Accélération de la simulation logique : architecture et algorithmes de LL3T

Yang Wu

THESE

présentée par

WU Yang

pour obtenir le titre de

DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 23 novembre 1988)

(Spécialité : Informatique)

Accélération de la Simulation Logique : Architecture et Algorithmes de LL3T

Date de soutenance : le 21 septembre 1990

Composition du jury :

Mr	J.P. VERJUS	Président
Mme	F. ANDRE	Rapporteurs
Mr	G. CAMBON	
Mme	D. BORRIONE	Examinateurs
Mr	G. MAZARE	
MR	G. MICHEL	

Thèse préparée au sein du Laboratoire de Génie Informatique (IMAG/LGI)

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

46 avenue Felix Viallet
38031 GRENOBLE cedex

Tél. : 76.57.45.00

Année universitaire 1989

Président de l'Institut :
Monsieur Georges LESPINARD

Professeurs des Universités

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

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

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

Chercheurs du C.N.R.S

DIRECTEURS DE RECHERCHE CLASSE 0

LANDEAU	Ioan
NAYROLLES	Bernard

Directeurs de recherche 1ère Classe

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

Directeurs de recherche 2ème Classe

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

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

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

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

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

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

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

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

Laboratoires extérieurs :

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

Situation particulière

PROFESSEURS D'UNIVERSITE

DETACHEMENT

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

SURNOMBRE

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

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. NEMOZ Alain

Année Universitaire 1988 - 1989

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

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

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

PROFESSEURS de 2ème Classe

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

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

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

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

PROFESSEURS de 2ème classe

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

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

PROFESSEURS DE PHARMACIE

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

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

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

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

PROFESSEURS 2ème CLASSE

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

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

à Dan

Remerciements

Je tiens à remercier

Monsieur G. MAZARE, professeur à l'ENSIMAG, qui m'a accueilli dans son équipe de recherche et qui m'a encadré tout au long de l'élaboration de cette thèse,

Monsieur G. MICHEL, directeur de l'agence Etudes de Réseaux Temps Réel d'APTOR, qui m'a fourni d'excellentes conditions de recherche, et qui m'a encouragé au cours de la préparation de cette thèse,

Monsieur J.P. VERJUS, professeur à l'ENSIMAG, directeur de l'IMAG, qui me fait l'honneur de présider ce jury de thèse,

Madame F. ANDRE, professeur à l'université de Rennes, et Monsieur G. CAMBON, professeur à l'université de Monpellier, qui ont accepté d'être rapporteurs de cette thèse,

Madame D. BORRIONE, professeur à l'université Joseph Fourier, qui a accepté de participer à ce jury.

Je tiens également à remercier

Messieurs Y. Ansade, C. Diot, E. Flamand, M. Kossa, G. Lorenzi, P. Périnet et Madame P. Prost, pour leur aide précieuse et pour les conseils nombreux qu'ils m'ont adressés lors de la rédaction de cette thèse.

Résumé

Cette thèse présente la conception d'un accélérateur matériel dédié à la simulation de circuits intégrés. Sur cet accélérateur sont développés un ensemble de logiciels constituant un environnement intégré de simulation.

Nous y discutons tout d'abord des concepts de base de la modélisation des circuits intégrés, de la simulation logico-fonctionnelle, de la simulation de pannes, des langages de description du matériel, ainsi que des techniques d'accélération de la simulation de circuits intégrés.

Nous présentons ensuite la structure générale de l'accélérateur. Il est basé sur une architecture parallèle : un réseau en anneau sur lequel sont disposées des unités de simulation, où chaque unité de simulation est composée de trois microprocesseurs exécutant trois tâches respectivement.

L'ensemble des logiciels implémentés sur cet accélérateur est présenté. Le simulateur réalise ainsi la simulation multi-niveaux (porte logique, fonctionnel, et interrupteur) et la simulation de pannes. Des outils de compilation permettent l'utilisation des langages de description du matériel pour modéliser les circuits intégrés de manière structurelle et fonctionnelle.

Enfin, différentes stratégies de parallélisation de la simulation ainsi que plusieurs algorithmes de simulation adaptés aux différents niveaux d'abstraction sont étudiés.

Mots-clefs

Accélérateur matériel, Architecture parallèle, Simulation logique, Simulation fonctionnelle, Simulation multi-niveaux, Simulation de pannes, Conception assistée par ordinateur, Circuit intégré, Langage de description du matériel, Compilation, CAO, HDL, VLSI.

Abstract

This thesis presents the design of a hardware accelerator dedicated to the simulation of integrated circuits. A suite of software has been developed, to form an integrated simulation environment.

The thesis commences with a discussion of some of the fundamental concepts of integrated circuit modeling, logic and functional simulation, fault simulation, hardware description languages, and acceleration techniques used in integrated circuit simulation.

This is followed by a presentation of the architecture and the realization of the accelerator. Based on a parallel architecture, the accelerator comprises a ring network on which are disposed the simulation units. Each unit is composed of three microprocessors each of which executes a specific task.

There is then a study of the software implemented on the accelerator. Both multi-level and fault simulation are realized. The multi-level simulation is conducted at three levels, gate, functional, and switch. The compilation tools allow the use of hardware description languages to model integrated circuits with respect to either structure or function.

Finally, different parallelization strategies of the simulation as well as simulation algorithms adapted to different abstraction levels are given.

Key words

Hardware accelerator, Parallel architecture, Logic simulation, Functional simulation, Multi-level simulation, Fault simulation, Computer aided design, Integrated circuit, Hardware description language, Compilation, CAD, HDL, VLSI.

Plan général

Chapitre 1

Introduction générale

Plan du chapitre 1

Préambule

LL3T est un simulateur logico-fonctionnel basé sur une architecture parallèle.

Les transputers, microprocesseurs RISC 32 bits d'INMOS, constituent l'élément de base de LL3T. Organisés par groupes de 3, en unités autonomes de simulation, ils forment un ensemble de 1 à 128 noeuds disposés le long d'un anneau.

Le parallélisme intervient à deux niveaux :

- A l'intérieur de l'unité de simulation, trois tâches s'exécutent de manière concurrente.
- Les unités de simulation fonctionnent en parallèle, et peuvent communiquer à travers l'anneau.

L'objectif du développement de LL3T est :

- de réaliser un simulateur accéléré grâce à l'utilisation massive du parallélisme;
- de construire autour du simulateur un environnement intégré comprenant les outils de pré-traitements et de post-traitements, ainsi que les gestionnaires de la base de données et des bibliothèques de simulation.

Le travail présenté dans cette thèse est le fruit de plusieurs années de recherches collectives. L'ensemble du projet LL3T a été supporté par la société APTOR.

Mr Y. Ansade a développé le noyau du simulateur et la partie post-traitement. Mr P. Périnet a fait des études sur l'intégration des nouveaux langages dans LL3T et a pris la suite du développement du générateur de code. Mr J. Rarivomanana s'est attaché à la réalisation de la simulation de pannes et à l'intégration de LL3T sur Apollo DN. L'auteur s'est chargé de la partie pré-traitement, notamment les compilateurs, et de la gestion de base de données et de bibliothèques de simulation.

1.1. Conception de circuits et simulation

1.1.1. Généralités

Les outils de la simulation de circuits intégrés permettent de modéliser des circuits digitaux, d'appliquer des stimuli sur ces circuits avec des signaux d'entrée, et de prédire leur comportement.

La conception d'un circuit VLSI ne peut s'effectuer qu'en plusieurs étapes hiérarchiques (prenons l'exemple de la conception descendante), chacune correspondante à un niveau d'abstraction :

• le niveau système : Pour ce qui concerne les aspects statiques du circuit, il faut spécifier l'architecture du circuit, les modules principaux, la structure de données et les ressources globales. Du côté des aspects dynamiques, il faut analyser la circulation des flots de données, la gestion des tâches, et les communications entre les processus.

• le niveau logique : La conception logique consiste en la décomposition du circuit en blocs fonctionnels, registres, portes logiques ou transistors. L'étude du comportement dynamique du circuit est plus approfondie et très répétitive. D'autre part, il est nécessaire de concevoir des séquences de tests capables de détecter les produits défectueux du circuit.

• le niveau électrique : Cette étape permet d'étudier et d'analyser les caractéristiques électriques du circuit et de ses composants. La validation électrique du circuit est réalisé par la résolution des équations, afin de calculer électriquement l'évolution du circuit en fonction du temps.

• le niveau physique : Après la validation comportementale, logique, et électrique, le circuit doit être implémenté selon une technologie sélectionnée. Un travail important à ce niveau est la conception de la structure physique et des dessins de masques : le placement, le routage et le tracé.

Il est constaté que toutes les étapes, à l'exception de la conception des dessins de masques, font intervenir les techniques de simulation aux différents niveaux d'abstraction tout au long de la conception du circuit.

1.1.2. Simulation

Depuis les années soixante, les simulateurs jouent un rôle de plus en plus important dans le domaine de la conception des circuits intégrés. Les

simulateurs procurent des avantages pour la vérification du fonctionnement et les analyses du comportement temporel des circuits :

• Les technologies actuelles permettent la réalisation de circuits intégrant des millions de transistors. Un tel circuit implanté sur le silicium ne peut être vérifié simplement par le prototype.

• La simulation fait appel à la puissance de calcul des machines. C'est une méthodologie permettant de réduire considérablement le temps, et par conséquent le coût, de la conception de circuits.

• Les simulateurs offrent un grand degré de précision. Pour une même conception, le comportement du circuit varie en fonction des circonstances particulières et des conditions réelles. Un prototype ne permet pas une étude exhaustive du fonctionnement du circuit. Les simulateurs permettent en revanche de prédire quel fonctionnement et quelles tolérances peut avoir un circuit dans les environnements supposés. De plus la simulation assure la visibilité des états internes du circuit.

• La simulation est une méthodologie déterministe. Les analyses statiques des circuits intégrés butent souvent sur des problèmes épineux. Prenons l'exemple de l'initialisation d'un circuit; le circuit entre dans un état inconnu après la stabilisation de ses noeuds internes. Cette stabilisation est aléatoire et il est pratiquement impossible d'en énumérer toutes les possibilités. Ce problème ne peut être résolu que par l'utilisation du simulateur ayant une modélisation adéquate.

• La détection des circuits défectueux reste un problème critique. Un simulateur de défauts est capable d'examiner le comportement des circuits en présence des défauts physiques, et de répondre au concepteur si un défaut est détectable sous une séquence de tests. La simulation de défauts est particulièrement importante car aujourd'hui le test du circuit coûte très cher.

Néanmoins le développement des simulateurs est relativement difficile, car les critères sont sévères :

• la vitesse : La vitesse de simulation affecte toutes les étapes de la conception d'un circuit. Un retard dans une conception moderne augmente sensiblement le coût du produit, et diminue sa compétitivité face à la concurrence.

• la capacité : Les techniques évoluent constamment. Un simulateur de grande capacité peut répondre à la croissance de la complexité des circuits, et avoir une longue durée de vie.

• la précision : Le simulateur doit fournir des résultats précis. Cela exige une bonne modélisation de circuits, de signaux et de temps.

• le coût : Un coût économique permet une utilisation massive des simulateurs dans la conception.

• l'abstraction : Un langage évolué qui permet une description modulaire, générique, et hiérarchique est nécessaire. Un simulateur aux multi-niveaux facilite la tâche de conception en étapes.

• l'extensibilité : Un simulateur doit autoriser la simulation de différentes technologies et suivre les évolutions de ces technologies.

• l'adaptation : L'interface utilisateur doit être personnalisable selon les circonstances de simulation et les préférences de l'utilisateur.

• l'intégration : Les outils de conception de circuits représentent un mélange de différentes méthodologies. L'environnement de la conception intègre non seulement des simulateurs mais aussi d'autres outils ou systèmes, qui partagent une base de données commune. Un simulateur doit être facilement intégré dans un tel environnement.

• la compatibilité : Le simulateur doit avoir une bonne interface et offrir un format standard pour les échanges de données entre différents environnements aux niveaux des sources et des résultats. Une bonne compatibilité permet aux simulateurs d'avoir moins de limitation d'utilisation.

Les qualités clefs des simulateurs sont la vitesse et la capacité. Un circuit trop complexe peut rendre le simulateur inexploitable : le temps de calcul est trop long ou la taille du circuit dépasse la capacité du simulateur.

1.2. Accélération de la simulation

1.2.1. Accélération et parallélisme

La conception du circuit VLSI d'aujourd'hui est caractérisée par les faits suivants :

• Un grand nombre de simulations répétitives sont nécessaires pour mettre au point la conception du circuit.

• La validation des séquences de tests coûte beaucoup plus cher que la vérification du fonctionnement du circuit. La simulation de défauts consomme un temps de calcul énorme.

• Plus le circuit est complexe, plus le coût de la simulation logique et celui de la simulation de défauts sont élevés par rapport au coût total de la conception. Il y a de moins en moins d'interventions humaines.

• Le temps de ces simulations s'accroît selon les algorithmes d'un ordre de grandeur de n^2 ou n^3, n étant la complexité du circuit.

La tendance est claire : l'exigence de simulateurs plus performants; d'où l'idée de développer des machines spécifiques dites accélérateurs de simulation. L'utilisation répétitive d'un accélérateur rend son coût faible, ce qui lui ouvre la voie des différentes applications de la simulation.

Le parallélisme introduit dans les accélérateurs permet d'améliorer les performances de simulation de manière impressionnante. L'exploitation du parallélisme dans le simulateur repose sur les propriétés suivantes :

• la nature parallèle existant dans le matériel qui supporte le simulateur : l'architecture multiprocesseur, le mécanisme de traitements parallèles, la gestion des ressources distribuées;

• la simultanéité et la concomitance des différents traitements dans les algorithmes : les processus parallèles, les tâches coexistantes, les calculs en pipeline;

• le parallélisme inhérent au circuit à simuler : la décomposition du circuit en sous-circuits simulés parallèlement de façon synchrone ou asynchrone;

• la multiplicité de cas et de circonstances de la simulation : la simulation du même circuit avec différents stimuli, défauts, ou conditions physiques.

1.2.2. Accélérateurs à base de processeurs généraux

Les accélérateurs spécifiques sont des machines spécialement développées et optimisées autour des algorithmes de simulation. Leur utilisation est souvent limitée à une application particulière. Une autre approche consiste à développer des machines d'accélération construites à base de microprocesseurs généraux performants comme les processeurs RISC, solution intéressante en raison de son faible coût économique et d'une puissance de calcul élevée grâce à l'exploitation du parallélisme.

Certains efforts ont été faits pour étudier les accélérateurs à base de microprocesseurs généraux. L'objectif est de développer des accélérateurs performants, qui conviennent à l'utilisation en quantité des outils de simulation dans le domaine de la conception de circuits intégrés. Les recherches sur de tels accélérateurs se concentrent sur les problèmes suivantes :

- la stratégie du développement : La spécification d'un accélérateur basé sur l'utilisation des microprocesseurs doit assurer un développement rapide et non risqué. D'autre part, l'accélérateur doit fournir un service durable, tout en prévoyant la possibilité de s'adapter aux futurs progrès des technologies des microprocesseurs.
- l'architecture d'accélérateurs : Il faut sélectionner des processeurs ayant une grande puissance de calcul, souvent des microprocesseurs RISC, et constituer une configuration multiprocesseur propre aux tâches de simulation. Des estimations doivent être faites : combien de processeurs peuvent être intégrés dans le système, quelles sont les performances d'un processeur dans le système, et à quel degré le système peut être reconfigurable en fonction du nombre de processeurs impliqués.
- l'exploitation du parallélisme : Il est nécessaire d'analyser les algorithmes pour pouvoir apporter le parallélisme dans le noyau du simulateur. La dégradation de puissance de chaque processeur ne doit pas être trop importante lorsque le parallélisme est massif et que le nombre de processeurs dans l'accélérateur est élevé.
- les pré-traitements et les post-traitements : La faiblesse des accélérateurs spécifiques est souvent liée aux traitements autres que la simulation, du fait qu'ils ont une architecture spécialisée pour la résolution des problèmes critiques de la simulation elle-même. Cependant, les accélérateurs constitués des processeurs généraux n'ont pas cette contrainte architecturale. Il est de leur devoir d'accélérer les pré-traitements et les

post-traitements. Au niveau du pré-traitement, ils doivent pouvoir, avant tout, accélérer la compilation de circuits et de stimuli, tandis qu'au niveau du post-traitement, les accélérateurs doivent supporter l'édition, le formatage et les analyses des résultats.

• l'adaptation de différentes méthodologies : Contrairement aux accélérateurs spécifiques qui matérialisent des algorithmes particuliers et qui se limitent à une seule application de simulation, les accélérateurs basés sur les microprocesseurs généraux doivent être capables d'intégrer plusieurs types de logiciels, tels que le simulateur logique et le simulateur de pannes, dans un environnement unique. C'est à la fois une nécessité puisque un très grand nombre d'accélérateurs sont trop spécifiques et trop orientés, et une optimisation puisqu'un accélérateur qui couvre plusieurs étapes de conception ou plusieurs types de simulations permet de partager une base de données commune, de diminuer le nombre total d'outils de conception, et en conséquence d'assurer la qualité de la conception de circuits.

1.3. Motivation du développement de LL3T

1.3.1. Diverses exigences

◊ Exigence des accélérateurs économiques

Le but du développement de LL3T est de construire un accélérateur à base de microprocesseurs généraux et de réaliser un environnement de simulation, afin de répondre aux besoins des accélérateurs économiquement avantageux :

• De grosses sociétés ont développé pour leurs propres besoins des accélérateurs très puissants basés sur des matériels gigantesques. Ce genre d'accélérateurs est souvent très spécifique et destiné à des utilisations internes.

• Des accélérateurs multiprocesseurs ont été commercialisés, dont les algorithmes sont souvent câblés ou microprogrammés, avec une architecture parallèle ou pipeline de haut niveau. Ces accélérateurs sont coûteux et destinés à de gros utilisateurs.

• Il existe des systèmes de simulation implémentés sur des machines générales. Mais leurs performances varient selon les machines sur lesquelles les simulateurs sont implémentés.

• Pour combler la lacune se trouvant entre les gros accélérateurs trop dispendieux et les simulateurs trop modestes, des accélérateurs économiques ont été développés qui permettent un usage massif des accélérateurs de simulation.

◊ Exigence des accélérateurs à l'usage multiple

Le développement de LL3T est orienté vers un domaine large de simulation. LL3T permet d'effectuer plusieurs types de simulation :

• A l'opposé de certains accélérateurs spécifiques, qui s'adaptent à un ou quelques algorithmes de simulation, LL3T a pour objectif d'accélérer la simulation logique avec une variété d'extensions : la modélisation aux niveaux porte logique, fonctionnel et transistor. D'autre part, il vise à réaliser plusieurs modes de simulation, variables selon la manière de la discrétisation du temps (différents types de retards) et de la modélisation des signaux (les états logiques avec différentes finesses : 4 ou 15 valeurs logiques)

• Sans risque de dégradation de performances causée par la limitation

architecturale, LL3T tente de réaliser également la simulation de pannes. La simulation normale et la simulation de pannes partagent la même description de circuits, et les informations compilées.

• Certains accélérateurs nécessitent une machine hôte ou un coprocesseur pour les pré-traitements et les post-traitements, à cause de leur architecture trop spécialisée. En revanche LL3T, à base de microprocesseurs généraux, est un système complet capable d'effectuer les pré-traitements (la saisie des sources, la compilation, l'édition de liens) et les post-traitements (l'édition des résultats, la gestion des fichiers de capture).

◊ Exigence des accélérateurs faciles à utiliser

Le développement de LL3T s'efforce d'apporter des facilités aux tâches de simulation :

• Comme l'accélérateur est à base de microprocesseurs généraux, les communications accélérateur-hôte peuvent être limitées au minimum : l'accès aux fichiers et à la console. Cette architecture est capable de supporter un système autonome qui incorpore toutes sortes d'outils auxiliaires (les compilateurs, l'éditeur, et le gestionnaire de bibliothèques) et une base de données qui lui est propre afin de fournir un environnement intégré de simulation.

• Les accélérateurs purs considèrent la simulation comme un calcul intensif. Par conséquent, trop d'interactions entre les accélérateurs et la machine hôte au cours du calcul ralentissent sensiblement la simulation. LL3T, à base de microprocesseurs, peut permettre une simulation très interactive, du fait qu'il est possible de configurer l'accélérateur en sorte qu'un processeur se charge de la gestion des données de résultats partiels et contrôle le déroulement de la simulation.

• Pour faciliter l'utilisation des accélérateurs, LL3T est prévu d'être configurable selon divers besoins. Il est capable d'intégrer des centaines de microprocesseurs lorsque le parallélisme est massif, ou quelques microprocesseurs pour être incorporé, sous forme de carte d'extension, dans un ordinateur individuel.

1.3.2. Intérêts scientifiques

◊ Matériel

L'intérêt est de développer un accélérateur de simulation à l'aide de

microprocesseurs généraux et d'analyser les performances d'accélération apportées par les microprocesseurs.

Certains accélérateurs dotés de processeurs spécialement conçus présentent des inconvénients :

- Le coût de développement est trop élevé.
- Le noyau du matériel est trop orienté vers quelques algorithmes spécifiques.
- L'évolution d'une machine spécifique est moins rapide que celle de processeurs généraux.

Dans le but de bénéficier de la puissance des microprocesseurs actuels, LL3T utilise des transputers, famille de microprocesseurs RISC 32 bits, comme le support matériel. Cela permet d'une part d'obtenir de bonnes performances, d'autre part de standardiser les composants du matériel.

◊ Architecture

Une autre tentative du développement de LL3T est de découvrir une architecture adéquate à la simulation.

Différentes topologies architecturales comme l'hypercube, le réseau cellulaire et les processeurs SIMD ont été adoptées par des accélérateurs. La synchronisation et les échanges de données s'effectuent soit par une matrice de communication, soit par une mémoire commune, soit par des bus à l'aide de buffers.

LL3T recherche une autre possibilité : un accélérateur basé sur une architecture en anneau, dont chaque noeud est une unité de simulation.

Le composant de base de LL3T est le transputer, disposant de quatre liens de communications. Chaque transputer gère une tâche, en utilisant une mémoire qui lui est propre. Une unité de simulation est constituée de trois transputers qui réalisent trois tâches concurrentes : deux tâches d'évaluation et une tâche de gestion d'échéancier local et de communications entre les unités de simulation. Un simulateur basé sur une telle architecture présente une originalité qui justifie des recherches poussées.

◊ Parallélisme

L'exploitation du parallélisme est un problème pointu dans le développement des accélérateurs de simulation.

Dans LL3T le parallélisme intervient aux deux niveaux. La gestion des trois tâches dans une unité de simulation, l'exploitation du parallélisme entre les unités de simulation et la synchronisation de l'anneau sans processeur de contrôle, nécessitent des études approfondies. De plus, il est important d'estimer l'efficacité de chaque processeur, l'équilibrage de la charge de travail des tâches réparties sur les processeurs, et les performances globales lorsque le parallélisme est à haut degré.

◊ Programmation

LL3T utilise le langage parallèle Occam 2 qui permet la programmation concurrente. La programmation concurrente ne peut être réalisée par un langage basé sur les simples notions de sous-routine et de coroutine. Le vrai langage concurrent porte sur le concept de processus. Il s'agit donc, de la communication, de la synchronisation et de la coopération entre les processus. L'exécution d'un programme de tel langage est souvent caractérisée par une nature dynamique et non déterministe. La simulation de circuits nécessite l'utilisation d'un certain nombre de tâches avec une potentialité de parallélisme massif, ce qui convient à la philosophie de la programmation concurrente.

L'entité de base du langage Occam 2 est le processus. Les relations entre les processus peuvent être séquentielles, parallèles, alternatives, etc. Chaque processus peut se composer de plusieurs sous-processus. Entre les processus, les échanges de données s'établissent sur des canaux de communication.

Ce type de langage peut être très avantageux pour la programmation du simulateur parallèle.

◊ Algorithme

Le développement de LL3T tâche d'améliorer et d'enrichir les algorithmes de simulation.

LL3T utilise l'algorithme de l'échéancier. Plusieurs efforts ont été faits.

D'une part, le parallélisme est introduit dans l'algorithme. L'utilisation des échéanciers locaux permet de partitionner le circuit à simuler en sous-circuits pouvant être distribués sur les unités de simulation qui fonctionnent en parallèle. D'autre part, LL3T intègre des modèles de différents niveaux d'abstraction : des modèles fonctionnels et des modèles transistors. Il est souhaitable de trouver des solutions optimales pour que ces modèles étendus soient évalués de façon efficace par le mécanisme de l'échéancier. En plus, LL3T essaie d'étendre l'algorithme de l'échéancier dans le domaine de la simulation de pannes. Un tel système de simulation représente une grande complexité et une grande difficulté algorithmique.

Organisation de la thèse

Le chapitre 2 fera respectivement une étude sur la simulation de circuits intégrés d'une part, et sur les langages de description du matériel d'autre part. En ce qui concerne la simulation, les problèmes de modélisation seront présentés, et les différentes méthodologies de la simulation discutées. Ces méthodologies de simulation font appel à l'utilisation des langages de description du matériel. La présentation des langages de description du matériel ne concernera qu'un certain nombre de concepts de base et s'orientera vers la simulation de circuits intégrés.

Le chapitre 3 sera consacré aux techniques d'accélération et présentera en particulier certains accélérateurs de simulation récemment développés. La discussion commencera par l'exposé des exigences des techniques d'accélération. Ensuite seront analysées les différentes accélérations algorithmiques et matérielles. Plusieurs stratégies pour la réalisation des accélérateurs seront étudiées à partir de ces analyses. Le chapitre se terminera par une présentation des machines de simulation afin de dresser un bilan sur les accélérateur actuels.

Le chapitre 4 abordera l'architecture matérielle et logicielle, et l'organisation générale de l'ensemble des modules de LL3T. Les fonctionnalités de base et la structure interne de chaque module seront détaillées. Le rôle principal de ces modules dans l'ensemble des logiciels de LL3T et leurs relations seront décrits. L'accent du chapitre sera mis sur le noyau du simulateur et l'ensemble des outils de compilation.

Le chapitre 5 présentera les algorithmes et les principes de base de la réalisation de LL3T. Ce chapitre sera consacré essentiellement à la présentation des aspects méthodologiques et algorithmiques, afin d'exposer les idées directrices du développement de LL3T. Les algorithmes et les méthodologies présentés concerneront aussi bien le simulateur que les outils de compilation. Ils représentent les stratégies et les tactiques principales de la réalisation de l'ensemble des logiciels de LL3T.

Le chapitre 6 évaluera les performances de différentes versions de LL3T. En conclusion, des considérations et des reflexions sur le développement de LL3T seront développées.

Chapitre 2

Simulation de circuits intégrés et Langages de description du matériel

Plan du chapitre 2

Introduction

Ce chapitre sera consacré à la présentation de différents types de simulations et de langages de description du matériel.

D'abord, nous présenterons le problème de la modélisation, qui est fortement liée aux niveaux d'abstraction des propriétés physiques de circuits intégrés.

La discussion autour de la simulation normale sera concentrée sur le niveau logique. Des algorithmes de simulation basés sur la méthode de l'échéancier et des variations seront développés.

Deux extensions de la simulation logique, celles aux niveaux fonctionnel et interrupteur, seront soulignées :

- En ce qui concerne la modélisation des transistors MOS, il sera prouvé que l'utilisation des modèles bidirectionnels est indispensable. Partant de ce principe, un algorithme de maximisation sera étudié.
- Pour les modèles fonctionnels, un algorithme de l'échéancier sera présenté.

Parmi trois principales méthodes de la simulation de défauts, l'approche concurrente sera détaillée, à l'aide d'un algorithme basé sur un mécanisme de l'échéancier.

Le deuxième grand sujet du chapitre concernera les langages de description du matériel. Ce sont des langages spécifiques pour résoudre des problèmes spécifiques dans le domaine de CAO de circuits intégrés.

Afin d'appliquer des critères appropriés à la mesure de ces langages, nous essayerons de développer d'abord des concepts de base et d'étudier les caractéristiques des langages de description du matériel.

Les propriétés fondamentales des descriptions du matériel qui existent dans les langages de description du matériel seront analysées. Leurs relations seront également clarifiées. L'accent sera mis sur la résolution des problèmes suivants :

- les relations entre les caractéristiques des langages et leurs

domaines d'application, en particulier la simulation et le test;

- les rapports entre la structuration de données dans les langages et les capacités descriptives des langages vis-à-vis de différents aspects du matériel concerné;
- l'influence mutuelle entre les propriétés sémantiques des langages et l'implémentation des outils de CAO.

2.1. Différentes méthodologies de simulation

2.1.1. Simulation

Le simulateur est un outil informatique qui permet de prédire les caractéristiques logiques et électriques d'un circuit en fonction d'un ensemble de signaux d'entrée dans un environnement opérationnel. Son rôle principal est de modéliser le circuit selon certains niveaux d'abstraction, de vérifier son fonctionnement logique, de prédire son comportement temporel, d'étudier les flots de données circulant à l'intérieur du circuit, et de valider la conception du circuit.

Le simulateur de défauts détermine le comportement de circuits en présence des conditions défectueuses. Son objectif est de valider des séquences de tests, de trouver des défauts non détectables, et d'examiner la redondance de tests.

2.1.2. Niveaux d'abstraction

◊ Abstraction

Un problème essentiel de la simulation est lié à l'abstraction. L'architecture du circuit, les caractéristiques temporelles des composants du circuit, les propriétés des signaux, et l'environnement physique du fonctionnement doivent être modélisés avec un certain niveau de description, basée sur un langage évolué. La complexité et les difficultés d'analyse d'un circuit peuvent être contournées par des études hiérarchiques et progressives en plusieurs niveaux d'abstraction, chaque niveau associé à une méthodologie. En effet, des simulateurs sont développés, basés sur différents niveaux d'abstraction.

◊ Simulateurs au niveau système (*system level, algorithmic level*)

C'est un niveau d'abstraction architecturale et algorithmique. Ce genre de simulateurs permet de créer et de gérer des activités comme des processus ou des tâches dynamiques. Le simulateur vérifie les communications et la synchronisation entre les processus, la gestion des tâches, l'allocation et l'accès aux ressources partagées, la cohérence liée au parallélisme, etc. SIMULA est un simulateur de ce genre [DAH 66].

◊ Simulateurs au niveau transfert de registre (*register transfer level*)

Le langage de description définit certains composants comme des registres, des bascules et des horloges. La structure du circuit à simuler est décrite en terme de composants déclenchés par transition ou par niveau logique des signaux. L'opération de base est le transfert de données (*assignment*) et l'activation des composants par certaines conditions (*guard*). Le langage CDL et les langages DDL, ADL sont présentés respectivement dans [CHU 74] et [BOS 87].

◊ Simulateurs au niveau fonctionnel (*functional level, behavioral level*)

Le circuit se compose d'un ensemble de blocs algorithmiques constitués d'une structure de contrôle (des instructions ou des procédures) et des données associées (des signaux et des variables). Chaque bloc peut effectuer des opérations simples ou complexes et avoir un comportement distinct [ABR 88b].

Le niveau fonctionnel et le niveau transfert de registre représentent différents degrés d'abstraction. La simulation au niveau transfert de registre vise à optimiser l'objectif de la conception, tandis que la simulation au niveau fonctionnel a pour but de vérifier le fonctionnement du circuit, d'analyser les caractéristiques temporelles des composants, et d'étudier les effets de la présence des défauts dans le circuit.

◊ Simulateurs au niveau logique (*logic level, gate level*)

Le circuit est considéré comme un ensemble de portes logiques interconnectées par des équipotentielles. Les signaux sont représentés par des valeurs logiques discrètes. TEGAS [THO 80] et YSE [PFI 86] sont les simulateurs basés sur ce niveau d'abstraction.

◊ Simulateurs au niveau interrupteur (*switch level*)

Les transistors sont les composants élémentaires du circuit MOS. Les phénomènes propres aux transistors sont la nature bidirectionnelle des transistors, la mémorisation due à la charge dynamique, le partage des charges, et le ratio de résistivité entre les transistors. Le simulateur au niveau interrupteur modélise les transistors comme des interrupteurs résistifs à travers lesquels circulent des signaux discrétisés. Le simulateur

MOSSIM II [BRY 84] est fondé sur ce principe.

◊ Simulateurs au niveau électrique (*electrical level, circuit level*)

La description du circuit est basée sur un ensemble d'éléments électriques actifs ou passifs interconnectés. Les signaux sont représentés par des valeurs continues. L'évolution du circuit est calculée par la résolution des équations. Le simulateur SPICE [VLA 81] est un exemple typique de ce type de simulateurs.

◊ Simulateurs au niveau physique (*physical level*)

La conception moderne utilise souvent des composants commerciaux, comme des microprocesseurs, des unités graphiques, etc. Un circuit constitué de ces composants ne peut être décrit par un langage quelconque, car l'architecture et le fonctionnement temporel d'un composant commercial sont parfois indisponibles ou trop ambigus. Pour assurer l'utilisation de ces composants dans une conception, un simulateur spécial doit être employé, muni d'une interface permettant de brancher des composants réels. Le simulateur calcule le reste du circuit en interagissant avec ces composants [CAR 87a].

◊ Simulateurs aux multi-niveaux (*multi-level, mixed level*)

En réalité, de nombreux simulateurs intègrent plusieurs niveaux de description. INTSIM [JOH 80] est un simulateur aux trois niveaux : fonctionnel, logique et électrique. MOZART [GAI 88] est un simulateur de pannes capable de simuler les circuits décrits aux quatre niveaux : interrupteur, logique, fonctionnel et transfert de registre.

2.1.3. Simulateurs logico-fonctionnels

Le simulateur logique est un outil élémentaire destiné à la conception de circuits. Son objectif est de :

- vérifier le fonctionnement logique de circuits.
- étudier le comportement temporel de circuits.

La philosophie de la simulation logique offre un niveau d'abstraction relativement indépendant des technologies de VLSI. La notion de portes

logiques est très extensible : Un transistor (vers le niveau plus bas) ou un registre (vers le niveau plus haut) peuvent être intégrés dans un environnement de simulation logique.

Un simulateur logico-fonctionnel inclut, en plus des portes logiques, les modèles fonctionnels.

Pour pouvoir s'adapter à la technologie MOS, une extension au niveau interrupteur est souvent adoptée dans les simulateurs tels que TEGAS [THO 80].

Un simulateur logique avec des extensions procure des avantages considérables :

• Une simulation peut s'effectuer pendant que la conception n'est pas complète. La description fonctionnelle néglige la structure interne des composants du circuit.

• Le temps de calcul et la mémoire consommée par la simulation peuvent être réduits par l'utilisation des descriptions de plus haut niveau d'abstraction pour modéliser certaines parties du circuit qui ne nécessitent pas d'analyse approfondie.

• La simulation simultanée de plusieurs niveaux de description d'un même composant du circuit et la vérification d'équivalence sont rendus possibles. Cela facilite l'utilisation de la méthodologie de la conception hiérarchique.

2.1.4. Simulateurs de pannes

Le simulateur de pannes est un outil de la simulation de défauts qui fonctionne au niveau logique.

Une panne est la manifestation logique d'un ou plusieurs défauts physiques. L'analyse de défauts s'effectue en général par la simulation de pannes, puisque, avec la complexité des circuits intégrés, il n'existe pas de techniques efficaces permettant la détection directe de défauts physiques du circuit. La détection de défauts consiste à propager les pannes, qui représentent l'effet logique des défauts de fabrication, sur les points de test du circuit, afin de visualiser ces pannes. C'est la raison pour laquelle le simulateur de pannes est un outil très important du domaine de la simulation de défauts.

Par conséquent, le but de la simulation de pannes est de savoir si une panne à l'intérieur d'un circuit peut produire, sous une séquence de stimuli spécifiques, des erreurs détectables aux points de tests ou aux sorties primitives du circuit. Ses rôles principaux sont :

- la validation d'une séquence de tests;
- la détermination des pannes non détectées par la séquence de tests;
- la construction des dictionnaires de pannes.

2.2. Modélisation logico-fonctionnelle

2.2.1. Description des circuits

◊ Modèles des portes logiques

Les portes logiques sont des objets élémentaires à comportement logique. Elles permettent de réaliser toute fonction logique.

Les portes logiques classiques sont l'inverseur, le OU, le ET, le NON-OU et le NON-ET.

Il existe en plus des portes OUEX (OU exclusif), NON-OUEX, et des portes spéciales permettant la modélisation des circuits intégrés de différentes technologies.

L'entrance d'une porte est le nombre d'entrées de la porte. La sortance d'une porte est le nombre de portes connectées à la sortie de cette porte. Une porte logique est paramétrée par la fonction logique, l'entrance, et la sortance.

◊ Modèles des transistors MOS

Un transistor MOS est modélisé comme un interrupteur résistif qui comprend trois états de base : ouvert, fermé ou inconnu. Des simulateurs logiques considèrent que les interrupteurs résistifs sont des modèles étendus de la description logique : Les portes classiques sont les éléments unidirectionnels, tandis que les interrupteurs sont les éléments bidirectionnels. Cela n'exclut pas l'utilisation des éléments unidirectionnels pour modéliser les transistors de charge et les transistors de types buffers à trois états.

Une connexion est modélisée comme un noeud capacitif qui peut mémoriser un état logique.

◊ Modèles fonctionnels

Au lieu de représenter le circuit par des portes primitives de manière structurelle, l'utilisateur peut décrire des blocs fonctionnels (des boîtes noires) afin d'avoir une abstraction de plus haut niveau. Chaque bloc peut

comprendre un algorithme et des données internes. Ce niveau de description consiste à décrire ce que les boîtes doivent faire, sans dire comment elles le font.

◊ Structuration des descriptions

La plupart des simulateurs et leurs compilateurs acceptent la description structurée. La structuration de données procure les avantages suivants :

• Le circuit se divise en sous-circuits, chacun ayant sa propre interface et pouvant être ensuite divisé de façon récursive. Cela garantit une bonne décomposition et une bonne modularité.

• Chaque circuit ou sous-circuit est paramétrable. Il représente un modèle générique qui définit les caractéristiques communes d'un ensemble de composants.

• Chaque circuit peut être considéré comme une cellule de base. Des bibliothèques contenant des cellules de base peuvent être créées et intégrées dans l'environnement de simulation.

• Il existe des bibliothèques indépendantes qui contiennent des modèles des circuits commerciaux. Ces bibliothèques sont spécialement conçues et compilées en code très efficace. Le simulateur interagit avec les bibliothèques à travers l'interface spécifique. Cela accélère d'une part la simulation et standardise d'autre part la modélisation des circuits.

• Le circuit hiérarchique est mis à plat par la compilation avant d'être simulé. A l'inverse, un simulateur hiérarchique fonctionne directement sur une description hiérarchique [ROG 87]. Cette méthodologie réduit la mémoire nécessaire pour la simulation. La structure hiérarchique du circuit étant conservée, la vérification d'équivalence et la substitution dynamique des modules du circuit sont rendus plus faciles.

2.2.2. Codage des signaux

◊ Modélisation

Les signaux électriques sont modélisés en états logiques. Un état logique se compose de deux informations : le niveau logique et la force. Le niveau logique représente la tension du signal tandis que la force caractérise l'intensité du courant du signal.

◊ Niveaux logiques

En ce qui concerne les valeurs statiques, les 0 et 1 sont les niveaux logiques de base, qui ne suffisent que pour valider la fonction booléenne du circuit. Le niveau inconnu X et le niveau haute impédance Z sont souvent introduits dans la simulation pour compléter la modélisation.

En ce qui concerne les valeurs dynamiques, deux types de valeurs sont utilisées : les transitions et les aléas. Un aléa est un changement bref de niveau d'un signal qui devrait être stable (aléa statique), ou un bref retour à son niveau initial d'un signal qui devrait changer de niveau (aléa dynamique). Les principaux niveaux logiques sont :

- 0 : le niveau bas;
- 1 : le niveau haut;
- X : la valeur invalide ou non initialisée;
- Z : la valeur haute impédance;
- Ø : la transition indéterminée;
- $Ø_{01}$: la transition montante;
- $Ø_{10}$: la transition descendante;
- $*_{00}$: le 0-aléa statique;
- $*_{11}$: le 1-aléa statique;
- $*_{01}$: le 0-aléa dynamique;
- $*_{10}$: le 1-aléa dynamique.

◊ Forces

La force est un autre aspect de l'état logique du signal. Elle représente la puissance d'un signal généré par un élément logique. En général, la force est fonction de l'impédance et de la puissance de sortie de l'élément. Les forces les plus utilisées sont les suivantes, dans l'ordre décroissant :

- externe : la force la plus puissante qui représente l'alimentation ou la masse;
- fort : la force forte qui représente un signal généré par un élément puissant;
- résistif : la force qui représente un signal passant par un élément avec un effet d'atténuation;
- faible : la force qui représente un signal assez faible;
- très haute impédance : la force qui représente un signal mémorisé

sur une grosse capacité;

• haute impédance : la force qui représente un signal mémorisé sur une petite capacité;

• inconnue : la force inconnue ou indéterminée.

Lorsque plusieurs signaux se trouvent sur une même équipotentielle, s'il existe un signal qui a la force la plus forte, alors il éliminera les autres; s'il y a plusieurs signaux ayant des forces égales et des niveaux logiques différents, alors le niveau indéterminé X se produira.

◊ Valeurs

Certains simulateurs utilisent directement des valeurs logiques qui unifient les deux aspects des états logiques : le niveau logique et la force. A titre d'exemple, une logique à 8 valeurs est constitué de :

• 0 : zéro fort;
• L : zéro faible;
• 1 : un fort;
• H : un faible;
• Z : haute impédance;
• N : zéro ambigu entre 0, L;
• P : un ambigu entre 1, H;
• X : valeur indéterminée.

2.2.3. Représentation du temps

◊ Modélisation

Le comportement dynamique d'un composant électrique est fortement lié à ses caractéristiques temporelles. Le temps de propagation d'un signal n'est jamais instantané et nécessite une modélisation adéquate. La représentation du temps de la simulation est soit continue (en cas de simulation électrique), soit discrète. Dans la simulation logique, le temps est discrétisé et représenté par des retards.

Un retard de propagation est le temps nécessaire pour un signal de traverser un élément tel que la porte logique.

Un retard inertiel représente la durée minimale d'une impulsion pour

vaincre l'inertie d'un élément.

Les simulateurs utilisent un ou plusieurs types de modèles de retard, en fonction de la précision de simulation.

◊ Retard nul

On suppose que la propagation d'un signal à travers un élément logique est instantanée. Ce modèle est utilisé pour vérifier la fonction logique de l'élément ou pour modéliser le circuit dont le retard est totalement négligeable.

◊ Retard unitaire

On associe la même unité de temps à tous les éléments du circuit à simuler. Toutes les portes du circuit ont un retard uniforme.

◊ Retard variable

Tous les éléments du circuit à simuler ont leur propre retard. Cette modélisation permet de décrire les éléments logiques de façon plus réaliste.

◊ Retard montée-descente

Pour chaque élément logique, le temps de montée et de descente peuvent être différents. Le retard est donc caractérisé par le temps de passage de 0 à 1 et de 1 à 0. Cette modélisation peut être complétée par d'autres transitions telles que $Z \rightarrow 0$, $0 \rightarrow Z$, $Z \rightarrow 1$ et $1 \rightarrow Z$.

◊ Retard multi-valeurs

Chaque élément a son retard individuel qui est représenté par plusieurs valeurs de temps : minimale, typique, maximale, etc. Lorsque le signal d'entrée change, la sortie ne change pas avant le temps minimal et la sortie sera stable après le temps maximal.

◊ Retard inertiel

Certains éléments exigent un temps minimal pendant lequel le signal d'entrée doit rester stable pour que le signal de sortie puisse être déterminé

correctement. Le retard inertiel définit le temps minimal de réaction d'un élément logique.

2.2.4. Hypothèses de défauts

◊ Classification des défauts

La détection des défauts qui peuvent survenir dans les produits nécessite des simulateurs de défauts qui utilisent des modèles de défauts, dits hypothèses de défauts.

Les défauts peuvent se manifester au niveau physique, logique et fonctionnel :

- une défaillance : un défaut physique;
- une panne : un défaut au niveau logique qui représente l'effet d'une ou plusieurs défaillances;
- une erreur : un défaut au niveau de données qui est la manifestation d'une ou plusieurs pannes.

◊ Hypothèses de pannes

Au niveau logique, les modèles de défauts sont représentés par les hypothèses de pannes qui sont en général définies par les collages :

- le collage à 0 : une équipotentielle constamment collée à 0;
- le collage à 1 : une équipotentielle constamment collée à 1;
- le collage à Z : une équipotentielle constamment collée à Z;
- le collage ouvert : un interrupteur constamment ouvert;
- le collage fermé : un interrupteur constamment fermé;
- le court-circuit : un court-circuit entre deux équipotentielles.

2.3. Simulateurs logiques

2.3.1. Diverses méthodes

◊ Simulation par code compilé

La simulation par code compilé (*compiled-code simulation, compiled simulation*) consiste à compiler une description source et à générer un code exécutable.

Pendant l'exécution, on évalue, sans branchement ni test conditionnel, la sortie de toutes les portes en fonction de l'état précédent des entrées. L'horloge de simulation s'incrémente pas à pas, jusqu'à ce que la simulation s'achève.

Ce type de technique s'adapte bien à la simulation à retard nul ou unitaire. Les portes dans le circuit à simuler doivent être organisées selon l'ordre de propagation des signaux afin qu'elles puissent être évaluées en séquence. En cas des sorties bouclées, un pré-traitement est utilisé pendant la compilation pour éliminer ces boucles, tout en rajoutant des noeuds supplémentaires dans le circuit.

Les circuits à simuler sont supposés synchrones. Si le circuit est asynchrone, une méthode spéciale est appliquée pour découper le circuit en blocs synchrones.

Cette technique est très intéressante lorsque les circuits sont synchrones et très actifs, c'est-à-dire que le pourcentage de portes qui changent l'état de sortie est élevé. J. ZASIO a présenté un simulateur par code compilé [ZAS 87] très performant puisqu'il génère un code machine compact et efficace (2 instructions par porte logique).

◊ Simulation par interprétation

La simulation par interprétation (*interpretative simulation*) est basée sur la technique d'interprétation qui consiste à exécuter directement instruction par instruction une description interprétative. Le circuit à simuler est décrit soit par un programme symbolique soit par des tables qui contiennent des informations (éventuellement ordonnées selon la séquence d'évaluation des portes) concernant la fonction de chaque porte et ses

connexions. Le simulateur joue un rôle d'interpréteur spécifique qui réalise la simulation en interprétant la description du circuit. Cette méthode est relativement peu utilisée dans la simulation logique.

◊ Simulation à échéancier

La simulation à échéancier (*event-driven simulation, table-driven simulation*) est une technique très employée. Le circuit à simuler est représenté par une structure de données qui décrit la topologie et l'état du circuit. Le déroulement de la simulation est dirigé par les événements discrets.

Le simulateur s'appuie sur le fait qu'à chaque instant, il y a une minorité d'éléments actifs dans le circuit. Il ne traite que ces éléments actifs.

Un changement d'état d'une équipotentielle est considéré comme un événement qui se produit instantanément ou avec un retard quelconque. Tous les événement à traiter contiennent au moins trois informations : un temps, un état et une équipotentielle. Ils sont mémorisés dans un échéancier par ordre chronologique.

Pendant la simulation, le simulateur cherche dans l'échéancier les premiers événements ayant la même heure, évalue tous les éléments associés à ces événements. Si la sortie d'un élément change d'état, un nouvel événement est créé et inséré dans l'échéancier.

Le simulateur récupère et traite de nouveau les premiers événements de l'échéancier. Et l'horloge de simulation se met à l'heure de ces événements si l'heure des événements est supérieure à l'heure courante de simulation. La simulation se termine lorsque l'échéancier ne contient plus d'événement ou l'heure de fin de simulation est atteinte.

De nombreux simulateurs logiques sont basés sur cette technique de simulation, tels que les simulateurs ULTIMATE [GLA 84] et Megalogician [CAR 87a]. Puisque cette technique permet de modéliser les circuits synchrones et asynchrones, et d'utiliser tous les types de retards.

2.3.2. Simulateur à échéancier

◊ Structures de base

Le noyau du simulateur à échéancier comporte deux modules de base : le module de gestion de l'échéancier et le module d'évaluation des éléments logiques. La gestion de l'échéancier consiste à insérer, récupérer, supprimer et mettre à jour les événements. L'évaluation des éléments concernés par l'événement actuel se déroule au sein du module d'évaluation.

La topologie du circuit est représentée dans le simulateur par des structures de tables.

◊ Gestion de l'échéancier

Un événement est une activité de base de la simulation. Il est créé par l'évaluation d'un élément ou par les stimuli externes. Un événement d'équipotentielle (t,e',i) représente un changement d'état e' sur une équipotentielle i à un instant donné t. Un événement de commande (t,c) est une commande de contrôle c à l'instant t, telle que la commande de fin de simulation.

Un échéancier est une liste dans laquelle sont mémorisés des événements ordonnés par ordre chronologique. Le module de gestion de l'échéancier ordonne dynamiquement, le long de la simulation, tous les événements dans l'échéancier et effectue des opérations sur les événements. La gestion de l'échéancier est souvent basée sur la méthode dite roue d'horloge (*time wheel*) : la liste d'événements est considérée comme une liste circulaire. La rotation se produit lorsque l'horloge de la simulation progresse, afin de placer les nouveaux événements au sommet de la liste.

Une insertion d'événement est une opération qui introduit dans l'échéancier un nouvel événement. Une récupération d'événement est une opération qui sort de l'échéancier un événement pour l'évaluer. Une suppression d'événement est une opération qui annule un événement explicitement ou implicitement. Une suppression explicite d'un événement s'effectue lors de l'arrivée d'un anti-événement (*anti-event*) dans l'échéancier pour que la simulation retourne en arrière si nécessaire ou que la simulation d'une partie du circuit puisse se synchroniser avec le reste dans un environnement distribué de simulation. Une suppression implicite s'effectue dans le cas où un élément génère dans l'ordre deux événements et que le deuxième anticipe sur le premier et donc annule celui-ci, ou dans le cas de l'annulation d'une impulsion trop courte lorsqu'elle traverse un

élément inertiel.

◊ Evaluation

L'évaluation d'un élément logique est une opération qui consiste à calculer, en fonction de l'état courant des entrées, le nouvel état des sorties. Pour les éléments fonctionnels l'évaluation tient compte de l'état de leurs variables internes.

Un élément actif est un élément dont au moins une entrée ou variable interne change d'état. A chaque pas de simulation, le simulateur évalue tous les éléments actifs, et crée de nouveaux événements.

L'évaluation des éléments et la création des nouveaux événements sont déterminées par la méthode de trace sélective (*selective trace*) : On n'évalue que les éléments actifs, et si le nouvel état sur une sortie ou une variable interne d'un élément actif est différent de l'ancien état, un événement est créé, avec le retard associé à la sortie ou à la variable.

Il existe nombreuses techniques pour évaluer les éléments primitifs, telles que la consultation de table de vérité, l'émulation, l'exécution directe du code objet, etc. Le choix de ces techniques dépend de la complexité des fonctions d'éléments. La consultation de tables semble très efficace pour évaluer les portes logiques et pour déterminer les retards de sorties.

◊ Algorithmes

Dans le domaine de la simulation à échéancier, il existe deux stratégies pour implémenter l'algorithme de l'échéancier : les simulateurs de simple phase et de double phase.

Les simulateurs de simple phase prennent le premier événement de l'échéancier et évaluent immédiatement les éléments concernés par cet événement :

```
TYPE
.       {événement (t,e',i) = [temps, nouvel état, équipotentielle]};
.       {élément (E,S) = [entrées, sorties]}
FIN TYPE;
TANT QUE {échéancier ≠ Ø} FAIRE
```

```
.       {récupérer le premier événement (t,e',i) de l'échéancier};
.       {heure de simulation := t};
.       {état de i := e'};
.       POUR {chaque élément (E,S) qui a une entrée connectée à i} FAIRE
.       .       RENOMMER
.       .       .       {f : fonction de l'élément}
.       .       FIN RENOMMER;
.       .       {S := f (E)};
.       .       POUR {chaque sortie j ∈ S} FAIRE
.       .       .       RENOMMER
.       .       .       .       {r_j : retard associé à j};
.       .       .       .       {e'_j : état de j}
.       .       .       FIN RENOMMER;
.       .       .       SI {e'_j ≠ ancien état}
.       .       .       ALORS
.       .       .       .       {créer un événement (t+r_j, e'_j, j)};
.       .       .       .       {insérer (t+r_j, e'_j, j) dans l'échéancier}
.       .       .       FIN SI
.       .       FIN POUR
.       FIN POUR
FIN TANT QUE;
```

Les simulateurs de double phase cherchent à chaque pas de simulation tous les premiers événements de la même heure et modifient l'état des équipotentielles correspondantes, avant d'évaluer les éléments associés.

Dans cette stratégie, une liste d'éléments actifs L_{ea} est utilisée pour contenir les éléments à évaluer. L'algorithme est le suivant :

```
TYPE
.       {événement (t,e',i)= [temps, nouvel état, équipotentielle]};
.       {élément (E,S)= [entrées, sorties]};
.       {liste d'éléments actives L_ea = [(élément actif)^n]}
FIN TYPE;
TANT QUE {échéancier ≠ Ø} FAIRE
.       {récupérer tous les premiers événements (t,e',i) de l'échéancier qui ont
.               la même heure minimale t};
.       {heure de simulation := t};
.       {L_ea := Ø};
.       POUR {chaque événement (t,e',i) récupéré} FAIRE
```

```
. . {état de i := e'};
. . {mettre tous les éléments (E,S), qui ont une entrée connectée à i,
        dans L_ea}
. FIN POUR;
. POUR {chaque élément (E,S) ∈ L_ea} FAIRE
. . RENOMMER
. . . {f : fonction de l'élément}
. . FIN RENOMMER;
. . {S := f (E)};
. . POUR {chaque sortie j ∈ S} FAIRE
. . . RENOMMER
. . . . {r_j : retard associé à j};
. . . . {e'_j : nouvel état de j}
. . . FIN RENOMMER;
. . . SI {e'_j ≠ ancien état}
. . . ALORS
. . . . {créer un événement (t+r_j, e'_j, j)};
. . . . {insérer (t+r_j, e'_j, j) dans l'échéancier}
. . . FIN SI
. . FIN POUR
. FIN POUR
FIN TANT QUE;
```

La stratégie de double phase essaie d'éviter de recalculer le même élément qui a plusieurs entrées qui changent d'état au même instant. Donc ces simulateurs sont assez efficaces.

Or, statistiquement, le cas de changements d'état multiples sur les entrées d'une porte logique est relativement rare. Un certain nombre de simulateurs logiques [ABR 83] préfèrent l'algorithme de simple phase pour minimiser le nombre de phases. Le simulateur MARS [AGR 87a] a adopté les deux algorithmes et les résultats ont montré que l'algorithme de simple phase était 50% plus rapide que l'algorithme de double phase.

2.3.3. Extension au niveau interrupteur

◊ Définitions

La théorie de l'interrupteur a été introduite par Shannon vers l'année 1940 pour modéliser les calculateurs électromécaniques. La notion

d'interrupteur est aujourd'hui utilisée pour représenter les transistors MOS : la grille est le port de contrôle, la source et le drain étant les ports bidirectionnels.

Les transistors de charge et les buffers à trois états peuvent être considérés comme des éléments unidirectionnels. Les autres types de transistors sont toujours bidirectionnels, sans distinction entre la source et le drain.

Un état (nl,f) d'un signal est caractérisé par son niveau logique nl et sa force f, nl et f étant des valeurs discrètes. Les valeurs de niveaux logiques sont en général 0, 1 et X. Les valeurs de forces sont ordonnées selon leur puissance.

Un noeud est une connexion capacitive qui peut mémoriser un état dynamiquement. Une force de noeud f_n caractérise la capacité relative du noeud n. Un noeud avec la force f_n sans être commandé par des transistors doit avoir l'état (nl_0,f_n), nl_0 étant le niveau logique du noeud la dernière fois qu'il a été commandé.

Un transistor passant est un interrupteur fermé (transistor type N commandé par 1, transistor type P commandé par 0, transistor type D). Un transistor bloqué est un interrupteur ouvert (transistor type N commandé par 0, transistor type P commandé par 1). Un transistor indéterminé est un interrupteur dont l'état du contrôle est inconnu (transistor type N ou P commandé par X). Une force de transistor f_t modélise la conductance relative du transistor t, représentant l'effet d'atténuation lorsqu'un signal traverse le transistor.

L'opérateur T (traversée) détermine le nouvel état (nl,f) d'un signal (nl_0,f_0) qui traverse un transistor passant ayant la force f_t :

$$(nl,f) := T((nl_0,f_0),f_t)$$

En général, nl reste inchangé $nl := nl_0$. Selon les algorithmes, f peut être calculé d'une des manières suivantes :

- $f := f_0$: f_t n'est jamais prise en compte.
- $f := f_t$: f_t est absolue.
- $f := f_0 - f_t$: Le transistor a un effet d'atténuation.

- $f := \min(f_0, f_t)$: Le transistor représente un seuil de force.
- autres.

L'opérateur C (combinaison) détermine le nouvel état (nl,f) d'un noeud lorsque plusieurs signaux (nl_1,f_1), (nl_2,f_2), ..., (nl_m,f_m) sont combinés sur le noeud :

$$(nl,f) := C((nl_1,f_1), (nl_2,f_2), ..., (nl_m,f_m))$$

C est défini comme suit :

- S'il existe un état (nl_i,f_i) où f_i est la plus forte, alors $(nl,f) := (nl_i,f_i)$;
- S'il existe des états (nl_{j1},f_j), (nl_{j2},f_j), ..., (nl_{jk},f_j) où f_j est la plus forte, dans ce cas, si $nl_{j1}=nl_{j2}= ... =nl_{jk}$ alors $(nl,f) := (nl_{j1},f_j)$, sinon $(nl,f) := (X,f_j)$.

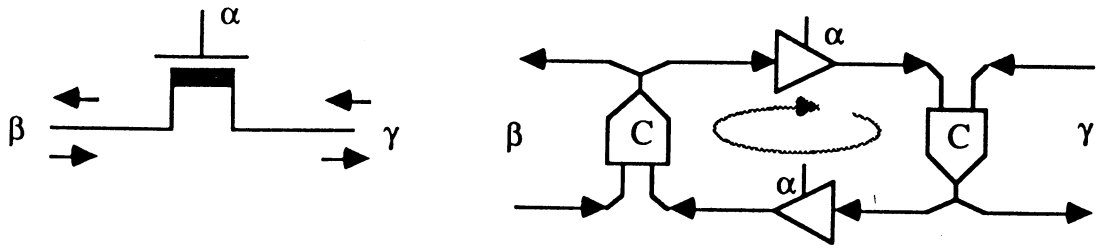
◊ Principe de fonctionnement

Il existe plusieurs méthodes pour réaliser les simulateurs au niveau interrupteur. La méthode de graphe (*graph methode*) est très utilisée. Un graphe est un ensemble de noeuds reliés par les ports bidirectionnels des transistors passants. Un supergraphe est un ensemble de graphes reliés par les ports bidirectionnels des transistors indéterminés. A chaque pas de simulation, le simulateur détermine tous les graphes et les supergraphes. Pour chaque graphe, l'état des noeuds est calculé par l'opérateur C. Ensuite le calcul sur les supergraphes procède avec la règle suivante : s'il y a un conflit possible entre les états de différents graphes reliés par les supergraphes, alors X est affecté à ces graphes.

Cette méthode, ne correspondant pas au principe des simulateurs à échéancier, est difficile à implémenter dans les simulateurs dont l'algorithme est dirigé par les événements.

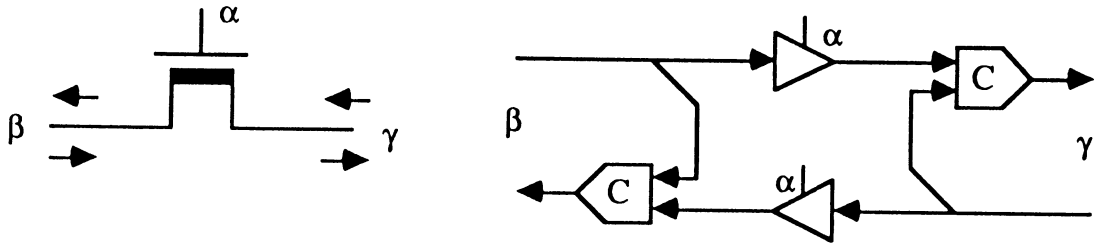
Pour pouvoir utiliser le mécanisme de l'échéancier, on cherche d'abord à remplacer les transistors, qui sont par nature bidirectionnels, par des éléments unidirectionnels.

Une première tentative est de simuler le transistor par l'utilisation des éléments unidirectionnels : Deux portes à trois états et deux portes d'opérateur C constituent un transistor bidirectionnel :

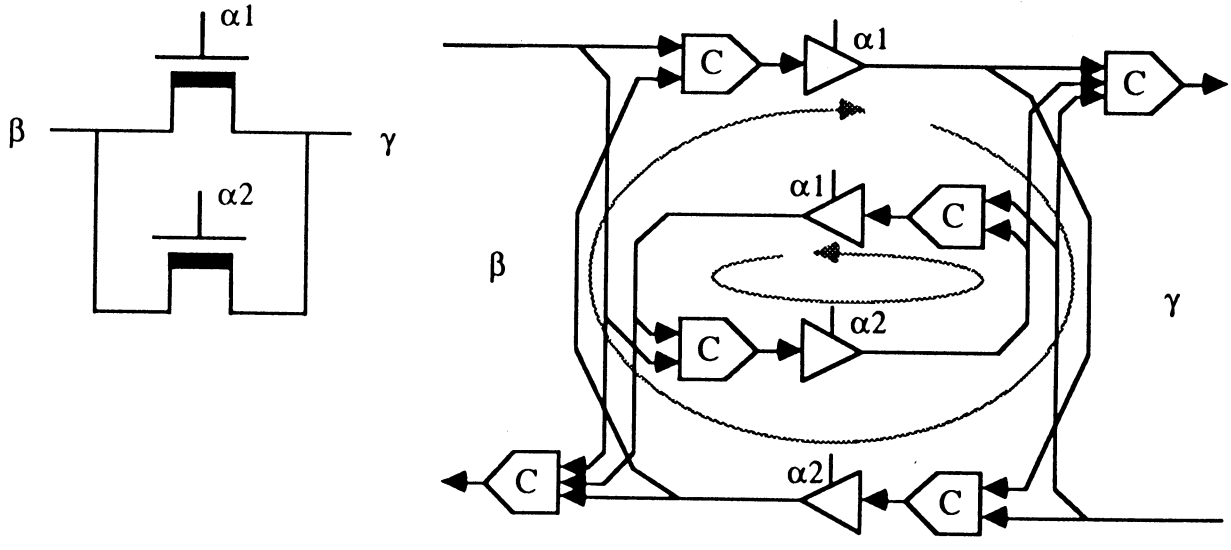


Le problème fatal de cette configuration est qu'il existe un cycle parasite dans lequel le signal peut être saisi et retenu.

Une autre solution consiste à découper le cycle parasite en éliminant les chemins rétroactifs :



Pourtant les transistors eux-mêmes peuvent former des cycles. Ces cycles intrinsèques sont inéluctables :



Afin de surmonter le problème de cycles parasites, la méthode itérative (*iterative method*) est introduite dans le simulateur. La technique est simple : lorsqu'un événement provoque une évaluation, l'état d'un noeud est propagé aux noeuds voisins à travers les transistors, le nouvel état de chaque noeud voisin est déterminé par les opérateurs T et C. Les nouveaux états sont propagés de la même façon. Le calcul se répète jusqu'à ce que tous

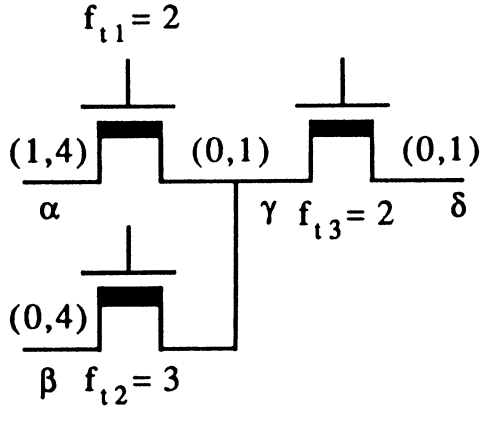
les états se stabilisent. La propagation d'états est considérée comme un pseudo-événement, puisque la propagation est instantanée, qui ne nécessite pas d'être géré dans l'échéancier.

En effet, cette méthode est basée sur les considérations suivantes :

• Il faut déterminer un ensemble de noeuds et de transistors concernés par un événement, lorsque celui-ci est à traiter. Cela représente une étendue dans laquelle le calcul et la propagation doivent se limiter afin d'évaluer un nombre minimal d'éléments du circuit. Cet ensemble de noeuds et de transistors est déterminé par des méthodes dynamiques ou statiques. La méthode dynamique consiste à chercher pendant la simulation, à partir du noeud concerné par l'événement, tous les noeuds reliés par les ports bidirectionnels des transistors passants ou indéterminés. La méthode statique consiste à regrouper tous les noeuds reliés par les ports bidirectionnels des transistors lors du pré-traitement.

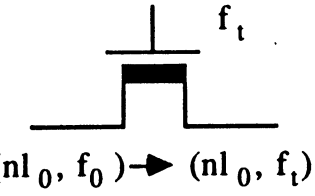
• Avant de calculer l'évaluation, il est nécessaire d'initialiser tous les noeuds pour éviter que des états impropres (notamment l'état X) soient retenus éternellement sur les noeuds. Si l'ancien état du noeud n, d'une force f_n, est (nl_0,f_0), alors le nouvel état (nl,f) de n est (nl_0,f_n). S'il existe des éléments unidirectionnels qui commandent le noeud n avec des états (nl_1,f_1), (nl_2,f_2), ..., (nl_k,f_k), alors (nl,f) := C $((nl_0,f_n), (nl_1,f_1), (nl_2,f_2), ..., (nl_k,f_k))$.

• Pendant la propagation des états, des erreurs peuvent se produire si la propagation s'effectue en désordre :



L'opérateur T est défini comme suit :

$(nl, f) := (nl_0, f_t)$



Dans l'ordre $\alpha(1,4)\rightarrow\gamma(1,2)\rightarrow\delta(1,2)$, $\beta(0,4)\rightarrow\gamma(0,3)\rightarrow\delta(X,2)$, on obtient sur δ l'état (X,2). Alors que dans l'ordre $\beta(0,4)\rightarrow\gamma(0,3)$, $\alpha(1,4)\rightarrow\gamma(0,3)\rightarrow\delta(0,2)$, on obtient sur δ l'état (0,2), qui est l'état correct. Donc l'ordre de la propagation doit respecter la règle "le plus fort d'abord". Une liste de propagation d'états L_{pe} est utilisée dans laquelle sont ordonnés des états à propager par ordre

décroissant de force.

• Le problème le plus délicat est la propagation des états à travers les transistors indéterminés. Le calcul de cette propagation ne doit être ni trop pessimiste ni trop optimiste. Le calcul idéal est de répéter pour chaque transistor indéterminé deux évaluations en supposant que le transistor est respectivement passant ou bloqué. Si les états d'un noeud dans les différentes hypothèses entrent en conflit, l'état du noeud doit être à X. Or cette technique est inapplicable, car pour n transistors, il faut effectuer 2^n calculs. Une meilleure solution est la technique de maximisation. D'abord on ne propage que les états 1 ou X à travers les transistors indéterminés. Ensuite on ne propage que les états 0 ou X à travers les transistors indéterminés. Si les états d'un noeud ne restent pas identiques, alors l'état du noeud est égal à X.

◊ Algorithme

La procédure de l'évaluation des noeuds et des transistors est la suivante :

```
TYPE
.       {état (nl,f) = [niveau logique, force]};
.       {liste de noeuds Ln = [(noeud)^n]};
.       {liste de transistors Lt = [(transistor)^n]};
.       {liste de propagation d'états Lpe = [(noeud)^n], dans laquelle les noeuds
.               sont ordonnés selon l'ordre décroissant de force}
FIN TYPE;
PROCEDURE {Evaluer_noeuds_transistors    (Ln : liste de noeuds,
                                          Lt : liste de transistors)};
.       PROCEDURE {Initialiser (n : noeud, Lpe : liste de propagation d'états)};
.       .       RENOMMER
.       .       .       {(nl0,f0) : ancien état de n};
.       .       .       {fn : force de n};
.       .       .       {(nl1,f1), (nl2,f2),...,(nlk,fk) :  état des sorties des éléments
.       .       .               unidirectionnels connectés à n}
.       .       FIN RENOMMER;
.       .       {état de n := C ((nl0,fn), (nl1,f1), (nl2,f2),...,(nlk,fk))};
.       .       {insérer n dans Lpe selon la force du nouvel état de n}
.       FIN PROCEDURE;
.       PROCEDURE {Maximiser (niveau : 0 ou 1, n : noeud)};
.       .       RENOMMER
```

```
. . . {(nl,f) : état de n}
. . FIN RENOMMER;
. . POUR {chaque transistor t dont un port bidirectionnel est
. .         connecté à n};
. . . SI {(transistor est passant) ou
. . .    ((transistor est indéterminé) et (nl = niveau ou X))}
. . . ALORS
. . . . RENOMMER
. . . . . {f_t : force de t};
. . . . . {n' : noeud connecté à l'autre port
. . . . .         bidirectionnel de t};
. . . . . {(nl',f') : état de n'}
. . . . FIN RENOMMER;
. . . . INTERNE
. . . . . {(nl",f") : état}
. . . . FIN INTERNE;
. . . . {(nl",f") := C ((nl',f'),T((nl,f),f_t))};
. . . . SI {(nl",f") ≠ (nl',f')}
. . . . ALORS
. . . . . {* COMMENTAIRE : état propagé *};
. . . . . {(nl',f') := (nl",f")};
. . . . . {mettre à jour n' dans L_pe selon f'}
. . . . FIN SI
. . . FIN SI
. . FIN POUR
. FIN PROCEDURE;
. {* COMMENTAIRE : début de l'évaluation *};
. {mettre à jour l'état des transistors};
. {L_pe := Ø};
. POUR {chaque noeud n ∈ L_n} FAIRE
. . {Initialiser (n,L_pe)}
. FIN POUR;
. POUR {chaque noeud n ∈ L_pe, selon l'ordre décroissant} FAIRE
. . {Maximiser (1,n)}
. FIN POUR;
. POUR {chaque noeud n ∈ L_pe, selon l'ordre décroissant} FAIRE
. . {Maximiser (0,n)}
. FIN POUR;
. {* COMMENTAIRE : début de création des événements *};
. POUR {chaque noeud n ∈ L_n} FAIRE
```

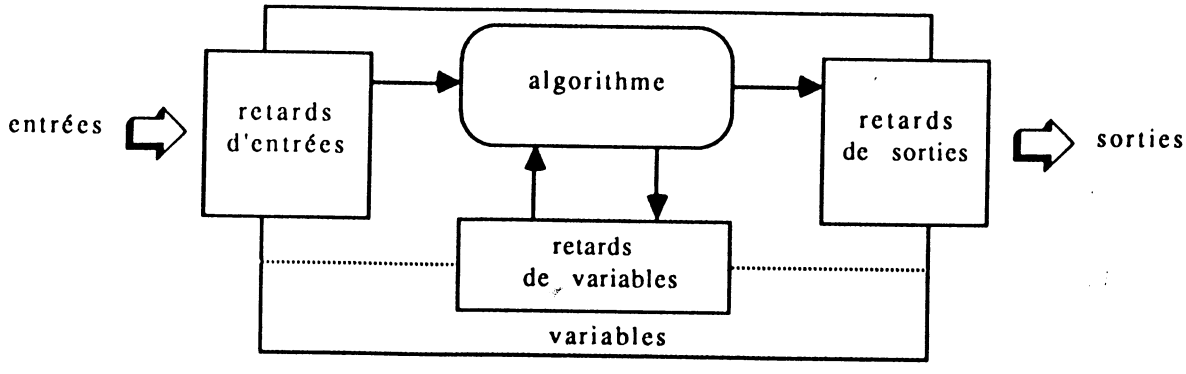
```
.       .       SI {état de n est changé};
.       .       ALORS
.       .       .       {créer un événement et l'insérer dans l'échéancier}
.       .       FIN SI
.       FIN POUR
FIN PROCEDURE;
```

Il faut noter que cet algorithme ne précise pas la modélisation des retards. La propagation des états entre les noeuds est supposée instantanée. Donc le retard est associé soit au noeud lui-même, soit au port de contrôle des transistors. L'utilisation de différents types de retards complexes est également possible, car la transition d'états est connue lors de la création des événements.

2.3.4. Extension au niveau fonctionnel

◊ Modèles

Le principe du fonctionnement du simulateur logique s'applique facilement à la simulation fonctionnelle, car la modélisation des signaux et du temps reste compatible. Pour réaliser un simulateur fonctionnel, il semble suffisant de compléter l'algorithme de l'évaluation des éléments fonctionnels dans le simulateur logique, qui est toujours dirigé par les événements. Toutefois le problème ne reste pas si simple : Il s'agit d'éléments fonctionnels sophistiqués définissant des algorithmes plus au moins complexes, qui peuvent s'opérer sur des variables internes, avec des retards associés aux sorties, aux variables internes, et même aux entrées.



Il existe deux types d'éléments fonctionnels :

• éléments comportementaux : Chaque élément a son interface et son implémentation. Son comportement est décrit par un algorithme procédural ou non procédural. Les éléments peuvent avoir des variables internes et des mécanismes de contrôle, se conduisant comme des automates d'états finis.

• éléments combinatoires : Un élément combinatoire décrit une fonction booléenne. Dans le simulateur certains opérateurs arithmétiques et logiques sont définis permettant de décrire des expressions booléennes simples ou complexes. Les variables internes peuvent être déclarées dans l'élément. Par contre les opérations sur ces variables sont limitées à la lecture et à l'écriture.

L'évaluation d'un élément combinatoire est basée sur le calcul des équations booléennes. Ce calcul s'applique à des expressions générées et optimisées à la compilation. L'algorithme d'un élément comportemental est souvent procédural avec des structures de contrôle. L'évaluation d'un tel élément n'est pas systématiquement instantanée. La synchronisation entre les éléments fonctionnels est possible.

◊ Structures de données

Un événement de sortie (*output event*) est un changement d'état e' sur une sortie i d'élément à l'instant t : (t,e',i). Une porte logique génère toujours des événements de sortie.

L'évaluation différée est utilisée lorsqu'il y a des retards associés aux entrées des éléments. Quand un événement apparaît sur une entrée d'un élément, au lieu d'évaluer immédiatement l'élément, un événement d'entrée (*input event*) est créé avec un retard associé à cette entrée. Le but de l'évaluation différée est d'avoir une modélisation plus réaliste, d'éviter des évaluations redondantes en cas de changements d'état multiples sur les entrées, et de permettre à une évaluation à calculer de tenir compte d'autres événements créés plus tard mais avec un retard plus court et qui affectent donc les résultats de l'évaluation.

Un événement d'état (*state event*) est un changement d'état e' d'une variable interne i d'un élément fonctionnel à l'instant t : (t,e',i). Si à l'instant t le changement d'état e→e' d'une variable interne i d'élément fonctionnel n'est pas instantané, un événement d'état est créé (t+r,e',i), r étant le retard associé à la variable correspondant à la transition e→e'.

En supposant qu'il y ait une porte OU-exclusive dont une entrée est à X, lorsque l'état de l'autre entrée passe de 0 à 1, la sortie reste statiquement à X mais physiquement, une transition d'état se produit. Une simulation sera erronée si la sortie de cette porte est connectée à un élément fonctionnel qui est déclenché par la transition d'état sur son entrée. Pour résoudre ce problème, un événement de transition inconnue (*u-event, unknown transition event*) est utilisé représentant une transition inconnue.

Un événement vectoriel représente un changement simultané d'un groupe de signaux, qui se propagent comme un seul événement. C'est une structure de données inspirée de la simulation au niveau transfert de registre. Cet événement est particulièrement efficace pour les modèles vectoriels (les bus, les registres, etc.).

◊ Algorithme

Un élément (E,S,V) est caractérisé par ses entrées E, ses sorties S et ses variables internes V. Puisque l'élément peut se comporter comme un automate d'états finis, l'algorithme de simulation de double phase est indispensable étant donné que l'algorithme de simple phase peut faire entrer un élément fonctionnel dans un état erroné en cas de changements d'état simultanés sur les entrées ou sur les variables. L'algorithme de l'évaluation des éléments fonctionnels est le suivant :

```
TYPE
.       {événement (t,e',i) = [temps, nouvel état, équipotentielle ou variable]};
.       {élément (E,S,V) = [entrées,sorties,variables]}
FIN TYPE;
PROCEDURE {Evaluer_élément_fonctionnel (e : élément)};
.       INTERNE
.       .       {évaluable : booléen}
.       FIN INTERNE;
.       {évaluable := FAUX};
.       POUR {chaque événement (t,e',i) relié à e, soit i ∈ E ou i ∈ V } FAIRE
.       .       RENOMMER
.       .       .       {ri : retard de i}
.       .       FIN RENOMMER;
.       .       SI {(i ∈ E ) et ((t,e',i) n'est pas un événement d'entrée) et (ri > 0)}
.       .       ALORS
```

```
. . . {créer un événement d'entrée (t+r_i,e',i)}
. . SINON
. . . {état de i := e'};
. . . {évaluable := VRAI}
. . FIN SI
. FIN POUR;
. SI {évaluable}
. ALORS
. . RENOMMER
. . . {f : fonction de l'élément}
. . FIN RENOMMER;
. . {(S,V) := f (E,V)};
. . POUR {chaque j, j ∈ S ou j ∈ V} FAIRE
. . . RENOMMER
. . . . {r_j : retard associé à j};
. . . . {e'_j : état de j}
. . . FIN RENOMMER;
. . . SI {e'_j ≠ ancien état}
. . . ALORS
. . . . {créer un événement (t+r_j, e'_j, j)};
. . . . {insérer (t+r_j, e'_j, j) dans l'échéancier}
. . . FIN SI
. . FIN POUR
. FIN SI
FIN PROCEDURE;
```

Le problème clef de cet algorithme est la manière de calculer les nouveaux états des éléments (S,V) := f (E,V). Cette propriété sémantique détermine les caractéristiques et la capacité du traitement fonctionnel du simulateur.

2.3.5. Caractéristiques des simulateurs à échéancier

Les simulateurs fondés sur l'algorithme de l'échéancier présentent des avantages considérables :

- Pour les simulateurs à échéancier, il n'existe pas de limitation algorithmique pour la modélisation des états logiques. INTSIM [JOH 80] utilise 11 états logiques.
- Ils permettent l'utilisation de différents modèles de retards, pour

que le temps soit discrétisé de façon la plus fine possible. Le simulateur HILO-3 [GEN 85] permet l'utilisation des retards variables, en différenciant le retard de montée et le retard de descente avec trois valeurs (minimale, typique et maximale) associées à chaque retard.

• Ils offrent en plus des primitives logiques, des modèles étendus au niveau plus haut (éléments fonctionnels) et plus bas (interrupteurs) pour faciliter la conception hiérarchique. Les simulateurs logiques peuvent fournir également des modèles non classiques comme les horloges multi-phases, et certaines fonctions câblées (ET-câblé, OU-câblé).

• Le simulateur inclut facilement les algorithmes de la simulation de pannes. La structure topologique du circuit est entièrement conservée au sein du simulateur. Le modèle d'un circuit est utilisable à la fois pour la simulation normale et pour la simulation de pannes.

2.4. Simulateurs de pannes

2.4.1. Diverses méthodes

◊ Méthode parallèle

Dans cette méthode, on suppose qu'un circuit défectueux comporte une panne, qui transforme le circuit normal en un circuit différent. Le circuit sans panne et des circuits défectueux sont simulés simultanément :

• Si M est le nombre total de pannes possibles, alors le circuit peut se transformer en M circuits défectueux, chacun distinct des autres.

• Il faut simuler ces circuits défectueux et le circuit normal. En total M+1 circuits doivent être simulés.

• Soit L étant le nombre de bits d'un mot de machine, si un état logique est codé en K bits, alors N = L/K est le nombre d'états logiques qu'un mot de mémoire peut coder. Donc une instruction machine est capable de calculer en parallèle N états.

• La simulation des M+1 circuits utilise les mêmes opérations logiques avec des états différents causés par des pannes. On peut en conclure qu'il est possible de regrouper N simulations en une seule.

• Partant de ce principe, le simulateur simule N circuits à la fois, en codant N états dans un seul mot. Il suffit d'effectuer (M+1)/N simulations au lieu de M+1.

Cette méthode accélère considérablement la vitesse de la simulation. De plus, peu de mémoire est nécessaire pour stocker les informations car la structure de données est simple. En principe cette méthode convient aux simulateurs par code compilé [SES 62]. Le simulateur HSS [BAR 87] utilise une technique inspirée de cette méthode.

La simulation devient moins efficace lorsque le nombre d'états logiques s'accroît, du fait que plus de bits seront employés pour coder les états logiques. D'autre part, il est difficile d'adapter cette méthode à la modélisation fonctionnelle.

◊ Méthode déductive

Les circuits défectueux ne sont pas simulés explicitement dans cette méthode. Le simulateur déduit, à l'aide de la propagation des listes de

pannes, toutes les pannes détectables sur tous les noeuds internes et les sorties du circuit. Chaque liste associée à un signal contient un ensemble de pannes qui, quand elles sont présentes individuellement dans le circuit, conduisent l'état du signal à devenir différent de son état normal :

• Un circuit est considéré comme un graphe, les éléments logiques étant les noeuds du graphe, les interconnexions étant les branches.

• Les noeuds ayant au moins une entrée qui est en même temps une entrée primitive du circuit, sont appelés les noeuds du premier niveau. Les noeuds qui ont au moins une entrée connectée à la sortie d'un noeud du premier niveau sont appelés les noeuds du deuxième niveau, et ainsi de suite.

• Le simulateur calcule les listes de pannes, chaque liste étant associée au signal de sortie d'un noeud du premier niveau. Une liste de pannes contient toutes les pannes observables sur le noeud. Une panne observable sur un noeud signifie que l'état de la sortie du noeud est perturbé par la présence de cette panne, et donc différent de sa valeur normale.

• Ensuite l'état correct du noeud du premier niveau et la liste de pannes sont propagés à un noeud du deuxième niveau. Le simulateur rajoute de nouvelles pannes observables sur ce noeud dans la liste de pannes. Par conséquent, cette liste contient deux sortes de pannes qui sont toutes observables à la sortie du deuxième noeud : les pannes observables du premier noeuds et les pannes du deuxième noeud lui-même.

• Le calcul se déroule niveau par niveau en traversant tout le circuit. On obtient alors toutes les pannes observables aux sorties du circuit.

Cette méthode nécessite une grande mémoire, qui conduit souvent à l'utilisation de la mémoire dynamique [ARM 72]. La complexité de l'algorithme de propagation des liste de pannes augmente lorsque plus d'états logiques sont introduits dans la simulation.

◊ Méthode concurrente

Cette méthode considère qu'un circuit défectueux se compose de deux parties : une partie contenant des éléments perturbés par les pannes, une autre partie contenant des éléments normaux, et que ce circuit défectueux peut avoir un comportement légèrement différent du circuit normal. On simule le circuit normal et les circuits défectueux à la fois. Dans un circuit défectueux seuls les éléments ayant un comportement explicitement différent sont simulés. Chaque élément est affecté d'une liste contenant des

copies de cet élément, qui représentent les effets des pannes :

• Un module de référence M représente cet élément qui fonctionne correctement. M dénote soit l'élément lui-même soit son module de référence.

• En supposant qu'il existe n pannes p_1 .. p_n dans le circuit, un module concurrent M_{pi} représente cet élément qui fonctionne en présence de la panne p_i.

• A un certain moment de la simulation, les états d'un module M_p peuvent être identiques ou différents de ceux du module M. S'ils sont identiques, le M_p est implicite car il peut être représenté par le module M; sinon il est explicite. Si à un instant le module M_p est implicite, il pourra soit rester implicite soit devenir explicite à l'instant suivant.

• Une liste appelée liste de pannes concurrentes LPC est un dispositif associé à chaque module M. Elle contient tous les modules M_{pi}, M_{pj}, ... M_{pk} explicites.

• Pour chaque module M du circuit, le simulateur ne tient compte que de ce qui est dans la LPC de M. La gestion des LPC est dynamique au cours de la simulation : selon le calcul, certains M_p sont effacés de la LPC, d'autres y sont rajoutés. Donc la LPC ne contient que les informations nécessaires pour minimiser la redondance de la simulation.

Cette méthode permet d'utiliser la technique et la structure de données de la simulation à échéancier. Elle en hérite aussi tous les avantages : la précision basée sur un grand nombre d'états logiques, l'exactitude du comportement temporel avec les retards raffinés et la modélisation des éléments fonctionnels [ULR 85] etc. Tout cela rend cette méthode de plus en plus populaire [ABR 77] [GAI 88].

2.4.2. Simulateur de pannes concurrentes

◊ Définitions

Un module M est caractérisé par les états logiques de ses entrées E, et de ses sorties S. Si le module contient des variables V, tel que les registres et les compteurs, il peut être représenté par sa configuration (E,S,V). Pour un module sans état interne, tel que les portes logiques, sa configuration peut être sous forme de (E,S).

Si p est une panne se trouvant sur le module, celui-ci est un module

défectueux (source de panne) M_p, représenté par sa configuration (E_p,S_p,V_p). Si la panne p se trouve à l'extérieur du module, et que le module a une configuration différente de celle de M, le module est un module erroné (effet de panne) M_p, soit $(E_p,S_p,V_p) \neq (E,S,V)$.

Une panne est dite visible sur une équipotentielle i qui est une des sorties de M, si l'état sur l'équipotentielle i de M et celui sur i de M_p sont différents. Sinon elle est invisible. Une panne est dite détectable sur l'équipotentielle i si elle est visible et que l'état sur i de M et celui sur i de M_p ont chacun une valeur déterminée (0 ou 1).

La liste de pannes concurrentes de M est notée LPC_M. Elle contient des modules défectueux ou erronés (E_p,S_p,V_p). Si un (E_p,S_p,V_p) est dans LPC_M, on dit que $(E_p,S_p,V_p) \in LPC_M$ et que $p \in LPC_M$.

Pendant l'évaluation du module M, un module défectueux M_p reste toujours dans LPC_M, même si sa configuration $(E_p,S_p,V_p) = (E,S,V)$, jusqu'à ce que la panne p soit détectée à une sortie primitive du circuit, tandis qu'un module erroné M_p, qui sert à propager la panne p, ne reste dans LPC_M que si $(E_p,S_p,V_p) \neq (E,S,V)$.

Un changement d'état normal $e \rightarrow e'$ signifie qu'une équipotentielle i du module sans pannes change l'état de e à e'. Un module M_p avec une panne p (la panne peut se trouver à l'extérieur du module) est actif lorsqu'il a dans le même contexte un événement $e_p \rightarrow e'_p$ sur l'équipotentielle i. On dit aussi que la panne est active. Sinon le module et la panne sont passifs, soit $e_p = e'_p$.

Une panne active est absolument visible si $e_p \neq e$ et $e'_p \neq e'$.

Une panne passive peut être visible pendant la simulation. Une panne passive p est de nouveau visible sur une équipotentielle i si e_p reste inchangé et égal à e ($e \neq e'$ alors $e_p \neq e'$ donc cette panne devient visible). Une panne passive p est de nouveau invisible sur une équipotentielle i si e_p reste inchangé et égal à e'.

Pour chaque module, il faut propager les informations suivantes aux modules connectés à une de ses sorties :

- l'état normal e' de la sortie;
- toutes les pannes p actives dont $e'_p \neq e'$: Une liste de pannes actives

L_{pa} est donc employée contenant des couples (p,e'_p) dans lesquels p est une panne active et que e'_p est l'état correspond à p;

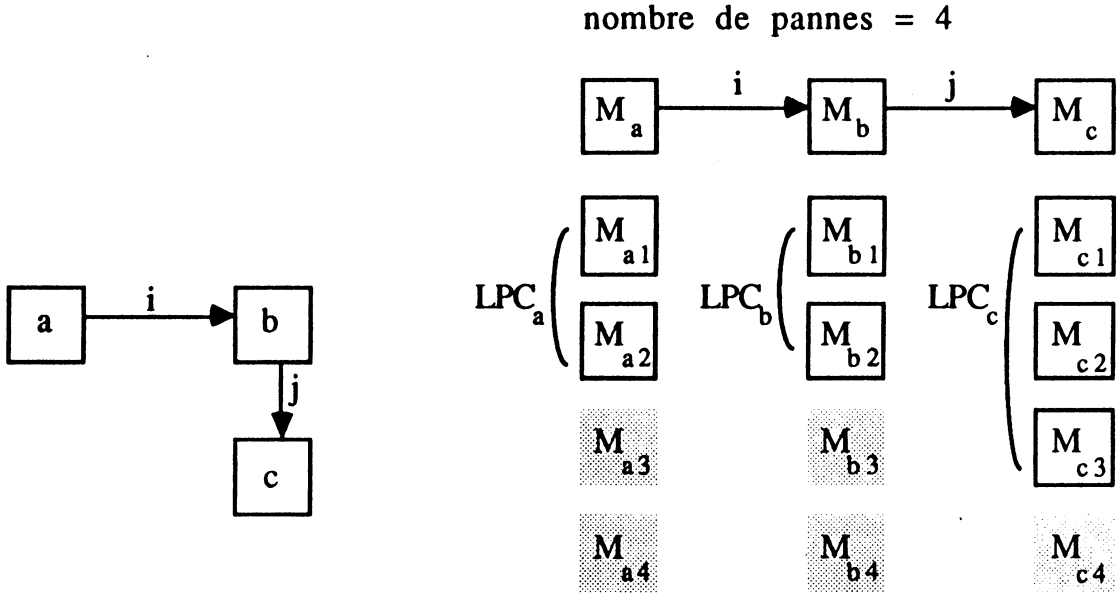
• toutes les pannes de nouveau visibles : Leur état sur i reste inchangé. Une liste de pannes de nouveau visibles L_{pdnv} contient toutes ces pannes.

Un événement composé (t,e',i,L_{pa},L_{pdnv}) est constitué de :

- un temps t;
- un nouvel état de sortie e';
- une équipotentielle i;
- une liste de pannes actives L_{pa};
- une liste de pannes de nouveau visibles L_{pdnv}.

◊ Evaluation

Le simulateur utilise l'algorithme de l'échéancier pour gérer les événements composés.



Circuit Modules

Pour chaque module du circuit, il existe un module de référence M (sans panne) et certains modules concurrents M_p (modules défectueux ou erronés). Le simulateur ne calcule que les modules concurrents qui sont explicitement différents de M, et ne mémorise, pour les modules M_p, que les informations différentes de M.

La liste de pannes concurrentes LPC_M contient des modules concurrents (E_p,S_p,V_p) qui nécessitent un calcul. La liste de pannes concurrentes LPC_M change de contenu dynamiquement durant la simulation. La divergence est le cas de création d'un module M_p dans LPC_M lorsque la configuration (E_p,S_p,V_p) de celui-ci devient différente de (E,S,V), tandis que la convergence est l'annihilation d'un module M_p au moment où sa configuration (E_p,S_p,V_p) devient identique à (E,S,V). La divergence et la convergence sont uniquement valables pour les modules erronés. Un module défectueux reste toujours dans LPC_M avant que la panne soit détectée. Quand une panne p est détectée, tous les M_p associés aux différents modules du circuit sont retirés de la simulation.

Une évaluation de M est une opération pour calculer les nouveaux états de M. Un traitement de M est une opération pour déterminer la divergence et la convergence de M.

Pendant la simulation, si un événement arrivé à l'instant t sur une entrée i de M est un événement composé (t,e',i,L_{pa},L_{pdnv}), le simulateur évalue d'abord M, puis évalue et traite toutes les pannes actives, traite toutes les pannes de nouveau visibles (elles ne nécessitent pas d'évaluation), et enfin génère des événements sur les sorties de M.

TYPE
. {module M = module de référence};
. {module M_p = module concurrent de panne p};
. {configuration (E,S,V) =
. [états des entrées,
. états des sorties,
. états des variables]};
. {liste de pannes actives L_{pa} = [(panne active p,e'_p)^n]};
. {liste de pannes de nouveau visibles L_{pdnv} =
. [(panne de nouveau visible p)^n]};
. {liste de pannes concurrentes LPC = [$(E_p,S_p,V_p)^n$]};
. {événement composé (t,e',i,L_{pa},L_{pdnv}) =
. [temps,
. nouvel état,
. équipotentielle ou variable,
. liste de pannes actives,
. liste de pannes de nouveau visibles]}

```
FIN TYPE;
PROCEDURE {Evaluer_traiter_module (M : module,
                                   (t,e',i,Lpa,Lpdnv) : événement composé)};
.   PROCEDURE {Evaluer_module (M : module)};
.   .   {calculer les nouveaux états de M};
.   .   {mettre à jour (E,S,V) de M}
.   FIN PROCEDURE;
.   PROCEDURE {Traiter_panne_active (p : panne active)}
.   .   EXTERNE
.   .   .   {LPCM : liste de pannes concurrentes de M}
.   .   FIN EXTERNE;
.   .   SI {(Ep,Sp,Vp) = (E,S,V) et Mp est erroné}
.   .   ALORS
.   .   .   {effacer Mp de LPCM};
.   .   .   {* COMMENTAIRE : convergence *}
.   .   SINON
.   .   .   {maintenir Mp dans LPCM}
.   .   FIN SI
.   FIN PROCEDURE;
.   PROCEDURE {Traiter_panne_passive (p : panne passive)};
.   .   EXTERNE
.   .   .   {LPCM : liste de pannes concurrentes de M};
.   .   .   {Lpdnv : liste de pannes de nouveau visibles}
.   .   FIN EXTERNE;
.   .   CAS
.   .   {(Ep,Sp,Vp) ≠ (E,S,V)}
.   .   .   {maintenir Mp dans LPCM };
.   .   .   SI {p ∈ Lpdnv}
.   .   .   ALORS
.   .   .   .   {effacer p de Lpdnv};
.   .   .   .   {* COMMENTAIRE : p est insérée dans LPCM *)
.   .   .   FIN SI
.   .   {(Ep,Sp,Vp) = (E,S,V)} et Mp est erroné}
.   .   .   {effacer Mp de LPCM };
.   .   .   {* COMMENTAIRE : convergence *}
.   .   AUTREMENT
.   .   .   {maintenir Mp dans LPCM }
.   .   FIN CAS
.   FIN PROCEDURE;
.   {* COMMENTAIRE : début de calcul d'évaluation et de traitement *};
```

```
{Evaluer_module (M)};
POUR {chaque p ∈ Lpa}
    {* COMMENTAIRE : p est active *};
    SI {p ∉ LPCM}
    ALORS
        {(Ep,Sp,Vp) := ancienne (E,S,V)};
        {ajouter Mp dans LPCM}
    FIN SI;
    {Evaluer_module (Mp)};
    {Traiter_panne_active (Mp)}
FIN POUR;
POUR {chaque p ∈ LPCM et p non encore calculé}
    {* COMMENTAIRE : p est passive *};
    {* COMMENTAIRE : l'évaluation n'est pas nécessaire *};
    {Traiter_panne_passive (Mp)}
FIN POUR;
POUR {chaque p reste dans Lpdnv}
    {(Ep,Sp,Vp) := ancienne (E,S,V)};
    {mettre Mp dans LPCM};
    {* COMMENTAIRE : ni évaluation ni traitement, car elles sont
                      calculées antérieurement *}
FIN POUR;
{* COMMENTAIRE : fin de calcul d'évaluation et de traitement *};
{* COMMENTAIRE : début de création des événements de sorties *};
POUR {chaque sortie ou variable j}
    RENOMMER
        {r : retard de j};
        {e : ancien état sur j de M};
        {e' : nouvel état sur j de M}
    FIN RENOMMER;
    {Lpa := Ø};
    {Lpdnv := Ø};
    POUR {chaque p ∈ LPCM}
        RENOMMER
            {ep : ancien état sur j de Mp};
            {e'p : nouvel état sur j de Mp}
        FIN RENOMMER;
        CAS
        {ep ≠ e'p et e'p ≠ e'}
            {mettre (p,e'p) dans Lpa};
```

```
. . . . {* COMMENTAIRE : p est active *}
. . . {e_p = e'_p et e'_p ≠ e' et e ≠ e'}
. . . . {mettre p dans L_pdnv};
. . . . {* COMMENTAIRE : p est de nouveau visible *}
. . . FIN CAS
. . FIN POUR;
. . {créer un événement composé (t+r, e', j, L_pa, L_pdnv)}
. FIN POUR;
. {* COMMENTAIRE : fin de création des événements de sorties *}
FIN PROCEDURE;
```

Comparée avec les méthodes parallèle et déductive, la méthode concurrente semble la plus rapide au niveau de vitesse d'exécution. De plus en plus de simulateurs de pannes récemment développés utilisent cette méthode.

2.4.3. Considérations sur les simulateurs de pannes

La flexibilité, la précision et l'efficacité sont les caractéristiques principales de la simulation de pannes concurrentes. Une majorité de types de pannes peuvent être simulées par cette technique [DOS 84]. Les avantages de la simulation de pannes concurrentes sont les suivants :

• En général, avec la méthode parallèle un élément doit être évalué autant de fois que le nombre de pannes possibles, même si le comportement de l'élément est identique pour la plupart des cas. Le calcul est très redondant.

• Dans la méthode déductive la propagation de listes de pannes est un algorithme sophistiqué. Lorsque le circuit est complexe, un grand nombre de pannes sont impliquées dans les listes de pannes. La propagation de ces listes est encore plus difficile à réaliser dans le cas où les modèles fonctionnels sont utilisés. D'autre part, lorsque les états logiques dépassent 3 valeurs (0,1,X), la gestion des listes de pannes devient très coûteuse.

• Le simulateur de pannes concurrentes utilise les même modèles que ceux du simulateur normal. Il suppose qu'une panne se produit toujours à l'extérieur des éléments primitifs ou qu'elle peut être représentée par une panne extérieure. En effet toutes les techniques de la simulation normale comme les retards complexes, les états logiques raffinés, les modèles fonctionnels, la structuration de données, et le mécanisme de l'échéancier sont directement utilisables dans le simulateur de pannes concurrentes. De

plus, cette méthode est indépendante de l'algorithme de l'évaluation des éléments du circuit et donc facile à implémenter.

2.5. Langages de description du matériel

2.5.1. Outils de CAO et langages

La conception de circuits intégrés nécessite des langages intermédiaires qui permettent de décrire le circuit à tous les niveaux d'abstraction pour établir des relations fiables entre l'homme et les outils informatiques et pour assurer des innovations dans le domaine de la conception. Afin de répondre à ces besoins forts, les langages de description du matériel (*Hardware Description Languages*) ont été développés à partir du milieu des années soixante. Récemment, de plus en plus de propositions ont vu le jour [GER 85] [PIL 85] [SHA 85] [SHI 85] [SUZ 85] [BAR 88a] [MAR 88] [GHO 88] [NUR 89].

Le langage de description du matériel joue un double rôle : d'une part le langage de conception qui établit la relation "homme-machine" permettant à l'homme d'utiliser et de contrôler les outils informatiques, et d'autre part le langage de description qui établit la relation "homme-homme" permettant les interactions entre différents processus de conception.

Du point de vue de l'utilisateur des outils de CAO, le langage doit lui assurer un exposé complet de tous les éventuels problèmes du matériel. La spécification de la conception doit être décrite par le langage sans ambiguïté ni dénaturation.

Le langage offre également une interface commune entre des outils de CAO. Il permet de créer des sources standard, qui facilitent les échanges de données, dans des domaines spécifiques de conception. A ce propos, un langage ayant une syntaxe et une sémantique standardisées assure la réalisation des environnements intégrés de conception fondés sur différentes méthodologies.

Les langages de description du matériel sont les langages spécifiques qui doivent être capables :

- d'exprimer les aspects algorithmiques, architecturaux, fonctionnels et structurels du matériel;
- de décrire de manière parallèle, temporelle et non récursive les propriétés du matériel;
- de fournir différents niveaux de description du matériel pour

assurer une conception descendante ou ascendante;

• de modéliser le matériel d'un manière modulaire, générique et hiérarchique pour structurer la description du matériel;

• de définir et de gérer, pour une conception de matériel, plusieurs versions ou implémentations alternatives qui permettent de décrire le matériel correspondant au différents niveaux d'abstraction ou différentes réalisations physiques;

• de permettre de réaliser différents types d'outils de CAO et des environnements intégrés de CAO pour faciliter les échanges d'informations entre les processus de la conception du matériel.

Les langages sont exploités par les outils de CAO dans les domaines suivants :

- la simulation;
- le test;
- la synthèse et la génération automatique;
- la vérification formelle;
- la documentation.

2.5.2. Applications à la simulation

Il s'agit des langages spécifiques dédiés à la description du circuit et du contexte de simulation. Les langages de description permettent :

• la modélisation du matériel : En ce qui concerne la simulation, le circuit doit être décrit par un langage qui correspond à un ou plusieurs niveaux d'abstraction. Différents aspects de l'architecture (structurel et comportemental) du circuit, différents styles algorithmiques (procédural et non procédural), différents mécanismes de contrôle (flots de contrôle et de données) doivent être descriptibles complètement ou partiellement dans le langage. Le langage doit avoir en plus une bonne structuration de données. Cela nécessite un langage modulaire, générique et hiérarchique.

• la description du contexte de simulation : L'essentiel est de pouvoir abstraire les conditions physiques d'un environnement opérationnel par l'utilisation d'un langage approprié. Par ailleurs, les interactions entre le circuit à simuler et le monde extérieur, notamment les stimuli et les réponses de la simulation, doivent être spécifiées par le langage. Cela nécessite un langage temporel et concurrent.

• le contrôle de l'exécution de la simulation : Le langage doit fournir

des mécanismes de contrôle pour permettre à l'utilisateur d'accéder à l'outil informatique. Le contrôle de l'outil peut être décrit par le même langage ou par un langage spécifique. La stratégie dépend de l'indépendance du langage, qui peut être dédié aux différents outils de conception. Cela nécessite un langage, éventuellement interactif, basé sur un ensemble de commandes.

