. LAI, "Functional testing of digital systems", PhD Thesis, Computer Science Dept. TR CMU-CS-81-148, Carnegie-Mellon University, Decembre 1981.

[LAI 83] K. W. LAI, D. P. SIEWIOREK, "Functional testing of digital systems", Design Automation Conference (Miami Beach), Juin 1983, pp.207-213.

[LAL 90] R. LALEMENT, "Logique, réduction, résolution", Editions Masson, 1990.

[LEV 82] Y. H. LEVENDEL, P. R. MENON, "Test generation algorithms for computer hardware description languages", IEEE Transactions on Computers, Vol. C-31, Juillet 1982, pp.577-588.

[LEV 83] Y. LEVENDEL, P.R. MENON, "The *-algorithm : critical traces for functions and CHDL constructs", Fault-Tolerant Computing Symposium, Milano (I), June 1983, pp. 90-97.

[LIN 84] T. LIN, S.Y. SU, "Functional test generation of digital LSI/VLSI systems using machine symbolic execution technique", International Test Conference, Philadelphia, Octobre 1984, pp. 660-668

[LIN 85a] T. LIN, S.Y. SU, "The S-algorithm : a promising solution for systematic functional test generation", IEEE Transactions on Computer-Aided Design, Vol. CAD-4, Juillet 1985, pp. 250-263.

[LIN 85b] T. LIN, Y.H. SU, "VLSI functional test pattern generation - a design and implementation", International Test Conference, Philadelphia, Novembre 1985, pp.922-929.

[MUR 86] B. T. MURRAY, J. P. HAYES, "Hierarchical test generation using precomputed tests for modules", International Test Conference (Washington), Septembre 1986, pp.221-229.

[O'NE 89] M. D. O'NEIL and al, "BTG : a behavioural test generator", IFIP Symposium on Computer Hardware Description Languages and their Applications, Washington, Juin 1989, pp. 347-360.

[ROB 79] C. ROBACH, "Test et testabilité de systèmes informatiques", Thèse Docteur ès Sciences, Université de Grenoble, Juin 1979.

[ROB 84] C. ROBACH, P. MALECHA, G. MICHEL, "CATA : a computer aided test analysis system", IEEE Design & Test of Computers, Vol.1, n°2, Mai 1984, pp.68-79.

[ROB 86] C. ROBACH, S. GUIBERT, "Information-based testability measures", Silicon Design Conference (Wembley), Juillet 1986, pp.429-438.

[ROB 88] C. ROBACH, "Choix et utilité des outils d'évaluation de testabilité", TSI - Technique et Science Informatiques, Vol.7, n°2, 1988, pp.201-218.

[ROB 89] C. ROBACH, P. WODEY, "Linking design and test tools : an implementation", IEEE Transactions on Industrial Electronics, Vol.36, Mai 1989, pp.286-295.

[ROB 89-2] C. ROBACH, D. LUTOFF, "Knowledge based Functional Specification of Test and Maintenance Programs", IEEE Transactions Computer Aided Design, Vol.8, n° 11, November 1989, pp. 1145-1156.

[ROT 66] J. P. ROTH, "Diagnosis of automata failures : a calculus and a method", IBM Journal of R&D, vol. 10, Juillet 1966, pp. 278-291.

[STI 81] M. E. STICKEL, "A Unification Algorithm for Associative-Commutative Functions", Journal of the ACM, vol. 28, n°3, Juillet 1981, pp. 423-434.

[SU 81] S.Y. SU, Y. HSIEH, "Testing functional faults in digital systems described by Register Transfer Language", International Test Conference, Philadelphia, pp. 447-457, Octobre 1981.

[WIL 82] T.W. WILLIAMS, K.P. PARKER, "Design for Testability - a survey", IEEE Transactions on Computers, Vol. C-31, n°1, January 1982, pp. 2-15.

[WOD 89] P. WODEY, C. ROBACH, "ASICs functional testing from a VHDL description", Workshop : VHDL and modeling in the DOD procurement process (Washington), Juin 1989.

[WOD 90-1] P. WODEY, C. ROBACH, "Testing Complex Devices from High Level Description Languages", Test'90 Conference, London GB, Avril 1990, pp.57-71.

[WOD 90-2] P. WODEY, C. ROBACH, "Analyse de testabilité dans le processus de conception : une étude de cas", Colloque International de la Fiabilité et de la Maintenabilité, Brest, Juin 1990, pp. 502-508.

[WOD 91] P. WODEY, C. ROBACH, "Using a VHDL Description to Generate Hardware Test", International Symposium on Computer Hardware Description Languages and their Applications - CHDL'91 (Marseille), 22-24 April 1991, pp. 413-431.

Table des Matières

Chapitre 3

Graphe de Transfert d'Information Statique 67

Chapitre 4

Graphe de Transfert d'Information Dynamique 113

Chapitre 5

Compilation d'une Description vhdl Structurelle 161

ANNEXE A

Rappels de définitions sur les graphes

bipartis orientés

Définition. *Graphe biparti.* Un graphe biparti orienté est un quadruplet $G = (P, T, \alpha, \beta)$ dont les composantes sont :

- P est l'ensemble des sommets du graphe appelés **places** ;
- T est l'ensemble des sommets du graphe appelés **transitions** ;
- $\alpha \subset P \times T$ est l'ensemble des arcs des places vers les transitions ;
- $\beta \subset T \times P$ est l'ensemble des arcs des transitions vers les places ;

Nous définissons également, sur un tel graphe, les ensembles de sommets et d'arcs successeur ou prédécesseur d'un sommet de la manière suivante :

(i) $\forall t \in T, \Gamma^{+}_{t,G}(t) = \{p \in P / (t,p) \in \beta\}$ est l'ensemble des places successeurs de la transition t dans le graphe G ;

(ii) $\forall t \in T, \Gamma^{-}_{t,G}(t) = \{p \in P / (p,t) \in \alpha\}$ est l'ensemble des places prédécesseurs de la transition t dans le graphe G ;

(iii) $\forall p \in P, \Gamma^{+}_{p,G}(p) = \{t \in T / (p,t) \in \alpha\}$ est l'ensemble des transitions successeurs de la place p dans le graphe G ;

(iv) $\forall p \in P, \Gamma^{-}_{p,G}(p) = \{t \in T / (t,p) \in \beta\}$ est l'ensemble des transitions prédécesseurs de la place p dans le graphe G .◊

(v) $\forall p \in P, A^{+}_{p}(p) = \{e \in \alpha / e = (p,t), t \in T\}$ est l'ensemble des arcs successeurs d'une place ;

(vi) $\forall p \in P, A^{-}_{p}(p) = \{e \in \beta / e = (t,p), t \in T\}$ est l'ensemble des arcs prédécesseurs d'une place ;

(vii) $\forall t \in T, A^{+}_{t}(t) = \{e \in \beta / e = (t,p), p \in P\}$ est l'ensemble des arcs successeurs d'une transition ;

(viii) $\forall p \in P, A^{-}_{t}(t) = \{e \in \alpha / e = (p,t), p \in P\}$ est l'ensemble des arcs prédécesseur d'une transition.