• la documentation ou les sources secondaires : Un bon langage assure une spécification exacte du comportement et de la structure du circuit. Une telle spécification doit rendre possible l'interprétation à la fois par l'homme et par la machine. Cela nécessite un langage descriptif.

• les échanges de données entre différents outils : D'une part, la description peut être une source partagée par plusieurs outils tels que le simulateur et le compilateur de silicium. D'autre part le langage définit une interface entre les outils ayant la relation producteur-consommateur telle que l'interface entre le synthétiseur logique et le simulateur. Cela nécessite un langage commun.

2.6. Concepts de base des langages

2.6.1. Types des descriptions

◊ Descriptions procédurale et non procédurale

Face à la complexité de la conception du circuit et à la diversité des outils de CAO de différents domaines, la description du matériel se divise en deux classes : procédurale et non procédurale.

Ces deux classes définissent des manières différentes de la spécification du circuit :

• la description procédurale (*procedural description*) : Dans cette description, les opérations sont organisées dans un ordre spécifique qui détermine la séquence à laquelle les opérations s'exécutent, en tenant compte des structures de contrôle et de branchement.

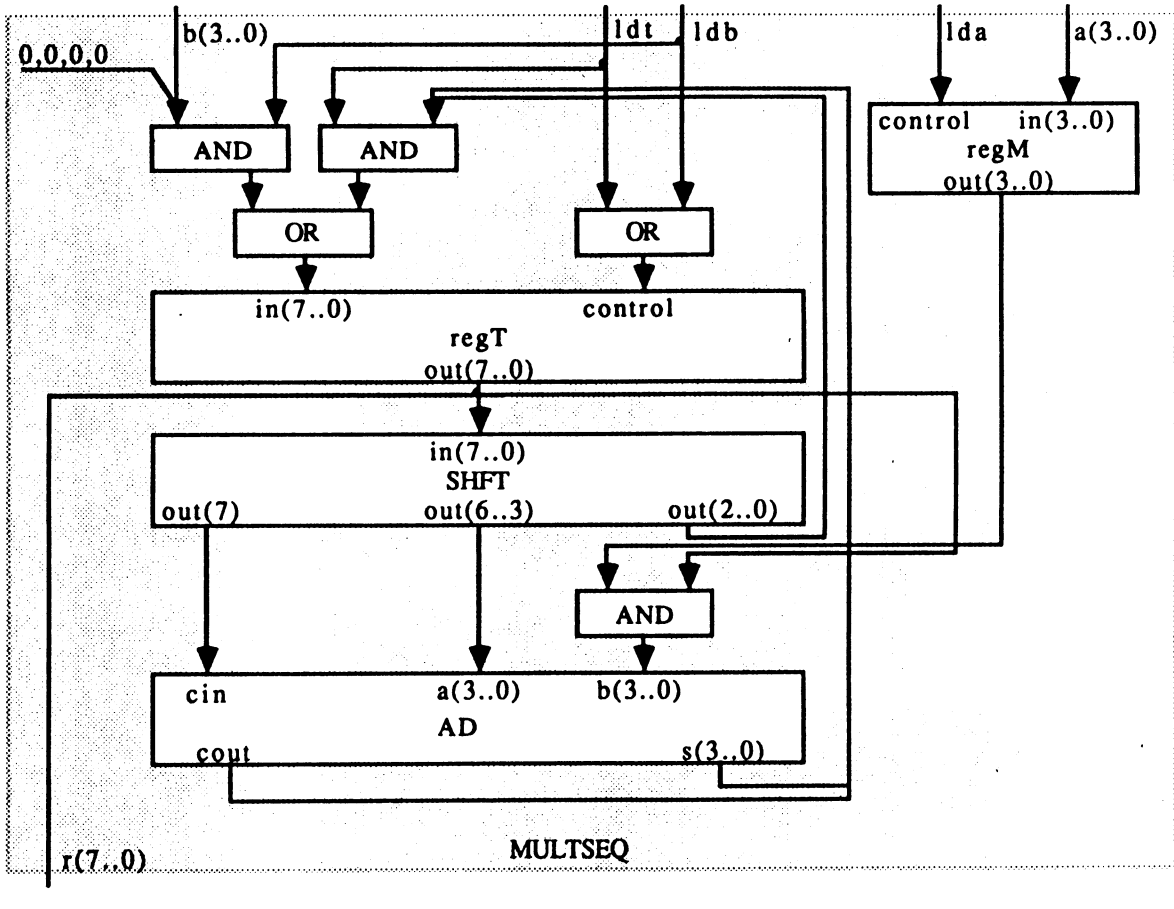
• la description non procédurale (*non-procedural description*) : Dans cette description, les opérations sont organisées en groupes avec ou sans conditions de déclenchement. Il n'existe pas d'enchaînement global qui contrôle toutes les opérations décrites. Les opérations sont alors parallélisables. L'ordre de l'exécution des opérations ne dépend pas de l'ordre dans lequel les opérations sont décrites.

Dans [DUD 83] on donne un exemple de multiplicateur séquentiel, décrit respectivement de manière procédurale et non procédurale.

La description suivante est procédurale. Car l'ordre d'écriture des instructions détermine l'ordre de leur exécution :

```
implementational MULTSEQ (a[3..0],b[3..0]; r[7..0]);
(* a,b and r are the multiplicand, multiplier and result resp. *)
terminal    ti : array[7..0] of boolean;
                    (* ti transfers the intermediate result *)
            d : array[3..0] of boolean;
                    (* d is an addition operand *)
storage     t : array [3..0] of boolean;
                    (* t stores the intermediate result *)
            m : array [3..0] of boolean;
                    (* m stores the multiplicand *)
```

```
begin
        m[3..0] := a[3..0]; (* store multiplicand *)
        t[7..0] := 0_0_0_0_b[3..0]; (* initialize *)
        for i := 0 to 3 do
                begin
                        c[4..0]_ti[2..0] := SHIFTRIGHT(t[7..0]);
                        d[3..0] := t[0] and m[3..0];
                        ti[7..3] := ADDER (c[4], c[3..0], d[3..0]);
                        t[7..0] := ti[7..0]
                end;
        r[7..0] := t[7..0]
end.
```



Multiplicateur séquentiel

La description suivante est non procédurale. Les instructions représentent les interconnexions entre les éléments. L'ordre de l'exécution des instructions ne dépend pas celui de l'écriture :

```
implementational MULTSEQ (a[3..0],b[3..0],lda,ldb,ldt; r[7..0]);
```

```
(* a,b and r are the multiplicand, multiplier and result resp. *)
(* lda,ldb,ldt are control signals *)
(* declaration of internal elements *)
element        regM : Dreg; (* D-type multiplicand register *)
               regT : Dreg; (* D-type intermediate result register *)
               SHFT : SHIFTRIGHT; (* right shift device *)
               AD : ADDER; (* 4-bit adder *)
interconnect
        begin
               regM.in := a[3..0]; regM.control := lda;
               regT.in := ((0_0_0_0_b) and ldb) or
                        ((AD.cout_AD.s_SHFT.out[2..0]) and ldst);
               regT.control := ldt or ldb;
               SHFT.in := regT.out;
               AD.cin := SHFT.out[7]; AD.a[3..0] := SHFT.out[6..3];
               AD.b[3..0] := regM.out[3..0] and regT.out[0];
               r[7..0] := regT.out[7..0]
        end.
```

La description procédurale permet de décrire le circuit d'une manière algorithmique. Différents types de variables, différentes structures de contrôle et d'itérations peuvent être utilisés, permettant au langage d'avoir une grande capacité descriptive. La description non procédurale exprime de façon naturelle le parallélisme inhérent au circuit, et reflète ses propriétés architecturales.

Les descriptions procédurale et non procédurale s'adaptent respectivement aux différentes phases de conception de circuits. Lorsque le niveau d'abstraction est élevé, le problème de la conception se concentre sur ce que le circuit doit accomplir. A ce stade, le circuit peut être décrit de façon procédurale puisque sa structure paraît encore très floue. Au fur et à mesure que la conception se raffine, de plus en plus de détails deviennent visibles. Le circuit peut être décrit de façon non procédurale, car le parallélisme du circuit devient clair.

◊ Descriptions comportementale et structurelle

Un matériel peut être décrit par son comportement ou par sa structure. Ce sont les deux manières qui se complètent pour décrire l'architecture du matériel :

• la description comportementale (*behavioral description*) : C'est une description relativement abstraite. Le circuit est représenté par un algorithme qui décrit son comportement. Si l'algorithme n'est basé que sur des structures de données abstraites, cette description est purement comportementale. Si l'algorithme manipule des données qui ont une correspondance forte ou faible à des objets physiques, cette description est fonctionnelle.

• la description structurelle (*structural description*) : C'est une description plus proche des propriétés physiques du matériel. Le circuit est représenté par des objets et leurs relations. Les objets correspondent aux composants physiques, tandis que les relations indiquent la manière d'interconnecter et de relier ces composants.

La description comportementale représente une entité, dont l'intérieur est invisible, qui réalise une certaine fonction spécifique. La description structurelle définit la topologie du circuit et les états internes du circuit.

Les descriptions suivantes représentent une bascule RS synchrone décrite en langage MADL [INF 83] de manières comportementale et structurelle respectivement :

```
BEH BLOCK RSLATCH (INPORT BIT R, S, CLK
                   IOPORT BIT Q, QBAR);
BEGIN
        Q := ~(~(R & CLK) & QBAR);
        QBAR := ~(~(S & CLK) & Q);
END;
```

```
BLOCK RSLATCH     (INPORT BIT R, S, CLK
                   IOPORT BIT Q, QBAR);
STRUCT BIT TMP1, TMP2;
BEGIN
        TMP1 := NAND (R, CLK);
        TMP2 := NAND (S, CLK);
        Q := NAND (TMP1, QBAR);
        QBAR := NAND (TMP2, Q);
END;
```

Une description comportementale peut être procédurale ou non procédurale. Une description structurelle est en général, par nature, non procédurale. Si une description comportementale est non procédurale, un ensemble d'opérations peuvent être exécutées en parallèle, contrôlées par des conditions de garde, par exemple des transferts de registres contrôlés par des conditions de déclenchement. Donc :

• Une description comportementale, si elle est non procédurale, est basée sur des objets élémentaires ou complexes qui représentent des composants physiques. La structure du matériel est implicitement décrite.

• La description structurelle exprime les détails physiques du matériel. La structure du matériel est explicitement décrite.

La description structurelle et la description comportementale, caractérisent deux aspects de l'architecture d'un matériel.

◊ Description dirigée par les flots de contrôle et de données

Du point de vue de flots d'informations et de mécanismes de contrôle de langages, deux types de descriptions peuvent être envisagées dans les langages :

• la description dirigée par le flot de contrôle (*control flow driven description*) : C'est une structure de contrôle très classique. Le parallélisme et la séquentialité sont explicitement définis par les règles du langage. L'algorithme qui décrit le matériel est piloté par des mécanismes de contrôle formellement spécifiés. Cette description, qui permet d'exprimer la nature algorithmique du matériel est en général comportementale.

• la description dirigée par le flot de données (*data flow driven description*) : Dans la description, les mécanismes de contrôle sont implicites, dépendants de la disponibilité des données. Les opérations ne sont pas contrôlées par un graphe de contrôle global. Dans cette gamme de description on trouve en effet la description dirigée par le flot de signaux ou par le flot d'événements logiques. Cette description, liée aux propriétés topologiques du matériel, est souvent structurelle, toujours caractérisée par la nature parallèle et concurrente.

L'exemple suivant est un additionneur 4 bits décrit en langage VHDL. C'est une description dirigée par le flot de données, car chaque instruction se déclenche lorsque les données sur les entrées sont disponibles :

```
architecture Data_Flow_Adder4 of Four_Bit_Adder is
    signal C : Bit_Vector (3 downto 0);
    component Full_Adder port (Cin, I1, I2 : in Bit; Cout, Res : out Bit);
    end component;
begin
    for i in 3 downto 0 generate
        if i = 0 generate
            First_Stage : Full_Adder port map (Cin, A(i), B(i), C(i), Sum(i));
        end generate;
        if i > 0 generate
            Other_Stage : Full_Adder port map (C(i-1), A(i), B(i), C(i), Sum(i));
        end generate;
    end generate;
    Cout <= C(3);
end Data_Flow_Adder4;
```

La description suivante décrit le même additionneur, toujours en VHDL, mais de manière comportementale. C'est une description dirigée par le flot de contrôle. Le point-virgule spécifie le flot de contrôle séquentiel :

```
architecture Control_Flow_Adder4 of Four_Bit_Adder is
begin
    process
        variable Sum : Bit_Vector (4 downto 0);
    begin
        Sum := "+" (('0, A), ('0, B));
        Sum := "+" (Sum, ('0, '0, '0, '0, Cin));
        Res := Sum (3 downto 0);
        Cout := Sum (4);
    end process;
end Control_Flow_Adder4;
```

2.6.2. Structuration de données

◊ Modularité

La modularité du langage de description du matériel est basée sur la notion de boîte noire, qui correspond à la décomposition physique du circuit. Un module se compose de :

• une interface qui définit les ports de connexion du module matériel avec l'extérieur, et des paramètres formels;

• une implémentation qui décrit la réalisation et le fonctionnement du module matériel. Cette implémentation peut être invisible de l'extérieur. Un module peut comprendre plusieurs implémentations alternatives.

La partie fondamentale de l'interface du module est caractérisée par les ports de connexion. Les ports de connexion sont en général typés, en fonction de :

• la direction du signal qui traverse le port : entrée, sortie ou bidirectionnel;

• l'effet du signal : contrôle ou données.

Les ports de connexion jouent les rôles suivants :

• les échanges d'informations entre le module et le monde extérieur : Les ports peuvent être unidirectionnels ou bidirectionnels.

• le contrôle du module et le déclenchement des opérations définies à l'intérieur du module : Les ports définissent les entrées qui représentent les conditions de contrôle.

L'implémentation définit un ensemble d'opérations. Ces opérations ont les effets suivants :

• Le module change d'état en consommant les valeurs sur les ports d'entrée.

• De nouvelles valeurs sont produites sur les ports de sortie en fonction des valeurs d'entrées et de l'état du module.

Un module peut être défini dans un autre. Cette imbrication détermine les règles de portée de la description.

Dans une description structurelle une telle imbrication n'autorise pas en général l'utilisation des variables déclarées dans le module englobant, car les variables globales ne correspondent pas aux propriétés physiques de l'architecture des modules matériels.

En plus, il n'est pas souhaitable non plus de permettre de référencer

depuis le module global les objets du module local sans passer par son interface.

La modularité facilite la décentralisation de la complexité du problème. La décomposition du circuit assure également la création des bases de données fondées sur des modules compilés.

◊ Modélisation générique

La modularité est souvent liée à la modélisation générique et à l'instantiation. Dans ce contexte, un module formel représente un modèle générique qui détermine les caractéristiques d'un ensemble d'instances réelles.

Une instance est une occurrence ou une copie dont les propriétés sont définies par le modèle générique :

- les propriétés temporelles qui sont relatives : Les valeurs des retards et des gardes temporelles sont paramétrables lors de l'instantiation;
- les propriétés fonctionnelles qui sont absolues : Les mécanismes de contrôle et les types d'opérations sont figés dans le modèle générique.

L'instantiation se fait par l'association des paramètres au modèle et par l'affectation d'une identification à l'instance. Deux types d'instantiation sont possibles :

- l'instantiation des modèles prédéfinis;
- l'instantiation des modèles construits par l'utilisateur.

Le modèle générique définit les attributs communs de ses instances. Les attributs ainsi définis peuvent être vérifiés statiquement ou dynamiquement :

- La vérification statique sur les paramètres se fait au moment de l'instantiation : Par exemple, la validité des valeurs de retards d'un élément peut être vérifiée lorsque l'élément est créé.
- La vérification dynamique sur les ports de connexion se fait lors de l'exécution : Par exemple, une impulsion trop courte ou des signaux hors de séquence.
- La vérification dynamique sur l'implémentation se fait également

lors de l'exécution : Par exemple, l'apparition des opérations interdites ou le débordement de calcul.

Dans Conlan [PIL 85], les instructions ASSERT permettent de définir des attributs et des relations entre les entrées et les sorties :

```
DESCRIPTION dff
        (tsu, th, tp: pint)
        (IN d, ck: signal(bool); OUT
        q, nq: bvar(0))
ASSERT tp>th END
BODY
ASSERT IF ck%th
&~ck%(th+1) THEN
        stable(d,tsu+th)
        & stable0(ck%th,tsu)
        & stable1(ck, th) ELSE 1
        ENDIF ENDASSERT
IF ck%(tp-1)&~ck%tp
THEN q := d, nq := ~d END
ENDdff
```

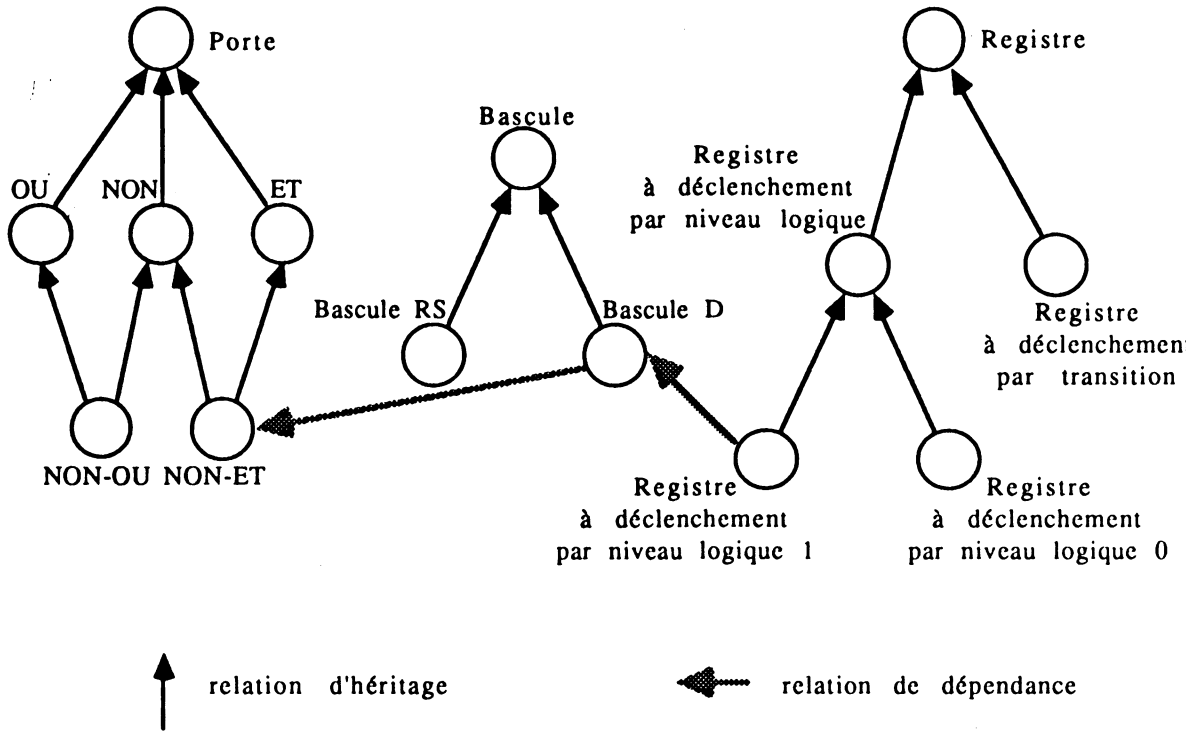
Le langage doit permettre d'exprimer les relations intrinsèques qui existent entre les modèles génériques. Ces relations permettent d'organiser les modèles génériques dans deux sens :

- Verticalement, des modèles génériques sont liées par des relations d'héritage. Le modèle ascendant définit les attributs généraux, tandis que le module descendant ne définit que les attributs complémentaires. Dans certains langages, les modèles génériques s'organisent en classes.
- Horizontalement, il existe des relations de dépendance entre les modèles génériques. Un modèle est dépendant des modèles qui définissent les composants le constituant.

L'exemple suivant montre :

- que le modèle du registre à déclenchement par le niveau logique 1 hérite les attributs du modèle du registre à déclenchement par le niveau logique, qui hérite alors à son tour les attributs communs du modèle du registre;

• que le modèle du registre à déclenchement par le niveau logique 1 dépend du modèle de la bascule D qui dépend lui-même du module de la porte NON-ET.



Relations d'héritage et de dépendance

◊ Hiérarchie

La difficulté de la conception d'un circuit sophistiqué peut être contournée par une description hiérarchique. La hiérarchie du matériel est construite à partir de trois types de mécanismes élémentaires fournis par le langage :

• le mécanisme qui permet de définir des modules avec une fonction spécifique;

• le mécanisme qui permet de réaliser les interconnexions entre les modules;

• le mécanisme qui permet de typer les ports de connexion de chaque module.

Pourtant, des problèmes se posent lorsque l'on compare le langage classique avec celui de description du matériel. L'exécution d'une description

hiérarchique basée sur un langage classique est réalisée par des appels et des retours des procédures hiérarchiques. Cette exécution est une suite d'opérations séquentielles, souvent à l'aide d'une pile de contextes. Alors qu'un module matériel, représentant une partie physique du circuit, n'a aucune équivalence avec une procédure.

Pour un module dont l'implémentation est structurelle, une approche naturelle est de copier selon le modèle générique autant de répliques que le nombre des instances effectives afin d'obtenir une description à plat, qui représente une correspondance complète entre la description et la structure physique du circuit.

La mise à plat peut s'effectuer par les étapes suivantes :

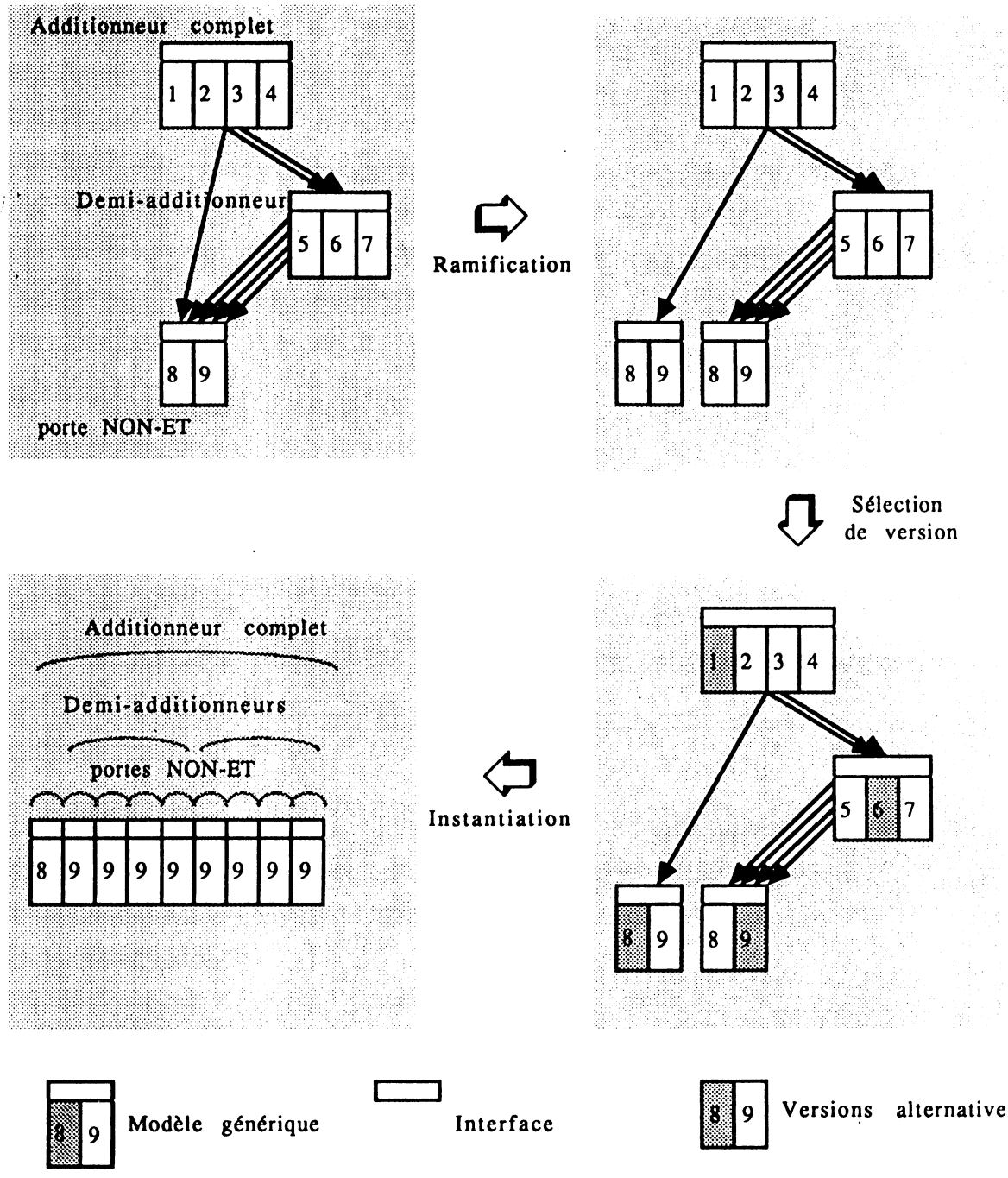
- la ramification qui consiste à répliquer chaque module descendant partagé par plusieurs modules ascendants, afin que le graphe hiérarchique se transforme en une structure arborescente;
- la sélection de version qui consiste à sélectionner, pour chaque module, une version d'implémentation à appliquer dans la construction du circuit;
- l'instantiation qui crée des copies de modules pour obtenir une structure de circuit à plat.

Une autre solution consiste à extraire, tout en gardant la hiérarchie de la description du circuit, des informations nécessaires qui décrivent les interconnectivités de l'ensemble des éléments du circuit.

Pour ce qui concerne les modules dont l'implémentation est comportementale, un autre problème apparaît lorsque l'implémentation est une description qui ne correspond pas à la structure physique du circuit. Dans ce cas, une telle description est constituée d'un ensemble d'opérations qui définissent plutôt un processus dynamique. Un tel processus peut avoir au moins deux états élémentaires : actif et inactif.

Donc la hiérarchisation du circuit doit assurer :

- la temporisation de chaque module : le déclenchement, l'activation et l'inactivation;
- la cohérence entre les modules : la synchronisation et les échanges de données.



Mise à plat d'une structure hiérarchique

2.6.3. Niveaux d'abstraction

Le niveau d'abstraction d'une description représente le niveau auquel le circuit est modélisé, par exemple :

- le niveau système;
- le niveau fonctionnel;
- le niveau logique;
- le niveau interrupteur.

Plus le niveau de description du matériel est bas, plus la description est proche de la réalité physique du matériel, et plus elle converge vers la description structurelle :

- le niveau logique : des portes logiques et leurs interconnexions;
- le niveau interrupteur : des transistors et leurs interconnexions.

Plus le niveau de description du matériel est élevé, plus la description se limite au comportement du matériel, et plus elle converge vers la description comportementale :

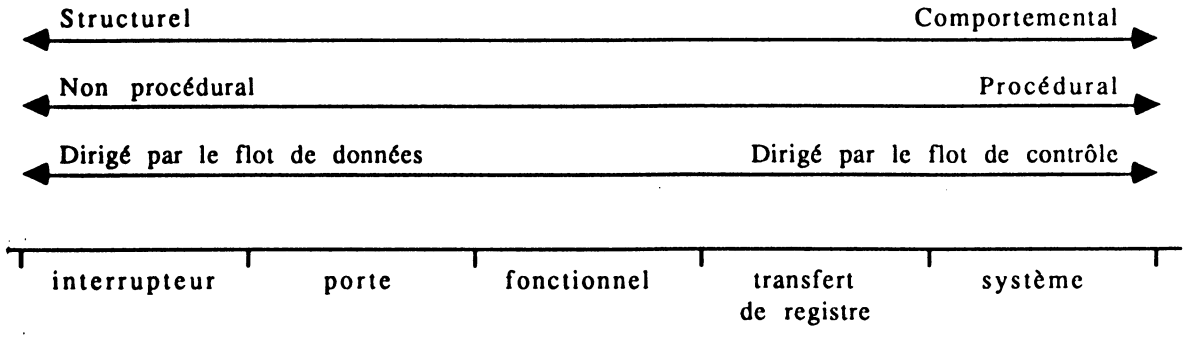
- le niveau fonctionnel : des boîtes noires et des éléments virtuels qui ne correspondent pas directement aux objets physiques du matériel;
- le niveau système : des algorithmes qui calculent des données abstraites.

D'un autre point de vue, si le niveau d'abstraction est bas, on a une connaissance architecturale ou structurelle du matériel. La description est généralement non procédurale.

Si le niveau d'abstraction est élevé, on n'a qu'une compréhension générale du matériel. Par manque de détails du matériel, la description est souvent procédurale.

A propos des flots de contrôle et de données, lorsque le niveau d'abstraction est bas, la topologie du circuit et les interconnexions entre les composants sont connues. Par conséquent, la circulation des signaux et les transferts de données sont déterminés. La description peut être dirigée par le flot de données.

Lorsque le niveau d'abstraction reste assez élevé, seuls les aspects algorithmiques du circuits sont connus. La description est souvent dirigée par le flot de contrôle. Dans ce cas, les structures de contrôle sont très utilisées dans la description pour exprimer les propriétés algorithmiques du circuit.



Relation entre les niveaux d'abstraction et les styles de description

Le niveau de description est un critère important des langages de description. Un langage basé sur un seul niveau est souvent trop limité malgré sa simplicité et son efficacité.

Une possibilité est de développer un langage complexe constitué d'un ensemble de sous-langages, chacun correspondant à un niveau d'abstraction. Les échanges d'informations entre les différents sous-langages sont alors standardisés. Beaucoup de langages offrent plusieurs niveaux d'abstraction, en prédéfinissant des primitives qui s'adaptent aux différents niveaux de modélisation. Cette mesure permet aux langages d'avoir moins de restriction dans les différents domaines d'application.

Un autre solution est plus propre : l'indépendance sémantique. En fait, le niveau d'abstraction est caractérisé par la sémantique des primitives. Un langage, en l'absence de primitives prédéfinies, s'adapte en principe à plusieurs niveaux de description.

2.7. Considérations sur les langages

2.7.1. Caractéristiques principales

◊ Nature temporelle

Le langage de description du matériel est un langage temporel. Il permet d'exprimer la nature dynamique et temporelle du circuit. D'une part, le comportement dynamique des composants du circuit doit être décrit. D'autre part, les relations temporelles entre les composants du circuit et les interactions temporelles entre le circuit et le monde extérieur doivent être représentées :

- Des contraintes temporelles sont associées aux éléments qui représentent des objets constituant le circuit.
- Des actions contrôlées par des gardes temporelles sont utilisées pour représenter les activités temporelles à l'intérieur du circuit et les échanges de données entre le matériel et son environnement.
- Des processus dynamiques peuvent être également utilisés, affectés d'une durée de vie et de deux états élémentaires (actif et inactif).

Les contraintes temporelles sont souvent représentées en terme de retard, affectées aux portes logiques ou aux interrupteurs. Deux types de retards sont définis par un grand nombre de langages : les retards propagationnels et les retards inertiels. Pour les modèles de niveaux plus élevés, la prise en compte du temps est réalisée par la synchronisation ou par des conditions gardées. Dans ce cas, un ensemble d'opérations sont déclenchées par une garde temporelle qui exprime la notion de temps de manière directe (un moment de déclenchement) ou indirectement (un événement auquel est associé un temps).

◊ Parallélisme

Il s'agit de la simultanéité et de la coexistence des activités concurrentes du circuit.

Pour la description structurelle qui est par nature non procédurale, le parallélisme est inhérent à la structure du circuit. Tous les éléments sont supposés prêts à communiquer par leurs interconnexions. Il n'existe pas de flot de contrôle dans la description du matériel.

En ce qui concerne la description comportementale, deux cas se distinguent. Pour une description non procédurale, les opérations élémentaires et les opérations gardées par une condition de contrôle représentent des actions parallèles. Pour une description procédurale, l'exploitation du parallélisme doit respecter les règles définies par le langage. Le déclenchement des opérations dépend des mécanismes de contrôle et de séquencement. Le parallélisme est alors déterminé par des spécifications explicites de la description.

◊ Non récursivité

Il est clair que la nature récursive des langages classiques ne correspond pas à la réalité physique du matériel. En général, les langages de description du matériel n'ont pas de notion de récursivité.

Pourtant, la récursivité peut exister dans l'instantiation récursive ou la construction récursive du circuit, lorsque la description contient des modèles génériques et hiérarchiques. Dans ce cas, le fait qu'un modèle s'appelle lui-même signifie que physiquement un composant est constitué de, ou connecté à, un ou plusieurs composants ayant les mêmes caractéristiques architecturales.

2.7.2. Sémantique de langages

Une description de circuit est une représentation formelle qui décrit la structure et le comportement du circuit. Un langage de description du matériel doit offrir les mécanismes suivants :

- les mécanismes qui traitent et calculent les données : les éléments, les fonctions et les opérateurs, etc.
- les mécanismes qui transportent et mémorisent les données : les équipotentielles et les variables, etc.
- les mécanismes qui réalisent les relations entre le matériel et son environnement : les interconnexions et les stimuli, etc.
- les mécanismes qui réalisent les structures de contrôle : le déclenchement de l'évaluation et les gardes conditionnelles, etc.

Les règles sémantiques de ces mécanismes caractérisent la sémantique du langage de description du matériel.

La sémantique du langage définit un ensemble de règles à respecter. Cet ensemble de règles est conforme à la manière d'implémenter le matériel. D'un autre côté, il détermine l'algorithme des outils informatiques auxquels le langage est associé.

2.7.3. Deux tendances

Du fait que la sémantique du langage influe sur l'implémentation des outils eux-mêmes, Il existe deux approches opposées :

La première approche est de développer des langages dont la sémantique est très orientée vers le matériel. Ces langages fournissent des primitives prédéfinies, chacune correspondant à un ou quelques objets physiques. Dans un sens élargi, les primitives prédéfinies peuvent être des éléments, des opérateurs, ou des structures de contrôle. Un tel langage permet de réaliser des outils informatiques efficaces dans un domaine spécifique, puisque la sémantique du langage peut être impliquée directement dans l'implémentation des outils.

Une autre approche est de développer des langages beaucoup plus généraux, en séparant la sémantique des outils informatiques associés à ces langages. Cette séparation sémantique procure des avantages. D'abord, un tel langage permet à l'utilisateur de définir des types spécifiques selon ses propres besoins et de créer des bibliothèques de références qui lui conviennent. En plus, le langage n'est plus limité à un ou quelques niveaux d'abstraction restreints par la sémantique. Et pour ce qui concerne les aspects dynamiques du langage, la sémantique définit uniquement les propriétés générales du temps, qui permet de construire des modules temporelles de plusieurs degrés de finesse.

Par conséquent, deux tendances se développent dans le domaine des langages de description du matériel.

Une tendance s'oriente vers le développement des langages dédiés aux applications spécifiques. Ces langages sont beaucoup plus proches des aspects matériels du circuit. La description du circuit est enrichie des primitives, ce qui assure une bonne efficacité des outils informatiques. Hilo-3 [GEN 85] est un de ces langages, qui s'adaptent aux applications de la simulation et du test.

Une autre tendance s'appuie sur la rigueur de la programmation conventionnelle. Les langages sont proches des langages classiques. Leur indépendance du matériel vient du fait qu'ils ont peu ou pas de primitives prédéfinies. Ils permettent souvent la description procédurale. Le langage le plus représentatif de ce genre est VHDL [SHI 85] [SHA 85] [NUR 89]. Ces langages ont un large domaine d'application, et sont très utilisés par la synthèse et la vérification formelle.

2.7.4. Hilo-3

Les langages Hilo-3, sont d'origine de *Brunel University* et ensuite commercialisé par GenRad.

Hilo-3 se compose de :

- Langage HDL (*Hardware Description Language*);
- Langage WDL (*Waveform Description Language*);
- Langage SCL (*Simulation Control Language*).

Les détails et la syntaxe de ces langages sont présentés dans les annexes A et B.

Langages HDL, WDL et SCL de Hilo-3 permettent respectivement les descriptions suivantes :

- la description du circuit : la décomposition hiérarchique du circuit, la structure et le fonctionnement du circuit, les fonctions logiques des composants, leurs caractéristiques temporelles, etc.
- la description des interactions : l'initialisation des équipotentielles d'entrée et de sortie du circuit, la stimulation du circuit par des signaux d'entrée, la comparaison des signaux de sortie avec les réponses souhaitées, la création des tâches dynamiques et parallèles qui représentent les interactions entre le circuit et son extérieur, etc.
- la description du contrôle de simulation : les conditions initiales du circuit, les modes de simulation, le contrôle de la visualisation, les injections de pannes, etc.

La sémantique proche du matériel des langages Hilo-3 permet d'implémenter le simulateur de manière très spécifique et efficace. Pourtant,

la dépendance sémantique peut rendre l'algorithme du noyau du simulateur fortement subordonné aux langages.

La nature non procédurale de la description du circuit, dirigée par les flots de données et d'événements, permet une exploitation relativement facile du parallélisme dans le simulateur.

Pour ce qui concerne la modélisation du temps, les langages permettent d'une part la description des contraintes temporelles associées aux éléments, d'autre part le déclenchement temporisé des activités dynamiques.

Les langages Hilo-3 permettent la structuration de données. Par contre, la modélisation générique, la modularité et la hiérarchisation de circuits sont restrictives.

Les langages de Hilo-3 sont parmi les langages les plus répandus dans l'industrie. Ils sont très utilisés actuellement dans le domaine de la simulation logico-fonctionnelle et le test du circuit. C'est la raison pour laquelle ils sont adoptés par LL3T pour réaliser son environnement de simulation.

Conclusion

Le simulateur logico-fonctionnel est un outil élémentaire de la conception de circuits intégrés. Parmi de nombreuses approches, l'algorithme de l'échéancier, la méthode la plus efficace et la moins restrictive, est très utilisée.

La méthode de maximisation de la simulation de transistors permet d'une part de résoudre des problèmes épineux de la modélisation des éléments bidirectionnels, et d'autre part d'obtenir des résultats ni trop pessimistes ni trop optimistes.

La simulation des modèles fonctionnels nécessite l'utilisation de l'algorithme de double phase. De nouveaux types d'événements sont nécessaires pour représenter les changements d'état sur les entrées d'éléments fonctionnels et les changements d'état sur les variables internes d'éléments.

Le meilleur algorithme destiné à la simulation de pannes est la méthode concurrente. Celle-ci permet l'utilisation de n'importe quel type de primitives. L'évaluation des primitives n'est pas sensible aux types de pannes introduites dans la simulation. Cette méthode suppose que les pannes se trouvent à l'extérieur des objets élémentaires. Donc l'algorithme peut être implémenté dans un simulateur à échéancier pour partager les mêmes mécanismes de simulation et les mêmes structures de données.

Quant aux langages de description du matériel, les discussions ont évolué sur les problèmes suivants :

- Les principales propriétés des descriptions sont basées sur les trois paires de notions : les descriptions non procédurale et procédurale, les descriptions structurelle et comportementale, les descriptions dirigées par les flots de données et de contrôle.
- Le parallélisme, la nature temporelle, et la non récursivité sont les caractéristiques fondamentales des langages de description du matériel. Ils représentent les différences importantes entre un langage de description de matériel et un langage de programmation classique.
- Au niveau de la structuration de données, la modularité, la modélisation générique et la hiérarchisation sont les problèmes clefs.

Chapitre 3

Techniques d'accélération de la simulation et Solutions proposées

Plan du chapitre 3

Introduction

Ce chapitre concerne les techniques d'accélération. Différentes approches y sont discutées après mise en évidence de la nécessité des accélérateurs de simulation.

Un ensemble d'analyse porte sur les algorithmes de simulation et sur les éventuelles améliorations à apporter :

- au noyau des algorithmes;
- aux structures de données.

Différentes accélérations matérielles et les problèmes soulevés par ces techniques sont présentés de manière approfondie. Ces techniques d'accélération matérielle sont basées sur :

- le jeu d'instructions spécifiques;
- l'architecture pipeline;
- l'architecture parallèle.

Nombreuses stratégies de développement des accélérateurs vont être discutées selon certains critères, ainsi que leurs caractéristiques décrites. Ces stratégies peuvent être classées, d'une façon simpliste, comme suit :

- les accélérateurs spécifiques;
- les accélérateurs généraux;
- les machines générales;
- les accélérateurs à base de processeurs généraux.

Enfin, des accélérateurs récemment développés seront présentés. L'accent sera mis sur les aspects suivants :

- leur architecture globale;
- leur principe de fonctionnement;
- l'ensemble des logiciels développés autour de ces accélérateurs;
- leurs performances totales.

Le but de la présentation de ces accélérateurs est d'établir un bilan sur la situation actuelle du développement des accélérateurs de simulation, et d'en dégager une conclusion générale.

3.1. Généralités

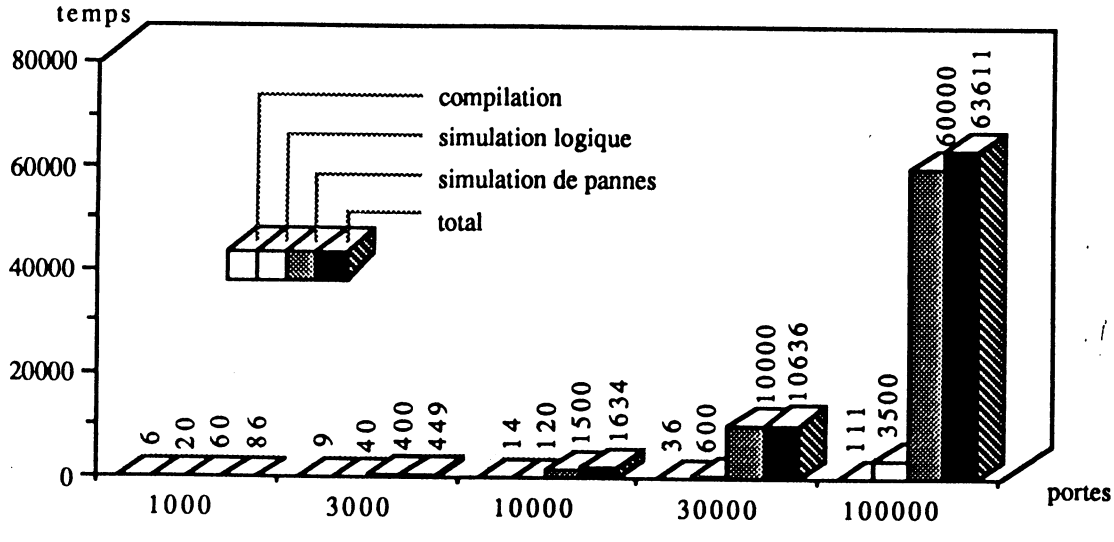
3.1.1. Nécessité des moyens d'accélération

Avec la croissance de la complexité des circuits intégrés, la simulation consomme de plus en plus de temps de calcul. Les techniques d'aujourd'hui autorisent la réalisation des circuits sophistiqués qui nécessitent des jours, même des mois de temps de calcul avec des simulateurs conventionnels. Le problème est sérieux : avec un circuit de taille n, le temps consommé par la simulation logique et par la simulation de pannes sont de l'ordre de grandeur de n^2 et n^3 respectivement [MOT 85]. D'autres statistiques ont été faites qui indiquent qu'en général, une simulation de pannes dépense 10^1 à 10^2 fois plus de temps de calcul qu'une simulation logique sur un même circuit, et que la simulation électrique est 10^4 à 10^5 fois plus lente que la simulation logique [LEW 88b].

Une analyse sur le temps de calcul consommé dans un cycle de simulation a été faite, dans lequel on poursuit les étapes suivantes :

- compiler le circuit logique et les stimuli;
- effectuer la simulation logique;
- analyser les résultats, retourner à la compilation si nécessaire;
- effectuer la simulation de pannes;
- analyser les résultats, retourner à la compilation si nécessaire.

La statistique sur le temps de calcul de ces opérations est obtenue en terme de temps relatif [SHE 87] :



Il est évident que le temps de compilation augmente de façon linéaire par rapport à la croissance du nombre de portes. Pourtant, le temps de la simulation logique et celui de la simulation de pannes augmentent proportionnellement beaucoup plus vite.

Supposons que la simulation logique d'un circuit de 10000 portes sous certains stimuli dure 2 heures, alors, pour un circuit de 100000 portes, la simulation durera plus de 2 jours, et la simulation de pannes consommera pratiquement plus de 1 mois de temps de calcul. La simulation électrique du même circuit sera dramatiquement impraticable. Donc, il est impératif d'étudier des moyens d'accélération pour réduire le temps de calcul, de sorte que la simulation coûte le moins cher possible pour éviter le goulot d'étranglement de la conception du circuit.

3.1.2. Techniques d'accélération

Afin d'améliorer les performances des simulateurs, deux types de mesures peuvent être prises :

- les améliorations logicielles;
- les accélérations matérielles.

Au niveau purement logiciel, les performances de simulateurs peuvent être améliorées par :

- l'optimisation du noyau des algorithmes.
- l'utilisation des structures de données spéciales;

Du côté matériel, des machines dites accélérateurs de simulation sont spécialement développées. Ces accélérateurs possèdent une configuration ou une architecture matérielle bien adaptée à la simulation. Les techniques de base utilisées par les accélérateurs sont :

- le jeu d'instructions spécifiques;
- l'architecture pipeline;
- l'architecture parallèle.

3.2. Améliorations des logiciels

3.2.1. Utilisation des structures de données spéciales

Certains efforts ont été faits pour optimiser les algorithmes de simulation, bien que les améliorations de performances soient assez limitées.

L'utilisation des langages spéciaux qui manipulent les structures de données spécifiques permet de réduire le nombre d'instructions nécessaires au calcul. De plus, les parties critiques du simulateur peuvent être programmées directement en assembleur.

Les simulateurs par code compilé usent de toutes les techniques d'optimisation de code destinées aux compilateurs traditionnels, telles que l'analyse de flot de données, la transformation d'expressions [HAN 88], afin d'optimiser les structures de données et de réduire le nombre d'opérations dans le code exécutable.

Pour les simulateurs à échéancier, les événements, l'échéancier et les éléments logiques sont des objets spécifiques. L'utilisation de structures de données spéciales permet de minimiser la mémoire consommée, de simplifier la gestion des événements et de réduire le temps de l'évaluation des éléments. Par exemple, un élément comme le registre peut avoir une entrance et une sortance bien élevées. La structure vectorielle permet de regrouper des informations dans un seul vecteur. Un vecteur peut représenter plusieurs changements d'états qui se produisent au même moment sur des entrées d'un registre. Le simulateur n'évalue qu'une fois le registre avec le vecteur d'entrée et génère un seul événement vectorisé pour toutes les sorties qui devront changer d'état.

3.2.2. Optimisations du noyau des algorithmes

Un modèle fonctionnel dépense moins de temps de calcul qu'un même modèle construit par des portes. Le rapport de vitesse est environ 10^1 à 10^3 selon les algorithmes [PFI 86]. Un simulateur multi-niveaux permet de modéliser une partie du circuit à simuler en blocs fonctionnels afin d'éviter les calculs trop détaillés. D'autant plus que certains simulateurs permettent d'utiliser directement des langages de programmation évolués, comme Pascal, C ou Fortran [DOS 84].

Certains simulateurs [ABR 83] utilisent l'algorithme de simple phase pour réduire le temps de calcul. Par contre, la dégradation de vitesse peut s'aggraver lorsque les éléments fonctionnels sont introduits dans la simulation. Les éléments fonctionnels ont en général des entrées plus nombreuses que celles des portes logiques. La simulation de simple phase risque de recalculer le même élément fonctionnel inutilement, et de faire l'élément fonctionnel ayant des variables internes entrer dans un état parasite. Une autre stratégie est employée par le simulateur ULTIMATE [GLA 84] qui consiste à séparer les éléments en deux catégories : les éléments primitifs avec une petite entrance, et les éléments non primitifs avec une grande entrance. A chaque pas de simulation le simulateur traite les éléments primitifs en une simple phase, et les éléments non primitifs en double phase.

Un circuit complexe nécessite plusieurs cycles de simulation répétitifs pour mettre au point le fonctionnement du circuit. A chaque reprise de simulation, le circuit est simulé à la suite des modifications parfois assez minimes. Des algorithmes (*increment-in-space, increment-in-time*) [CHO 88] [HWA 89] récemment développés permettent aux simulateurs de fonctionner dans un environnement incrémental. Le simulateur conserve alors les informations historiques du circuit simulé. Lorsque l'on répète une simulation, le simulateur est capable de ne traiter que ce qui est modifié par rapport à la dernière version du circuit.

3.3. Accélérations à l'aide des matériels

3.3.1. Machines au jeu d'instructions spécifiques

Une instruction machine se compose de deux informations : l'opérateur et les opérandes. D'une part, des études ont été menées pour analyser les opérations les plus utilisées dans la simulation, afin de les coder directement en instructions machines avec des opérateurs spécialement définis. D'autre part, au niveau des opérandes, on essaie de maximiser les informations inclues dans une opérande, pour que des évaluations puissent se faire simultanément en une seule instruction. D'ou l'utilisation des machines vectorielles et des architectures SIMD.

Une machine avec un jeu d'instructions spécifiques peut accélérer considérablement la simulation :

• Les simulateurs par code compilé génèrent directement des codes exécutables. Une technique très efficace est de développer des machines qui ont un jeu d'instructions propre à la simulation, ce qui correspond à la philosophie de cette méthode de simulation : les performances dépendent de l'efficacité du code exécutable. Toutes les fonctions complexes peuvent être également câblées par les instructions. L'évaluation d'une porte ne peut coûter que 2 instructions [ZAS 87], alors qu'environ 25 instructions sont nécessaires sur une machine générale.

• Quant aux simulateurs à échéancier, les points critiques se trouvent dans le gestionnaire de l'échéancier et le module de l'évaluation. En général, la gestion de l'échéancier et l'évaluation des éléments exigent des structures de données particulières et des opérations spécifiques, et consomment trop d'instructions machines. Toutefois, le noyau de l'algorithme des simulateurs est stable et la plupart du temps de calcul se concentre sur l'exécution de ce noyau. Une machine spécialement microprogrammée [ABR 83] permet de coder le noyau de l'algorithme en peu d'instructions et accélère sensiblement la vitesse de simulation.

• Pour les modèles interrupteurs, les opérateurs T et C sont des calculs élémentaires et peuvent être câblés en priorité. L'initialisation des noeuds et le contrôle des itérations sont des opérations qui se répètent dans chaque évaluation. L'utilisation des instructions spécifiques permet d'avoir une bonne efficacité de simulation. Certains algorithmes exigent parfois des opérations algébriques spécifiques, ce qui conduit également à une optimisation sur le jeu d'instructions.

• En ce qui concerne l'optimisation des opérandes d'instructions, une mesure souvent adoptée est l'utilisation de la machine vectorielle. Il existe dans un simulateur logique un nombre limité de types de portes. Chaque type implique un opérateur logique. Une machine vectorielle est capable d'évaluer des portes logiques simultanément. Cette technique est aussi efficace pour la simulation à échéancier [KAZ 88] que pour la simulation par code compilé [ISH 87]. D'autre part, les modèles fonctionnels ont en général une entrance et une sortance assez élevées, conduisant à l'utilisation des événements vectoriels. Par ailleurs, les machines vectorielles s'adaptent parfaitement à la simulation de pannes basée sur la méthode parallèle. En ce qui concerne les simulations de pannes concurrentes, les évaluations d'un module et de ses modules concurrents peuvent être également vectorisées. Dans [SON 85] est présenté un simulateur de pannes qui utilise la méthode concurrente en codant sur une opérande d'instruction machine plusieurs états logiques, qui peuvent être calculés simultanément.

3.3.2. Machines pipeline

La technique de pipeline semble relativement facile à introduire dans les simulateurs par code compilé et les simulateurs à échéancier :

• Pour le simulateur par code compilé le mécanisme de pipeline est très classique : le chargement de l'instruction, l'analyse de l'instruction, la lecture de données, l'exécution de l'instruction et l'écriture des données.

• En ce qui concerne le simulateur à échéancier, le module de la gestion de l'échéancier et le module de l'évaluation sont les deux dispositifs, ayant la relation producteur-consommateur et vice versa, qui peuvent être organisés de manière pipeline. Ces deux modules peuvent être raffinés en trois modules pipelinés : l'évaluation des éléments, la mise à jour des états des équipotentielles et la gestion des événements. A l'intérieur d'un module, les étapes sont également pipelinables. Par exemple, le module de l'évaluation peut se décomposer en plusieurs étapes : la récupération et l'analyse de l'événement, l'évaluation de la fonction de l'élément et la création des nouveaux événements. Dans le simulateur ULTIMATE [GLA 84], le mécanisme de pipeline est constitué de 11 étapes.

• Le mécanisme de pipeline s'applique également aux simulateurs ayant une extension fonctionnelle ou interrupteur. L'évaluation d'un groupe de transistors s'effectue en étapes : la mise à jour des états des transistors, l'initialisation des noeuds, le calcul des nouveaux états et la création des événements. L'évaluation d'un élément fonctionnel peut être aussi en étapes

de même manière que celle d'une porte logique. Toutefois, pour que le pipeline soit efficace, une condition est indispensable : toutes les étapes du pipeline consomment un temps de calcul identique ou approché. Pour les transistors, le calcul des nouveaux états exigent une opération itérative qui se répète jusqu'à ce que les états sur les noeuds se stabilisent. En ce qui concerne les éléments fonctionnels, la complexité des opérations impliquées dans l'évaluation varie selon l'algorithme défini par le modèle fonctionnel. Il est difficile d'estimer la durée moyenne de calcul. Il se peut que les modèles fonctionnels consomment trop de temps sur une étape d'évaluation, et par conséquent, immobilisent l'ensemble du pipeline. Une solution consiste à différer, à chaque pas de simulation, l'évaluation des modèles qui risquent de bloquer une ou plusieurs étapes du pipeline. Une autre solution est d'utiliser un coprocesseur qui calcule les éléments non primitifs, pendant que les éléments primitifs sont calculés sur le pipeline. Dans [ABR 83], l'étape d'évaluation du pipeline est dotée d'un évaluateur logique et de plusieurs évaluateurs fonctionnels. Cette mesure évite efficacement des blocages sur l'étape d'évaluation du pipeline.

3.3.3. Machines parallèles

La technique de parallélisme consiste à répartir les tâches de simulation sur plusieurs processeurs, de sorte qu'elles se poursuivent en parallèle sur ces processeurs :

• Dans les simulateurs par code compilé parallèles, le circuit est partitionné et réparti sur les processeurs qui fonctionnent de façon synchronisée. Les éléments du circuit sont statiquement distribués sur les processeurs, et chaque processeur évalue à chaque pas tous les éléments qui lui sont distribués. Les processeurs sont synchronisés à la fin du pas pour assurer que tous les nouveaux états du circuit sont évalués. Pourtant, cela ne résout pas le problème de redondance de la simulation par code compilé : Tous les éléments sont évalués à chaque pas, même si le taux d'activités des éléments du circuit est très bas.

• La parallélisation des simulateurs à échéancier est basée également sur le principe du partitionnement du circuit. Chaque processeur simule une partie du circuit en utilisant un échéancier local propre au processeur. Alors le problème se pose sur la gestion globale de ces échéanciers locaux. D'une part, un événement porte fréquemment des informations non locales. Les échanges d'informations entre des processeurs risquent de coûter cher. D'autre part, pendant la simulation tous les processeurs doivent être

synchronisés à la même heure de simulation. A chaque pas de simulation, les processeurs surchargés rendent les autres dans un état d'attente. D'autant plus que le partitionnement du circuit est un algorithme très complexe et difficile à optimiser pour éviter les surchargements de processeurs et les communications supplémentaires entre les processeurs.

• Les modèles d'interrupteurs du circuit peuvent être distribués sur des processeurs fonctionnant en parallèle. Ce qu'il faut éviter dans ce cas est d'avoir des connexions bidirectionnelles de transistors entre les processeurs [THA 87]. De l'autre côté, l'algorithme de l'évaluation des modèles fonctionnels peut être très différent d'une porte logique. Certains simulateurs dotés d'évaluateurs fonctionnels séparent ces deux types d'évaluations en plaçant les modèles fonctionnels sur des évaluateurs fonctionnels qui travaillent en parallèle [ABR 83]. Lorsque la configuration d'un élément fonctionnel se trouvent sur un autre processeur, une communication est établie avant l'évaluation de l'élément pour obtenir les informations nécessaires. Le problème se pose lorsque l'élément dispose d'un grand nombre de données, tel que les mémoires. Dans ce cas, une portion minimale nécessaire à l'évaluation, par exemple, un seul mot de mémoire, doit être sélectionnée et transmise. Une alternative est de placer ce genre de données sur une mémoire accessible par tous les processeurs.

• Pour ce qui concerne la simulation de pannes, notamment la méthode concurrente, l'exploitation du parallélisme paraît sans peine. En raison du nombre très élevé de pannes dans le circuit (plusieurs pannes par noeud), une stratégie consiste à diviser toutes les classes de pannes en certains groupes, en fonction du nombre de processeurs disponibles. Chaque processeur simule le circuit entier avec un groupe de pannes distinctes, tout en parallèle avec les autre processeurs. Les pannes peuvent être groupées de façon qu'un groupe de pannes n'infecte qu'une partie du circuit. Donc pour chaque processeur, seule la partie contenant des pannes est modélisée au niveau logique, le reste du circuit pouvant être décrit au niveau plus élevé (la description fonctionnelle, par exemple), afin d'améliorer la vitesse de simulation.

Pour bénéficier au maximum des techniques parallèle et pipeline, certains accélérateurs combinent ces deux techniques ensemble. Ils sont constitués de plusieurs unités fonctionnant en parallèle, au sein desquelles les traitements sont organisés en pipeline. En plus, les étapes du pipeline sont microprogrammables ou dotées d'un jeu d'instructions spécifiques.

3.4. Stratégies de développement des accélérateurs

3.4.1. Accélérateurs spécifiques

Les accélérateurs spécifiques sont spécialement conçus et optimisés autour des algorithmes de simulation. Ils ont un jeu d'instructions dédiée à la simulation. Ils ont des mécanismes matérialisés pour les traitements pipeline. Ils peuvent être basés sur une architecture multiprocesseur. Le parallélisme peut être assez massif et le système est souvent reconfigurable selon le nombre de processeurs intégrés dans le système.

Ce genre d'accélérateurs est très puissant. Il offre une vitesse au moins 10^3 fois plus élevée qu'une machine générale, et le coût du développement d'un tel accélérateur reste bien inférieur à 10^3 fois plus que celui d'une machine générale [LEW 88b].

Un exemple d'accélérateur spécifique dédié à la simulation par code compilé est la machine YSE [PFI 86] développée par IBM. La vitesse de la configuration maximale de 256 processeurs logiques atteint $3 \cdot 10^9$ portes par seconde avec une capacité de $2 \cdot 10^6$ portes.

L'accélérateur spécifique SP [SAI 88] est un système de simulation à échéancier. Le système peut avoir 64 processeurs de portes et permettre de simuler un circuit de $4 \cdot 10^9$ portes à une vitesse de $8 \cdot 10^8$ évaluations par seconde.

3.4.2. Accélérateurs généraux

La réalisation d'un accélérateur spécifique est très coûteuse, car il est développé autour d'un algorithme spécifique de simulation. Un algorithme très câblé accélère sensiblement la simulation. Pourtant, si l'algorithme évolue, l'architecture de l'accélérateur sera mise à jour de nouveau pour s'adapter au nouvel algorithme.

Par ailleurs, les accélérateurs spécifiques nécessitent un algorithme stable pour résoudre un problème stable. Le problème stable n'existe pas, car cela ne correspond pas aux progrès des technologies de fabrication de circuits VLSI.

Le développement d'un logiciel de simulation est long et coûteux. Son

indépendance par rapport à la machine est un facteur non négligeable. Il est souhaitable que le logiciel de simulation ait une longue durée de vie et qu'il soit facile de le transplanter d'un accélérateur à un autre.

Afin d'éviter ces problèmes, des machines dites accélérateurs généraux sont spécialement conçus et développés, tels que l'accélérateur MegaLogician [CAR 87a]. Ces accélérateurs ne sont pas destinés à une application particulière. Ils ont en général une architecture pipeline et parallèle. Leur noyau est souvent microprogrammable pour que ces accélérateurs puissent être reconfigurés selon les algorithmes de simulation.

Ce type d'accélérateurs peut supporter d'autres outils de CAO comme le générateur de tests, le compilateur de silicium etc.

MARS [AGR 87a] [AGR 87b] [AGR 90] est aussi un accélérateur général. Lorsqu'il est microprogrammé et configuré en un simulateur logique, sa vitesse peut atteindre 10^6 portes par seconde.

3.4.3. Machines générales

Il existe des réseaux de stations de travail et des machines multiprocesseurs. Bien que leur architecture ne s'adaptent pas parfaitement à la simulation parallèle, ils offrent des mécanismes efficaces pour la gestion des traitements multi-tâches. Des algorithmes sont développés autour de ces architectures.

Le développement des machines générales avance très vite grâce à l'utilisation de nouvelles technologies, alors que le renouvellement d'un accélérateur de simulation nécessite parfois une reconception totale pour bénéficier des nouvelles technologies. Comme les accélérateurs de simulation font toujours appel au parallélisme massif basé sur une architecture géante et complexe, leur renouvellement risque d'être difficile et coûteux.

Des algorithmes de simulation sont considérablement accélérés par l'exploitation du parallélisme et du pipeline pour réaliser des simulateurs économiques et à long terme. Toutefois le développement des simulateurs sur les machines générale est difficile. Il n'existe pas d'algorithme de simulation purement parallèle sans tenir compte de l'architecture de la machine sur laquelle il est implémenté, malgré le parallélisme inhérent à ces algorithmes. Deux problèmes restent sophistiqués : comment partitionner

une tâche en sous-tâches (*task partitionning*) et comment allouer ces sous-tâches sur les processeurs (*task allocation*).

Parmi les machines générales, les machines multiprocesseurs sont préférables pour accélérer la simulation, que ce soit au niveau architectural comme les machines de flot de données [ABR 88b], ou au niveau logiciel comme la programmation orientée-objets [MEH 87]. La machine Balance 8000 constituée de 10 à 12 processeurs NS32032 a été utilisée par deux équipes distinctes [YOS 87] et [ODE 87] pour développer des simulateurs parallèles.

Les machines générales ayant une architecture parallèle sont typiquement avantagées pour la simulation de pannes concurrentes. L'ensemble des pannes peuvent être divisées en groupes selon le nombre de processeurs disponibles ou le nombre de tâches qui peuvent coexister dans la machine. Une alternative est de laisser l'utilisateur distribuer les pannes sur les processeurs [ROG 87] [DUB 88].

3.4.4. Accélérateurs à base de processeurs généraux

Contrairement au développement des accélérateurs spécifiques et généraux qui consiste à concevoir entièrement le matériel, une autre approche est de réaliser des accélérateurs à base de processeurs généraux. Il existe des processeurs, comme les processeurs RISC, idéaux pour constituer une configuration pipeline et parallèle permettant d'effectuer des tâches de simulation.

Par rapport aux accélérateurs spécifiques et généraux, ce genre de développement réduit le temps de réalisation des accélérateurs et minimise le coût du matériel. Sans contraintes architecturales, les accélérateurs de ce type peuvent intégrer facilement différents sortes de simulateurs, même différents outils de CAO.

Par rapport aux machines générales de type multiprocesseur, l'architecture des accélérateurs à base de microprocesseurs est plus facile à reconfigurer. La gestion de multi-tâches est plus orientée et plus efficace.

La conception parallèle nécessite des outils de CAO plus commodes et plus personnels. C'est un besoin très fort qui pousse le développement des accélérateurs contrôlables par des ordinateurs individuels et par des stations

de travail. Ce type d'accélérateur est économique, permettant de créer des environnements personnels qui assurent à toute équipe de conception de travailler en parallèle avec une productivité bien améliorée.

Dans [MEH 87] ont été recommandés les processeurs RISC après avoir comparé les performances d'un simulateur expérimental SILKE implémenté sur plusieurs type d'accélérateurs. Hylas I et Hylas II sont les simulateurs multi-niveaux (fonctionnel, logique, interrupteur et électrique) implémentés sur les accélérateurs construits à base de processeurs généraux : une configuration 68000-8087, une machine VAX 11/785 avec accélérateur, et une architecture basée sur transputers T414 et T800 développés par INMOS [DYS 87].

Un autre système a été développé qui fonctionne sur un hypercube de processeurs iPSC [THA 87].

3.5. Problèmes liés au parallélisme

3.5.1. Problème de la synchronisation inter-processeur

Les processeurs de simulation peuvent fonctionner de façon synchrone ou asynchrone. En cas de parallélisme asynchrone, deux stratégies (*worst case strategy, optimistic strategy*) [MEH 87] sont utilisées pour résoudre le problème de conflit entre les processeurs.

En général, la synchronisation est réalisée par les messages de synchronisation qui circulent entre les processeurs. A la fin de chaque pas de simulation, chaque processeur envoie aux processeurs voisins ou à l'unité d'arbitrage un message de synchronisation véhiculant l'heure du prochain pas proposée par ce processeur. Avant la réception de la réponse, chaque processeur reste dans un état d'attente.

Une méthode dite retour en arrière (*roll back*) est utilisée dans certains simulateurs, pour réaliser le parallélisme asynchrone. Chaque processeur simule, indépendamment des autres, une partition du circuit. S'il simule trop loin, c'est-à-dire qu'il reçoit un événement en provenance d'un autre processeur, qui aurait dû être traité dans un temps passé T, alors tous les faux événements qui se sont produits après T doivent être supprimés par l'utilisation d'anti-événements [SOU 88].

Dans un simulateur dont le degré de parallélisme est faible, le circuit à simuler peut être mémorisé, au lieu d'être partitionné sur les processeurs de simulation, sur un processeur de gestion de ressources accessible par tous les processeurs de simulation. A chaque pas de simulation, si un processeur termine le traitement des événements qui lui sont distribués, sans attendre il cherche et reprend une partie d'événements à traiter dans l'échéancier d'un processeur surchargé. Cet équilibrage dynamique coûte peu cher puisqu'il est accompli par un processeur qui est dans un état d'attente.

Une autre approche permettant aux processeurs de fonctionner de façon asynchrone consiste à utiliser l'algorithme T (*time-first evaluation algorithm*) [ISH 87]. Contrairement aux algorithmes conventionnels, les événements ne sont pas évalués dans l'ordre chronologiques de simulation. Ils sont évalués immédiatement dès qu'ils sont disponibles, quelle que soit l'heure de ces événements. Par contre la simulation devient très difficile à contrôler en cas de présence des boucles de connexions dans le circuit.

Il est constaté que le parallélisme synchrone est facile à introduire dans les algorithmes de simulation. Le problème majeur est que les processeurs surchargés peuvent rendre les autres entièrement inactifs. Cela diminue sensiblement l'efficacité de l'ensemble du système, surtout lorsque le parallélisme est massif.

Le parallélisme asynchrone offre une meilleure efficacité. Par contre il est plus difficile à introduire dans les algorithmes de simulation. Cette difficulté est d'autant plus importante que le circuit est séquentiel.

3.5.2. Problème du partitionnement du circuit

Les simulateurs multiprocesseurs nécessitent le partitionnement de circuit qui découpe un circuit en composants entre lesquels le couplage doit être le plus faible possible. Ces composants doivent représenter une complexité identique ou approchée, nécessiter peu de communications entre eux, et avoir peu d'informations globales à accéder.

La plus simple technique est le partitionnement linéaire. On distribue les N premiers éléments sur le premier processeur, les N deuxièmes éléments sur le deuxième processeur, etc. Cette méthode nécessite peu de temps de calcul.

Le partitionnement par la connectivité consiste à regrouper les éléments, fortement reliés par leurs entrées et sorties, dans un même processeur afin de minimiser les communications inter-processeurs. Cette méthode convient non seulement à la simulation à échéancier mais aussi à la simulation par code compilé. Les simulateurs YSE [PFI 86] et EVE [BEE 88] adoptent cette méthode de partitionnement.

Un grand nombre de machines multiprocesseurs utilisent soit une matrice de communication soit des bus de haute vitesse pour établir les communications inter-processeurs. Cependant, il existe des architectures ayant une topologie spéciale, comme la topologie hypercube, ou le réseau cellulaire. Pour ce genre d'architectures, le partitionnement et la répartition du circuit doivent prendre en compte la distance logique entre les processeurs [THA 87].

Pour les simulateurs qui acceptent les modèles de transistors, le

partitionnement du circuit doit respecter en priorité la division des transistors en graphes (ou groupes), car un graphe définit une unité qui regroupe des transistors. Cette unité peut être évaluée comme un seul composant du circuit lors de la simulation. Le calcul de l'évaluation est souvent itératif et intensif, avec des propagations des états. Il est souhaitable de ne pas partitionner un tel graphe sur plusieurs processeurs.

La description hiérarchique du circuit permet un partitionnement naturel, du fait qu'un circuit peut être décomposé en un ensemble de sous-circuits et que les sous-circuits peuvent être décomposés de la même façon. Le partitionnement consiste à décomposer les circuits en leurs sous-circuits et les regrouper en groupes, jusqu'à ce que le nombre de groupes soit égal au nombre de processeurs et que les groupes aient approximativement la même taille.

Le partitionnement peut être déterminé par l'utilisateur [SAI 88]. La notion de bloc peut être introduite dans la description du circuit pour définir un assemblage d'éléments spécifié par l'utilisateur lorsqu'il décrit le circuit. Le circuit se compose donc d'un certain nombre de blocs. Et naturellement, le partitionnement du circuit respecte le découpage en blocs [TAK 87].

Quelles que soient les méthodes employées, elles doivent assurer l'équilibrage du partitionnement de circuit :

- statiquement au niveau de l'espace mémoire consommé par les processeurs;
- dynamiquement au niveau du temps de calcul des processeurs.

A ce propos, il semble que la méthode linéaire et la méthode par la hiérarchie fonctionnent bien pour les circuits hiérarchiques et réguliers, et que la méthode par la connectivité s'adapte à tous types de circuits.

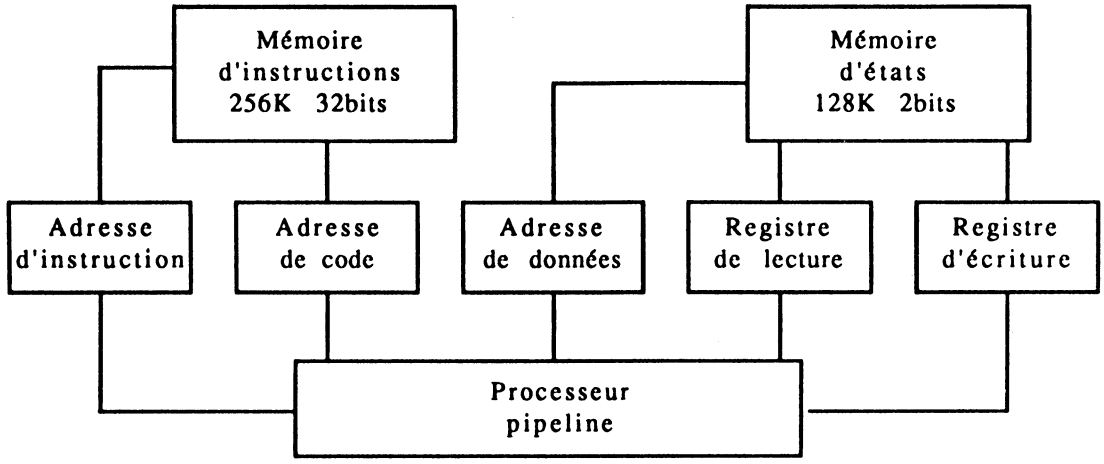
3.6. Accélérateurs de simulation actuels

3.6.1. AIDA

AIDA [ZAS 87] est un accélérateur à architecture RISC, commercialisée par la société AIDA. C'est un simulateur par code compilé qui utilise 4 états (0, 1, X et Z) et 10 primitives logiques.

L'ensemble de l'accélérateur est composé de :

- une mémoire dynamique d'instructions;
- une mémoire statique d'états;
- un processeur pipeline.



Architecture générale

Un pipeline de 7 étapes est utilisé dans la simulation :

- incrémenter le compteur et charger l'adresse de l'instruction;
- lire l'instruction dans la mémoire d'instructions;
- décoder le code et l'adresse de l'instruction lue;
- lire l'opérande dans la mémoire d'états;
- exécuter l'opération de l'instruction;
- stocker le résultat dans le registre d'écriture;
- écrire le résultat dans la mémoire d'états.

Le processeur pipeline dispose d'une architecture RISC. Le noyau du jeu d'instructions est constitué de 16 instructions dont les opérateurs sont codés en 4 bits. Chaque instruction manipule deux opérandes. L'exécution

d'une instruction est pipelinée en trois étapes :

- le décodage de l'instruction;
- l'accès aux données;
- l'opération logique.

Le code exécutable généré par le compilateur est très compact et efficace.

Une comparaison entre le simulateur purement logiciel et le simulateur à base d'accélérateur est donnée :

	Approche logicielle	Approche d'accélérateur
Nombre d'instructions par porte	25	2
Temps pour une simulation de 500K portes	12,5 secondes	0,1 secondes
Avec 300 cycles d'horloge	1 heure	30 secondes

Afin de mesurer les performances de la simulation, un circuit de 102100 portes est simulé avec un séquence de 2688 stimuli. Les résultats sont les suivants :

- 84 secondes de temps de compilation;
- 136450 instructions générées;
- 50 secondes de temps de simulation;
- 21% de taux d'activité de portes.

Ce qui donne les performances suivantes :

- le nombre d'instructions par porte : 1,34;
- La vitesse de simulation : $5,5 \cdot 10^6$ portes par seconde.

3.6.2. HAL

HAL (*Hardware Logic Simulator*) [TAK 87] est un simulateur logique développé par NEC.

L'objectif du développement de HAL est de construire un accélérateur permettant la simulation des gros circuits et systèmes. Le simulateur utilise

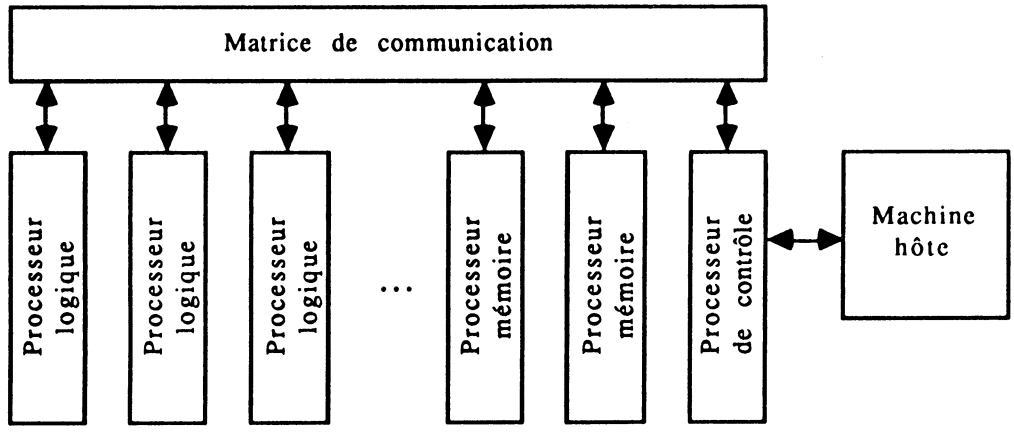
la logique à deux valeurs 0 et 1, et le retard nul. Il permet la simulation aux niveaux porte et fonctionnel.

La notion de bloc est introduite dans la modélisation du circuit à simuler. Un bloc est une unité fonctionnelle qui peut être :

- un bloc logique qui représente un ensemble de portes logiques;
- un bloc mémoire qui représente un ensemble de mémoires.

HAL dispose d'une architecture de 32 processeurs dont les échanges d'informations sont réalisés par une matrice de communication. Deux configurations sont possibles :

- 29 processeurs logiques, 2 processeurs mémoires et 1 processeur de contrôle;
- 27 processeurs logiques, 4 processeurs mémoires et 1 processeur de contrôle.

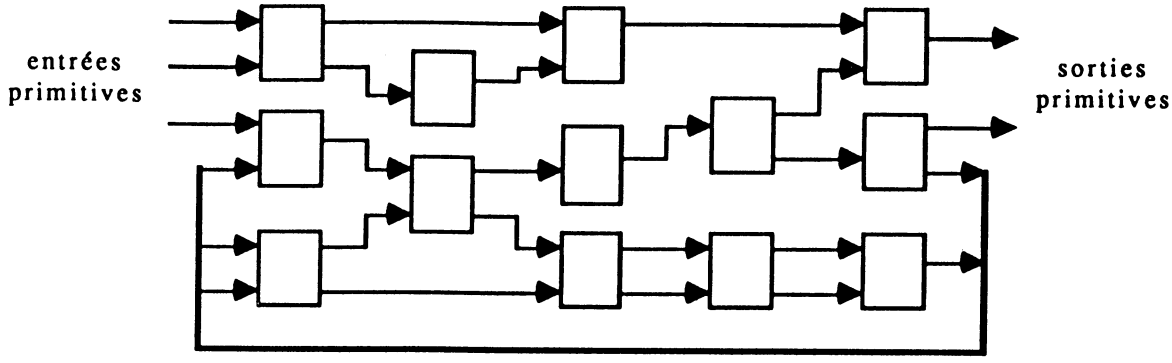


Architecture générale

Le processeur logique est un simulateur pipeline qui évalue les blocs logiques qui lui sont affectés.

Le processeur mémoire simule des ROMs ou des RAMs statiques ou dynamiques.

Le processeur de contrôle réalise le contrôle de l'ensemble du système et les communications entre HAL et la machine hôte.



Organisation des blocs

Le partitionnement du circuit respecte la notion de bloc. Chaque processeur est affecté d'un certain nombre de blocs qui s'organisent couche par couche dans le sens de la propagation des signaux.

Lorsqu'il y a des changements de valeurs sur les entrées primitives, le processeur évalue les blocs de la première couche dont les entrées sont changées. Ensuite sont évalués les blocs de la deuxième couche dont les entrées sont changées. Ce calcul se poursuit jusqu'à la dernière couche. A la fin de l'évaluation, s'il y a des changements de valeurs aux sorties, les informations sont envoyées en paquets aux processeurs concernés.

On trouve les caractéristiques suivantes :

- Les algorithmes sont câblés dans le processeur.
- Le calcul est réalisé en étapes qui s'organisent en pipeline.
- Les unités de base de l'évaluation sont les blocs. Le mécanisme de la gestion d'événements ne fonctionne que sur les blocs, ce qui permet de réduire le temps de l'évaluation.

Les performances sont les suivantes :

- la capacité maximale : HAL permet une simulation de 29696 blocs logiques et 2048 blocs mémoires, soit $3,0 \cdot 10^6$ portes logiques et 4M octets de mémoire.
- le temps de calcul : Pour un circuit de 500000 portes, 6ms de temps de cycle de simulation est nécessaire, ce qui est 10^3 fois plus rapide qu'un simulateur logiciel.

3.6.3. Machine de simulation d'ABRAMOVICI

Il s'agit d'une machine pipeline et parallèle spécialement développée pour la simulation logico-fonctionnelle [ABR 83].

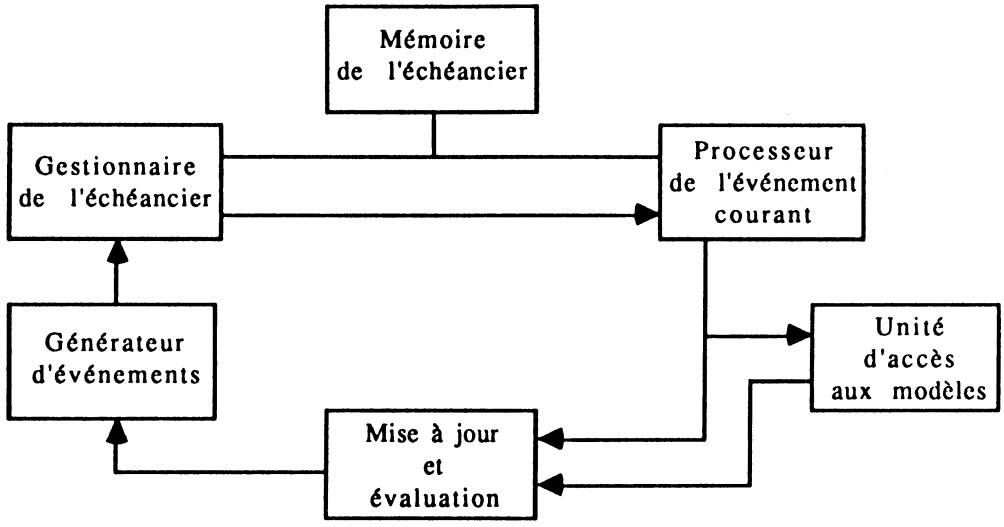
Cette machine utilise les techniques d'accélération suivantes :

• Les différentes tâches sont exécutées par les unités de traitement spécifiques, qui fonctionnent en pipeline.

• Les évaluations fonctionnelles qui consomment un temps de calcul important sont calculées par des processeurs fonctionnels qui travaillent en parallèles. Les éléments fonctionnels sont affectés à ces processeurs d'une manière statique ou dynamique.

• Les processeurs sont microprogrammables. Leur fonction est implémentée directement en microcode.

L'architecture de la machine est la suivante :



Architecture de la machine

L'ensemble de la machine se compose des unités suivantes :

• le gestionnaire de l'échéancier : Il reçoit de nouveaux événements en provenance du générateur d'événements, et ordonne les événements dans l'échéancier par ordre chronologique.

• le processeur de l'événement courant : Il cherche les événements qui se produisent à l'heure courante et les transmet à l'unité d'accès aux modèles.

• l'unité d'accès aux modèles : Elle reçoit chaque événement à traiter et détermine les éléments concernés par cet événement.

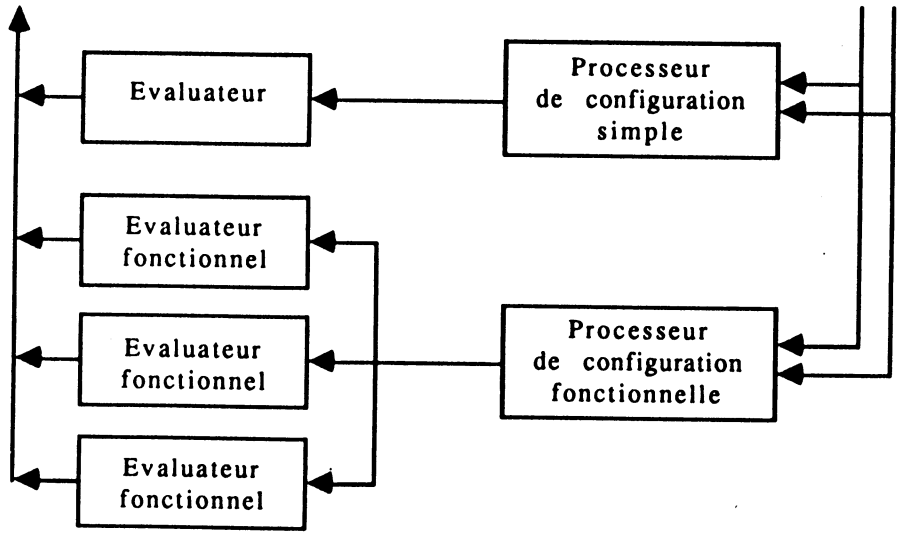
• la partie de mise à jour et d'évaluation : Elle met à jour les nouveaux états et évalue les éléments.

• le générateur d'événements : Il détermine les nouveaux événements.

La partie de mise à jour et d'évaluation est constituée de :

• un processeur de configuration simple et un évaluateur : Le processeur de configuration simple met à jour les états de la porte qui est ensuite évaluée par l'évaluateur.

• un processeur de configuration fonctionnelle et des évaluateurs fonctionnels : Le processeur de configuration fonctionnelle maintient et met à jour la configuration (les états des entrées, des sorties et des variables internes) de l'élément fonctionnel. Comme l'évaluation de l'élément fonctionnel est très coûteuse au niveau du temps de calcul, plusieurs évaluateurs fonctionnels sont utilisés en parallèle. Les éléments fonctionnels sont distribués statiquement ou dynamiquement à ces évaluateurs fonctionnels.



Architecture de la partie de mise à jour et d'évaluation

Les auteurs estiment que cette machine permet une simulation 10 à 60 fois plus rapide que les simulateurs purement logiciels et qu'avec la technologie disponible, la vitesse peut atteindre 10^6 évaluations par seconde.

Les auteurs ont proposé également quelques futures améliorations :

• l'utilisation de la mémoire cache pour la partie active du circuit simulé;

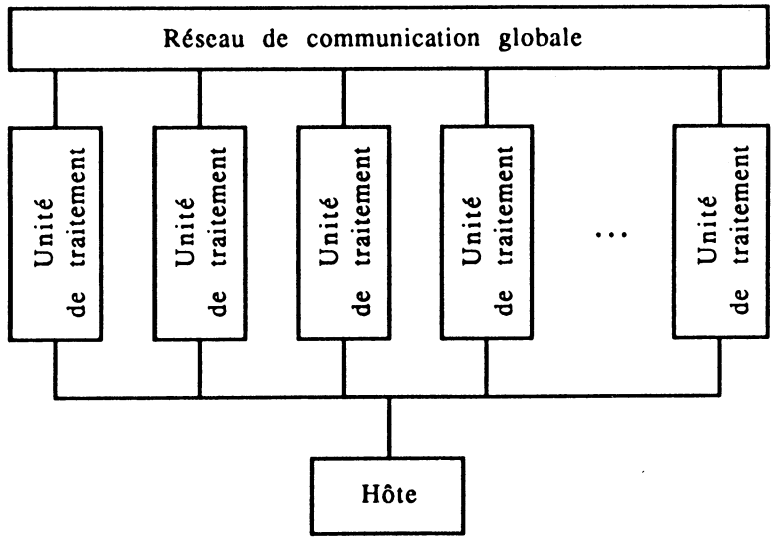
• l'utilisation du mécanisme de l'accès anticipé aux données;

• l'utilisation de l'échéancier matériel.

3.6.4. MARS

MARS (*Microprogrammable Accelerator for Rapid Simulations*) [AGR 87a] [AGR 87b] [AGR 90] est une machine développée par les laboratoires AT&T Bell. C'est un accélérateur matériel à l'architecture multiprocesseur microprogrammable et reconfigurable, dédié aux applications de CAO, notamment à la simulation. Le développement de MARS a pour but de construire, tout en gardant la flexibilité et la programmabilité, une architecture pipeline et parallèle qui offre une puissance de calcul comparable à celle des accélérateurs spécifiques. Selon les auteurs, MARS convient particulièrement à la simulation à échéancier.

L'architecture de MARS consiste en un certain nombre (jusqu'à 256) d'unités de traitement interconnectées par un réseau de communication :



Architecture de MARS

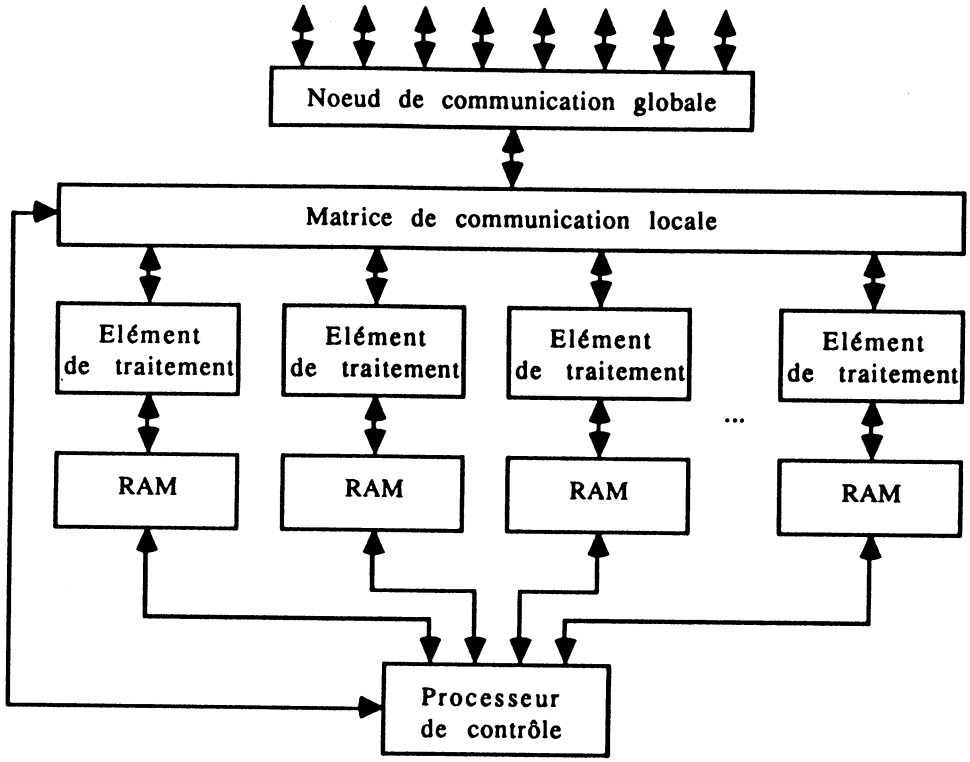
Les unités de traitement ont les caractéristiques suivantes :

• Les traitements dans chaque unité peuvent être organisés en pipeline de 14 étapes.

• Chaque étape de pipeline est réalisée par un élément de traitement

programmable.

• Les interconnexions entre les éléments de traitement sont également programmables.



Architecture de l'unité de traitement

Une unité de traitement se compose de :

- un noeud de communication globale;
- une matrice de communication locale;
- 14 éléments de traitement, chacun ayant une mémoire associée;
- un processeur de contrôle.

Chaque élément de traitement peut être programmé comme une étape spécifique du pipeline. Les éléments de traitement se communiquent par la matrice de communication locale à 16 x 16 points de communications.

MARS permet la réalisation de diverses applications de CAO, telles que la simulation logique, la simulation de pannes, la génération de test, le placement et le routage, etc. Par contre l'objectif initial du développement est orienté vers la simulation à échéancier.

Trois algorithmes de simulation logique (simple phase, double phase et

retard unitaire) sont développés. Les performances sont les suivantes :

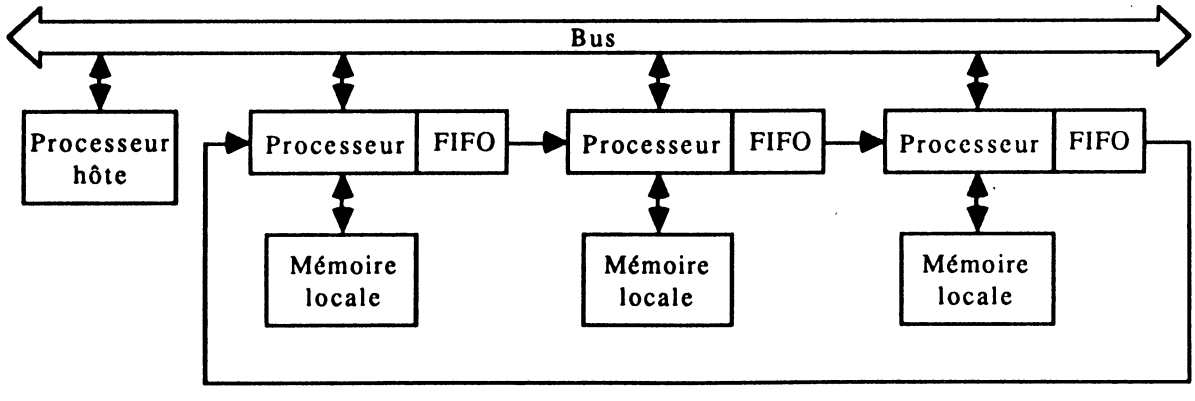
- Pour la simulation de double phase, la vitesse atteint 8,33 • 10^5 évaluations par seconde.
- Pour la simulation de simple phase, la vitesse atteint 1,25 • 10^6 évaluations par seconde.
- Pour la simulation de retard unitaire, la vitesse atteint 1,67 • 10^6 évaluations par seconde.

3.6.5. MegaLogician

Megalogician [CAR 87a] est un accélérateur pipeline développé par la société Daisy.

MegaLogician est un accélérateur à l'usage général sur lequel peuvent être implémentés différents types de simulations : la simulation logique et la simulation de pannes. La conception de l'accélérateur permet la modélisation de circuits aux niveaux suivants :

- le niveau porte logique;
- le niveau purement fonctionnel;
- le niveau comportemental;
- le niveau transistor;
- le niveau physique.



Architecture de MegaLogician

Pour pouvoir s'adapter aux différents algorithmes de simulation, l'accélérateur est conçu de la manière suivante :

• L'ensemble de l'accélérateur est basé sur trois processeurs organisés en anneau. Les connexions entre ces processeurs sont réalisées par des FIFO rapides unidirectionnelles.

• Chaque processeur est microcodé. Le microcode est placé dans la RAM. Il est chargé dans le processeur lors de la simulation.

L'algorithme de la simulation logique à échéancier est implémenté sur une configuration suivante :

• un processeur de gestion de l'échéancier qui effectue des opérations sur la liste d'événements;

• un processeur de gestion des états des noeuds qui mémorise et met à jour les états des noeuds en fonction des événements en provenance du processeur de gestion de l'échéancier;

• un processeur d'évaluation qui calcule les nouveaux états de sorties des éléments du circuit et transmet les événements créés au processeur de gestion de l'échéancier.

Les performances de l'accélérateur varient selon les algorithmes implémentés. En général, l'algorithme implémenté sur chacun des trois processeurs microcodés permettent une vitesse 30 fois plus élevée qu'un simulateur purement logiciel. L'organisation en pipeline des trois processeurs triple cette vitesse qui donne globalement une vitesse de 90 fois plus élevée.

3.6.6. Proteus-1

Proteus-1 [SMI 87] est un accélérateur général dédié aux applications de CAO conçu pour construire une machine puissante et peu coûteuse avec une haute reconfigurabilité, permettant de développer un ensemble d'outils de CAO.

La configuration maximale contient :

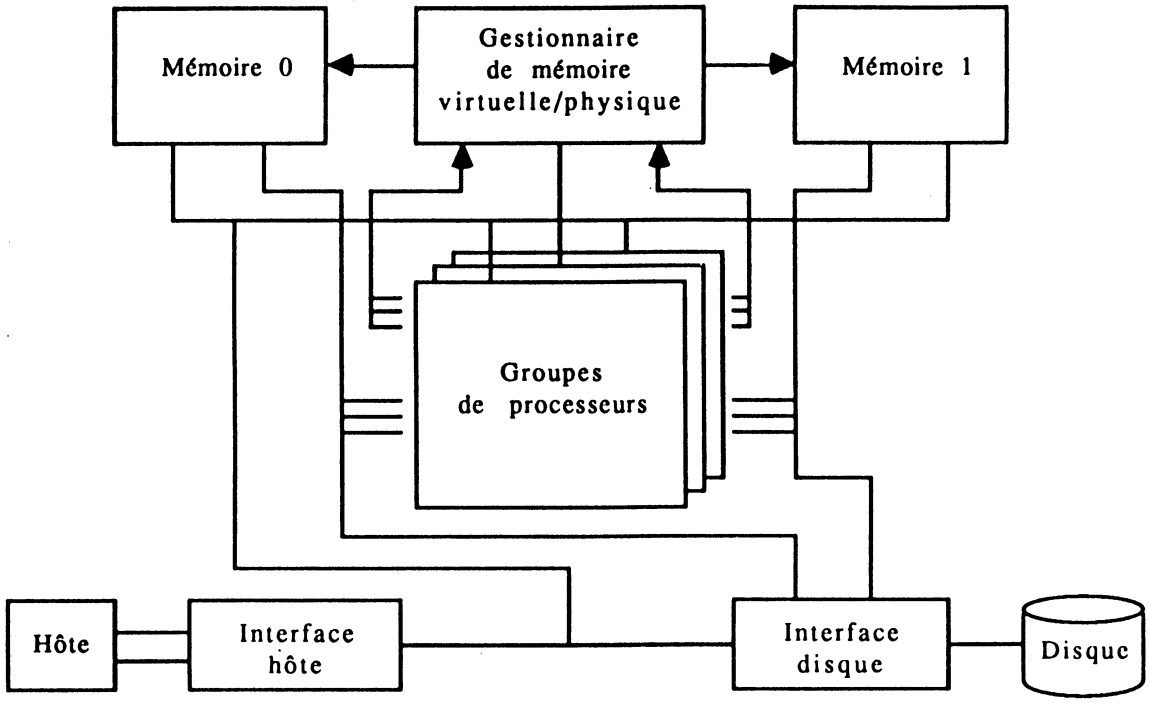
• trois groupes de processeurs, chaque groupe composé de quatre processeurs et de cinq buffers;

• deux mémoires indépendantes qui ont chacune une capacité de 16M octets;

• un gestionnaire de mémoire virtuelle/physique;

• une interface disque;

• une interface hôte.



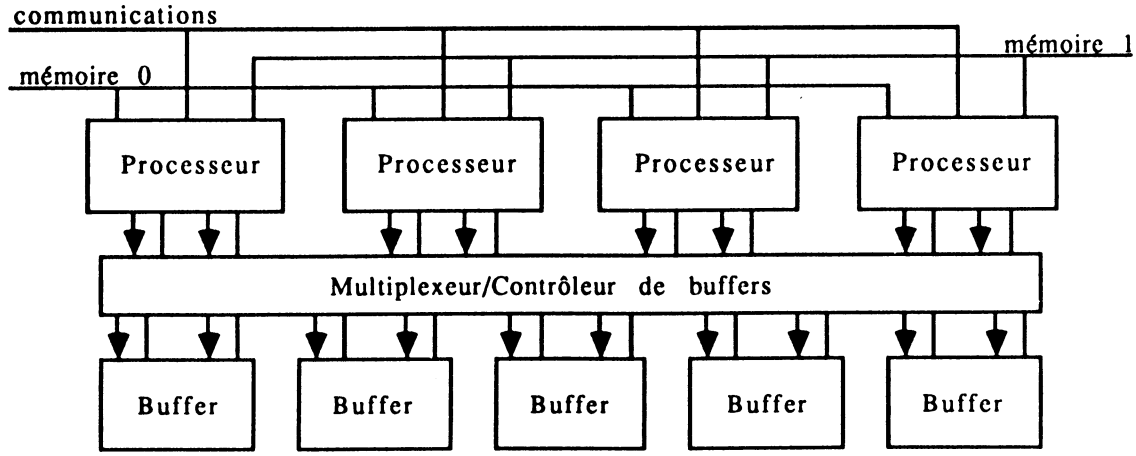
Architecture de Proteus-1

Un groupe de processeurs est constitué de :

• 4 processeurs microprogrammables qui peuvent être configurés en pipeline de 4 étapes ou en mode SIMD;

• 5 buffers rapides d'une taille 4K x 32 bits;

• un multiplexeur/contrôleur qui permet d'allouer, désallouer et d'accéder aux buffers.



Groupe de processeurs

Sur Proteus-1 est implémenté un simulateur à échéancier. Les pré-traitements et les post-traitements y sont également implémentés. Chaque groupe de processeurs s'organise de la façon suivante :

- Les processeurs sont proprement microcodés pour réaliser les fonctions logiques.
- Ils sont configurés en pipeline de 3 étapes.
- Pour la deuxième étape, deux processeurs fonctionnent en mode SIMD pour éliminer le goulot d'étranglement de cette étape.
- Les buffers rapides assurent le transfert de données entre les étapes du pipeline.
- L'utilisation de deux mémoires permet la séparation entre les opérations de mise à jour des états et d'évaluation, et celles de gestion de l'échéancier.

Les performances de Proteus-1 sont très dépendantes de la qualité et de l'efficacité du microcode. Des logiciels sont développés pour générer de manières différentes le microcode.

Le langage C est utilisé comme un des langage de programmation de Proteus-1. Un langage de type assembleur est disponible qui permet de produire le microcode séquentiel. Un compresseur de microcode a été développé qui transforme le microcode séquentiel en microcode parallèle.

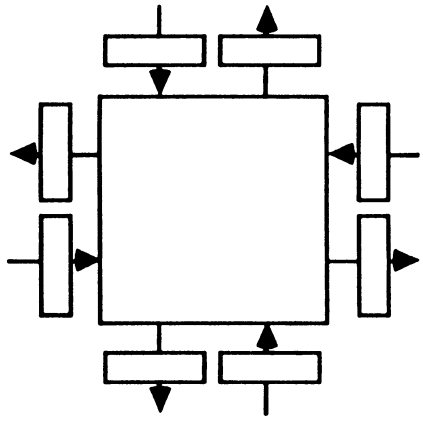
Les performances d'un groupe de processeurs ont été mesurées et atteignent 240000 évaluations par seconde, lorsque le circuit contient moins de 500000 éléments qui ne nécessitent pas l'utilisation de la mémoire virtuelle.

3.6.7. Réseau cellulaire du LGI

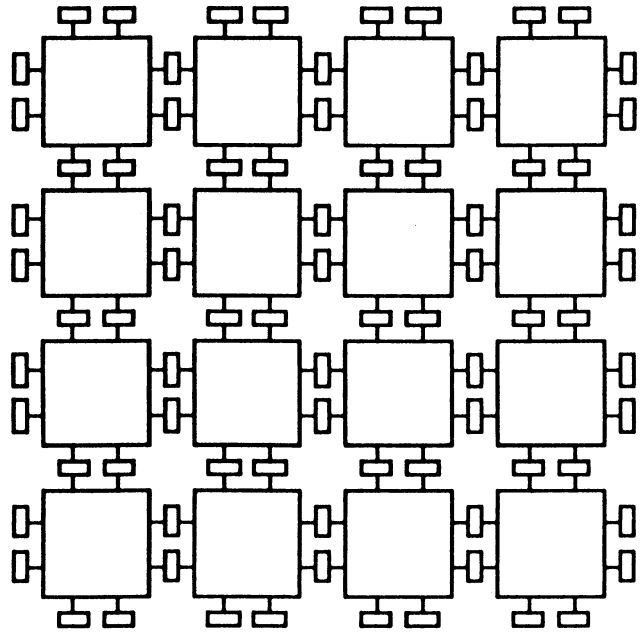
Le réseau cellulaire [ANS 87] [OBJ 88] est un réseau des cellules asynchrones organisées en une matrice plane. C'est une architecture qui permet d'accélérer tout algorithme massivement parallèle.

Le réseau est constitué de NxM cellules.

Chaque cellule réalise une fonction et échange des données avec les cellules voisines immédiates qui l'entourent.

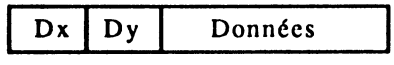


Cellule



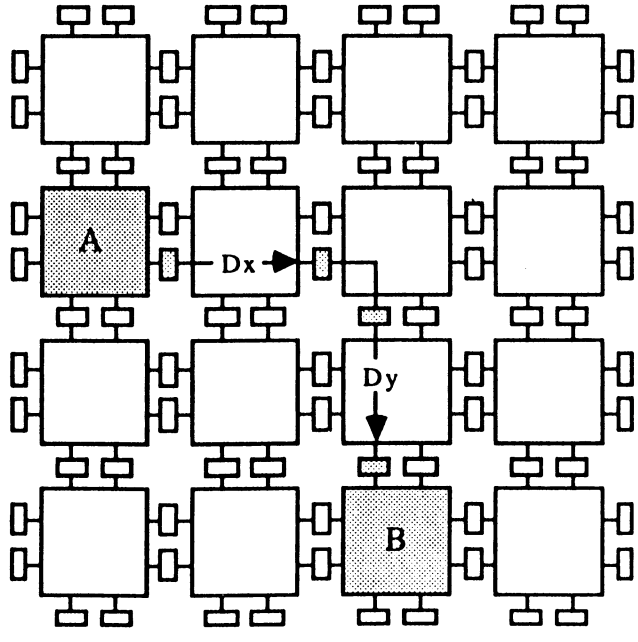
Réseau cellulaire

Une cellule peut communiquer avec n'importe quelle autre cellule du réseau. Un mécanisme d'acheminement de message est implémenté dans les cellules qui permet les échanges de données entre deux cellules qui n'ont pas de connexions matérielles directes. Le message de communication est composé de deux champs : le champ d'adresse qui contient la position relative de la destination par rapport à la position courante du message, et le champ d'information qui contient les données à transmettre.



$Dx = X(B) - X(A)$

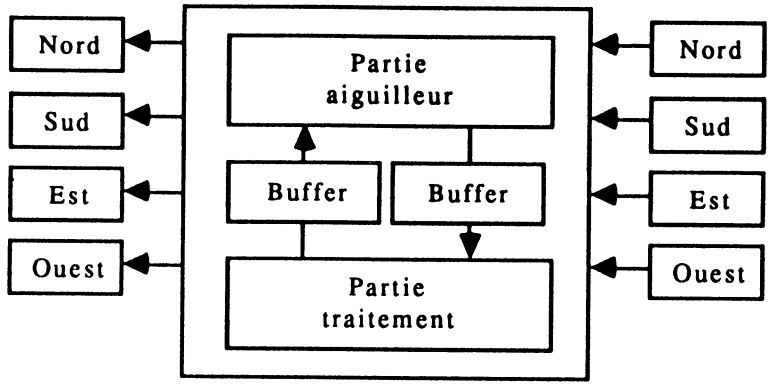
$Dy = Y(B) - Y(A)$



La cellule est constituée de deux parties distinctes fonctionnant en parallèle :

• la partie aiguilleur : Elle est chargée de l'acheminement des messages.

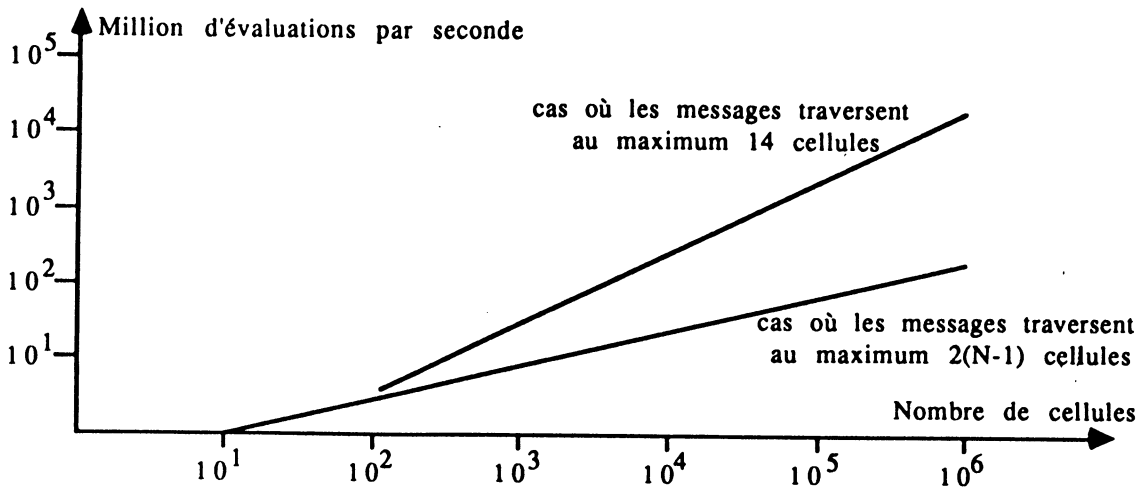
• la partie traitement : Elle effectue un algorithme spécifique qui réalise une application.



Le simulateur implémenté sur le réseau permet la simulation logique à retard unitaire, à trois états (0, 1 et X).

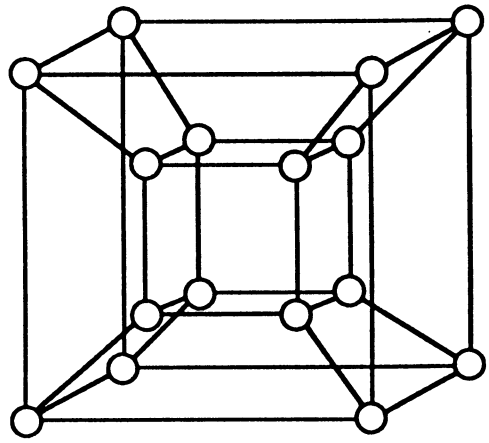
Plusieurs algorithmes de simulation sont réalisés avec différents modes de synchronisation.

Les performances estimées sont les suivantes :



3.6.8. Simulateur implémenté sur iPSC

Le simulateur MOSSIM a été implémenté sur une machine multiprocesseur iPSC d'Intel à architecture hypercube [THA 87].



Architecture de la machine iPSC

La machine iPSC se compose de deux unités :

- le gestionnaire de l'hypercube;
- l'hypercube.

Le gestionnaire de l'hypercube supporte l'environnement de programmation et se comporte comme une machine hôte autour de l'hypercube.

L'hypercube est une machine MIMD organisée en une topologie de 16 processeurs constituant une architecture 4 dimensions de traitement parallèle. Chaque processeur est connecté à ses quatre voisins par des canaux de communication bidirectionnels.

Le circuit à simuler est décomposé en partitions qui sont ensuite réparties sur les processeurs de l'hypercube. Les processeurs fonctionnent en parallèle de manière synchrone ou asynchrone.

L'algorithme de partitionnement du circuit met l'accent sur les optimisations suivantes :

- la localisation des activités du circuit : Il faut que les activités soient

localisées à l'intérieur des processeurs afin de réduire au minimum les communications inter-processeurs. Le nombre de liens bidirectionnels de transistors qui traversent les partitions du circuit doit être réduit.

• la minimisation des distances logiques de communications entre les processeurs : Une communication peut traverser plusieurs processeurs intermédiaires pour atteindre la destination. Une analyse statique est utilisée pour minimiser les distances logiques.

• la réplication des noeuds du circuit : Les équipotentielles de source et les entrées primitives du circuit sont répliquées et placées sur différents processeurs afin de supprimer les communications inter-processeurs non nécessaires.

• l'équilibrage des charges de travail des processeurs : La répartition des charges affectées aux processeurs doit assurer que les processeurs ne soient pas bloqués par ceux qui ont trop d'activités.

La synchronisation est réalisée à l'aide du gestionnaire de l'hypercube qui joue le rôle de coordinateur.

Chaque pas de simulation est composé de quatre phases qui peuvent être organisées en pipeline.

Il est relativement difficile de comparer les performances du simulateur car trop de paramètres sont impliqués. En gros, la vitesse de simulation est 2 ou 3 fois plus élevée que celle de VAX 11/780.

3.6.9. Simulateurs logiques de l'université de Kyoto

L'université de Kyoto a développé trois simulateurs logiques à base de processeurs vectoriels [ISH 87] :

- un simulateur au retard nul pour les circuits combinatoires;
- un simulateur au retard nul pour les circuits séquentiels synchrones;
- un simulateur au retard variable.

Le simulateur des circuits combinatoires au retard nul utilise la technique de la simulation par code compilé.

Les portes du circuit à simuler sont ordonnées dans l'ordre de propagation des signaux. Les calculs commencent par les entrées primitives du circuit et se terminent aux sorties primitives.

Dans les calculs, une séquence d'états logiques, qui représente l'historique des changements d'état d'une équipotentielle, est codée en un vecteur. L'évaluation d'une porte est effectuée par une opération logique vectorielle correspondante à la fonction logique de la porte. La vectorisation permet l'évaluation simultanée d'une séquence de changements d'état de l'équipotentielle, même si ceux-ci ne se produisent pas au même moment.

Le simulateur est implémenté sur une machine à processeur vectoriel FACOM VP-200, avec une vitesse de $7,7 \cdot 10^9$ évaluations par seconde.

Le simulateur des circuits séquentiels synchrones utilise la technique de la simulation par code compilé. La vectorisation des calculs se fait par le regroupement des portes. Les portes regroupées sont évaluées par une opération vectorielle. Cette technique permet d'améliorer considérablement les performances de la simulation.

Le simulateur est réalisé sur les machines à processeurs vectoriels FACOM VP-200 et S-810 qui offrent une vitesse de simulation de $1,4 \cdot 10^9$ évaluations par seconde.

Le troisième simulateur est un simulateur à échéancier qui utilise un algorithme de double phase.

A chaque pas de simulation, il récupère tous les événements de l'heure courante sous forme de vecteur. Les portes concernées par les événements sont ensuite évaluées, toujours sous forme de vecteur. Les nouveaux événements sont insérés dans l'échéancier qui a également une structure vectorielle.

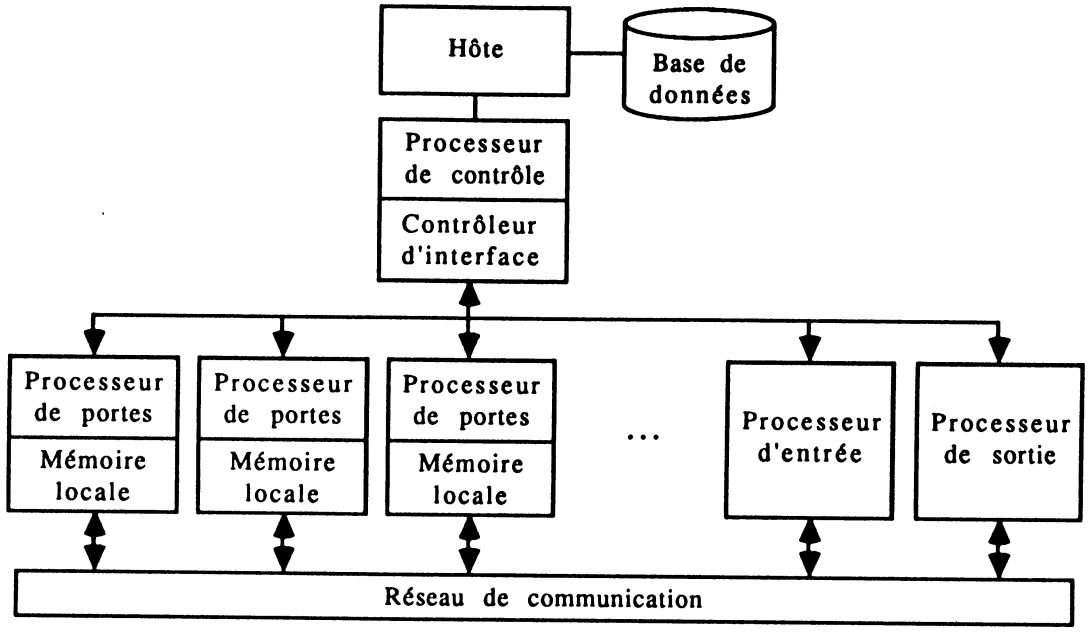
Le simulateur est implémenté sur HITAC S-810/20 qui permet d'avoir une vitesse de simulation de l'ordre de $2,3 \cdot 10^5$ événements par seconde ou $4,4 \cdot 10^5$ évaluations par seconde.

3.6.10. SP

SP [SAI 88] est une machine spécifique pour la simulation à échéancier, développée par Fujitsu.

SP se compose de :

- 64 processeurs de portes;
- un processeur d'entrée;
- un processeur de sortie;
- un processeur de contrôle et un contrôleur d'interface;
- un réseau de communication.



Architecture de SP

Les processeurs de portes simulent les primitives de portes et de mémoire. Chaque processeur peut traiter 64K portes et 512K octets de mémoire.

L'algorithme de l'échéancier est implémenté qui permet d'utiliser 16 valeurs logiques, et le retard unitaire. La modélisation fonctionnelle est rendue possible par l'utilisation d'un traducteur qui transforme automatiquement le circuit fonctionnel en une représentation de portes.

Les processeurs de portes sont organisés en architecture parallèle et pipeline.

Le processeur d'entrée gère les signaux d'entrée qui stimulent le circuit. Le processeur de sortie assure l'édition et l'analyse des résultats de simulation.

Le système logiciel développé sur SP contient les modèles suivants :

• le convertisseur des modèles de circuits logiques qui génère, à partir des modèles de circuits hiérarchiques mémorisés dans la base de données, des circuits sous forme de portes interconnectées;

• le traducteur qui transforme les circuits fonctionnels en circuits structurels;

• le constructeur de modèles qui génère d'une part le fichier de modèles qui décrit l'ensemble des circuits à simuler et distribue d'autre part les modèles sur les processeurs de portes;

• le compilateur du langage de contrôle de simulation;

• le compilateur des stimuli;

• l'exécuteur de simulation qui charge les modèles sur les processeurs de portes;

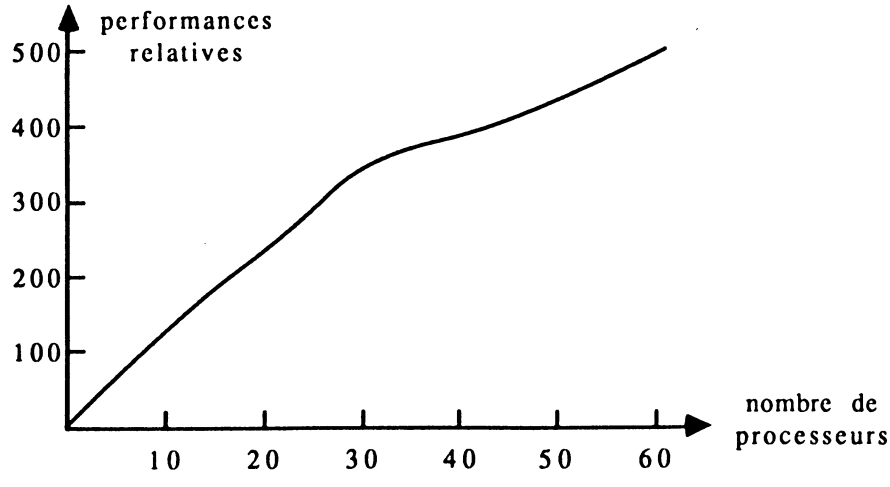
• le programme de rétro-annotation qui permet la modification des circuits à simuler sans recompilation;

• l'éditeur de résultats de simulation;

• le simulateur purement logiciel qui facilite la simulation de petits circuits.

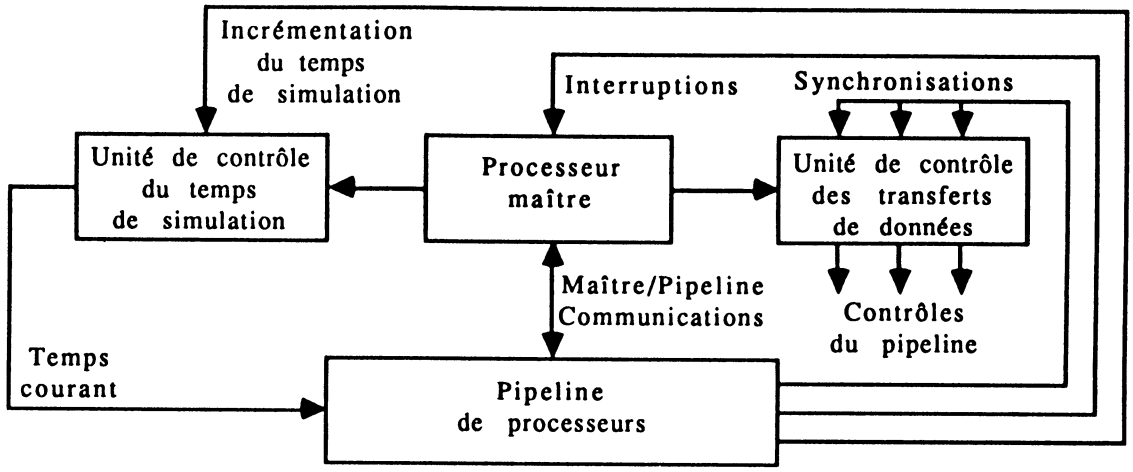
La capacité de SP est de $4 \cdot 10^6$ primitives logiques et 32M octets de mémoire. La vitesse de calcul est de l'ordre de $8 \cdot 10^8$ évaluations par seconde.

Le rapport entre les performances et le nombre de processeurs a été mesuré sur un circuit contenant 468K primitives logiques et 211K octets de mémoire. Les résultats montrent que les performances augmentent d'une manière linéaire lorsque le nombre de processeurs est inférieur à 20.



3.6.11. ULTIMATE

ULTIMATE (*UMIST Logic, TIMing And Test Evaluator*) [GLA 84] est une machine développée par l'université de Manchester. C'est un simulateur matériel à architecture pipeline, dédié à la simulation logique.



Architecture de l'accélérateur ULTIMATE

ULTIMATE utilise le mécanisme de l'échéancier. Il distingue les primitives et les non primitives. Les primitives sont évaluées par l'algorithme de simple phase, tandis que les non primitives sont évaluées par l'algorithme de double phase.

L'évaluation des éléments et la gestion des événements sont décomposées en une séquence d'opérations élémentaires. Ces opérations sont réalisées respectivement par des unités de traitement spécifiques.

Le pipeline est constitué de 11 étapes avec des mémoires associées. Ces 11 étapes sont :

- la détermination de l'adresse de l'événement à évaluer dans l'échéancier;
- l'accès aux données de l'événement à évaluer;
- la détection de la présence de l'événement dans la liste de visualisation, la détection de l'oscillation et l'accès aux éléments concernés par l'événement;
- la détermination du type de l'élément;
- l'accès aux entrées de l'élément;

• l'accès aux états des entrées, l'accès au retard inertiel, ainsi que la détermination de la fonction logique;

• la détermination des conditions d'aléa sur les entrées, l'évaluation des entrées et la mise à jour du nouvel état du noeud;

• le calcul du retard;

• la détection de la présence de l'événement déjà dans l'échéancier et l'évaluation de la sortie;

• la suppression des événements non nécessaires et la réservation de l'espace dans l'échéancier pour les nouveaux événements;

• la mise à jour de l'adresse de l'événement prochain dans la mémoire correspondante et l'insertion des nouveaux événements.

Les performances sont respectivement de l'ordre de 3,3 • 10^6, 2,2 • 10^6 et 1,6 • 10^6 évaluations par seconde, pour des circuits à entrance moyenne de 2, 3 et 4.

3.6.12. VELVET

Le simulateur logique VELVET de Hitachi [KAZ 88] est implémenté sur une machine à processeur vectoriel S-810.

Par rapport aux autres simulateurs à base de processeurs vectoriels, le simulateur présente deux originalités :

• Au lieu d'utiliser la technique de la simulation par code compilé comme le font la plupart des machines à processeurs vectoriels, l'algorithme est dirigé par les événements.

• Le simulateur autorise deux niveaux de modélisation : le niveau logique et le niveau transfert de registre.

L'architecture vectorielle permet d'évaluer simultanément plusieurs événements à traiter, même si les opérateurs correspondants à ces événements sont différents. De plus, d'autres mesures ont été prises pour améliorer les performances du simulateur :

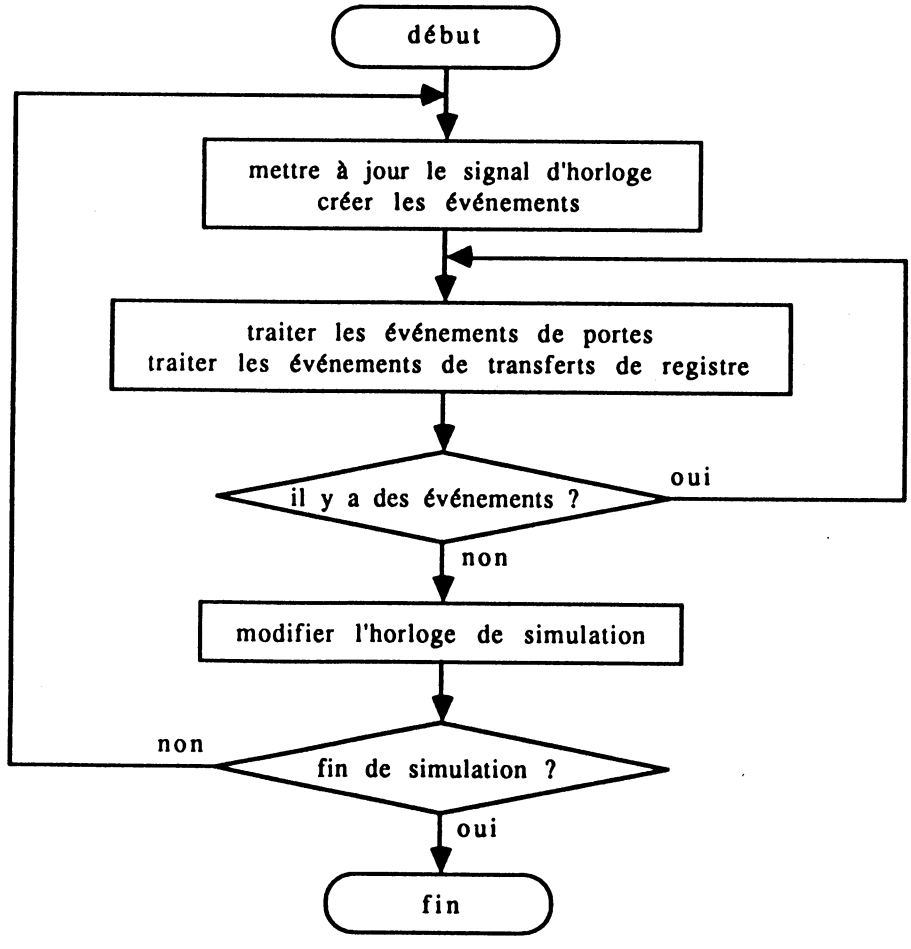
• l'ajout des instructions spéciales dans la machine : Les opérations qui évaluent les fonctions des portes logiques sont câblées. La gestion des structures de données propres à la simulation et les mécanismes de contrôle fréquemment utilisés dans l'algorithme de l'échéancier sont réalisés par des instructions spéciales. Ces instructions spéciales permettent un temps de

calcul 10 fois plus court.

• l'utilisation des mémoires externes : L'accès au disque est un goulot d'étranglement de la simulation. L'utilisation des mémoires externes de haute vitesse réduit considérablement le temps de calcul (également 10 fois plus court).

• la suppression des événements : Un algorithme a été développé pour supprimer les événements d'horloge non utiles à la simulation.

Le simulateur utilise un algorithme à retard nul avec la synchronisation de l'horloge : Il répète le traitement des portes et le traitement des transferts de registre alternativement pour chaque cycle d'horloge, jusqu'à ce qu'il n'y ait plus de changement d'état sur le circuit.



La vitesse de VELVET est 100 fois plus élevée que les simulateurs classiques. En plus, cette vitesse est presque indépendante de la taille du circuit à simuler, ce qui rend VELVET plus avantageux pour la simulation des gros circuits.

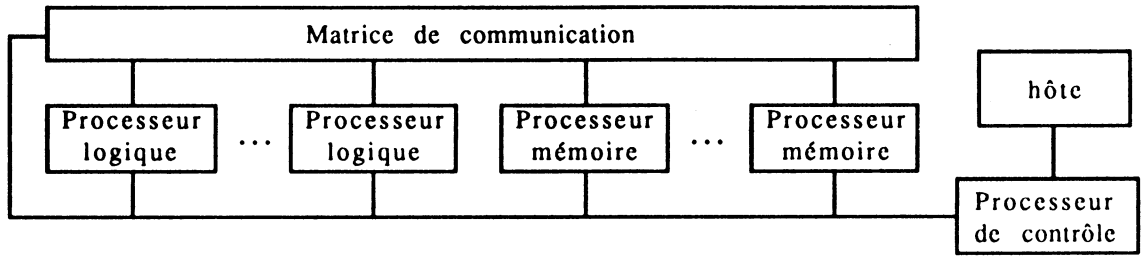
3.6.13. YSE et EVE

YSE (*Yorktown Simulation Engine*) [PFI 86] et EVE (*Engineering Verification Engine*) [BEE 88] sont les machines multiprocesseurs hautement parallèles, développées par IBM, pour la simulation logique.

La machine YSE était un prototype développé dans le cadre d'un projet de recherche. La machine EVE est une version opérationnelle, qui a repris l'architecture de YSE, pour répondre aux besoins internes des projets de conception d'IBM.

YSE et EVE utilisent la technique de la simulation par code compilé. Le compilateur partitionne le circuit et le répartit sur les processeurs qui fonctionnent en parallèle. La configuration maximale contient 256 processeurs.

YSE et EVE ont une architecture identique :



Architecture de YSE et de EVE

Les principaux composants de YSE et EVE sont :

• les processeurs logiques : Chacun peut simuler un circuit de 8K portes à une vitesse de 80ns par porte (100ns pour EVE).

• les processeurs mémoires : Ils simulent les RAMs et les ROMs. Sur YSE, ces processeurs ne sont pas opérationnels. Sur EVE, chaque processeur mémoire occupe 9 ports de communication et peut simuler 12,4M octets de mémoire.

• la matrice de communication : Elle offre une capacité de 256 x 256 ports de communication.

• le processeur de contrôle : Il réalise les échanges de données.

Le compilateur est la partie critique du logiciel. Il génère les instructions à exécuter à partir du circuit à simuler. De plus, il réalise les fonctions suivantes :

- le partitionnement du circuit sur les processeurs;
- le pré-traitement pour les communications inter-processeurs;
- la mise en ordre de l'exécution des instructions;
- la création des tables de symboles.

Le compilateur n'accepte que les circuits ayant moins de 32K portes. L'utilisation de l'éditeur de liens rend possibles les compilations séparées.

Un convertisseur a été développé qui transforme une description de transfert de registre en un assemblage de portes. Le convertisseur génère un circuit 4,5 à 7 fois moins de portes qu'une conception manuelle.

Théoriquement, la vitesse de YSE en configuration maximale est de l'ordre de $3,2 \cdot 10^9$ évaluations par seconde. En réalité, la vitesse moyenne mesurée sur des circuits de test n'atteint que 25% de cette vitesse théorique. D'autre part, il faut noter que 80% - 95% des évaluations sont inutiles car à chaque instant la majorité des portes ne change pas d'état pendant la simulation.

Quant à EVE, la configuration maximale est constituée de 220 processeurs logiques et 4 processeurs mémoires, ce qui donne une capacité de $1,8 \cdot 10^6$ portes et 50M octets de mémoire. Les performances annoncées sont de l'ordre de $2,2 \cdot 10^9$ évaluations par seconde.

Conclusion

Des améliorations algorithmiques et des accélérations matérielles sont les techniques principales pour augmenter la vitesse de simulation. Le parallélisme, le pipeline et le jeu d'instructions spécifiques sont des techniques très utilisées pour construire les accélérateurs de simulation. Certains accélérateurs combinent plusieurs techniques.

Il existe des accélérateurs spécifiques, qui peuvent être trop coûteux et trop orientés. Leur développement est très retardataire par rapport aux machines générales. D'autres solutions ont été étudiées : les accélérateurs généraux et les machines générales.

Un autre possibilité consiste à construire des accélérateurs à base de processeurs généraux. ce qui permet d'avoir un accélérateur puissant et économique. Le développement de LL3T porte sur cette philosophie.

Un certain nombre d'accélérateurs actuels ont été présentés. Les analyses sur ces accélérateurs mettent en évidence que :

- Les accélérateurs spécifiques utilisent souvent une architecture spécifique pour s'adapter aux particularités des structures de données et des algorithmes de simulation. Certains accélérateurs sont microprogrammables.
- Le pipeline est une technique fondamentale, relativement simple à introduire dans les machines de simulation. Le nombre d'étapes de pipeline varie de 2 à une dizaine.
- Le parallélisme massif permet d'améliorer considérablement les performances des simulateurs. Les accélérateurs puissants font intervenir systématiquement le parallélisme.
- Beaucoup d'accélérateurs utilisent la combinaison de plusieurs techniques. Leur architecture repose sur un ensemble de processeurs fonctionnant en parallèle. Chaque processeur est basé sur un mécanisme de pipeline. Chaque étape du pipeline est réalisée par une unité spécifique ou microprogrammée.
- Les simulateurs à base de machines générales sont souvent implémentés sur les machines multiprocesseurs ou à processeurs vectoriels pour bénéficier de la puissance de calcul de ces machines.

Chapitre 4

Architecture de LL3T : Organisation générale

Plan du chapitre 4

Introduction

L'objectif de ce chapitre est de présenter, d'un point de vue général, l'architecture de LL3T et l'ensemble des logiciels implémentés.

Les transputers constituent les éléments de base du matériel de LL3T. Ils s'organisent par groupes de 3 et forment un ensemble d'unités de simulation disposées le long d'un anneau. Le choix de cette architecture est basé sur les considérations suivantes :

- La répartition des tâches doit être équilibrée : L'évaluation des événements est plus coûteuse que la gestion de l'échéancier. Une unité de simulation est donc composée de 3 transputers.
- La structure topologique du matériel doit respecter les contraintes architecturales des microprocesseurs : Le transputer n'a que quatre liens de communication. Une topologie trop complexe n'est pas envisageable.
- L'ensemble du matériel doit être facile à mettre en oeuvre : En effet, le réseau en anneau est une architecture simple et efficace permettant d'exploiter le parallélisme. Par ailleurs, le matériel peut être facilement reconfiguré en fonction des besoins par l'ajout et le retrait directs d'unités de simulation.

L'ensemble des logiciels constitue un environnement de simulation qui se comporte comme un système intégré comprenant des outils nécessaires, qui permettent de gérer des sources et des données intermédiaires et d'effectuer des opérations sur ces données, avant, pendant et après la simulation. Cet environnement consiste en :

- un noyau de simulation qui permet différents types de simulation et de post-traitements;
- les langages de description, supportés par des compilateurs dédiés;
- une base de données dans laquelle sont gérés des modules compilés qui représentent les circuits et les stimuli-réponses;
- des bibliothèques externes créées par l'environnement ou par l'utilisateur;
- des commandes de contrôle et des interpréteurs associés.

Le simulateur, le module le plus important de l'environnement de simulation, sera présenté en premier. Son interface est un langage sémantiquement très proche de la structure interne du simulateur lui-même,

permettant de décrire la topologie du circuit, la répartition du circuit sur les unités de simulation et les tâches de simulation à effectuer.

Ensuite, les compilateurs seront décrits. Leur rôle est de traduire les descriptions Hilo-3 en images internes, c'est-à-dire les modules compilés et gérés dans la base de données de LL3T. La structure de données et les propriétés sémantiques des images internes dépendent des caractéristiques des langages Hilo-3.

En ce qui concerne l'édition de liens, non seulement les éditeurs de liens mais également les configurateurs seront présentés. L'existence des configurateurs est due au fait qu'il n'existe pas de mécanisme statique dans les langages Hilo-3 qui permettent d'organiser, pendant ou après la compilation, les images internes en une structure hiérarchique par leurs relations, notamment la relation d'englobement de circuits. Les configurateurs établissent les relations suivantes, qui seront prises en compte lors de l'édition de liens :

- la dépendance des bibliothèques;
- la hiérarchie des circuits impliqués dans la simulation.

L'écart sémantique entre les images internes de circuits, créées par la compilation et liées par la configuration et l'édition de liens, et les images exécutables de circuits destinées au simulateur est comblé par la transformation des images effectuée par le générateur de code. Le générateur extrait, à la suite de l'édition de liens, les informations nécessaires et génère un code exécutable destiné au simulateur.

4.1. Architecture générale

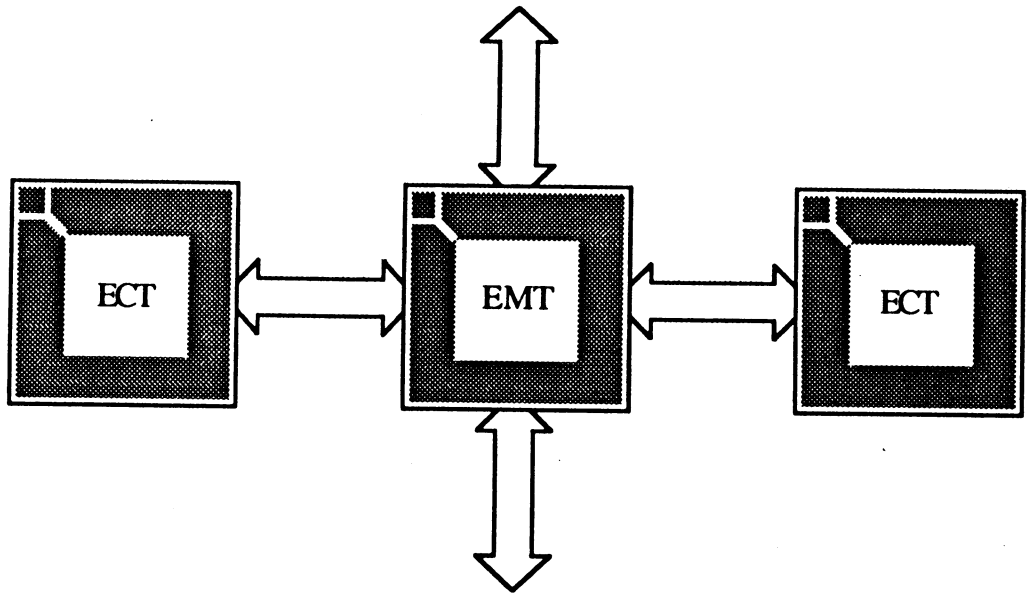
4.1.1. Support matériel

Le matériel associé à LL3T est un réseau de transputers organisés en anneau, sur lequel chaque noeud est une unité de simulation, composée de trois transputers.

Chaque transputer réalise une des tâches suivantes :

- l'évaluation des événements : le transputer ECT;
- la gestion des événements : le transputer EMT;
- la visualisation des résultats : le transputer DMT.

Une unité de simulation est constituée de deux ECTs et un EMT :

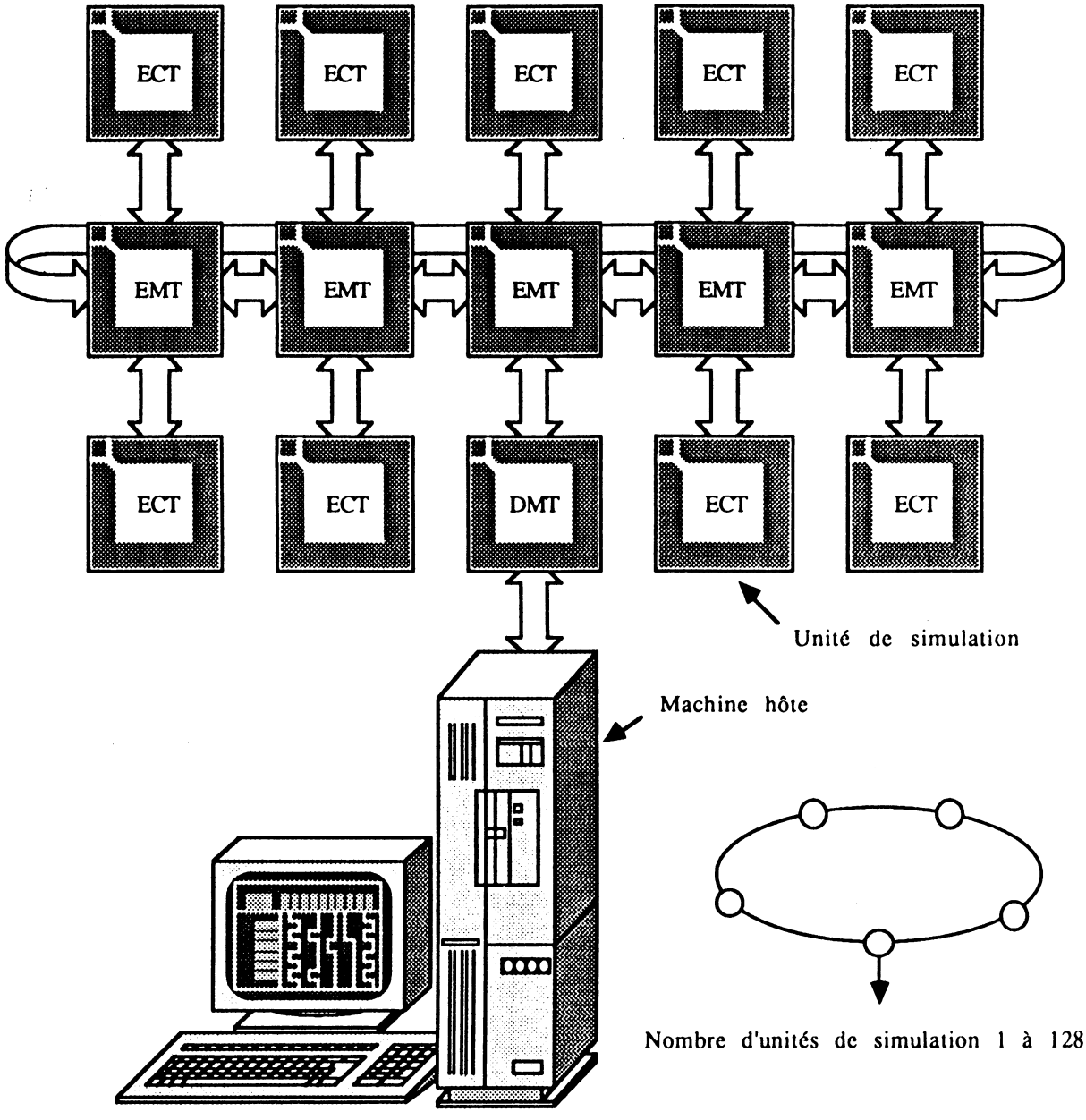


Unité de simulation

Le transputer ECT se charge d'évaluer des événements qui lui sont distribués par l'EMT et de créer de nouveaux événements en fonction du résultat de l'évaluation. Le transputer EMT gère l'échéancier local de son unité de simulation et établit les communications entre son unité et les voisines directes ou indirectes. Le transputer DMT permet les échanges de données et les interactions entre le réseau et la machine hôte.

Le réseau contient 1 à 128 unités de simulation, parmi lesquelles il existe une unité constituée d'un ECT, un EMT et un DMT connecté à la

machine hôte :



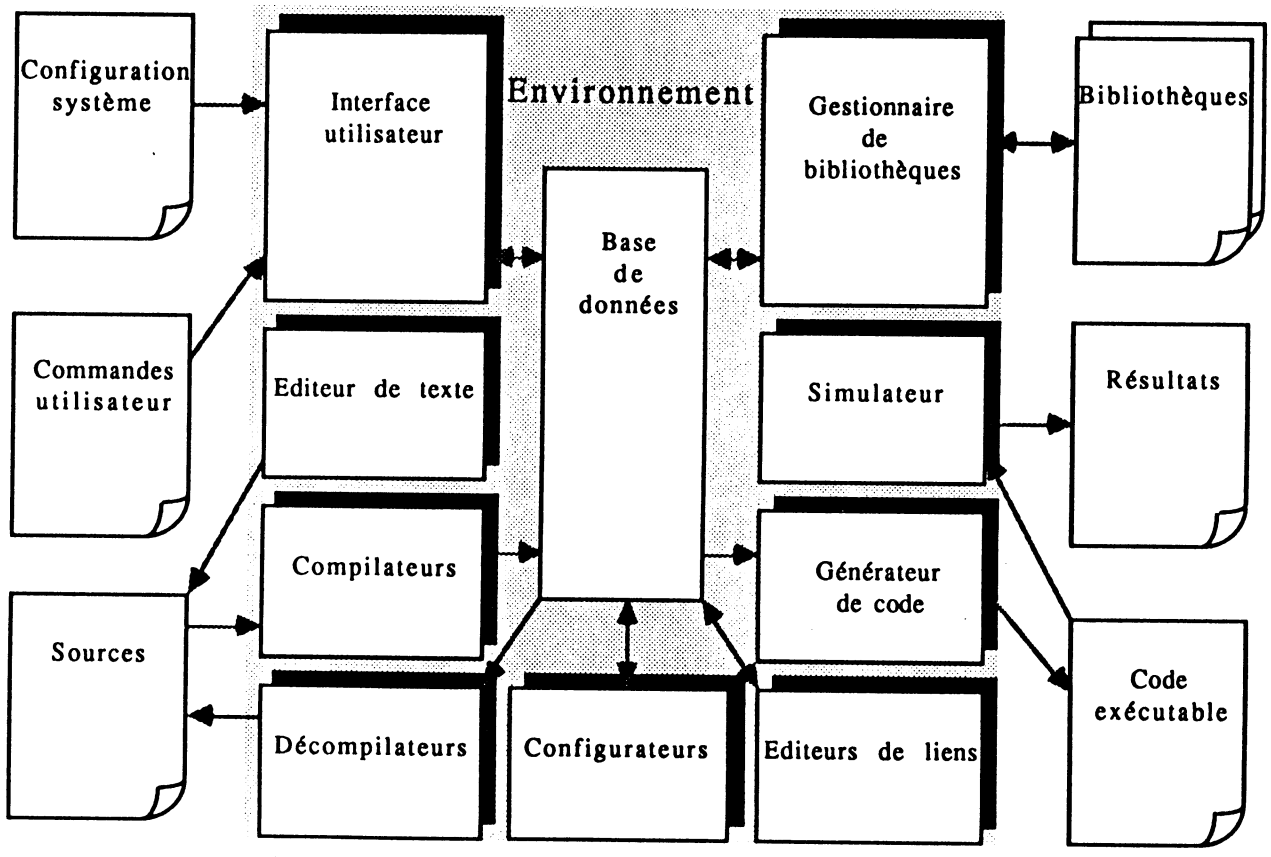
Architecture de LL3T

4.1.2. Environnement logiciel

Sur ce matériel est implémenté un environnement de simulation, qui intègre non seulement le simulateur, mais aussi les outils de pré-traitements et de post-traitements de la simulation :

- l'éditeur de texte;
- le simulateur;

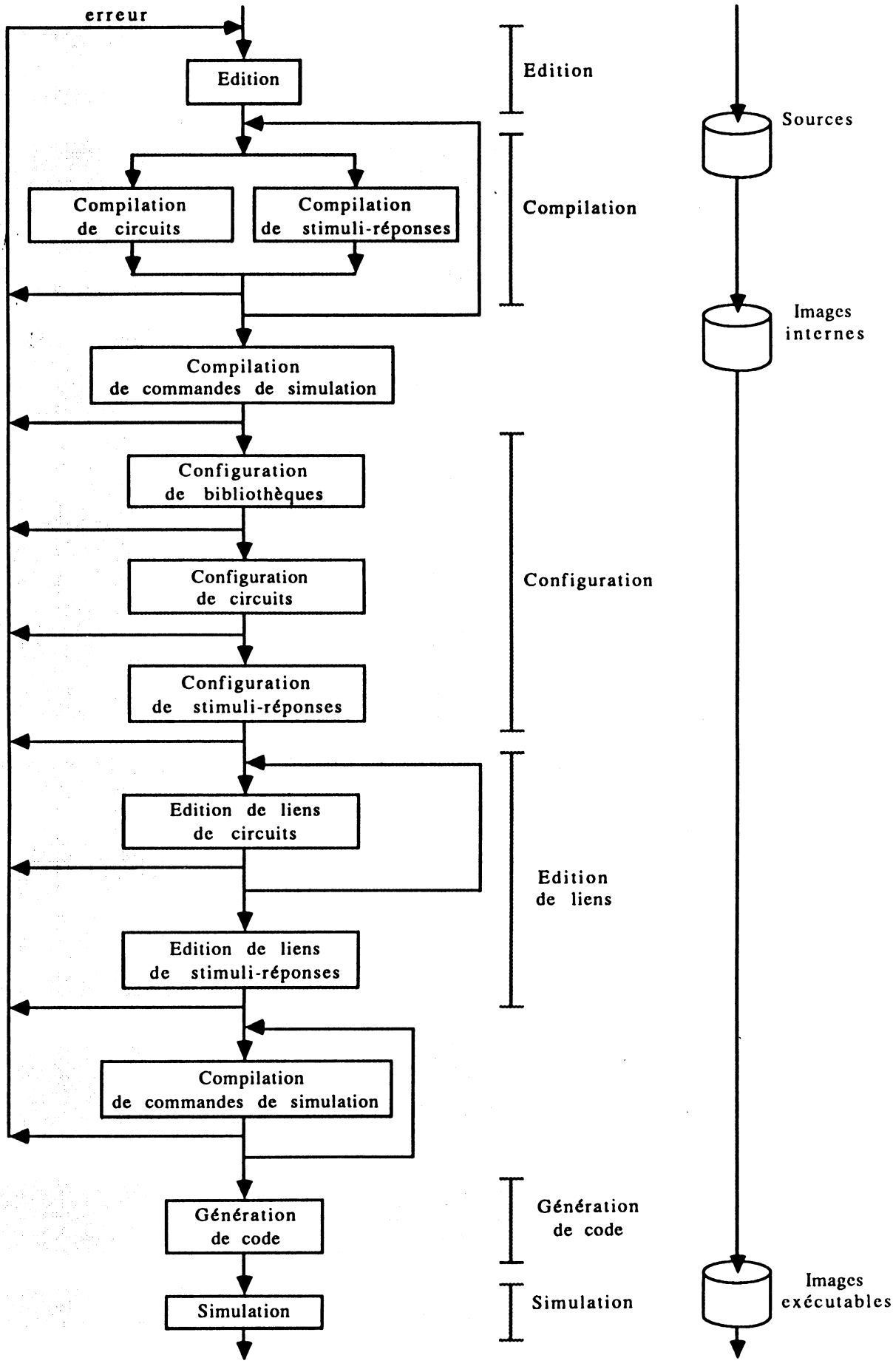
- les compilateurs;
- les configurateurs;
- les éditeurs de liens;
- le générateur de code;
- les décompilateurs;
- le gestionnaire de bibliothèques;
- l'interpréteur de commandes.



Environnement de LL3T

Les langages Hilo-3 implémentés sur LL3T sont :

- le langage HDL qui permet de modéliser les circuits à simuler de manière structurelle et fonctionnelle;
- le langage WDL qui permet de décrire les stimuli-réponses à appliquer aux circuits;
- le langage SCL qui permet de définir les conditions initiales, les opérations de rétro-annotation et le contrôle de la simulation.



Processus de compilation et de simulation

Les compilateurs compilent les descriptions du circuit et celles des stimuli-réponses en images internes. Une image interne est la représentation intermédiaire d'un circuit ou des stimuli-réponses compilés. C'est une unité élémentaire pouvant être gérée dans la base de données de LL3T.

Lors d'une session de simulation déclenchée par des commandes de simulation, les configurateurs établissent la hiérarchie du circuit à simuler, c'est-à-dire un ensemble de sous-circuits constituant le circuit principal reliés par leurs relations d'englobement.

Les images internes stockées dans les bibliothèques externes sont chargées dans la base de données de l'environnement de simulation si elles font partie des sous-circuits directs ou indirects du circuit principal. La configuration hiérarchique du circuit à simuler est donc construite dynamiquement.

L'édition de liens s'opère sur la base de la hiérarchie du circuit ainsi construite. La cohérence des connexions entre les circuits est déterminée par l'analyse des ports de connexion définis sur les interfaces des circuits. Les attributs dynamiques des sous-circuits convergent vers le circuit principal à travers ces ports de connexion. Les points de stimulus et de réponse sont également associés au circuit à simuler.

Le générateur de code transforme l'ensemble des circuits et des stimuli-réponses en image exécutable, sous forme de fichier de code contenant des informations codées en langage du simulateur.

Une image interne est très proche de la sémantique des langages Hilo-3, tandis qu'une image exécutable, qui décrit des tâches de simulation en langage du simulateur, est très proche des propriétés intrinsèques du simulateur. La transformation entre ces deux types d'images est une projection sémantique basée sur les actions suivantes :

- La hiérarchisation et la structuration des circuits ainsi que des stimuli-réponses sont réalisées par les configurateurs;
- La propagation des attributs dynamiques et la vérification de la cohérence entre les circuits sont effectuées par les éditeurs de liens.
- La transformation est faite par le générateur de code qui a une connaissance totale des règles syntaxiques et sémantiques du langage du simulateur.

• La prise en compte des conditions de simulation et le contrôle de la transformation sont réalisés par l'utilisation des commandes de simulation, qui interviennent à partir de la configuration du circuit et jusqu'à l'édition de liens.

L'ensemble des tâches de simulation est alors représenté par une image exécutable générée dans le fichier de code. Cette image est chargée et distribuée sur les unités de simulation par le DMT qui réalise l'interface entre le simulateur et l'extérieur, et supervise le déroulement de la simulation.

4.2. Simulateur

4.2.1. Principe de base

◊ Généralités

Le simulateur est un simulateur à échéancier implémenté sur une architecture d'anneau de 1 à 128 unités de simulation.

Chaque unité de simulation est dotée d'un échéancier local. Les unités de simulation simulent le circuit en s'échangeant des données entre elles.

Les unités de simulation se synchronisent à chaque étape de simulation pour pouvoir fonctionner à la même cadence.

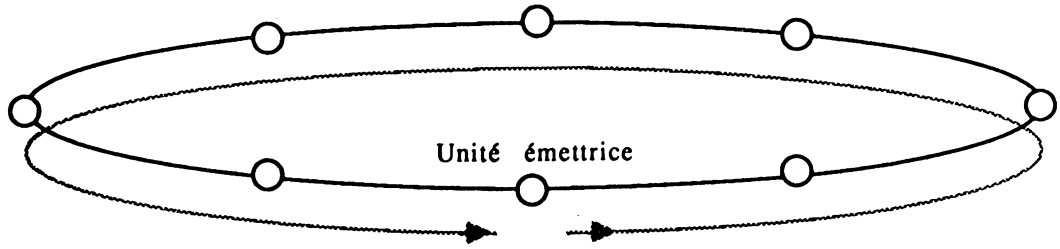
La synchronisation entre les unités de simulation est réalisée par les échanges de messages de synchronisation. Il n'existe ni processeur de contrôle, ni horloge globale.

A l'intérieur d'une unité de simulation, les processeurs s'organisent de manière pipeline. L'EMT est connecté à l'ECT nord et l'ECT sud. Les ECTs évaluent les événements, génèrent de nouveaux événements, et les envoient à l'EMT. L'EMT gère l'échéancier et distribue aux ECTs des événements à évaluer.

Une unité de simulation est connectée directement à deux unités voisines : l'unité de simulation ouest et l'unité de simulation est. Elle est capable de communiquer avec n'importe quelle autre unité de simulation du réseau.

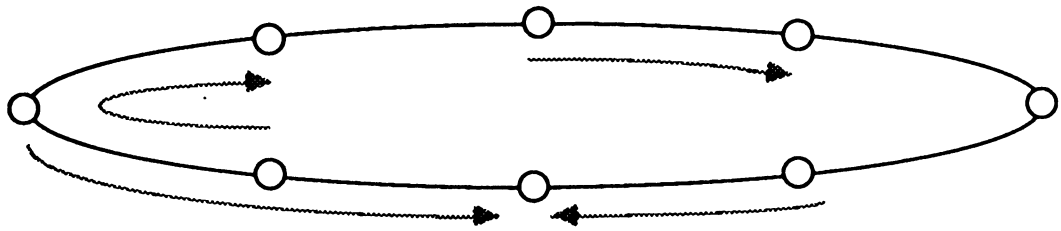
L'acheminement des messages est contrôlé par les EMTs appartenant aux unités de simulation que les messages traversent. Chaque EMT dispose d'un mécanisme qui réalise les échanges de messages.

Un message de synchronisation part d'une unité de simulation, traverse toutes les unités de simulation, et parvient à l'unité de départ. Ce type de message est systématiquement envoyé dans le sens ouest-est pour assurer une synchronisation simple et efficace de l'ensemble des unités de simulation.



Synchronisation

Les autres types de messages sont envoyés vers l'ouest ou l'est, selon la longueur minimale, pour atteindre la destination.



Messages de simulation

◊ Messages

Dans le simulateur, trois types de messages sont utilisés :

- les messages de simulation;
- les messages de contrôle;
- les messages de synchronisation.

Un message de simulation est un événement sur lequel sont codées les informations suivantes :

- le temps : l'heure à laquelle l'événement se produit;
- le status : l'insertion ou la suppression de l'événement;
- l'état logique : le niveau logique et la force;
- la destination : les identificateurs de l'unité de simulation et de l'ECT dans l'unité (nord ou sud);
- l'identificateur de l'équipotentielle : l'équipotentielle sur laquelle l'événement se produit.

Un message de contrôle permet d'échanger des informations de contrôle à l'intérieur d'une unité, notamment entre l'ECT et l'EMT. Ces messages contiennent les informations suivantes :

- le temps;
- le type de contrôle;
- l'identificateur de l'émetteur;
- les données.

Un message de synchronisation permet d'échanger des informations de synchronisation entre les unités de simulation. L'émetteur et le récepteur des messages sont tous les EMTs. Ces messages contiennent les informations suivantes :

- le temps;
- le type de synchronisation;
- l'identificateur de l'émetteur;
- les données.

◊ Synchronisation

La synchronisation entre les unités de simulation contient deux aspects distincts :

- la synchronisation globale : Sans processeur maître, les unités de simulation doivent être synchronisées pour assurer qu'à la fin de chaque étape de simulation elles ont toutes terminé leur calcul.
- la synchronisation temporelle : Sans horloge globale, les unités de simulation doivent toutes prendre à chaque étape, la même heure de simulation.

La synchronisation globale exige que les unités de simulation terminent toutes leur calcul à la fin de chaque étape de simulation. Une unité de simulation termine le calcul lorsque deux messages de contrôle sont envoyés par l'ECT nord et l'ECT sud. L'EMT de cette unité de simulation émet un message de synchronisation à son unité est. Le message circule à travers toutes les unités de simulation sur l'anneau, et revient à l'unité émettrice.

Chaque EMT a son compteur de synchronisation. Lorsqu'un message de synchronisation arrive sur une unité de simulation, elle doit le retransmettre

si elle n'est pas l'unité émettrice du message, et incrémenter le compteur de synchronisation. La synchronisation est vérifiée par une simple comparaison entre le contenu du compteur de synchronisation et le nombre d'unités de simulation dans l'anneau.

La synchronisation temporelle permet de déterminer l'heure de la prochaine étape de simulation de toutes les unités de simulation.

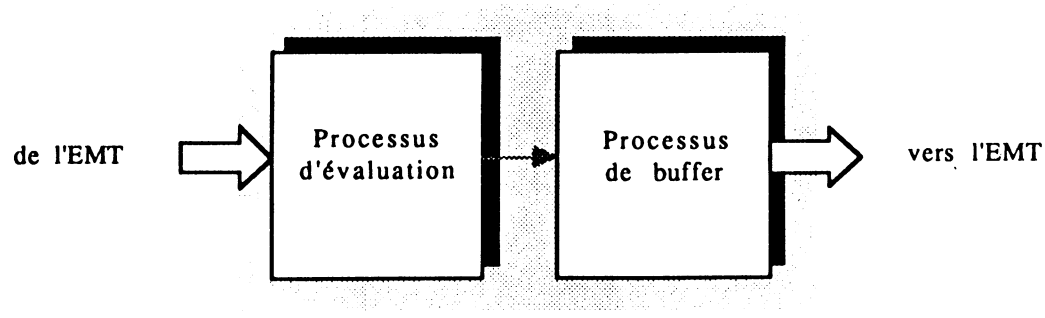
Chaque unité de simulation connaît l'heure de sa prochaine étape de simulation en consultant le premier événement de son échéancier. Cette information est envoyée sous forme de message de synchronisation à l'unité de simulation est.

Dès qu'une unité de simulation connaît les propositions en provenance de toutes les unités, elle sélectionne la valeur minimale comme l'heure de la prochaine étape de simulation, et commence immédiatement la simulation.

4.2.2. ECT

◊ Architecture

L'ECT contient un processus d'évaluation des événements, et un processus de buffer dans lequel sont stockés des événements à envoyer à l'EMT :



Architecture de l'ECT

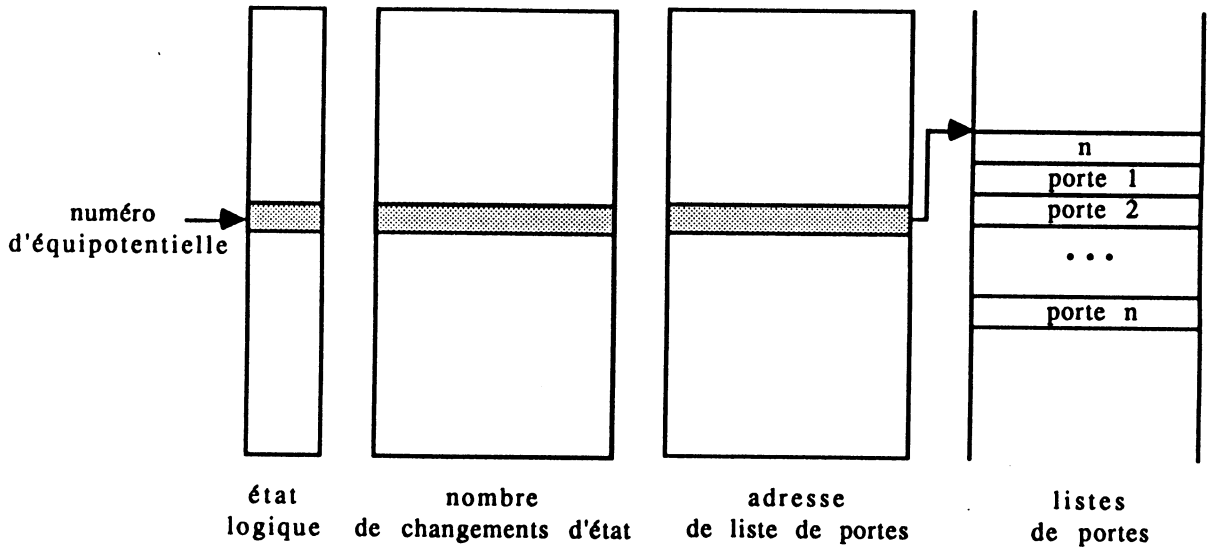
◊ Structure de données

Chaque ECT mémorise une partie du circuit à simuler, décrite par les informations suivantes :

- les informations des équipotentielles;
- les informations des portes.

Les informations d'une équipotentielle sont :

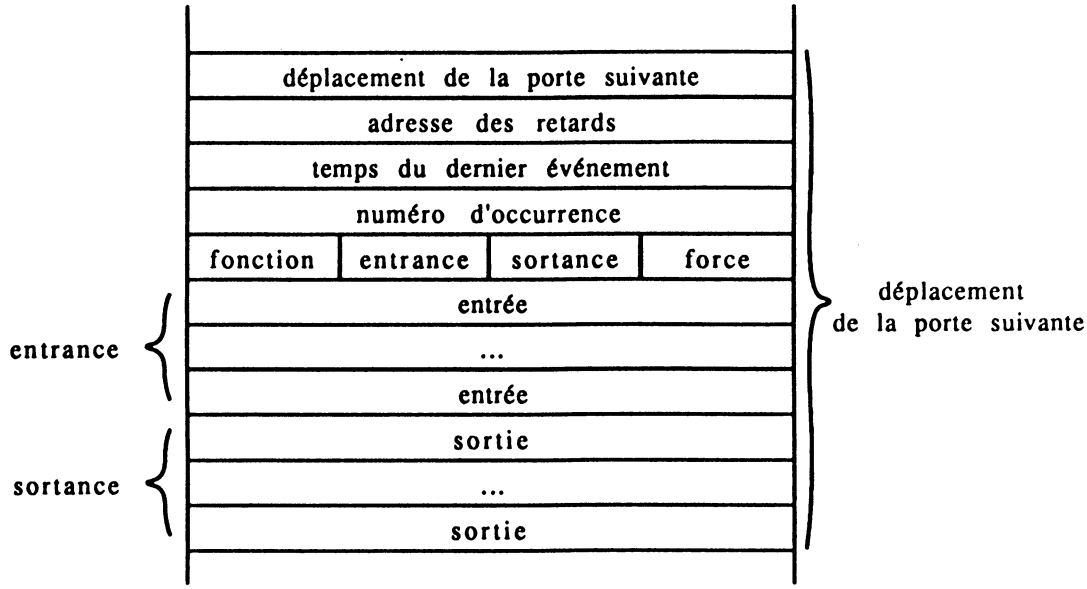
- l'état logique courant de l'équipotentielle;
- le nombre courant de changements d'état sur l'équipotentielle;
- la liste des portes ayant l'équipotentielle comme entrée.



Informations d'une équipotentielle

Les informations d'une porte sont :

- le déplacement de la porte suivante;
- l'adresse des retards;
- le temps du dernier événement;
- le numéro d'occurrence;
- la fonction de la porte;
- l'entrance de la porte;
- la sortance de la porte;
- la force de sortie;
- la liste des équipotentielles d'entrée;
- la liste des équipotentielles de sortie.



Informations d'une porte

Un élément fonctionnel est décrit par, en plus de ces informations, une liste d'instructions fonctionnelles.

◊ Interpréteur fonctionnel

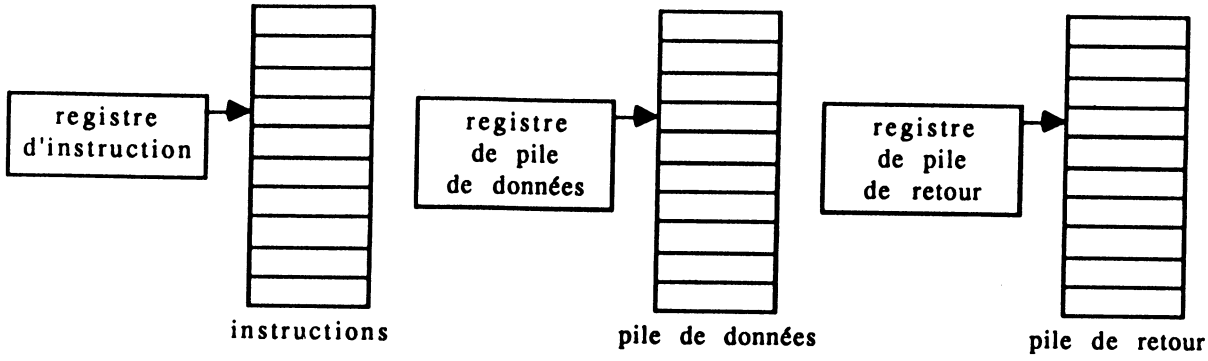
L'interpréteur fonctionnel est une machine virtuelle faisant partie du processus d'évaluation de l'ECT. C'est une machine à structure de pile qui utilise deux piles et trois registres. Lors de l'évaluation d'un élément fonctionnel, la machine virtuelle interprète les instructions fonctionnelles définies dans cet élément.

Les deux piles de l'interpréteur fonctionnel sont utilisées pour le calcul des données et les appels :

- la pile de données;
- la pile de retour.

Les trois registres sont :

- le registre d'instruction;
- le registre de pile de données;
- le registre de pile de retour.



Structure de données de l'interpréteur fonctionnel

Le jeu d'instructions de cette machine permet d'effectuer différentes opérations fonctionnelles.

Les instructions suivantes effectuent les empilements et dépilements des adresses et des données :

- LDC données : l'empilement d'une constante;
- LDH adresse : l'empilement d'une variable locale;
- STH adresse : le dépilement de données vers une variable locale;
- LDG adresse : l'empilement d'une variable globale;
- STG adresse : le dépilement de données vers une variable globale;
- LDR adresse : l'empilement de l'adresse d'une ROM;
- LDM adresse : l'empilement de l'adresse d'une RAM;
- LDRG adresse : l'empilement de l'adresse d'un registre;
- LDW adresse : l'empilement de l'adresse d'une équipotentielle.

Les instructions suivantes effectuent des opérations unaires et binaires :

- SME opérateur, longueur : l'opération unaire sur un vecteur (les opérateurs sont : NOP, NOT, EVENT et REGISTER);
- SDE opérateur, longueur : l'opération binaire sur deux vecteurs (les opérateurs sont : AND, OR, XOR, SAME, NAND, NOR, NXOR, PLUS et DIFF).

Les instructions suivantes effectuent des comparaisons :

- BDE opérateur, longueur : la comparaison entre deux vecteurs (les opérateurs sont : EQ, GE, LE, NE, GT et LT);
- EQC données : la comparaison avec une constante;

Les instructions suivantes effectuent des branchements :

- BRA déplacement : le branchement relatif;
- BRZ déplacement : le branchement relatif si zéro;
- BRNZ déplacement : le branchement relatif si non zéro;
- CALL déplacement : l'appel d'une sous-routine;
- RETURN : le retour à la routine appelante;
- EXIT : la fin.

Les instructions suivantes effectuent des opérations temporelles :

- SEND opérateur, longueur : l'envoi des événements avec un retard spécifique (les opérateurs sont : NOP, NOT et EVENT);
- TTIME opérateur, longueur : les fonctions temporelles (les opérateurs sont : STEADY et WIDTH);
- CHGW adresse : la génération d'un chronogramme scalaire;
- CHGV adresse : la génération d'un chronogramme vectoriel;
- PRT opérateur, adresse : la visualisation d'un message à l'heure courante (les opérateurs sont : STROBE, MSG, TIME et STATE).

Les instructions suivantes permettent des opérations de test :

- CHECK opérateur, longueur : le test sur les réponses du circuit (les opérateurs sont : CHECK, IGNORE et LEARN);
- FRTO états logiques, longueur : le test de changement d'état;
- CKMEM longueur : le test sur l'adresse d'une ROM ou RAM;
- INSET longueur : le test sur un intervalle.

4.2.3. EMT

◊ Architecture

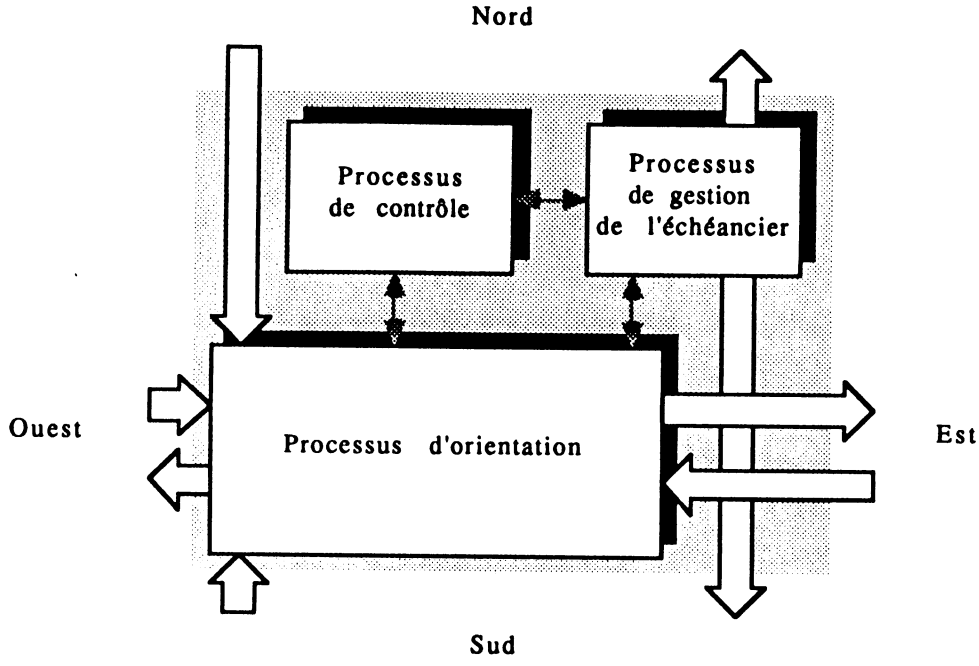
Les fonctions principales de l'EMT sont :

- la gestion de l'échéancier local de l'unité de simulation;
- le contrôle d'acheminement des messages;
- la synchronisation de la simulation.

Donc l'EMT est constitué de trois processus qui accomplissent les trois

fonctions suivantes :

- le processus de gestion de l'échéancier;
- le processus d'orientation;
- le processus de contrôle.



Architecture de l'EMT

◊ Processus de gestion de l'échéancier

Le processus gère l'échéancier local de l'unité de simulation. Il communique avec le processus de contrôle et le processus d'orientation.

Les informations en provenance du processus de contrôle permettent la synchronisation de la gestion de l'échéancier. Les informations en provenance du processus d'orientation sont les événements à gérer dans l'échéancier.

Le processus de gestion de l'échéancier transmet, à l'ECT nord et à l'ECT sud, des événements à évaluer.

◊ Processus d'orientation

Le processus gère l'acheminement des messages. Il reçoit les informations provenant des unités de simulation ouest ou est, ou provenant

des deux ECTs nord et sud.

Il reçoit trois types de messages :

- les messages de simulation;
- les messages de synchronisation provenant des unités de simulation ouest et est;
- les messages de contrôle et de simulation provenant des ECTs nord et sud de l'unité de simulation.

Pour un message de simulation, son adresse est d'abord identifiée par le processus. Si c'est un message destiné à l'ECT nord ou sud de l'unité de simulation, alors il est transmit au processus de gestion de l'échéancier. Sinon, il est renvoyé à l'unité de simulation correspondante.

Lorsque le message reçu par le processus est un message de contrôle ou de synchronisation, le processus d'orientation le transmet au processus de contrôle, ou le renvoie à l'unité de simulation correspondante.

Le choix de l'orientation des messages à envoyer aux unités de simulation voisines est déterminé par les règles suivantes :

- Si c'est un message de synchronisation, il est envoyé à l'unité de simulation est systématiquement;
- Sinon, le message est envoyé vers l'est ou l'ouest en fonction de la distance minimale entre l'unité de simulation courante et la destination.

◊ Processus de contrôle

Le processus de contrôle assure la synchronisation globale entre les unités de simulation. Cette synchronisation est basée sur la technique de comptage des messages suivants :

- les messages de synchronisation en provenance des autres unités de simulation;
- les messages de contrôle en provenance des ECTs nord et sud.

Un compteur de synchronisation est utilisé qui s'incrémente lors de la réception d'un message de synchronisation. La synchronisation est établie si le contenu du compteur est égal au nombre d'unités de simulation.

4.2.4. DMT

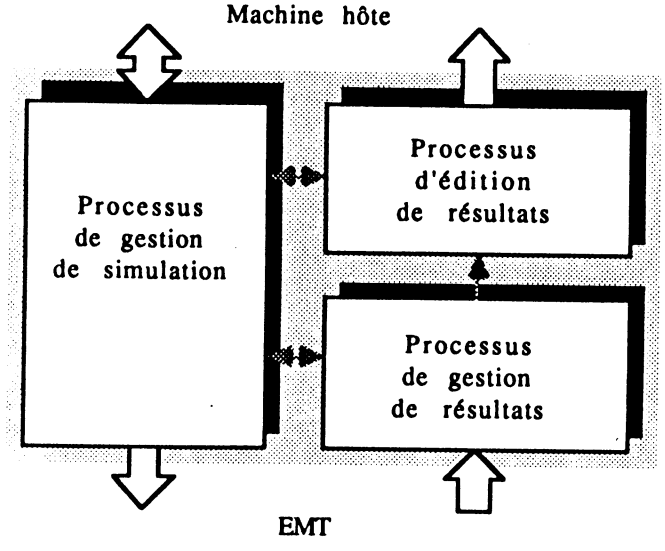
◊ Architecture

Le DMT établit les communications entre le simulateur et la machine hôte. D'une part, il permet à l'utilisateur de superviser le simulateur. D'autre part, il reçoit les événements en provenance de différentes unités de simulation et constitue l'historique de changements d'état logique de chaque équipotentielle à visualiser. Ses principales fonctions sont :

- l'initialisation et le contrôle de la simulation;
- la saisie et la gestion des résultats de la simulation;
- l'édition des résultats durant la simulation.

Il se compose de trois processus :

- le processus de gestion de simulation;
- le processus de gestion de résultats;
- le processus d'édition de résultats.



Architecture du DMT

◊ Principaux processus du DMT

Le processus de gestion de simulation contrôle le déroulement de la simulation :

- le chargement du circuit et des stimuli;
- l'initialisation de la simulation;
- la visualisation des informations durant la simulation;
- l'exécution des commandes de contrôle.

Le processus de gestion de résultats permet la saisie des résultats partiels sous forme de messages en provenance de toutes les unités de simulation.

Le processus d'édition de résultats est une tâche concurrente qui permet d'éditer les résultats partiels pendant que la simulation se déroule. Le processus d'édition de résultats autorise les interactions suivantes :

- le formatage des résultats;
- l'analyse des résultats, telle que la recherche des configurations spécifiques, l'échantillonnage;
- la création des fichiers de résultats et de capture.

4.3. Compilateurs

4.3.1. Structure générale

◊ Généralités

LL3T comprend trois compilateurs, chacun correspondant à un langage de Hilo-3 :

• le compilateur de circuits qui compile les descriptions du langage HDL;

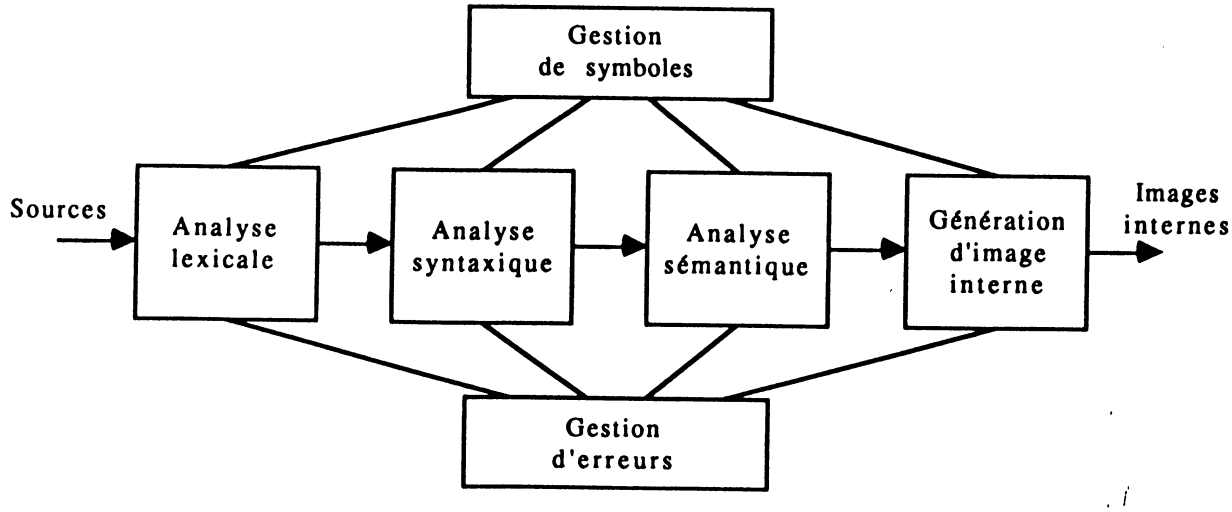
• le compilateur de stimuli-réponses qui compile les descriptions du langage WDL;

• le compilateur de commandes de simulation qui compile les descriptions du langage SCL.

Les compilateurs de circuits et de stimuli-réponses réalisent les fonctions suivantes :

• la vérification de la syntaxe et la vérification statique de la sémantique des descriptions;

• La création des images internes qui sont les représentations intermédiaires des circuits et des stimuli-réponses.



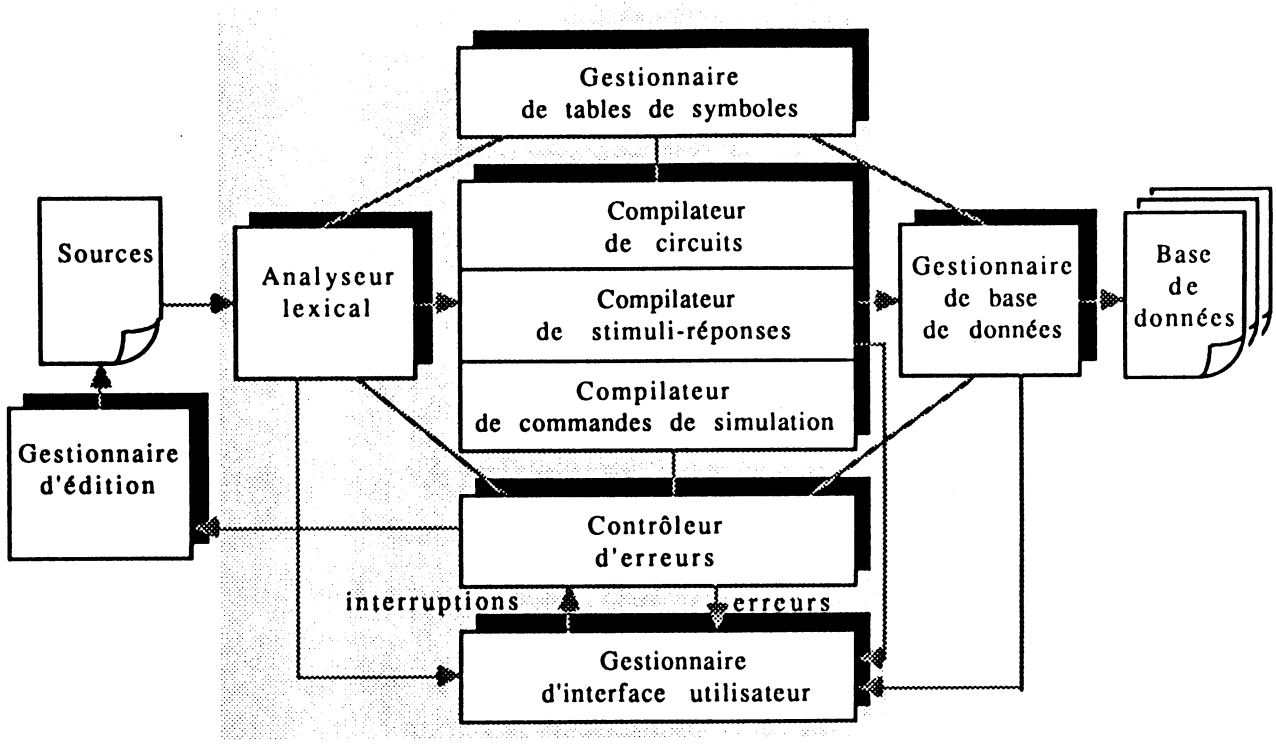
Processus de compilation de circuits et de stimuli-réponses

La compilation de commandes de simulation réalise les fonctions suivantes :

• la vérification de la syntaxe et de la sémantique des commandes de simulation;
• les manipulations des images internes;
• les opérations de rétro-annotation;
• le contrôle de la session de simulation.

◊ Organisation générale

La compilation est réalisée par un ensemble de processus qui s'organisent de manière suivante :



Organisation générale des compilateurs

L'analyseur lexical réalise les fonctions suivantes :

• la gestion et l'analyse des fichiers sources;
• la constitution des unités lexicales.

Le gestionnaire de base de données réalise les opérations suivantes :

• la gestion des images internes dans la base de données;
• la coordination entre la base de données et les bibliothèques de système et d'utilisateur.

Le contrôleur d'erreurs joue les rôles suivants :

• la gestion des erreurs en provenance des compilateurs;
• la détection des interruptions externes;
• le contrôle des compilateurs;
• l'appel de l'éditeur de texte pour la correction des erreurs lexicales, syntaxiques ou sémantiques.

Les erreurs pouvant être détectées et éventuellement localisées sont les suivantes :

• les erreurs lexicales en provenance de l'analyseur lexical;
• les erreurs syntaxiques en provenance des compilateurs;
• les erreurs sémantiques en provenance des compilateurs;
• les erreurs logiques en provenance du gestionnaire de base de données, telles que des englobements récursifs de circuits ou des sous-circuits non définis;
• les erreurs internes telles qu'un débordement de mémoire ou un problème d'accès aux fichiers.

Le gestionnaire d'interface utilisateur réalise les fonctions suivantes :

• la visualisation des informations en provenance des compilateurs, du contrôleur d'erreurs, et du gestionnaire de base de données;
• le formatage des dialogues : le multi-fenêtrage, le questionnaire, le formulaire et les touches de fonction et de contrôle.

Le gestionnaire d'édition assure les fonctions suivantes :

• le contrôle de l'éditeur de texte;
• la localisation automatique de l'erreur détectée par la compilation;
• la gestion du contexte de compilation.

Le gestionnaire de tables de symboles gère deux tables de symboles :

• la table des mots clefs;

• la table des identificateurs définis par l'utilisateur.

◊ Images internes

Une image interne contient des informations compilées d'une description de circuit, ou de stimuli-réponses. L'utilisation des images internes est due aux considérations suivantes :

• Les langages Hilo-3 sont les langages qui ont des caractéristiques interactives, permettant de construire, dans un environnement de simulation, des modèles de circuit et des patterns de stimuli-réponses d'une manière incrémentale. Les compilateurs transforment les descriptions en images internes qui peuvent être ensuite gérées dans l'environnement de simulation.

• Les images internes facilitent la gestion de la base de données de l'environnement de simulation et la création des bibliothèques des modèles pré-compilés.

• Il existe des commandes de simulation qui peuvent référencer et modifier les circuits déjà compilés, en particulier des commandes de rétro-annotation. Les propriétés des objets définis dans les images internes sont proches de la sémantique des langages Hilo-3. Cette nature assure la réalisation de ce genre de commandes.

• L'édition de liens est réalisée dynamiquement qui a lieu lors d'une session de simulation. Statiquement, il n'existe pas de relations de dépendance entre les images internes. Par conséquent, la modification d'un circuit n'entraîne pas une recompilation totale des circuits qui englobent le circuit modifié.

• L'image interne représente une étape intermédiaire de la compilation. Cette image interne comble la lacune entre la sémantique des langages Hilo-3 et celle du simulateur.

• L'image interne a une structure de données permettant la décompilation qui facilite la récupération de sources en cas de la mise à jour de la conception des circuits.

Les compilateurs de circuits et de stimuli-réponses utilisent la techniques de la traduction dirigée par syntaxe. La structure de données de l'image interne est basée sur des objets syntaxiques auxquels sont associés des attributs :

• Un ensemble de règles syntaxiques permettent de construire la

structure arborescente des objets de l'image interne.

• Un ensemble de règles sémantiques permettent de déterminer et de vérifier les attributs associés à ces objets.

4.3.2. Compilateur de circuits

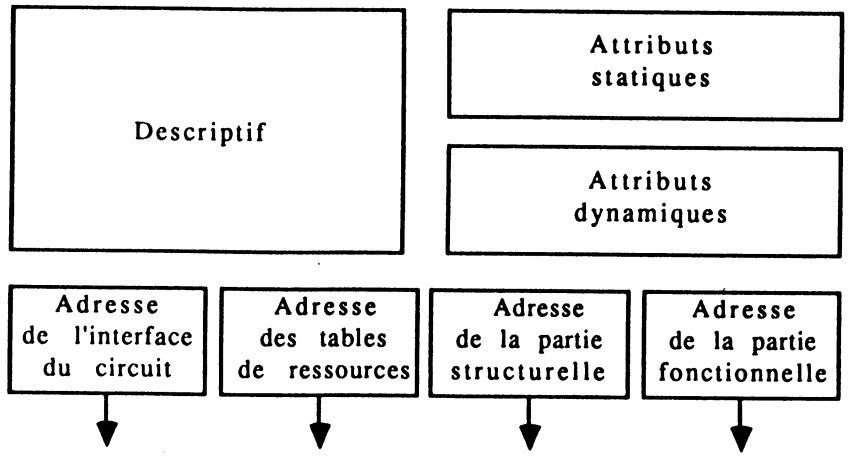
◊ Structure de l'image interne

L'image interne du circuit se compose de :

- l'en-tête;
- l'interface du circuit;
- les tables de ressources;
- la description structurelle;
- la description fonctionnelle.

L'en-tête contient les informations suivantes :

- le descriptif de l'ensemble du circuit;
- les attributs statiques déterminés par la compilation;
- les attributs dynamiques déterminés par la configuration et l'édition de liens.



En-tête

L'interface du circuit est constituée de :

- une liste de paramètres formels;

• une liste de ports de connexion.

Les paramètres sont typés, pouvant être utilisés dans la description structurelle (les retards) ou fonctionnelle (les entiers impliqués dans la structure de contrôle). Un paramètre peut représenter un, deux ou trois valeurs effectives lors de l'instantiation.

Les ports de connexion sont caractérisés par leur type, leur direction et leur fonction logique. En plus chaque port conserve un champs permettant de propager les attributs dynamiques des équipotentielles entre les circuits pendant la configuration et l'édition de liens.

Chaque circuit possède ses propres ressources gérées dans :

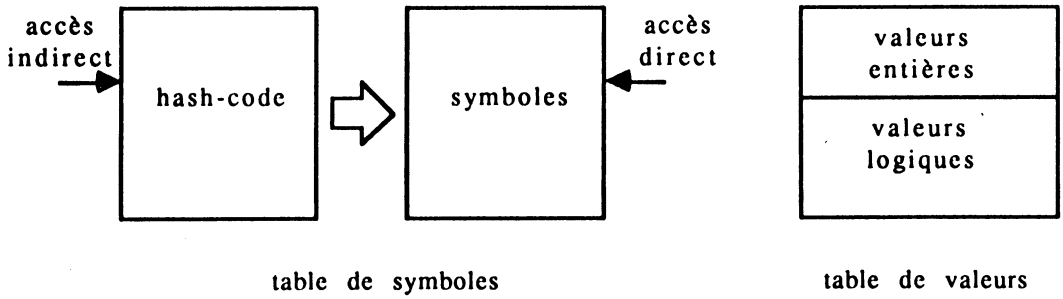
• une table de symboles;
• une table de valeurs.

La table de symboles permet deux sortes d'accès :

• l'accès direct aux symboles par l'indexation;
• l'accès indirect par la table de hash-code.

La table de valeurs contient :

• les valeurs entières;
• les valeurs logiques.

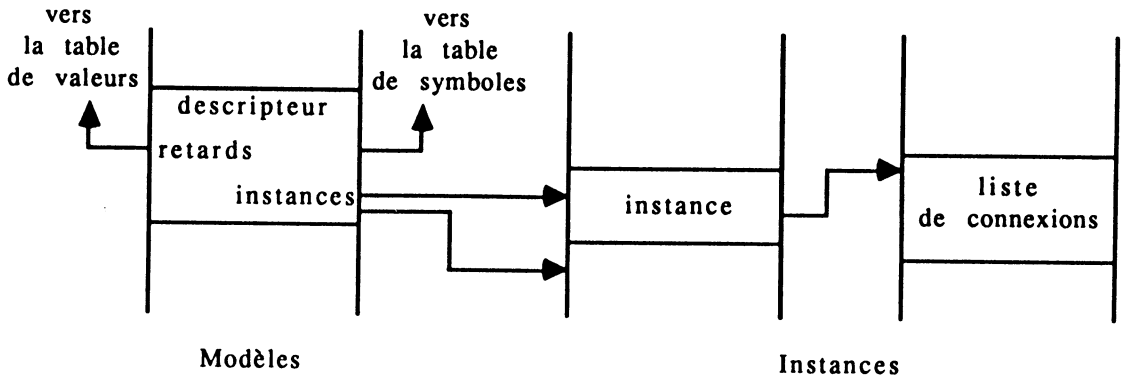


Ressources

La description structurelle du circuit est représentée par un ensemble d'éléments et un ensemble d'équipotentielles. Ces deux ensembles sont organisés sous forme de table et de liste pour assurer une structure de données linéaire et un mécanisme d'accès direct.

Les éléments sont représentés par :

• la liste de modèles qui décrivent les propriétés génériques;
• la liste d'instances qui représentent les objets individuels, et la liste de connexions.



Eléments

Le modèle peut être un des éléments suivants :

• une porte logique;
• un transistor;
• un élément fonctionnel;
• une instance de sous-circuit.

Il définit :

• la classe de l'élément qui distingue les primitives et les sous-circuits, ainsi que la classification des primitives;
• les attributs internes de l'élément;
• la fonction logique de l'élément;
• les forces de sortie;
• les retards de sortie.

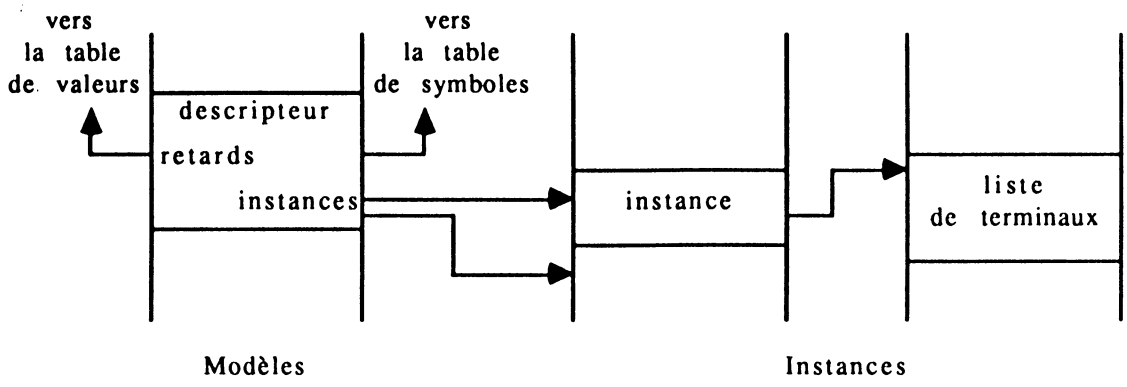
Une instance d'élément, qui hérite les propriétés définies par son modèle, contient deux informations complémentaires :

• le descripteur de l'instance;
• la liste de connexions décrivant les équipotentielles connectées à

cette instance.

Les équipotentielles sont représentées par :

- la liste de modèles des équipotentielles;
- la liste d'instances.



Equipotentielles

Les modèles définissent les propriétés suivantes :

- la classe : interne (locale au circuit), externe (connectée à l'extérieur du circuit), et absolue (0, 1, X, Z ou une équipotentielle nulle);
- les attributs internes;
- le type;
- la fonction logique;
- les retards associés.

En plus des propriétés définies par leurs modèles, les instances contiennent également :

- le descripteur de l'instance;
- la liste de terminaux qui décrivent tous les éléments connectés à l'instance.

La description fonctionnelle du circuit est représentée par des graphes arborescents. Elle est basée sur quatre types d'objets :

- les expressions arithmétiques et logiques, qui représentent les structures combinatoires;
- les registres virtuels, qui pouvent avoir une expression de

déclenchement associée;

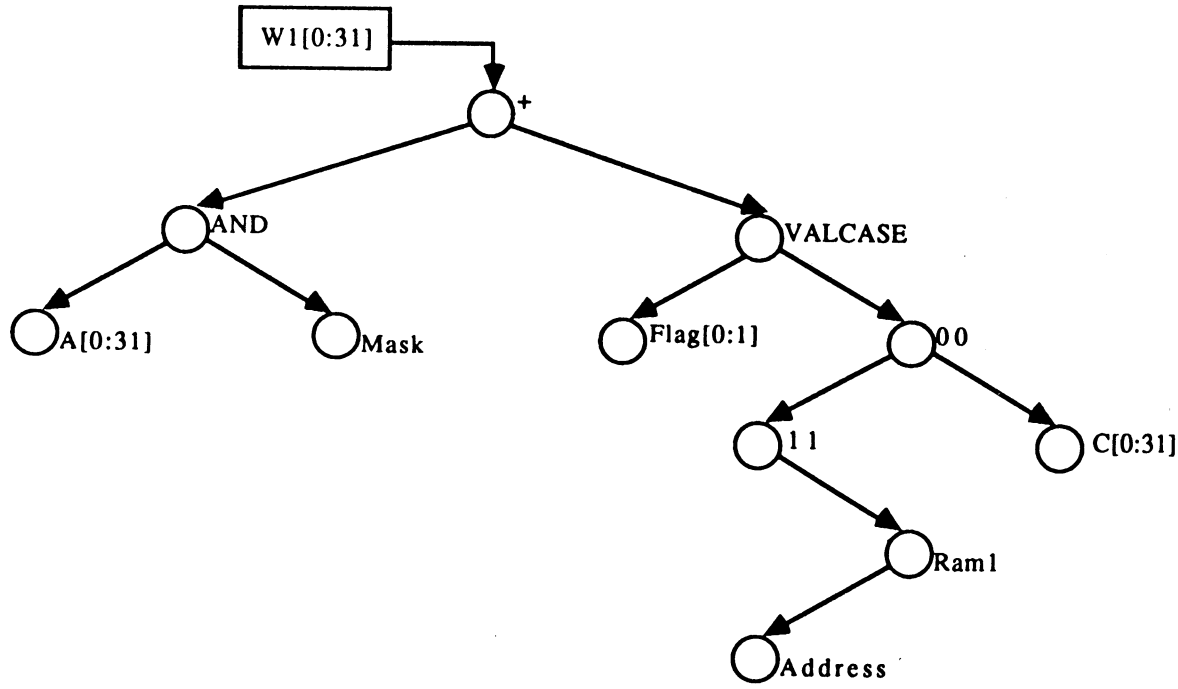
• les expressions d'événements, qui définissent les conditions de déclenchement des instructions gardées;

• les listes d'actions, qui décrivent les opérations à entreprendre dans les instructions gardées.

Les graphes arborescents expriment de façon naturelle l'arbre syntaxique des expressions.

Voici un exemple d'une expression arithmétique et logique :

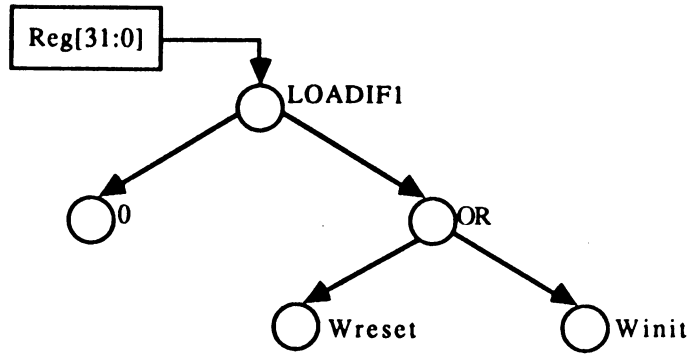
```
WIRE (10:12:14,10:11:13) W1 [0:31] = (A[0:31] AND Mask) + VALCASE Flag[0:1],
                                                          00 : C[0:31],
                                                          11 : Ram1[Address],
                                                          ENDCASE;
```



Expression arithmétique et logique

L'exemple suivant est un registre virtuel avec une expression de déclenchement associée :

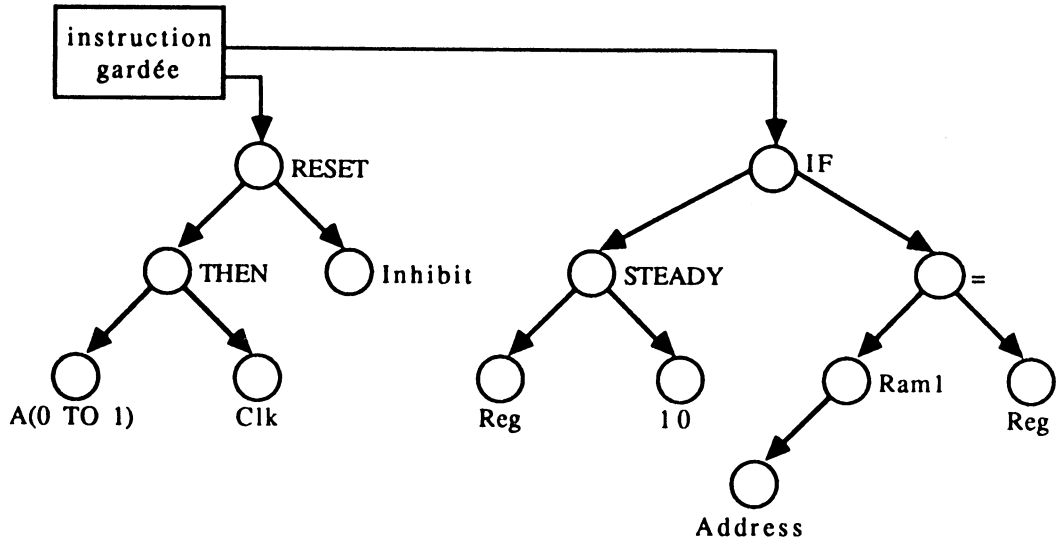
```
REGISTER (10,10)
        Reg [31:0] = 0 LOADIF1 Wreset OR Winit;
```



Expression de registre

L'exemple suivant est une instruction gardée constituée d'une expression d'événements et d'une liste d'actions :

```
WHEN A(0 TO 1) THEN Clk RESET Inhibit DO
        IF STEADY (Reg,10) THEN Ram1[Address] = Reg ENDIF;
```



Expression d'événements et liste d'actions

L'utilisation du graphe arborescent dans l'image interne est due aux raisons suivantes :

• Les expressions ne sont pas complètement arithmétiques. Des structures de données spéciales et des fonctions non classiques peuvent être impliquées dans l'expression.

• Le graphe est constitué non seulement des expressions, mais aussi des mécanismes de contrôle qui définissent le flot de contrôle.

• La notation postfixée ne permet pas d'exprimer la structure arborescente des expressions explicitement.

• Les opérations décrites dans les expressions peuvent être temporelles. Elles ne sont pas exécutées instantanément dans un ordre séquentiel.

Les objets utilisés dans les expressions peuvent être :

• les équipotentielles scalaires ou vectorielles;
• les mémoires;
• les événements fonctionnels;
• les constantes logiques.

Les éléments structurels (les portes et les sous-circuits) ne sont pas impliqués directement dans les expressions fonctionnelles.

Les opérations utilisées dans les expressions peuvent être :

• les évaluations logiques et arithmétiques : AND, OR, +, -, etc.
• les opérations de branchement : IF, CASE, etc.
• l'enchaînement des événements : THEN, OR, RESET, etc.
• les fonctions temporelles : WAIT, WIDTH, STEADY, etc.
• les transferts de données : les affectations de registre et de RAM, etc.
• les commandes : DISPLAY, HALT, etc.

◊ Phases de compilation

La compilation du circuit se déroule en quatre phases :

• la vérification syntaxique de la description du circuit et la création de l'image interne du circuit;
• la création de la structure topologique du circuit;
• l'analyse sémantique, la détermination des attributs statiques du circuit, des éléments et des équipotentielles;
• le raffinement et l'optimisation de l'image interne du circuit, et la création des références internes et externes du circuit.

La première phase de la compilation détecte et avertit toutes les erreurs syntaxiques. L'image interne rudimentaire du circuit est construite à la fin de cette phase. La structure topologique du circuit n'est que

la fin de cette phase. La structure topologique du circuit n'est que partiellement connue. Les informations complémentaires des objets non déclarés restent manquantes.

La phase suivante complète la structure topologique du circuit. Cela consiste à :

- compléter les informations manquantes des objets non déclarés;
- compléter les listes de connexions des instances d'éléments si nécessaire;
- compléter les listes de terminaux des instances d'équipotentielles si nécessaire.

La phase d'analyse sémantique du circuit se fait lorsque la structure topologique est complète. Pour la description structurelle, les éléments, les équipotentielles et le circuit lui-même sont affectés des attributs statiques. En ce qui concerne la description fonctionnelle, les différents types de données sont vérifiés ou convertis. Les erreurs sémantiques statiquement détectables sont signalées.

La dernière phase consiste à raffiner et à optimiser l'image interne. Les relations internes entre différents types de structures de données sont établies par la création des références internes et externes.

4.3.3. Compilateur de stimuli-réponses

◊ Structure de données

Le compilateur de stimuli-réponses compile une description des stimuli-réponses, et la transforme en image interne.

La description des stimuli-réponses définit :

- les points de stimulus, de réponse et bidirectionnels du circuit à simuler;
- les stimuli à appliquer au circuit;
- les réponses souhaitées du circuit;
- les opérations de contrôle de la simulation.

Les points de stimulus, de réponse et bidirectionnels peuvent être les

ports de connexion ou n'importe quelle équipotentielle interne du circuit à simuler.

Les stimuli et les réponses sont représentés sous forme de chronogrammes. L'ensemble de la description définit les activités temporelles et concurrentes du monde extérieur du circuit simulé.

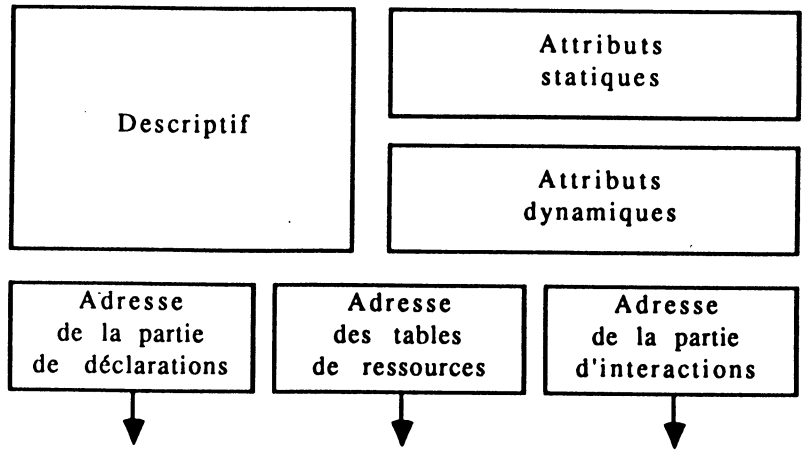
La structure de l'image interne permet d'exprimer d'une part les caractéristiques des points de stimulus et de réponse du circuit, d'autre part la nature temporelle et concurrente des interactions entre le circuit et le monde extérieur.

L'image interne est constituée de :

- l'en-tête;
- la partie de déclarations;
- les tables de ressources;
- la partie d'interactions.

L'en-tête définit les informations suivantes :

- le descriptif;
- les attributs statiques;
- les attributs dynamiques.



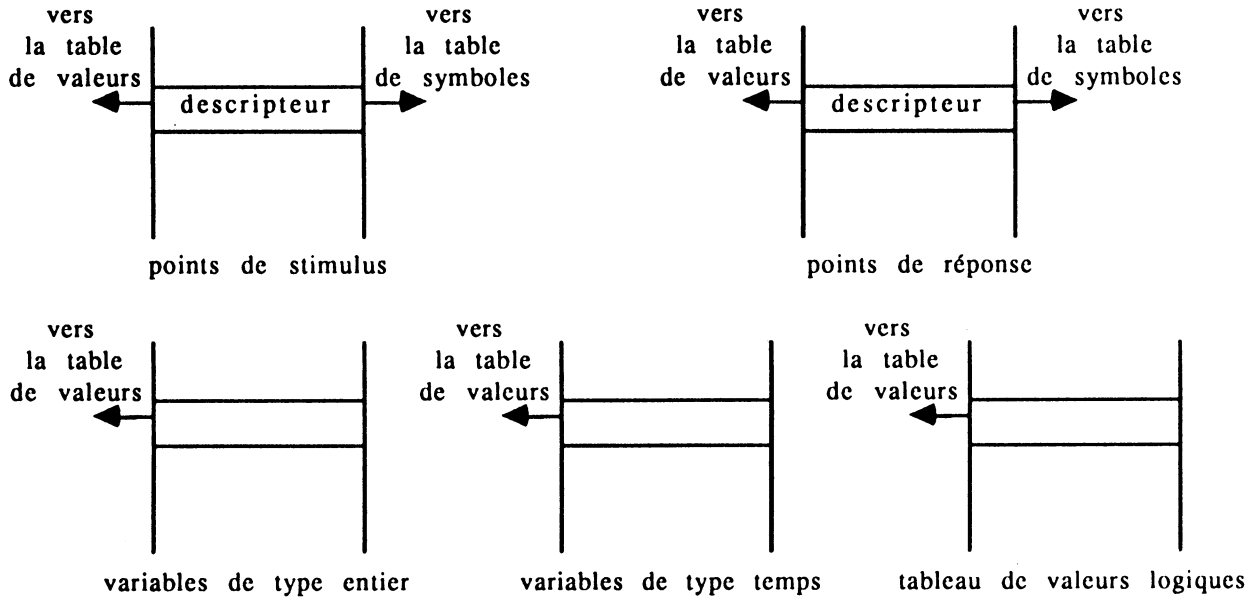
En-tête

Le descriptif contient les informations générales de l'image interne. Les attributs statiques, déterminés par la compilation, décrivent les

propriétés internes de l'image. Les attributs dynamiques, déterminés par la configuration et l'éditions de liens, caractérisent la liaison entre le contexte de simulation et le circuit.

La partie de déclarations contient :

- les points de stimulus;
- les points de réponse;
- les points bidirectionnels;
- les variables de type entier;
- les variables de type temps;
- les tableaux de valeurs logiques.



Partie de déclaration

Les tables de ressources contiennent les valeurs suivantes :

- la table de symboles;
- la table de valeurs;

La table de symboles contient les symboles utilisées dans la partie de déclaration. La table de valeurs contient deux types de valeurs :

- les valeurs entières;
- les valeurs logiques.

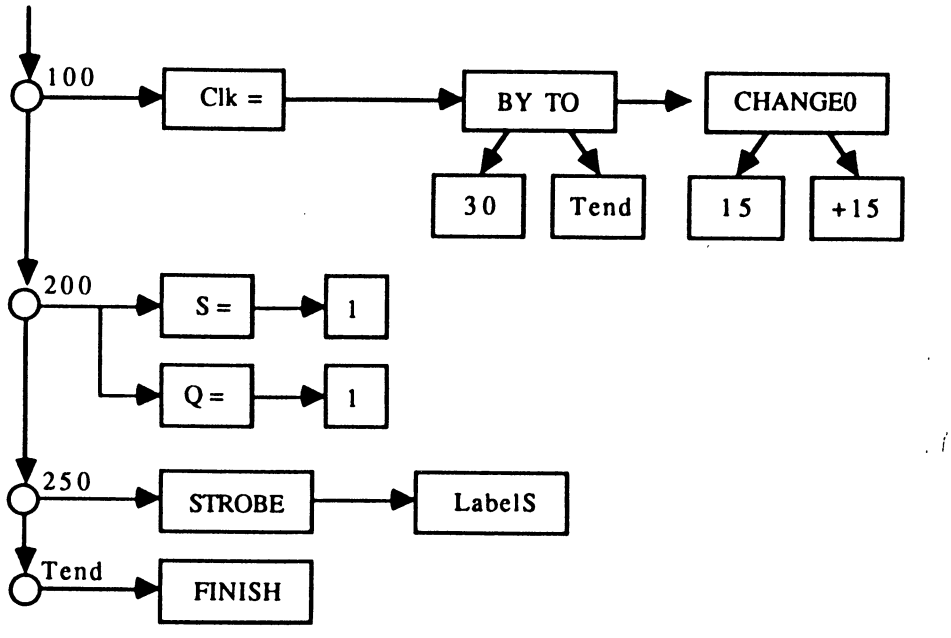
Les interactions sont :

- les affectations sur les points de stimulus;
- les affectations sur les points de réponse;
- les affectations des variables de types entier et temps;
- les commandes.

Les interactions sont représentées par un graphe :

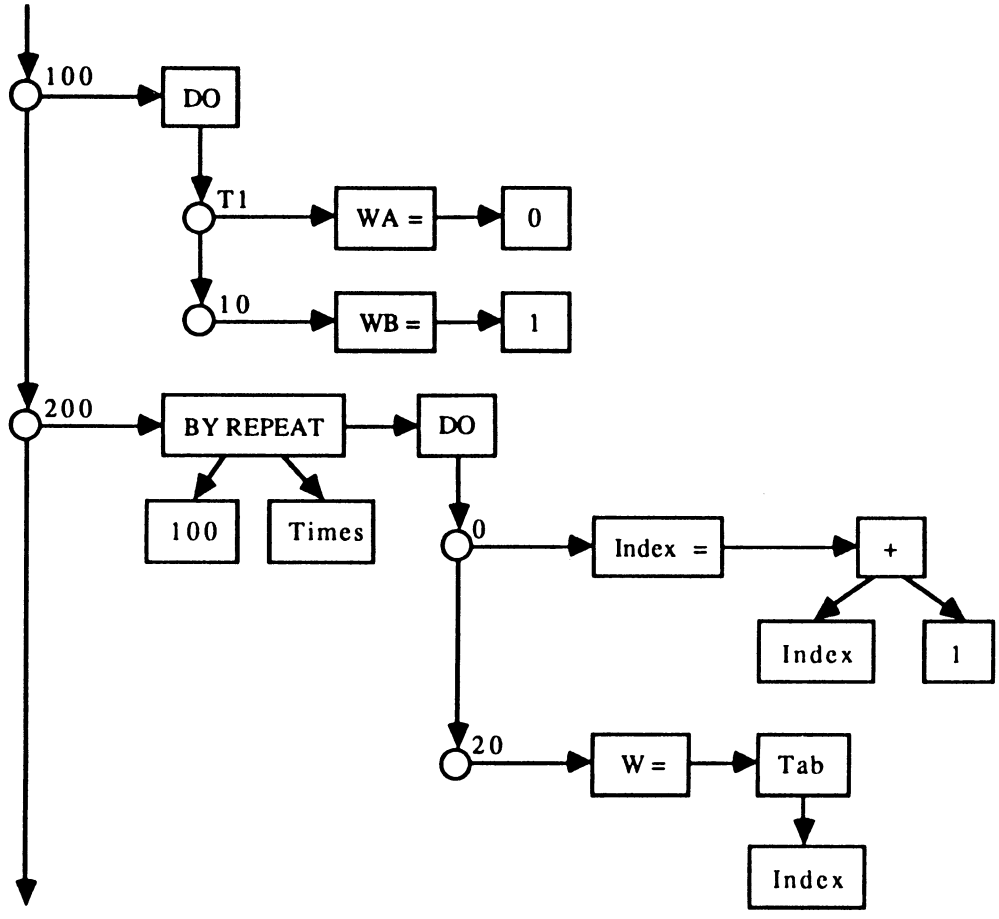
- Les noeuds temporels sont ordonnés par ordre chronologique.
- Sont associées à chaque noeud temporel, un ensemble d'activités qui se produisent simultanément à l'instant défini par le noeud;
- Une activité peut avoir une durée, par exemple celle d'un bloc. Cette activité est donc un processus concurrent qui coexiste avec le processus principal.
- Les blocs peuvent être imbriqués. Les blocs englobants déterminent les conditions de déclenchement temporel des blocs englobés.

```
100    Clk = BY 30 TO Tend CHANGE0 (15,+15);
200    S = 1 Q = 1;
250    STROBE LabelS;
Tend   FINISH.
```



Un bloc est un ensemble d'interactions ordonnées chronologiquement, qui a une base de temps relative.

```
100 DO (T1     WA = 0;
        10     WB = 1);
200 BY 100 REPEAT Times
                DO     (0      Index = Index + 1;
                        20     W = Tab[Index]);
```



Imbrication des blocs

◊ Phases de compilation

La compilation comprend trois phases :

• La vérification syntaxique et la construction de l'image interne;
• L'analyse sémantique et l'affectation des attributs statiques aux objets de l'image interne;

• L'optimisation de l'image et la création des références internes et externes.

La première phase de la compilation consiste à analyser la description source et à vérifier la syntaxe pour créer une première version de l'image interne.

La deuxième phase effectue l'analyse sémantique pour déterminer les attributs statiques des objets du circuit. Les aspects temporels des interactions sont également vérifiés. Les erreurs statiquement détectables telles que le retour en arrière de l'évolution du temps sont détectées.

La dernière phase optimise l'image construite et complète les relations entre les objets et crée les références internes et externes. L'optimisation de l'image interne est basée sur les attributs statiques déterminés à la phase précédente, par exemple, la substitution des variables jamais affectées par les constantes. Les références internes et externes sont ensuite établies et seront utilisées par la configuration et l'édition de liens.

4.4. Configurateurs

4.4.1. Généralités

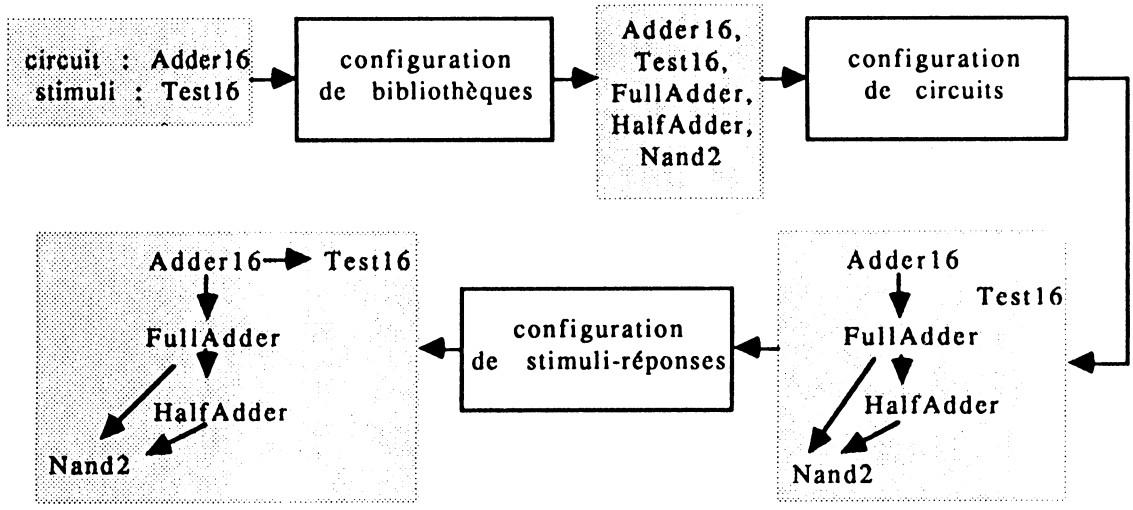
LL3T comprend les configurateurs suivants :

• le configurateur de bibliothèques qui permet de faire référence aux bibliothèques;

• le configurateur de circuits qui crée les relations de dépendance entre les circuits;

• le configurateur de stimuli-réponses qui établit l'interface et les interconnexions entre le circuit et les stimuli-réponses.

L'exemple suivant montre la configuration d'un circuit additionneur 16 bits :



Configuration d'un additionneur 16 bits

Les configurateurs jouent les rôles suivants :

• Les langages Hilo-3 ne définissent ni les relations d'héritage ni celles de dépendance entre les modèles de circuits. Aucune relation entre le circuit et ses sous-circuits n'est établie au moment de la compilation du circuit. Pour effectuer une édition de liens dynamique lors d'une session de simulation, il faut d'abord déterminer, en utilisant les configurateurs, quels sont les circuits concernés par la simulation et où se situent les images internes de ces circuits, et quelles sont les relations de dépendance entre les circuits.

• Lors de la compilation d'une description de stimuli-réponses, il n'existe pas de liaison entre le circuit et les stimuli-réponses. L'utilisation du configurateur permet de localiser et de charger dans la base de données l'image interne des stimuli-réponses et surtout d'établir les relations entre cette image interne et celle du circuit, pour assurer l'édition de liens.

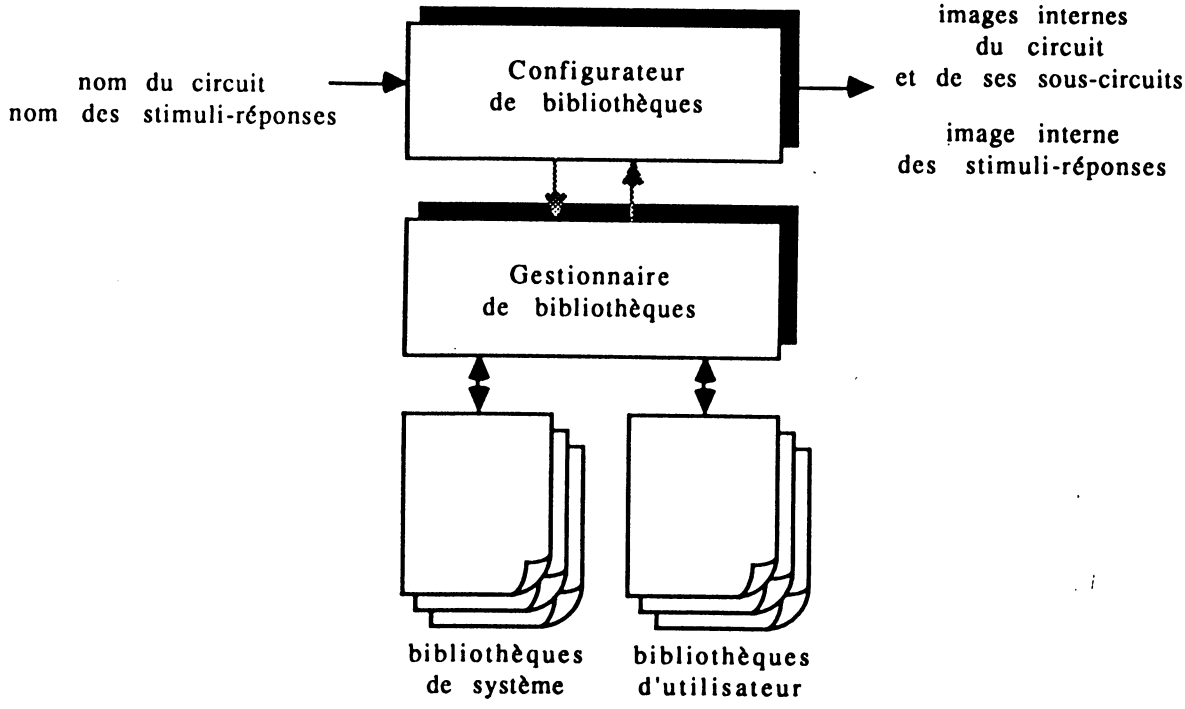
• Un circuit peut avoir plusieurs versions d'images internes définies dans différentes bibliothèques. Le configurateur respecte l'ordre de priorité des bibliothèques afin d'assurer la bonne sélection de la version désirée.

4.4.2. Configurateur de bibliothèques

Les images internes peuvent être stockées dans :

- la base de données de l'environnement de LL3T;
- les bibliothèques de système gérées automatiquement par LL3T;
- les bibliothèques d'utilisateur créées pour différentes applications.

Le rôle principal du configurateur de bibliothèques est de gérer statiquement les images internes dans les bibliothèques, et de charger dynamiquement des images internes, lors d'une session de simulation, dans la base de données de l'environnement de simulation.



Configurateur de bibliothèques

Les bibliothèques contiennent les images internes créées par les compilateurs. Il y a trois sortes d'images internes qui peuvent être inclues dans les bibliothèques :

- les modèles de circuits;
- les modèles de stimuli-réponses;
- les textes.

Lors d'une session de simulation, le configurateur de bibliothèques détermine les bibliothèques concernées et charge les images internes à utiliser dans la base de données de l'environnement de simulation.

◊ Bibliothèques

Deux noms sont associés à chaque bibliothèque :

- un nom logique pour l'identification de la bibliothèque;
- un nom physique pour la localisation physique de la bibliothèque.

Une bibliothèque est constituée de :

- un fichier d'images qui contient les images internes;
- un fichier de références qui contient les interfaces et les références externes des images.

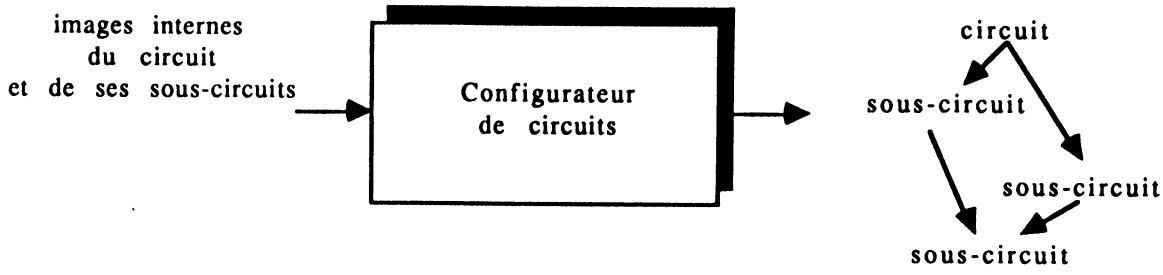
Des mesures ont été prises pour améliorer l'efficacité de la gestion de bibliothèques :

- Pendant la configuration des bibliothèques, seuls les fichiers de références des bibliothèques sont consultés pour établir dynamiquement la dépendance entre les modèles de circuits et de stimuli-réponses.
- Selon les relations de dépendance, un ensemble de circuits concernés par la session de simulation est sélectionné.
- Seules les images internes de cet ensemble de circuits seront prises en compte et chargées dans l'environnement de simulation.

4.4.3. Configurateur de circuits

Le configurateur de bibliothèques ne détermine que l'ensemble des

circuits concernés par la simulation. Le configurateur de circuits construit réellement la hiérarchie des circuits qui exprime la dépendance et les relations d'englobement entre les circuits. La construction de la hiérarchie commence par le circuit principal et se termine sur les sous-circuits élémentaires.



Configurateur de circuits

En même temps que cette construction, sont effectuées les opérations suivantes :

• La vérification de l'interface entre les circuits englobants et leurs sous-circuits;

• la propagation des attributs dynamiques des circuits à leurs sous-circuits;

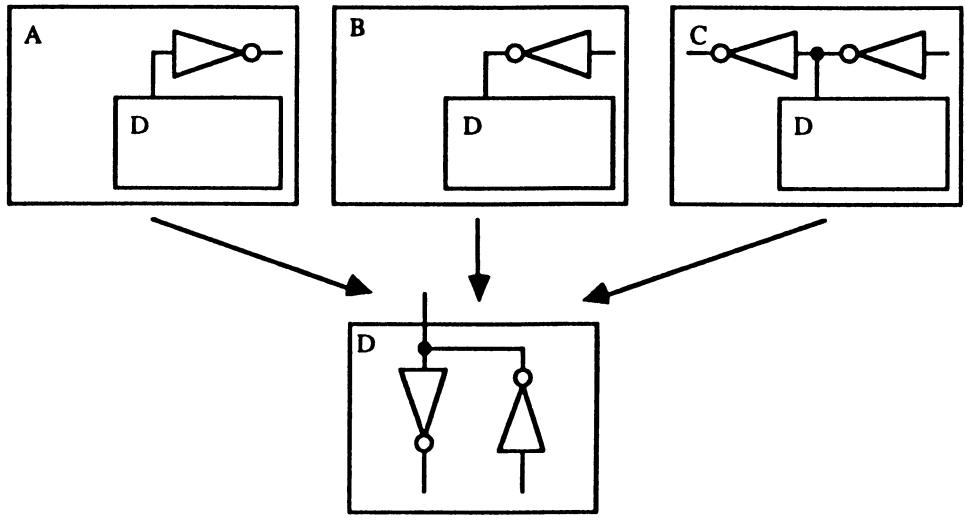
La vérification de l'interface entre les circuits ne peut être entreprise qu'au moment de la configuration car aucune relation statique entre les circuits n'est établie lors de la compilation. La vérification contient deux aspects :

• La liste des paramètres effectifs doit être compatible avec celle des paramètres formels.

• L'équipotentielle effective de chaque port de connexion doit correspondre à l'équipotentielle formelle. Leurs types peuvent être différents, mais ne doivent avoir aucun conflit.

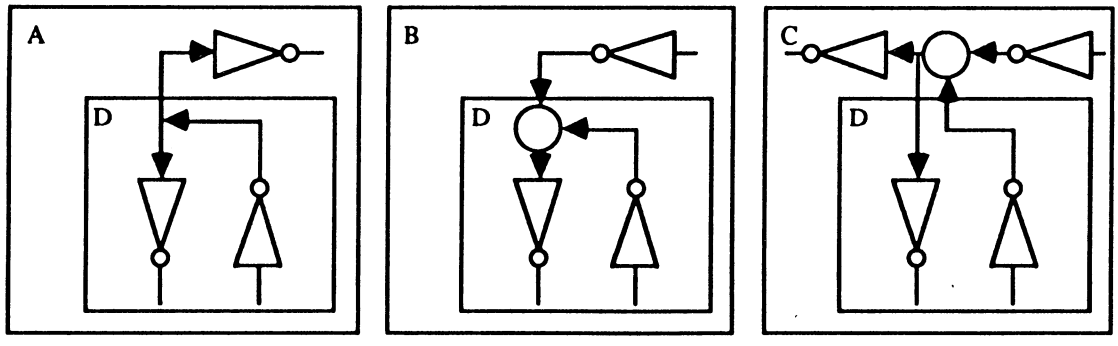
La propagation des attributs a pour but de déterminer les propriétés dynamiques du sous-circuit commun à un ensemble de circuits qui l'englobent, pour que le sous-circuit commun soit compatible avec tous les circuits qui l'utilisent.

Prenons un exemple, trois circuits A, B et C partagent le même modèle de sous-circuit D :



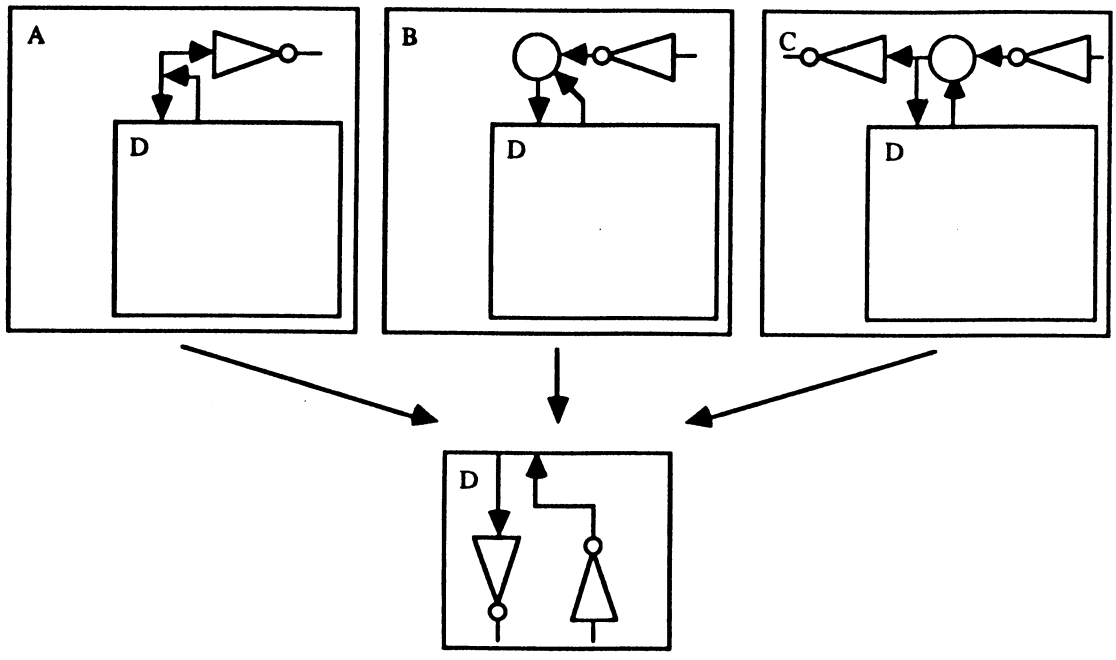
Lors de la configuration, les circuits A, B et C propagent leurs attributs au sous-circuit D, pour avoir un sous-circuit qui leur convient.

Individuellement, chacun des circuits A, B et C exige, pour le même sous-circuit D, une structure topologique distincte :

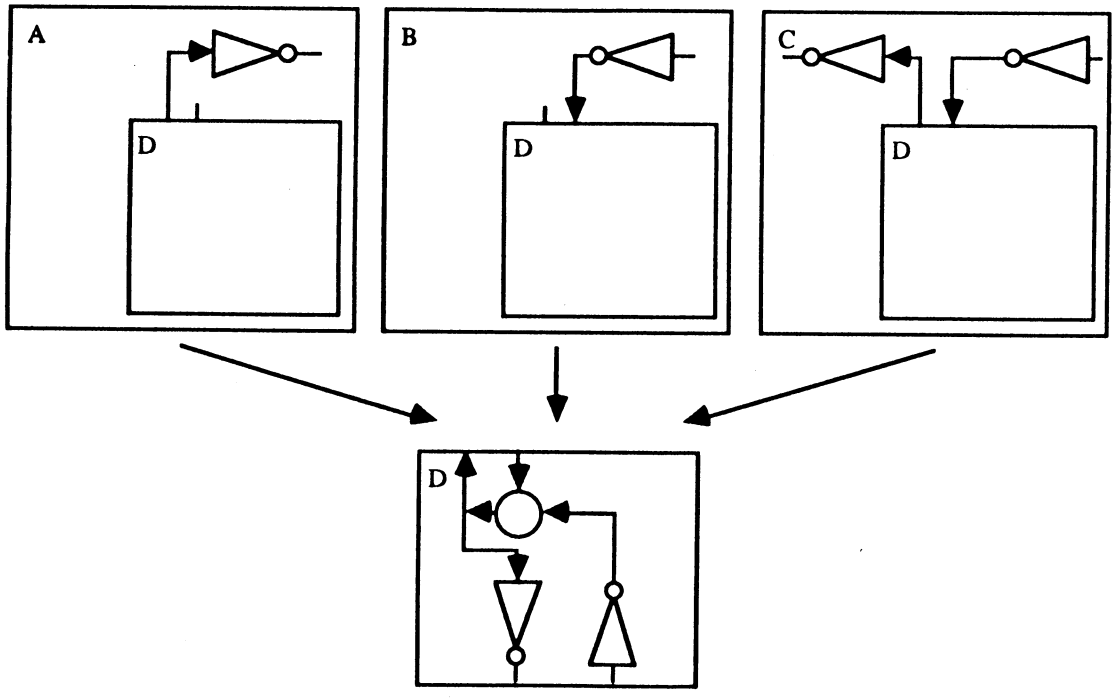


Il est donc nécessaire que le configurateur propage les attributs de A, B et C au sous-circuit D pour que D prenne une topologique convenable à tous les circuits qui l'englobent.

Alors la topologie idéale du sous-circuit D, compatible avec tous les trois circuits englobants, doit être comme suit :



ou bien :



Il convient d'en conclure que les attributs statiques et dynamiques d'un circuit sont déterminés par :

- la compilation : le compilateur déduit selon un ensemble de règles sémantiques, les attributs statiques du circuit;
- la configuration : le configurateur propage les attributs des circuits

vers leurs sous-circuits. Les attributs dynamiques des sous-circuits peuvent être déterminés complètement (les sous-circuits élémentaires) ou partiellement (les sous-circuits qui contiennent à leur tour des sous-circuits) lors de la configuration.

• l'édition de liens : Lorsque les attributs dynamiques d'un sous-circuit sont déterminés, ils sont propagés vers les circuits englobants par l'éditeur de liens de circuits pour déterminer les attributs dynamiques des circuits englobants.

Dynamiquement, une image interne de circuit peut avoir différents attributs selon différentes configurations. Ces attributs dynamiques sont :

• les attributs dynamiques des équipotentielles : Par exemple, une équipotentielle peut avoir différentes directions.

• les attributs dynamiques des éléments : Par exemple, les entrées et les sorties peuvent être modifiées.

• les attributs dynamiques de l'interface du circuit : Par exemple une connexion bidirectionnelle peut être substituée par deux ou plusieurs connexions unidirectionnelles.

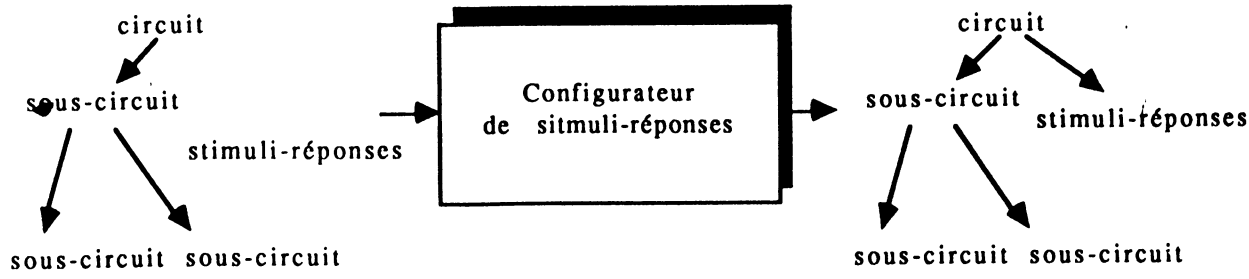
• les attributs dynamiques du corps du circuit : Par exemple, des éléments ou des équipotentielles complémentaires peuvent être ajoutés dans le circuit.

4.4.4 Configurateur de stimuli-réponses

Le configurateur de stimuli-réponses réalise les fonctions suivantes :

• la vérification de l'interface et des interconnexions entre le circuit et les stimuli-réponses;

• la propagation des attributs de l'image interne du circuit à celle des stimuli-réponses.



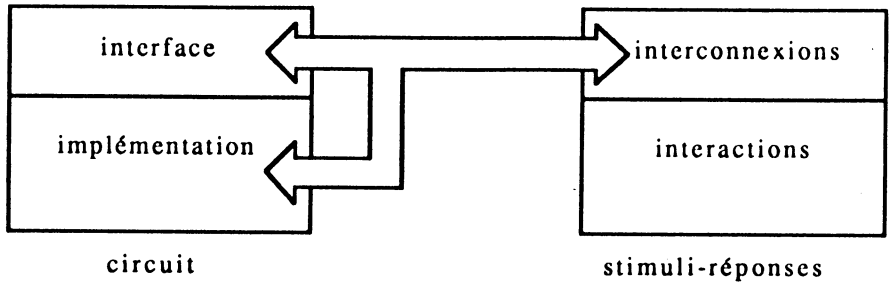
Configurateur de stimuli-réponses

La vérification de l'interface et des interconnexions permet de :

• localiser les points de stimulus et de réponse dans le circuit : Cette localisation consiste à identifier dans le circuit les équipotentielles qui correspondent à ces points de stimulus et de réponse;

• examiner la compatibilité entre les points de stimulus et de réponse et les équipotentielles du circuit : La compatibilité dépend du type de l'équipotentielle et de la direction dans laquelle le signal circule.

Les interconnexions entre le circuit et les stimuli-réponses ne sont pas nécessairement définies par l'interface du circuit. Les points de stimulus et de réponse peuvent être n'importe quelle équipotentielle du circuit à simuler. Cela permet d'étudier l'évolution des états de n'importe quelle équipotentielle interne du circuit durant la simulation.



Interconnexions entre le circuit et les stimuli-réponses

Les interconnexions influent sur les attributs du circuit et des stimuli-réponses. Les attributs dynamiques des points de stimulus et de réponse dépendent de ceux des équipotentielles du circuit, en particulier :

• le type de l'équipotentielle qui correspond au point de stimulus ou celui de réponse : par exemple une équipotentielle unidirectionnelle ou bidirectionnelle;

• la fonction de l'équipotentielle : par exemple, une fonction câblée;

• la taille dynamique du point : par exemple, un point de réponse qui correspond à un vecteur d'équipotentielles dont la taille est paramétrée par le circuit.

4.5. Editeurs de liens

4.5.1. Editeur de liens de circuits

Le rôle de l'éditeur de liens de circuits consiste à déterminer les attributs dynamiques des sous-circuits et à les propager aux circuits englobants. Pour un sous-circuit, les attributs dynamiques de tous les circuits englobants lui ont été propagés pendant la phase de configuration. Donc l'édition de liens détermine d'une manière décisive tous les attributs dynamiques du sous-circuit et propage ces attributs vers les circuits englobants.

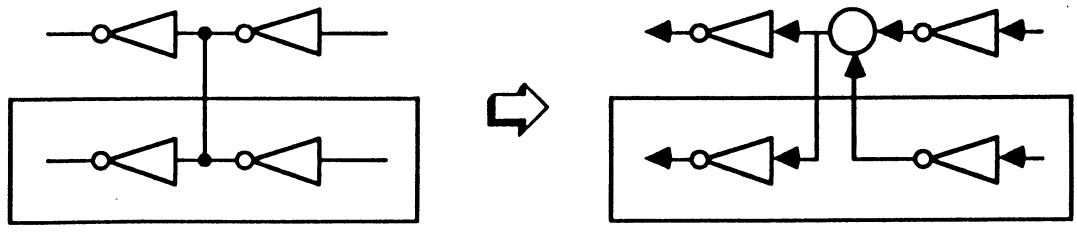
L'éditeur de liens de circuits s'opère sur la hiérarchie des circuits construite par les configurateurs. L'édition de liens commence par les sous-circuits et se termine sur le circuit principal.

Les attributs, qui sont propagés à travers chaque port de connexion de l'interface du sous-circuit, portent les informations suivantes :

- le type du port de connexion;
- le nombre d'équipotentielles qui correspondent au port;
- la fonction associée à ces équipotentielles;
- les éléments complémentaires qui doivent être ajoutés dans le circuit englobant.

Prenons un exemple simple, lorsqu'une équipotentielle connectée au circuit englobant est définie dans le sous-circuit comme la masse ou qu'elle est affectée d'une fonction câblée, le type masse ou la fonction câblée doivent être propagés à l'équipotentielle correspondante du circuit englobant.

Dans l'exemple suivant, après l'édition de liens, le port de connexion entre le sous-circuit et le circuit englobant est représenté par deux équipotentielles unidirectionnelles et un élément complémentaire :



4.5.2. Editeur de liens de stimuli-réponses

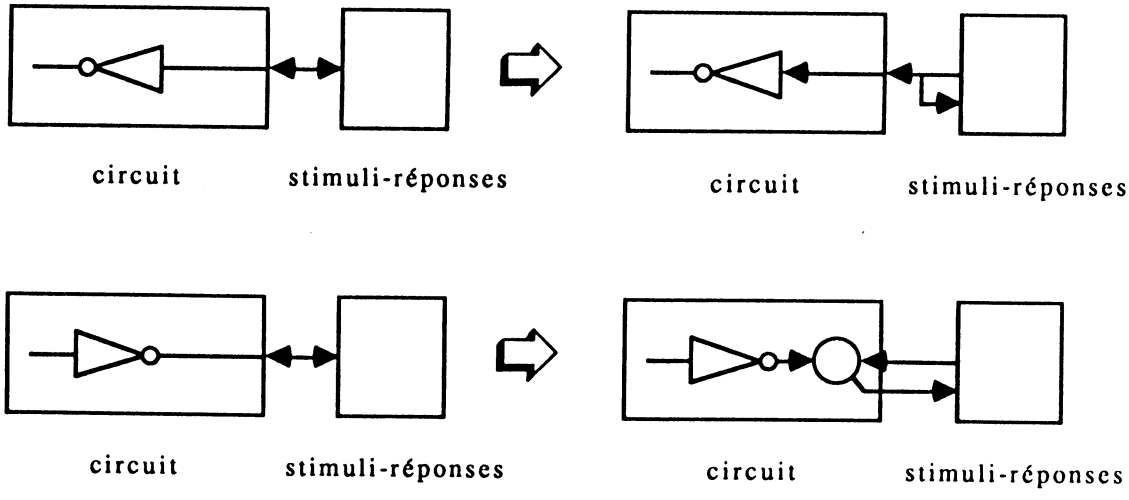
Le rôle de l'éditeur de liens de stimuli-réponses consiste à déterminer les attributs dynamiques des stimuli-réponses et à les propager au circuit.

Avant l'exécution de l'édition de liens de stimuli-réponses, le configurateur de stimuli-réponses a établi les interconnexions entre le circuit et les stimuli-réponses. Les attributs dynamiques du circuit ont été propagés aux points de stimulus et de réponse.

L'édition de liens détermine les attributs dynamiques des points de stimulus et de réponse et les propage ensuite au circuit. Les attributs définissent les informations suivantes :

- le type de l'équipotentielle;
- le nombre d'équipotentielles correspondantes;
- l'éventuelle modification de la structure topologique du circuit, due à la présence des stimuli-réponses.

Les exemples suivants montrent qu'un port bidirectionnel peut correspondre à plusieurs d'équipotentielles et que ce port bidirectionnel peut nécessiter l'ajout d'un élément complémentaire dans le circuit :



4.6. Générateur de code

4.6.1. Langage du simulateur

Le noyau du simulateur LL3T est un module autonome qui travaille sur un langage intermédiaire. C'est un langage codé en entiers 32 bits.

Pour le simulateur, l'utilisation d'un langage spécifique procure des avantages :

• La sémantique du langage peut être très proche des propriétés internes du simulateur. La structure de données du langage représente implicitement l'architecture physique du simulateur. L'utilisation du simulateur peut être rendu optimale.

• Le simulateur peut être muni de différents compilateurs qui ont tous le langage du simulateur comme le langage cible. Différents environnements de simulation peuvent être construits qui partagent le même noyau de simulation.

• Du côté de l'utilisateur, la dépendance vis-à-vis de la machine peut être réduite au minimum. Les circuits conçus par l'utilisateur sont compilés d'une manière générale et mémorisés dans la base de données ou dans les bibliothèques externes. Lors d'une session de simulation, différents codes peuvent être générés en fonction de différentes configurations physiques du simulateur, et de différentes conditions de simulation.

• Le code est un fichier exécutable qui représente une session de simulation. La simulation peut se répéter sous différents modes.

• Le langage est une interface située entre le simulateur et son environnement. Cette interface permet de développer LL3T d'une manière modulaire et d'optimiser différents logiciels de LL3T.

Le code est composé de deux fichiers :

• le fichier qui contient le code lui-même;
• le fichier qui contient tous les identificateurs référencés par le code.

Les informations décrites dans le code sont :

• les informations générales de la simulation : le mode de simulation, les conditions d'initialisation, etc.

• les informations pour le DMT : les équipotentielles à visualiser et

leur format, les points de réponse, etc.

• les informations pour les ECTs : les éléments et leurs interconnexions distribués sur chaque ECT, etc.

• les informations pour les EMTs : l'initialisation des équipotentielles, et les stimuli pour chaque unité de simulation, etc.

Le fichier qui contient les identificateurs permet de référencer par nom tous les objets définis dans le circuit et ses sous-circuits.

4.6.2. Génération des circuits

Le circuit et ses sous-circuits sont répartis sur les ECTs. Chaque ECT est affecté d'un ensemble d'éléments. Ces éléments peuvent être :

• des primitives dont le comportement est pré-défini;
• des éléments fonctionnels dont le comportement est défini par une liste d'instructions;
• des définitions de sous-circuits;
• des instances de sous-circuits.

Une primitive représente une porte logique ou un élément similaire. Les informations d'une telle porte sont :

• le type;
• la liste de forces;
• la liste de retards;
• la liste d'entrées;
• la liste de sorties.

Un élément fonctionnel se compose des informations suivantes :

• le type;
• la liste de forces;
• la liste de retards;
• la liste d'entrées;
• la liste de sorties;
• la liste d'instructions.

Le fonctionnement de l'élément fonctionnel est défini par les règles suivantes :

• Le déclenchement de l'élément est déterminé par le changement d'état sur les entrées.

• La fonction de l'élément est définie par les instructions;

• Si les états sur les sorties sont changés, les nouveaux événements de sortie sont créés, dont les forces et les retards dépendent de la liste de forces et de celle de retards.

Une définition de sous-circuit est constituée d'un ensemble d'éléments qui peuvent être :

- les primitives pré-définies;
- les éléments fonctionnels;
- les instances de sous-circuits.

Une instance de sous-circuit contient :

- l'identification du circuit;
- la liste de paramètres effectifs;
- la liste de connexions effectives.

4.6.3. Génération de stimuli-réponses

Les stimuli-réponses peuvent être réalisés d'une des façons suivantes :

- statiquement : par la création d'une liste de stimuli;
- dynamiquement : par la création d'un élément fonctionnel.

La liste de stimuli définit les stimuli d'une manière statique. Dans cette liste, un stimulus contient les informations suivantes :

- le temps;
- le nouvel état;
- l'unité de simulation;
- l'identification de l'équipotentielle.

En effet, ces stimuli sont considérés comme des événements et insérés dans les échéanciers des unités de simulation, lors de l'initialisation de la simulation.

Les stimuli-réponses peuvent être réalisés dynamiquement par l'utilisation des éléments fonctionnels. Le générateur de code crée, à la place des stimuli-réponses, des éléments fonctionnels. Ces éléments engendrent, pendant la simulation, des signaux conformes aux chronogrammes de stimuli-réponses.

Dans les cas suivants, le générateur de code crée des éléments fonctionnels pour réaliser les stimuli-réponses :

- Il existe des structures de contrôle complexes.
- Il existe des mécanismes d'itération.
- Il existe des blocs imbriqués.
- Des activités concurrentes sont définies.
- Des opérations spéciales sont utilisées.

Conclusion

LL3T est un accélérateur de simulation à base de transputers. L'ensemble de logiciels développés constitue un environnement intégré de simulation, qui permet non seulement la simulation, mais également les pré-traitements, les post-traitements, et la gestion de données.

Le simulateur est implémenté sur un réseau en anneau de 1 à 128 unités de simulation, chacune constituée de trois transputers. L'unité de simulation possède son échéancier local. L'ensemble des unités de simulation est synchronisé à chaque étape de simulation.

Les compilateurs permettent l'utilisation des langages Hilo-3 pour modéliser les circuits, décrire les stimuli-réponses, et définir les conditions de simulation.

La base de données et les bibliothèques, permettant de gérer les images internes créées par les compilateurs, rendent l'environnement de simulation plus opérationnel.

Les configurateurs établissent les relations de dépendance entre les circuits lors d'une session de simulation, et construisent la hiérarchie des circuits concernés par la simulation et des stimuli-réponses à appliquer dans la simulation. Cette configuration dynamique assure l'indépendance de la compilation de circuits.

Les attributs dynamiques des circuits et des stimuli-réponses qui ne peuvent être déterminés lors de la compilation, sont déduits par les configurateurs et les éditeurs de liens.

Le générateur de code crée un code exécutable, en langage du simulateur, à partir des circuits et des stimuli-réponses ainsi configurés et liés. Le langage du simulateur reflète l'architecture du simulateur et permet d'utiliser au maximum la puissance du simulateur.

Chapitre 5

Algorithmes de LL3T : Réalisation

Plan du chapitre 5

Introduction

Ce chapitre traite des algorithmes développés et les stratégies utilisées dans la réalisation de l'ensemble des logiciels de LL3T :

- les algorithmes développés autour de la modélisation des éléments qui ne sont pas des portes logiques, tels que les modèles de transistors et fonctionnels;
- les algorithmes utilisés pour la modélisation des équipotentielles;
- les algorithmes de simulation de pannes.

Le chapitre commence par une discussion sur les problèmes liés au parallélisme. Pour la simulation normale, il convient de partitionner le circuit afin qu'il soit simulé de façon parallèle par les unités de simulation. Tandis que pour la simulation de pannes, une autre stratégie est utilisée qui consiste à distribuer les pannes sur les unités de simulation. Chaque unité simulent chacune le circuit entier avec les pannes qui lui sont allouées. Cette stratégie est envisageable, car une simulation de pannes est très coûteuse en terme de temps de calcul et de mémoire consommée.

Dans le simulateur, les transistors sont considérés comme des éléments spéciaux et traités de manières différentes des portes logiques. Toutefois, l'évaluation de ces modèles de transistors doit rester compatible avec l'algorithme de l'échéancier qui constitue la base du simulateur.

La modélisation fonctionnelle consiste à associer à chaque élément fonctionnel une liste d'instructions qui définit la fonction de celui-ci. Du côté de la simulation, l'essentiel est de réaliser une machine virtuelle qui exécute les instructions pour évaluer la fonction de l'élément. Du côté de la compilation, l'essentiel est de développer des algorithmes qui permettent de créer, pour chaque élément fonctionnel, une liste d'instructions, capables de réaliser n'importe quels types de modèles fonctionnels. Les solutions des trois types de modèles fonctionnels suivants seront présentées dans le chapitre :

- les expressions arithmétiques et logiques;
- les registres virtuels;
- les expressions d'événements.

Pour la compilation, la configuration, et l'édition de liens, la complexité

de la modélisation des équipotentielles se trouve également dans l'algorithme de typage des équipotentielles. Puisque d'une part les modèles d'équipotentielles définis dans les langages Hilo-3 sont riches, et d'autre part, les interconnexions entre les circuits exigent la compatibilité des types des ports de connexion, et la propagation des attributs dynamiques des équipotentielles lors de la configuration et de l'édition de liens.

5.1. Problèmes liés au parallélisme

5.1.1. Partitionnement du circuit

L'introduction du parallélisme dans la simulation normale consiste à partitionner le circuit à simuler afin que celui-ci soit simulé en parallèle par les unités de simulation.

Ce partitionnement est réalisé avec les hypothèses suivantes :

- Les éléments du circuit sont distribués sur les ECTs.
- Les équipotentielles sont considérées soit locales à un ECT, soit globales définies dans plusieurs ECTs.

Les équipotentielles globales peuvent être classées en plusieurs catégories :

- L'équipotentielle est connectée à une sortie d'élément.
- L'équipotentielle est connectée à plusieurs sorties d'éléments.
- L'équipotentielle est connectée à une ou plusieurs sorties d'éléments et à un point de stimulus.
- L'équipotentielle n'est connectée à aucune sortie d'élément.

Dans la première catégorie, une équipotentielle représente une connexion unidirectionnelle, c'est-à-dire :

- Il n'existe dans le circuit qu'une sortie d'élément s_i connectée à cette équipotentielle.
- Il existe dans le circuit une ou des entrées d'éléments $e_{j1} .. e_{jN}$ connectées à cette équipotentielle.

Donc cette équipotentielle est prise en compte par :

- l'ECT T_i où $s_i \in T_i$;
- les ECTs $T_{j1} .. T_{jM}$ où $T_{j1} .. T_{jM} \in \{T_j \mid e_{jk} \in T_j, k = 1 .. N\}$;
- le DMT T_0 si l'équipotentielle est à visualiser, ou qu'elle est un point de réponse.

Par conséquent, lors du partitionnement du circuit, l'équipotentielle est représentée par la configuration suivante :

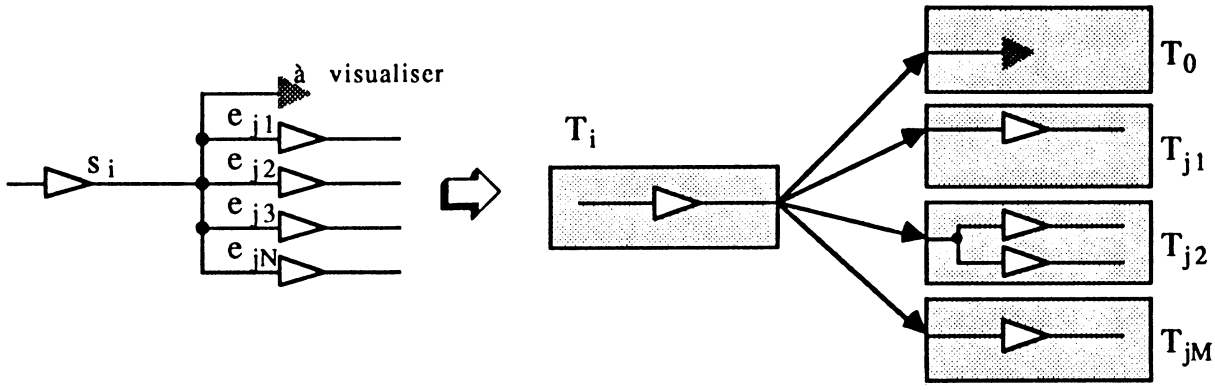
• L'équipotentielle est définie respectivement dans les ECTs T_i, T_{j1} .. T_{jM}, et éventuellement dans le DMT T_0.

• L'élément dont la sortie s_i est connectée à l'équipotentielle est défini dans l'ECT T_i.

• La sortie de l'élément s_i est définie soit par M+1 interconnexions si elle est visualisée ou liée à un point de réponse, soit par M interconnexions.

• Les retards associés à la sortie s_i sont partagés par ces M+1 ou M interconnexions.

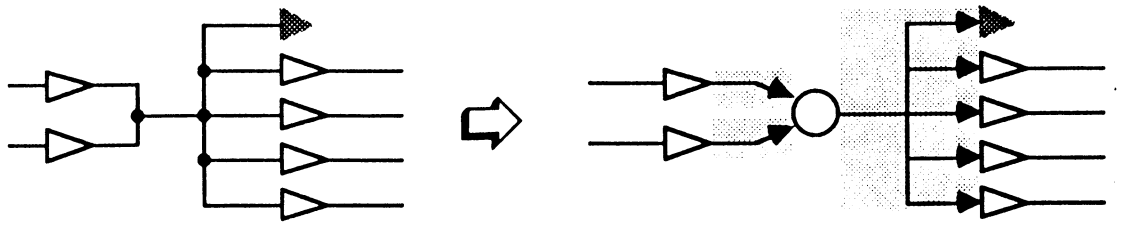
• Lors de l'évaluation de l'éléments, M+1 ou M événements sont créés et transmis aux unités de simulation correspondantes, si la sortie s_i change d'état.



Dans la deuxième catégorie, une équipotentielle est connectée à plusieurs sorties d'éléments.

Dans ce cas, un élément complémentaire doit être ajouté dans le circuit à simuler.

L'équipotentielle originale est représentée par N+1 équipotentielles, N étant le nombre de sorties d'éléments connectées à l'équipotentielle :



L'élément ajouté joue les rôles suivants :

• Il réalise les connexions unidirectionnelles de l'équipotentielle

originale avec les sorties d'éléments.

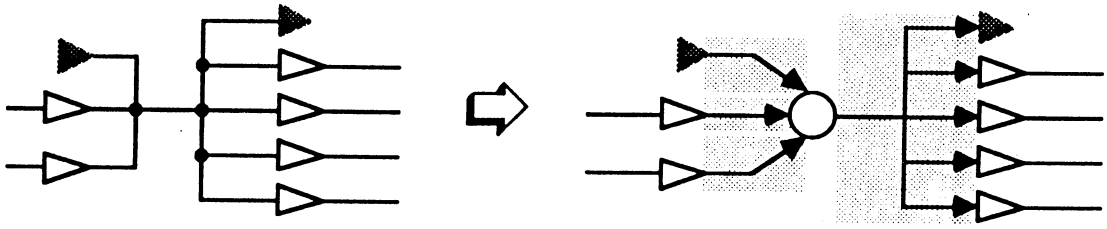
• Il détermine l'état de l'équipotentielle originale en fonction des états des N sorties d'éléments.

• Il réalise la connexion unidirectionnelle de l'équipotentielle originale avec les entrées d'éléments.

L'équipotentielle originale étant représentée par N+1 équipotentielles unidirectionnelles, chacune d'entre elles peut être traitée comme une équipotentielle de la première catégorie :

• Elles ne sont connectées qu'à une sortie d'élément.
• Elles sont connectées à une ou plusieurs entrées d'éléments.

Dans la troisième catégorie, une équipotentielle est connectée à une ou plusieurs sorties d'éléments et à un point de stimulus. Dans ce cas le point de stimulus est équivalent à une sortie d'élément :



Dans la quatrième catégorie, une équipotentielle n'a pas de sortie d'élément connectée.

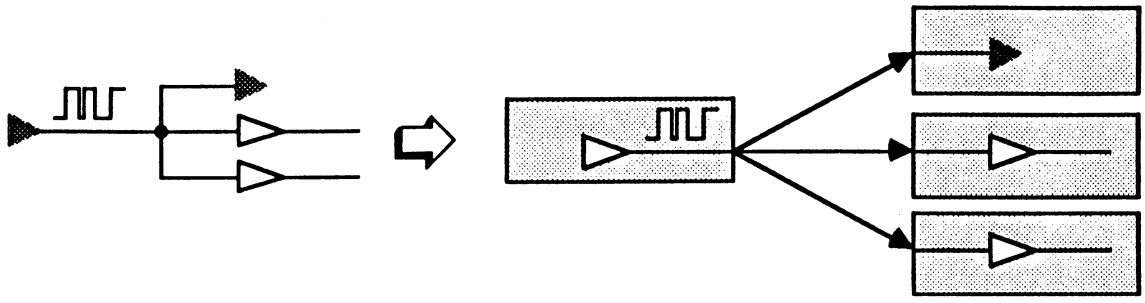
Si cette équipotentielle est connectée à un point de stimulus, alors le point de stimulus peut être considéré comme une sortie d'élément connectée à l'équipotentielle. L'équipotentielle est traitée comme la première catégorie.

Plus précisément, le point de stimulus peut être associé à des stimuli statiques ou dynamiques :

• Les stimuli statiques sont des signaux ordonnés chronologiquement, créés par le pré-traitement.
• Les stimuli dynamiques sont représentés par un élément fonctionnel qui génère des signaux pendant la simulation.

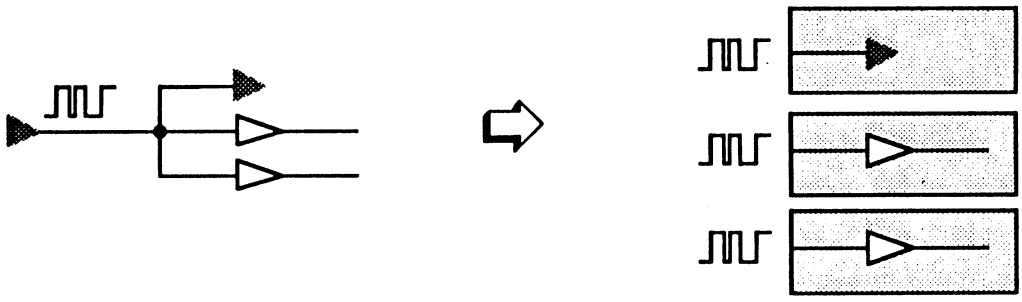
Si les stimuli sont sous forme d'élément fonctionnel, alors le point de stimulus est considéré comme une sortie de porte qui génère des signaux sur

l'équipotentielle :

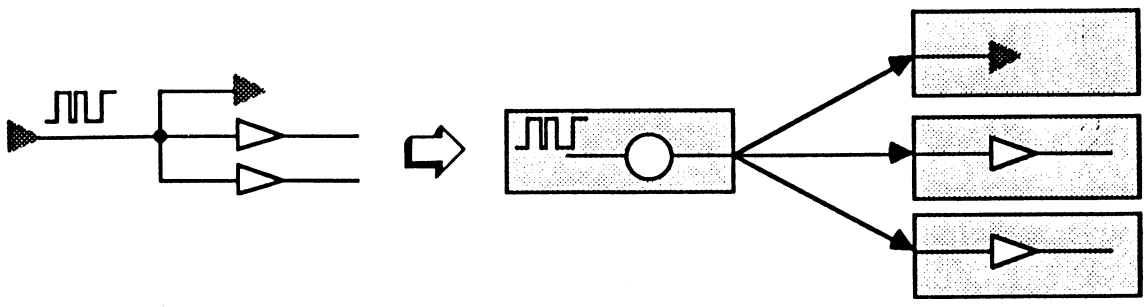


Si les stimuli sont statiques, ils seront représentés par un ensemble d'événements. Dans ce cas, deux solutions sont possibles.

Une solution consiste à utiliser une équipotentielle locale dans chaque ECT concerné (même le DMT si nécessaire) et à répliquer les événements statiquement qui seront distribués, au moment de l'initialisation de la simulation, dans les échéanciers correspondants.



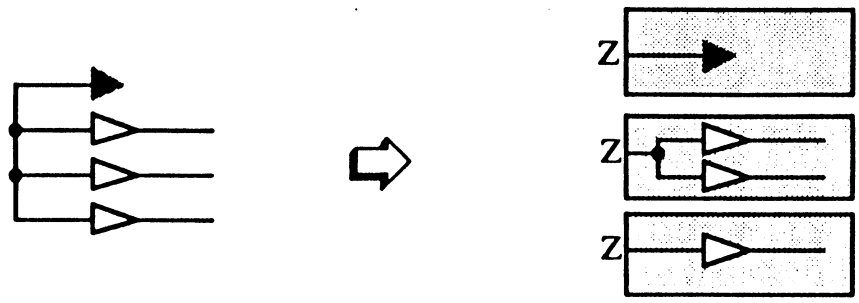
Une autre solution est de créer un élément complémentaire. L'élément peut se situer dans n'importe quel ECT. L'ensemble des événements qui représentent les stimuli statiques seront chargés dans l'échéancier correspondant à l'ECT contenant l'élément complémentaire. Pendant la simulation, les signaux de stimuli seront transmis aux autres ECTs ou DMT dynamiquement, à travers les connexions de sortie de l'élément.



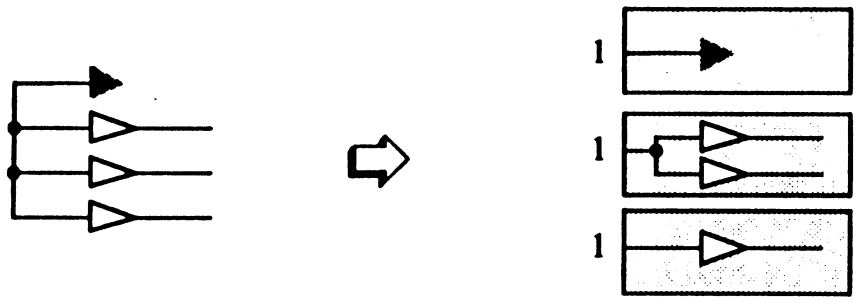
Si aucun stimulus n'est associé à l'équipotentielle, l'état de cette équipotentielle reste à Z :

• Supposé que N entrées d'éléments e_{ji} .. e_{jN} soient connectées à l'équipotentielle et que ces éléments se situent dans M ECTs T_{j1} .. $T_{jM} \in \{T_j \mid e_{jk} \in T_j, k = 1 .. N\}$, alors cette équipotentielle est représentée par la valeur Z dans T_{j1} .. T_{jM}.

• Si l'équipotentielle est visualisée ou qu'elle est connectée à un point de réponse, alors elle sera représentée par la valeur Z dans le DMT :



Si c'est une équipotentielle à laquelle est associée une valeur pré-définie, alors cette valeur sera utilisée à la place de Z :



En général, la visualisation d'une équipotentielle peut être considérée comme s'il y avait dans le DMT une entrée d'élément connectée à l'équipotentielle.

Le point de stimulus est une source de signal provenant de l'extérieur du circuit. Il est considéré comme une sortie d'élément qui génère des signaux sur l'équipotentielle.

Le point de réponse est une sonde qui échantillonne des signaux et les compare avec les valeurs prévues. Les opérations de comparaison sont effectuées par le DMT. Donc le point de réponse peut être considéré comme l'entrée d'un élément situé dans le DMT.

En conclusion, l'introduction du parallélisme dans la simulation nécessite dans certains cas la décomposition d'une équipotentielle en plusieurs équipotentielles, chacune de celles-ci étant définie dans un ECT.

5.1.2. Equilibrage du partitionnement

Le partitionnement de circuit est un problème critique pour les simulateurs parallèles. Il influence directement l'efficacité de la simulation. Il est souhaitable que la répartition du circuit soit équilibrée.

Différentes méthodes sont utilisées pour assurer l'équilibrage du partitionnement. Cet équilibrage peut être :

- statique au niveau de l'espace consommé par chaque unité de simulation;
- dynamique au niveau du temps de calcul de chaque unité de simulation.

L'équilibrage statique peut être réalisé par un pré-traitement. Tous les ECTs doivent avoir une partition du circuit d'une taille identique ou approximative. Cet équilibrage contient deux aspects :

- Le nombre d'éléments distribués sur chaque ECT doit être proche de la moyenne.
- Le nombre d'équipotentielles définies sur chaque ECT doit être proche de la moyenne.

Le premier problème semble facile à résoudre. Il suffit de distribuer le même nombre d'éléments sur les ECTs. Par contre, quelques considérations doivent être prises, car la taille d'un élément dépend des facteurs suivants :

- l'entrance de l'élément N_e;
- la sortance de l'élément N_s;
- le nombre des contraintes temporelles de l'élément N_c;
- le nombre d'instructions de l'élément N_i s'il est fonctionnel;
- le nombre des variables internes de l'élément N_v.

Il est convenable d'associer un poids à l'élément :

$$P_e = f(N_e, N_s, N_c, N_i, N_v)$$

Et pour chaque ECT, $\sum P_e$ doit être proche de la moyenne.

Les équipotentielles doivent être aussi distribuées de manière équilibrée sur les ECTs. Cependant, il est impossible d'équilibrer les éléments et les équipotentielles à la fois. Car le partitionnement des éléments et des équipotentielles doit respecter les propriétés topologiques du circuit.

Quel que soit le partitionnement du circuit, le nombre total des éléments du circuit reste inchangé. Pourtant le nombre total des équipotentielles est dépendant du partitionnement du circuit. Une équipotentielle créée par le générateur de code est toujours une connexion unidirectionnelle entre une sortie d'élément et plusieurs entrées d'éléments. Si les éléments reliés à cette équipotentielle se situent sur différents ECTs, celle-ci doit être représentée par autant d'équipotentielles que les ECTs concernés. Il est donc souhaitable de placer les éléments reliés par une équipotentielle sur un même ECT. Cette mesure diminue d'une part le nombre d'équipotentielles, d'autre part la sortance des éléments.

L'effet de l'équilibrage dynamique se manifeste pendant la simulation. Il s'agit principalement du temps consommé pour la synchronisation et les communications entre les ECTs :

- A chaque étape de simulation, les ECTs qui terminent l'évaluation sont obligés d'attendre les ECTs qui sont surchargés.
- Si les éléments ont une sortie connectée aux équipotentielles globales, ils doivent engendrer des événements sous forme de messages de simulation qui nécessitent des communications entre les ECTs.

L'équilibrage de synchronisation exige que les ECTs consomment une même durée de temps de calcul à chaque étape de simulation, pour éviter qu'il y ait des ECTs qui soient dans un état d'attente. Or cet équilibrage est très difficile à achever :

- Le temps de l'évaluation des éléments varie des uns aux autres : L'élément fonctionnel contient des instructions. Selon la complexité de sa fonction, cet élément peut avoir quelques ou une centaines d'instructions, où des branchements et des itérations peuvent être utilisés. L'évaluation des transistors est également itérative et se termine lorsque les états des

transistors deviennent stables.

• Les activités du circuit sont dynamiques : A un moment donné, les éléments d'un ECT peuvent être tous inactifs pendant que ceux d'un autre ECT sont très actifs. A un autre moment, d'autres éléments deviennent à leur tour actifs.

Les communications doivent être réduites au minimum, notamment les messages de simulation qui représentent les événements transmis entre les unités de simulation. Une mesure à entreprendre est d'éviter des équipotentielles globales qui traversent plusieurs ECTs, pour que les événements soient locaux. Si un élément appartient à un ECT, il est souhaitable d'inclure ses successeurs dans le même ECT.

En plus, d'autres problèmes doivent être pris en compte :

• La distance logique d'un message est non négligeable. Elle représente le nombre d'unités de simulation qu'un message traverse. Une transmission de message, qui traverse plusieurs unités de simulation jouant le rôle de relais, coûte plus cher qu'une simple transmission entre deux unités de simulation voisines. Pour les événements locaux, la distance logique est égale à 0.

• Il y a des équipotentielles très actives ou peu actives. Plus l'équipotentielle est active, plus la distance logique doit être réduite. Par contre, l'activité d'une équipotentielle ne peut être déterminée préablement. En conséquence, il est difficile d'obtenir une solution satisfaisante qui permet de minimiser en priorité la distance logique des équipotentielles actives.

D'autre part, l'algorithme de partitionnement ne doit pas être trop complexe à exploiter, qui risque de coûter cher lorsque la taille du circuit est importante. L'ordre de grandeur de l'algorithme doit être limité à O (n). Un algorithme qui effectue des optimisations lors du partitionnement peut être de l'ordre de O (n^2), puisque le traitement de chaque élément doit tenir compte de ses éléments voisins directs ou indirects, ou même de l'ensemble des éléments du circuit.

5.1.3. Méthodes de partitionnement

Les méthodes les plus simples sont les partitionnements aléatoire et linéaire :

• Le partitionnement aléatoire consiste à distribuer les éléments sur les ECTs d'une manière aléatoire.

• Le partitionnement linéaire consiste à affecter les N/M premiers éléments sur le premier ECT, les N/M deuxièmes sur le deuxième ECT, et ainsi de suite, N étant le nombre total des éléments du circuit, M étant le nombre d'ECTs dans le simulateur.

Ces deux méthodes nécessitent peu de temps de calcul. Par contre, le partitionnement ne prend pas en compte l'équilibrage dynamique du circuit.

Deux autres méthodes sont utilisées qui consiste à grouper les éléments fortement couplés dans un même ECT. Ces deux méthodes sont basées sur l'analyse de la connectivité des éléments :

• Une méthode consiste à intégrer, lorsqu'un élément est placé sur un ECT, les éléments connectés à ses sorties dans le même ECT. Pour chaque ECT, on sélectionne un ou un ensemble d'éléments initiaux. On intègre, dans cet ensemble d'éléments, les éléments connectés aux sorties de ces éléments initiaux. Et on répète le même processus sur le nouvel ensemble, jusqu'à ce que le nombre d'éléments prévu soit atteint. Cette méthode a pour objectif de réduire le nombre d'équipotentielles globales, en sorte que les communications entre les ECTs soient minimales.

• Une autre méthode consiste à intégrer, lorsqu'un élément est placé sur un ECT, les éléments connectés à ses entrées dans le même ECT. Cette méthode procède de la même manière que la précédente, mais dans un sens inverse. Le but est également de minimiser les communications entre les ECTs.

Ces deux méthodes améliorent sensiblement l'équilibrage dynamique du partitionnement du circuit. L'inconvénient est que le temps de calcul du partitionnement est plus élevé.

Pour obtenir des résultats assez satisfaisants tout en gardant la simplicité de l'algorithme, une autre méthode est utilisée qui consiste à respecter le découpage naturel du circuit. Cette méthode repose sur la notion de sous-circuit.

Le circuit est représenté par une structure hiérarchique de sous-circuits. Un sous-circuit est une unité logique qui groupe un ensemble d'éléments. Les éléments d'un sous-circuit ont une liaison étroite entre eux.

L'interface du sous-circuit est le seul lien avec l'extérieur. L'algorithme du partitionnement consiste à distribuer les sous-circuits sur les ECTs en descendant dans la hiérarchie des sous-circuits :

• Si un sous-circuit est trop gros pour être placé sur un ECT, alors il est décomposé en sous-circuits qui le constituent.

• Chaque ECT est affecté d'un certain nombre de sous-circuits qui permettent à l'ECT d'avoir une partition de circuit d'une taille adéquate.

• En cas de nécessité, le sous-circuit peut être décomposé en éléments primitifs qui sont distribués ensuite sur des ECTs.

Cette méthode de partitionnement a les caractéristiques suivantes :

• Les communications entre les ECTs peuvent être réduites. Le sous-circuit définit d'une manière naturelle un module dont les éléments sont entièrement placés sur le même ECT.

• L'algorithme est simple : Au lieu de traiter les éléments primitifs, l'analyse est limitée sur un nombre minimal de sous-circuits. La décomposition des sous-circuits en éléments primitifs ne s'effectue que dans le cas nécessaire.

• L'ordre de grandeur de l'algorithme ne dépend que de la hiérarchie du circuit. Le temps de calcul du partitionnement reste pratiquement inchangé lorsque la taille du circuit augmente. L'essentiel de l'algorithme est de trouver un certain nombre de sous-circuits en fonction des ECTs disponibles dans le système, quel que soit le nombre total des éléments primitifs du circuit.

• Chaque partition du circuit peut garder sa structure hiérarchique. La mise à plat peut s'effectuer parallèlement dans les ECTs, ce qui permet de diminuer le temps d'initialisation de la simulation.

L'efficacité de l'algorithme dépend de la structure hiérarchique du circuit définie par l'utilisateur. Lorsque le circuit est très à plat, l'algorithme du partitionnement a de la peine à fonctionner efficacement. Dans ce cas le résultat du partitionnement est proche de celui du partitionnement linéaire.

5.1.4. Comparaisons des méthodes

◊ Conditions générales

Afin de dresser un bilan et d'en tirer une conclusion, les différentes

méthodes de partitionnement ont été testées et comparées sur des circuits sélectionnés.

Quatre circuits sont testés :

- un additionneur 32 bits;
- une unité arithmétique et logique 32 bits;
- un circuit non hiérarchique créé par un générateur automatique;
- une représentation fonctionnelle du microprocesseur MC68000.

Le choix de ces quatre circuits est dû au fait que chacun de ces quatre circuits a ses propres caractéristiques :

- L'additionneur est un circuit très hiérarchique et symétrique.
- L'unité arithmétique et logique est également hiérarchique, mais moins symétrique que l'additionneur. En plus, une partie du circuit est décrite en éléments fonctionnels.
- Le circuit créé par un générateur automatique est une description à plat. Les éléments du circuit sont décrits en désordre.
- Le microprocesseur MC68000 est un circuit à plat, décrit par une description fonctionnelle. Il contient très peu d'éléments structurels. De plus, les équipotentielles sont groupées en vecteurs.

Les méthodes de partitionnement à comparer sont :

- la méthode aléatoire;
- la méthode linéaire;
- la méthode par la connectivité de sortie;
- la méthode par la connectivité d'entrée;
- la méthode par la hiérarchie.

En ce qui concerne les résultats du partitionnement, les statistiques prennent en compte les aspects suivants :

- la déviation moyenne des nombres d'éléments sur les ECTs;
- la déviation moyenne des poids d'éléments sur les ECTs;
- la déviation moyenne des nombres d'équipotentielles sur les ECTs;
- le nombre total des équipotentielles globales;
- la somme des distances logiques des équipotentielles.

Les détails sont présentés dans l'annexe C.

◊ Résultats obtenus

Les comparaisons mettent l'accent sur deux problèmes :

- l'équilibrage des charges statiques des ECTs;
- la minimisation statique des communications entre les ECTs.

Du côté de l'équilibrage des charges statiques, les statistiques montrent les faits suivants :

- Au niveau de l'équilibrage des nombres d'éléments sur les ECTs, les méthodes donnent de bons résultats : la déviation reste toujours à 0%, sauf celle de la méthode aléatoire.
- Le poids de l'élément représente la charge réelle de l'élément. Dans le cas où il y a pas ou peu d'éléments fonctionnels, les charges sur les ECTs sont relativement équilibrées (typiquement l'additionneur 32 bits). La déviation devient importante lorsque les éléments fonctionnels sont nombreux (le cas du circuit MC68000 fonctionnel). Une autre tendance est que la méthode linéaire et la méthode par la hiérarchie sont plus avantageuses pour le partitionnement des circuits très hiérarchiques et très symétriques (l'additionneur et l'unité arithmétique et logique).
- L'équilibrage des équipotentielles varie en fonction des circuits. Plus le circuit est régulier et symétrique, plus le partitionnement est équilibré. L'existence des éléments fonctionnels peut déséquilibrer les nombres d'équipotentielles sur les ECTs, car normalement un élément fonctionnel peut avoir une entrance et une sortance très élevées.

En ce qui concerne la minimisation statique des communications, les statistiques montrent les faits suivants :

- Le nombre des équipotentielles globales est très élevé dans le circuit partitionné par la méthode aléatoire par rapport aux autres méthodes. Cela signifie qu'une méthode adéquate peut réduire considérablement le nombre d'équipotentielles globales. En particulier pour les circuits hiérarchiques ou symétriques, les méthodes linéaire et hiérarchique permettent de diminuer sensiblement le nombre des équipotentielles globales.
- La somme des distances logiques représente d'une façon statique le coût de communication potentielle que la simulation peut avoir. Les

méthodes linéaire et hiérarchique donnent des résultats satisfaisants sur les circuits hiérarchiques et symétriques (l'additionneur ainsi que l'unité arithmétique et logique 32 bits). Tandis que les méthodes par la connectivité de sortie et d'entrée fonctionnent mieux sur les circuits à plat dont les éléments ne sont pas organisés dans un ordre régulier (le cas du circuit créé par un générateur automatique).

Les résultats amènent les conclusions suivantes :

• En comparant avec les autres méthodes, la méthode aléatoire ne donne pas de résultat satisfaisant, ni au niveau de l'équilibrage des charges des ECTs, ni au niveau de la minimisation des communications potentielles.

• La méthode linéaire et la méthode par la hiérarchie sont efficaces pour les circuits hiérarchiques et symétriques. L'utilisation de ces deux méthodes permet d'avoir un équilibrage des charges et une réduction statique des communications de simulation. En plus, ces deux méthodes nécessitent peu de temps de calcul. Les performances de la méthode par la hiérarchie convergent vers celles de la méthode linéaire lorsque le circuit est à plat.

• Les méthodes par la connectivité de sortie et d'entrée coûtent beaucoup plus cher au niveau du temps de calcul que les autres méthodes. Par contre leurs avantages se manifestent lorsque les circuits à partitionner ne sont pas hiérarchiques ou que la structure du circuit a peu de régularité et de symétrie. Ces deux méthodes assurent l'équilibrage des charges et la minimisation statique des communications, quelles que soient la structure et les caractéristiques des circuits à partitionner.

• Pour toutes les méthodes, lorsque le nombre d'unités de simulation s'accroît, le nombre d'équipotentielles globales et la somme des distances logiques des équipotentielles augmentent d'une façon non linéaire, beaucoup plus vite que la croissance du nombre d'unités de simulation. Ce qui montre que l'efficacité de chaque unité de simulation diminue lorsque le nombre d'unités de simulation devient important.

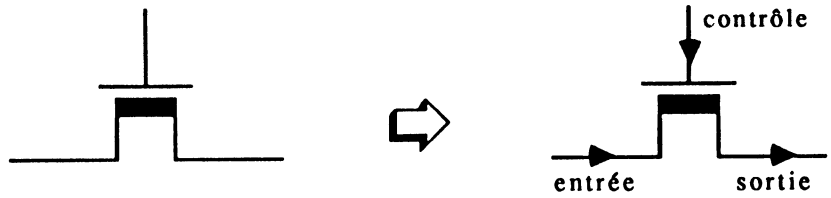
5.2. Modélisation des transistors

5.2.1. Modèles des transistors et des équipotentielles

Un transistor peut être modélisé comme un élément unidirectionnel ou bidirectionnel :

• Les primitives MOVEIF0 et MOVEIF1 sont unidirectionnelles avec une entrée, une sortie et un port de contrôle.
• Les primitives TRANIF0 et TRANIF1 sont bidirectionnelles avec deux ports sources/drains et un port de contrôle.

Les primitives MOVEIF0 et MOVEIF1 sont modélisées de la même façon que les portes logiques. La seule différence est l'état de la sortie : En plus des états 0, 1 et X, la sortie peut être à Z lorsque le transistor est bloqué.



Modèle unidirectionnel

Pour les primitives TRANIF0 et TRANIF1, les ports sources/drains n'ont pas de notion d'entrée ni de sortie. Ces ports peuvent être connectés à :

• des entrées d'éléments unidirectionnelles;
• des sorties d'éléments unidirectionnelles;
• des ports sources/drains de transistors;

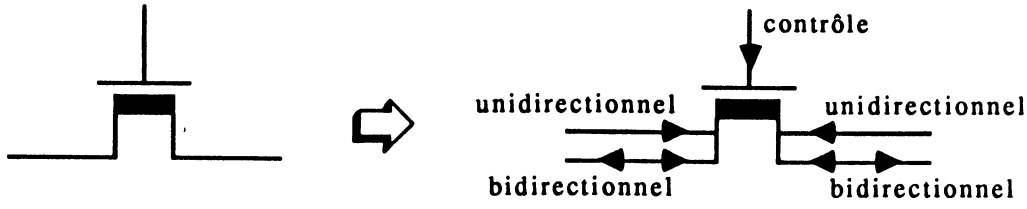
Un port source/drain réalise donc deux types de communications entre le transistor et son environnement :

• les signaux qui proviennent des éléments unidirectionnels ou y parviennent;
• les signaux qui proviennent des transistors bidirectionnels ou y parviennent.

Dans le simulateur, le port source/drain du transistor est représenté

par deux ports (unidirectionnel et bidirectionnel) qui correspondent à ces deux types de communications. Le modèle contient alors :

- deux ports unidirectionnels;
- deux ports bidirectionnels;
- un port de contrôle.



Modèle bidirectionnel

La distinction entre les ports unidirectionnels et bidirectionnels permet d'optimiser l'évaluation des transistors :

- Les ports unidirectionnels sont les entrées de transistors connectées aux sorties des éléments unidirectionnels. A chaque instant donné, l'état des sorties de ces éléments peut être déterminé immédiatement et ne nécessite pas de calcul répétitif.
- Les ports bidirectionnels permettent les communications entre les transistors. Au moment de l'évaluation d'un transistor, une méthode itérative est appliquée au transistor et aux transistors reliés directement ou indirectement par les ports bidirectionnels à ce transistor pour calculer l'état final des équipotentielles. Le calcul peut nécessiter plusieurs reprises d'évaluation.
- Le changement d'état sur les ports unidirectionnels et celui sur les ports bidirectionnels sont les deux types d'événements distincts. Ces deux types d'événements entraînent deux différents algorithmes d'évaluation qui permettent de calculer d'une manière plus précise et plus propre le modèle de transistor.

L'intégration des modèles de transistors dans le circuit nécessite non seulement la modélisation des transistors eux-mêmes, mais aussi celle des équipotentielles.

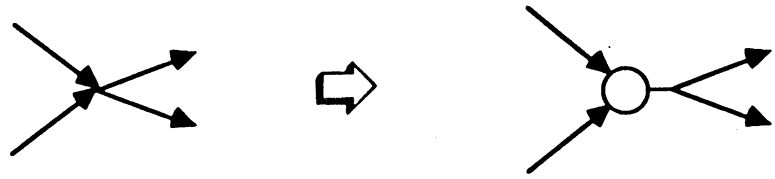
Une équipotentielle reliée aux ports sources/drains de transistors a les propriétés suivantes :

• la nature bidirectionnelle : Contrairement aux équipotentielles reliées aux entrées et sorties des portes classiques, les signaux peuvent circuler dans les deux sens;

• la logique à trois états : Physiquement, le port source/drain de transistor peut être à 0, 1 ou Z.

La nature bidirectionnelle de l'équipotentielle exige que les ports bidirectionnels utilisent les équipotentielles spéciales qui ne doivent pas être connectées directement aux sorties de portes unidirectionnelles.

La logique à trois états nécessite un mécanisme de gestion de l'état de l'équipotentielle. Pour avoir de bonnes performances de simulation, les pré-traitements (la compilation, l'édition de liens, et la génération de code) interviennent pour générer des éléments TRI à la place des équipotentielles. Le calcul de l'état de l'équipotentielle est alors explicite et ne s'effectue que si un élément TRI est présent :



La fonction TRI permet de calculer l'état de l'équipotentielle à trois états lorsque des signaux se rencontrent sur celle-ci.

Cette fonction est :

• commutative : TRI (α,β) = TRI (β,α);
• associative : TRI (TRI (α,β),γ) = TRI (α,TRI (β,γ)).

Dans le domaine {0,1,X,Z} la fonction TRI est définie comme suit :

	0	1	X	Z
0	0	X	X	0
1	X	1	X	1
X	X	X	X	X
Z	0	1	X	Z

Dans le domaine de la logique à 15 valeurs {0, N, L, B, F, D, X, W, Z, U, R, T, H, P, 1} (la définition de ces 15 valeurs est donnée dans l'annexe A), la fonction TRI est définie par la table de vérité suivante :

	0	N	L	B	F	D	X	W	Z	U	R	T	H	P	1
0	0	0	0	0	0	0	X	0	0	X	0	X	0	X	X
N	0	N	N	N	N	D	X	D	N	X	D	X	D	X	X
L	0	N	L	N	L	D	X	W	L	U	W	U	W	U	1
B	0	N	N	B	B	D	X	D	B	X	D	X	D	X	X
F	0	N	L	B	F	D	X	W	F	U	W	U	W	U	1
D	0	D	D	D	D	D	X	D	D	X	D	X	D	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
W	0	D	W	D	W	D	X	W	W	U	W	U	W	U	1
Z	0	N	L	B	F	D	X	W	Z	U	R	T	H	P	1
U	X	X	U	X	U	X	X	U	U	U	U	U	U	U	1
R	0	D	W	D	W	D	X	W	R	U	R	T	H	P	1
T	X	X	U	X	U	X	X	U	T	U	T	T	P	P	1
H	0	D	W	D	W	D	X	W	H	U	H	P	H	P	1
P	X	X	U	X	U	X	X	U	P	U	P	P	P	P	1
1	X	X	1	X	1	X	X	1	1	1	1	1	1	1	1

5.2.2. Intégration des transistors dans le circuit

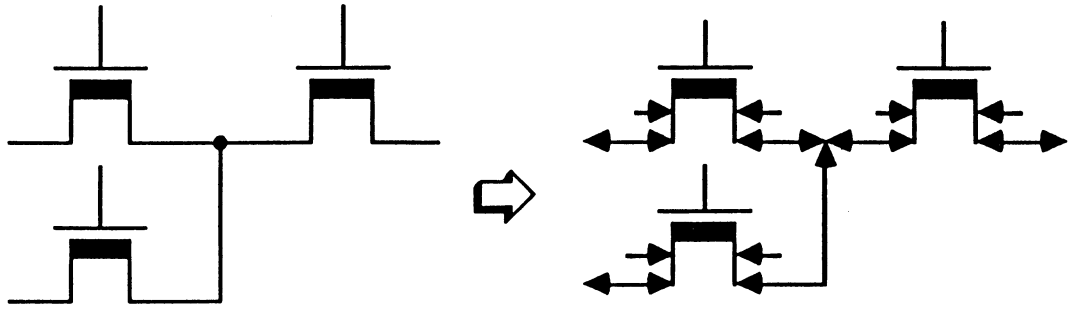
L'intégration des modèles de transistors dans le circuit nécessite de résoudre les problèmes suivants :

- Comment s'organisent les ports unidirectionnels et bidirectionnels des transistors dans le circuit?
- Dans quels cas faut-il ajouter des éléments TRI à la place des équipotentielles?
- Comment sont réalisées les interconnexions entre les transistors, les éléments TRI et les autres types d'éléments?
- De quelle manière sont introduits les points de stimulus et de réponse dans le circuit contenant des transistors?

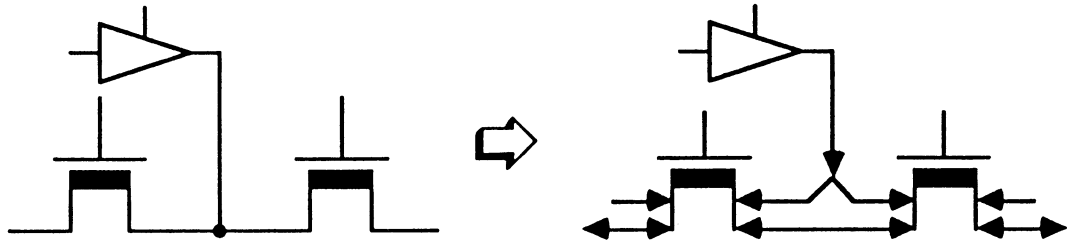
Ces problèmes sont pris en compte par les outils de compilation qui suivent un ensemble de règles :

- Les transistors sont interconnectés directement par leurs ports

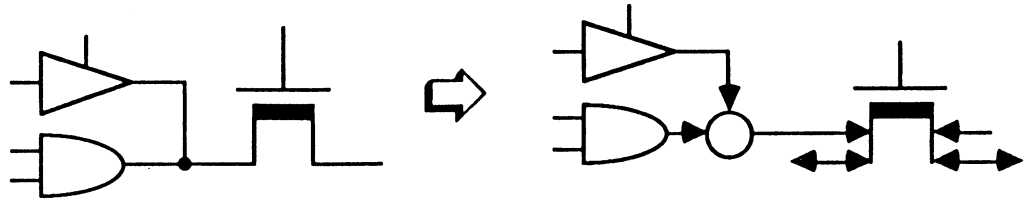
bidirectionnels :



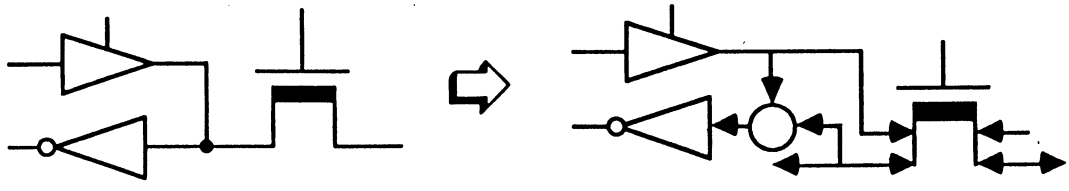
• La sortie de porte est connectée aux ports unidirectionnels de transistors :



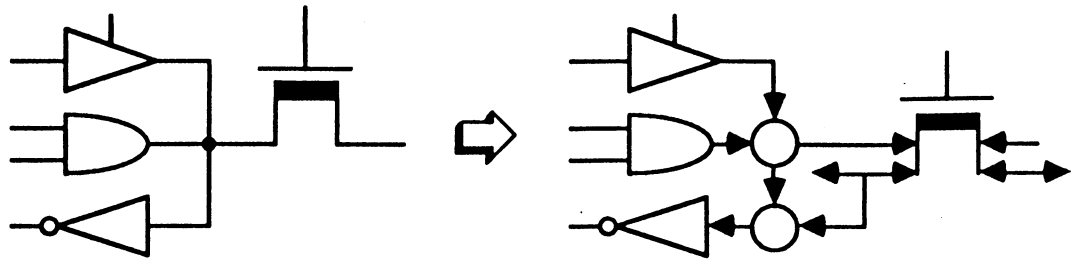
• Lorsque plusieurs sorties de portes sont connectées au port source/drain du transistor, un élément complémentaire TRI est utilisé :



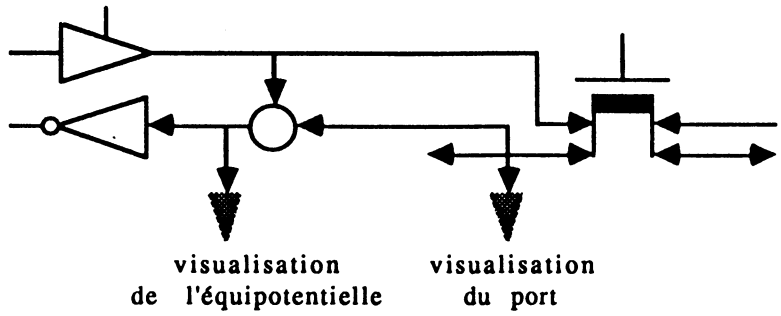
• S'il y a des entrées de portes connectées à l'équipotentielle du port source/drain du transistor, un élément TRI est ajouté sur l'équipotentielle :



• S'il y a des sorties de portes connectées à l'équipotentielle et que des entrées de portes y sont connectées également, deux éléments TRI sont utilisés :



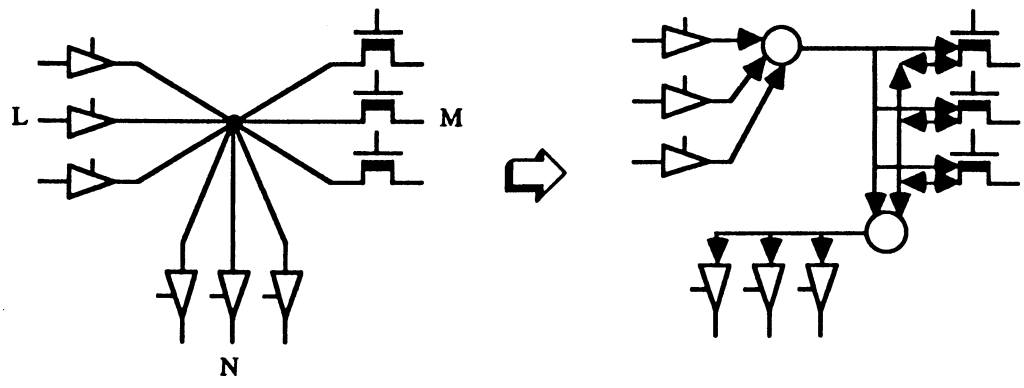
• La visualisation de l'état de l'équipotentielle suit le même principe. Il est possible de visualiser, pour une équipotentielle, l'état final de celle-ci ou l'état du port bidirectionnel :



En général, lorsque L sorties unidirectionnelles, M ports sources/drains de transistors et N entrées unidirectionnelles sont tous connectés à l'équipotentielle, elle sera représentée dans le circuit par 2 éléments TRI et L+3 équipotentielles :

• un élément TRI dont les L entrées sont connectées aux sorties unidirectionnelles et que la sortie est connectées à M ports unidirectionnels;

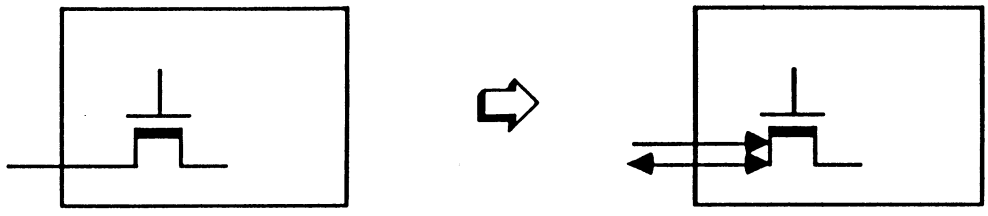
• un élément TRI dont la sortie est connectée à N entrées unidirectionnelles.



5.2.3. Prise en compte de la hiérarchie de circuits

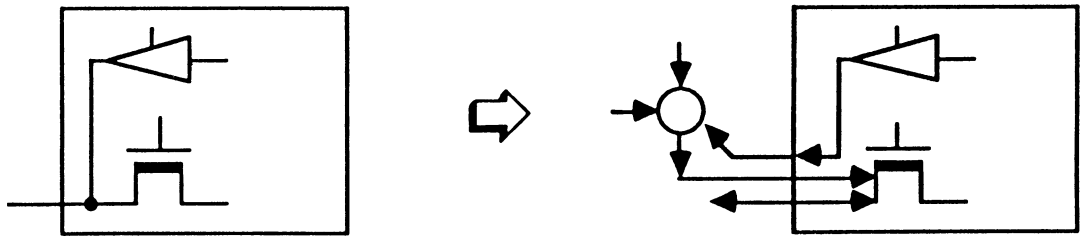
La mise à plat du circuit à simuler peut être effectuée par le simulateur. Dans ce cas l'image exécutable du circuit générée par le générateur de code reste hiérarchique.

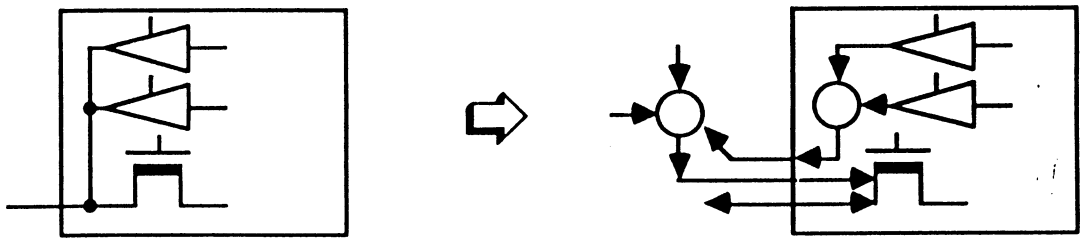
Le problème essentiel de la représentation hiérarchique se trouve sur l'interface entre le sous-circuit et le circuit englobant. L'utilisation des transistors nécessite des connexions bidirectionnelles entre les deux circuits :



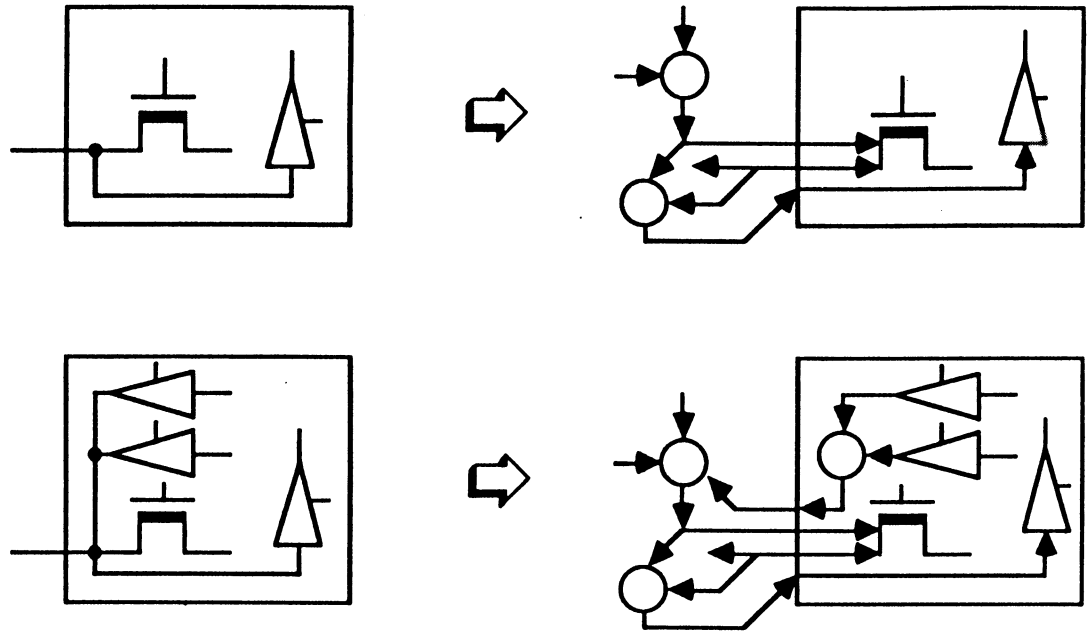
Dans les outils de compilation, un ensemble de règles ont été définies pour assurer la compatibilité d'interfaces de circuits :

• Si l'équipotentielle est connectée à une ou plusieurs sorties de portes dans le sous-circuit, trois connexions sont nécessaires :



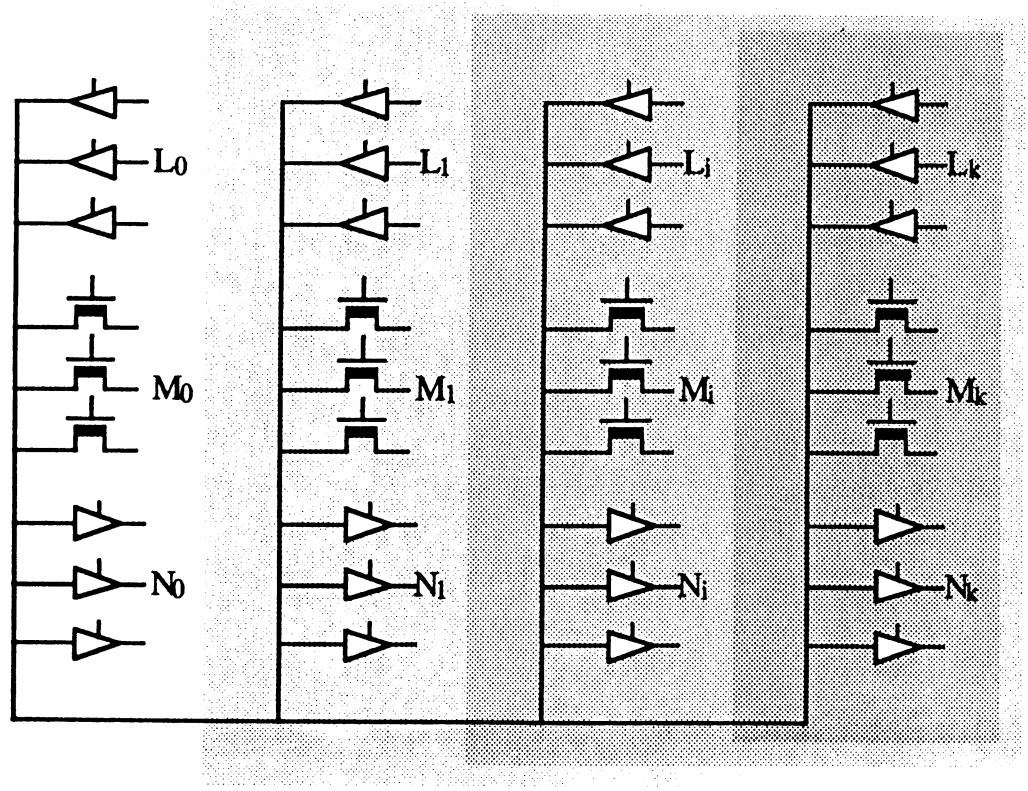


• Si l'équipotentielle est connectée à des entrées de portes dans le sous-circuit, trois ou quatre connexions doivent être utilisées :

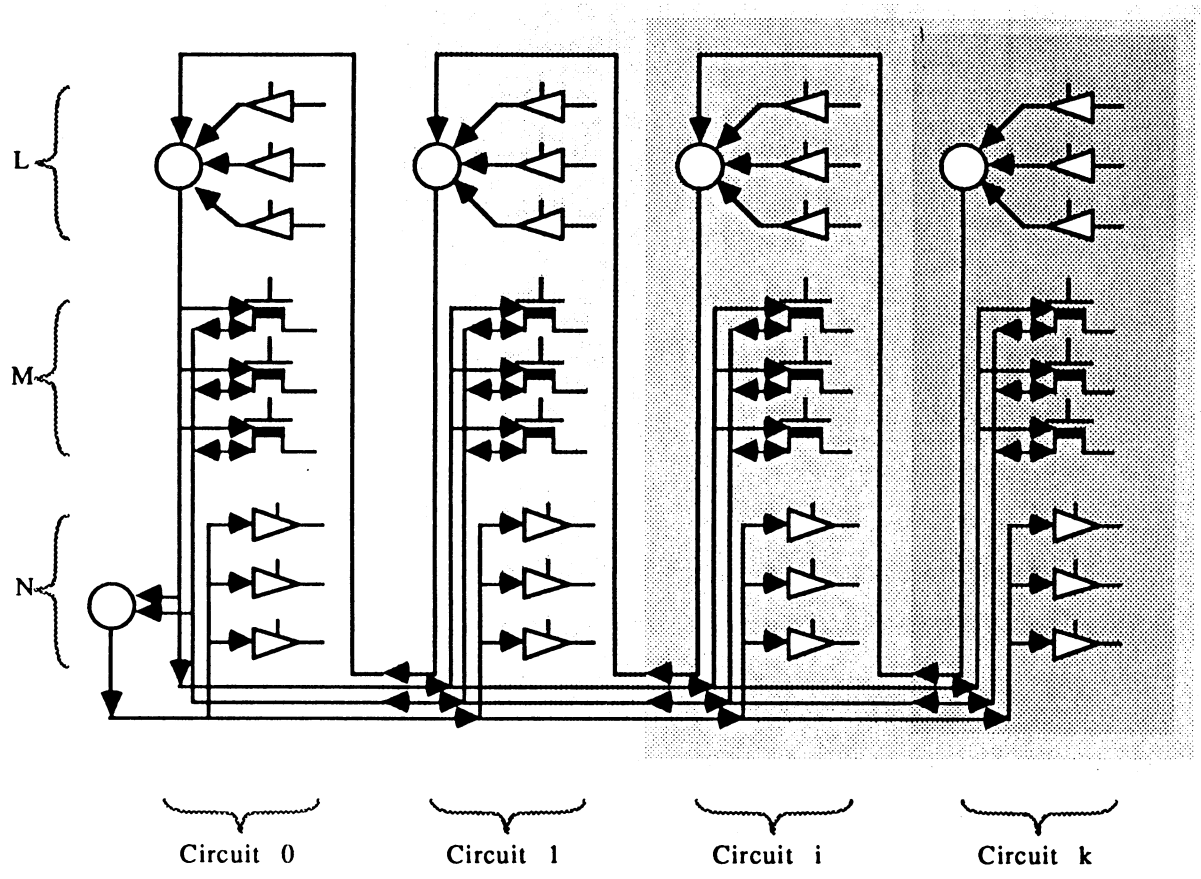


Considérons le cas général, le nombre de niveaux de la hiérarchie étant K+1, l'équipotentielle traverse K+1 circuits imbriqués.

Dans chaque circuit i, l'équipotentielle est connectée à L_i sorties unidirectionnelles, M_i ports sources/drains de transistors, et N_i entrées unidirectionnelles :



Pour cette équipotentielle, les outils de compilation génèrent la configuration suivante :



Sur les interfaces, l'équipotentielle originale est systématiquement représentée par quatre équipotentielles :

- l'équipotentielle de sorties unidirectionnelles;
- l'équipotentielle de ports unidirectionnels des transistors;
- l'équipotentielle de ports bidirectionnels des transistors;
- l'équipotentielle d'entrées unidirectionnelles.

Dans chaque circuit i, un élément TRI_i est utilisé dont l'entrance est égale à L_i+1, $0 \leq i < k$, sauf le circuit k dans lequel l'entrance de l'élément TRI_k est égale à L_k. La sortance de TRI_i est égale à 1, $0 < i \leq k$, sauf l'élément TRI_0 dont la sortance est $1+\Sigma M_i$. Dans le circuit principal, un deuxième élément TRI est utilisé dont la sortance est ΣN_i.

En total, $K+3+\Sigma L_i$ équipotentielles et K+2 éléments TRI sont utilisés.

5.2.4. Evaluation des modèles de transistors

◊ Principe de base

Dans l'algorithme, les ports de transistors sont dénotées comme suit :

- le port de contrôle : p_c;
- les ports unidirectionnels : p_{u1} et p_{u2};
- les ports bidirectionnels : p_{b1} et p_{b2}.

Trois fonctions sont utilisées par l'algorithme :

- la fonction f_{tri} (équivalente à l'opération C présentée dans le chapitre 2) qui définit l'état de l'équipotentielle dans le cas où plusieurs signaux se rencontrent sur l'équipotentielle;
- la fonction $f_{strength}$ (équivalente à l'opération T présentée dans le chapitre 2) qui décrit l'effet de l'atténuation du transistor lorsque le signal traverse le transistor;
- la fonction f_{moveif} qui représente la fonction de sortie du modèle unidirectionnel du transistor.

Pour un modèle MOVEIF1, lorsque le port de contrôle est à 0 ou 1, le transistor est bloqué ou passant :

- sortie = f_{moveif} (0, entrée) = Z;
- sortie = f_{moveif} (1, entrée) = entrée.

L'effet de l'évaluation du modèle de transistor est d'une part la mise à jour de l'état des équipotentielles et d'autre part la création des nouveaux événements.

Dans les descriptions ci-après, une affectation :

équipotentielle i ← nouvel état e'

signifie à la fois deux opérations suivantes :

- la mise à jour de l'état de l'équipotentielle i;
- la création de l'événement (t,e',i).

Les retards du transistor sont pris en compte pour calculer le temps t du nouvel événement créé.

Le transistor est évalué lorsque des événements y arrivent. Il existe trois types d'événements, dans l'ordre de priorité décroissant :

- le changement d'état sur le port de contrôle;
- le changement d'état sur les ports unidirectionnels;
- le changement d'état sur les ports bidirectionnels.

Ces trois types d'événements nécessitent chacun une évaluation distincte. Dans le cas où plusieurs types d'événements arrivent au même moment sur le transistor, l'évaluation qui correspond à l'événement le plus prioritaire sera entreprise.

◊ Algorithme

L'algorithme distingue les cas suivants :

- le changement d'état sur le port de contrôle ou sur les deux ports unidirectionnels;
- le changement d'état sur un port unidirectionnel;
- le changement d'état sur un port bidirectionnel;
- le changement d'état sur deux ports bidirectionnels.

Pour simplifier le problème, on ne prend en compte que le modèle bidirectionnel TRANIF1 qui est passant quand le port de contrôle est à 1.

Lorsque le port de contrôle ou les deux ports unidirectionnels changent d'état, les deux ports bidirectionnels seront affectés. Les opérations à entreprendre sont les suivantes :

$$p_{b1} \leftarrow f_{tri}\,(p_{u1},\, f_{moveif}\,(p_c,\, f_{strength}\,(p_{u2})))$$
$$p_{b2} \leftarrow f_{tri}\,(p_{u2},\, f_{moveif}\,(p_c,\, f_{strength}\,(p_{u1})))$$

Plus précisément :

• Dans le cas où le nouvel état du port de contrôle est à 0, il y a f_{tri} (p_{u1}, f_{moveif} (p_c, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, f_{moveif} (0, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, Z) = p_{u1}. Donc les opérations sont :

$p_{b1} \leftarrow p_{u1}$
$p_{b2} \leftarrow p_{u2}$

• Dans le cas où le nouvel état du port de contrôle est à 1, il y a f_{tri} (p_{u1}, f_{moveif} (p_c, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, f_{moveif} (1, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, $f_{strength}$(p_{u2})). Donc les opérations sont :

$p_{b1} \leftarrow f_{tri}$ (p_{u1}, $f_{strength}$ (p_{u2}))
$p_{b2} \leftarrow f_{tri}$ (p_{u2}, $f_{strength}$ (p_{u1}))

• Dans le cas où le nouvel état du port de contrôle est à X, alors :

$p_{b1} \leftarrow f_{tri}$ (p_{u1}, f_{moveif} (X, $f_{strength}$ (p_{u2})))
$p_{b2} \leftarrow f_{tri}$ (p_{u2}, f_{moveif} (X, $f_{strength}$ (p_{u1})))

Lorsqu'un port unidirectionnel change d'état (prenons le cas de p_{u1}), Les calculs suivent le même principe :

$p_{b1} \leftarrow f_{tri}$ (p_{u1}, f_{moveif} (p_c, $f_{strength}$ (p_{u2})))
$p_{b2} \leftarrow f_{tri}$ (p_{u2}, f_{moveif} (p_c, $f_{strength}$ (p_{u1})))

Plus précisément :

• Dans le cas où le nouvel état du port de contrôle est à 0, le transistor est bloqué. L'état de p_{b2} ne sera pas influencé par le changement d'état de p_{u1}. De l'autre côté, il y a f_{tri} (p_{u1}, f_{moveif} (p_c, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, f_{moveif} (0, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, Z) = p_{u1}. Donc l'opération à entreprendre est :

$p_{b1} \leftarrow p_{u1}$

• Dans le cas où le nouvel état du port de contrôle est à 1, il y a f_{tri} (p_{u1}, f_{moveif} (p_c, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, f_{moveif} (1, $f_{strength}$ (p_{u2}))) = f_{tri} (p_{u1}, $f_{strength}$ (p_{u2})). Donc les opérations sont :

$$p_{b1} \leftarrow f_{tri}\,(p_{u1},\, f_{strength}\,(p_{u2}))$$
$$p_{b2} \leftarrow f_{tri}\,(p_{u2},\, f_{strength}\,(p_{u1}))$$

• Dans le cas où le nouvel état du port de contrôle est à X, alors :

$$p_{b1} \leftarrow f_{tri}\,(p_{u1},\, f_{moveif}\,(X,\, f_{strength}\,(p_{u2})))$$
$$p_{b2} \leftarrow f_{tri}\,(p_{u2},\, f_{moveif}\,(X,\, f_{strength}\,(p_{u1})))$$

Lorsqu'un port bidirectionnel change d'état (prenons le cas de p_{b1}), l'opération à entreprendre est la suivante :

$$p_{b2} \leftarrow f_{tri}\,(p_{u2},\, f_{moveif}\,(p_c,\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1}))))$$

Plus précisément :

• Dans le cas où le nouvel état du port de contrôle est à 0, le transistor est bloqué. L'état de p_{b2} ne sera pas influencé par le changement d'état de p_{b1}. Donc aucune opération ne sera entreprise.

• Dans le cas où le nouvel état du port de contrôle est à 1, il y a $f_{tri}\,(p_{u2},\, f_{moveif}\,(p_c,\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1})))) = f_{tri}\,(p_{u2},\, f_{moveif}\,(1,\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1})))) = f_{tri}\,(p_{u2},\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1})))$. Donc l'opération est :

$$p_{b2} \leftarrow f_{tri}\,(p_{u2},\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1})))$$

• Dans le cas où le nouvel état du port de contrôle est à X, alors :

$$p_{b2} \leftarrow f_{tri}\,(p_{u2},\, f_{moveif}\,(X,\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1}))))$$

Lorsque deux ports bidirectionnels changent d'état simultanément, le calcul est le suivant :

$$p_{b1} \leftarrow f_{tri}\,(f_{tri}\,(p_{u1},\, p_{b1}),\, f_{moveif}\,(p_c,\, f_{strength}\,(f_{tri}\,(p_{u2},\, p_{b2}))))$$
$$p_{b2} \leftarrow f_{tri}\,(f_{tri}\,(p_{u2},\, p_{b2}),\, f_{moveif}\,(p_c,\, f_{strength}\,(f_{tri}\,(p_{u1},\, p_{b1}))))$$

Plus précisément :

• Dans le cas où le nouvel état du port de contrôle est à 0, le transistor est bloqué. L'état de p_{b2} ne sera pas influencé par le changement d'état de p_{b1}, et réciproquement. Donc aucune opération ne sera entreprise.

• Dans le cas où le nouvel état du port de contrôle est à 1, il y a f_{tri} (f_{tri} (p_{u1}, p_{b1}), f_{moveif} (p_c, $f_{strength}$ (f_{tri} (p_{u2}, p_{b2})))) = f_{tri} (f_{tri} (p_{u1}, p_{b1}), f_{moveif} (1, $f_{strength}$ (f_{tri} (p_{u2}, p_{b2})))) = f_{tri} (f_{tri} (p_{u1}, p_{b1}), $f_{strength}$ (f_{tri} (p_{u2}, p_{b2}))). Donc les opérations sont :

$$p_{b1} \leftarrow f_{tri}\ (f_{tri}\ (p_{u1}, p_{b1}), f_{strength}\ (f_{tri}\ (p_{u2}, p_{b2})))$$
$$p_{b2} \leftarrow f_{tri}\ (f_{tri}\ (p_{u2}, p_{b2}), f_{strength}\ (f_{tri}\ (p_{u1}, p_{b1})))$$

• Dans le cas où le nouvel état du port de contrôle est à X, alors :

$$p_{b1} \leftarrow f_{tri}\ (f_{tri}\ (p_{u1}, p_{b1}), f_{moveif}\ (X, f_{strength}\ (f_{tri}\ (p_{u2}, p_{b2}))))$$
$$p_{b2} \leftarrow f_{tri}\ (f_{tri}\ (p_{u2}, p_{b2}), f_{moveif}\ (X, f_{strength}\ (f_{tri}\ (p_{u1}, p_{b1}))))$$

En conclusion, lorsqu'il y a un changement d'état sur un transistor, ce changement d'état provoque une série de changements d'état sur les transistors connectés directement ou indirectement à ce transistor. L'évaluation d'un événement peut entraîner la créations de nouveaux événements et nécessiter plusieurs reprises de calcul. Les événements cessent de se propager lorsque les états deviennent stables.

5.3. Modélisation des éléments fonctionnels

5.3.1. Modèles fonctionnels

◊ Structure de données

Dans le simulateur, un élément fonctionnel se compose de :

- un type;
- une liste d'entrées
- une liste de sorties;
- une liste de retards;
- une liste de forces;
- une liste d'instructions.

L'élément fonctionnel se distingue de la porte logique par la liste d'instructions qui permet de réaliser n'importe quels types de fonction.

L'évaluation de l'élément fonctionnel est déclenchée par un ou des changements d'état sur les entrées de l'élément. Celui-ci est évalué par l'exécution de ses instructions sur une machine virtuelle.

Dans l'élément fonctionnel, les instructions peuvent accéder aux entrées et sorties de l'élément ou directement aux objets définis dans le circuit, tels que les équipotentielles, les registres, les mémoires, etc. Lorsque ces instructions accèdent aux sorties de l'élément, la liste de retards et la liste de forces de l'élément peuvent être utilisées pour créer des événements avec des retards et des états appropriés.

Les instructions sont exécutées instantanément. L'interprétation des instructions de l'élément fonctionnel se déroule sans interruption. En conséquence, l'évaluation des éléments fonctionnels est instantanée. Cette nature permet de gérer les éléments fonctionnels d'une façon identique aux portes logiques.

D'autre part, il est possible de définir et d'utiliser des variables internes dans l'élément, ce qui permet de programmer un élément en un automate d'états finis. Les opérations temporelles ou à retardement peuvent être réalisées à l'aide de l'utilisation de l'automate et de la création des événements.

Les modèles fonctionnels ont alors les caractéristiques suivantes :

• Les expressions booléennes sont réalisées par une séquence d'instructions.

• Les opérations séquentielles sont réalisées par l'utilisation des automates.

• Les opérations temporelles sont réalisées à l'aide d'événements et d'automates.

• La nature non procédurale et le parallélisme inhérent aux éléments fonctionnels sont assurés par le mécanisme de l'échéancier.

Les outils de compilation, en particulier le générateur de code, jouent un rôle important pour la création de la liste d'instructions de l'élément fonctionnel, à partir de l'arbre syntaxique abstrait de celui-ci et des attributs associés aux noeuds de l'arbre.

◊ Algorithme de l'évaluation

L'ECT utilise un algorithme de multi-phases pour assurer l'évaluation correcte des éléments fonctionnels. Ceux-ci peuvent se comporter comme des automates dont le changement d'état interne est sensible à l'ordre d'apparition des événements sur les entrées :

• A chaque étape de simulation, l'ECT reçoit, de l'échéancier, tous les événements de l'instant le plus proche.

• Les états des équipotentielles correspondantes à ces événements sont mis à jour.

• Les portes logiques, les transistors et certains types d'éléments fonctionnels sont évalués immédiatement.

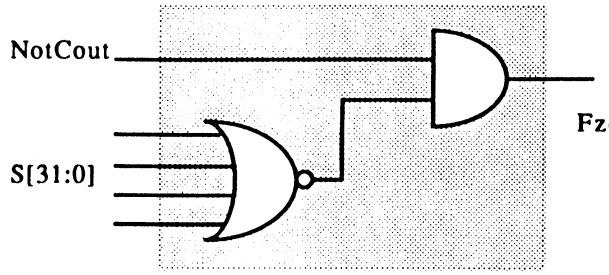
• L'évaluation des éléments fonctionnels ne commence que si les éléments primitifs sont évalués, et que les états des équipotentielles sont stables.

5.3.2. Expressions arithmétiques et logiques

◊ Expressions purement arithmétiques et logiques

Une expression arithmétique et logique est un élément fonctionnel qui représente un circuit combinatoire.

L'exemple suivant est une expression logique :




```
WIRE (10:12,11:15)
   Fzero = NotCout AND
           (NOR (S[31:0]));
```

Pour cette expression, le générateur de code génère l'élément fonctionnel suivant, décrit ici sous une représentation symbolique :

```
TYPE :
        FUNCTIONAL;
INPUT LIST :
        NotCout, S[31:0];
OUTPUT LIST :
        Fzero;
DELAY LIST :
        10:12, 11:15;
INSTRUCTION LIST :
        LDIW   NotCout,        -- Load Source Wire NotCout
        LDIW   S[31:0],        -- Load Source Wire S[31:0]
        LDL    0,              -- Load Destination Local 0
        SVR    .NOR   32,      -- Local 0 := NOR (S[31:0])
        LDW    Fzero,          -- Load Destination Wire Fzero
        SDE    .AND,           -- Fzero := NotCout AND Local 0
        EXIT.
```

◊ Expressions avec structures de branchement

Une expression peut avoir, en plus des opérateurs arithmétiques et logiques, des branchements conditionnels, lorsque la structure VALCASE est utilisée dans l'expression :

```
WIRE R = NOT (VALCASE C,
                    0 = 1,
                    DEFAULT = B
              ENDCASE);
```

Cette expression sera réalisée par un élément fonctionnel dont la liste d'instructions est la suivante :

```
        LDIW  C,          -- Load Source Wire C
        BRNZ  label1,     -- Branch to label1 if not 0
        LDC   1,          -- Load Constant 1
        BRA   label2,     -- Branch to label 2
label1  LDIW  B,          -- Load Source Wire B
label2  LDIW  R,          -- Load Destination R
        SME   .NOT  1,    -- R := NOT (Top of stack)
        EXIT.
```

5.3.3. Registres virtuels

◊ Réalisation des modèles de registres

Un registre est un élément fonctionnel auquel est associée une liste d'instructions définissant le déclenchement de l'affectation du registre.

L'exemple suivant est un registre 32 bits :

```
REGISTER (103:107:110,102:106:110)
        Reg[0:31] = 0 LOADIF1 (Winit OR Wreset);
```

Le comportement du registre est défini comme suit :

- Si (Winit OR Wreset) = 0, alors Reg[0:31] reste inchangé.
- Si (Winit OR Wreset) = 1, alors Reg[0:31] := 0.
- Si (Winit OR Wreset) = X, alors Reg[0:31] := Reg[0:31] SAME 0.

Le générateur de code crée l'élément fonctionnel suivant pour réaliser ce registre :

```
TYPE :
        REGISTER;
INPUT LIST :
        Reg_In[0:31],
        Winit,
        Wreset;
```

```
OUTPUT LIST :
        Reg_Out[0:31];
DELAY LIST :
        103:107:110,
        102:106:110;
INSTRUCTION LIST :
        LDIW   Winit,              -- Load Source Wire Winit
        LDIW   Wreset,             -- Load Source Wire Wreset
        LDL    0,                  -- Load Destination Local 0
        SDE    .OR    1,           -- Local 0 := Winit OR Wreset
        BRNZ   label1,             -- Branch to label1 if not 0
        EXIT,
label1  LDL    0,                  -- Load Source Local 0
        LDC    1,                  -- Load Constant 1
        BDE    .EQ,                -- Local 0 = 1 ?
        BRZ    label2,             -- Branch to label2 if false
        LDC    0,                  -- Load Constant 0
        LDIW   Reg_Out[0:31],      -- Load Destination Wire Reg_Out[0:31]
        SME    .NOP   32,          -- Reg_Out[0:31] := 0
        EXIT,
label2  LDIW   Reg_In[0:31],       -- Load Source Wire Reg_In[0:31]
        LDC    0,                  -- Load Constant 0
        LDIW   Reg_Out[0:31],      -- Load Destination Wire Reg_Out[0:31]
        SDE    .SAME 32,           -- Reg_Out[0:31] := Reg_In[0:31] SAME 0
        EXIT.
```

◊ Intégration des modèles de registres dans le circuit

Pour intégrer les modèles de registres dans le circuit, plusieurs problèmes doivent être résolus, notamment les accès multiples aux registres.

Pour un élément structurel, tel que la porte logique, lorsqu'il existe plusieurs signaux qui peuvent arriver sur une entrée de la porte, un élément TRI est ajouté dans le circuit pour l'arbitrage de l'état final. Chacune des entrées de l'élément TRI est connectée à un signal, sa sortie connectée à l'entrée de la porte.

Cependant, pour un élément fonctionnel l'ajout de l'élément TRI est impossible, car les communications entre les éléments fonctionnels sont réalisées par les opérations d'affectation, au lieu des connexions logiques.

Pour un registre, le conflit des accès multiples se produit lorsque le registre est affecté simultanément des états différents :

```
WHEN condition1 DO Reg [32:1] := expression1;
WHEN condition2 DO Reg [32:1] := expression2;
                ...
WHEN conditionn DO Reg [32:1] := expressionn;
```

La solution consiste à associer au registre des instructions spécifiques :

```
LDIW   Reg_In,
LDIW   Reg_Out,
REG    .NOP  32,
EXIT.
```

Lorsque plusieurs affectations d'un même registre se produisent à la même heure de simulation, les états à affecter au registre sont vérifiés. Si au même instant l'état à affecter est différent de celui déjà affecté au registre, alors le registre est affecté de l'état X.

L'utilisation des instructions fonctionnelles permet en plus de résoudre le problème de priorité de l'évaluation d'une manière très simple, qui n'influence pas la sémantique du simulateur.

Dans un cas complexe, le registre peut être affecté simultanément par des structures gardées WHEN, pendant que l'expression associée au registre lui-même reste valide :

```
REGISTER (10,10) Reg = W1 LOADIF1 A;

WHEN A DO Reg = W2;
WHEN A DO Reg = W3;
```

Dans ce cas l'expression associée au registre Reg est prioritaire par rapport aux affectations définies dans les structures gardées WHEN.

Alors, dans le code généré, le modèle fonctionnel du registre Reg contient :

- les instructions qui réalisent l'expression associée;
- les instructions qui réalisent les affectations définies dans WHEN.

Les outils de compilation génère ces instructions automatiquement dans un ordre conforme à la priorité de l'évaluation, de sorte que l'expression associée au registre soit prise en compte d'abord.

5.3.4. Expressions d'événements

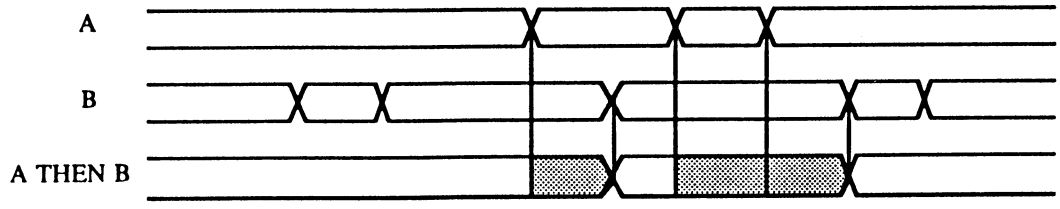
◊ Expressions d'événements et automates déterministes d'états finis

Les expressions d'événements réalisent une condition de garde qui déclenche une série d'actions.

L'expression d'événements contient des opérations temporelles ou synchronisées :

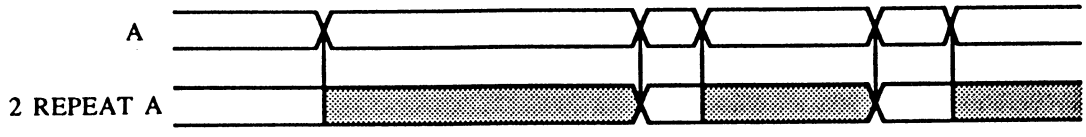
- La séquence des événements constitue un événement composé qui est valide lorsque les deux événements constituants se produisent en ordre :

expression d'événements ::=
expression d'événements THEN expression d'événements



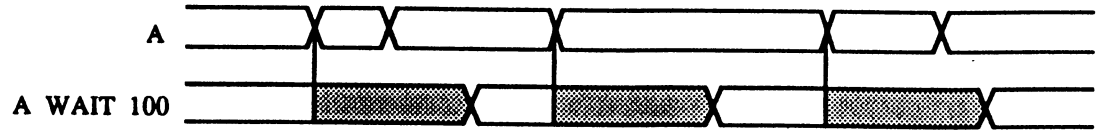
- La répétition des événements définit un événement composé :

expression d'événements ::= nombre REPEAT (expression d'événements)



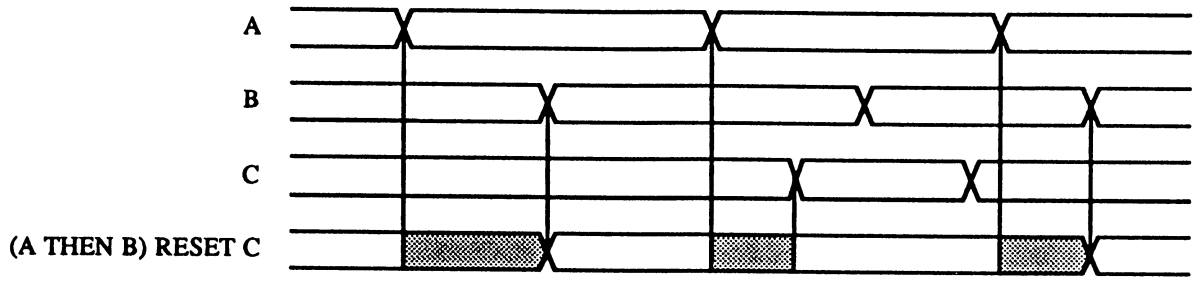
- Le retardement associe un délai à une expression d'événements :

expression d'événements ::= (expression d'événements) WAIT retard



• La détection d'événements hors séquence est basée sur deux sous-expressions :

expression d'événements ::=
expression d'événements RESET expression d'événements



En effet, l'expression d'événements a par nature les caractéristiques suivantes :

• Chaque étape de l'évaluation de l'expression a sa propre durée, qui est déterminée par l'intervalle de deux événements ou par un retardement.

• L'évaluation de l'expression d'événements progresse au fur et à mesure que les événements apparaissent. Par conséquent, Différentes séquences d'événements donnent différents résultats. L'évaluation de l'expression peut être interrompue ou reinitialisée par des événements.

• Deux évaluations succésives de la même expression ne peuvent ni coïncider ni chevaucher. Une nouvelle évaluation ne commence que lorsque l'évaluation précédente est terminée.

Toutes ces caractéristiques sont conformes à la notion d'automate. Un automate déterministe d'états finis a les propriétés suivantes :

• L'évaluation de l'automate est temporelle. La transition se produit lorsqu'une condition de transition devient vraie, ce qui permet à l'automate d'avoir la notion de durée de vie.

• L'automate peut transiter d'un état à plusieurs états possibles. La détermination du prochain état dépend de l'état courant de l'automate et des

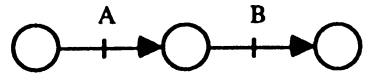
conditions actuelles. Il n'existe pas de transition à une condition nulle. L'automate est déterministe car une seule transition est possible pour les mêmes conditions.

• L'automate représente un processus dynamique. Le flot de contrôle est défini par le graphe de transition. Il est donc impossible d'avoir plusieurs sous-processus qui coexistent dans l'automate.

La compatibilité entre les propriétés intrinsèques des expressions d'événements et celles des automates permet de représenter les expressions d'événements par les automates :

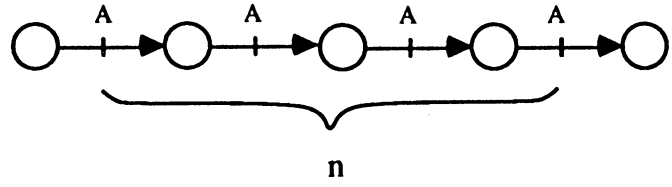
• la séquence des événements :

A THEN B



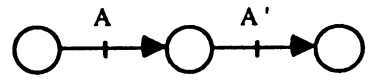
• la répétition des événements :

n REPEAT A



• le retardement des événements :

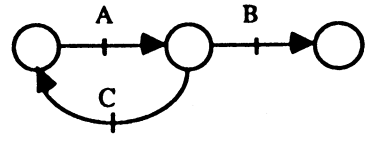
A WAIT t



A' = événement qui se produit à l'heure de A + t

• la détection des événements hors séquence :

(A THEN B) RESET C



La construction de l'automate d'états finis est réalisée de la façon suivante :

• Les équipotentielles, sur lesquelles les événements primitifs (les changements d'états et les événements fonctionnels) se produisent qui peuvent provoquer une transition d'état de l'automate, sont mises dans la liste d'entrées de l'élément fonctionnel. Lorsqu'une des entrées change, l'élément est évalué. L'instruction FRTO permet de détecter le changement d'état sur une entrée.

• L'état de l'automate est représenté par une variable interne de l'élément fonctionnel. Les instructions LDH et STH permettent la lecture et l'écriture de cette variable. Lorsque l'élément est évalué, la variable est lue, et sera modifiée en fonction de la transition d'état de l'automate.

• Pour l'élément fonctionnel, différents états d'automate nécessitent différents traitements. L'instruction BRT permet de brancher au traitement qui correspond à l'état courant de l'automate.

Lors de sa construction, l'automate est optimisé. Cette optimisation est réalisée par la maximisation et la minimisation des états. La maximisation des états consiste à remplacer un état complexe, en particulier l'état final de l'automate, par plusieurs états simples. La minimisation des états consiste à fusionner plusieurs états, notamment les états transitoires, en un seul.

Dans l'élément fonctionnel, l'automate est réalisé par la structure suivante :

```
        LDH   State,
        BRT   n,
        state0,
        state1,
        state2,
        ...,
        staten-1,
state0  (processing state0),
        EXIT,
state1  (processing state1)
        EXIT,
state2  (processing state2)
        EXIT,
        ...
staten-1 (processing staten-1)
        EXIT,
```

◊ Séquence d'événements

Pour réaliser la séquence d'événements, par exemple A THEN B, le générateur de code crée l'élément suivant :

```
INPUT LIST :
        A, B;
VARIABLE LIST :
        State;
INSTRUCTION LIST :
        LDH     State,          -- Load State
        BRT     2,              -- Branch according to current state
        state0,                 -- Label of state0
        state1,                 -- Label of state1

state0  LDIW    A,              -- Load Source Wire A
        FRTO    ? TO ? 1,       -- Any change on A ?
        BRZ     label1,         -- Branch to label1 if false
        LDC     1,              -- Load Constant 1
        STH     State,          -- State := 1
label1  EXIT,

state1  LDIW    B,              -- Load Source Wire B
        FRTO    ? TO ? 1,       -- Any change on B ?
        BRNZ    valid,          -- Branch to valid if true
        EXIT,

valid   LDC     0,              -- Load Constant 0
        STH     State,          -- State := 0
        ...
```

◊ Répétition d'événements

Pour réaliser la répétition d'événements, l'automate utilise une variable interne comme compteur de répétition.

L'expression A THEN (8 * B[31:0] (0 TO 1)) sera réalisée par l'élément fonctionnel suivant :

```
INPUT LIST :
        A, B[31:0];
VARIABLE LIST :
        State, Counter;
INSTRUCTION LIST :
        LDH     State,          -- Load State
        BRT     3,              -- Branch according to current state
        state0,                 -- Label of state0
        state1,                 -- Label of state1
        state2,                 -- Label of state2

state0  LDIW    A,              -- Load Source Wire A
        FRTO    ? TO ?  1,      -- Any change on A ?
        BRZ     label1,         -- Branch to label1 if false
        LDC     1,              -- Load Constant 1
        STH     State,          -- State := 1
        LDC     8,              -- Load Constant 8
        STH     Counter,        -- Counter := 8
label1  EXIT,

state1  LDIW    B[31:0],        -- Load Source Wire B [31:0]
        FRTO    0 TO 1  32,     -- Any change on B ?
        BRZ     label2,         -- Branch to label2 if false
        LDH     Counter,        -- Load Source Counter
        ADC     -1,             -- Do Counter -1
        STH     Counter         -- Counter := Counter -1
        BRZ     valid,          -- Branch to valid if 0
label2  EXIT,

valid   LDC     0,              -- Load Constant 0
        STH     State,          -- State := 0
        ...
```

◊ Retardement d'événement

Le retardement nécessite l'instruction SEND qui génère un événement avec un retard spécifique, comme le montre l'exemple suivant :

A WAIT 100

```
INPUT LIST :
        A, Wait;
VARIABLE LIST :
        State;
INSTRUCTION LIST :
        LDH    State,          -- Load State
        BRT    2,              -- Branch according to current state
        state0,                -- Label of state0
        state1,                -- Label of state1

state0  LDIW   A,              -- Load Source Wire A
        FRTO   ? TO ? 1,       -- Any change on A ?
        BRZ    label1,         -- Branch to label1 if false
        LDC    100,            -- Load Constant 100
        LDIW   Wait,           -- Load Destination Wire Wait
        SEND   .EVENT,         -- Send event (current time + 100, Wait)
        LDC    1,              -- Load Constant 1
        STH    State,          -- State := 1
label1  EXIT

state1  LDIW   Wait,           -- Load Source Wire Wait
        FRTO   ? TO ? 1,       -- Any change on Wait ?
        BRNZ   valid,          -- Branch to valid if true
        EXIT,

valid   LDC    0,              -- Load Constant 0
        STH    State,          -- State := 0
        ...
```

◊ Détection d'événements hors séquence

L'opération RESET nécessite l'utilisation de deux éléments fonctionnels pour réaliser l'expression d'événements :

sous-expression d'événements 1 RESET sous-expression d'événements 2

Car les deux sous-expressions d'événements s'évaluent en parallèle : pendant l'évaluation de la première sous-expression, si la deuxième sous-expression devient valide, alors l'ensemble de l'expression cesse de s'évaluer, et retourne au point de départ de l'évaluation.

L'élément fonctionnel qui réalise la première sous-expression est le suivant :

```
INPUT LIST :
        ...
OUTPUT LIST :
        ...
        Reset;
VARIABLE LIST :
        ...
        State;
INSTRUCTION LIST :
        ...

valid   ...
        LDIW   Reset,          -- Load Source Wire Reset
        SEND   .EVENT,         -- Send event (current time, Reset)
        ...
```

L'élément fonctionnel qui réalise la deuxième sous-expression et la fonction RESET est le suivant :

```
INPUT LIST :
        ...
        Reset;
OUTPUT LIST :
        ...
VARIABLE LIST :
        ...
INSTRUCTION LIST :
        LDIW   Reset,          -- Load Source Wire Reset
        FRTO   ? TO ? 1        -- Any change on Reset
        BRZ    Start,          -- Branch to start if false
        LDC    0,              -- Load Constant 0
        STH    State,          -- State := 0

start   LDH    state,          -- Load State
        BRT    n,              -- Branch according current state
        state0,
```

```
        state1,
        ...
        staten-1,
        ...
```

◊ Intégration de l'expression d'événements dans les structures gardées

Les expressions d'événements s'utilisent dans les structures gardées telles que :

```
WHEN expression d'événements DO liste d'actions;
NEXT  expression d'événements DO liste d'actions;
```

Dans les éléments fonctionnels, les listes d'actions sont réalisées par des instructions spécifiques qui suivent les expressions d'événements :

```
INPUT LIST :
        ...
OUTPUT LIST :
        ...
VARIABLE LIST :
        ...
INSTRUCTION LIST :
        ...
        (event expression)              -- Evaluate event expression (automaton)

valid   (reset automaton state)         -- Reset automaton if expression is valid
        (action list)                   -- Start action list
```

5.4. Modélisation des équipotentielles

5.4.1. Modèles des équipotentielles

Le simulateur ne dispose pas de notion de type d'équipotentielle, pour des raisons d'efficacité. Par conséquent, la modélisation des équipotentielles est une des tâches les plus importantes de la compilation. Cette modélisation consiste à typer les équipotentielles en leur associant des attributs distincts.

Les attributs d'une équipotentielle sont déterminés par son type déclaré et par son contexte d'utilisation :

• Si l'équipotentielle est explicitement déclarée avec un type sans ambiguïté, alors les attributs de l'équipotentielle peuvent être déterminés pendant la compilation. Ces attributs seront vérifiés et modifiés d'une manière dynamique lors de la configuration et de l'édition de liens.

• Si l'équipotentielle est déclarée avec un type ambigu ou qu'elle n'est pas déclarée, alors ses attributs dépendent entièrement du contexte de l'utilisation. Les attributs ne sont pas déterminables lors de la compilation. Ceux qui manquent seront complétés pendant la configuration et l'édition de liens.

Les principaux attributs d'une équipotentielle sont les suivants :

• Le type de l'équipotentielle est utilisé par la vérification sémantique tout au long de la compilation, de la configuration et de l'édition de liens.

• La classe de l'équipotentielle indique si l'équipotentielle est interne au circuit, externe comme un paramètre ou absolue comme un objet global.

• La fonction logique est définie, s'il y a une fonction câblée ou une valeur logique associée à l'équipotentielle, ou qu'il existe une fonction intrinsèque de l'équipotentielle.

• Les entrées de l'équipotentielle sont les sorties des éléments unidirectionnels ou les ports bidirectionnels des transistors connectés à l'équipotentielle.

• Les sorties de l'équipotentielle sont les entrées des éléments unidirectionnels ou les ports unidirectionnels des transistors connectés à l'équipotentielle.

• Les retards et la capacité définissent les propriétés temporelles de l'équipotentielle.

En fonction de ces attributs, l'équipotentielle sera représentée, au moment de la génération de code, par une des configurations suivantes :

- une simple équipotentielle;
- un élément et des équipotentielles associées;
- deux éléments et des équipotentielles associées.

Un élément sera ajouté à la place de l'équipotentielle dans un des cas suivants :

- L'équipotentielle est connectée à plusieurs sorties d'éléments.
- L'équipotentielle est à trois états.
- L'équipotentielle est affectée d'une fonction câblée.
- L'équipotentielle a des retards non nuls.

Deux éléments seront utilisés à la place de l'équipotentielle :

- L'équipotentielle est connectée aux ports sources/drains de transistors.
- L'équipotentielle est affectée d'une fonction câblée et liée à la logique à trois états.

5.4.2. Types des équipotentielles

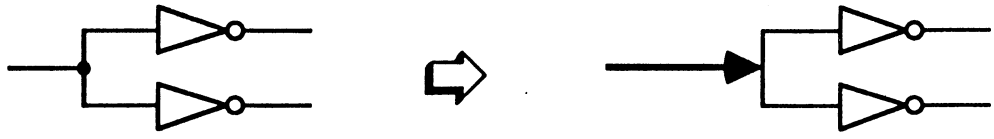
◊ Définition des types

Dans le compilateur, le configurateur et l'éditeur de liens, un ensemble de types sont définis pour décrire la nature et les propriétés intrinsèques des équipotentielles.

Le type N (*null*) représente une équipotentielle qui n'est connectée à aucun élément.

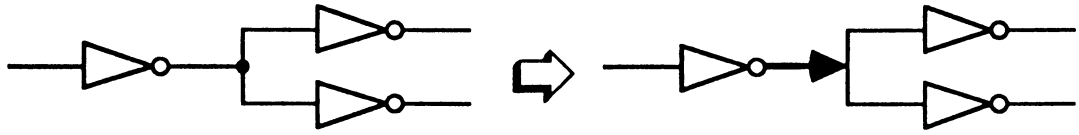
Type N

Le type P (*passive*) représente une équipotentielle qui n'est connectée qu'à une ou plusieurs entrées d'éléments.



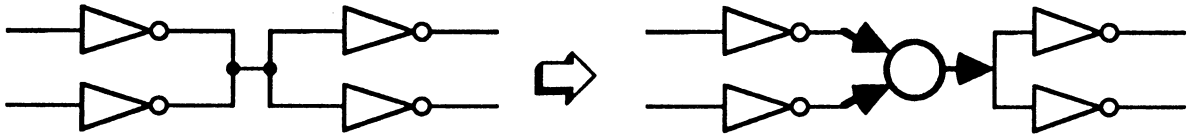
Type P

Le type D (*driven*) représente une équipotentielle connectée à une seule sortie d'élément.



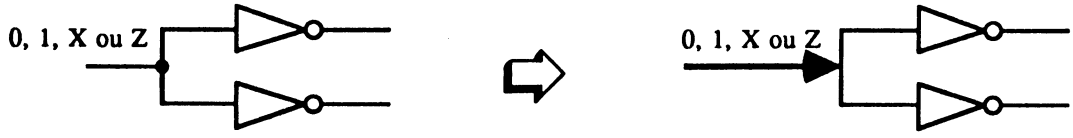
Type D

Le type MD (*multi-driven*) représente une équipotentielle connectée à plusieurs sorties d'éléments.



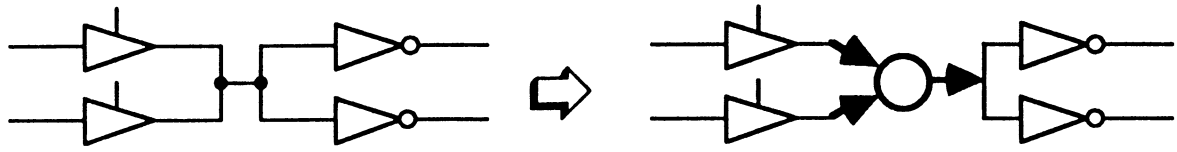
Type MD

Le type V (*valued*) représente une équipotentielle affectée d'une valeur logique comme 0, 1, X et Z.



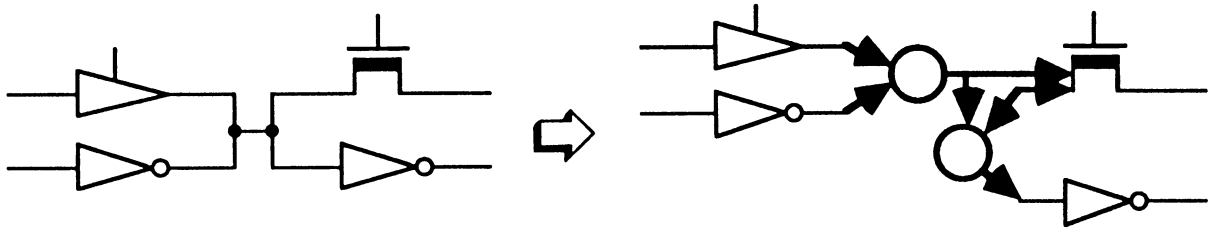
Type V

Le type DT (*driven-tri*) représente une équipotentielle à trois états mais sans connexion aux ports sources/drains de transistors.



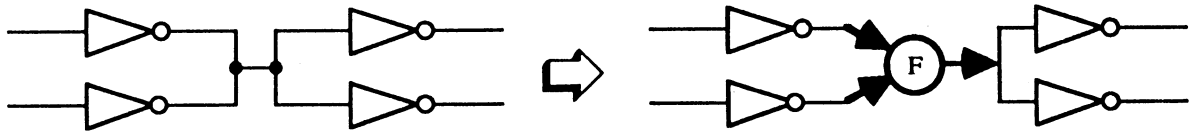
Type DT

Le type T (*tri*) représente une équipotentielle à trois états connectée à des entrées et sorties d'éléments et à des ports sources/drains de transistors.



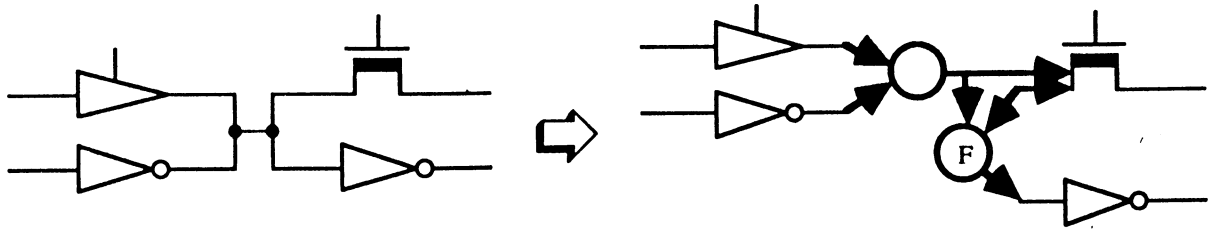
Type T

Le type W (*wired*) représente une équipotentielle affectée d'une fonction câblée.



Type W

Le type WT (*wired-tri*) représente une équipotentielle affectée d'une fonction câblée et connectée à des ports sources/drains de transistors.



Type WT

◊ Détermination des types d'équipotentielles

Il existe une correspondance entre les types d'équipotentielles de Hilo-3 et ceux de LL3T :

- INPUT : N, P, D, MD, V, DT, T, W et WT;
- INPUT0 : N, P, D, MD, V, DT, T, W et WT;
- INPUT1 : N, P, D, MD, V, DT, T, W et WT;
- SUPPLY0 : V;
- SUPPLY1 : V;
- TRI : DT, T et WT;
- TRI0 : T et WT;
- TRI1 : T et WT;
- TRIREG : T et WT;
- WIRE : N, P, D, MD, V, DT, T, W et WT;
- WAND : W et WT;
- WOR : W et WT.

Par ailleurs, la nature des connexions de l'équipotentielle détermine également le type de l'équipotentielle. On distingue les différentes connexions suivantes :

- les entrées d'éléments;
- les sorties d'éléments sauf celles des portes à trois états;
- les valeurs logiques;
- les sorties des portes à trois états;
- les ports sources/drains de transistors.

Le type d'une équipotentielle peut être transformé en un autre selon la nature de connexions des éléments reliés à cette équipotentielle :

connexion \ type	N	P	D	MD	V	DT	T	W	WT
entrée d'élément	P	P	D	MD	V	DT	T	W	WT
sortie d'élément	D	D	MD	MD	MD	DT	T	W	WT
valeur logique	V	V	MD	MD	V	DT	T	W	WT
sortie de porte à 3 états	DT	DT	DT	DT	DT	DT	T	WT	WT
source/drain de transistor	T	T	T	T	T	T	T	WT	WT

Transformation des types selon la nature de connexion

5.4.3. Hiérarchisation des modèles

La configuration et l'édition de liens prennent en compte la hiérarchie

des circuits.

Pour que les modèles d'équipotentielles puissent être utilisés dans une représentation hiérarchique de circuits, les problèmes suivants doivent être résolus :

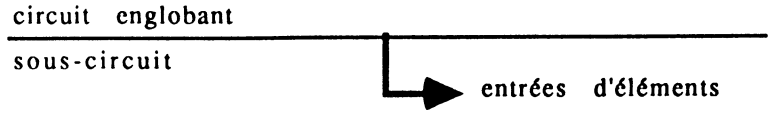
• la manière de lier l'équipotentielle au port de connexion de l'interface du sous-circuit;
• la compatibilité entre l'équipotentielle du circuit englobant et celle du sous-circuit.

La manière de lier l'équipotentielle au port de connexion dépend des attributs de l'équipotentielle. Cela consiste en :

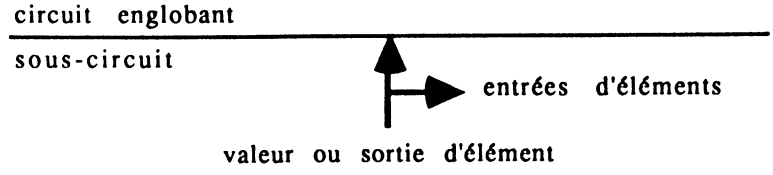
• le nombre d'équipotentielles réellement utilisées pour représenter l'équipotentielle original connectée au port de connexion;
• la direction de chaque équipotentielle;
• les attributs à propager par le port de connexion entre le sous-circuit et le circuit englobant.

Le port de connexion peut être sous une des quatre formes suivantes :

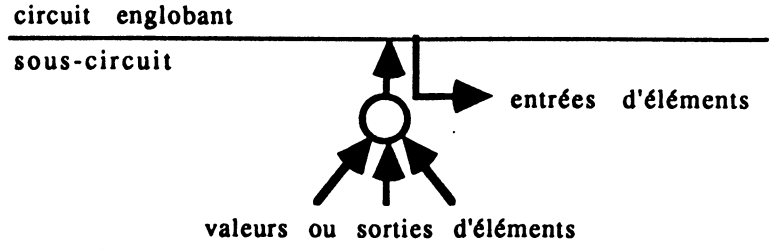
• entrée (les types N et P) :



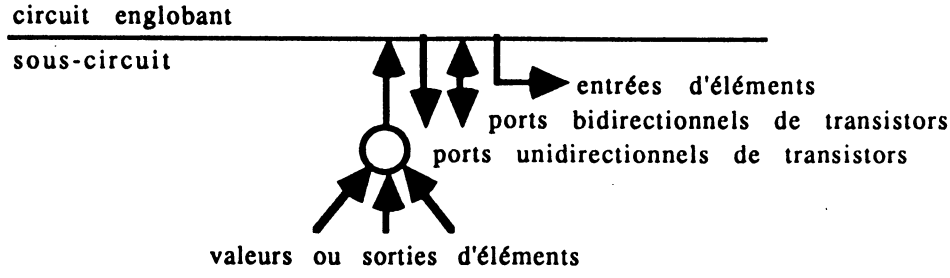
• sortie (les types D et V) :



• sortie-entrée (les types MD, DT et W) :



• sortie-unidirectionnelle-bidirectionnelle-entrée (les types T et WT) :



Lors de la configuration et l'édition de liens, les interfaces entre les sous-circuits et leurs circuits englobants sont prises en compte pour assurer les opérations suivantes :

• L'équipotentielle du sous-circuit et celle du circuit englobant reliées par le port de connexion de l'interface doivent être compatibles.

• Les attributs d'une équipotentielle peuvent se propager, à travers le port de connexion, à l'autre équipotentielle, pour permettre de vérifier la cohérence entre les équipotentielles, et de compléter les informations qui manquent. Les attributs se propagent des circuits englobants vers les sous-circuits lors de la phase de configuration, et des sous-circuits vers les circuits englobants lors de la phase d'édition de liens.

• Par conséquent, le type d'une équipotentielle peut se convertir dynamiquement en fonction des attributs qui lui sont propagés, pendant les phases de configuration et d'édition de liens.

La compatibilité entre les types d'équipotentielles de circuits englobants et ceux de sous-circuits est définie d'une manière simplifiée par la table suivante :

sous-circuit \ circuit	N	P	D	MD	V	DT	T	W	WT
N	√	√	√	√	√	√	√	√	√
P	√	√	√	√	√	√	√	√	√
D	√	√	–	√	–	√	√	√	√
MD	√	√	√	√	√	√	√	√	√
V	√	√	–	√	!	√	√	√	√
DT	√	√	√	√	√	!	!	–	!
T	√	√	√	√	√	!	!	–	!
W	√	√	√	–	√	–	–	!	!
WT	√	√	√	–	√	–	–	!	!

√ : compatible
– : impossible
! : nécessiter une vérification

Compatibilité des types

Il convient de signaler que l'ensemble des traitements liés aux types d'équipotentielles sont relativement complexes :

• Statiquement, la compilation n'est pas en mesure de déterminer les types de toutes les équipotentielles, à cause des équipotentielles non déclarées, ou par manque du contexte d'utilisation de celles-ci.

• Dynamiquement, la configuration et l'édition de liens doivent se charger de compléter, de vérifier et de convertir les types d'équipotentielles en propageant des attributs des équipotentielles entre les circuits, au lieu d'examiner uniquement la compatibilité des types d'équipotentielles.

Cette complexité se manifeste à deux niveaux :

• Au niveau du circuit englobant, une équipotentielle peut être connectée aux sous-circuits qui ne sont pris en compte qu'au moment de l'édition de liens. Le type de l'équipotentielle peut être modifié ou converti dynamiquement en fonction des attributs en provenance des sous-circuits, de sorte que le type de l'équipotentielle soit compatible à celui des différents ports de connexion des sous-circuits. Cette compatibilité contient plusieurs aspects : la nature des connexions, le nombre de connexions sur le port, la

direction des signaux, la fonction ou la valeur inhérentes à l'équipotentielle, etc.

• Au niveau du sous-circuit, il se peut que celui-ci soit partagé par plusieurs circuits englobants. Dans ce cas, les relations entre les circuits englobants et les sous-circuits constituent une structure hiérarchique, mais non arborescente. Cette nature exige que le même sous-circuit doive s'adapter à la fois aux contextes de tous les circuits englobants. Cette adaptation se fait notamment au niveau des équipotentielles qui réalisent les interconnexions entre le sous-circuit et le monde extérieur. Chacune de ces équipotentielles doit avoir un type unique qui est compatible avec les ports de connexions des différents circuits englobants, même si ces ports de connexion ont des propriétés très différentes les uns et les autres. Le choix de ce type unique qui s'adapte à ces différents contextes d'utilisation est ainsi une tâche sophistiquée pour les outils de compilation.

5.5. Réalisation de la simulation de pannes

5.5.1. Structure générale

La simulation de pannes repose sur la méthode concurrente qui facilite l'intégration de différents modèles de pannes dans l'algorithme de l'échéancier, et qui assure l'utilisation des éléments fonctionnels.

Le parallélisme est réalisé par la répartition de l'ensemble des classes de pannes sur les unités de simulation. Chaque unité de simulation est affectée d'une description complète du circuit à simuler avec des classes de pannes distinctes.

L'implémentation de la simulation de pannes nécessite les fonctions suivantes :

- l'analyses des pannes;
- la distribution des classes de pannes;
- la simulation des pannes;
- la collecte des résultats;
- la gestion des pannes.

L'analyse des pannes consiste en :

- l'analyse de la description du circuit et des hypothèses de pannes qui sont définies par l'environnement de simulation;
- la création des classes de pannes qui représentent chacune un ensemble de pannes équivalentes.

La distribution des classes de pannes consiste à distribuer les pannes sur les unités de simulation, en fonction de :

- la configuration matérielle;
- le nombre de classes de pannes.

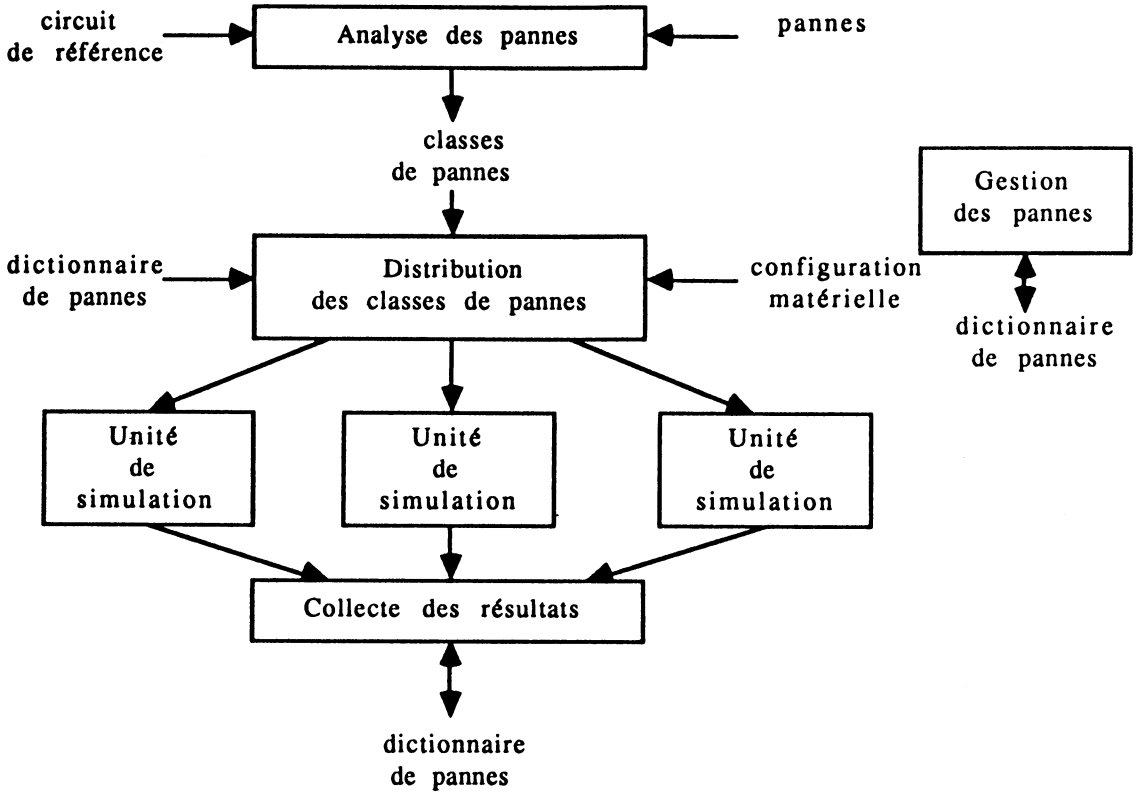
La collecte des résultats consiste à :

- recevoir les informations en provenance des unités de simulation;
- analyser ces informations à l'aide du dictionnaire de pannes pour déterminer les pannes détectées ou non détectées.

La gestion des pannes réalise :

- la gestion du dictionnaire de pannes;
- la gestion des informations des pannes détectées ou non détectées;
- la gestion des stimuli de test et de leurs historiques.

L'ensemble de ces fonctions s'organisent de la façon suivante :



Structure générale

5.5.2. Simulateur

◊ Structure du simulateur

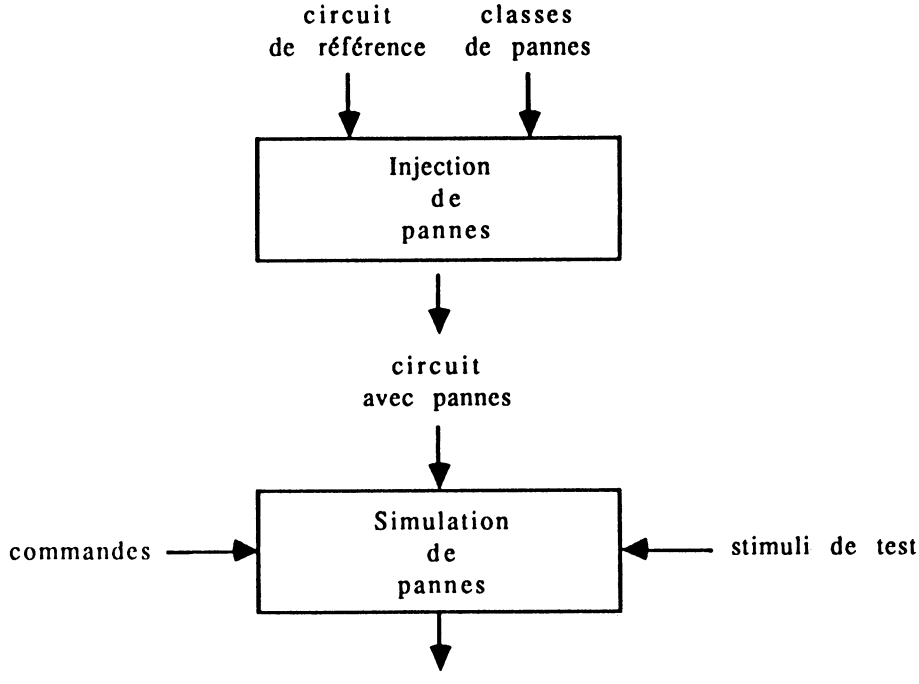
A chaque unité de simulation sont associés les quatre types d'informations d'entrée suivants :

- le circuit de référence;
- les classes de pannes;

- les stimuli de test;
- les commandes.

Une unité de simulation se compose de deux processus :

- l'injection de pannes qui transforme, en utilisant les classes de pannes, le circuit en circuits erronés;
- la simulation du circuit avec les pannes.



Unité de simulation

◊ Algorithmes

L'algorithme de la simulation normale peut être utilisé directement par le noyau du simulateur pour effectuer la simulation de pannes.

Dans le cas de N classes de pannes, N+1 circuits doivent être simulés. Le distributeur de pannes distribue statiquement ou dynamiquement les pannes sur les unités de simulation.

En réalité cet algorithme est inexploitable car le nombre de classes peut être très élevé.

La méthode parallèle et la méthode déductive ne conviennent ni à la complexité du problème à résoudre ni à la complexité de l'algorithme déjà

implémenté dans le simulateur. La méthode concurrente, qui assure l'utilisation de différents types de modélisation de circuit et la compatibilité avec l'algorithme de l'échéancier, tout en améliorant considérablement les performances de simulation, est choisie pour implémenter le noyau du simulateur.

Une autre amélioration consiste à coder, en reposant sur la méthode concurrente, plusieurs états logiques sur un seul mot machine. Cette mesure permet de procurer les avantages de la méthode parallèle en utilisant la méthode concurrente comme la base de l'algorithme.

L'ensemble des logiciels de la simulation de pannes est en cours de dévoloppement.

Conclusion

Le partitionnement du circuit reste un des problèmes critiques du pré-traitement de la simulation parallèle. Différentes méthodes ont été étudiées. Les statistiques ont montré les faits suivants :

- La méthode du partitionnement par la hiérarchie et la méthode linéaire donnent des résultats satisfaisants sur les circuits hiérarchiques et réguliers.
- Les méthodes par la connectivité d'entrées et de sortie coûtent plus cher au niveau du temps de calcul mais fonctionnent sur toutes sortes de circuits.

L'intégration des modèles de transistors dans la simulation logique est réalisée par les mesures suivantes :

- l'utilisation du modèle de transistor à cinq ports, et l'utilisation des traitements spécifiques;
- la représentation d'une équipotentielle complexe par plusieurs équipotentielles, chacune de celles-ci jouant un rôle unique;
- l'ajout des éléments TRI pour gérer la logique à trois états;
- la réalisation d'un algorithme d'évaluation adapté au mécanisme de l'échéancier et à la simulation parallèle.

La simulation fonctionnelle repose sur l'utilisation d'éléments fonctionnels dont le comportement est décrit par une liste d'instructions. Cette méthode permet de gérer et d'évaluer les portes logiques et les éléments fonctionnels d'une manière homogène. Une machine virtuelle est implémentée pour interpréter les instructions fonctionnelles. Le pré-traitement compile les modèles fonctionnels et les convertit en éléments ayant une liste d'instructions, ce qui permet l'utilisation de tous les types de modèles fonctionnels, combinatoires ou séquentiels, sans aucune restriction. Sémantiquement, l'algorithme du simulateur est relativement indépendant des modèles fonctionnels.

Dans LL3T, la gestion et l'évaluation des équipotentielles ont les caractéristiques suivantes :

- Contrairement à certains algorithmes, le noyau de simulation n'évalue pas les équipotentielles d'une manière systématique pour éviter les

calculs inutiles.

• Au moment du pré-traitement, des éléments et des équipotentielles complémentaires sont ajoutés dans le circuit, à la place des équipotentielles qui nécessitent une évaluation.

• Normalement dans un simulateur distribué, la gestion des équipotentielles globales qui sont définies dans plusieurs unités de simulation nécessite des traitements spéciaux pour assurer la cohérence et l'uniformité de l'état de chaque équipotentielle globale dans les différentes unités de simulation. Ce problème est contourné par une mesure qui consiste à représenter une équipotentielle globale par des interconnexions unidirectionnelles, qui lient les entrées et les sorties des éléments situés dans différentes unités de simulation.

En ce qui concerne la simulation de pannes, le parallélisme consiste à distribuer les classes de pannes sur les unités de simulation. La méthode concurrente assure l'intégration des modèles fonctionnels et de transistors dans l'algorithme de l'échéancier. De plus, la combinaison de cette méthode avec la méthode parallèle permet au simulateur d'avoir de meilleures performances.

Chapitre 6

Conclusion générale

Plan du chapitre 6

6.1. Evaluation des performances

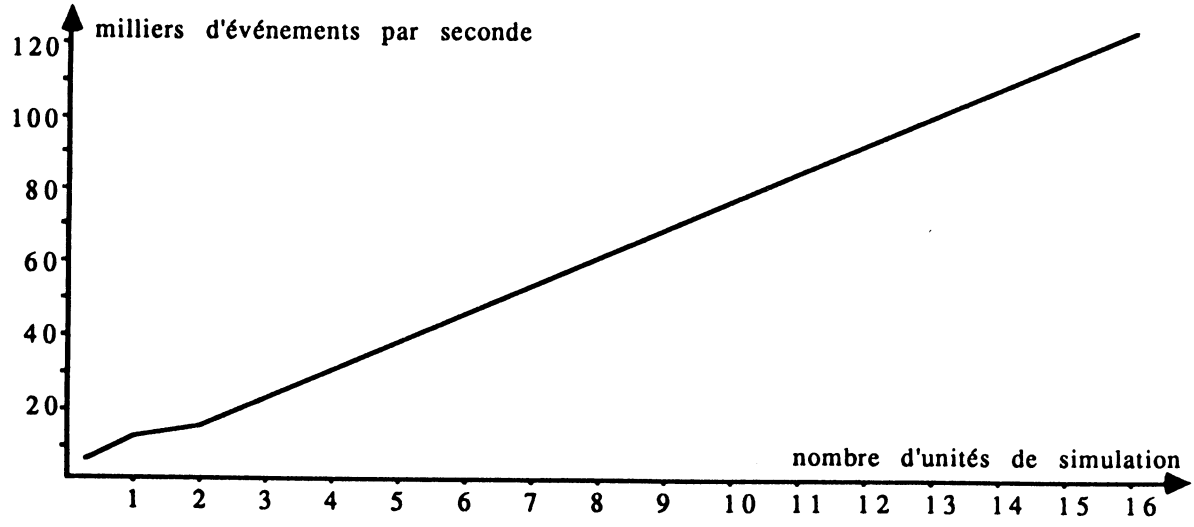
6.1.1. Performances des versions expérimentales

Dans le cadre du projet Esprit, des versions expérimentales du simulateur ont été réalisées.

Les statistiques sur ces versions montrent les faits suivants :

- La configuration minimale de LL3T comprend un seul transputer qui réalise les trois tâches concurrentes : ECT, EMT et DMT. La vitesse testée est d'environ 6000 événements par seconde.
- L'utilisation de trois transputers constitue une unité de simulation qui permet de doubler cette vitesse de simulation, soit 12000 événements par seconde.
- Lorsque plusieurs unités de simulation sont introduites dans LL3T, elles s'organisent en anneau. Des mécanismes de synchronisation et de communication sont utilisés qui provoquent une diminution de 30% de performances de l'ensemble du simulateur.
- En plus, à partir de deux unités de simulation, chaque unité de simulation a une perte de 10% de ses performances à cause notamment des communications et des synchronisations supplémentaires.

Par conséquent, les performances estimées doivent être les suivantes :



La configuration maximale permet une simulation à la vitesse de $9,5 \cdot 10^5$ événements par seconde.

6.1.2. Performances des versions commerciales

Le développement de LL3T a aboutit à des versions commerciales.

Ces versions commerciales sont :

- compatibles 100 % avec le système Hilo-3;
- utilisables dans les ordinateurs individuels et les stations de travail.

La version LL3T/1T est basée sur une carte de 1 transputer et de 4 ou 8 méga-octets de mémoire. Elle a les caractéristiques suivantes :

- une capacité de simuler des circuits de 100000 portes;
- une puissance équivalente à 4,38 Micro Vax II, 1,84 Sun 3/260 ou 0,80 Vax 8600.

La version LL3T/3T est basée sur une carte configurée de 3 transputers et de 12 ou 20 méga-octets de mémoire. Cette version a les caractéristiques suivantes :

- une capacité de simuler des circuits de 250000 portes;
- une puissance équivalente à 6,27 Micro Vax II, 2,63 Sun 3/260 ou 1,14 Vax 8600.

L'ensemble de LL3T est programmé en Occam 2, à l'exception de la partie serveur qui est programmée en Pascal ou C. La version actuelle comprend plus de 150000 lignes source.

6.2. Discussions

6.2.1. Considérations sur LL3T

Le développement de LL3T a permis la réalisation d'un système de simulation opérationnel et hautement parallèle, ayant les caractéristiques suivantes :

• Un accélérateur puissant et économique est réalisé à base de microprocesseurs généraux. Il convient de signaler que la réalisation matérielle de l'accélérateur est relativement rapide et peu coûteuse. L'utilisation des microprocesseurs généraux permet d'une part de bénéficier de la puissance de processeurs performants, d'autre part d'avoir la possibilité d'intégrer dans l'accélérateur les futurs processeurs basés sur les dernières technologies qui progressent constamment. Sans limitation matérielle ni architecturale, les algorithmes peuvent être améliorés et optimisés sans difficulté, tout en suivant l'évolution des méthodologies de la simulation. En outre, non seulement le simulateur, mais aussi nombreux outils auxiliaires ont été implémentés sur l'accélérateur lui-même.

• Le parallélisme massif aux niveaux matériel (de 3 à 384 transputers) et logiciel (la programmation concurrente et multi-tâche) est introduit dans LL3T. L'apparition du transputer offre une nouvelle philosophie pour mieux exploiter le potentiel de la technologie VLSI, qui consiste à réaliser un système complexe par l'utilisation des composants identiques et programmables. Ces composants peuvent être interconnectés directement par leurs liens de communication qui assurent une haute vitesse de communication. Le langage Occam 2 est basé sur un ensemble de concepts tels que le processus et le canal de communication qui permettent de programmer des systèmes distribués et parallèles d'une manière efficace. La simulation normale et la simulation de pannes utilisent deux stratégies distinctes pour introduire le parallélisme dans leur algorithme : le partitionnement du circuit et la répartition des classes de pannes sur les unités de simulation. D'autres mesures ont été prises pour améliorer les performances du simulateur : les processeurs de chaque unité de simulation s'organisent en pipeline; la partie critique du noyau du simulateur est écrite en assembleur afin de réduire le nombre d'instructions.

• LL3T est configurable en fonction des besoins d'utilisation. La configuration maximale est constituée de centaines de microprocesseurs pour exploiter d'une manière massive le parallélisme. La configuration minimale contient quelques microprocesseurs pour être incorporée dans un ordinateur

individuel.

• La modélisation du circuit est basée sur plusieurs niveaux d'abstraction : porte logique, fonctionnel et interrupteur. L'évaluation de ces différents modèles est réalisée par différents algorithmes. Par contre, ces algorithmes sont tous basés sur le mécanisme de l'échéancier, qui assure la compatibilité totale entre ces modèles ainsi que l'homogénéité du noyau du simulateur. Du côté de l'interface utilisateur, la description du circuit est modulaire, générique et hiérarchique. Différents styles, structurel et comportemental, peuvent s'utiliser conjointement dans la modélisation du circuit.

• L'algorithme de simulation supporte différents modes de simulation, en fonction de la manière de discrétisation du temps et de modélisation des signaux.

• La simulation normale et la simulation de pannes fonctionnent sur les mêmes descriptions de circuits et de stimuli-réponses. Les outils de compilation génèrent des données internes qui sont totalement compatibles pour les deux types de simulations. La méthode concurrente de la simulation de pannes n'est pas interférente avec le mécanisme de l'échéancier du noyau du simulateur.

• Dans l'accélérateur, un processeur joue un rôle spécifique, qui consiste à gérer les données de résultats et à superviser la simulation, ce qui donne au simulateur un caractère très interactif. D'une part, pendant la simulation il est possible d'éditer et d'analyser les résultats partiels. D'autre part, les interactions ne donnent aucune influence sur la vitesse de simulation. En cas de nécessité, l'utilisateur peut toujours intervenir à temps, pour vérifier et contrôler le déroulement de la simulation.

• Les outils de pré-traitements et de post-traitements sont développés autour de la simulation, qui constituent un environnement de simulation. Cet environnement améliore l'utilité de l'ensemble des outils et facilite sensiblement les tâches de simulation. La réalisation de certaines fonctions telles que les opérations de rétro-annotation rend l'environnement de simulation plus opérationnel.

• L'environnement de simulation ainsi constitué repose également sur une base de données et des bibliothèques externes, pour assurer une gestion automatique des données de source, intermédiaires et de résultat. Il faut noter que les accélérateurs purs ne supportent que la simulation et nécessitent d'autres machines pour les pré-traitements, les post-traitements, et la gestion de données. Tandis que l'environnement de simulation LL3T est entièrement implémenté sur l'accélérateur.

6.2.2. Quelques réflexions

Le développement de LL3T donne matière à des réflexions, notamment sur les aspects suivants :

• Les problèmes clefs de la parallélisation de la simulation normale sont la synchronisation entre les processeurs et le partitionnement du circuit à simuler. Ces problèmes deviennent d'autant plus critiques que le parallélisme est massif. La synchronisation entre les processeurs ayant peu d'activité et ceux ayant trop de charges conduit à une dégradation globale des performances. En ce qui concerne le partitionnement du circuit, il est important d'équilibrer les charges statiques et dynamiques des unités de simulation. D'autre part, la minimisation des distances logiques de communication inter-processeur peut être réalisée par des analyses statiques sur les connexions entre les partitions. Une autre future amélioration consisterait à introduire dans le simulateur le mécanisme d'anti-événements qui permet la simulation distribuée et asynchrone.

• L'introduction des éléments fonctionnels et des transistors dans l'algorithme de l'échéancier a soulevé des problèmes sophistiqués. La modélisation fonctionnelle représente plusieurs degrés d'abstraction et amène de nouveaux concepts. Dans ce domaine, la solution adoptée par LL3T est une réussite. Elle permet de réaliser, d'une manière générale, toutes les fonctionnalités nécessaires (les opérations séquentielles, parallèles et temporelles), ainsi que les mécanismes de contrôle (dirigé par le flot de données ou par le flot de contrôle) des différents modèles fonctionnels. Cette solution autorise par ailleurs l'utilisation des échéanciers distribués. Cependant, comme toutes les implémentations fonctionnelles, elle exige l'algorithme de double phase, qui rend plus coûteuse la synchronisation inter-processeur. Quant aux modèles de transistors, ils sont intégrés dans l'algorithme et considérés comme des éléments structurels. Parmi les nombreuses propriétés de transistors, tels que la nature bidirectionnelle, l'effet d'atténuation de signal, la mémorisation dynamique et le rapport de résistances, la première constitue un obstacle important de l'implémentation des modèles de transistors. La solution adoptée par LL3T a pour but d'intégrer ces modèles dans l'algorithme de l'échéancier et de paralléliser la simulation sans contraintes algorithmiques, bien que les résultats de la simulation sont plus pessimistes que ceux des méthodes de graphe et d'itération qui sont essentiellement utilisées par des simulateurs d'interrupteur purs.

• Il est non négligeable que les pré-traitements, la gestion de données

et les post-traitements nécessitent un temps de traitement relativement important. Un accélérateur de simulation pur n'est pas entièrement opérationnel pour répondre aux besoins des applications de CAO de circuits intégrés. Quoique l'environnement de simulation LL3T, qui intègre des logiciels de pré-traitements, de post-traitements et de gestion de données, soit implémenté sur l'accélérateur, le potentiel de la puissance matérielle de l'accélérateur n'est exploité que par le noyau du simulateur actuellement. Il reste un problème non résolu de pouvoir tirer parti de la puissance de traitement d'une architecture hautement parallèle pour les pré-traitements, tels que la compilation qui est très répétitive lors de la mise au point de la conception de circuit. La compilation pourrait devenir dans certains cas un nouveau goulot d'étranglement des processus de simulation.

• Il est constaté que le choix des langages de description influe non seulement sur l'implémentation des outils de compilation, mais aussi sur la réalisation du simulateur. La sémantique des langages proches des propriétés physiques de circuits intégrés permet de réaliser un simulateur efficace dans un domaine spécifique. Cependant elle oblige le simulateur à s'orienter vers des fonctionnalités particulières définies par les langages. Il est difficile dans ce cas de réaliser un simulateur neutre. Car d'une part, les performances de la simulation ne seront pas assurées lorsqu'il y a un écart important entre la sémantique des langages et celle du simulateur, d'autre part la compilation n'est pas toujours en mesure de combler cet écart sémantique. En plus, la simulation est une méthodologie déterministe qui représente des phénomènes physiques pouvant être non déterministes. La correspondance entre le déterminisme et le non déterminisme peut être définie dans les langages et dans le noyau de simulation de manières différentes, qui complique les relations trilatérales entre les langages, l'implémentation du simulateur et les outils de compilation. Les expériences ont bien prouvé que les influences sémantiques des langages donnent des effets négatifs sur la flexibilité du simulateur, ce qui s'est manifesté d'ailleurs au moment où nous essayions d'implémenter d'autres langages sur LL3T.

Références bibliographiques

[ABR 77] M. ABRAMOVICI, M.A. BREUER, K. KUMAR
"Concurrent Fault Simulation and Functional Level Modeling"
14th Design Automation Conference, June 1977, pp. 128 - 137.

[ABR 83] M. ABRAMOVICI, Y.H. LEVENDEL, P.R. MENON
"A Logic Simulation Machine"
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems Vol. CAD-2, No. 2, April 1983, pp. 82 - 94.

[ABR 86] J.A. ABRAHAM, W.K. FUCHS
"Fault and Error Models for VLSI"
Proceedings of the IEEE, Vol. 74, No. 5, May 1986, pp. 639 - 654.

[ABR 88a] S.G. ABRAHAM
"Parallel Simulation of Shared Memory Multiprocessors"
Proceedings of the Third International Conference on Supercomputing, 1988, Volume III, pp. 313 - 322.

[ABR 88b] D. ABRAMSON
"Using a Dataflow Multiprocessor for Functional Logic Simulator"
Proceedings of the Third International Conference on Supercomputing, 1988, Volume III, pp. 288 - 297.

[AGR 87a] P. AGRAWAL, W. DALLY, R. TUTUNDJIAN
"Logic Simulation Algorithms for Pipelined Hardware Architectures"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 90 - 99.

[AGR 87b] P. AGRAWAL et al.
"MARS : A Multiprocessor-Based Programmable Accelerator"
IEEE Design & Test of Computers, October 1987, pp. 28 - 36.

[AGR 90] P. AGRAWAL, W.J. DALLY
"A Hardware Logic Simulation System"
IEEE Transactions on Computer-Aided Design, Vol. 9, No. 1, January 1990, pp. 19 - 29.

[ANS 85] Y. ANSADE, J.P. BERNARD, G. MARAZE
"A Highly Parallel Architecture Dedicated to Logical Simulation"
International Symposium on VLSI Technology, Systems and Applications,1985, pp. 71 - 75.

[ANS 87] Y. ANSADE, R. CORNU-EMIEUX, G. MARAZE, P. OBJOIS
"Highly Parallel Logic Simulation Accelerators based upon Distributed Discrete-Event Simulation"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 69 - 79.

[ANT 87] K.J. ANTREICH, M.H. SCHULZ
"Accelerated Fault Simulation and Fault Grading in Combinational Circuits"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 5, September 1987, pp. 704 - 711.

[ARM 72] D.B. ARMSTRONG
"A Deductive Method for Simulating Faults in Logic Circuits"
IEEE Transactions on Computer, Vol. C-21, No.5, May 1972, pp. 464 - 471.

[BAR 85] M.R. BARBACCI, T. UEHARA
"Computer Hardware Description Languages : The Bridge Between Software and Hardware"
Computer, February 1985, pp. 6 - 8.

[BAR 87] Z. BARZILAI, J.L. CARTER, B.K. ROSEN, J.D. RUTLEDGE
"HSS - A High-Speed Simulator"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 4, July 1987, pp. 601 - 617.

[BAR 88a] D.L. BARTON
"Behavioral Description in VHDL"
VLSI Systems Design, June 1988, pp. 28 - 33.

[BAR 88b] Z. BARZILAI et al.
"SLS - A Fast Switch-Level Simulator"
IEEE Transactions on Computer-Aided Design, Vol. 7, No. 8, August 1988, pp. 838 - 849.

[BEE 88] D.K. BEECE, G. DEIBERT, G. PAPP, F. VILLANTE
"The IBM Engineering Verification Engine"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 218 - 224.

[BOS 87] S. BOSE
"Functional Simulation at the Registre Transfer Level"
Research Report No. CMUCAD-87-50, Carnegie-Mellon University, December 1987.

[BRE 72] M.A. BREUER
"Recent developments in disign automation"
Computer, May/June 1972.

[BRE 81] M.A. BREUER, A.C. PARKER
"Digital System Simulation : Current Status and Future Trends"
18th Design Automation Conference, 1981, pp. 269 - 275.

[BRY 84] R.E. BRYANT
"A Switch-Level Model and Simulator for MOS Digital Systems"
IEEE Transactions on Computers, Vol. C-33, No. 2, February 1984, pp. 160 - 177.

[BRY 87] R.E. BRYANT
"A Survey of Switch-Level Algorithms"
IEEE Design & Test, August 1987, pp. 26 - 40.

[BUE 86] F. BUELOW, E. PORTER
"The Need for Fault Simulation"
VLSI Systems Design, October 1986, pp. 84 - 86.

[BUT 85] M.R. BUTTS
"A General-Purpose Accelerator"
VLSI Systems Design, October 1985, pp. 85 - 86.

[CAR 86] H.W. CARTER
"Computer-Aided Design of Integrated Circuits"
Computer, April 1986, pp. 19 - 36.

[CAR 87a] T. CARLSTEDT-DUKE
"Development of a General Purpose Hardware Accelerator"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 57 - 64.

[CAR 87b] A. CARPENTER, J. GOSLING, H. KAHN
"The Flexibility Handling of Complexity in a Hierarchical Simulation Engine"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 100 - 108.

[CHA 75] H.Y. CHANG, S.G. CHAPPELL
"Deductive Techniques for Simulating Logic Circuits"
Computer, March 1975, pp. 52 - 59.

[CHO 88] K. CHOI, S.Y. HWANG, T. BLANK
"Incremental-in-Time Algorithm for Digital Simulation"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 501 - 505.

[CHU 74] Y.H. CHU
"Introducing CDL"
Computer, December 1974, pp. 31 - 33.

[COL 87] N. COLEMAN, T. AMBLER
"A Multiprocessor for General VLSI Design Acceleration"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 187 - 198.

[DAH 66] O.J. DAHL, K.NYGAARD
"SIMULA - an ALGOL-Based Simulation Language"
Communications of the ACM, Vol. 9, No. 9, September 1966, pp. 671 - 678.

[DAR 87] F. DAREMA, G.F. PFISTER
"Multipurpose Parallelism for VLSI CAD on the RP3"
IEEE Design & Test of Computers, October 1987, pp. 19 - 27.

[DAS 82] **H.W. DASEKING, R.I. GARDNER, P.B. WEIL**
"VISTA : A VLSI CAD System"
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. CAD-1, No. 1, January 1982, pp. 36 - 52.

[DEN 85] **P. DENYER et al.**
"Toward a Benchmark for Logic Simulators"
VLSI Design, August 1985, pp. 18 - 26.

[DEU 86] **J.T. DEUSTCH, T.D. LOVETT, M.L. SQUIRES**
"Parallel Computing for VLSI Circuit Simulation"
VLSI Systems Design, July 1986, pp. 46 - 52.

[DEW 84] **L.A. DEWEY**
"The VHSIC Hardware Description Language"
VLSI Design, Novembre 1984, pp. 33 - 39.

[DOS 84] **M.H. DOSHI, R.B. SULLIVAN, D.M. SCHULER**
"THEMIS Logic Simulator - a Mix Mode, Multi-Level, Hierarchical, Interactive Digital Circuit Simulator"
ACM IEEE 21st Design Automation Conference, June 1984, pp. 24 - 31.

[DUB 88] **P.A. DUBA, R.K. ROY, J.A. ABRAHAM, W.A. ROGERS**
"Fault Simulation in a Distributed Environment"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 686 - 691.

[DUD 83] **S. DUDANI, E. STABLER**
"Types of Hardware Description"
Proceedings of the IFIP WG 10.2 Sixth International Symposium on Computer Hardware Description Languages and their Applications, May 1983, PP. 127 - 136.

[DUM 83] **D. DUMLUGÖL, H.J. DE MAN, P. STEVENS, G.G. SHROOTEN**
"Local Relaxation Algorithms for Event-Driven Simulation of MOS Networks Including Assignable Delay Modeling"
IEEE Transactions on Computer-Aided Design, Vol. CAD-2, No. 3, July 1983, pp. 193 - 201.

[DYS 87] C. DYSON, A. GRAY
"Mixed-Mode Simulation on Transputers"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 145 - 153.

[EMA 86] S. EMAMI, S. BRUNNER
"Logic Simulation on PCs"
VLSI Systems Design, January 1986, pp. 36 - 37.

[EVA 87] S. EVANCZUK
"Mixed-Level Simulation Acceleration"
VLSI Systems Design, February 1987, pp. 62 - 70.

[FLA 75] P.L. FLAKE, G. MUSGRAVE
"The HILO Logic Simulation Language"
Proceedings of the 1975 International Symposium on Computer Hardware Description Languages and their Applications, September 1975, pp. 134 - 143.

[FRA 75] W.R. FRANTA, W.K. GILOI
"APL*DS : A Hardware Description Language For Design And Simulation"
Proceedings of the 1975 International Symposium on Computer Hardware Description Languages and their Applications, September 1975, pp. 45 - 52.

[GAI 87a] S. GAI, F. SOMENZI, M. SPALLA
"Fast and Coherent Simulation with Zero Delay Elements"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 1, January 1987, pp. 85 - 92.

[GAI 87b] S. GAI, F. SOMENZI, E. ULRICH
"Advances in Concurrent Multilevel Simulation"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 6, November 1987, pp. 1006 - 1012.

[GAI 88] S. GAI, P.L. MONTESSORO, F. SOMENZI
"MOZART : A Concurrent Multilevel Simulator"
IEEE Transactions on Computer-Aided Design, Vol. 7, No. 9, September 1988, pp. 1005 - 1016.

[GEN 85] GENRAD
"HILO-3 User Manual"
GenRad, 1985.

[GER 85] S.M. GERMAN, K.J. LIEBERHERR
"Zeus : A Language for Expressing Algorithms in Hardware"
Computer, February 1985, pp. 55 - 65.

[GHO 88] S. GHOSH
"Using ADA as an HDL"
IEEE Design & Test of Computers, February 1988, pp. 30 - 42.

[GIA 79] N. GIAMBIASI, A. MIARA, D. MURIACH
"SILOG : A Pratical Tool For Large Digital Network Simulation"
16th Design Automation Conference, June 1979, pp. 263 - 271.

[GLA 84] M.E. GLAZIER, A.P. AMBLER
"ULTIMATE : A Hardware Logic Simulation Engine"
ACM IEEE 21st Design Automation Conference, June 1984, pp. 336 - 342.

[HAH 87] W. HAHN
"The MUNICH Simulation Computer : Design Principles and Performance Prediction"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 109 - 120.

[HAN 88] C. HANSEN
"Hardware Logic Simulation by Compilation"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 712 - 715.

[HAY 87] J.P. HAYES
"An introduction to Switch-Level Modeling"
IEEE Design & Test, August 1987, pp. 18 - 25.

[HEM 75] C.W. HEMMING, J.M. HEMPHILL
"Digital Logic Simulation Models and Evolving Technology"
12th Design Automation Conference, June 1975, pp. 85 - 91.

[HIL 79] D. HILL, W. VANCLEEMPUT
"SABLE : A Tool For Generating Structured, Multi-Level Simulations"
16th Design Automation Conference, June 1979, pp. 272 - 279.

[HIL 80] D.D. HILL, W. VANCLEEMPUT
"SABLE : Multi-Level Simulation For Hierarchical Design"
Proceedings of the 1980 IEEE International Symposium on Circuits and Systems, 1980, pp. 431 - 434.

[HWA 88] S.Y. HWANG, T. BLANK, K. CHOI
"Fast Functional Simulation : An Incremental Approach"
IEEE Transactions on Computer-Aided Design, Vol. 7, No. 7, July 1988, pp. 765 - 774.

[HWA 89] S.Y. HWANG
"Incremental algorithms for digital simulation"
Integration, the VLSI journal, 7 1989, pp. 21 - 34.

[INF 83] B. INFANTE, M. BALES, E. LOCK
"MADL : A Language for Describing Mixed Dehavior and Structure"
Proceedings of the IFIP WG 10.2 Sixth International Symposium on Computer Hardware Description Languages and their Applications, May 1983, PP. 115 - 126.

[INM 88] INMOS
"Occam 2 Reference Manual"
INMOS 1988.

[INM 89] INMOS
"Transputer Technical Notes"
INMOS 1989.

[ISH 87] N. ISHIURA, H. YASUURA, S. YAJIMA
"High-Speed Logic Simulation on Vector Processors"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 3, May 1987, pp. 305 - 321.

[JAI 85] S.K. JAIN, V.D. AGRAWAL
"Statistical Fault Analysis"
IEEE Design & Test, February 1985, pp. 38 - 44.

[JOH 80] W. JOHNSON, J. CROWLEY, M. STEGER, E. WOOSLEY
"Mixed-Level Simulation from a Hierarchical Language"
Journal of Digital Systems, Vol. 4, No. 3, Fall 1980, pp. 305 - 335.

[KAU 88] S. KAUL, N. JACOBSON, I. LIVNAT
"Multiprocessor Accelerator for Mixed-Level Simulation"
VLSI Systems Design, May 1988, pp. 64 - 71.

[KAZ 88] Y. KAZAMA et al.
"Algorithm for Vectorizing Logic Simulation and Evaluation of VELVET Performance"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 231 - 236.

[KER 88] A. KERN, A. BLAZEVICIUS
"A Concurrent Hardware And Software Design Environment"
VLSI Systems Design, August 1988, pp. 34 - 40.

[KIM 89] Y.H. KIM, S.H. HWANG, A.R. NEWTON
"Electrical-Logic Simulation and Its Applications"
IEEE Transactions on Computer-Aided Design, Vol. 8, No. 1, January 1989, pp. 8 - 22.

[LEV 80] Y.H. LEVENDEL, P.R. MENON
"Comparison of Fault Simulation Methods - Treatment of Unknown Signal Values"
Journal of Digital Systems, Vol. 4, No. 4, Winter 1980, pp. 443 - 459.

[LEW 87] D. LEWIS
"Design Considerations in an Advanced Circuit Simulation Accelerateur"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 174 - 186.

[LEW 88a] D.M. LEWIS
"A Programmable Hardware Accelerator for Compiled Electrical Simulation"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 172 - 177.

[LEW 88b] D.M. LEWIS
"Hardware Accelerators for Timing Simulation of VLSI Digital Circuits"
IEEE Transactions on Computer-Aided Design, Vol. 7, No. 11, November 1988, pp. 1134 - 1149.

[LIG 87] M.R. LIGHTNER
"Modeling and Simulation of VLSI Digital Systems"
Proceedings of the IEEE, Vol. 75, No. 6, June 1987, pp. 786 - 796.

[MAR 88] E. MARSCHNER
"A VHDL Design Environment"
VLSI Systems Design, September 1988, pp. 40 - 49.

[MEH 87] P. MEHRING, E. APOSPORIDIS
"Multi-level Simulator for VLSI - an Overview"
Proceedings of the 4th Annual ESPRIT Conference, Brussels, September 1987, Part 1, pp. 736 - 749.

[MEY 85] E.L. MEYER
"Application-Specific Hardware Accelerators : An Overview"
VLSI Systems Design, October 1985, pp. 48 - 59.

[MOT 85] G. MOTT, R.B. HALL
"The Utility of Hardware Accelerators In the Design Environment"
VLSI Systems Design, October 1985, pp. 62 - 70.

[NAK 87] Y. NAKAMURA
"An Integrated Logic Design Environment Based on Behavioral Description"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 3, May 1987, pp. 322 - 336.

[NAV 79] Z. NAVABI, F.J. HILL
"Efficient Simulation of AHPL"
16th Design Automation Conference, June 1979, pp. 255 - 262.

[NEW 79] A.R. NEWTON
"Techniques for the Simulation of Large-Scale Integrated Circuits"
IEEE Transactions on Circuits and Systems, Vol. CAS-26, No. 9, September 1979, pp. 741 - 749.

[NEW 81] A.R. NEWTON
"Computer-Aided Design of VLSI Circuits"
Proceedings of the IEEE, Vol. 69, No. 10, October 1981, pp. 1189 - 1199.

[NEW 86] A.R. NEWTON, A.L. SANGIOVANNI-VINCENTELLI
"Computer-Aided Design for VLSI Circuits"
Computer, April 1986, pp. 38 - 60.

[NEW 87] J. NEWKIRK
"An Introduction to CAE Accelerators"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 5 - 21.

[NIE 86] R.D. NIELSON
"Algorithmically Accelerated CAD"
VLSI Systems Design, February 1986, pp. 65 - 66.

[NUR 89] G.M. NURIE, P.J. MENCHINI
"VHDL Model Portability"
High Performance Systems, July 1989, pp. 76 - 85.

[OBJ 88] P. OBJOIS
"Réseau de cellules intégré : mécanisme de communication inter-cellulaire et application à la simulation logique"
Thèse INPG de microélectronique, Grenoble, Septembre 1988.

[ODE 87] P. ODENT, D. DUMLOGOL, H. DE MAN
"Hardware Acceleration of Circuit Simulation on a Multi-Microprocessor System"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 154 - 163.

[PFI 86] G.F. PFISTER
"The IBM Yorktown Simulation Engine"
Proceedings of the IEEE, Vol. 74, No. 6, June 1986, pp. 850 - 860.

[PIL 85] R. PILOTY, D. BORRIONE
"The Conlan Project : Concepts, Implementations, and Applications"
Computer, February 1985, pp. 81 - 92.

[RIG 89] R. RIGHTER, J.C. WALRAND
"Distributed Simulation of Discrete Event Systems"
Proceedings of the IEEE, Vol. 77, No. 1, January 1989, pp. 99 - 113.

[ROG 87] W.A. ROGERS, J.F. GUZOLEK, J.A. ABRAHAM
"Concurrent Hierarchical Fault Simulation : A Performance Model and Two Optimizations"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 5, September 1987.

[RUE 83] A.E. RUEHLI, G.S. DITLOW
"Circuit Analysis, Logic Simulation, and Design Verification for VLSI"
Proceedings of the IEEE, Vol. 71, No. 1, January 1983, pp. 34 - 48.

[SAI 88] M. SAITOH et al.
"Logic Simulation System Using Simulation Processor (SP)"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 225 - 230.

[SES 62] S. SESHU, D.N. FREEMAN
"The Diagnosis of Asynchronous Sequential Switching Systems"
IRE Transactions on Electronic Computers, August 1962, pp. 459 - 465.

[SHA 85] M. SHAHDAD et al.
"VHSIC Hardware Description Language"
Computer, February 1985, pp. 94 - 103.

[SHE 79] W. SHERWOOD
"A Hybrid Scheduling Technique For Hierarchical Logic Simulators"
16th Design Automation Conference, June 1979, pp. 249 - 254.

[SHE 87] C. SHELDON
"The Analysis of Simulation Acceleration Concepts and the Associated Trade-Offs in the CAE Environment"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 22 - 34.

[SHI 85] S.G. SHIVA, P.F. KLON
"The VHSIC Hardware Description Language"
VLSI Systems Design, June 1985, pp. 86 - 106.

[SIE 85] S. SIEGEL, M.E. KASZYNSKI
"The Design of a Logic Simulation Accelerator"
VLSI Systems Design, October 1985, pp. 76 - 80.

[SMI 87] S. SMITH, B. WOOD
"Architecture of a General Accelerator for CAD : the PROTEUS-1"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 261 - 270.

[SON 85] K. SON
"Fault Simulation with the Parallel Value List Algorithm"
VLSI Systems Design, December 1985, pp. 36 - 43.

[SOU 88] L. SOULE, T. BLANK
"Parallel Logic Simulation on General Purpose Machines"
25th ACM/IEEE Design Automation Conference, June 1988, pp. 166 - 171.

[SPI 86] I. SPILLINGER, G.M. SILBERMAN
"Improving the Performance of a Switch-Level Simulator Targeted for a Logic Simulation Machine"
IEEE Transactions on Computer-Aided Design, Vol. CAD-5, NO. 3, July 1986, pp. 396 - 404.

[SUZ 85] N. SUZUKI
"Concurrent Prolog as an Efficient VLSI Design Language"
Computer, February 1985, pp. 33 - 40.

[SVE 88] C. SVENSSON, R. TJÄRNSTRÖM
"Switch-Level Simulation and the Pass Transistor Exor Gate"
IEEE Transactions on Computer-Aided Design, Vol. 7, No. 9, September 1988, pp. 994 - 997.

[SZY 72] S.A. SZYGENDA
"TEGAS2 - Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic"
9th Design Automation Conference, June 1972, pp. 116 - 127.

[SZY 75] S.A. SZYGENDA, E.W. THOMPSON
"Digital Logic Simulation In a Time-Based, Table-Driven Environment - Part 1. Design Verification"
Computer, March 1975, pp. 24 - 36.

[SZY 76] S.A. SZYGENDA, E.W. THOMPSON
"Modeling and Digital Simulation for Design Verification and Diagnosis"
IEEE Transactions on Compouters, Vol. C-25, No. 12, December 1976, pp. 1242 - 1253.

[TAK 87] S. TAKASAKI et al.
"Block-Level Hardware Logic Simulation Machine"
IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 1, January 1987, pp. 46 - 54.

[THA 87] K. THAM
"Switch-Level Simulation on a Hypercube MIMD Computer"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 121 - 130.

[THO 75] E.W. THOMPSON, S.A. SZYGENDA
"Digital Logic Simulation In a Time-Based, Table-Driven Environment - Part 2. Parallel Fault Simulation"
Computer, March 1975, pp. 38 - 49.

[THO 80] E. THOMPSON, A. LEKKOS, D. ROSS, R. VON BLUCHER
"TEGAS Design Language (TDL) - A System for Modular Description and Top-Down/Bottom-Up Verification of LSI and VLSI"
Proceedings of the 1980 IEEE International Symposium on Circuits and Systems, Volume 1, 1980, pp. 300 - 303.

[ULR 85] E. ULRICH
"Concurrent Simulation at the Switch, Gate and Register Levels"
1985 International Test Conference, November 1985, pp. 703 - 709.

[ULR 86] E. ULRICH, I. SUETSUGU
"Techniques for Logic and Fault Simulation"
VLSI Systems Design, October 1986, pp. 68 - 81.

[VAN 79] W.M. VANCLEEMPUT
"Computer Hardware Description Language and their Applications"
16th Design Automation Conference, June 1979, pp. 554 - 560.

[VAU 75] J.G. VAUCHER, P. DUVAL
"A Comparison of Simulation Event List Algorithms"
Communications of the ACM, Vol. 18, No. 4, April 1975, PP. 223 - 230.

[VLA 81] A. VLADIMIRESCU et al.
"SPICE Version 2G User's guide"
University of California, Berkeley, August 1981.

[VLS 86a] VLSI Systems Design Staff
"1986 Survey of Logic Simulators"
VLSI Systems Design, February 1986, pp. 32 - 40.

[VLS 86b] VLSI Systems Design Staff
"Survey of CAE Systems"
VLSI Systems Design, June 1986, pp. 85 - 105.

[VLS 87a] VLSI Systems Design Staff
"1987 Survey of Logic Simulators"
VLSI Systems Design, February 1987, pp. 71 - 86.

[VLS 87b] VLSI Systems Design Staff
"Survey of CAE Systems"
VLSI Systems Design, June 1987, pp. 51 - 69.

[VLS 88a] VLSI Systems Design Staff
"1988 Survey of Logic Simulators"
VLSI Systems Design, February 1988, pp. 50 - 61.

[VLS 88b] VLSI Systems Design Staff
"Hardware Accelerators"
VLSI Systems Design, August 1988, pp. 41 - 47.

[VLS 88c] VLSI Systems Design Staff
"Hardware Modelers"
VLSI Systems Design, September 1988, pp. 32 - 36.

[WAI 85] J.A. WAICUKAUSKI et al.
"Fault Simulation for Structured VLSI"
VLSI Systems Design, December 1985, pp. 20 - 32.

[WAI 87] J.A. WAICUKAUSKI et al.
"Transition Fault Simulation"
IEEE Design & Test, April 1987, pp. 32 - 38.

[WAT 87] P.J. WATERMAN
"On Parallel Circuit Simulation"
VLSI Systems Design, July 1987, pp. 56 - 61.

[WAX 86] R. WAXMAN
"Hardware Design Languages for Computer Design and Test"
Computer, April 1986, pp. 90 - pp. 97.

[WIL 79] P. WILCOX
"Digital Logic Simulation at the Gate and Functional Level"
16th Design Automation Conference, June 1979, pp. 242 - 248.

[WIT 85] B.I. WITT
"Parallelism, Pipelines, and Partitions : Variations on Communicating Modules"
Computer, February 1985, pp. 105 - 112.

[WHA 86] D.J. WHARTON
"Behavioral Modeling in Logic Simulation"
VLSI Systems Design, August 1986, pp. 46 - 54.

[WON 87] K. WONG, M. FRANKLIN
"Load and Communications Balancing on Multiprocessor Logic Simulation Engines"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 80 - 89.

[YOS 87] H. YOSHIDA, S. KUMAGAI, I. SHIRAKAWA, S. KODAMA
"A Parallel Implementation of Large-Scale Circuit Simulation"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 164 - 173.

[ZAS 87] J. ZASIO, P. HWANG
"A Low-Cost High-Performance Levelized Compiled-Code Simulation Accelerator"
International Workshop on Hardware Accelerators, Oxford, September 1987, pp. 46 - 56.

Annexe A

Langages Hilo-3

A.1. Hilo-3 HDL : langage de description du matériel

A.1.1. Modélisation

◊ Circuit

Le circuit est une entité formelle qui caractérise la modularité du langage. Un circuit est typé et paramétrable, pouvant être utilisé comme un modèle générique pour constituer des circuits plus complexes dans une conception hiérarchique.

L'interface du circuit est décrite par un en-tête dans lequel sont définis l'identification du circuit, le type du circuit, la technologie utilisée, les paramètres formels et les ports de connexion.

Le corps du circuit définit la structure et le comportement du circuit.

La description structurelle consiste à décrire le circuit par une liste d'éléments reliés par des interconnexions. Un élément peut être une porte logique, ou une instance de sous-circuit. La description comportementale est basée sur les expressions arithmétiques et logiques, les primitives fonctionnelles, et les instructions gardées.

La simulation normale et la simulation de pannes partagent les mêmes modèles de circuits.

◊ Valeurs logiques

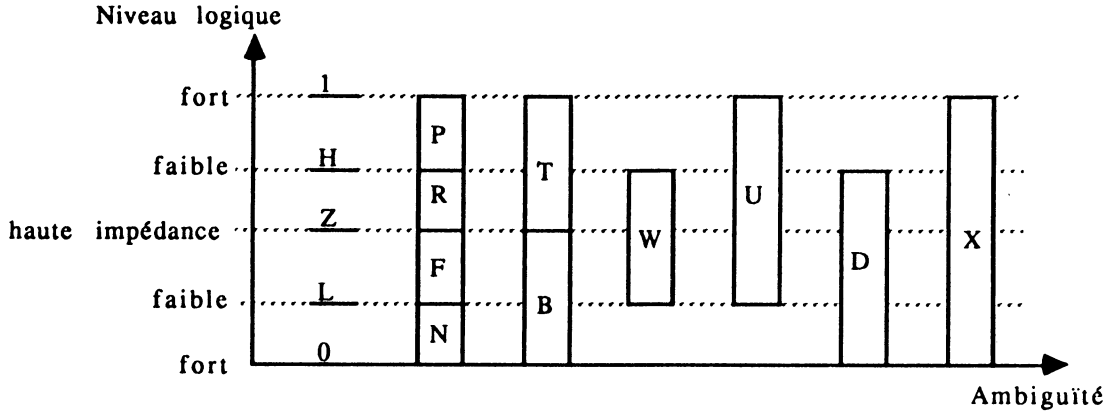
Le langage supporte deux types de valeurs logiques : la logique à 4 valeurs et à 15 valeurs.

La logique à 4 valeurs se compose de 0, 1, X et Z.

La logique à 15 valeurs prend en compte la force des signaux et l'ambiguïté du niveau logique. Elle contient :

- 0 : zéro fort;
- L : zéro faible;
- N : valeur ambiguë entre 0 et L;
- H : un faible;

- 1 : un fort;
- P : valeur ambiguë entre 1 et H;
- Z : haute impédance;
- B : valeur ambiguë entre 0, L et Z;
- R : valeur ambiguë entre H et Z;
- F : valeur ambiguë entre L et Z;
- T : valeur ambiguë entre 1, H et Z;
- W : valeur ambiguë entre H, Z et L;
- D : valeur ambiguë entre 0, L, Z, et H;
- U : valeur ambiguë entre 1, H, Z et L;
- X : valeur indéterminée.



Logique à 15 états

Une valeur complémentaire est définie qui représente un événement fonctionnel : E.

◊ Retards

Le langage autorise des retards variables. Un retard se compose de trois valeurs : minimale, typique et maximale.

La modélisation du retard est raffinée par la distinction des retards de montée et de descente. Le calcul du retard réel d'une connexion tient compte de l'effet de la capacité. Chaque sortie d'éléments peut être affectée des retards, sous une des formes suivantes :

$(retard_{montée}, retard_{descente})$

$(retard_{montée}, retard_{descente}, marginal_{montée}, marginal_{descente}, capacité)$

Le calcul du retard supplémentaire dû à la capacité utilise les coefficients marginaux et la somme des capacités de tous les éléments connectés à cette sortie en tenant compte des points capacitifs explicitement déclarés. Le retard total est :

$$retard_{total} := retard + marginal * (somme\ de\ toutes\ les\ capacités)$$

◊ Forces

Il s'agit des forces de sorties d'éléments. A chaque sortie sont associées deux forces, qui correspondent aux cas des valeurs 0 ou 1 générées à la sortie. Les forces sont déterminées implicitement en fonction de la technologie du circuit, ou définies explicitement.

Il existe 5 forces :

- WEAK : faible;
- STRONG : fort;
- HIGHIMP : haute impédance;
- UNCHANGED : inchangé;
- RESISTANCE : résistance.

◊ Pannes

Les hypothèses de pannes sont supposées situées à l'extérieur des éléments primitifs :

- le collage à 0 ou à 1 : sur les équipotentielles, les registres, les entrées et les sorties de portes;
- le collage à Z : sur les entrées de portes;
- le court-circuit : entre deux équipotentielles;
- l'inhibition : uniquement pour les événements fonctionnels.

A.1.2. Description structurelle

◊ Généralités

Le corps du circuit se compose d'un ensemble d'éléments et d'un ensemble d'équipotentielles. La structure topologique du circuit est décrite

conjointement par les listes de connexions des éléments et par les listes de terminaux des équipotentielles.

Le langage permet d'utiliser des sous-circuits comme des modèles génériques. L'instantiation s'applique non seulement aux modèles de sous-circuits, mais aussi aux éléments primitifs et aux équipotentielles.

Cette structure de données assure une description modulaire et hiérarchique du circuit. En plus, les instances peuvent être regroupées en vecteurs. Un vecteur est une combinaison unidimensionnelle des éléments ou des équipotentielles conforme à la structure physique du circuit réel : le bus, le registre, etc. Les objets individuels du vecteur peuvent être référencés par l'indexation.

◊ Eléments

Le langage définit un ensemble de types d'éléments pour décrire les éléments primitifs. Ce sont les portes classiques et des composants spécifiques utilisés dans certaines technologies. Les principaux types sont :

- les fonctions logiques : AND, NAND, OR et NOR;
- la technologie I^2L : BUF et NOT;
- la technologie MOS : TRANIF0, TRANIF1, MOVEIF0 et MOVEIF1;
- la logique à trois états : BUFIF0, BUFIF1, NOTIF0, NOTIF1;
- les horloges : CLOCK0 et CLOCK1;
- le point capacitif : CAPACITOR.

La fonction d'un élément primitif est définie par son type. Chaque élément peut avoir une liste de retards, une liste de forces, et une liste de connexions.

◊ Equipotentielles

Une équipotentielle est définie par son type, ses retards, sa capacité, et une liste de terminaux qui permet de décrire les éléments connectés à cette équipotentielle. Les équipotentielles peuvent avoir un des types suivants :

- les entrées du circuit : INPUT, INPUT0 et INPUT1;
- les sources externes : SUPPLY0 et SUPPLY1;
- les connexions à trois états : TRI, TRI0, TRI1 et TRIREG;

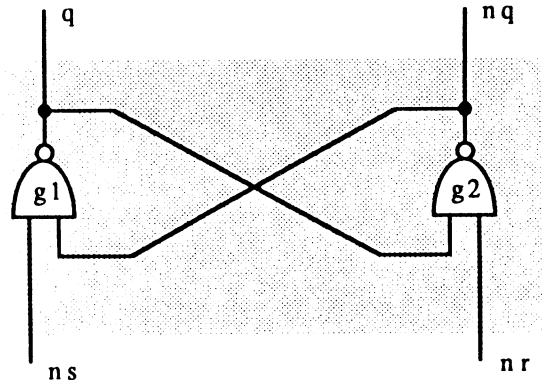
- les fonctions câblées : WAND et WOR;
- le type ambigu : WIRE.

◊ Exemples

Une bascule RS peut être décrite d'une des façons suivantes (redondante et non redondante) :

```
CCT (r1,r2) RSFF (q,nq,ns,nr)
   NAND (r1,r2)  g1 (q,ns,nq)
                 g2 (nq,nr,q);
   INPUT         ns (g1.2) nr (g2.2);
   WIRE          q (g1.1, g2.3)
                 nq (g2.1, g1.3).
```

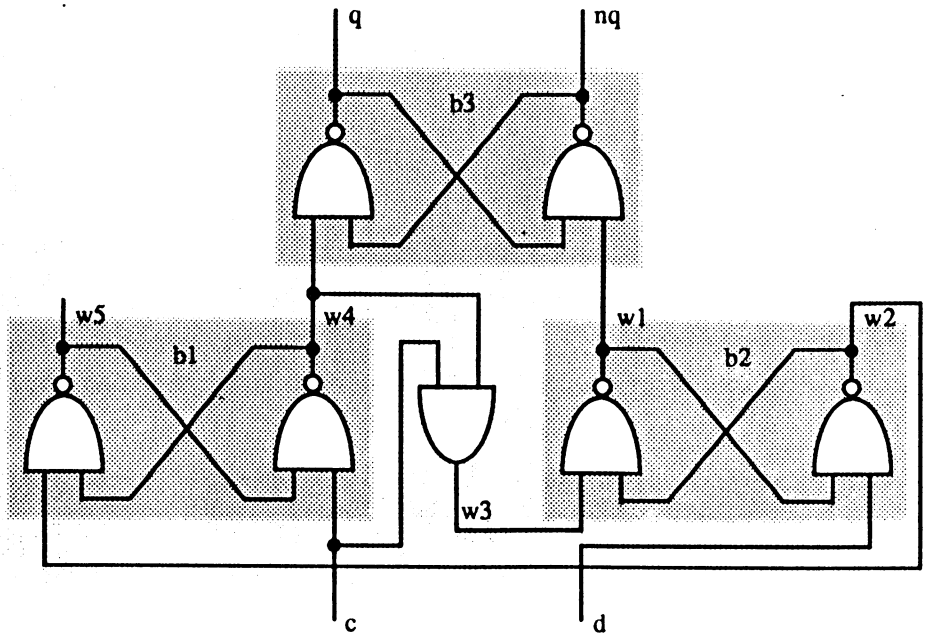
```
CCT (r1,r2) RSFF (q,nq,ns,nr)
   NAND (r1,r2)  g1 (q,ns,nq)
                 g2 (nq,nr,q);
   INPUT ns nr;
   WIRE q nq.
```



Bascule RS

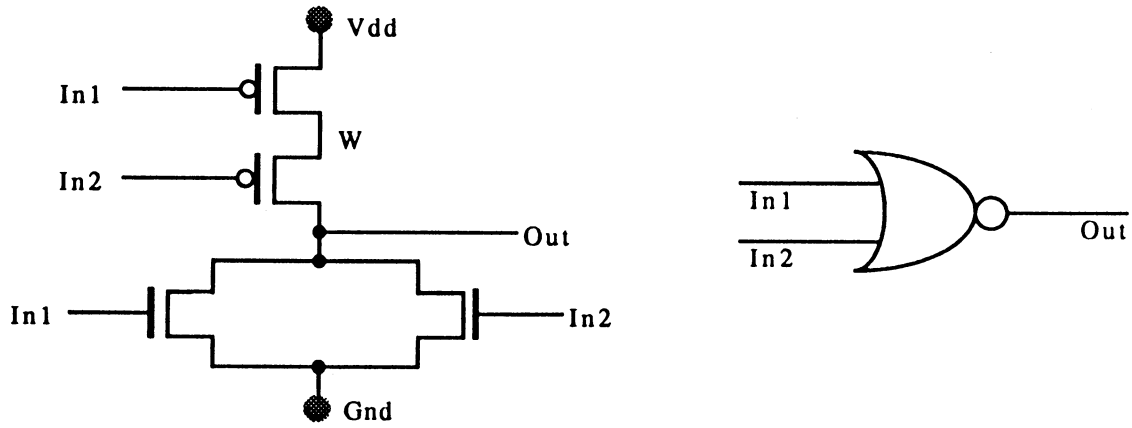
Une bascule D peut être construite à base des bascules RS :

```
CCT (r1,r2) DFF (q,nq,c,d)
        RSFF (r1,r2)   b1 (w5,w4,w2,c)
                       b2 (w1,w2,w3,d)
                       b3 (q,nq,w4,w1);
        AND g (w3,w4,c);
        WIRE w1 w2 w3 w4 w5;
        INPUT c d;
        WIRE q nq.
```



Bascule D

Le langage permet également de décrire le circuit MOS au niveau interrupteur :



Porte NON-OU en CMOS

```
CCT CMOS NorGate (Out, In1, In2)
        TRANIF0     TP1  (W,Vdd,In1)
                    TP2  (Out,W,In2);
        TRANIF1     TN1  (Out,Gnd,In1)
                    TN2  (Out,Gnd,In2);
        SUPPLY0 Gnd;
        SUPPLY1 Vdd;
        TRIREG Out;
        INPUT In1 In2.
```

A.1.3. Description fonctionnelle et comportementale

◊ Généralités

La description fonctionnelle et comportementale du circuit est supportée par le langage FML (*Functional Modeling Language*), un sous-ensemble du langage HDL.

L'essentiel du langage FML consiste en trois mécanismes de base :

- les expressions arithmétiques et logiques;
- les primitives fonctionnelles;
- les instructions gardées.

Les expressions arithmétiques et logiques définissent les fonctions booléennes. Ces expressions permettent d'utiliser des opérateurs prédéfinis pour évaluer des valeurs logiques des équipotentielles. Elles représentent des circuits combinatoires.

Les primitives fonctionnelles permettent de décrire directement des éléments composés, comme le registre, la ROM et la RAM, d'une dimension arbitraire. Elles peuvent être affectées des retards de la même manière que les éléments logiques.

Les instructions gardées permettent de décrire un ensemble d'opérations déclenchées sous certaines conditions de garde. Les opérations de base sont les transferts de registres et des commandes fonctionnelles. La garde est une expression d'événements.

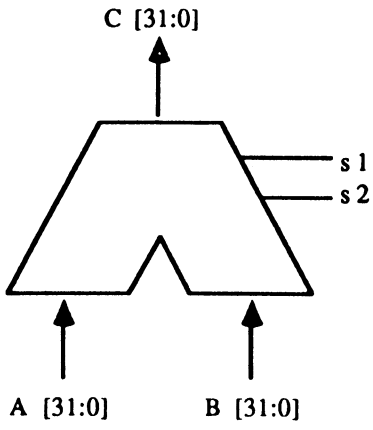
◊ Expressions arithmétiques et logiques

Une expression est associée à une équipotentielle ou un vecteur d'équipotentielles. Les retards peuvent être utilisés dans cette expression.

Les opérations autorisées dans les expressions sont :

- les opérations logiques : AND, NAND, OR, NOR, XOR et NXOR, etc;
- les opérations arithmétiques : + et -;
- l'opération sélective : VALCASE.

L'exemple suivant est une unité arithmétique et logique 32 bits décrite par une expression :




```
WIRE (20:22:26,21:24:27)
    C[31:0] = VALCASE {s1,s2},
                00 = A[31:0] + B[31:0],
                01 = A[31:0] - B[31:0],
                10 = A[31:0] AND B[31:0],
                11 = A[31:0] OR B[31:0],
                DEFAULT = X
            ENDCASE;
```

◊ Primitives fonctionnelles

Les primitives fonctionnelles modélisent les éléments composés fréquemment utilisés ou ayant une structure très symétrique :

- les mémoires : ROM et RAM;
- les registres : REGISTER.

Un modèle de registre représente soit un registre classique ou un registre virtuel. La notion de registre virtuel est conforme au concept de boîte noire des langages fonctionnels.

L'activation d'un tel registre est contrôlée par une expression associée. Cette expression, qui décrit un mécanisme de déclenchement du registre, est une des opérations suivantes :

- le déclenchement par niveau logique 0 : LOADIF0;
- le déclenchement par niveau logique 1 : LOADIF1;
- le déclenchement par cas : LOADCASE.

L'exemple suivant est une déclaration de deux registres :

```
REGISTER (11:26, 63:65)
```

```
Reg1[31:0] = 0 LOADIF1 (Winit OR Wzero)
Reg2 =  LOADCASE flag [0:1],
            00 = NOR (Wa,Wb),
            11:01 = 1
        ENDCASE;
```

◊ Instructions gardées

Une instruction gardée représente un ensemble d'opérations (liste d'actions) dont le déclenchement est contrôlé par une garde (expression d'événements), sous une des formes suivantes :

```
WHEN expression d'événements DO liste d'actions;
NEXT  expression d'événements DO liste d'actions;
```

Une expression d'événements peut comprendre deux types d'événements de base :

- un événement fonctionnel;
- un changement de valeur identifiable d'une équipotentielle.

Les opérateurs suivants peuvent être utilisés dans les expressions d'événements :

- la conjonction : OR;
- la séquence : THEN;
- la répétition : REPEAT;
- le retardement : WAIT;
- la détection des événements hors séquence : RESET.

L'exemple suivant est une expression d'événements :

```
WHEN 2 REPEAT (A THEN B) THEN C WAIT 5 THEN C
    RESET D OR E
        DO DISPLAY "OK";
```

Pour cette expression, la séquence valide d'événements doit être la suivante, à condition que les événements D et E ne se produisent pas pendant cette séquence :

A, B, A, B, C, attendre 5 unités de temps, C

La liste d'actions peut être soit une séquence d'opérations avec des branchements conditionnels, soit un simple transfert de données ou une commande de simulation. Les opérations ne sont effectuées que si l'expression d'événements est valide.

La liste d'actions peut avoir les opérations suivantes :

- les transferts de données : les affectations;
- la création des événements fonctionnels : EVENT;
- la visualisation des messages ou des valeurs : DISPLAY et PRINT;
- le contrôle de simulation : HALT et FINISH.

Les structures de branchements peuvent être également utilisées dans la liste d'actions :

- les branchements conditionnels : IF et IFNOT;
- les branchements sélectifs : CASE.

Les expressions conditionnelles sont utilisées dans les branchements. Une condition est soit une expression relationnelle, soit une fonction booléenne. Deux fonctions spécifiques pour la vérification temporelle sont définies :

- la largeur d'une impulsion : WIDTH;
- la stabilité d'un signal : STEADY.

L'exemple suivant illustre une liste d'actions avec des branchements conditionnels :

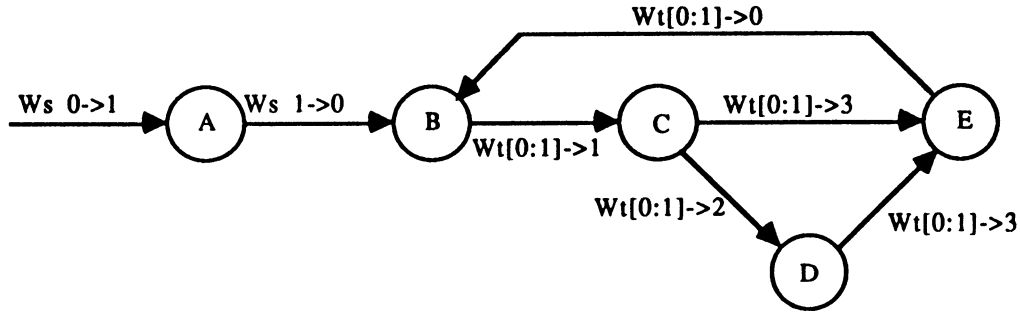
```
WHEN select (0:X TO 1) DO
        IF STEADY (ReadWrite, 10) & ReadWrite NE X
                THEN
                        IF ReadWrite EQ 1
                                THEN Reg[31:0] = Mem[address]
                                ELSE Mem[address] = Reg[31:0]
                        ENDIF
                ELSE
                        EVENT error
```

```
                    DISPLAY ("Not steady : ",ReadWrite)
        ENDIF;
```

L'enchaînement des instructions gardées se fait à l'aide de deux mécanismes :

- l'instruction NEXT;
- l'événement fonctionnel EVENT.

L'automate suivant peut être décrit par l'utilisation des instructions gardées et des événements fonctionnels :




```
WHEN Ws (0 TO 1) DO DISPLAY "State A";
NEXT  Ws (1 TO 0) DO EVENT B  DISPLAY "State B";
WHEN B THEN Wt[0:1] (?? TO 01) DO EVENT C  DISPLAY "State C";
WHEN C THEN Wt[0:1] (?? TO 10:11)
        DO CASE Wt[0:1],
                10 - EVENT D  DISPLAY "State D",
                11 - EVENT E  DISPLAY "State E"
            ENDCASE;
WHEN D THEN Wt[0:1] (?? TO 11) DO EVENT E  DISPLAY "State E";
WHEN E THEN Wt[0:1] (?? TO 00) DO EVENT B  DISPLAY "State B";
```

A.2. Hilo-3 WDL : langage de description des interactions

A.2.1. Généralités

Ce langage permet de définir l'interface entre le circuit et son environnement, et les interactions qui auront lieu entre eux pendant la simulation. Les interactions sont dynamiques et temporelles. Le langage est donc caractérisé par la nature concurrente. Les activités décrites dans le langage sont dirigées par le flot du temps.

Le langage permet de définir :

- des stimuli à appliquer au circuit;
- des réponses souhaitées du circuit;
- des commandes de test et de contrôle de la simulation.

Lors de la simulation, des signaux externes sont injectés, en fonction du temps, dans le circuit. Des signaux en provenance du circuit sont vérifiés et comparés avec des réponses souhaitées. Leurs incohérences doivent être identifiées par le simulateur. Le langage permet d'autre part d'effectuer des contrôles sur la simulation.

En effet, une description en WDL permet non seulement la vérification du fonctionnement du circuit, mais aussi la validation des séquences de tests en cas de la simulation de pannes. La séquence de tests est décrite par des stimuli. Les opérations de test sont représentées par des commandes spécifiques.

A.2.2. Déclarations

Les points de stimulus et de réponse du circuit sont déclarées avec un type et des valeurs initiales :

- les points de stimulus : STIMULUS;
- les points de réponse : RESPONSE;
- les points bidirectionnels : BIDIRECTIONAL.

Des variables ou des constantes peuvent être également déclarés :

- les variables de type entier : INTEGER;

- les variables de type temps : TIME;
- les tableaux des valeurs logiques : TABLE.

Bien que les variables pures représentent des données algorithmiques, les opérations sur ces variables sont également temporelles. Les variables changent leur valeur chronologiquement, ce qui est différent de certains langages qui distinguent des variables statiques et dynamiques au niveau du temps.

A.2.3. Affectations, commandes et blocs

Les activités de base sont les affectations et les commandes. Elles sont toutes des opérations temporisées.

L'affectation d'un point de stimulus représente un stimulus injecté dans le circuit à un instant donné :

```
10 S = 0;
```

L'affectation d'un point de réponse représente une réponse du circuit souhaitée à un instant donné :

```
20 R1 = 0  R2 = 1;
```

Que ce soit pour un stimulus ou pour une réponse, les affectations constituent un historique de valeurs logiques des signaux. Il est naturel que dans la description les affectations et les commandes sont organisées par ordre chronologique.

Des affectations sont appliquées également aux autres types de variables, toujours avec le respect du temps.

Il existe des commandes qui permettent la création des chronogrammes spécifiques, c'est-à-dire des séquences de valeurs avec des informations temporelles associées :

- Les chronogrammes scalaires : CHANGE0 et CHANGE1;
- Les chronogrammes vecteurs : CHANGE.

Ces commandes sont aussi utilisables pour les stimuli que pour les

réponses. L'exemple suivant est une commande qui crée un chronogramme scalaire :

```
100 s = CHANGE0 (10,30,+10);
```

Elle est équivalente à la séquence suivante :

```
100 s = 0;
110 s = 1;
130 s = 0;
140 s = 1;
```

Les commandes suivantes réalisent la vérification des réponses du circuit :

- la désignation des réponses à vérifier : CHECK;
- la désignation des réponses à négliger : IGNORE;
- l'échantillonnage des valeurs réelles du circuit : INSERT;
- le déclenchement de la vérification des réponses : STROBE.

Dans le langage, certaines commandes de test et de contrôle sont définies :

- les tests : TESTINITIALISED et TESTSTABLE;
- la visualisation : MONITORON, MONITOROFF et MONITOR;
- le contrôle : FINISH.

Un bloc regroupe un ensemble d'opérations qui partage une base de temps relative. Il peut être imbriqué dans un autre. Les constructeurs de blocs sont :

- l'imbrication de bloc : DO;
- la répétition de bloc : BY REPEAT et BY TO.

L'exemple suivant montre la répétition et l'imbrication de bloc :

```
100 BY 30 REPEAT 6 DO (5 STROBE);
```

Ce qui est équivalent à :

```
105 STROBE;
135 STROBE;
165 STROBE;
195 STROBE;
225 STROBE;
255 STROBE;
```

A.3. Hilo-3 SCL : langage de commandes de simulation

A.3.1. Généralités

Les commandes de simulation constituent, en un sens, un langage distinct, car ce sont des commandes non interprétables et qu'elles nécessitent des pré-traitements spécifiques. Un ensemble de commandes constituent une description complète qui définit une tâche de simulation.

Elle contient essentiellement les informations suivantes :

• le mode de simulation : Il s'agit d'effectuer une simulation normale ou une simulation de pannes sur un circuit désigné. En plus, les stimuli à appliquer dans la simulation doivent être spécifiés. La simulation de pannes peut être sous le mode incrémental, en utilisant un dictionnaire diagnostique.

• les conditions de simulation : Il est nécessaire de décrire les conditions initiales de la simulation qui consiste à définir les valeurs initiales du circuit, en tenant compte de l'instabilité de celui-ci. Différents modes de retard peuvent être sélectionnés pour avoir un comportement temporel conforme aux différentes circonstances. De plus, les caractéristiques du circuit dérivent d'un environnement opérationnel à l'autre. Donc il est autorisé de réviser, en fonction des conditions supposées, les attributs du circuit, sans recompilation.

• la précision : La simulation est une méthodologie déterministe à un certain degré. Le compromis entre la précision et la vitesse de la simulation doit être contrôlable par des commandes. L'utilisateur est capable de rectifier la simulation afin que les résultats ne soient ni trop optimistes ni trop pessimistes. Pour ce qui concerne la simulation de pannes, le taux de détection et celui de détection potentielle (la détection avec la valeur X) sont également contrôlables.

• la gestion des données : Il faut permettre à l'utilisateur de définir les points de visualisation pour l'édition et l'analyse des résultats. En plus, les données de la simulation peuvent être capturées et stockées dans des fichiers pour les post-traitements. Le langage offre différents formats de résultats pour rendre les données plus significatives et claires. D'un autre côté, des commandes sont définies pour gérer les informations nécessaires à la simulation de pannes, par exemple, des commandes pour la gestion du dictionnaire diagnostique qui mémorise des données intermédiaires, supportant la simulation incrémentale.

• le contrôle de la simulation : D'une part, des commandes sont

définies pour contrôler l'ensemble de la simulation de façon globale. D'autre part, tous les objets du circuit, même s'ils se situent au plus profond de la hiérarchie de circuits sont également accessibles et contrôlables par l'utilisateur.

A.3.2. Simulation normale

◊ Visualisation

Les commandes de visualisation sont les suivantes :

- vers l'écran : DISPLAYCHANGE, DISPLAYSTABLE et DISPLAYSTROBE;
- vers un fichier : PRINTCHANGE, PRINTSTABLE et PRINTSTROBE;
- la création du fichier de capture : CAPTURE et CAPTURESTROBE.

◊ Contrôle

Les commandes de contrôle sont les suivantes :

- la logique à 4 ou à 15 valeurs : COERCE;
- l'opération de rétro-annotation : UPDATE;
- le mode de retard : MIN, TYP, MAX, MINTYP, MINMAX et TYPMAX;
- le contrôle de simulation : HALT et FINISH;
- l'initialisation : ARBITRARY.

A.3.3. Simulation de pannes

◊ Spécification de pannes :

Les commandes suivantes permettent de spécifier des pannes dans le circuit :

- STUCK : le collage à 0 et à 1 d'une équipotentielle ou d'un registre;
- STUCK0 : le collage à 0 d'une équipotentielle ou d'un registre;
- STUCK1 : le collage à 1 d'une équipotentielle ou d'un registre;
- SHORT : un court-circuit entre deux équipotentielles;
- OPEN : le collage à 0 et à 1 d'une entrée d'élément;
- OPEN0 : le collage à 0 d'une entrée d'élément;
- OPEN1 : le collage à 1 d'une entrée d'élément;
- OPENZ : le collage à Z d'une entrée d'élément;

- DRIVE : le collage à 0 et à 1 d'une sortie d'élément;
- DRIVE0 : le collage à 0 d'une sortie d'élément;
- DRIVE1 : le collage à 1 d'une sortie d'élément;
- INHIBIT : la panne d'un événement fonctionnel.

Les pannes peuvent être spécifiées automatiquement :

- sur toutes les équipotentielles;
- sur toutes les entrées d'éléments;
- sur toutes les sorties d'éléments.

◊ Dictionnaire diagnostique

Le dictionnaire diagnostique contient une liste de pannes détectées ou non détectées, et une liste de tests.

Les commandes qui s'opèrent sur le dictionnaire diagnostique sont :

- la création du dictionnaire : DIAG;
- la spécification des points de tests : PINS.

◊ Contrôle

Diverses commandes sont définies dans le langage :

- la spécification des pannes par défaut : DEFAULTSET;
- le nombre de détections de pannes : DROP;
- le nombre de détections potentielles de pannes : DROPXFAULTS;
- la simulation de pannes incrémentale : INCREMENTAL.

A.4. Discussions sur les langages Hilo-3

A.4.1. Styles de description

◊ Description non procédurale

Le circuit est décrit de façon non procédurale. Cette nature non procédurale vient du fait que le langage Hilo-3 HDL est un langage très proche des propriétés du matériel, permettant d'exprimer le parallélisme inhérent au circuit.

D'une part, le langage offre un ensemble d'éléments prédéfinis à plusieurs niveaux d'abstraction. D'autre part, le sous-langage FML permet l'utilisation des expressions arithmétiques et logiques, des primitives fonctionnelles, et des instructions gardées. Les expressions et les primitives sont par nature non procédurales. Une instruction gardée se compose des conditions de déclenchement et d'un ensemble d'actions concurrentes, ce qui est également non procédural.

Ces propriétés permettent de fonder le noyau du simulateur sur l'algorithme de l'échéancier.

◊ Descriptions structurelle et comportementale

Ces deux types de descriptions représentent respectivement deux aspects d'une architecture : sa structure topologique et son comportement algorithmique.

L'aspect comportemental du circuit est décrit par des modèles fonctionnels, tandis que l'aspect structurel décrit par des primitifs et des sous-circuits, ainsi que leurs interconnexions. Ces deux types de descriptions permettent la vérification de la cohérence et de l'équivalence des caractéristiques structurelles et comportementales du circuit.

Dans le langage HDL, la description structurelle est basée sur des primitives prédéfinies aux niveaux interrupteur, logique et fonctionnel. Tandis que la description comportementale porte essentiellement sur des structures gardées.

Il n'existe pas de description procedurale. Il est donc impossible de

décrire le circuit de façon algorithmique et abstraite. En fait, la description comportementale est fondée sur les instructions déclenchées par des conditions de garde. C'est une notion très proche de la modélisation au niveau du transfert de registre.

◊ Description dirigée par le flot de données

L'interprétation du langage de description du circuit dépend des flots de données. Contrairement à une description dirigée par le flot de contrôle qui représente un organigramme dans lequel les opérations sont ordonnées selon le graphe de contrôle, l'évaluation des éléments logiques ou fonctionnels est activée par des signaux ou des événements.

Les opérations sur les éléments structurels s'effectuent dès que les données sont disponibles. Quant aux éléments fonctionnels, leur évaluation est déclenchée par des séquences d'événements.

◊ Description multi-niveaux

Le langage offre trois niveaux d'abstraction :

- le niveau logique : les portes logiques;
- le niveau interrupteur : les transistors bidirectionnels;
- le niveau fonctionnel : les expressions, les primitives fonctionnelles et les instructions gardées.

La modélisation au niveau interrupteur est assez restrictive. Seules deux primitives bidirectionnelles et deux primitives unidirectionnelles sont définies pour modéliser les transistors.

Au niveau fonctionnel, les objets représentent plus ou moins les propriétés structurelles du circuit :

- Les expressions arithmétiques et logiques représentent des circuits combinatoires.
- Les primitives fonctionnelles correspondent à des composants du circuit.
- Les instructions gardées sont proches des structures du transfert de registres.

A.4.2. Structuration de données

◊ Modularité

La notion de circuit est la seule unité modulaire. L'interface du circuit définit les ports de connexion et les paramètres formels. Les échanges de données et les communications ne passent que par les ports de connexion. Les ports de connexion peuvent être une entrée, une sortie ou bidirectionnel.

Il n'existe pas de notion de versions alternatives du circuit qui partagent la même interface du circuit.

Les langages Hilo-3 s'adaptent à l'environnement de simulation interactif. Les objets définis à l'intérieur du circuit peuvent être référencés par des commandes de rétro-annotation. Le corps du circuit est modifiable après la compilation et visible durant la simulation. Ce sont des propriétés importantes qui permettent de simuler le circuit sous différentes dérivations de conditions physiques, et de visualiser les flots de données de l'intérieur du circuit.

La définition des modules ne peut être imbriquée. Il n'est pas possible de définir, dans un circuit, des sous-circuits propres à lui pour délimiter la visibilité de ceux-ci. Un module est identifiable à une portée globale.

◊ Modélisation générique

Les modèles primitifs ou définis par l'utilisateur sont génériques. Ces modèles sont considérés comme des types caractérisant un ensemble d'instances.

En ce qui concerne les primitives prédéfinies, leur instantiation se fait par l'affectation des attributs. Cependant, pour une primitive, les attributs affectables sont limités aux paramètres de retard et de force. Le reste des attributs est figé par la sémantique du langage. Il est impossible de créer de nouveaux types en leur associant des attributs. Le domaine d'utilisation des langages Hilo-3 est limité aux technologies connues par ces langages.

Les modèles génériques créés par l'utilisateur sont basés sur la notion de circuit. Un circuit est une unité pouvant être considérée comme un modèle générique, lorsqu'il est utilisé dans un circuit englobant. Pourtant, un

modèle générique ainsi défini est très simpliste. D'une part, il n'existe pas de relation entre les modèles génériques permettant l'héritage des attributs entre les modèles. D'autre part, il n'existe pas de relation de dépendance ou d'englobement entre les modèles génériques.

Pour les modèles créés par l'utilisateur, il n'est pas possible non plus d'associer des attributs abstraits à ces modules, tel que l'utilisation des assertions qui existent dans certains langages de description du matériel pour les vérification statiques (lors de l'instantiation) ou dynamiques (lors de la simulation) des instances créées.

◊ Hiérarchie

Un circuit complexe peut être décrit en une structure hiérarchique de sous-circuits.

La hiérarchisation des circuits est réalisée par les trois mécanismes suivants :

• les modules : Ce sont les circuits définis par l'utilisateur de manière structurelle ou comportementale.
• les interconnexions : Les équipotentielles relient les modules.
• les ports de connexion : Chaque circuit possède sa propre interface. Les règles de communication des ports sont basées sur la compatibilité de types des équipotentielles connectées à ces ports.

Cette hiérarchisation est dynamique. Statiquement, les circuits n'ont pas de relation de dépendance. L'organisation des bibliothèques est simple : une bibliothèque est un ensemble de circuits, sans configuration arborescente ni dépendance hiérarchique. Et les relations hiérarchiques entre les circuits sont établies dynamiquement lors d'une session de simulation. Cela procure des avantages :

• La modification du sous-circuit n'entraîne pas une recompilation globale des circuits utilisant ce sous-circuit. Cette nature rend un environnement de simulation très interactif et facile à gérer.
• Un circuit peut avoir plusieurs versions distinctes définies dans différentes bibliothèques. La sélection de la version désirée du circuit est réalisée par l'utilisation de la bibliothèque correspondante.

A.4.3. Caractéristiques

◊ Parallélisme

Contrairement à certains langages trop procéduraux, les langages de Hilo-3 assurent que les tâches de simulation peuvent être décentralisées et parallélisées, et que le circuit peut être partitionné et simulé sur des unités de simulation.

La description structurelle (les transistors et les portes logiques) est parallélisable. La description comportementale l'est également puisqu'elle est basée sur les instructions gardées déclenchées par des événements, et que les opérations définies dans les instructions gardées sont non procédurales.

◊ Nature temporelle

Les langages permettent d'exprimer la nature temporelle du circuit par l'association des contraintes temporelles aux éléments et par la temporisation des opérations.

Pour ce qui concerne la description des contraintes temporelles du circuit, les éléments et les équipotentielles peuvent être affectés des retards. Ce sont des retards de montée et de descente, chacun pouvant avoir trois valeurs : minimale, typique, maximale. De plus, les retards sont rectifiés automatiquement selon la sortance et la capacité des éléments et des équipotentielles.

Pour ce qui concerne la temporisation des opérations, les mécanismes suivants sont utilisés :

• le mécanisme dirigé par l'événement : Le langage HDL utilise la structure gardée pour le déclenchement des opérations.
• le mécanisme dirigé par le temps : Le langage WDL associe à toutes les opérations un temps de déclenchement.

◊ Description proche des propriétés physiques du circuit

A l'opposé de certains langages de description trop classiques et trop indépendants du circuit, les langages Hilo-3 sont orientés vers la simulation et le test du circuit. Bien que les langages soient très proches du matériel et

que leur indépendance sémantique soit très faible, la richesse des primitives prédéfinies et la nature non procédurale des modèles fonctionnels permettent de réaliser d'une manière efficace des simulateurs et des environnements de simulation associés.

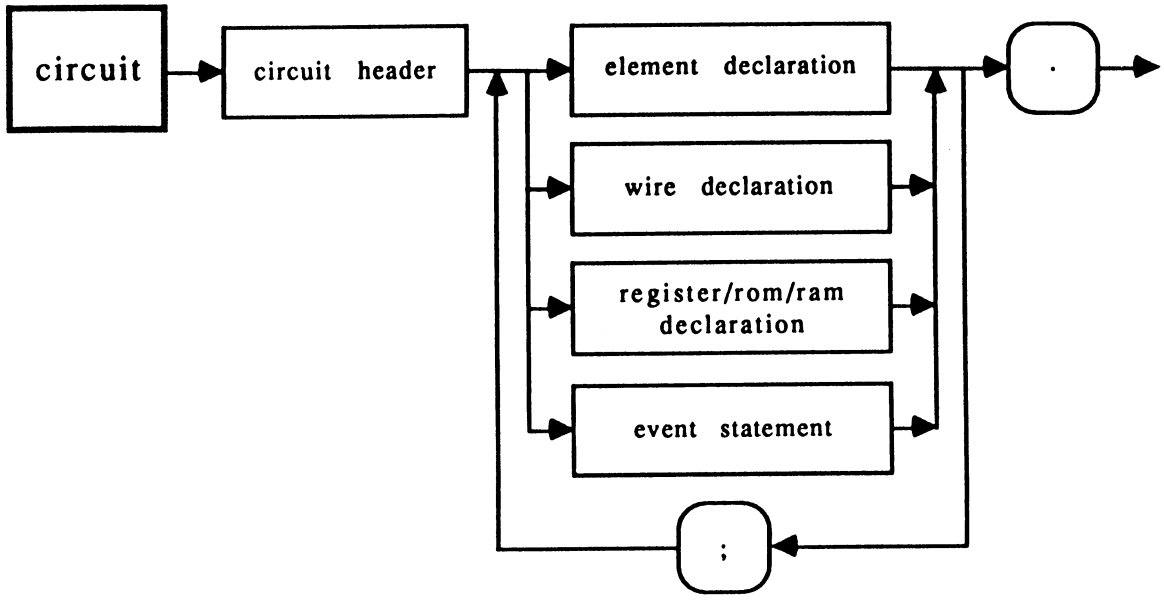
La nature proche des propriétés physiques du circuit rend les langages très sensibles aux technologies de fabrication de circuits. Les langages Hilo-3 ne permettent pas de créer de nouveaux types. Il est difficile de personnaliser le langage en fonction des technologies. Son extensibilité est dont relativement faible.

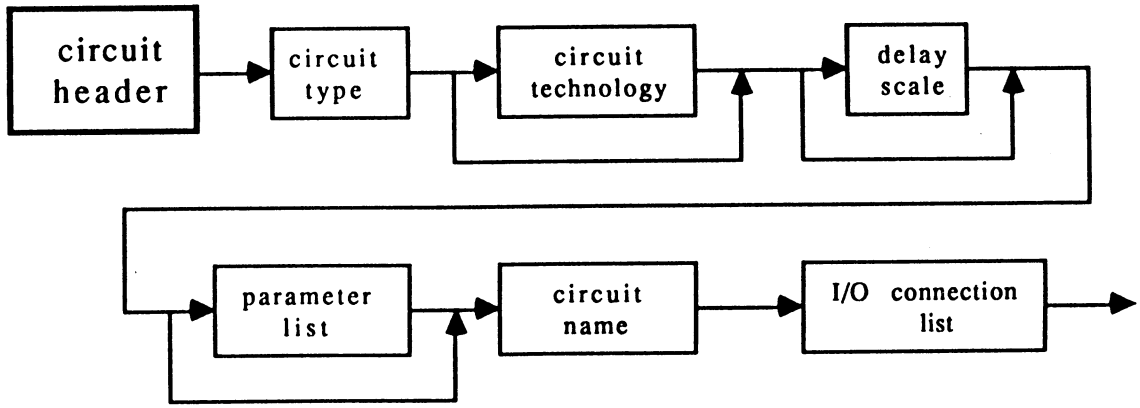
D'autre part, comme il est proche du matériel, le niveau d'abstraction ne peut être très élevé. Pour la modélisation fonctionnelle, il n'existe pas de description purement comportementale basée sur des algorithmes abstraits sans tenir compte des caractéristiques structurelles du circuit.

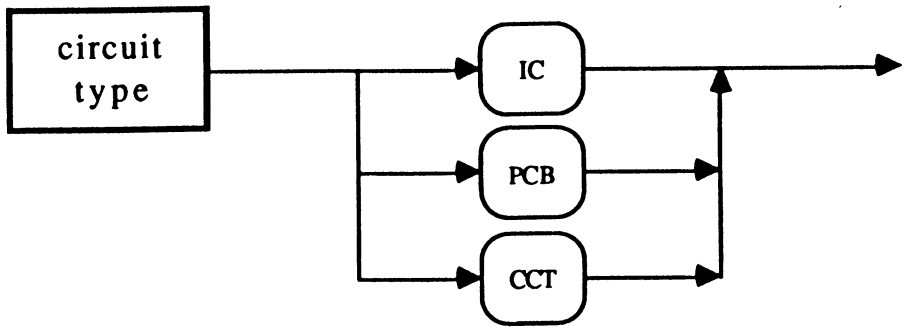
Annexe B

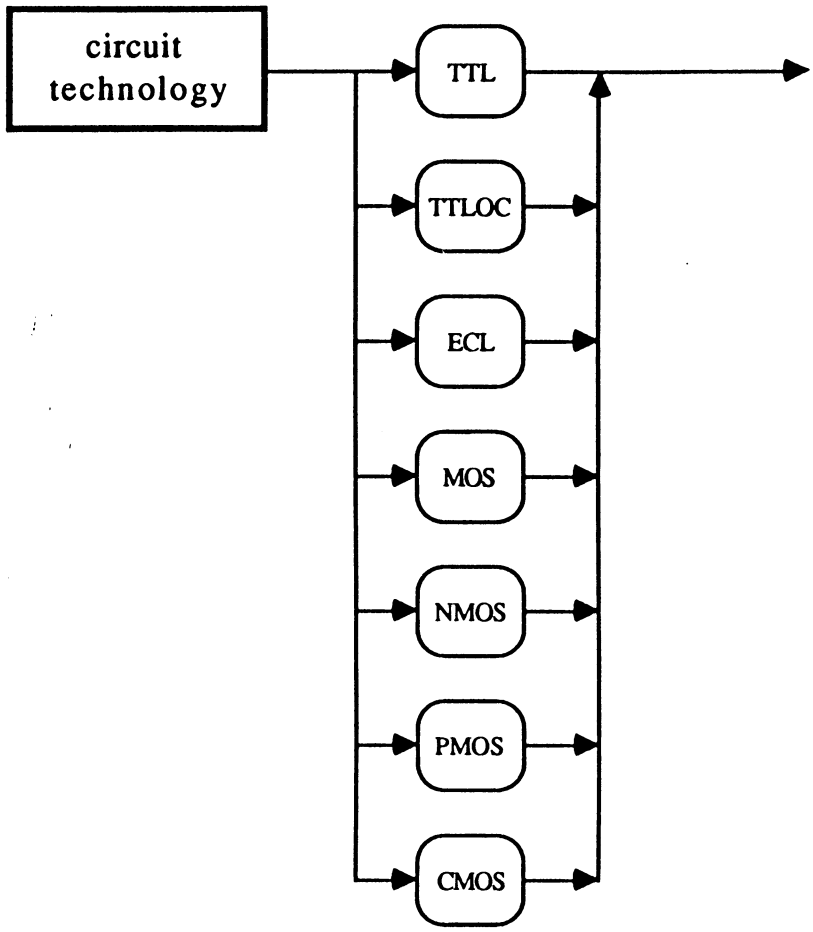
Syntaxe de HDL et WDL de Hilo-3

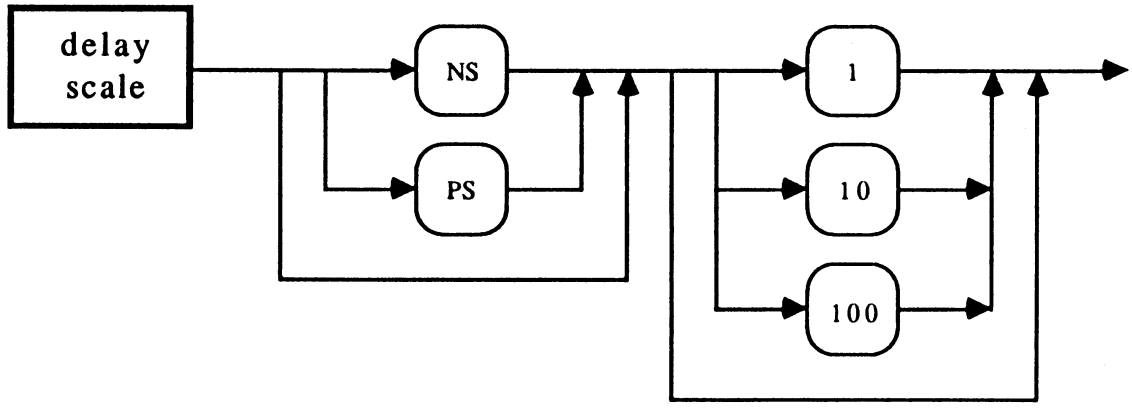
B.1. Circuit

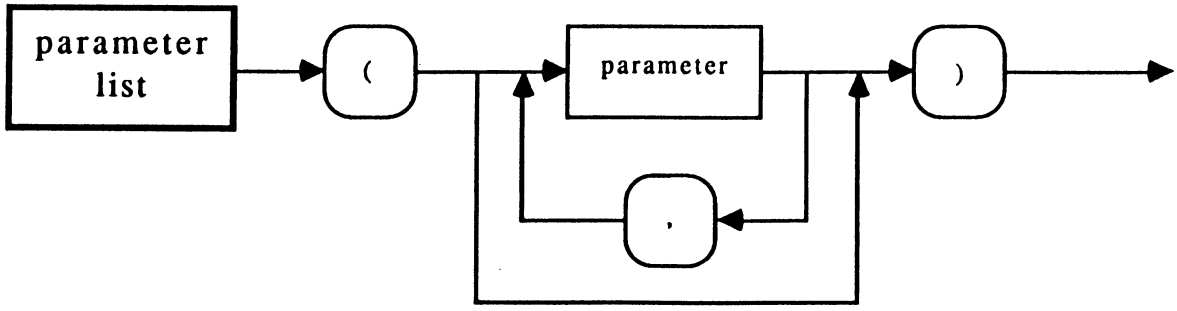

circuit
circuit header
element declaration
wire declaration
register/rom/ram declaration
event statement
.
;

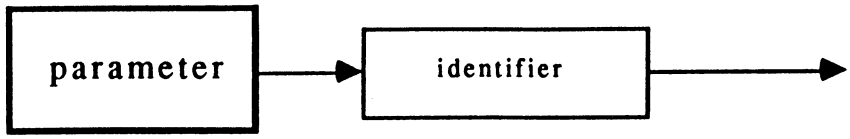

circuit header
circuit type
circuit technology
delay scale
parameter list
circuit name
I/O connection list

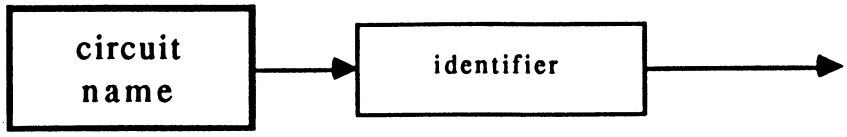

circuit type
IC
PCB
CCT

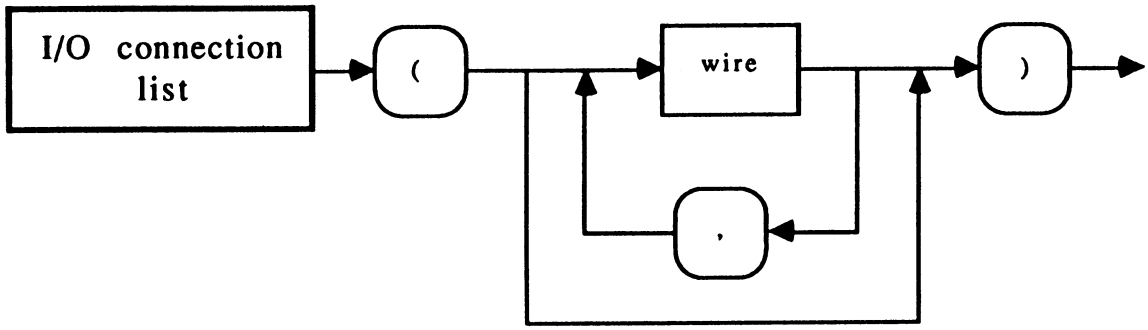

circuit technology
TTL
TTLOC
ECL
MOS
NMOS
PMOS
CMOS

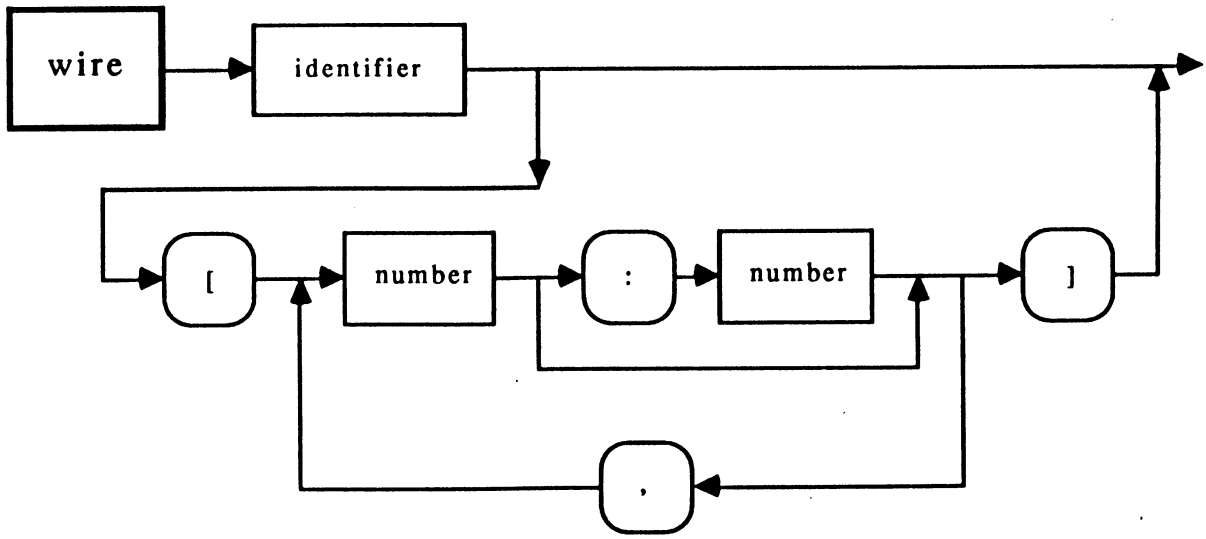

delay scale
NS
PS
1
10
100

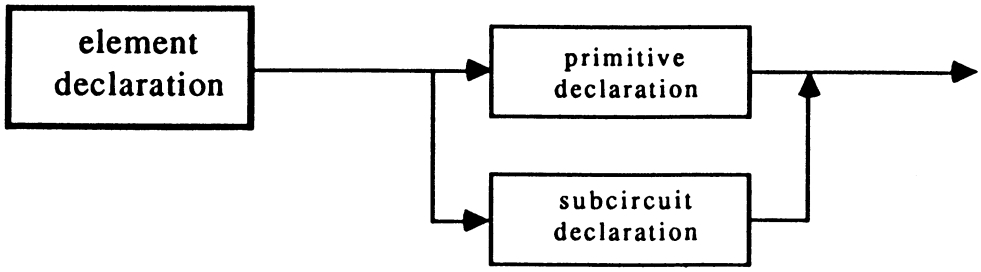

parameter list
(
parameter
)
,

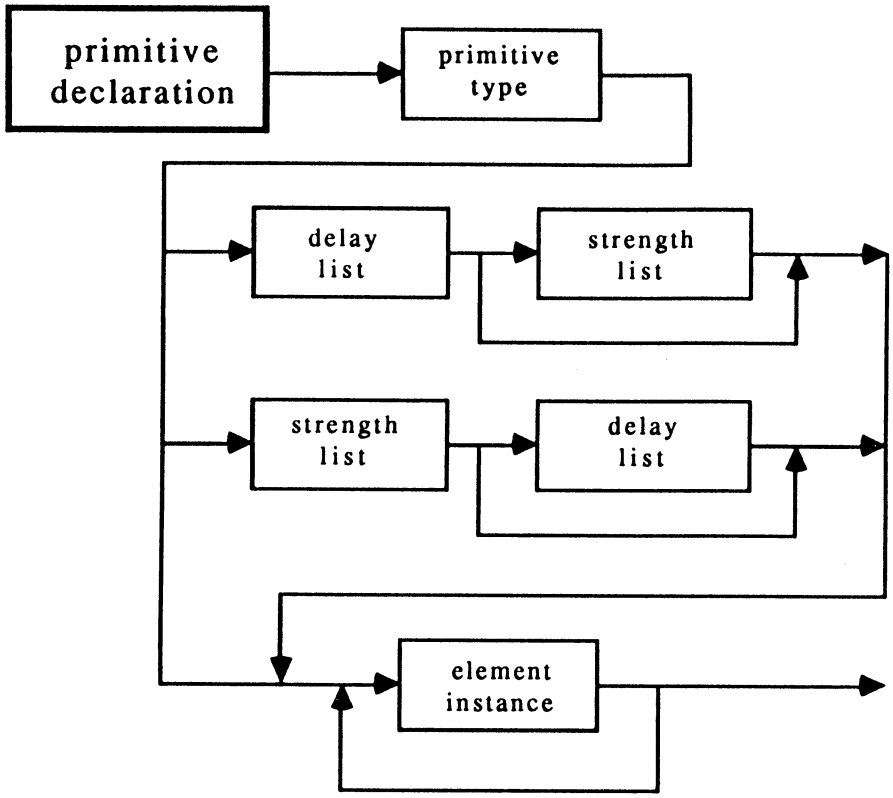

parameter
identifier

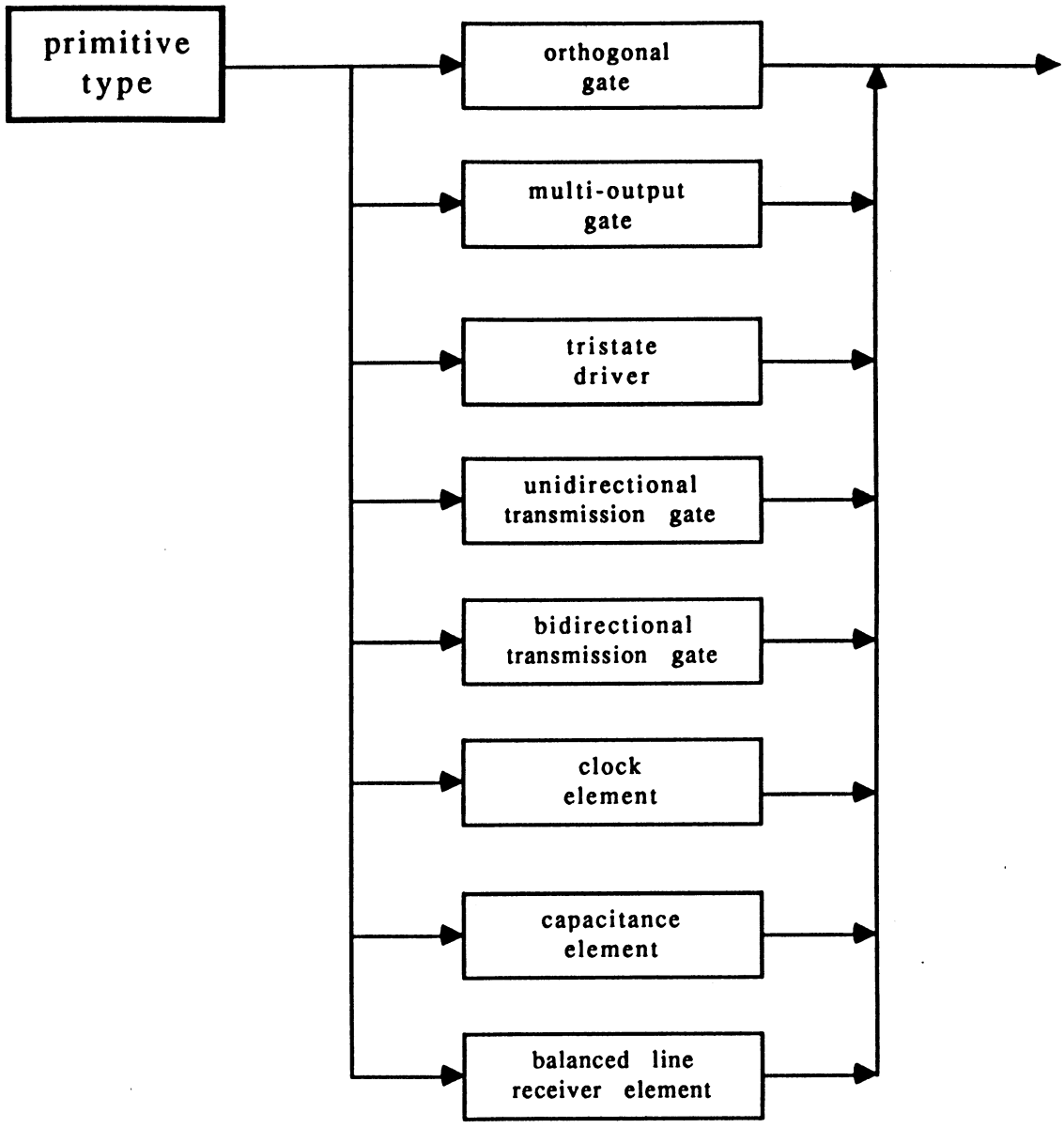

circuit name
identifier

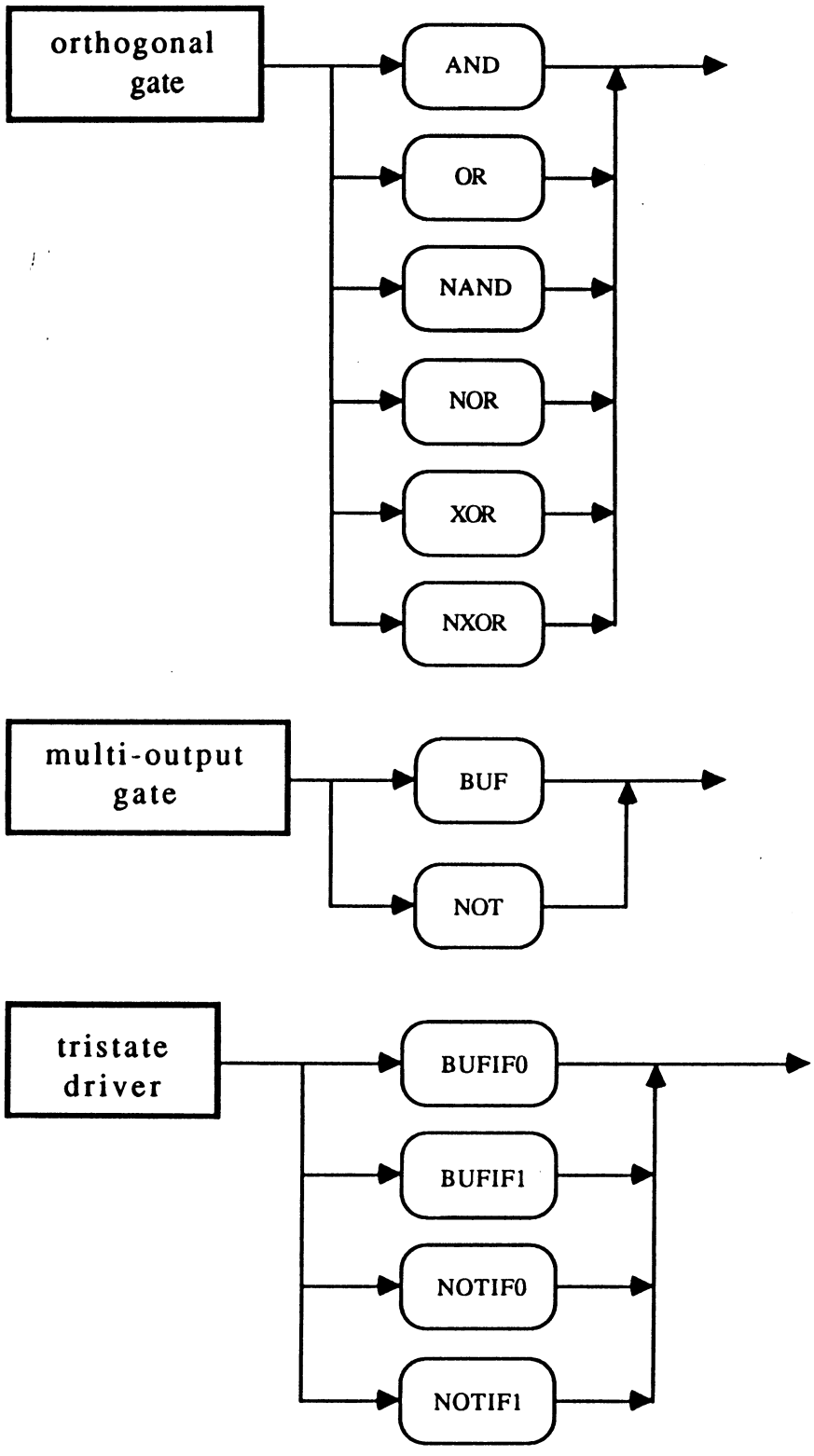

I/O connection list
(
wire
)
,

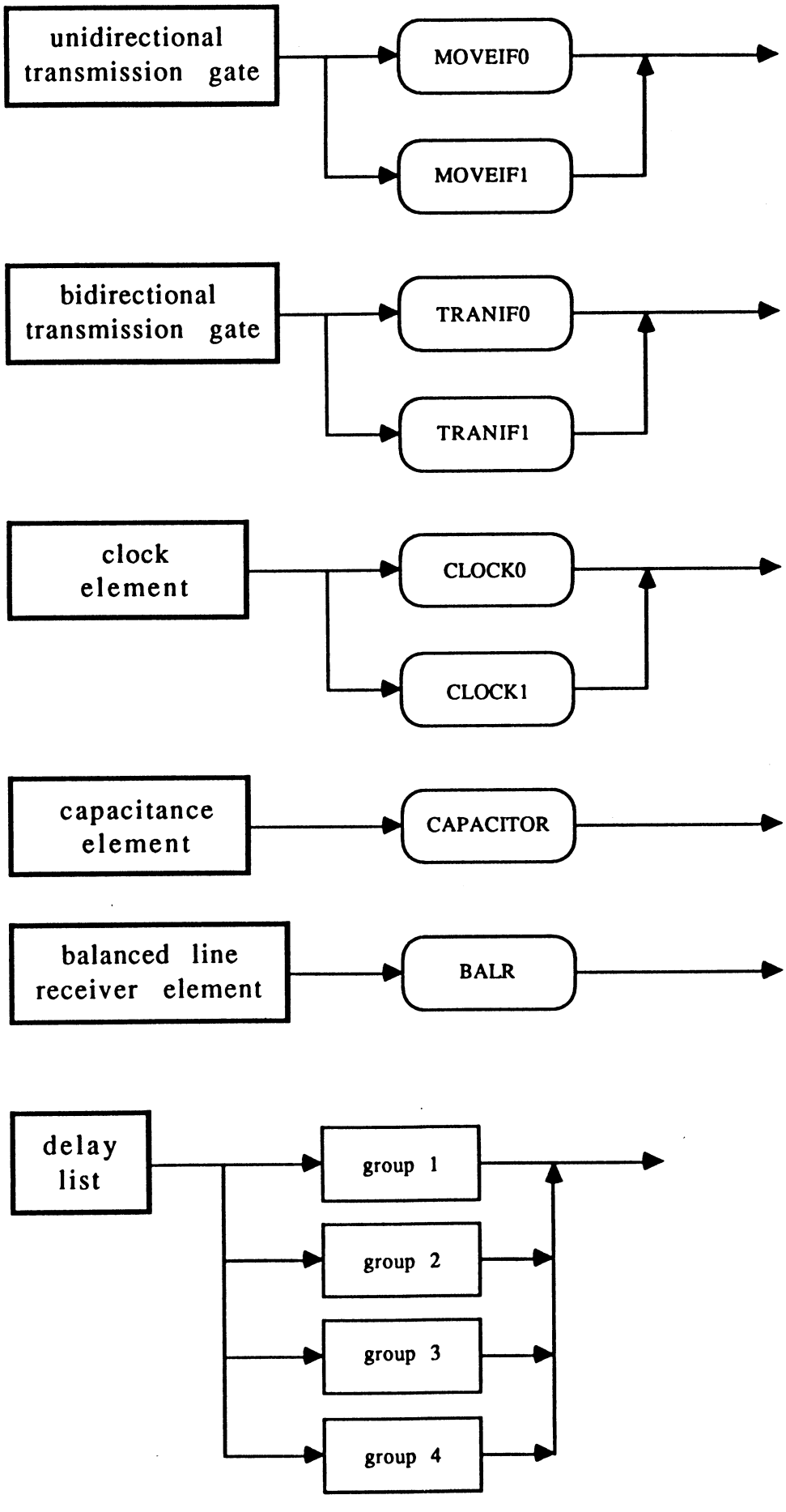

wire
identifier
[
number
:
number
]
,

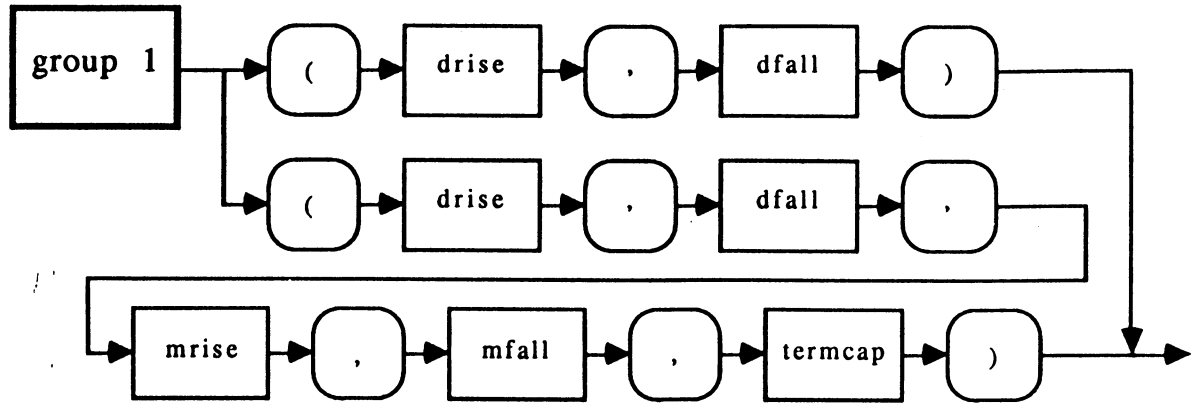

element
declaration
primitive
declaration
subcircuit
declaration

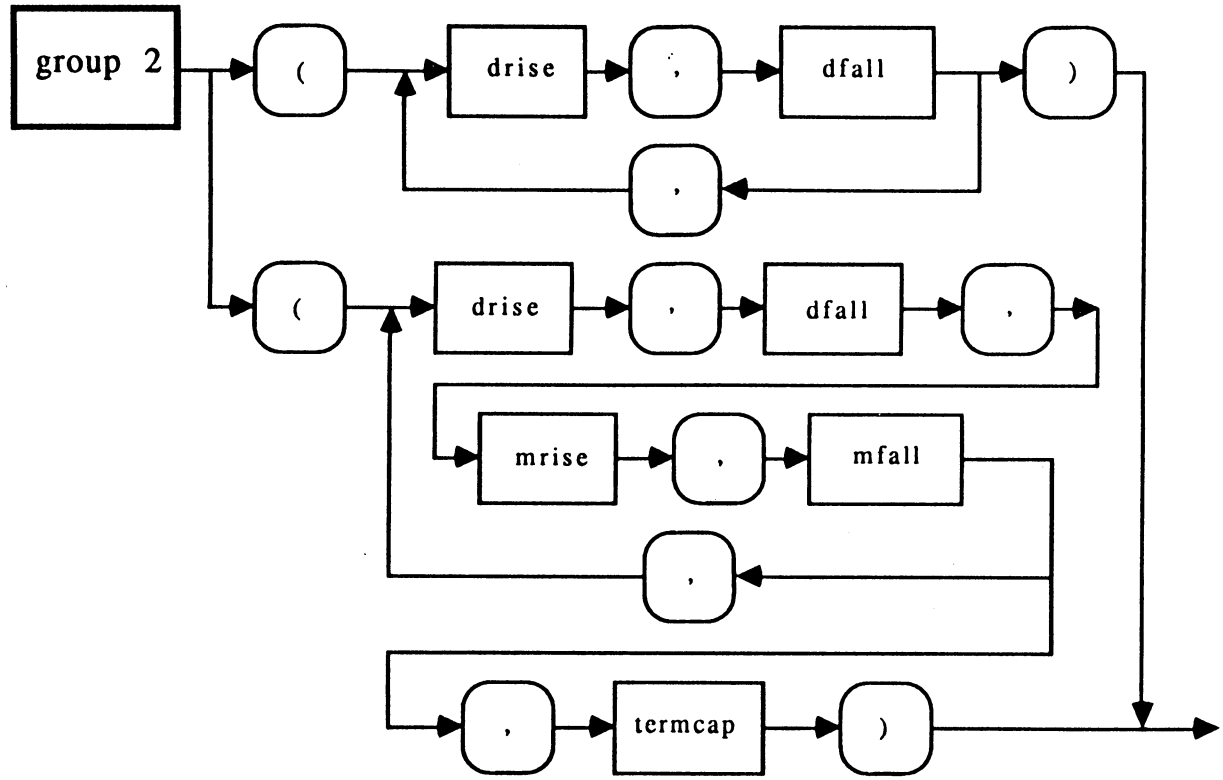

primitive
declaration
primitive
type
delay
list
strength
list
strength
list
delay
list
element
instance

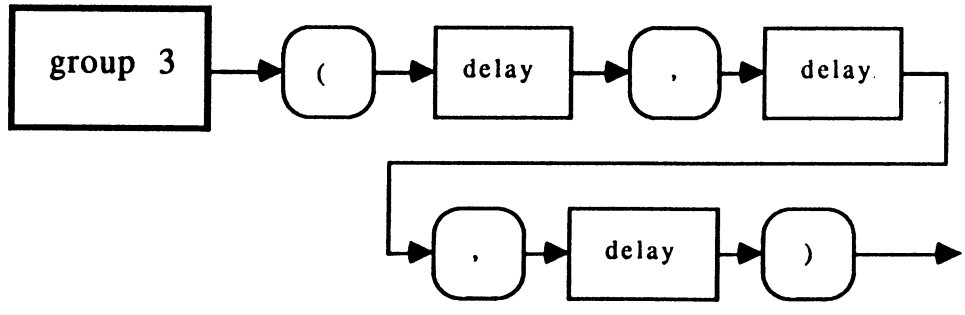

primitive type
orthogonal gate
multi-output gate
tristate driver
unidirectional transmission gate
bidirectional transmission gate
clock element
capacitance element
balanced line receiver element

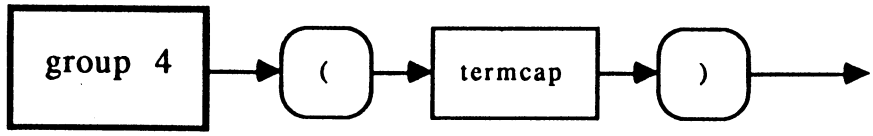

orthogonal gate
AND
OR
NAND
NOR
XOR
NXOR
multi-output gate
BUF
NOT
tristate driver
BUFIF0
BUFIF1
NOTIF0
NOTIF1

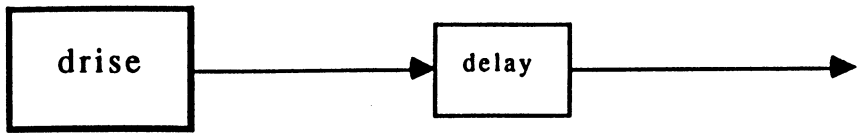

unidirectional transmission gate
MOVEIF0
MOVEIF1
bidirectional transmission gate
TRANIF0
TRANIF1
clock element
CLOCK0
CLOCK1
capacitance element
CAPACITOR
balanced line receiver element
BALR
delay list
group 1
group 2
group 3
group 4

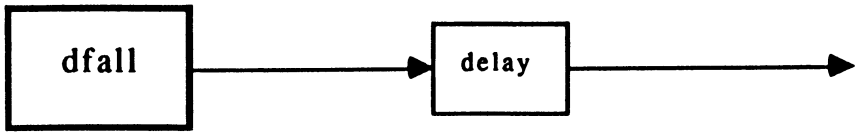

group 1
(
drise
,
dfall
)
(
drise
,
dfall
,
mrise
,
mfall
,
termcap
)

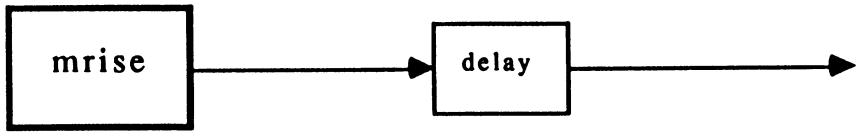

group 2
(
drise
,
dfall
)
,
(
drise
,
dfall
,
mrise
,
mfall
,
,
termcap
)

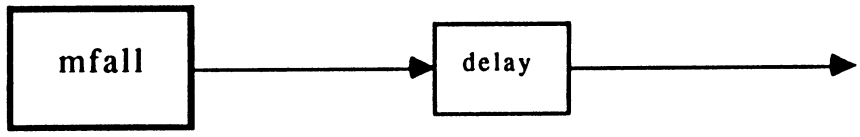

group 3
(
delay
,
delay
,
delay
)

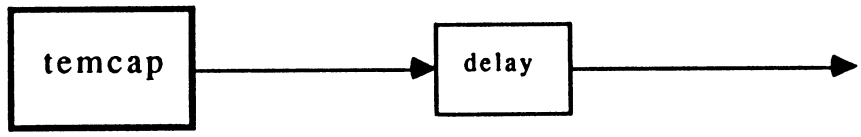

group 4
(
termcap
)

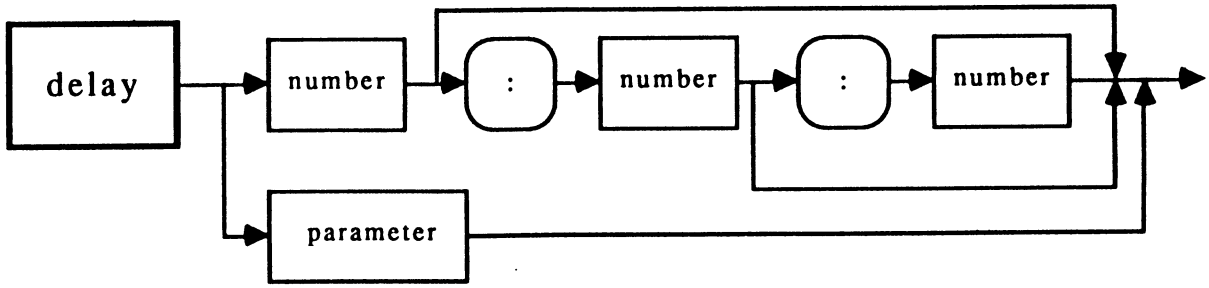

drise
delay

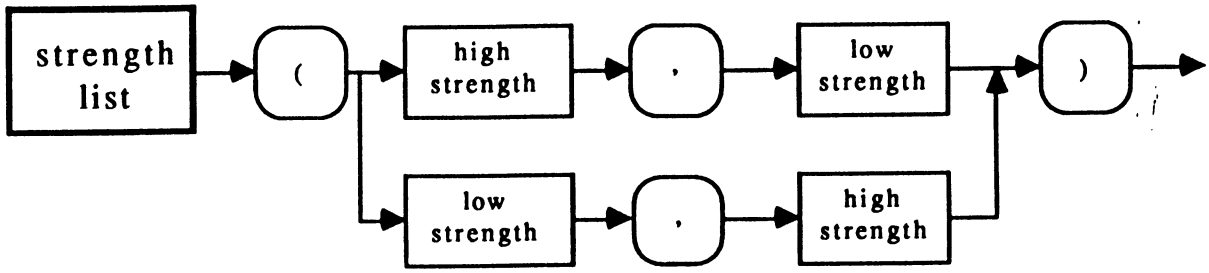

dfall
delay

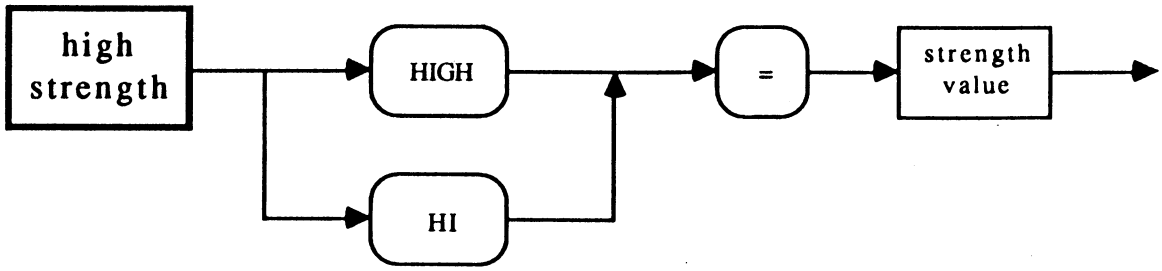

mrise
delay

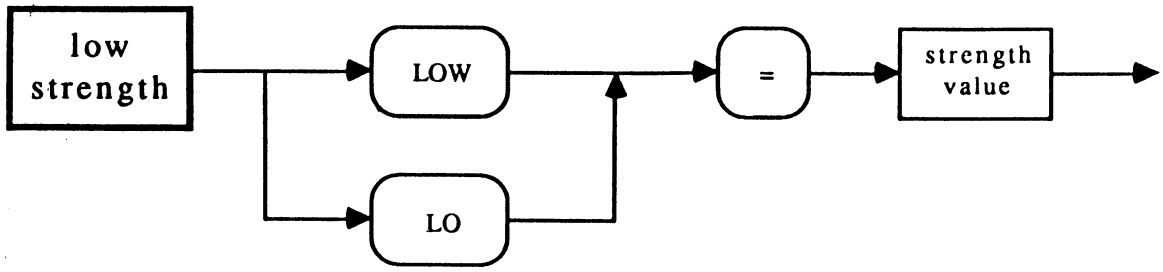

mfall
delay

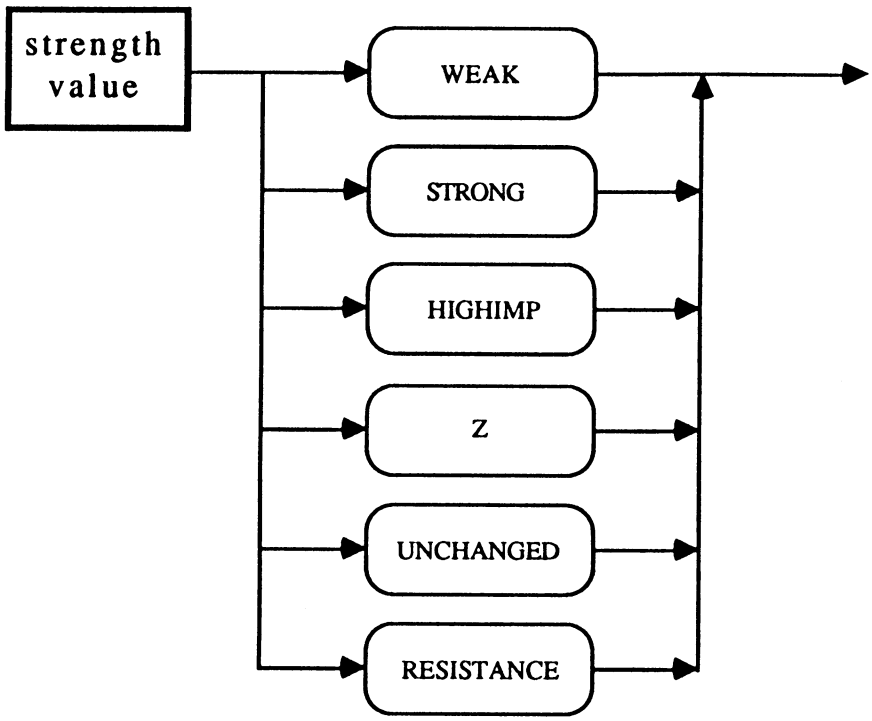

temcap
delay

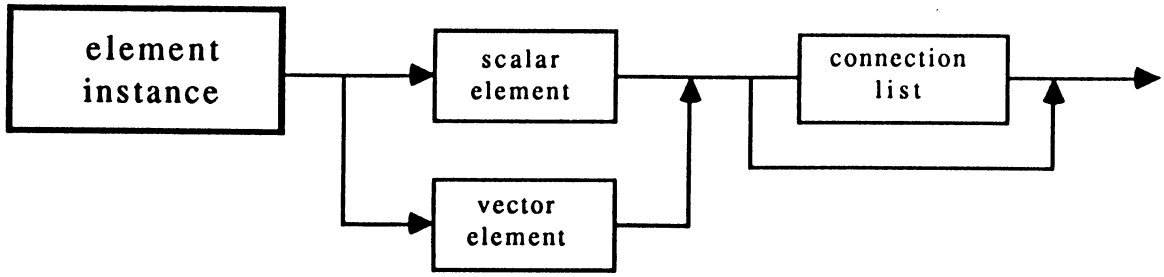

delay
number
:
number
:
number
parameter

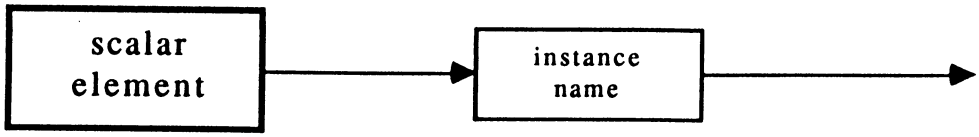

strength list
(
high strength
,
low strength
)
low strength
,
high strength

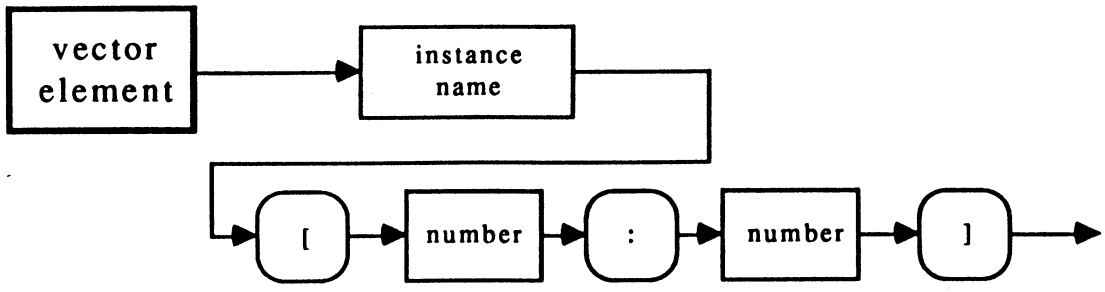

high strength
HIGH
HI
=
strength value

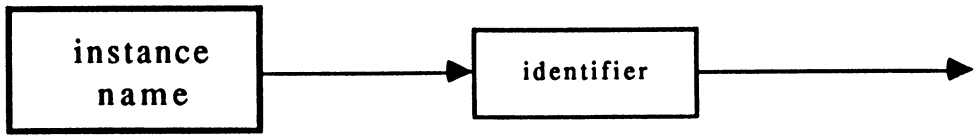

low strength
LOW
LO
=
strength value

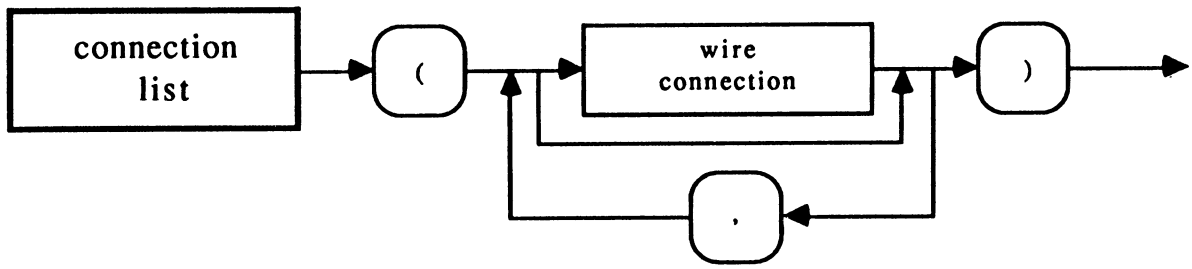

strength value
WEAK
STRONG
HIGHIMP
Z
UNCHANGED
RESISTANCE

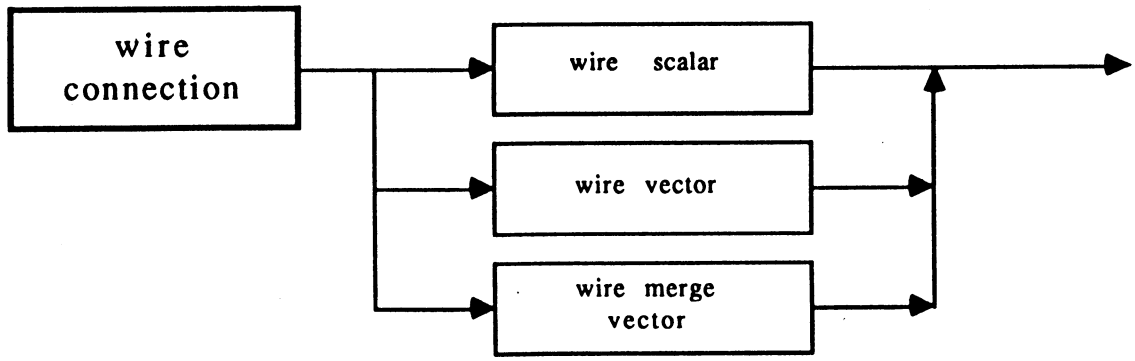

element instance
scalar element
vector element
connection list

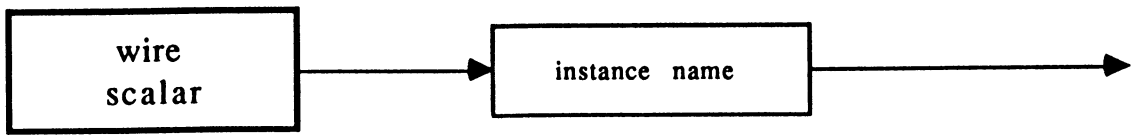

scalar element
instance name

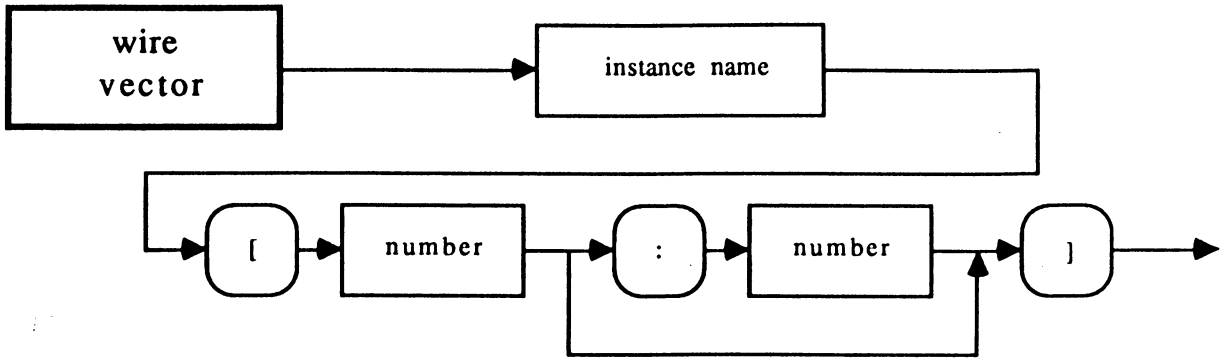

vector element
instance name
[
number
:
number
]

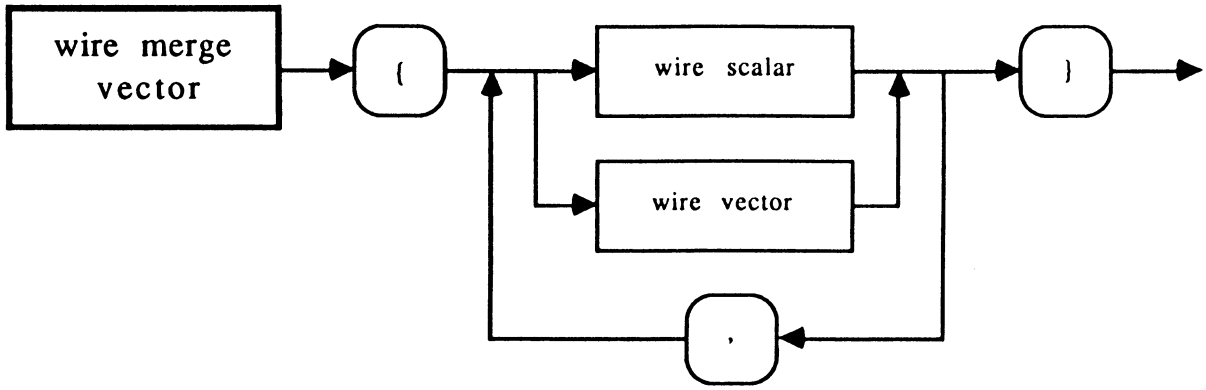

instance name
identifier

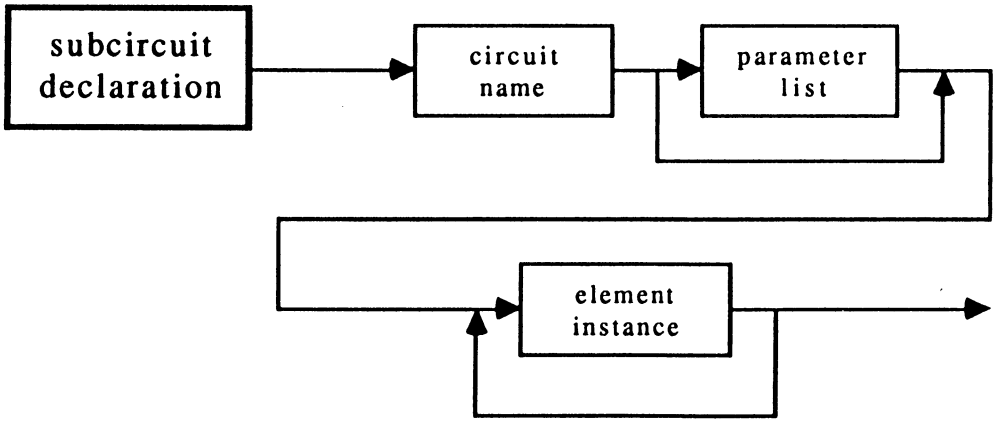

connection list
(
wire connection
)
,

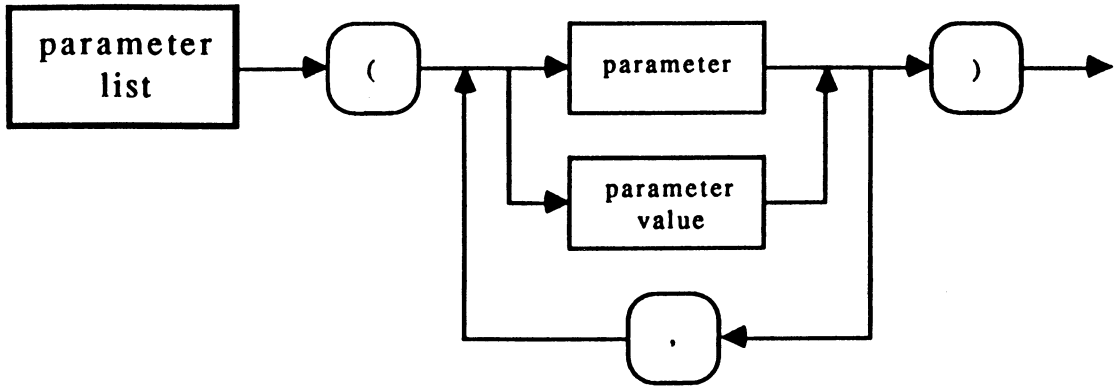

wire connection
wire scalar
wire vector
wire merge vector

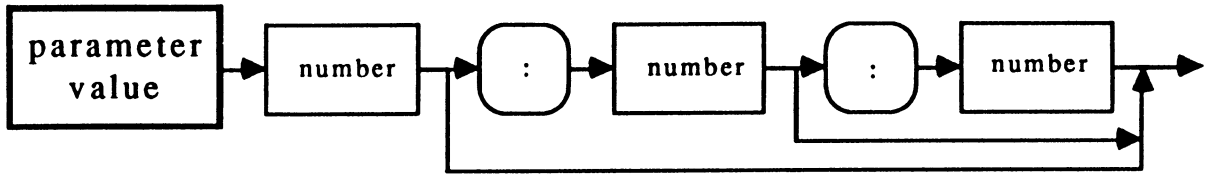

wire scalar
instance name

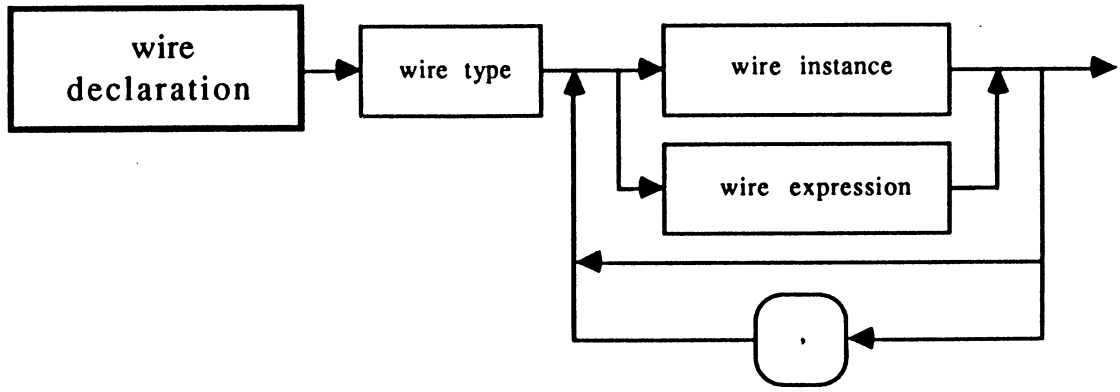

wire
vector
instance name
[
number
:
number
]

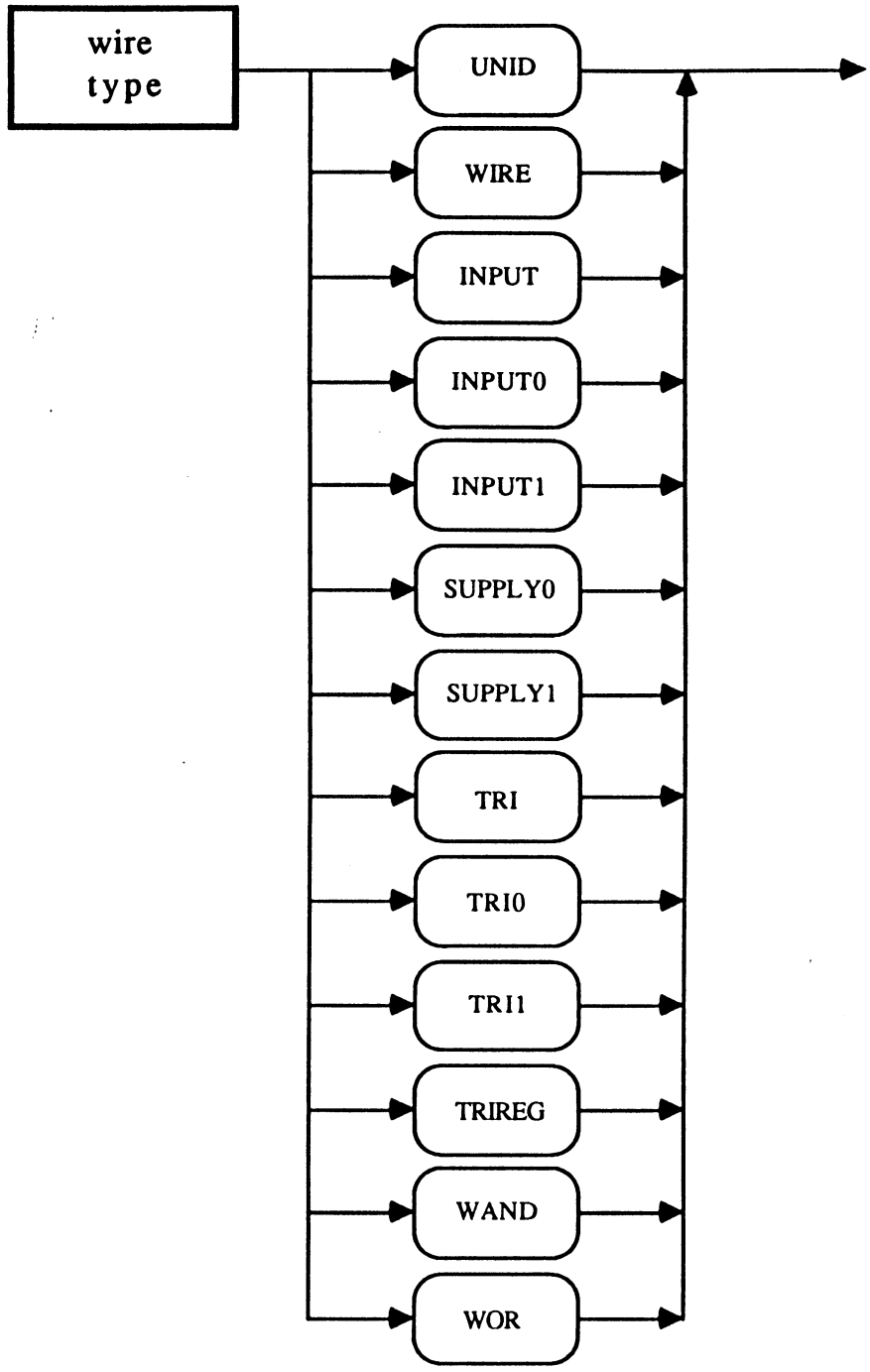

wire merge
vector
{
wire scalar
wire vector
,
}

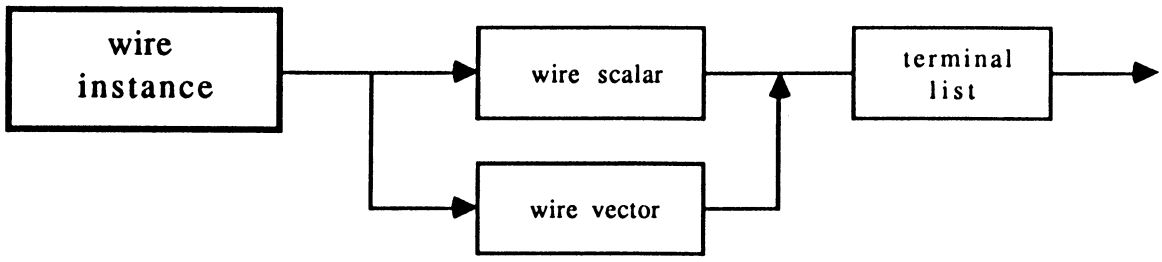

subcircuit
declaration
circuit
name
parameter
list
element
instance

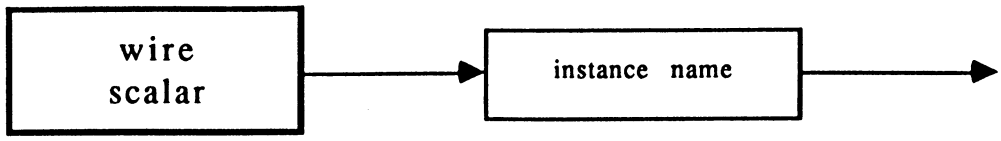

parameter list
(
parameter
parameter value
)
,

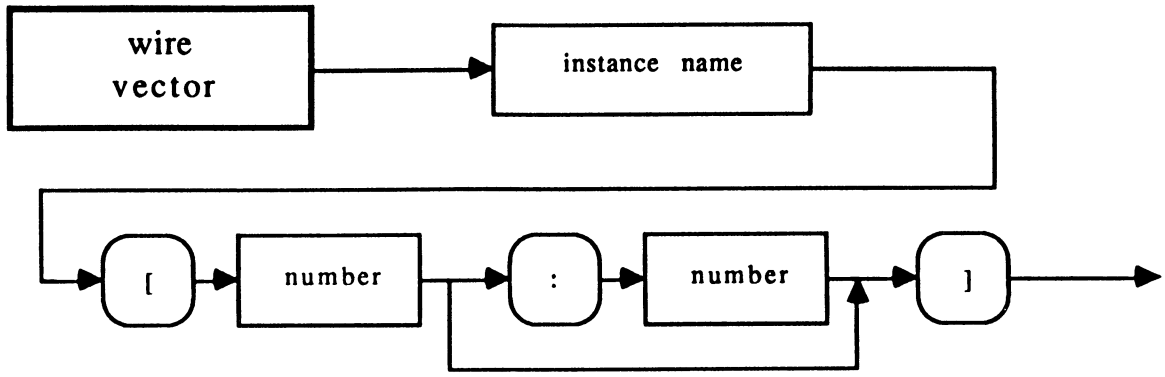

parameter value
number
:
number
:
number

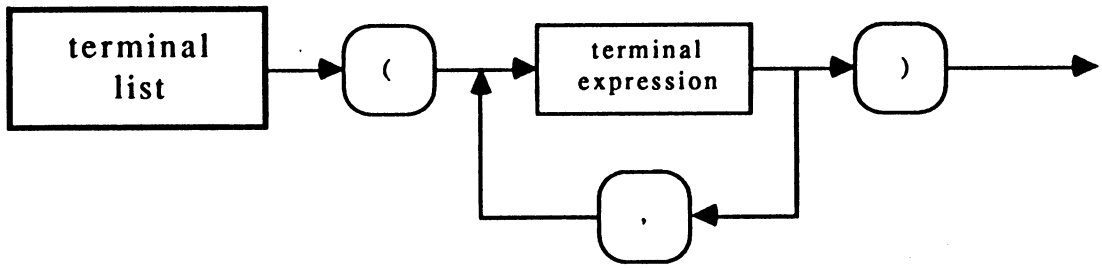

wire declaration
wire type
wire instance
wire expression
,

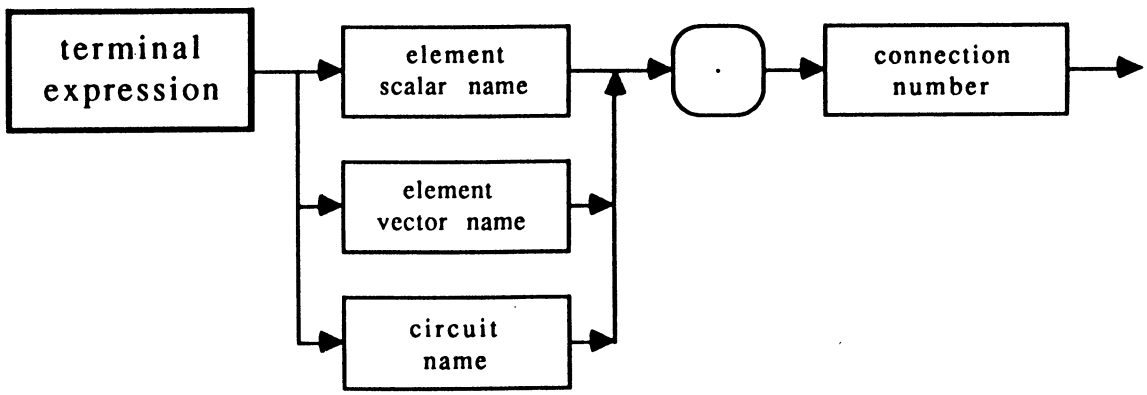

wire
type
UNID
WIRE
INPUT
INPUT0
INPUT1
SUPPLY0
SUPPLY1
TRI
TRI0
TRI1
TRIREG
WAND
WOR

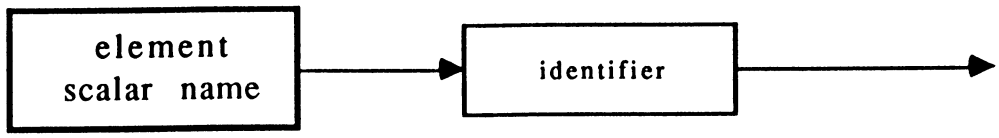

wire
instance
wire scalar
wire vector
terminal
list

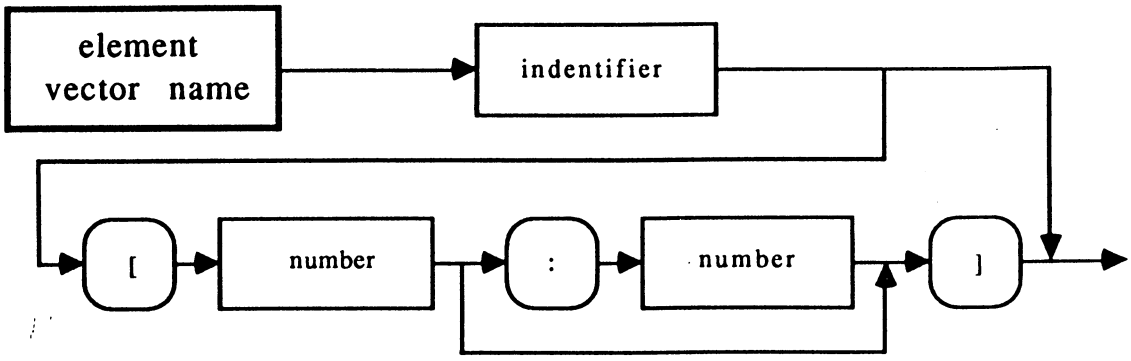

wire
scalar
instance name

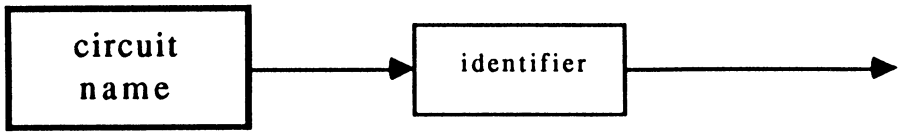

wire
vector
instance name
[
number
:
number
]

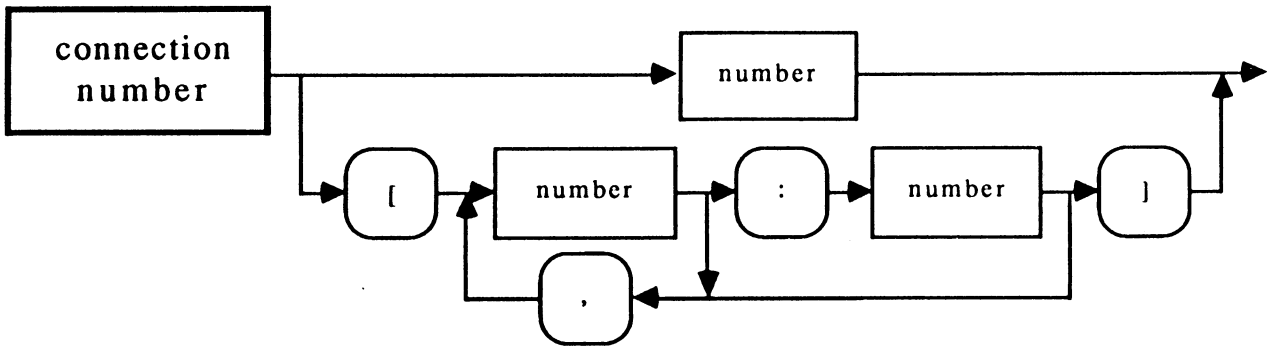

terminal
list
(
terminal
expression
)
,

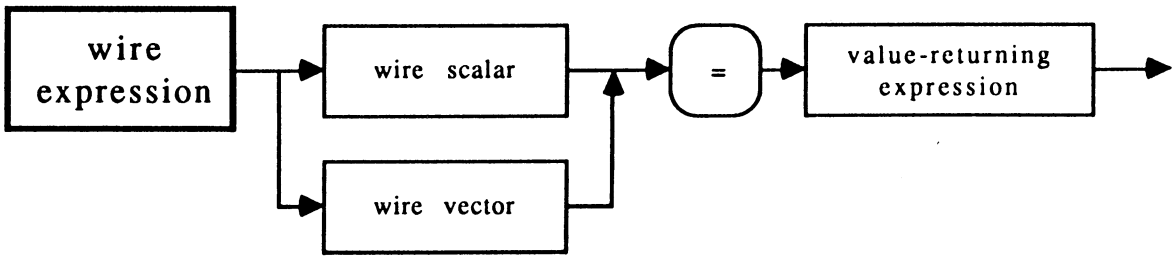

terminal
expression
element
scalar name
.
connection
number
element
vector name
circuit
name

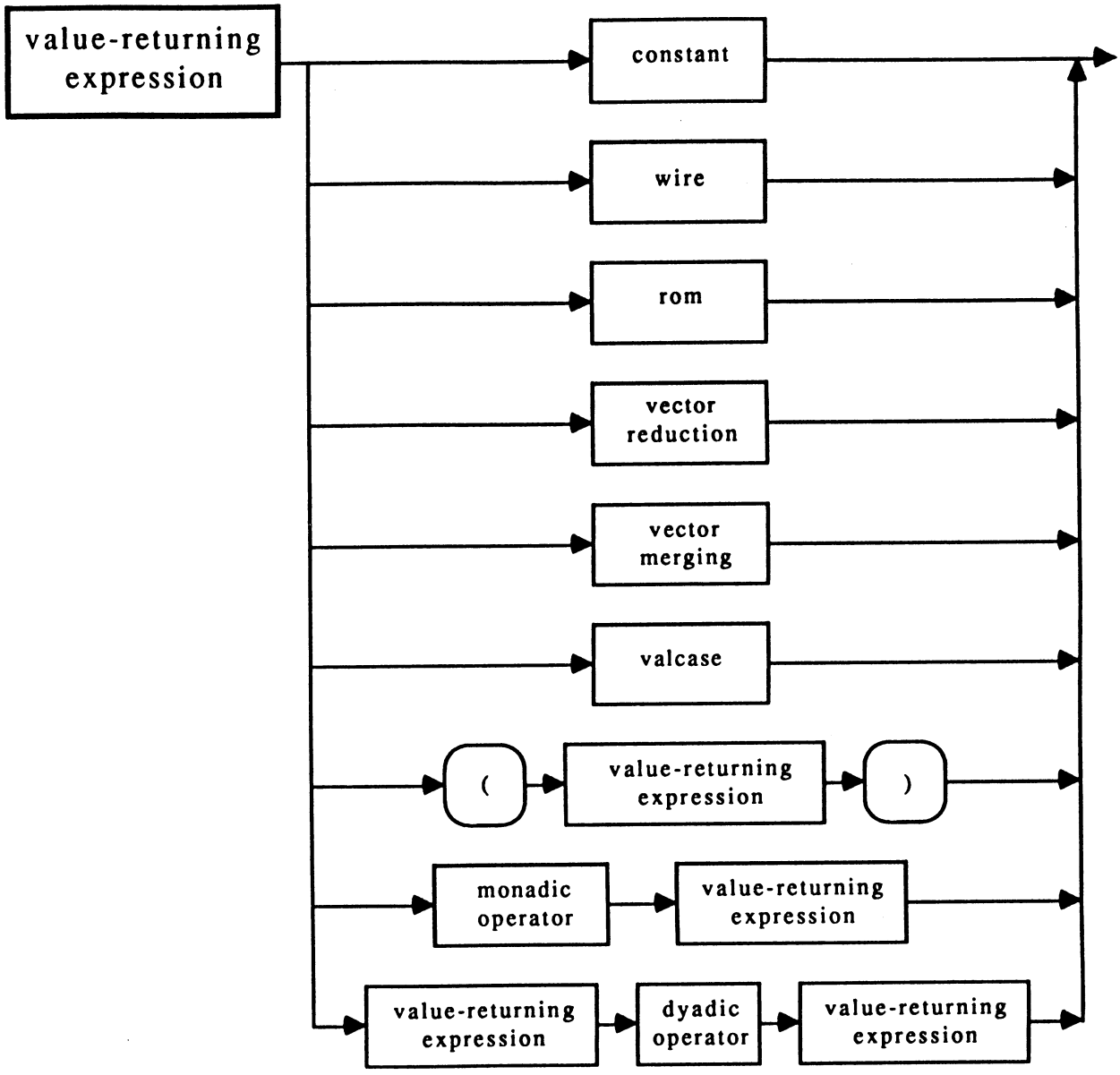

element
scalar name
identifier

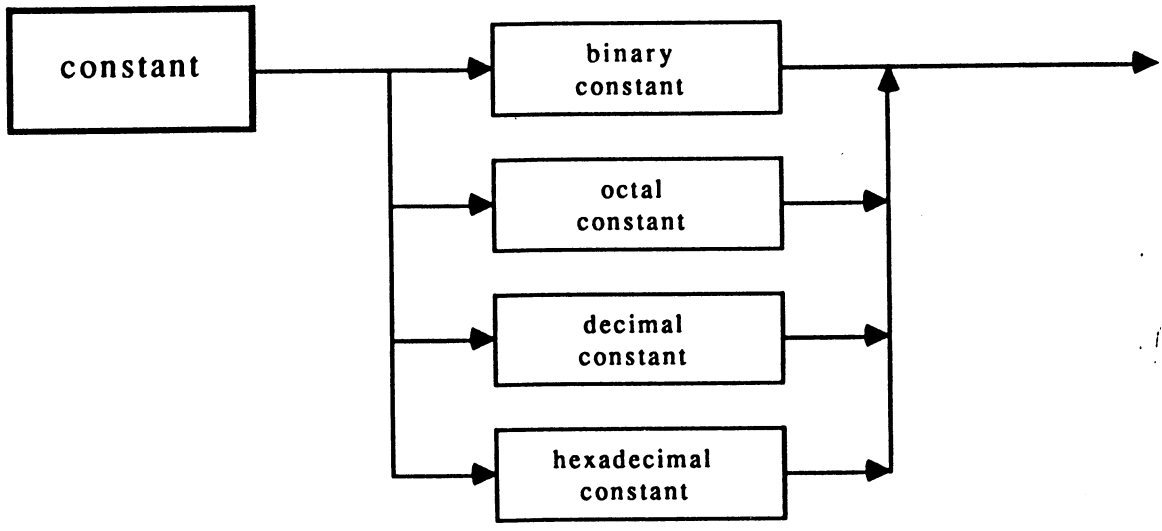

element
vector name
indentifier
[
number
:
number
]

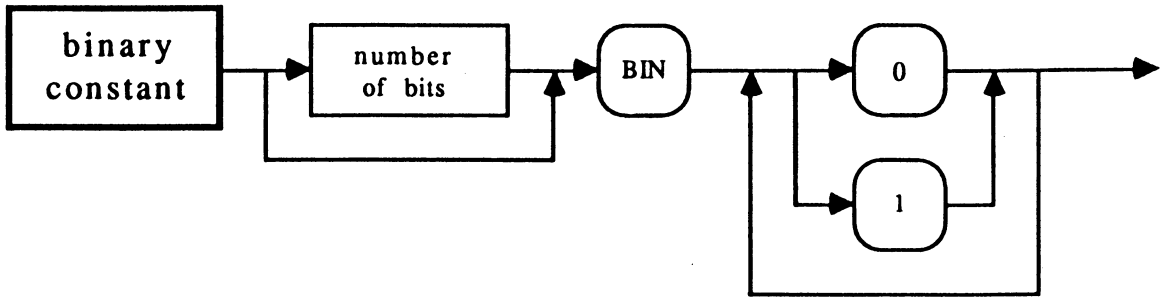

circuit
name
identifier

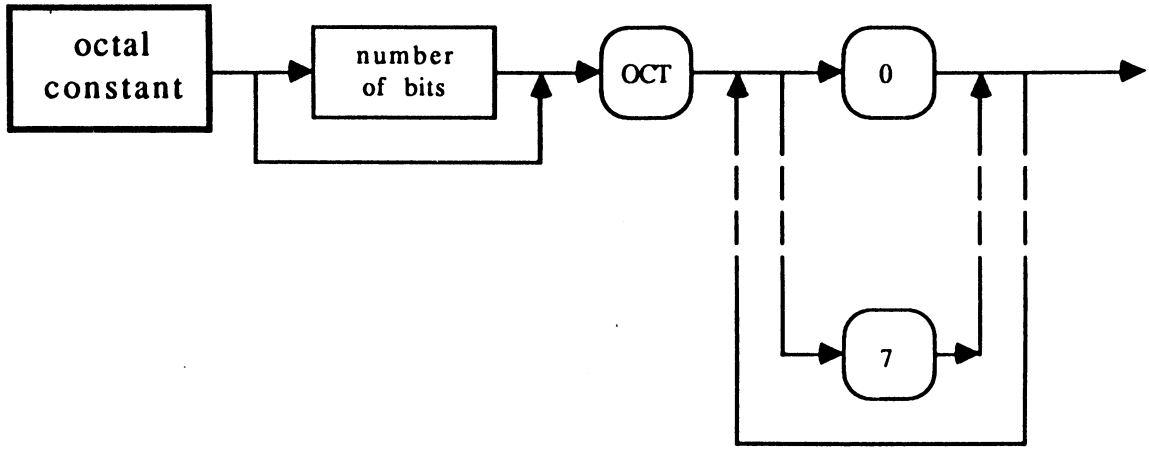

connection
number
number
[
number
:
number
]
,

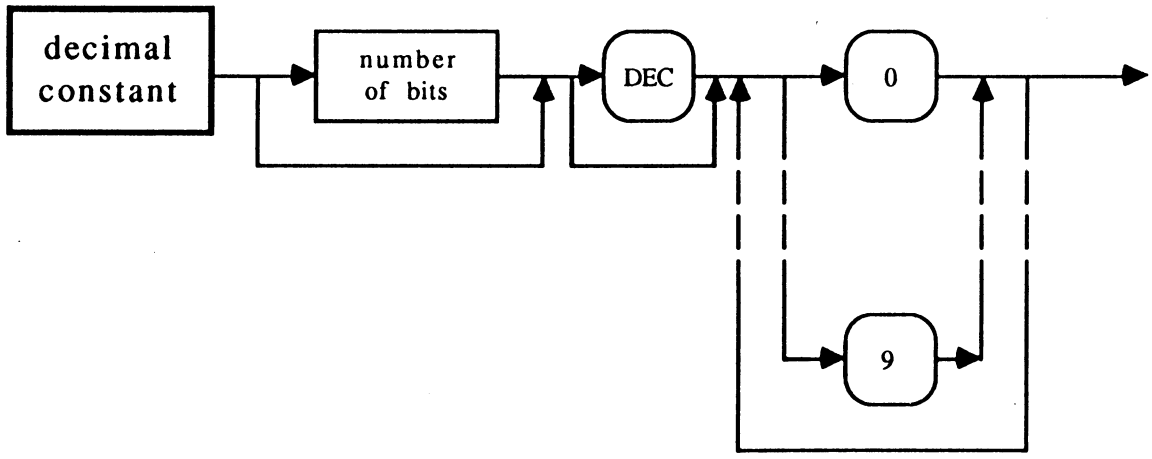

wire
expression
wire scalar
wire vector
=
value-returning
expression

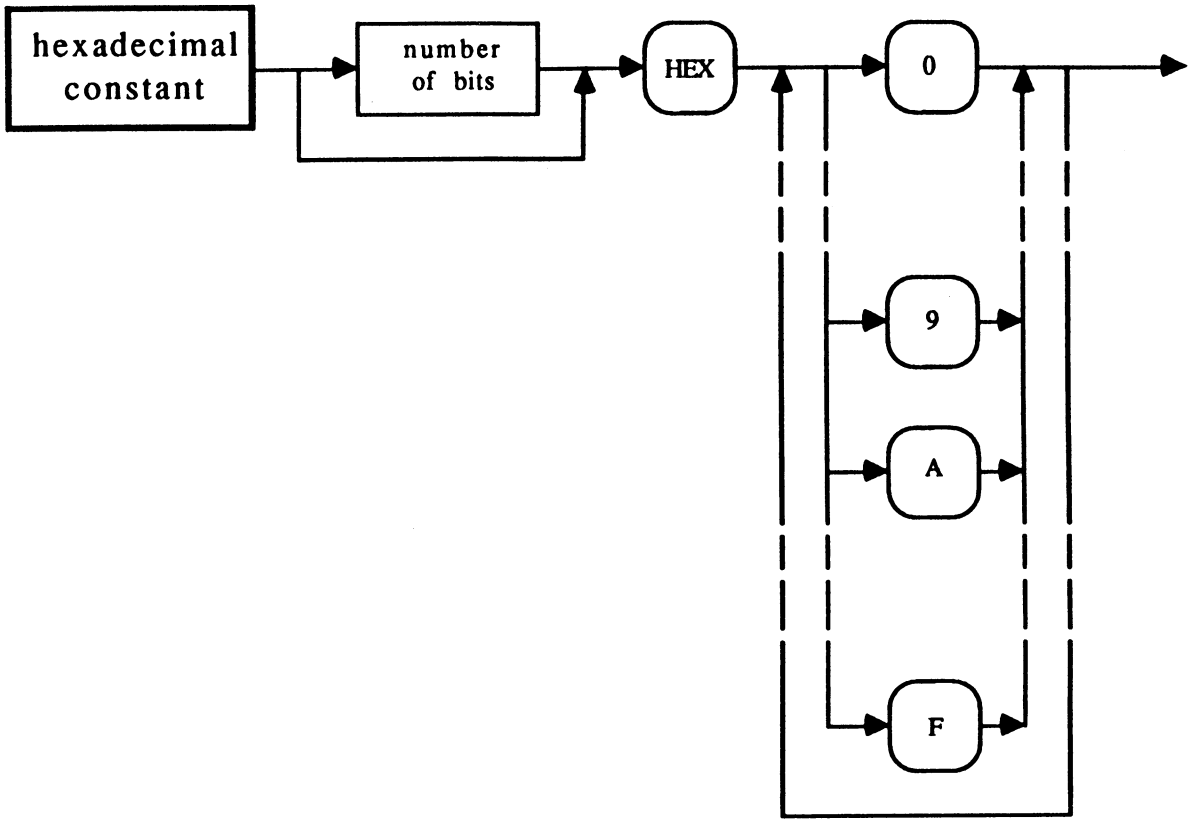

value-returning expression
constant
wire
rom
vector reduction
vector merging
valcase
(
value-returning expression
)
monadic operator
value-returning expression
value-returning expression
dyadic operator
value-returning expression

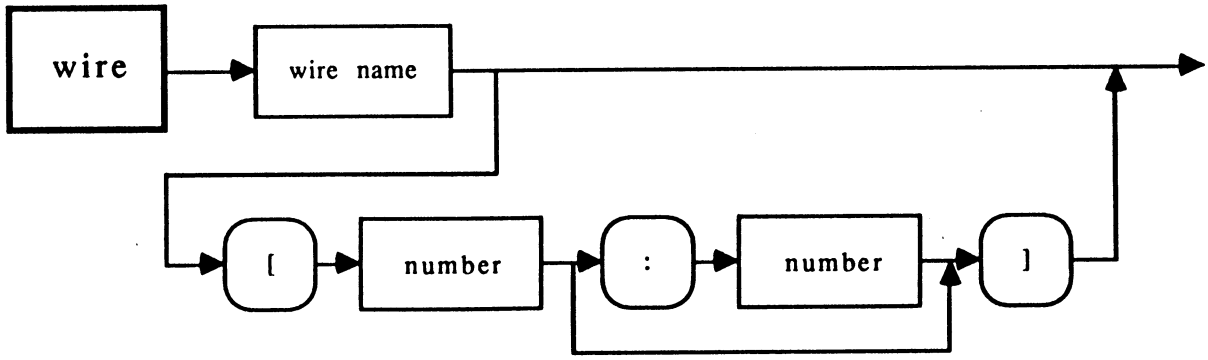

constant
binary constant
octal constant
decimal constant
hexadecimal constant

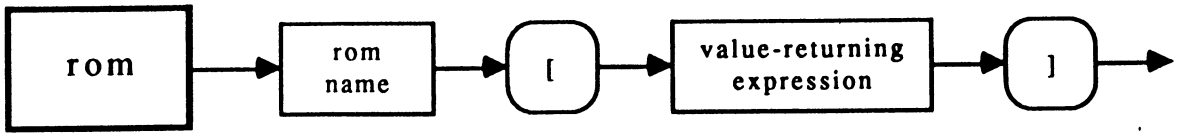

binary constant
number of bits
BIN
0
1

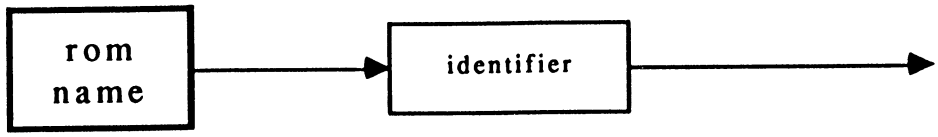

octal constant
number of bits
OCT
0
7

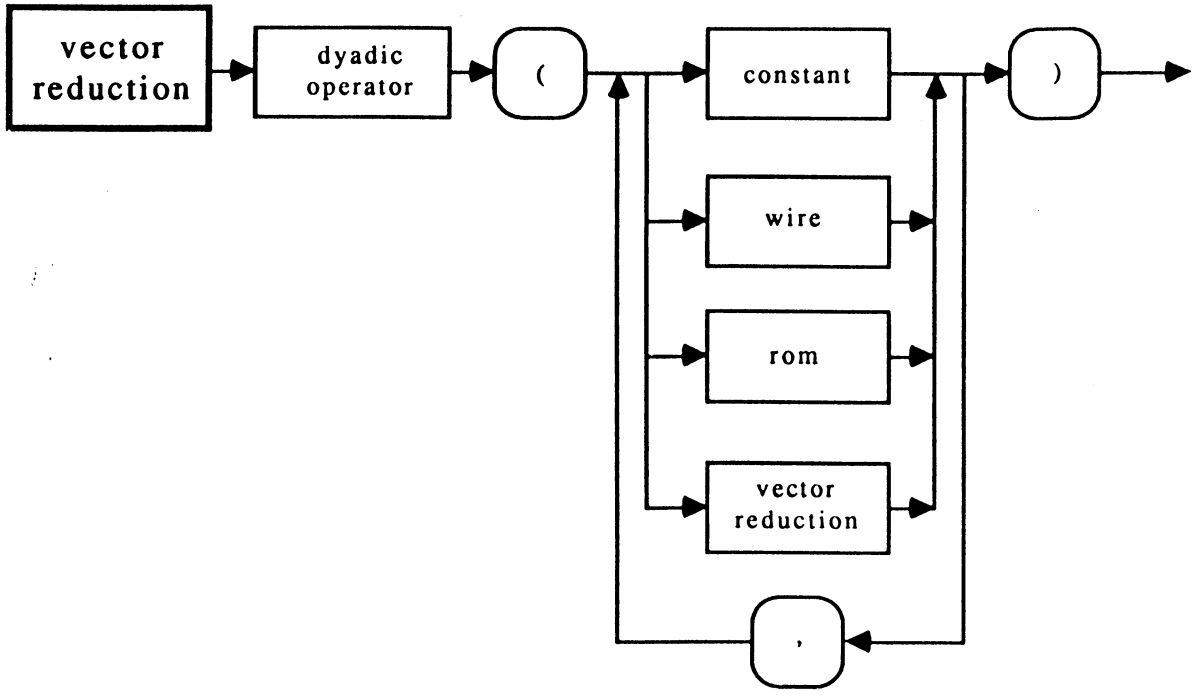

decimal constant
number of bits
DEC
0
9

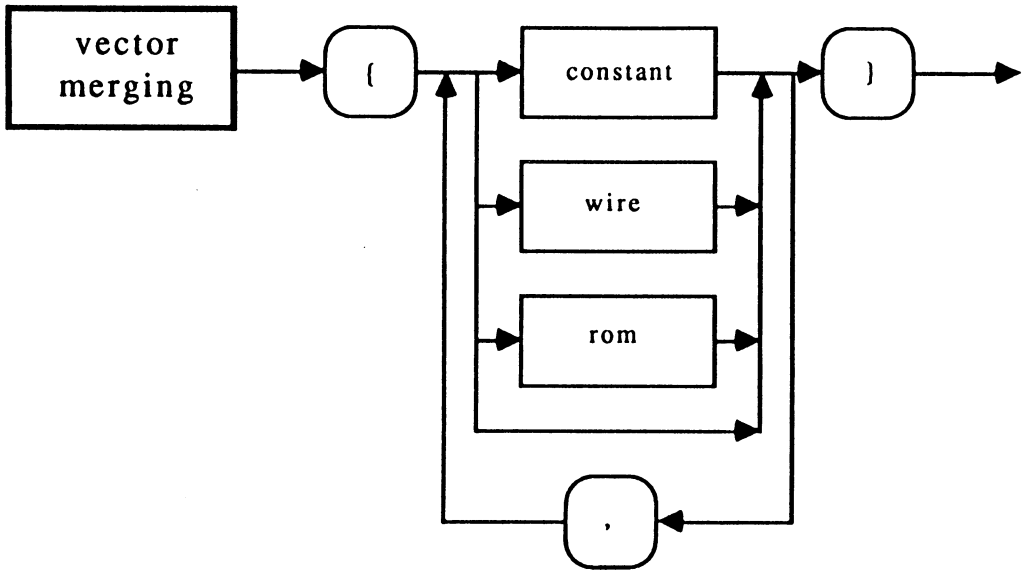

hexadecimal constant
number of bits
HEX
0
9
A
F

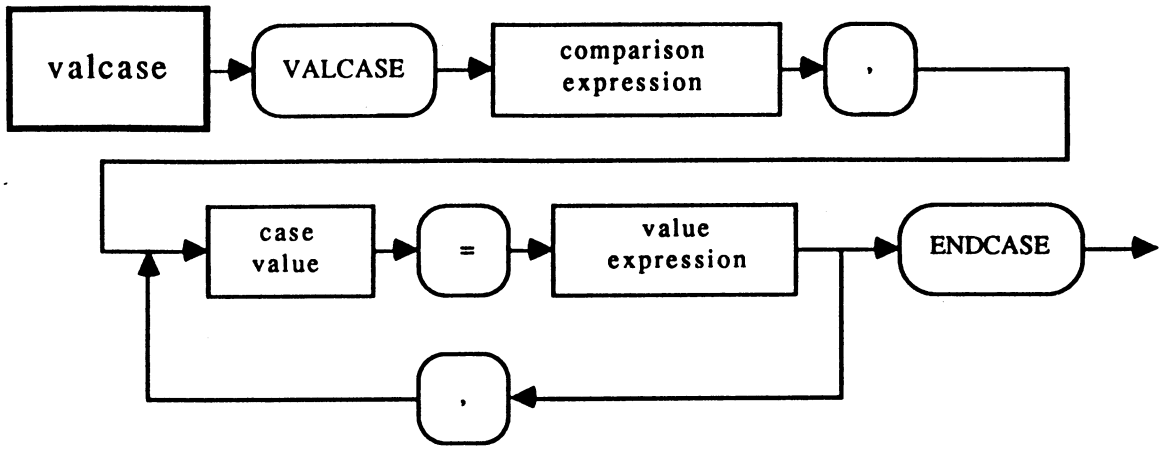

wire
wire name
[
number
:
number
]

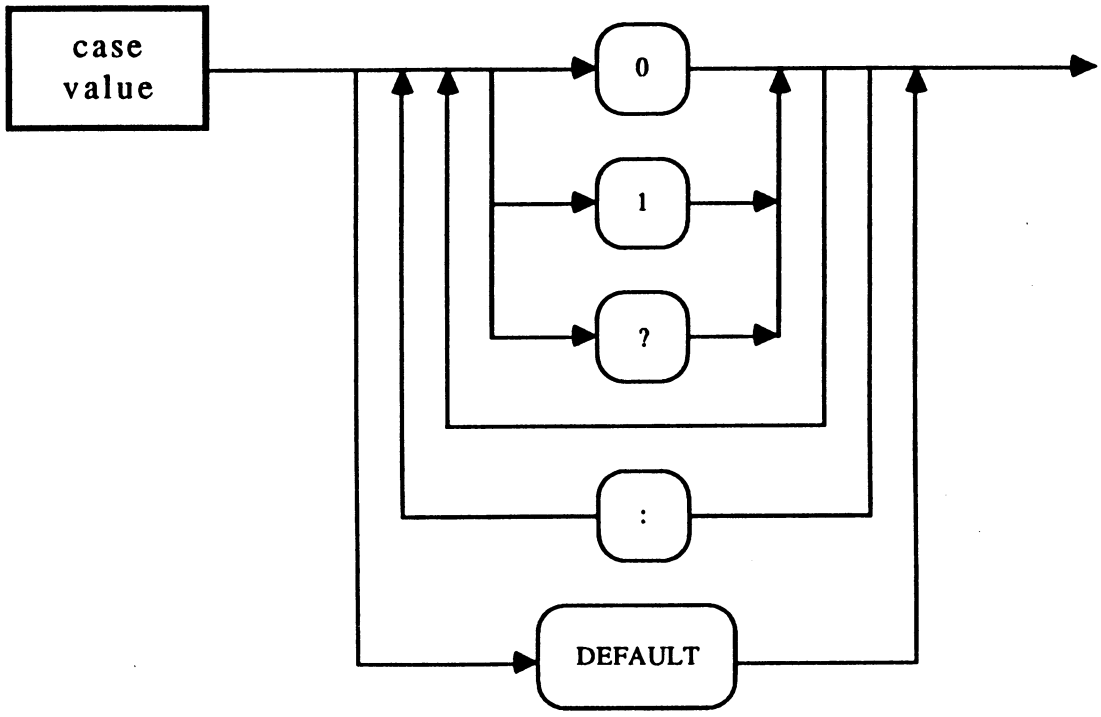

rom
rom name
[
value-returning expression
]

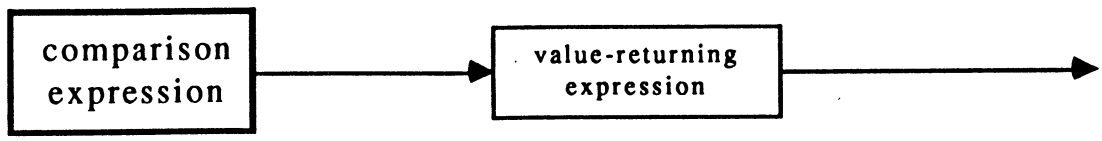

rom name
identifier

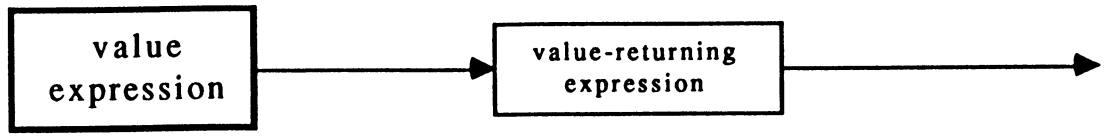

vector reduction
dyadic operator
(
constant
)
wire
rom
vector reduction
,

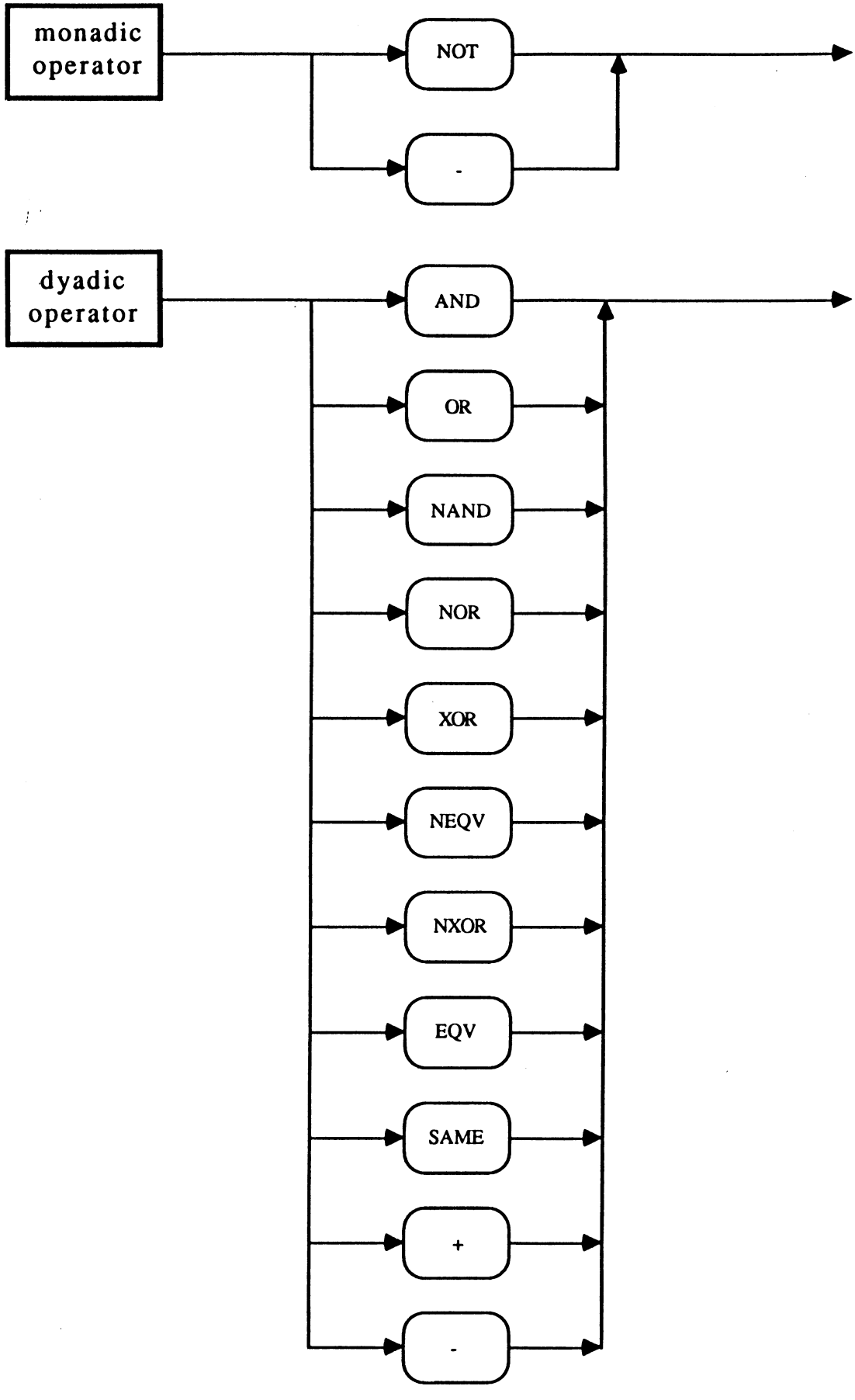

vector merging
{
constant
}
wire
rom
,

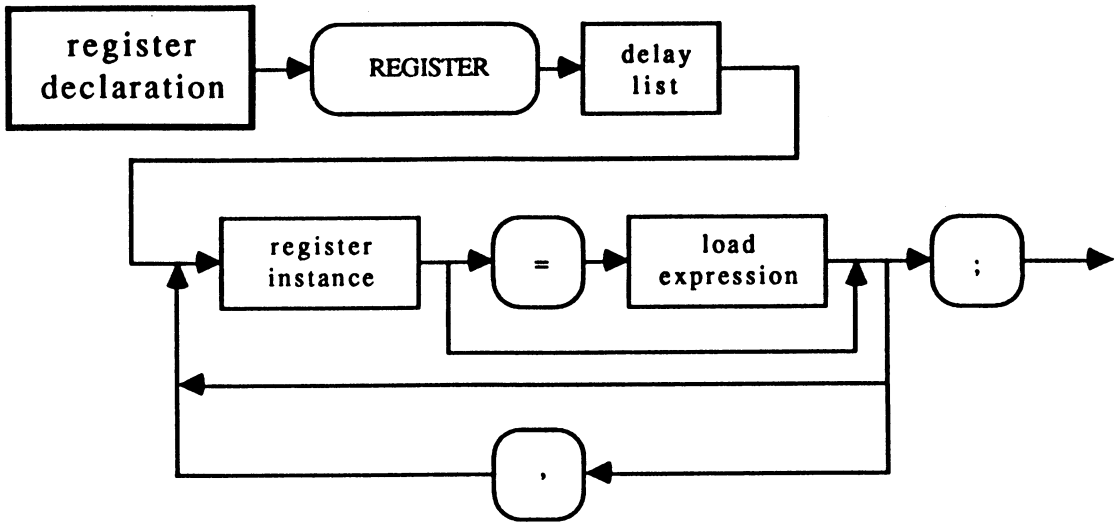

valcase
VALCASE
comparison expression
,
case value
=
value expression
ENDCASE
,

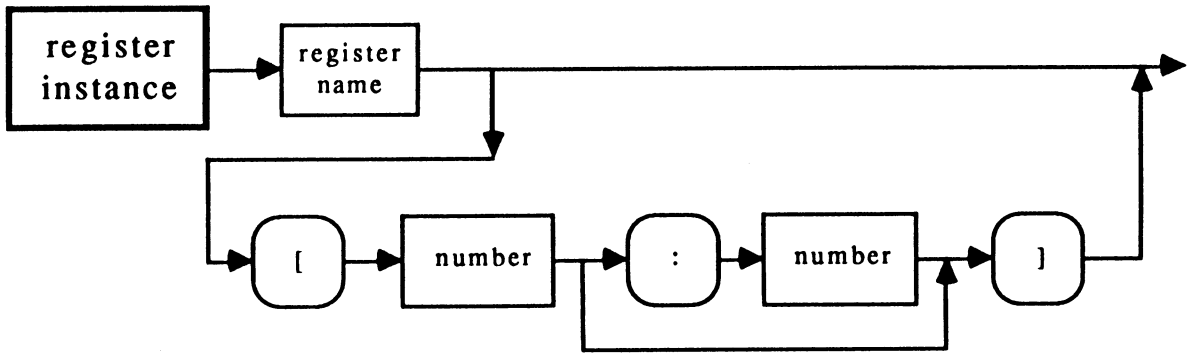

case value
0
1
?
:
DEFAULT

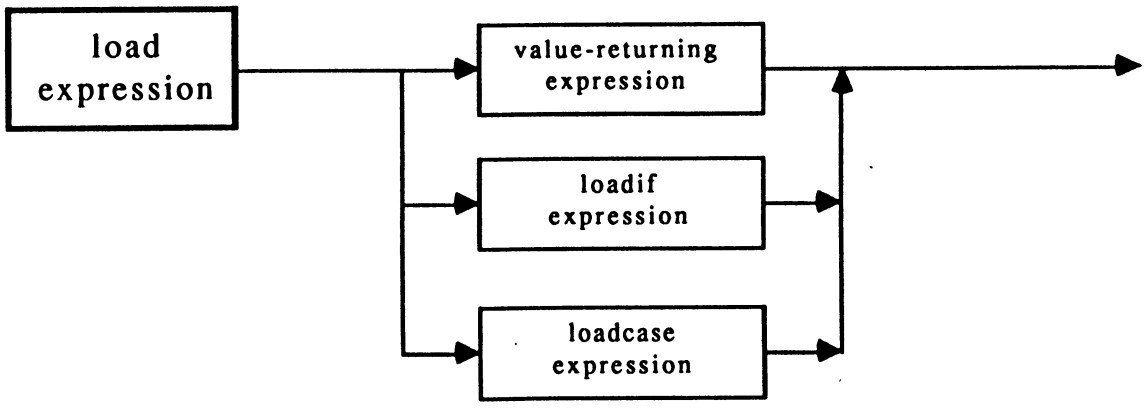

comparison expression
value-returning expression

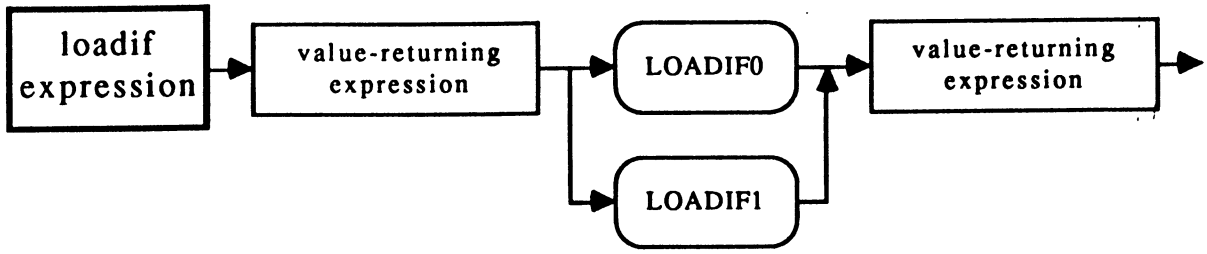

value expression
value-returning expression

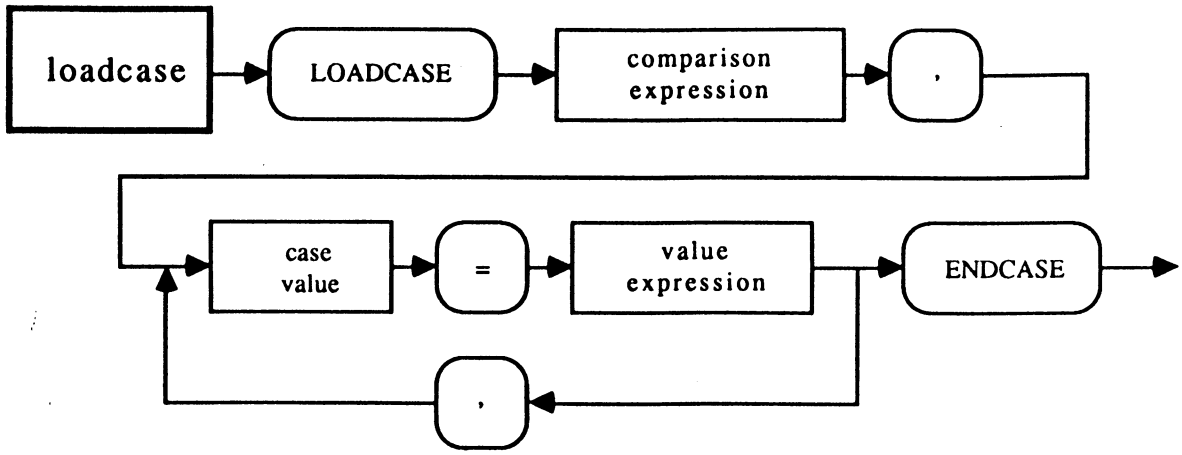

monadic operator
NOT
-
dyadic operator
AND
OR
NAND
NOR
XOR
NEQV
NXOR
EQV
SAME
+
-

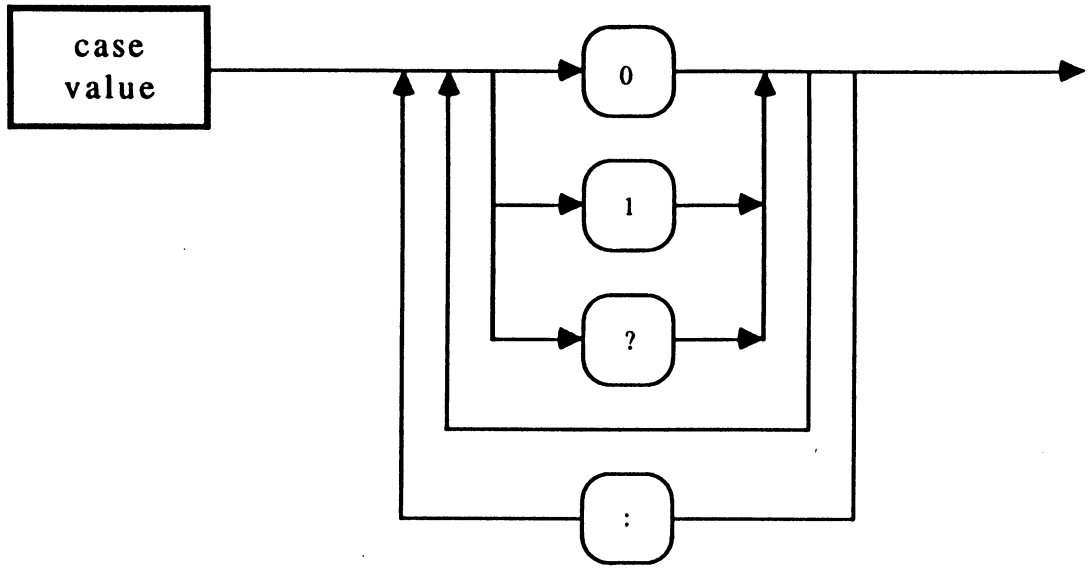

register declaration
REGISTER
delay list
register instance
=
load expression
;
,

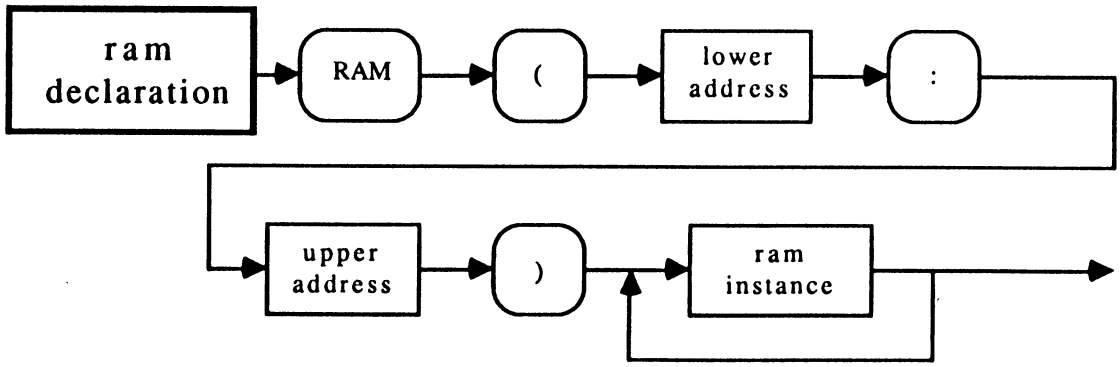

register instance
register name
[
number
:
number
]

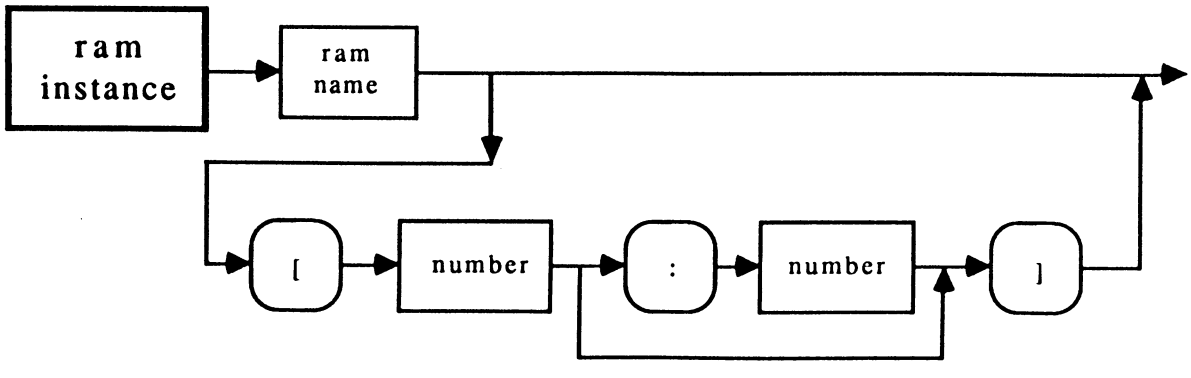

load expression
value-returning expression
loadif expression
loadcase expression

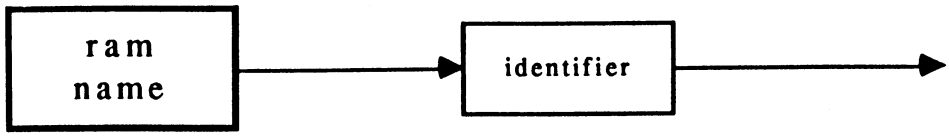

loadif expression
value-returning expression
LOADIF0
LOADIF1
value-returning expression

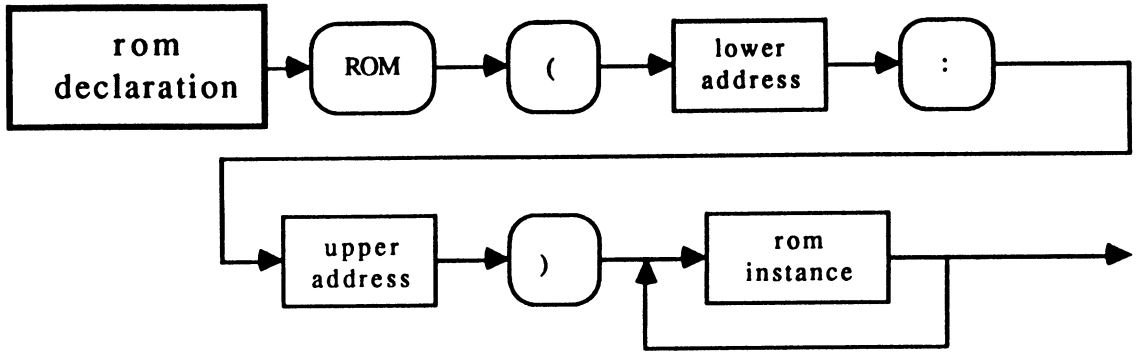

loadcase
LOADCASE
comparison expression
,
case value
=
value expression
ENDCASE
,

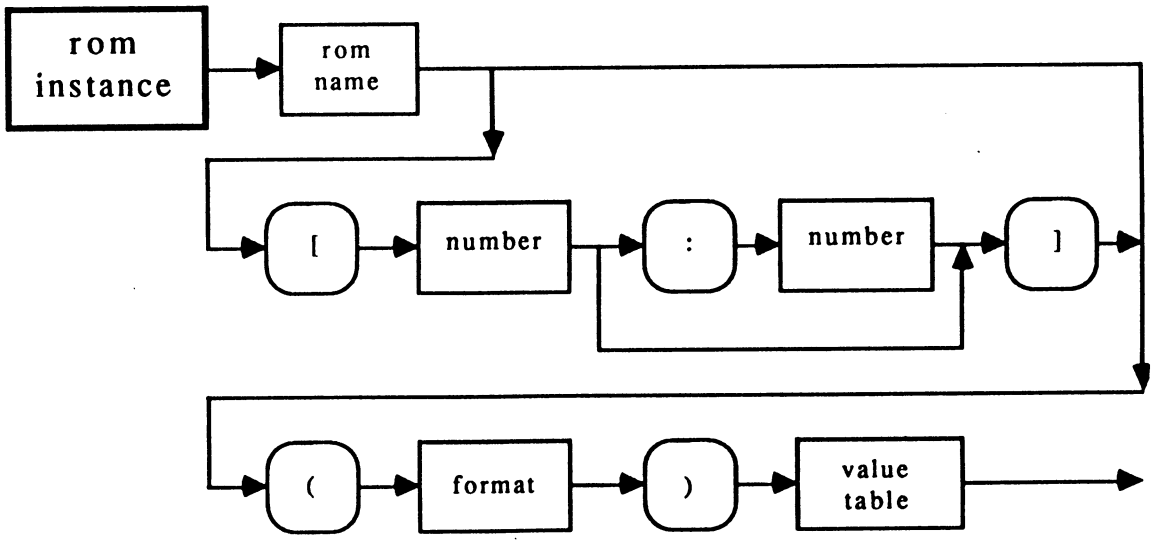

case value
0
1
?
:

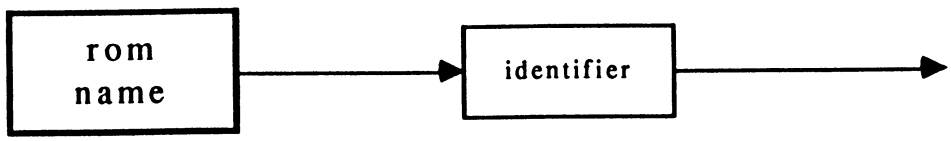

ram declaration
RAM
(
lower address
:
upper address
)
ram instance

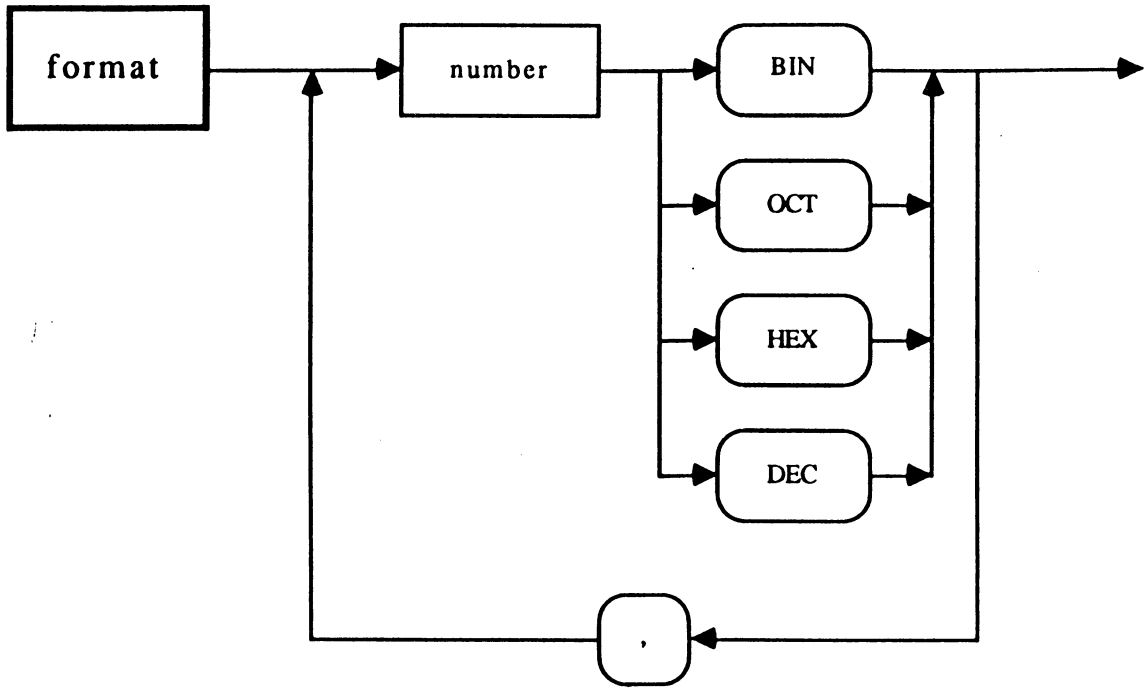

ram instance
ram name
[
number
:
number
]

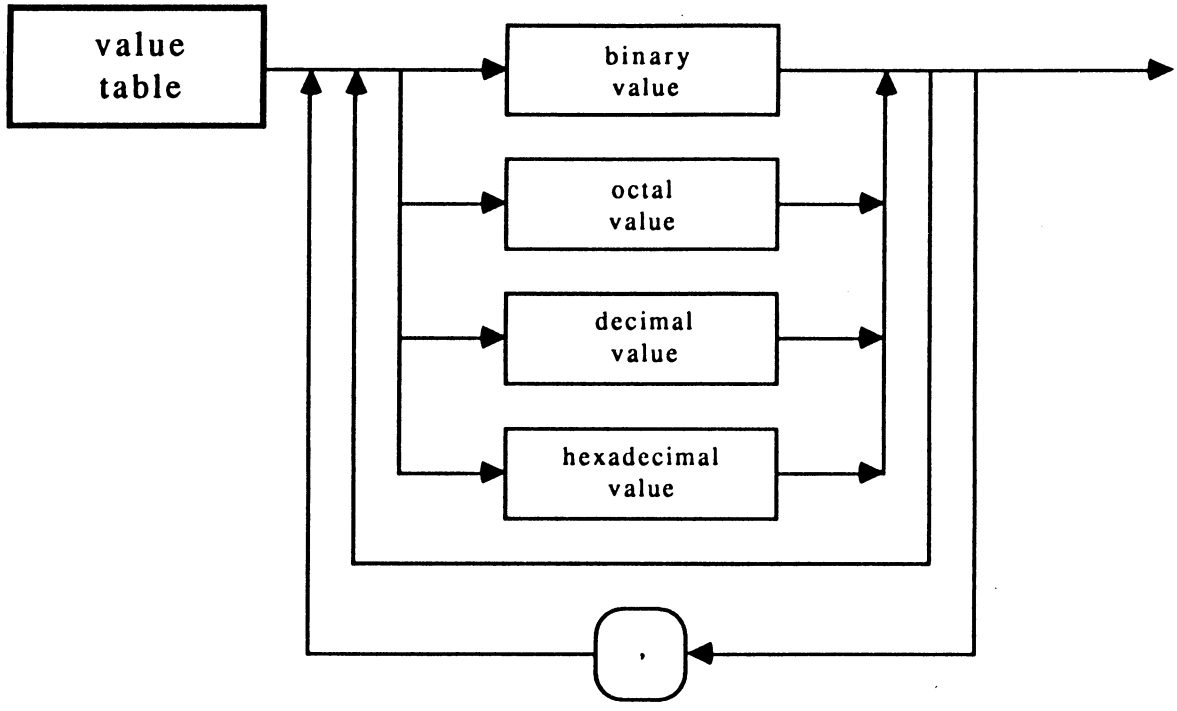

ram
name
identifier

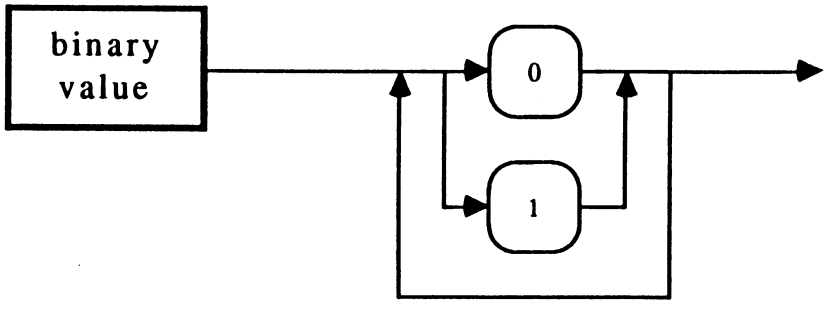

rom
declaration
ROM
(
lower
address
:
upper
address
)
rom
instance

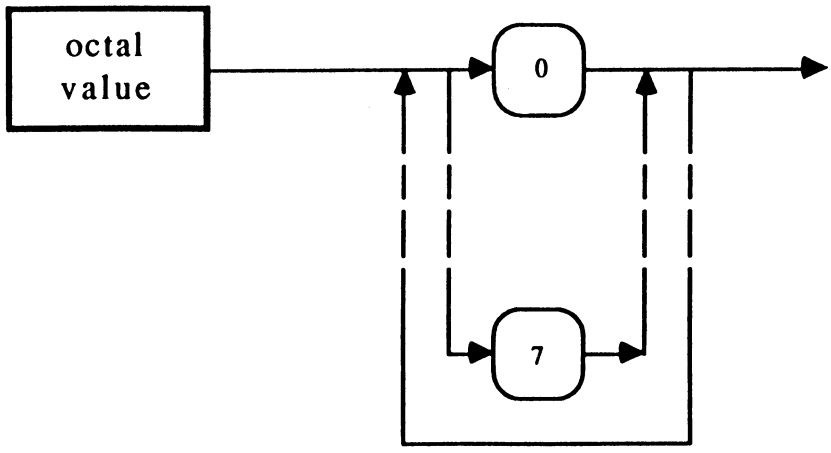

rom
instance
rom
name
[
number
:
number
]
(
format
)
value
table

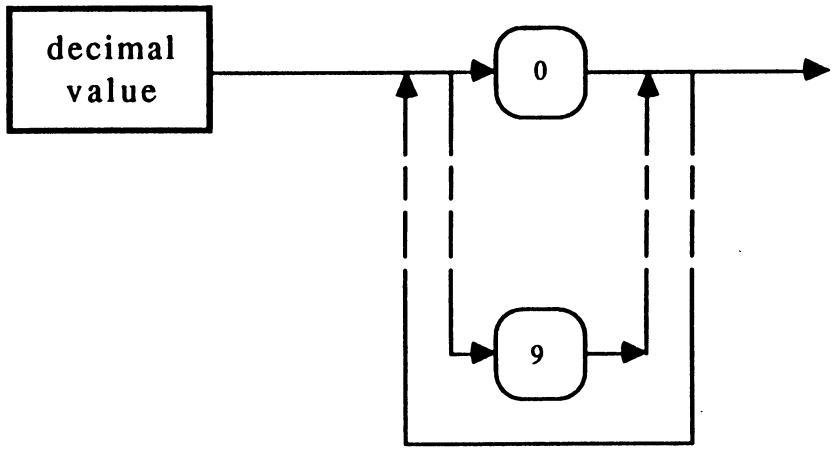

rom
name
identifier

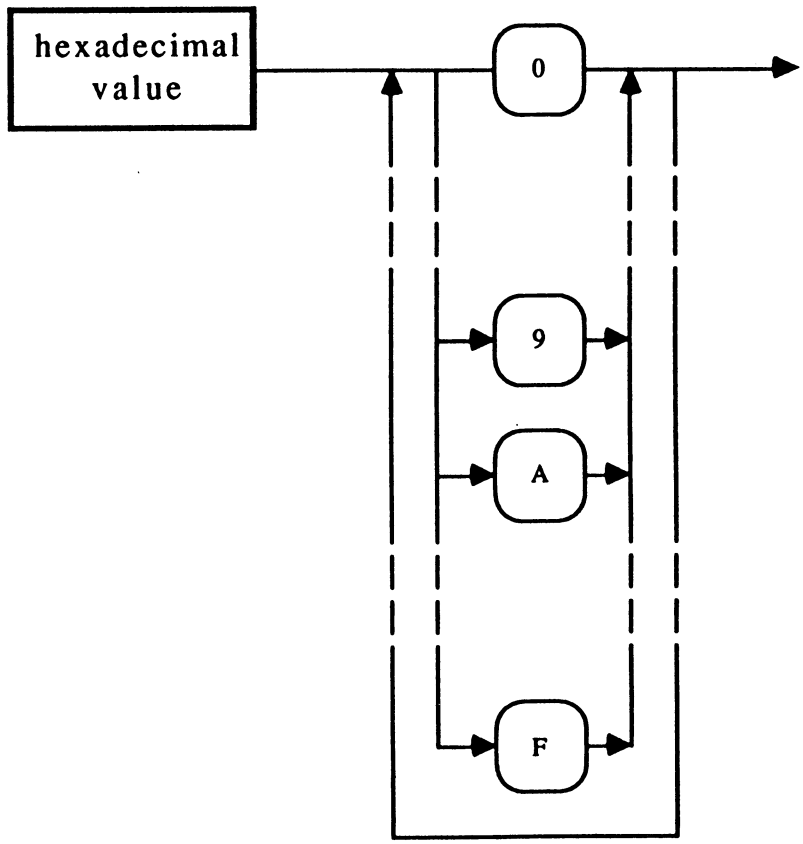

format
number
BIN
OCT
HEX
DEC
,

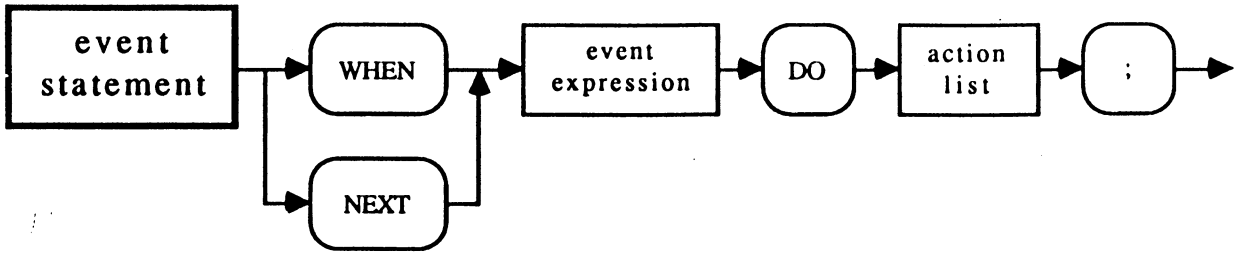

value
table
binary
value
octal
value
decimal
value
hexadecimal
value
,

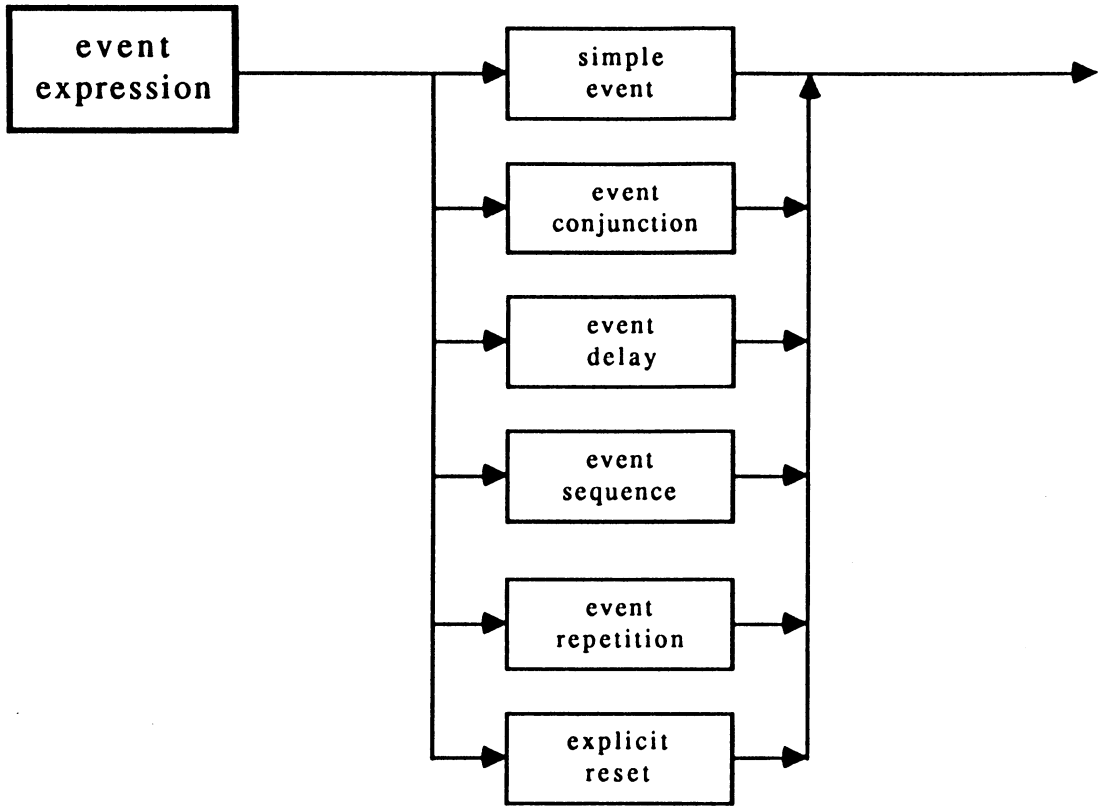

binary
value
0
1

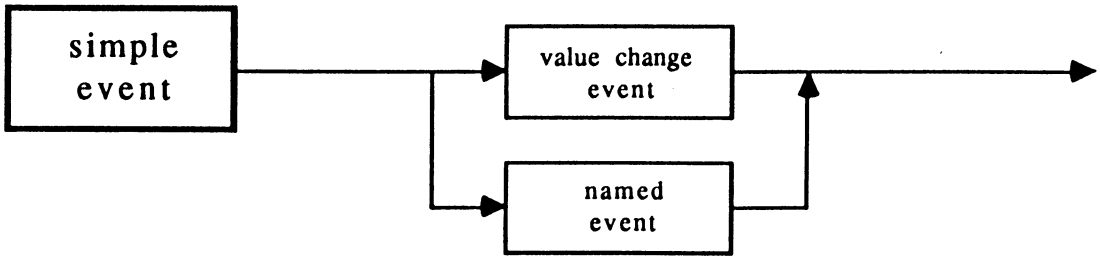

octal
value
0
7

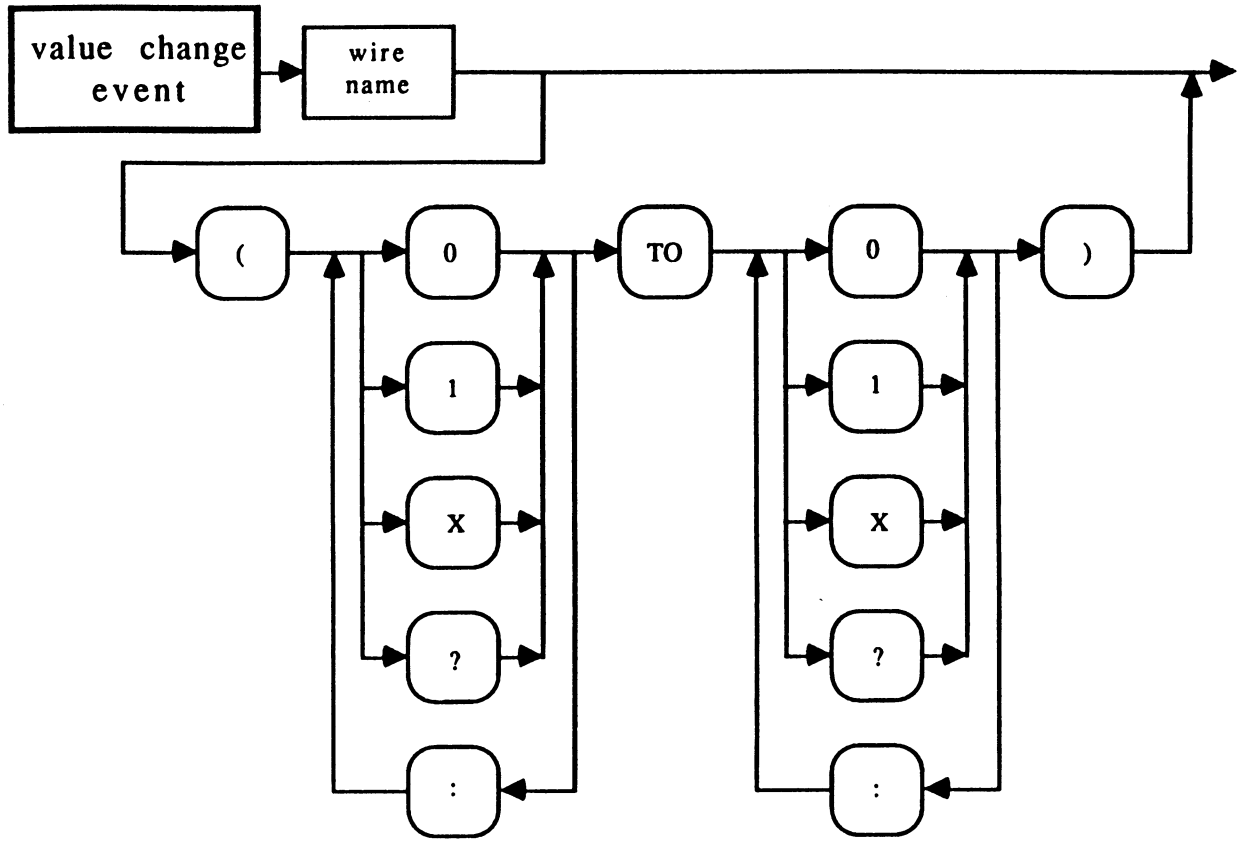

decimal
value
0
9

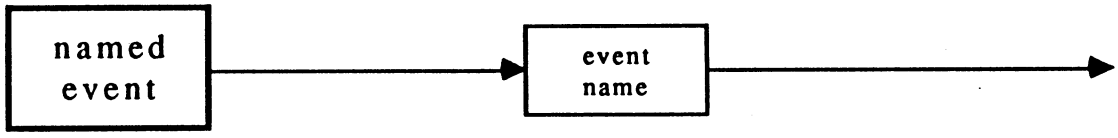

hexadecimal
value
0
9
A
F

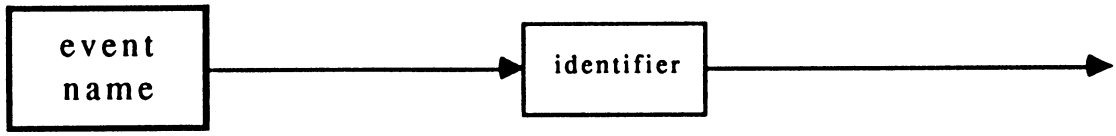

event statement
WHEN
NEXT
event expression
DO
action list
;

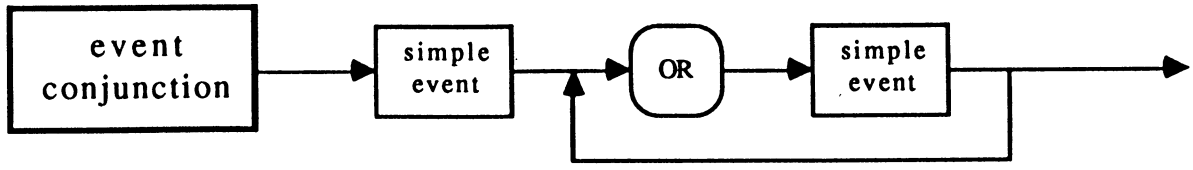

event expression
simple event
event conjunction
event delay
event sequence
event repetition
explicit reset

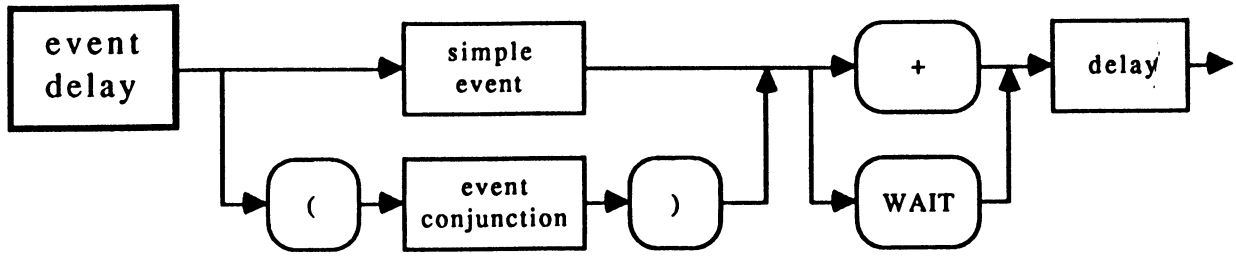

simple event
value change event
named event

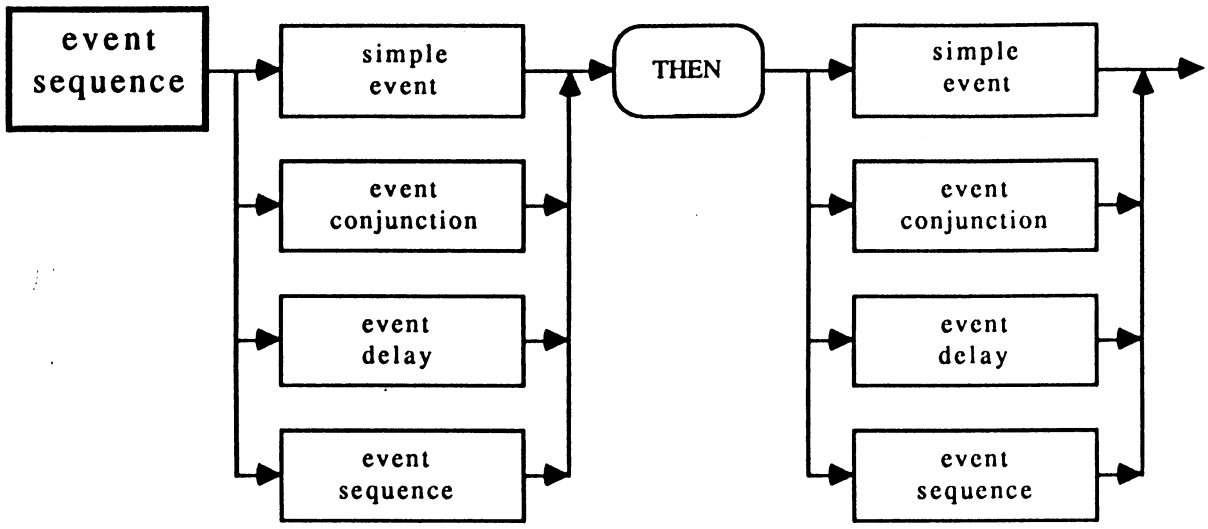

value change event
wire name
(
0
1
X
?
:
TO
0
1
X
?
:
)

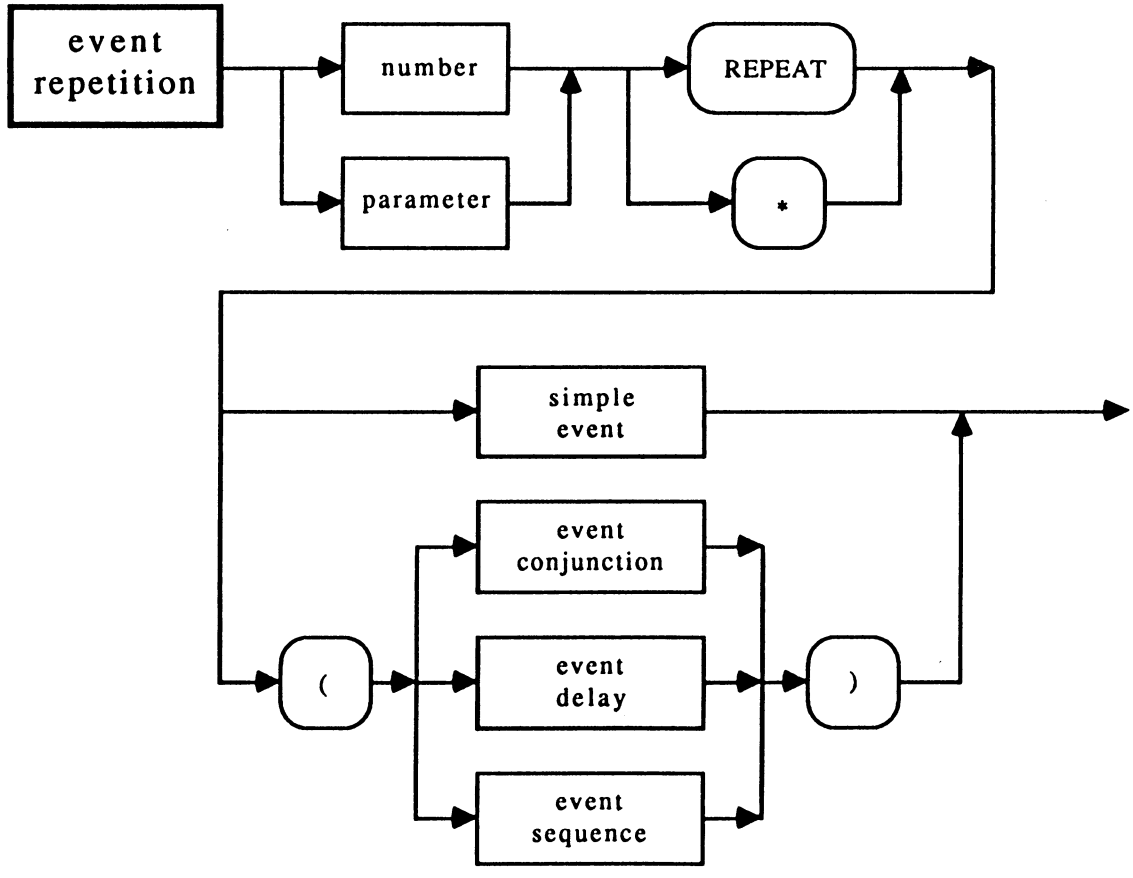

named event
event name

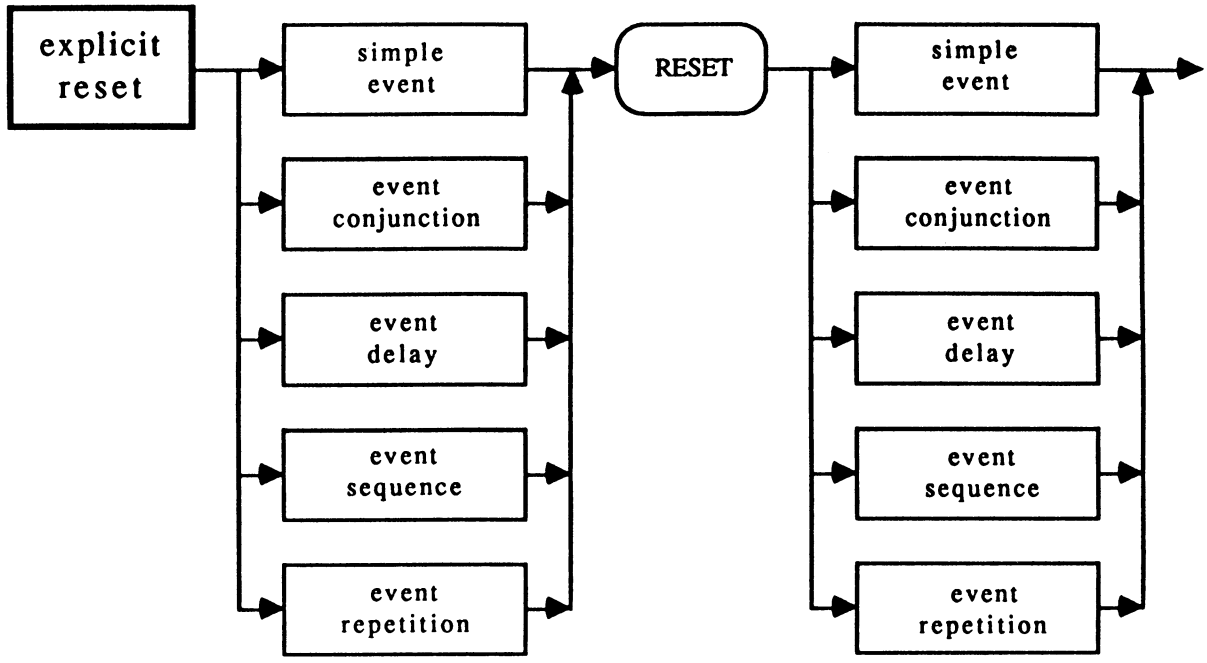

event name
identifier

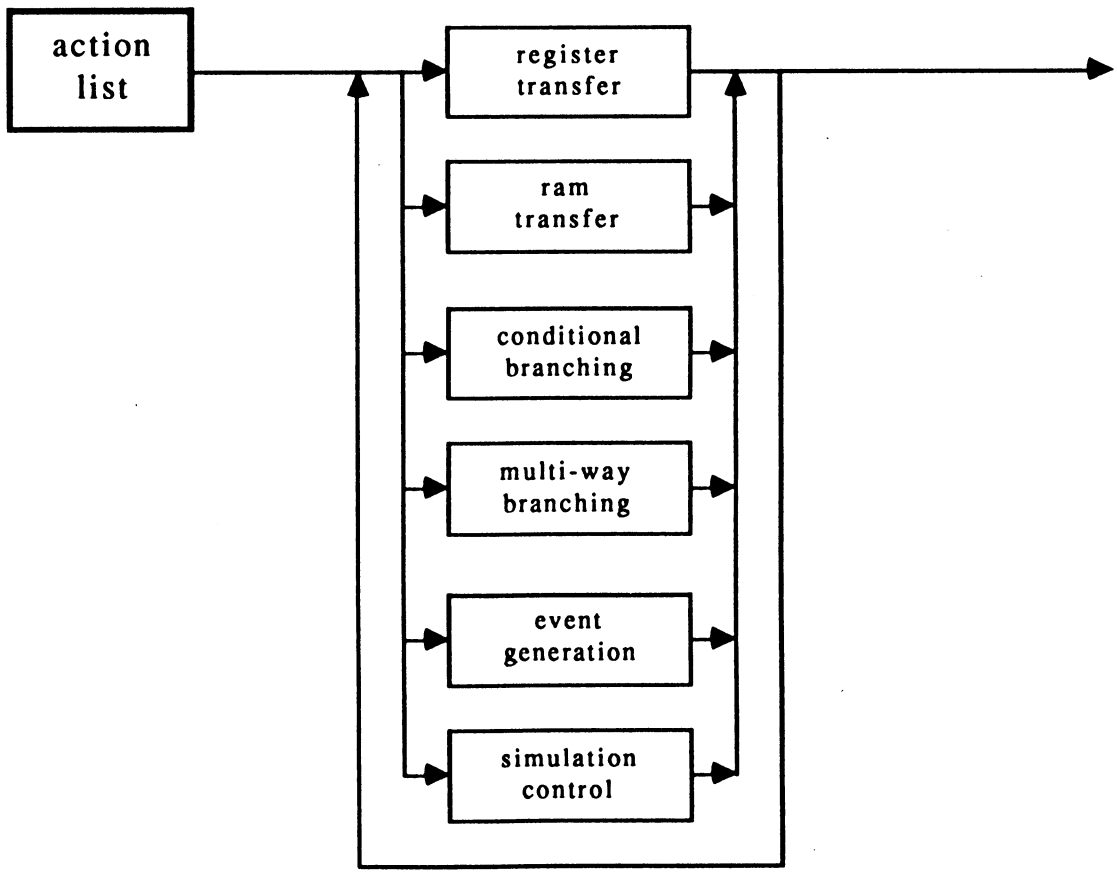

event conjunction
simple event
OR
simple event

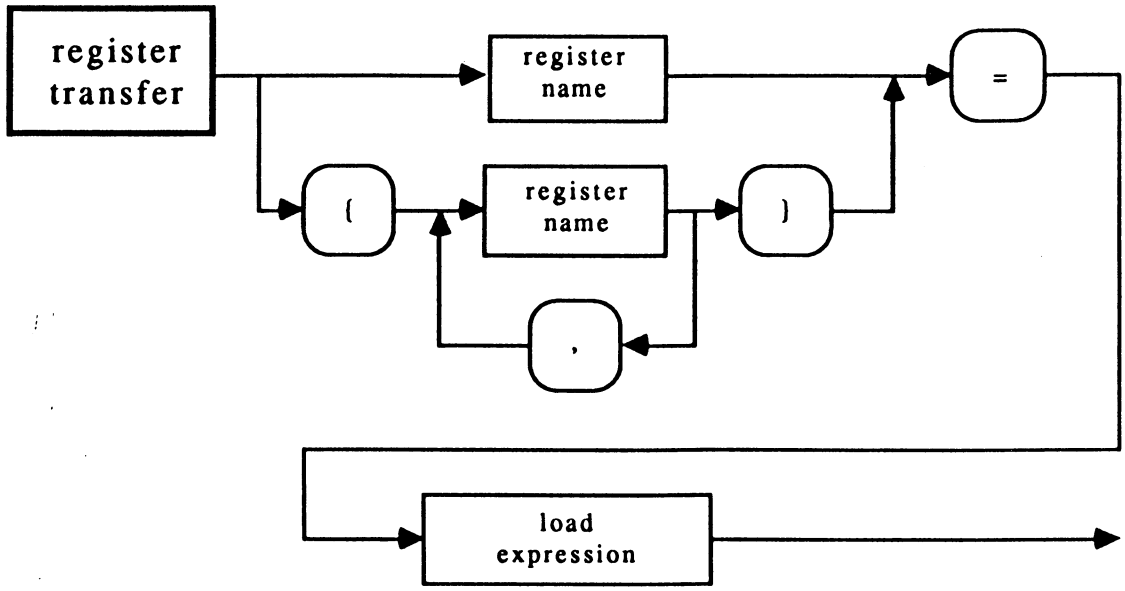

event delay
simple event
(
event conjunction
)
+
WAIT
delay

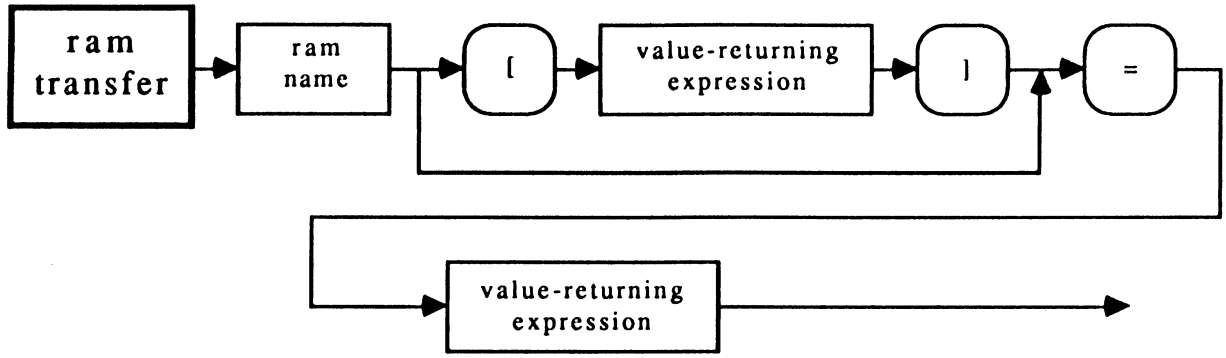

event sequence
simple event
event conjunction
event delay
event sequence
THEN
simple event
event conjunction
event delay
event sequence

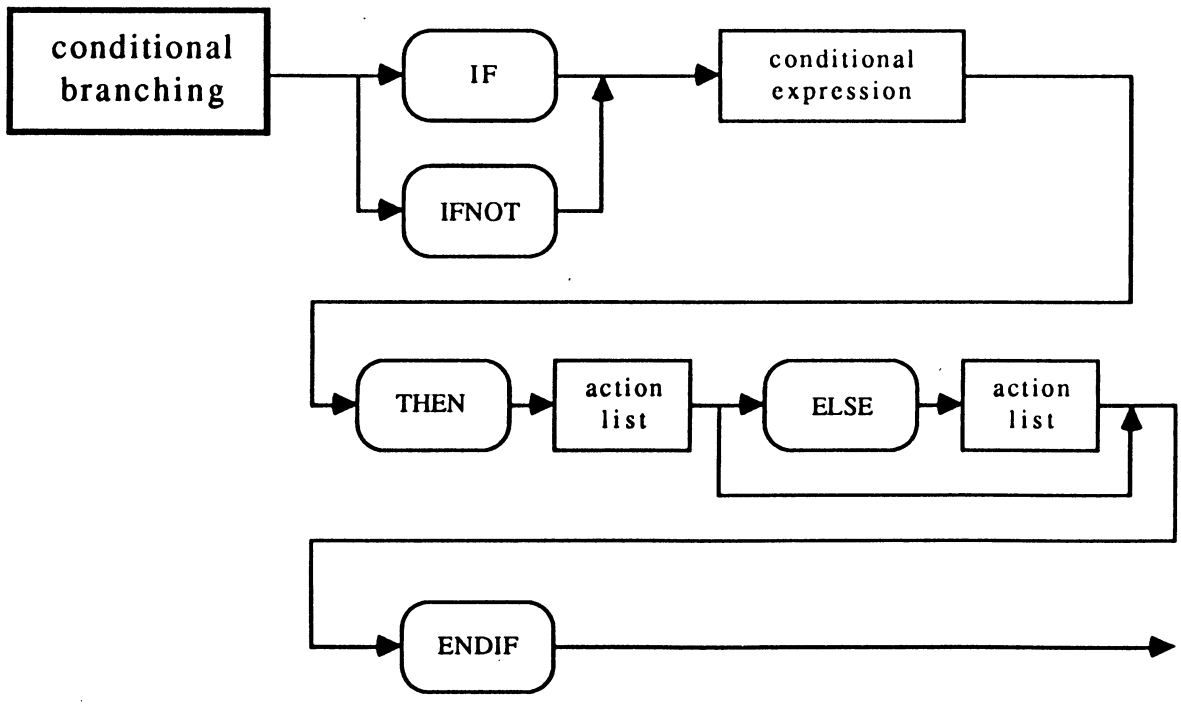

event repetition
number
parameter
REPEAT
*
simple event
event conjunction
(
event delay
)
event sequence

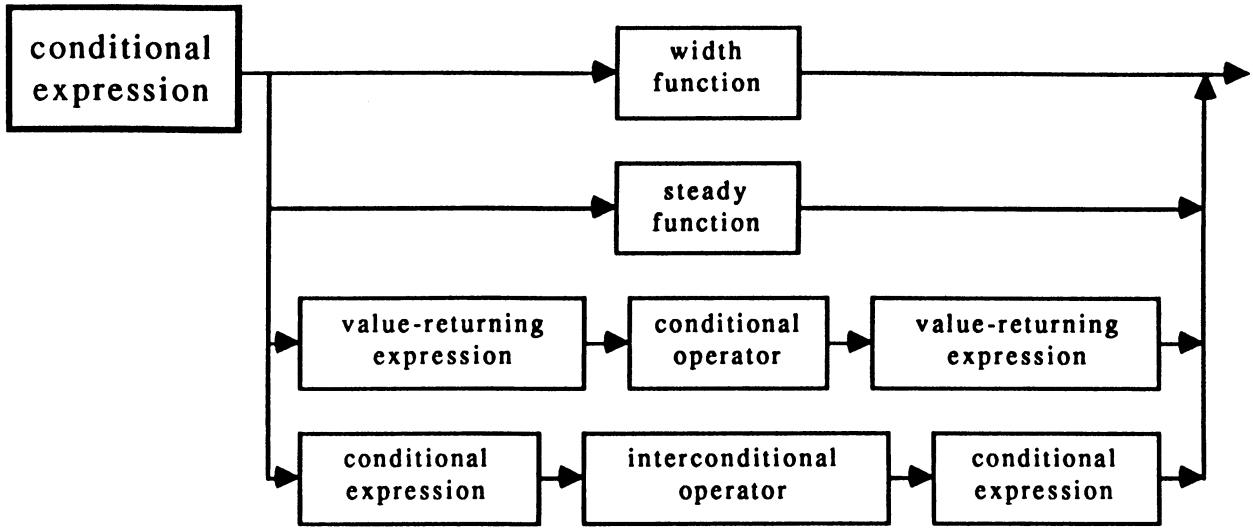

explicit reset
simple event
event conjunction
event delay
event sequence
event repetition
RESET
simple event
event conjunction
event delay
event sequence
event repetition

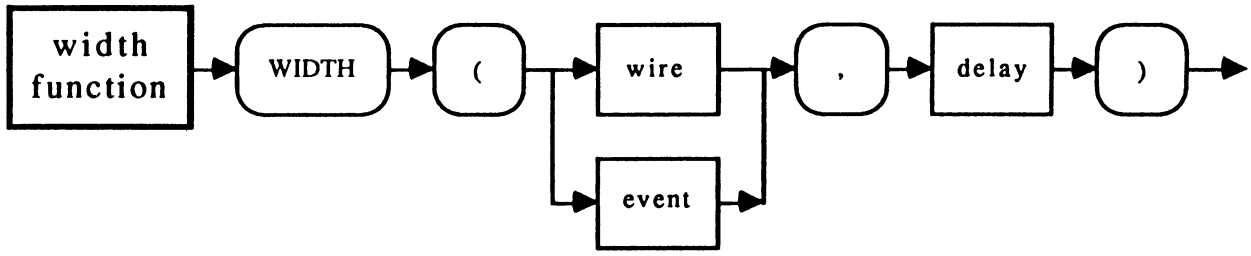

action list
register transfer
ram transfer
conditional branching
multi-way branching
event generation
simulation control

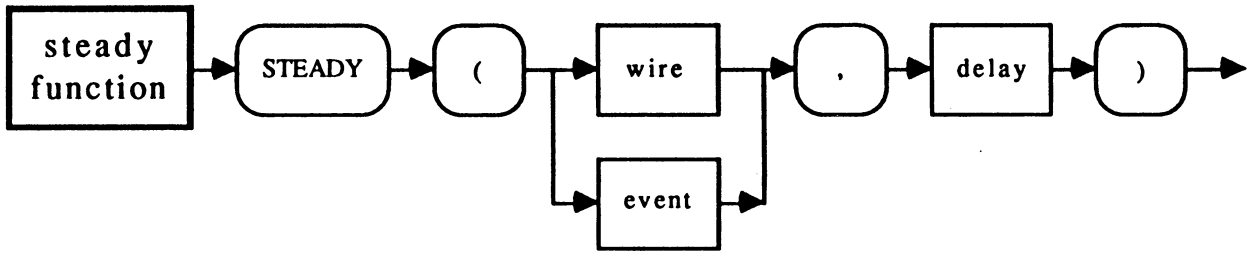

register transfer
register name
=
(
register name
)
,
load expression

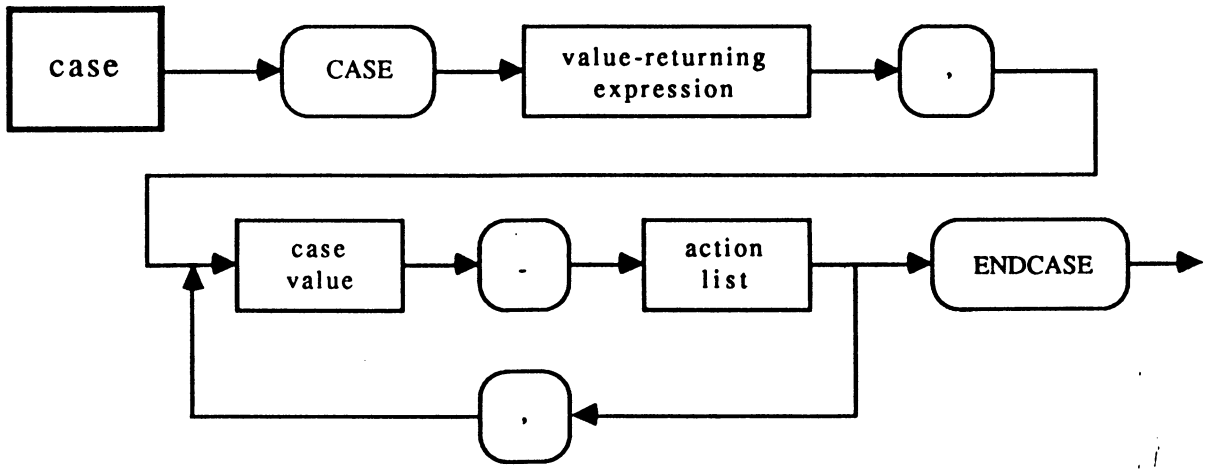

ram transfer
ram name
[
value-returning expression
]
=
value-returning expression

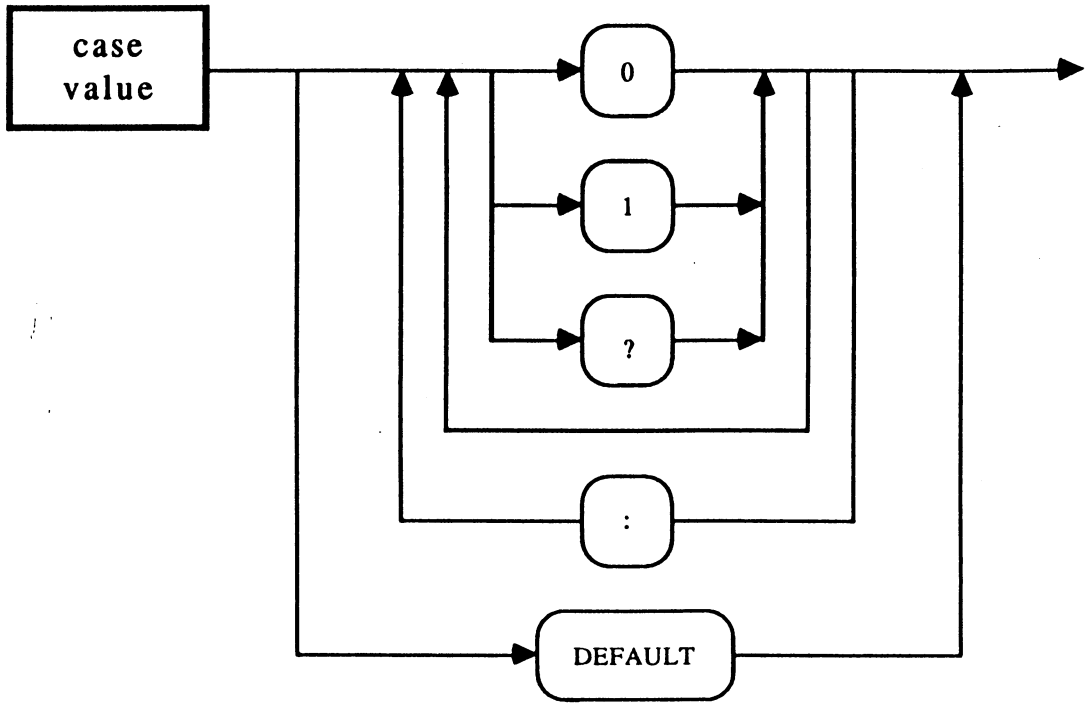

conditional branching
IF
IFNOT
conditional expression
THEN
action list
ELSE
action list
ENDIF

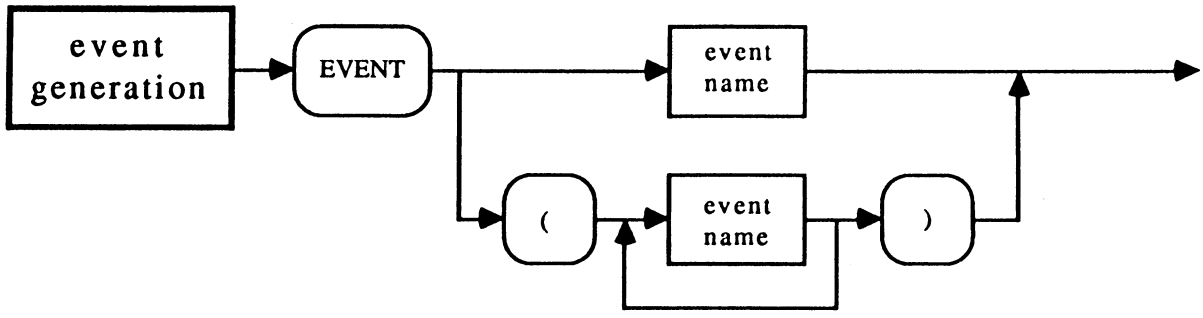

conditional expression
width function
steady function
value-returning expression
conditional operator
value-returning expression
conditional expression
interconditional operator
conditional expression

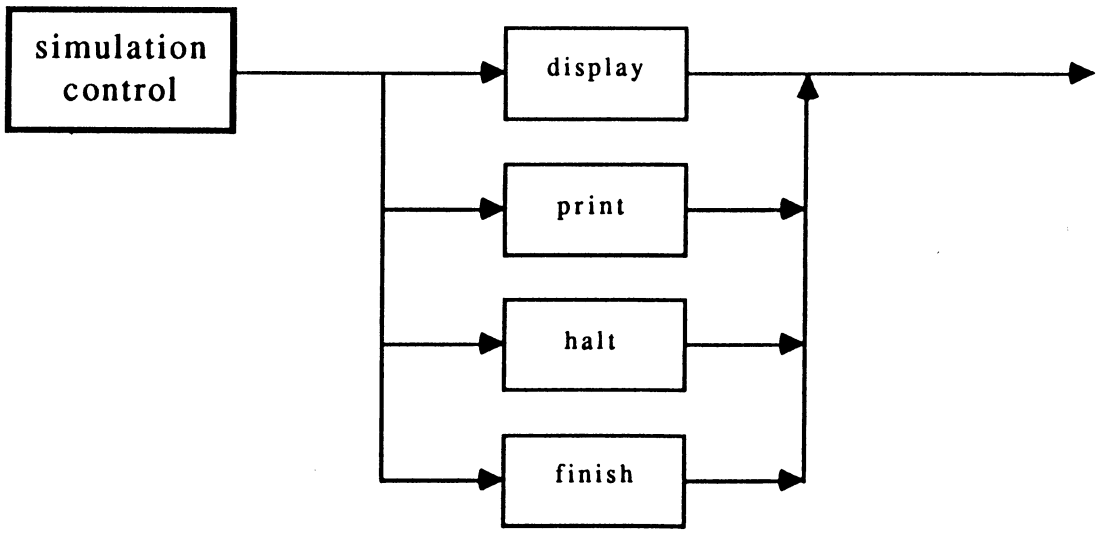

width function
WIDTH
(
wire
event
,
delay
)

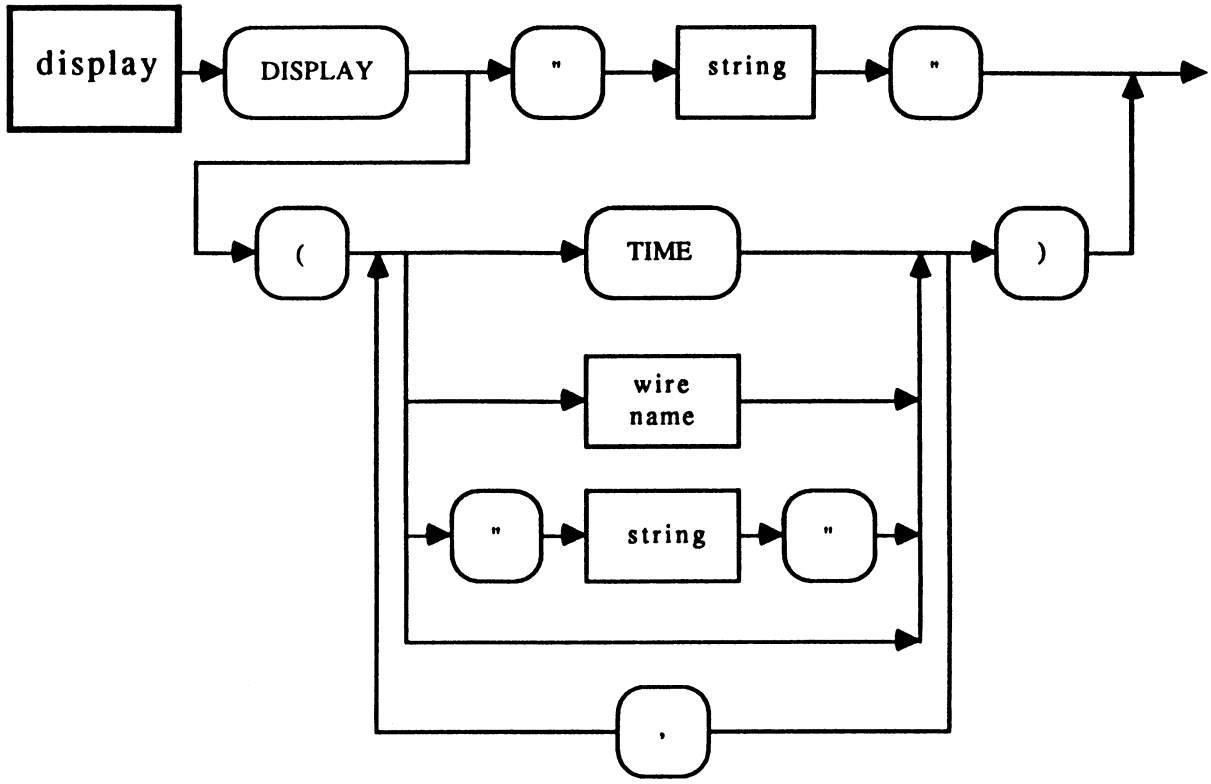

steady function
STEADY
(
wire
event
,
delay
)

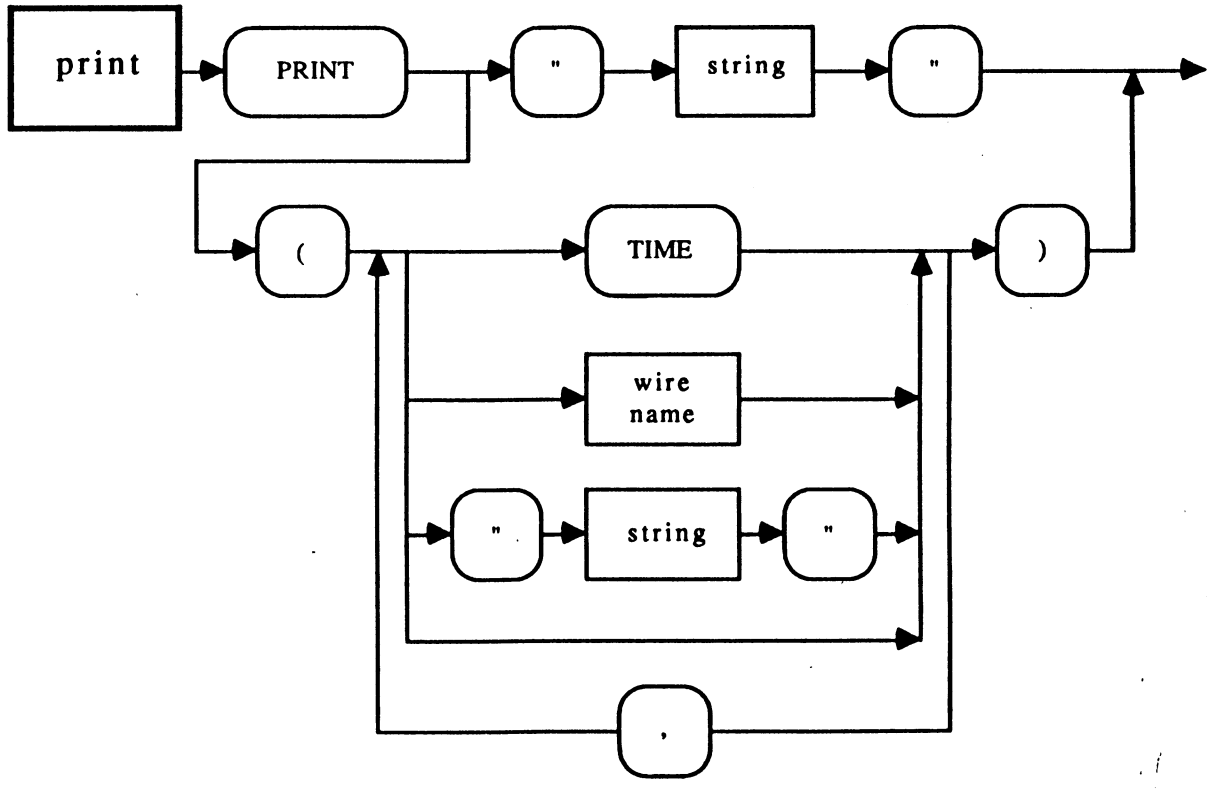

case
CASE
value-returning expression
,
case value
-
action list
ENDCASE
,

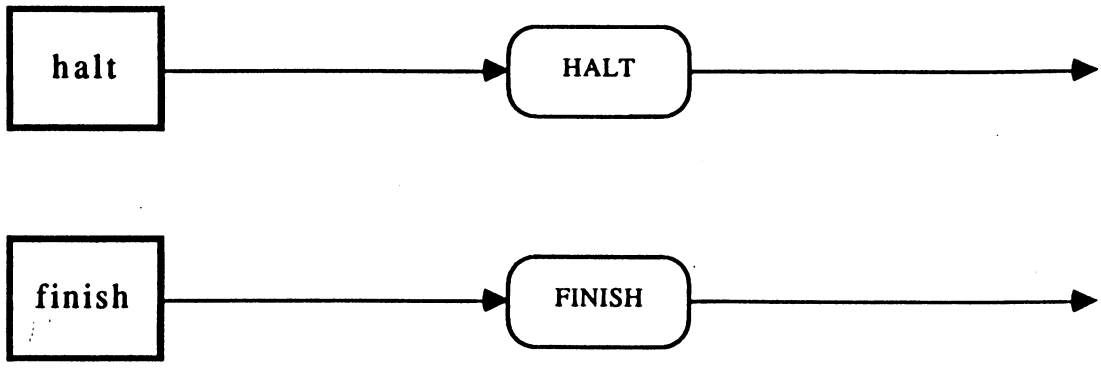

case value
0
1
?
:
DEFAULT


event generation
EVENT
event name
(
event name
)

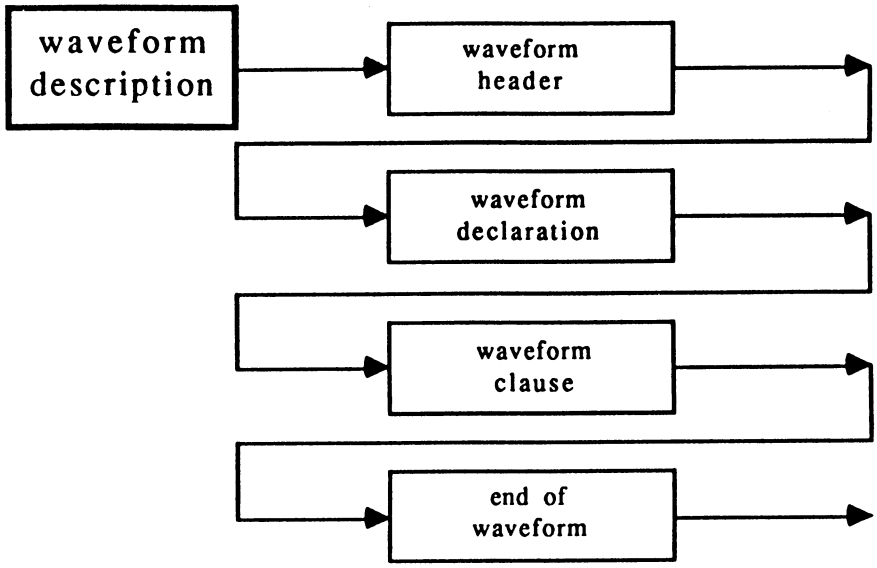

simulation control
display
print
halt
finish

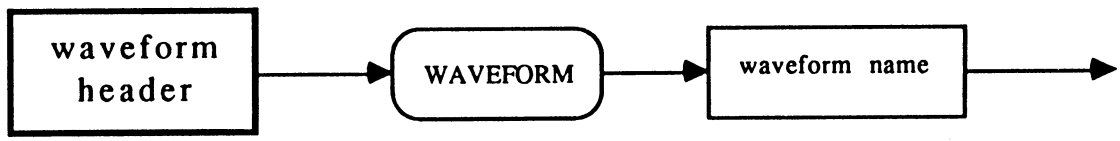

display
DISPLAY
"
string
"
(
TIME
)
wire
name
"
string
"
,

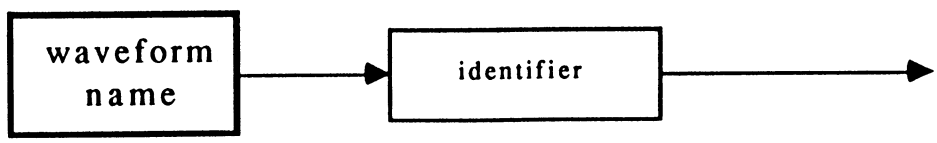

print
PRINT
"
string
"
(
TIME
)
wire
name
"
string
"
,

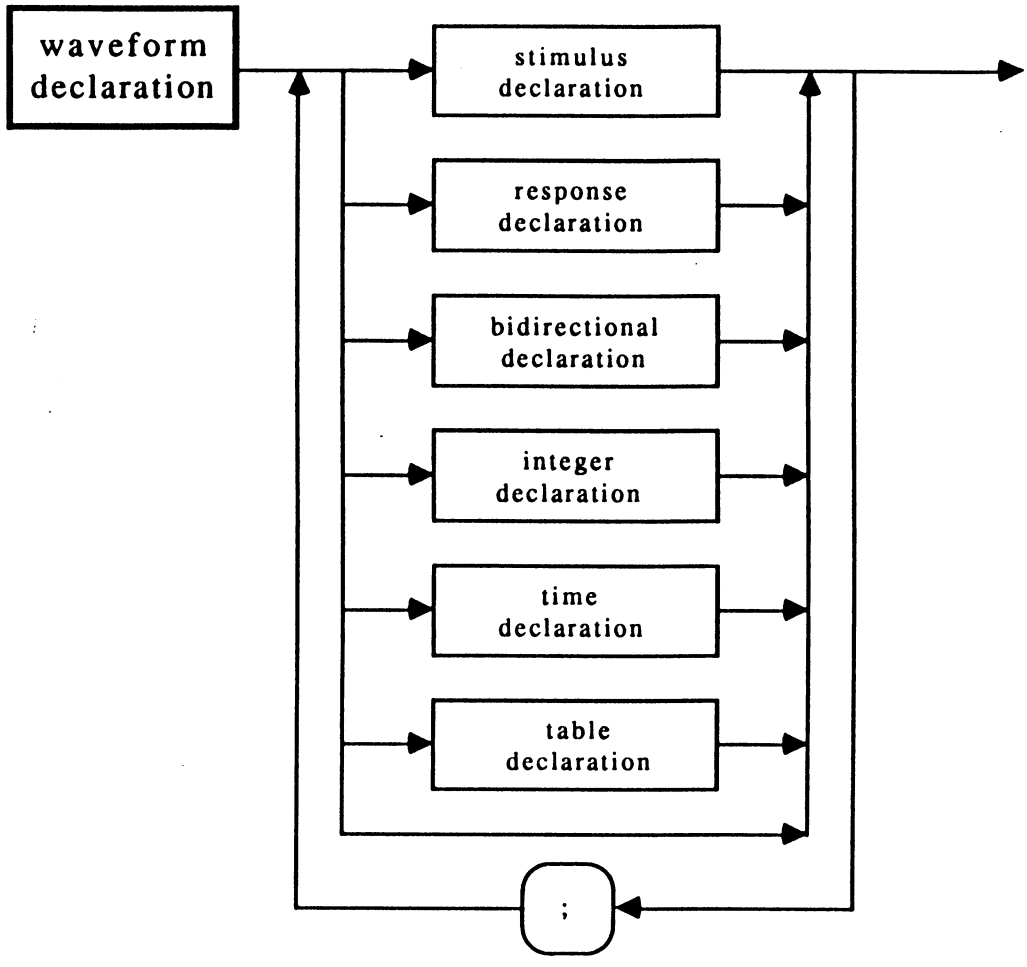

halt
HALT
finish
FINISH

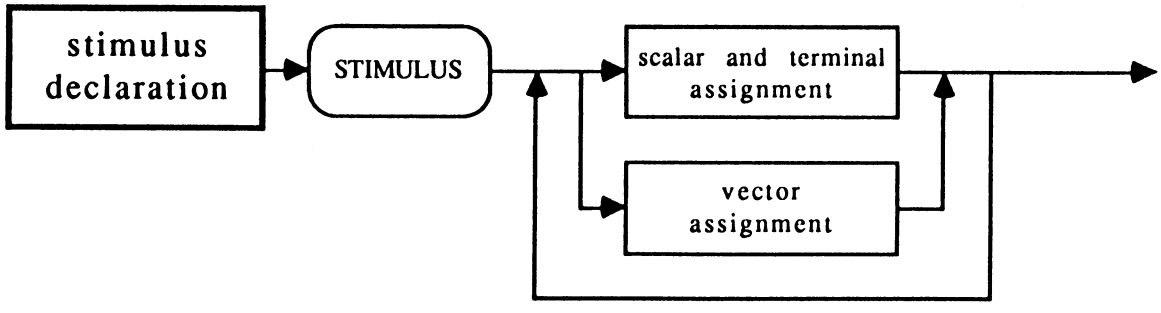
B.2 Waveform

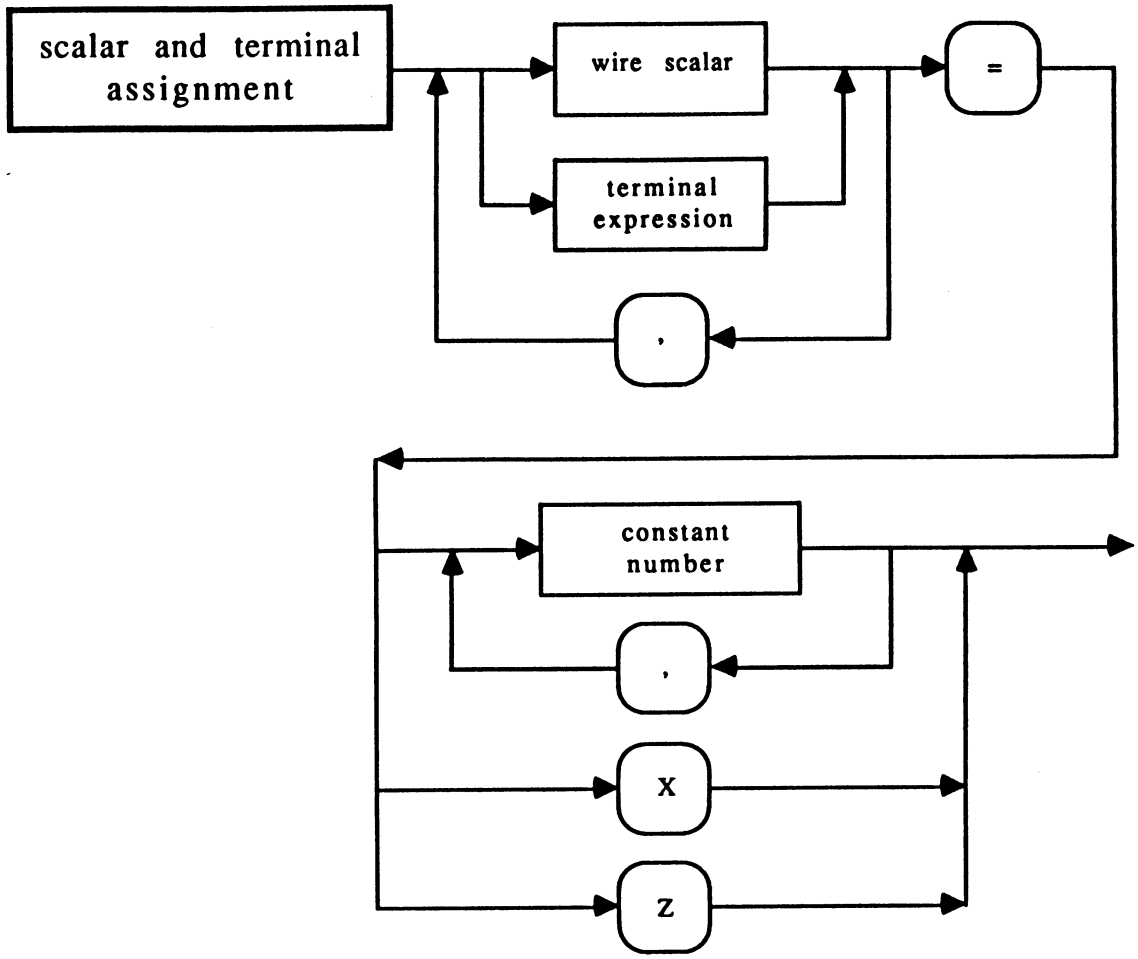

waveform description
waveform header
waveform declaration
waveform clause
end of waveform

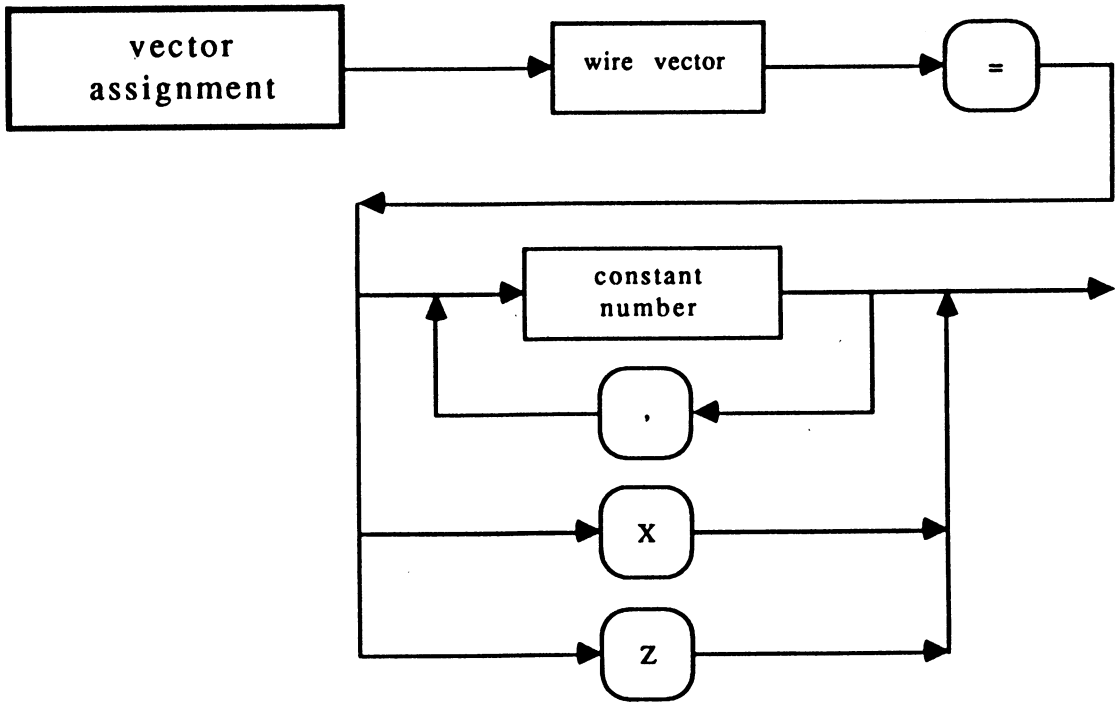

waveform header
WAVEFORM
waveform name

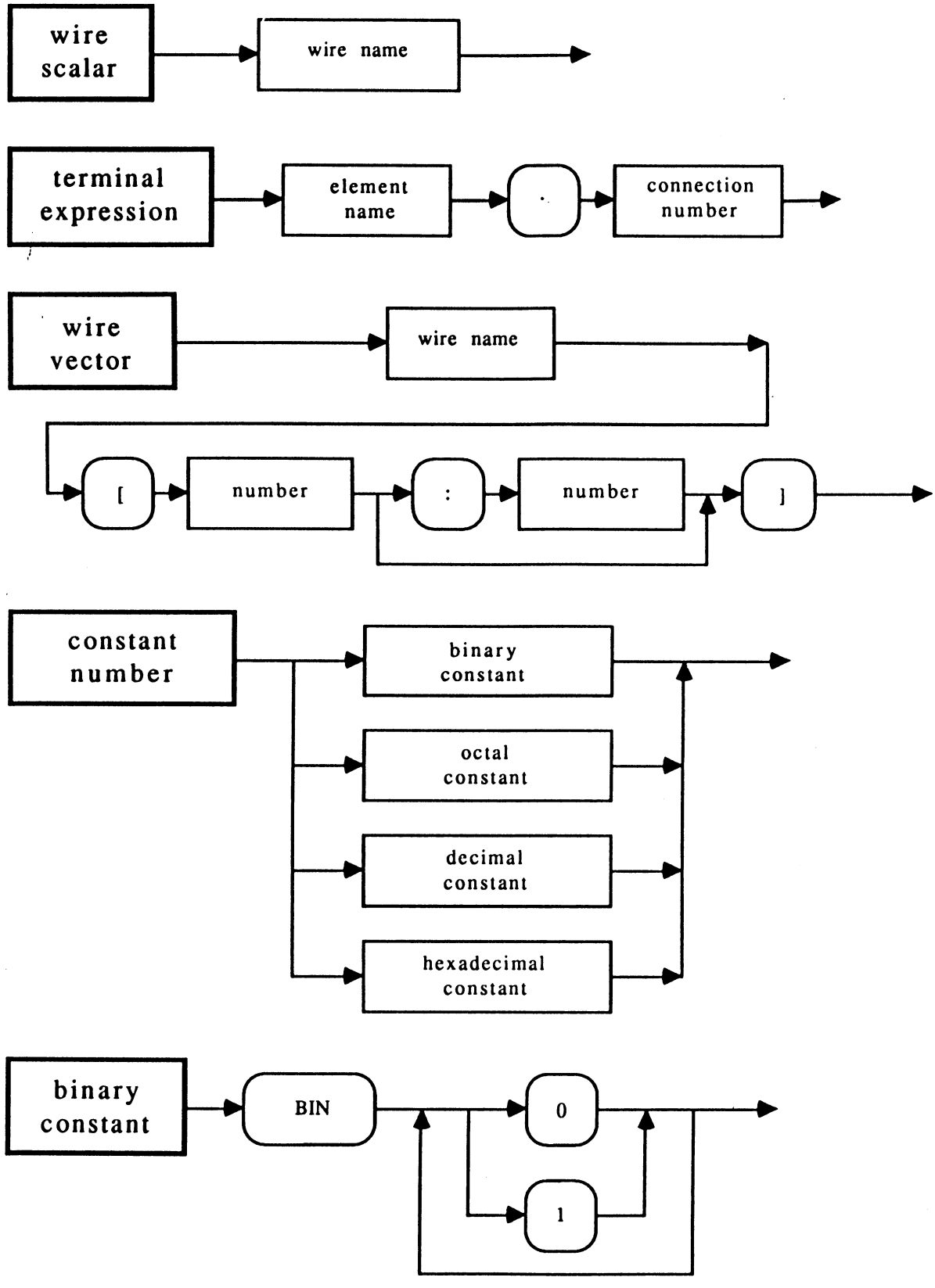

waveform name
identifier

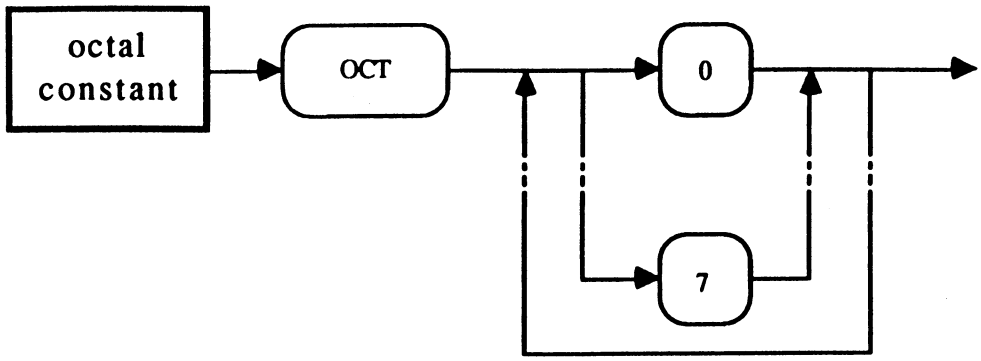

waveform declaration
stimulus declaration
response declaration
bidirectional declaration
integer declaration
time declaration
table declaration
;

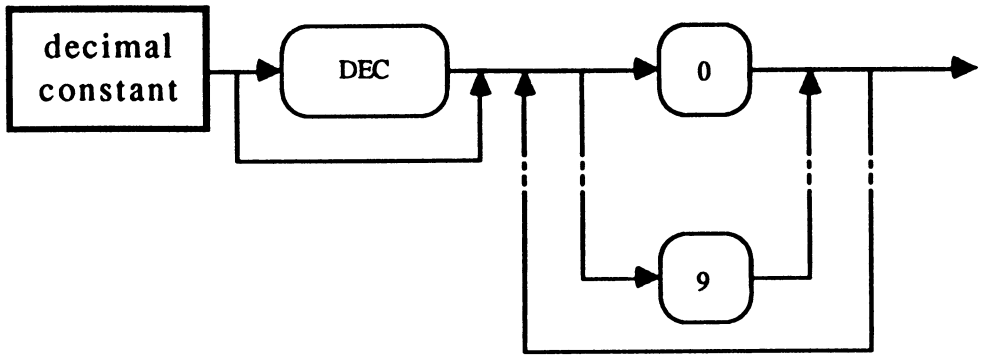

stimulus declaration
STIMULUS
scalar and terminal assignment
vector assignment

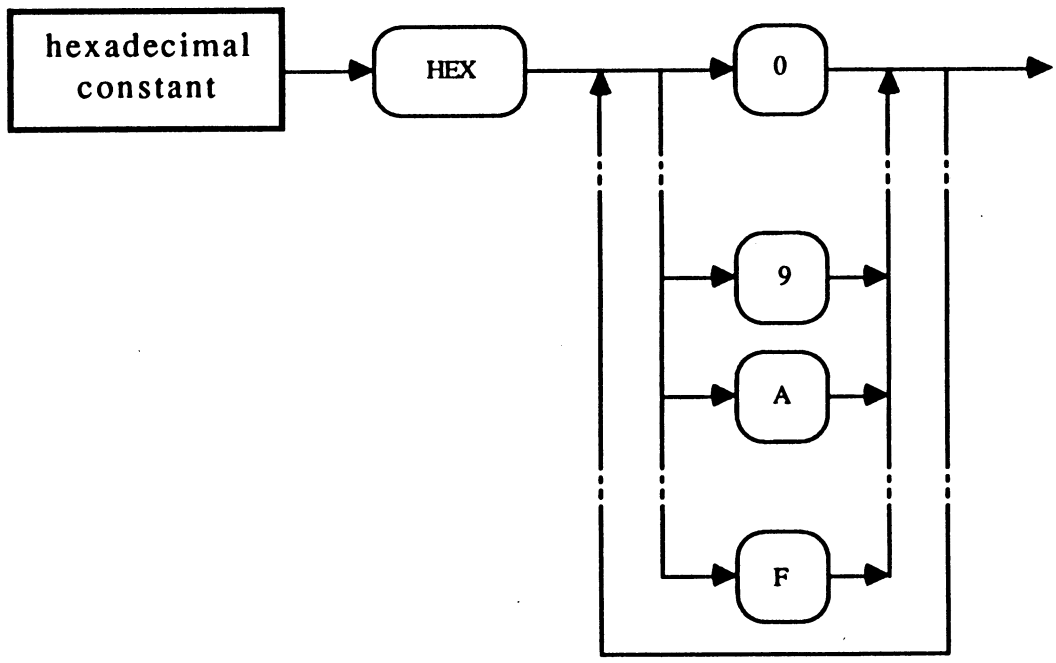

scalar and terminal assignment
wire scalar
terminal expression
=
,
constant number
,
X
Z

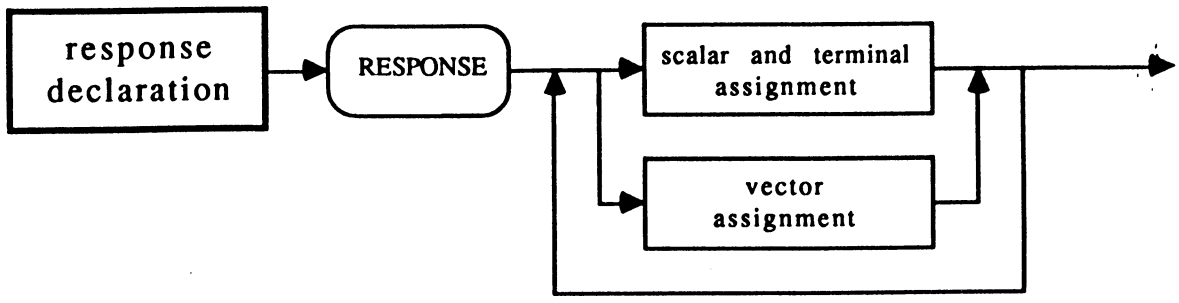

vector assignment
wire vector
=
constant number
,
X
Z

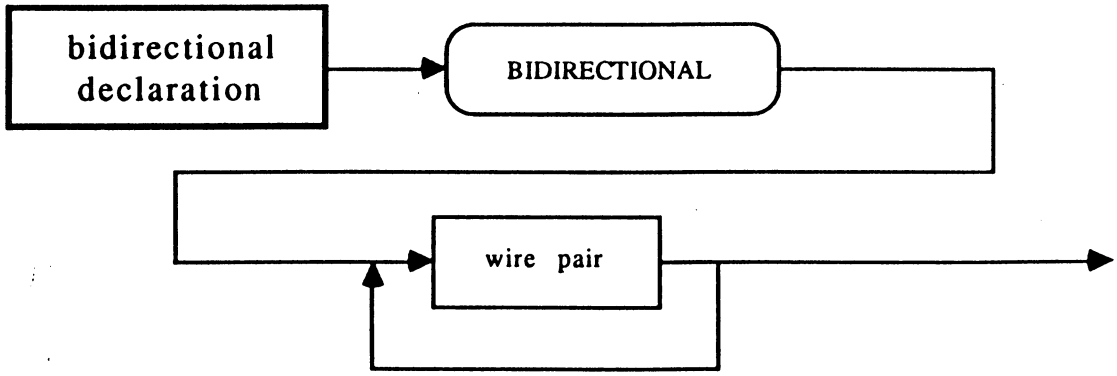

wire scalar
wire name
terminal expression
element name
.
connection number
wire vector
wire name
[
number
:
number
]
constant number
binary constant
octal constant
decimal constant
hexadecimal constant
binary constant
BIN
0
1

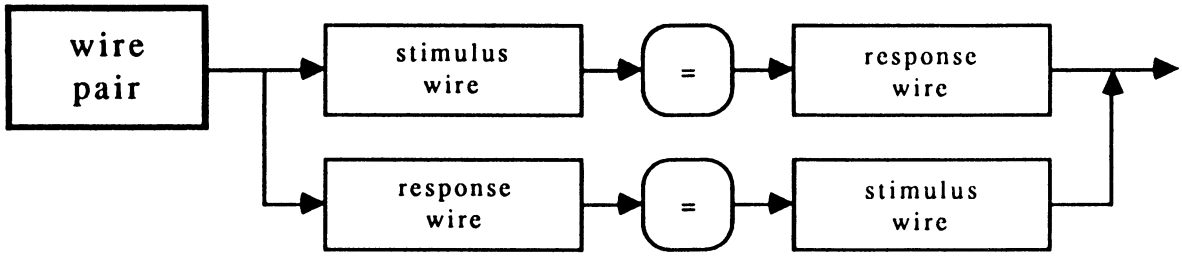

octal constant
OCT
0
7

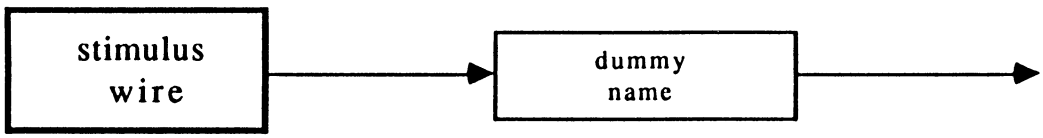

decimal constant
DEC
0
9

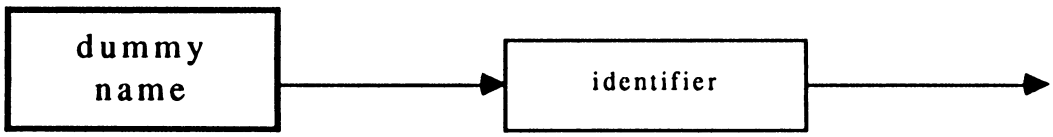

hexadecimal constant
HEX
0
9
A
F

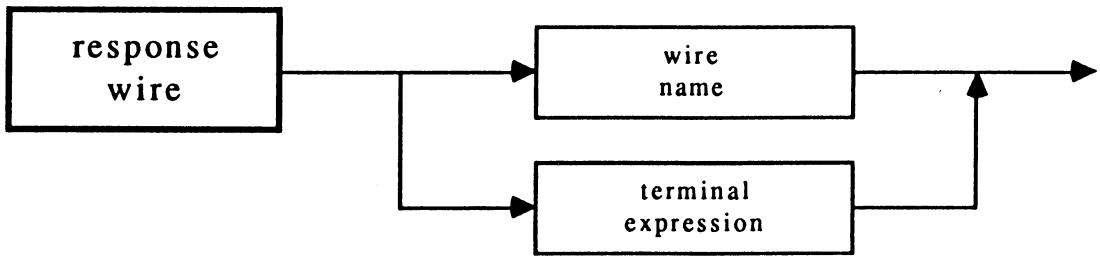

response declaration
RESPONSE
scalar and terminal assignment
vector assignment

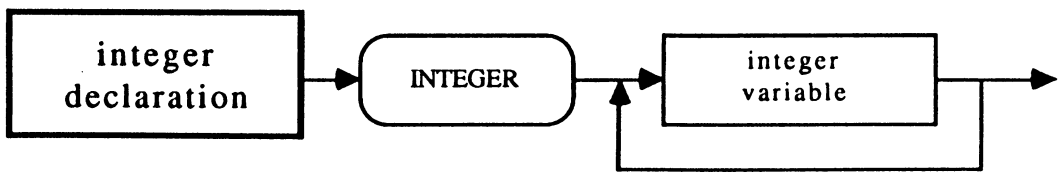

bidirectional declaration
BIDIRECTIONAL
wire pair

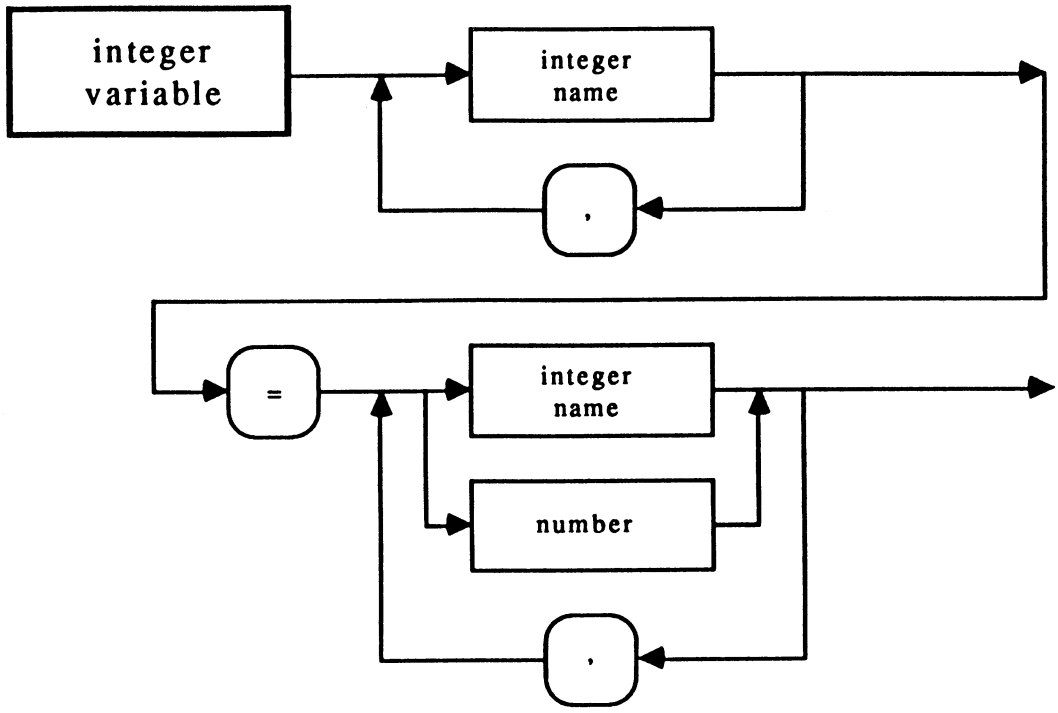

wire pair
stimulus wire
=
response wire
response wire
=
stimulus wire

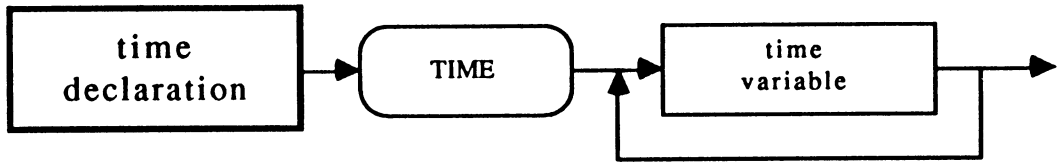

stimulus wire
dummy name

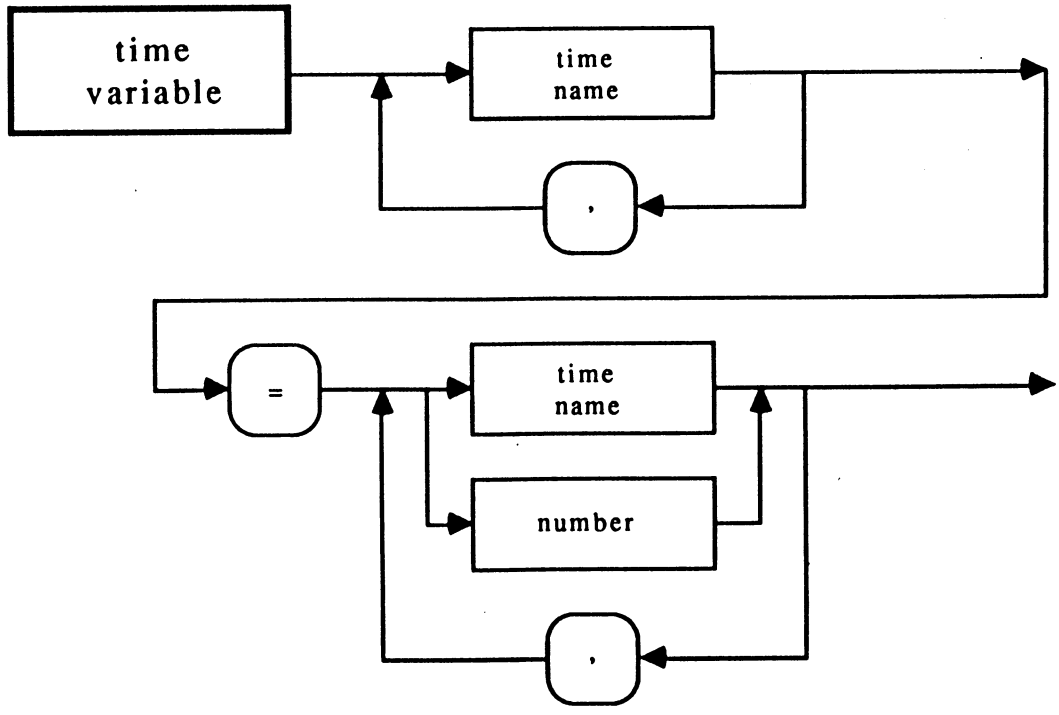

dummy name
identifier


response wire
wire name
terminal expression


integer declaration
INTEGER
integer variable


integer variable
integer name
,
=
integer name
number
,


time declaration
TIME
time variable


time variable
time name
,
=
time name
number
,

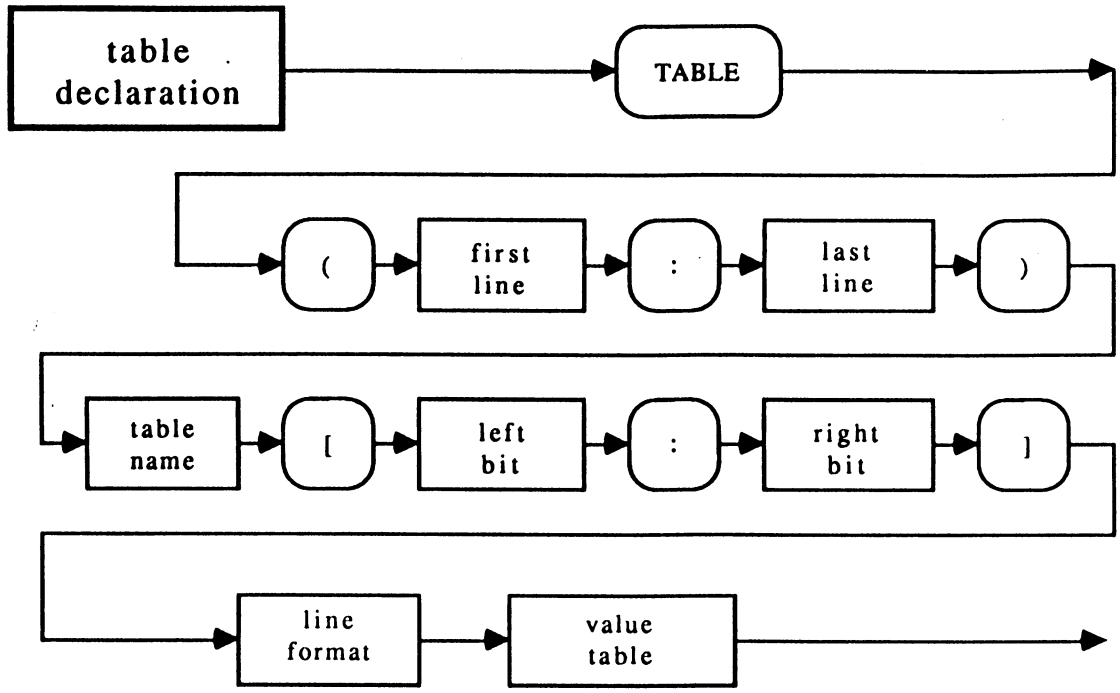
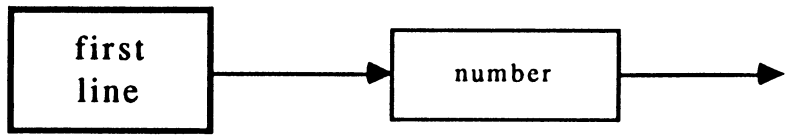
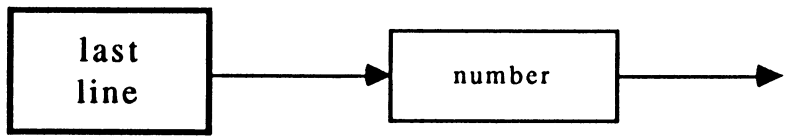


table declaration
TABLE
(
first line
:
last line
)
table name
[
left bit
:
right bit
]
line format
value table


first line
number


last line
number

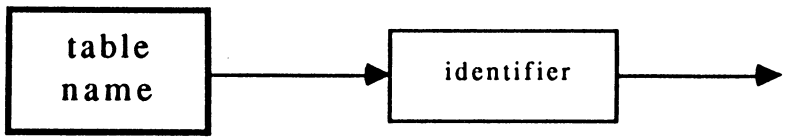
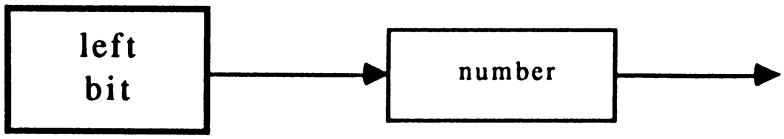
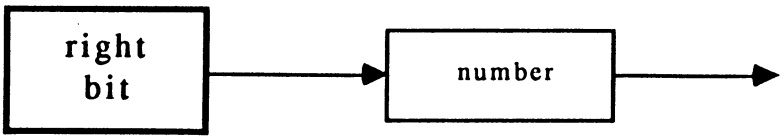
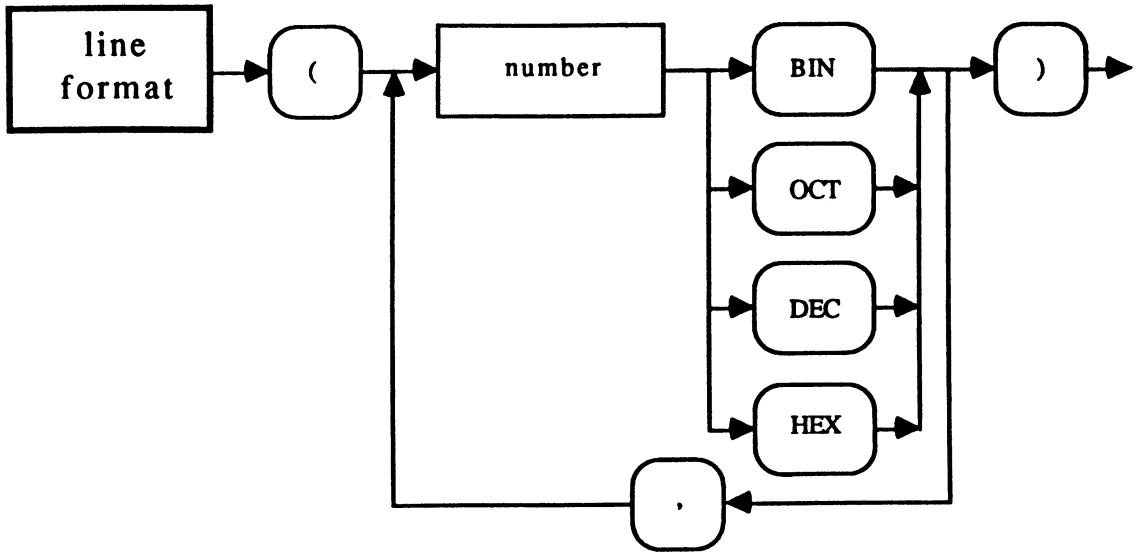
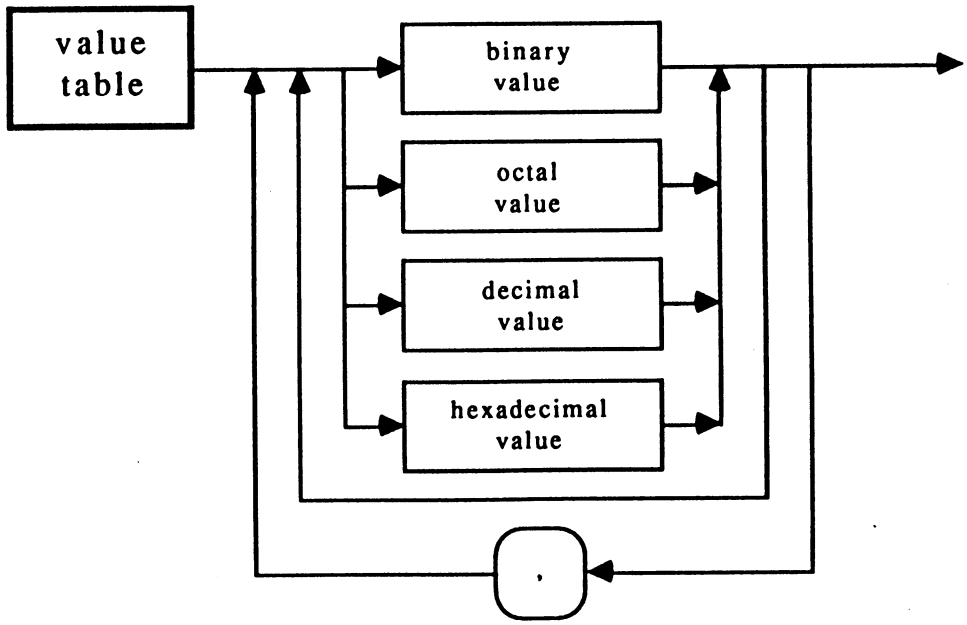


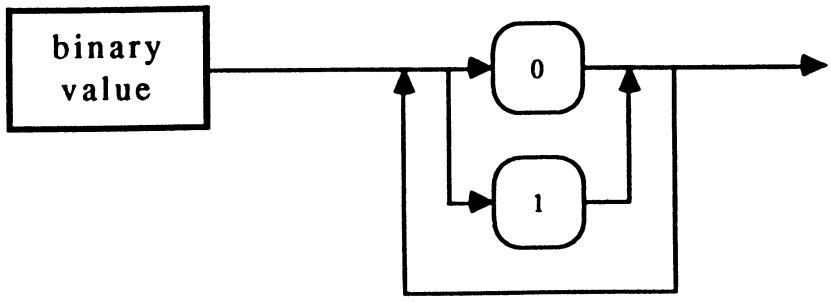
table name
identifier

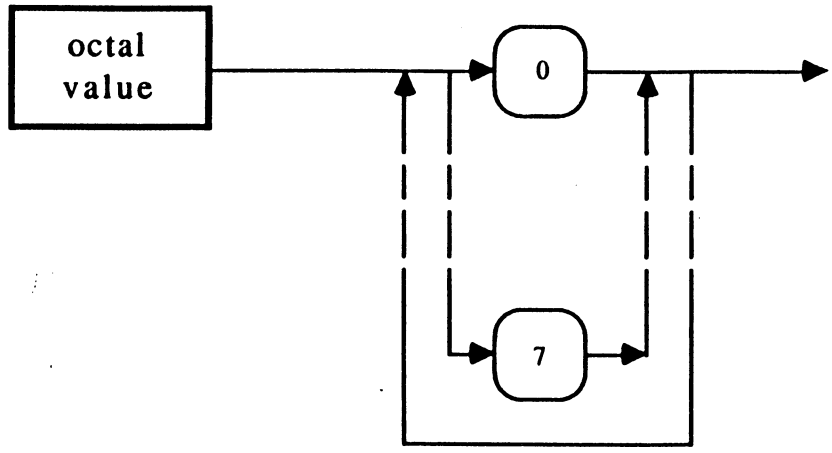

left bit
number

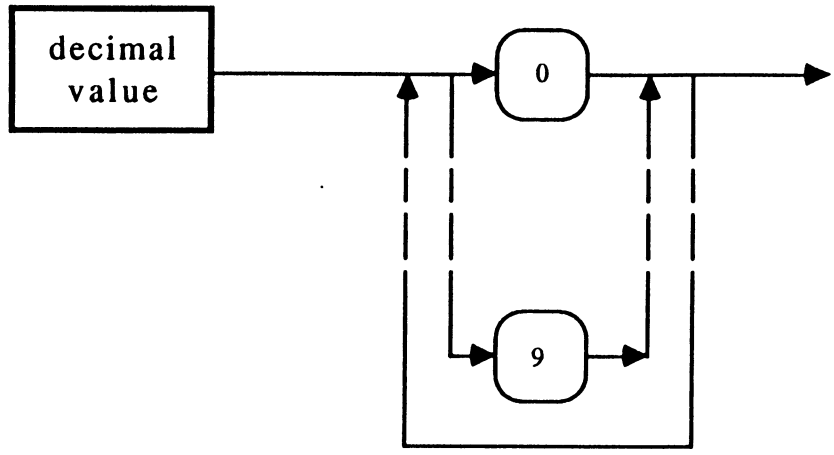

right bit
number

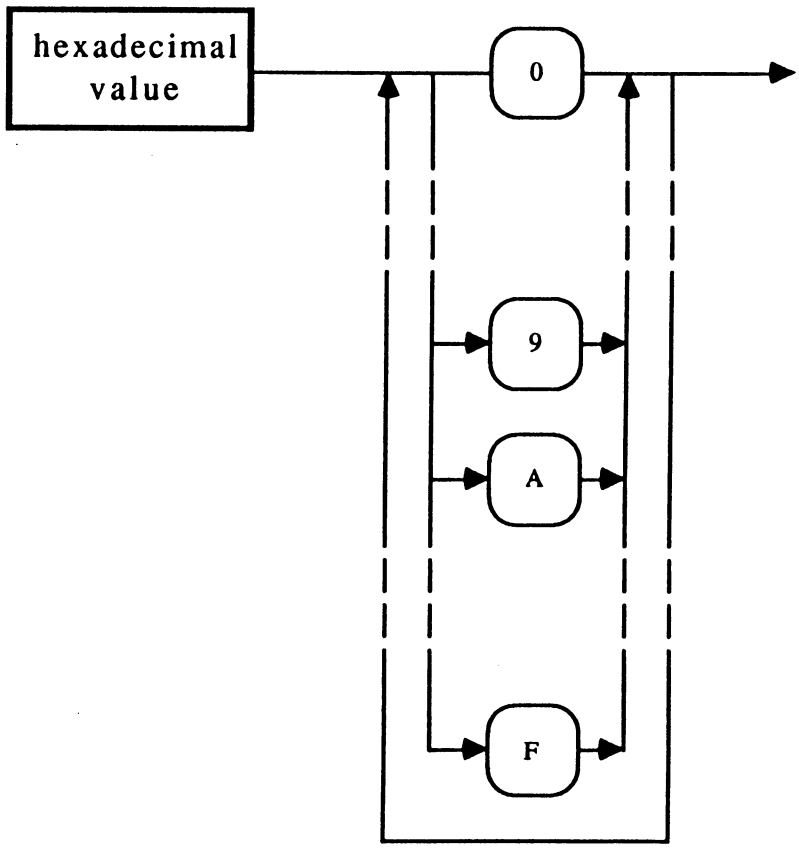

line format
(
number
BIN
OCT
DEC
HEX
)
,

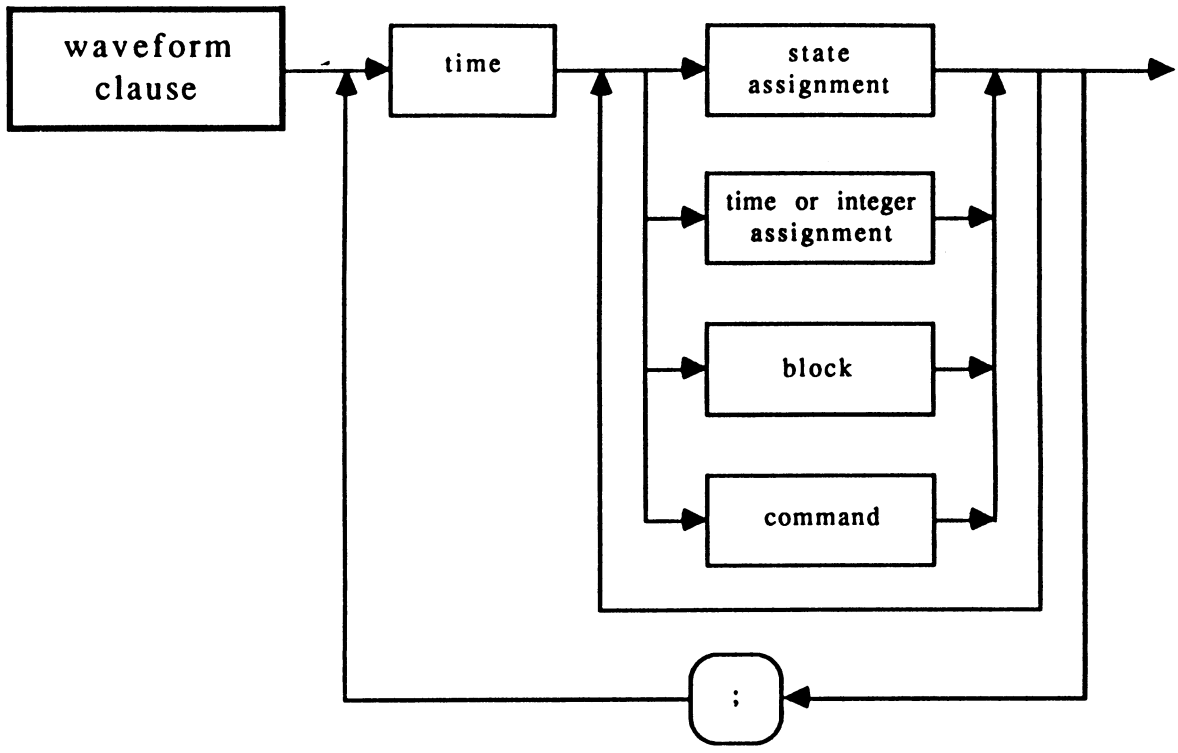

value table
binary value
octal value
decimal value
hexadecimal value
,

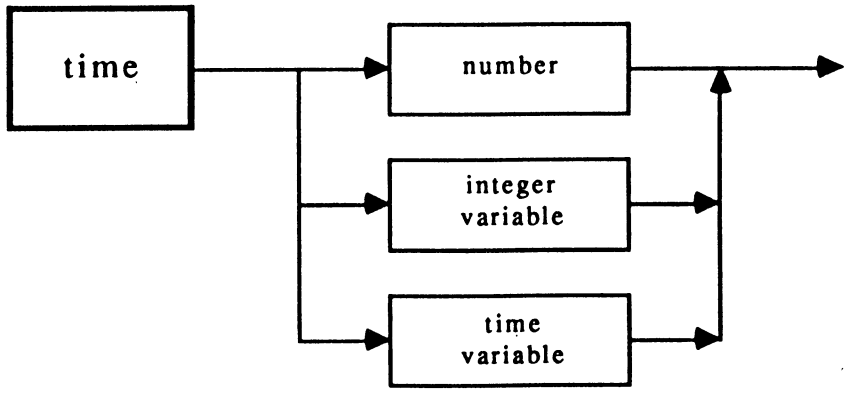

binary value
0
1

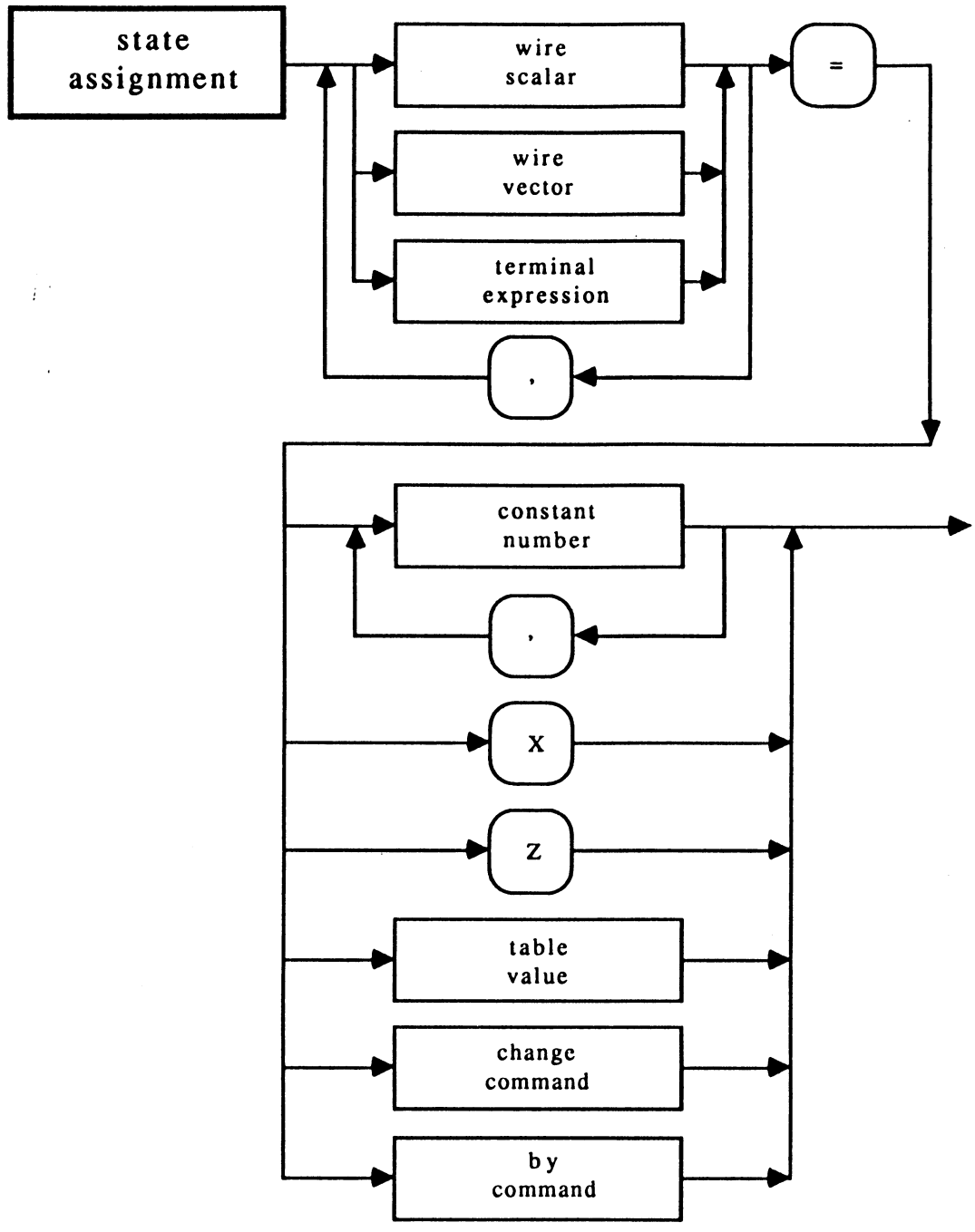

octal
value
0
7


decimal
value
0
9


hexadecimal
value
0
9
A
F


waveform clause
time
state assignment
time or integer assignment
block
command
;


time
number
integer variable
time variable


state assignment
wire scalar
wire vector
terminal expression
,
=
constant number
,
X
Z
table value
change command
by command

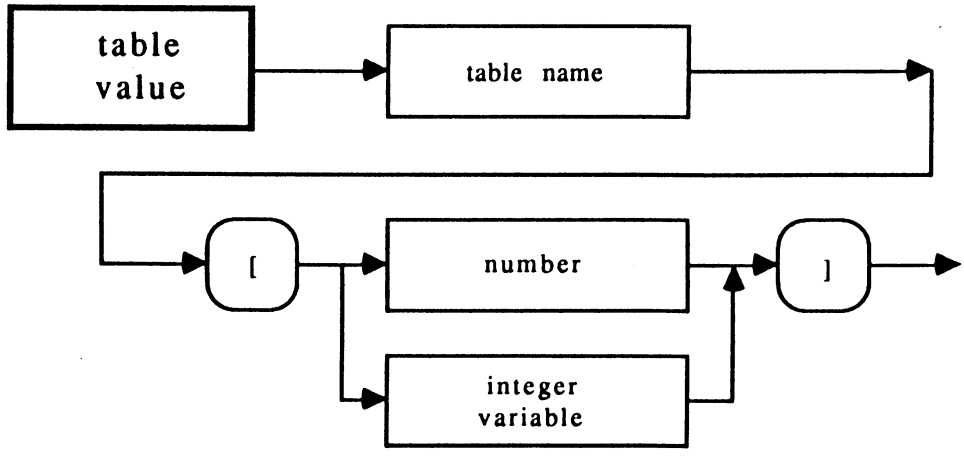
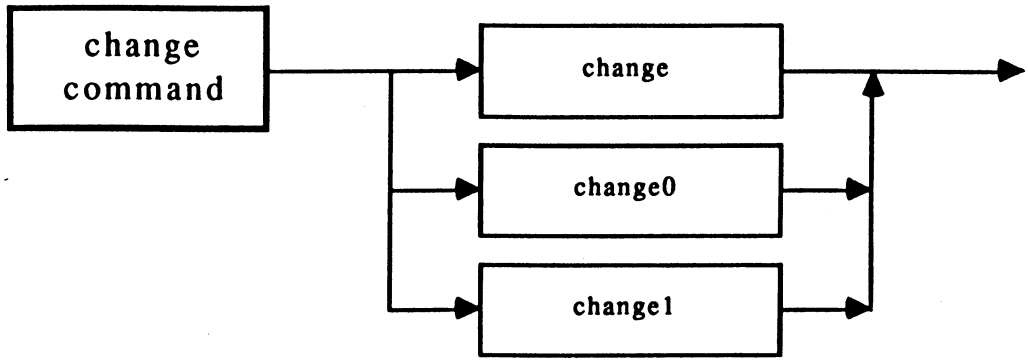
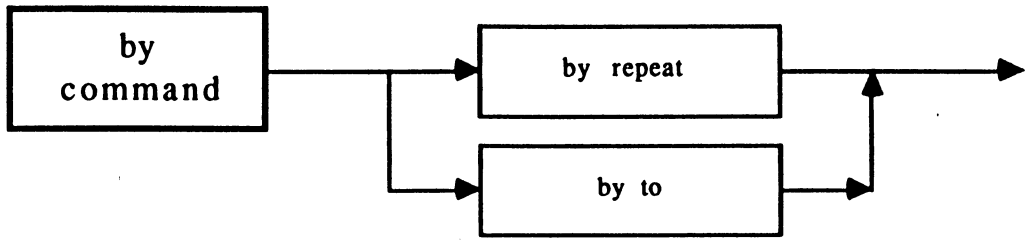
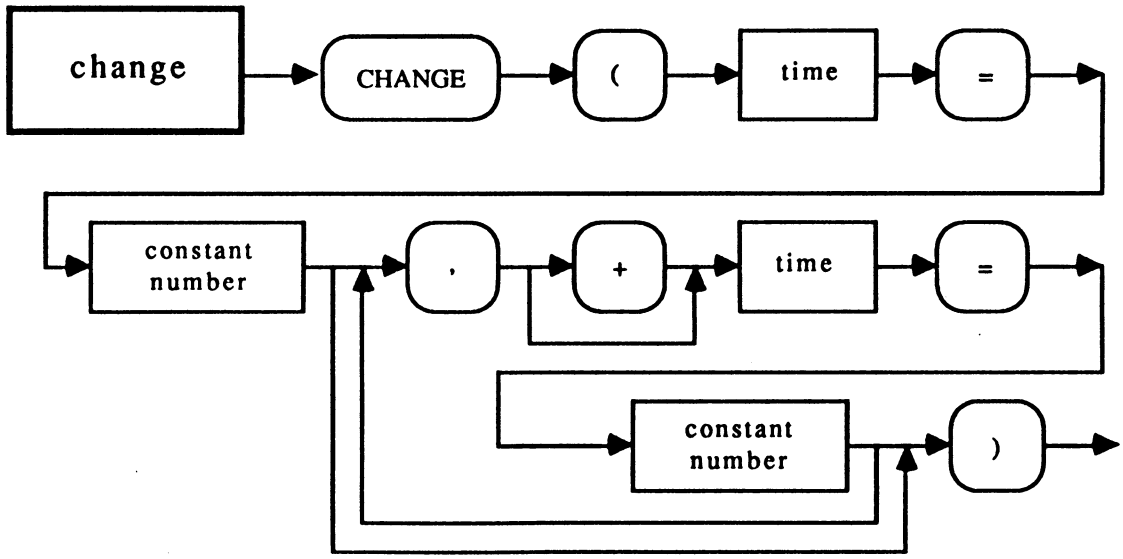
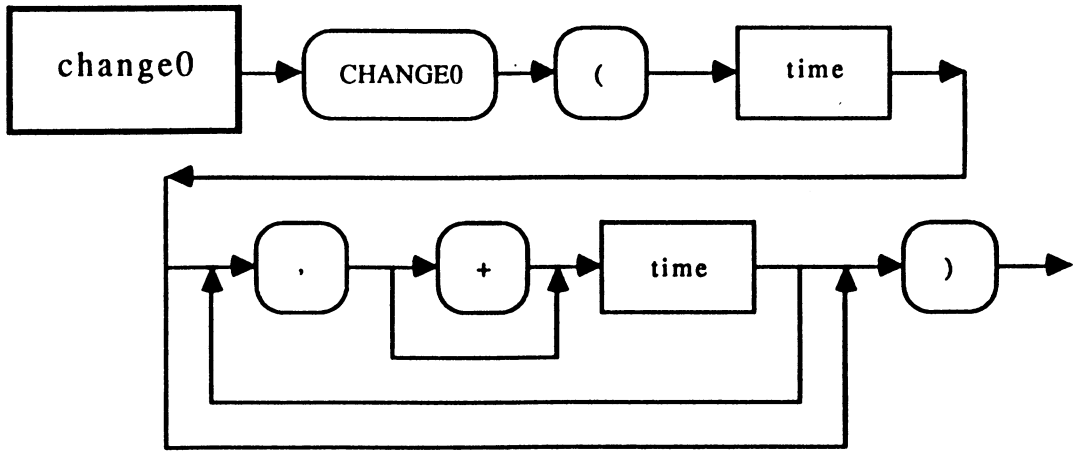


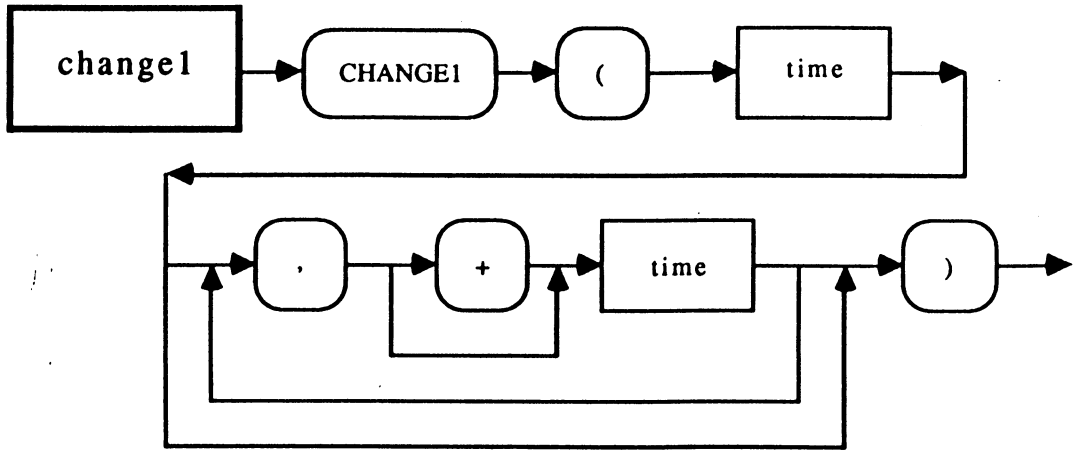
table value
table name
[
number
]
integer variable

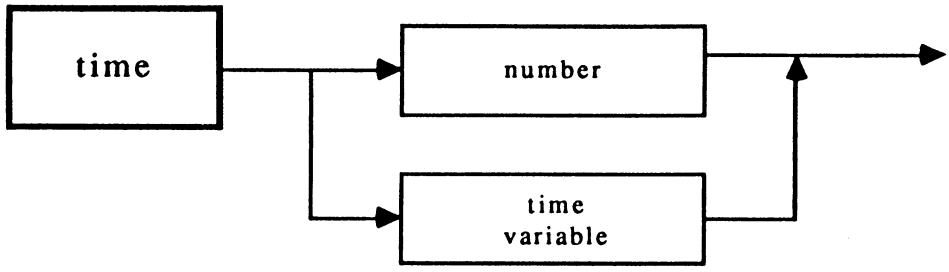

change command
change
change0
change1

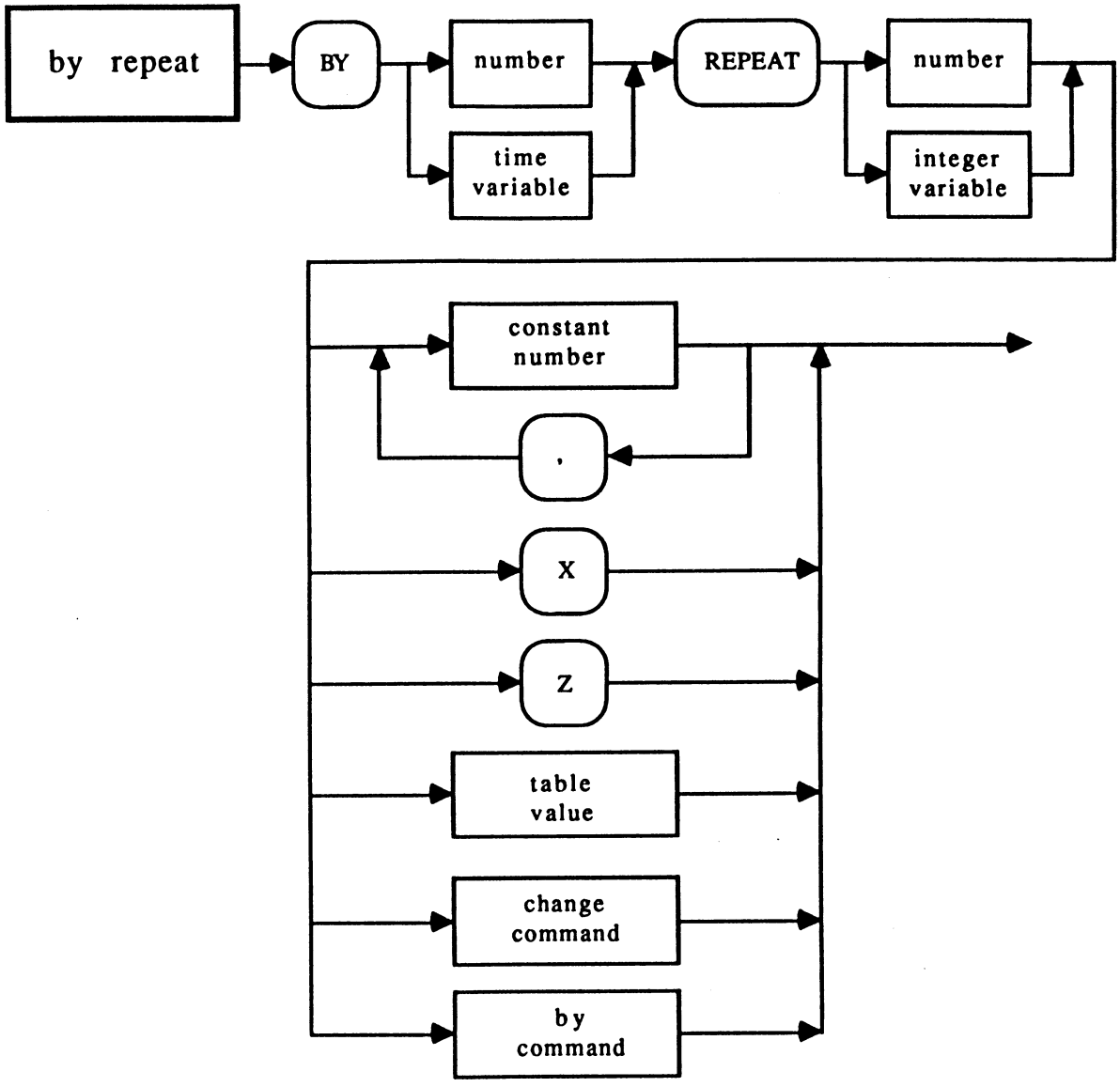

by command
by repeat
by to

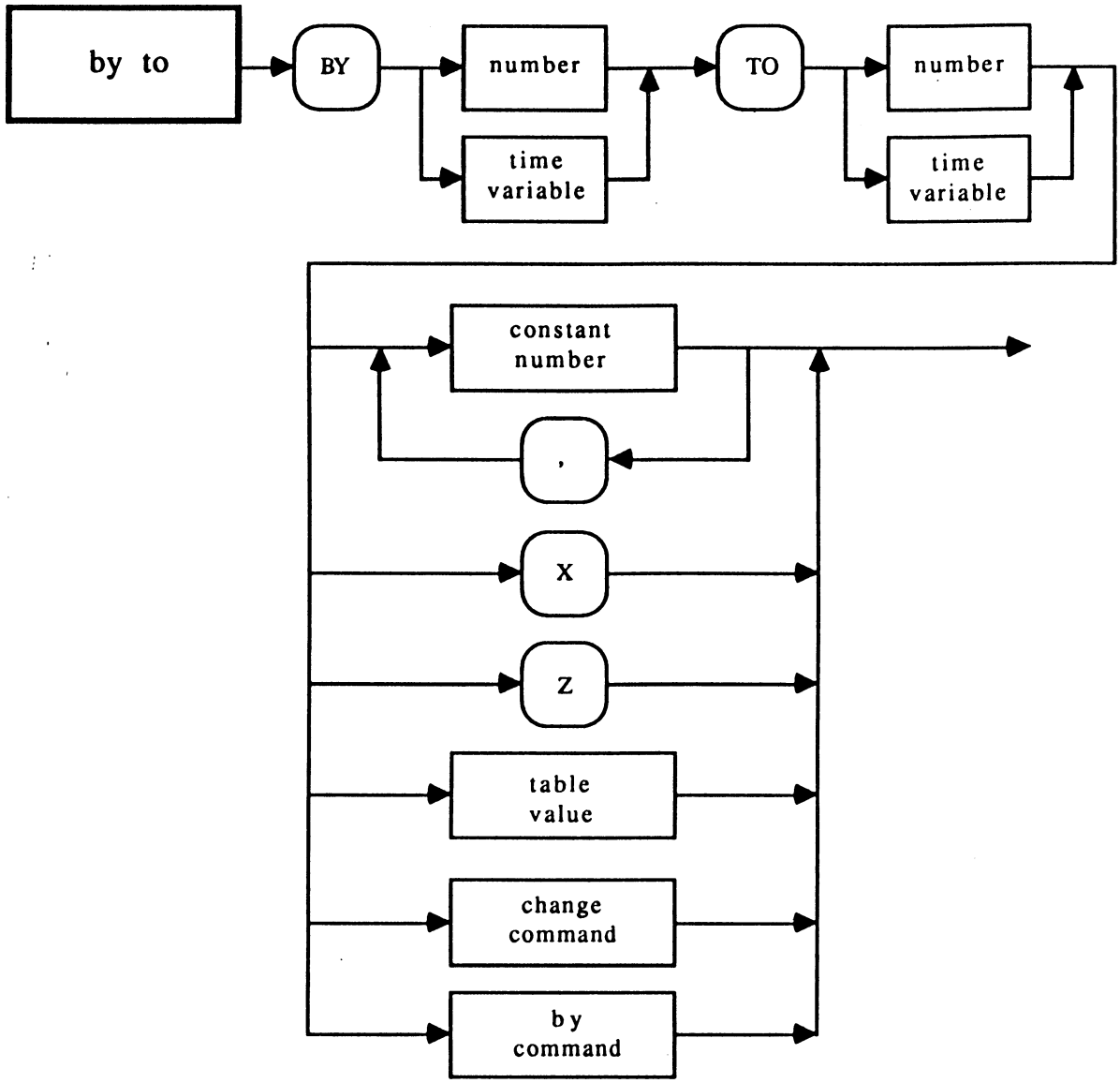

change
CHANGE
(
time
=
constant number
,
+
time
=
constant number
)

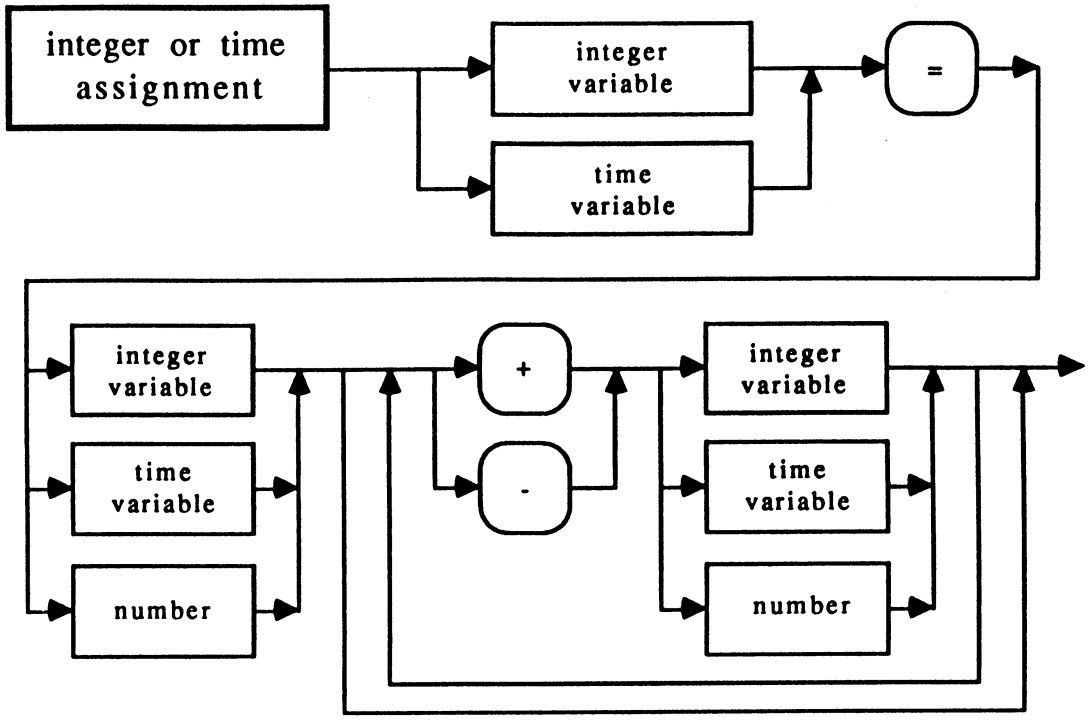

change0
CHANGE0
(
time
,
+
time
)

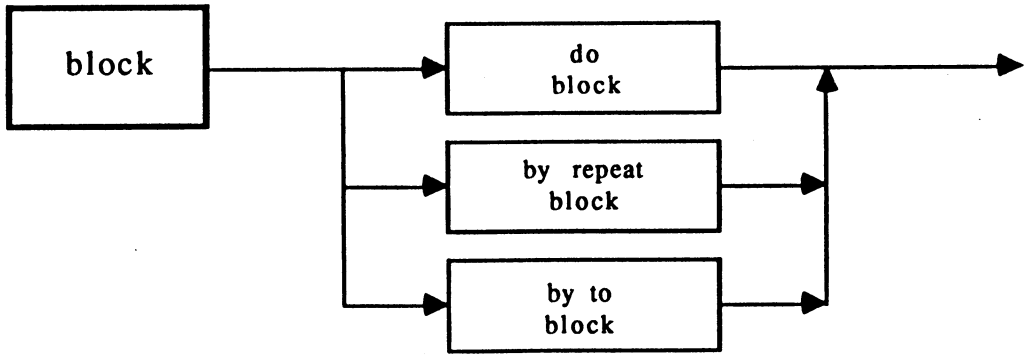

change1
CHANGE1
(
time
,
+
time
)

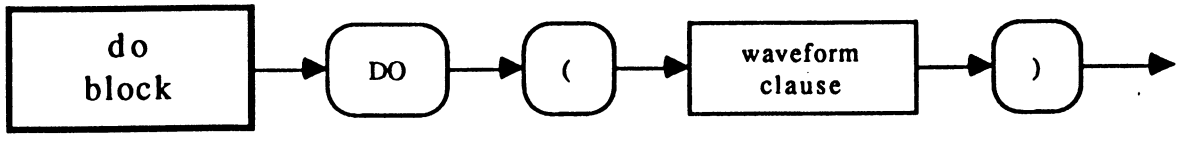

time
number
time
variable

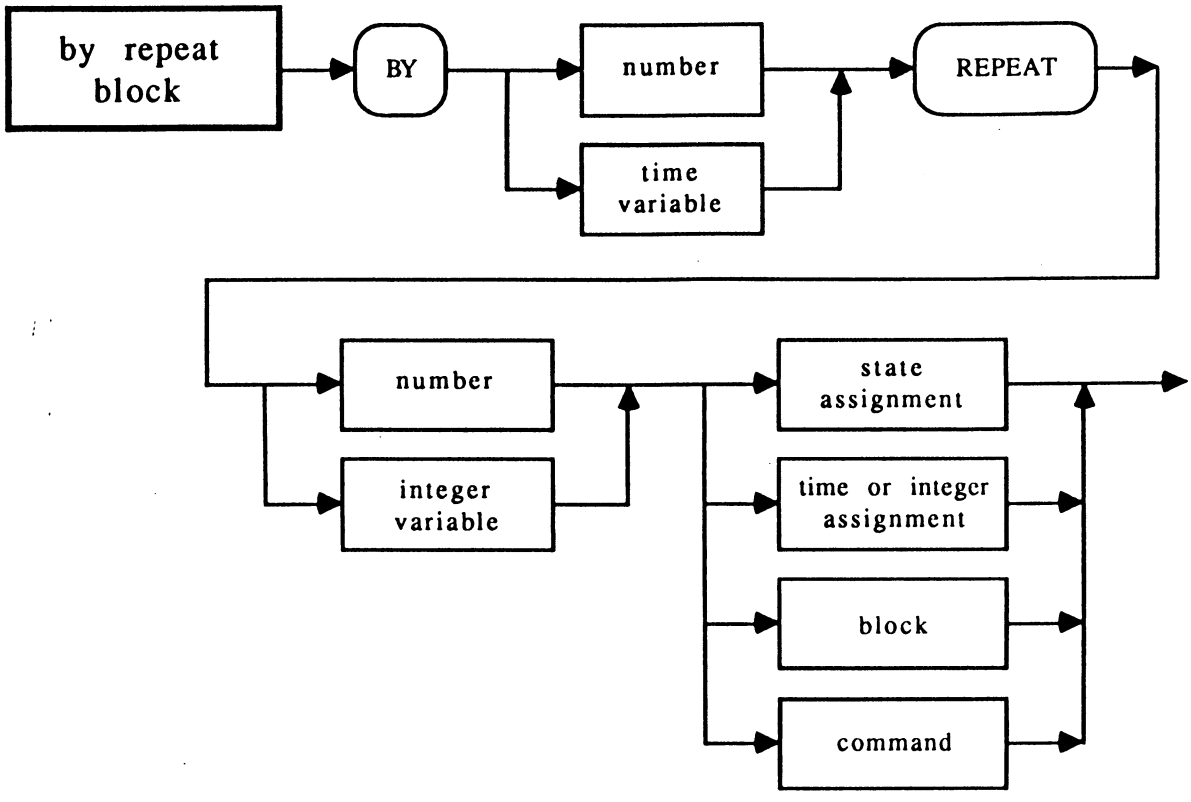

by repeat
BY
number
time variable
REPEAT
number
integer variable
constant number
,
X
Z
table value
change command
by command

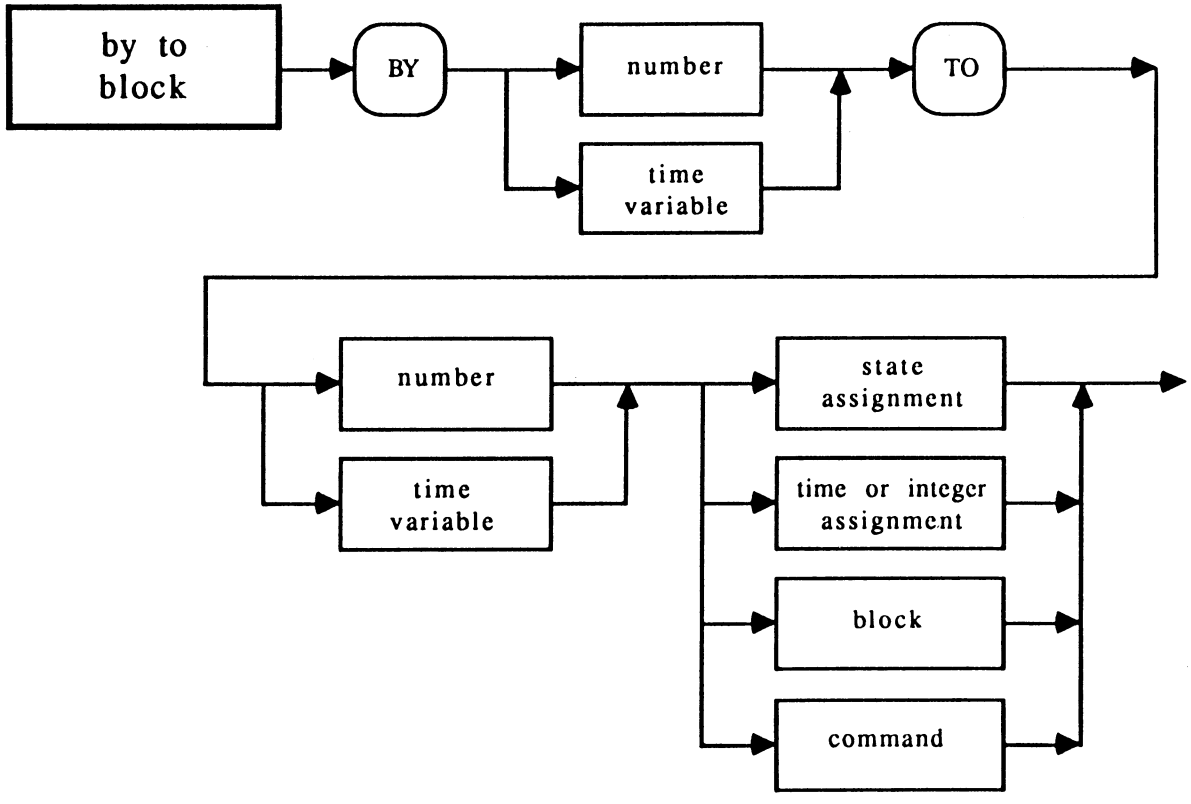

by to
BY
number
time variable
TO
number
time variable
constant number
,
X
Z
table value
change command
by command

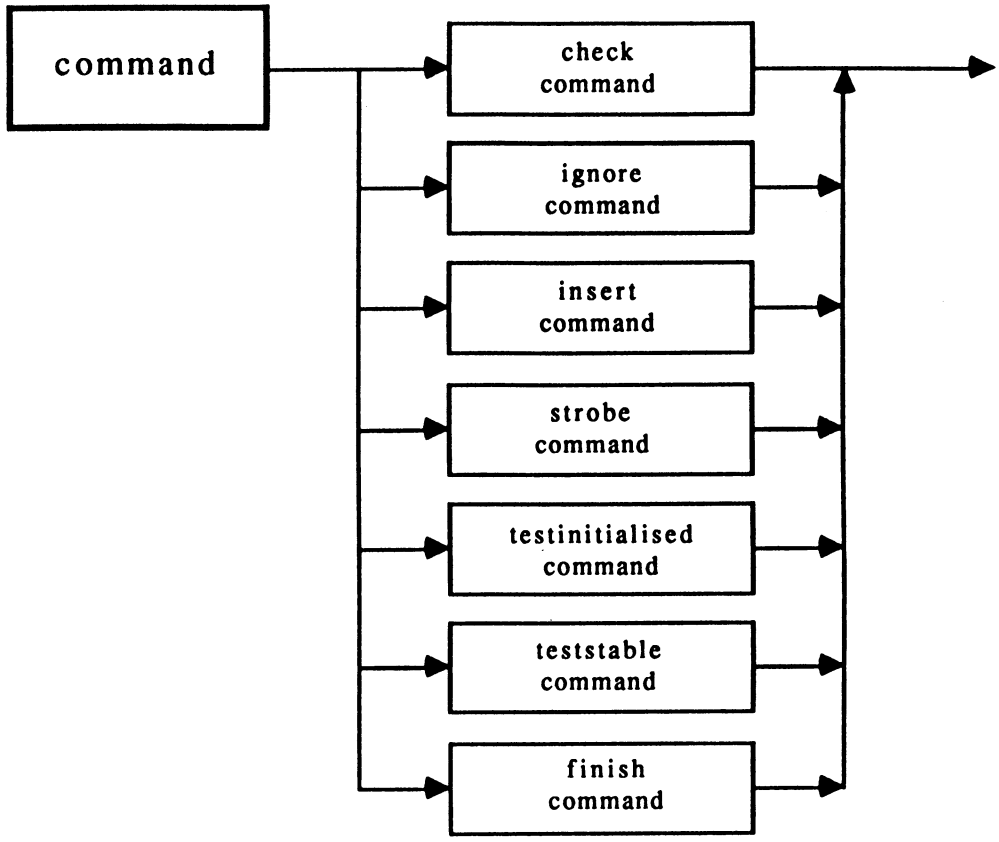

integer or time assignment
integer variable
time variable
=
integer variable
time variable
number
+
-
integer variable
time variable
number

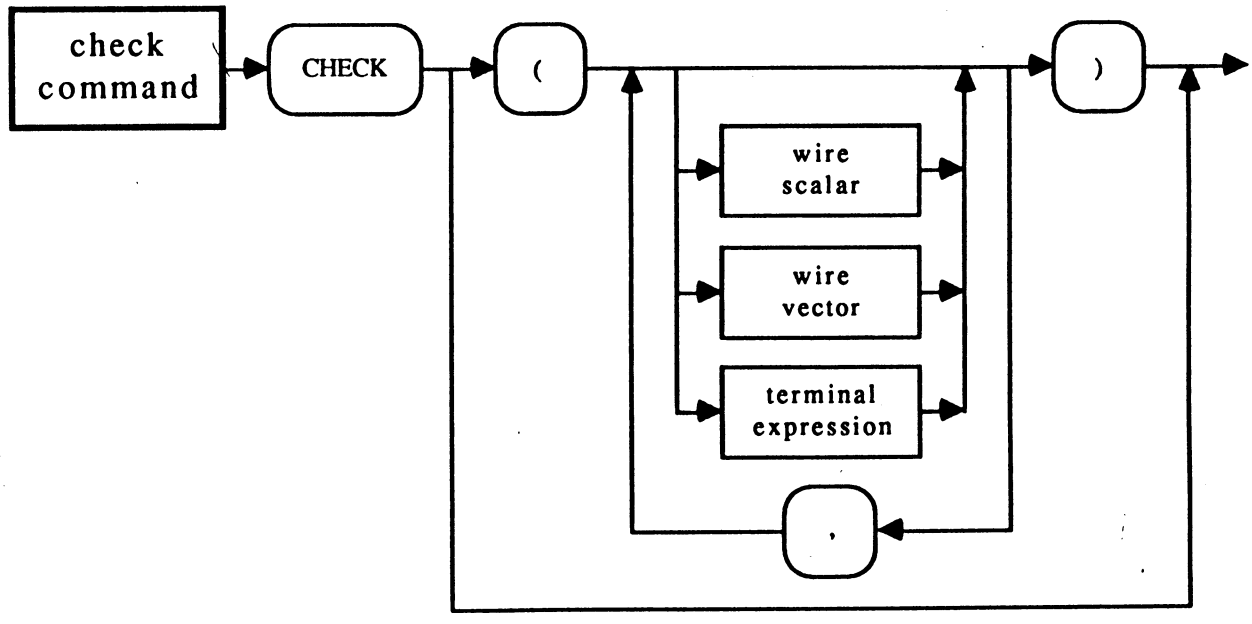

block
do block
by repeat block
by to block

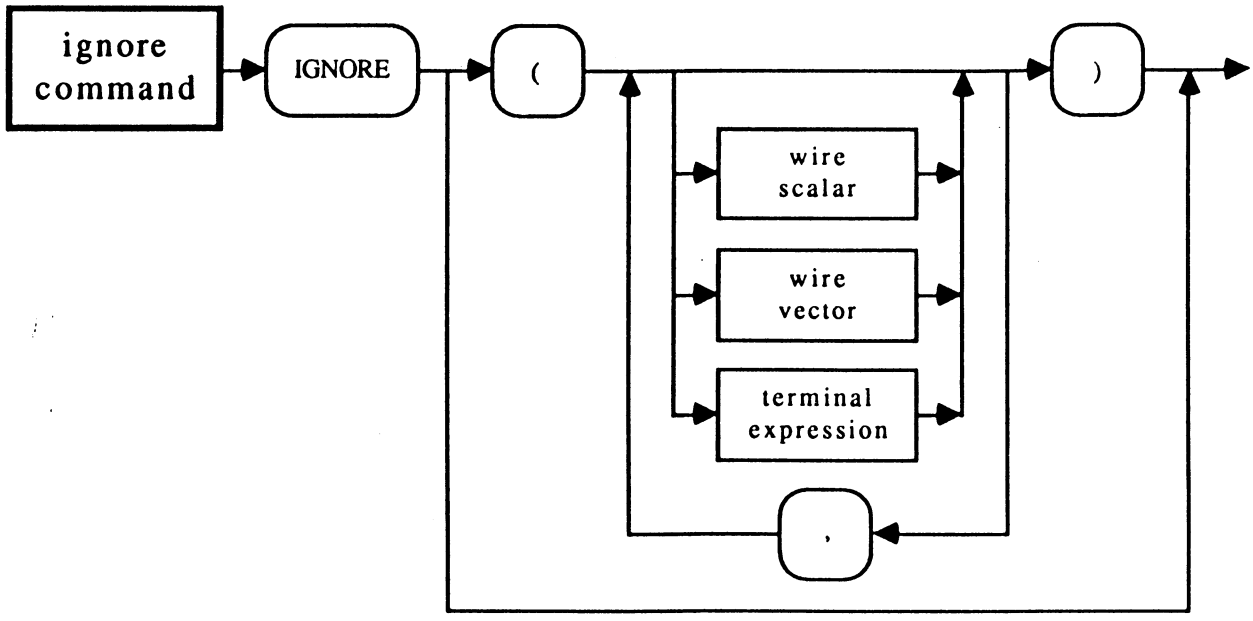

do block
DO
(
waveform clause
)

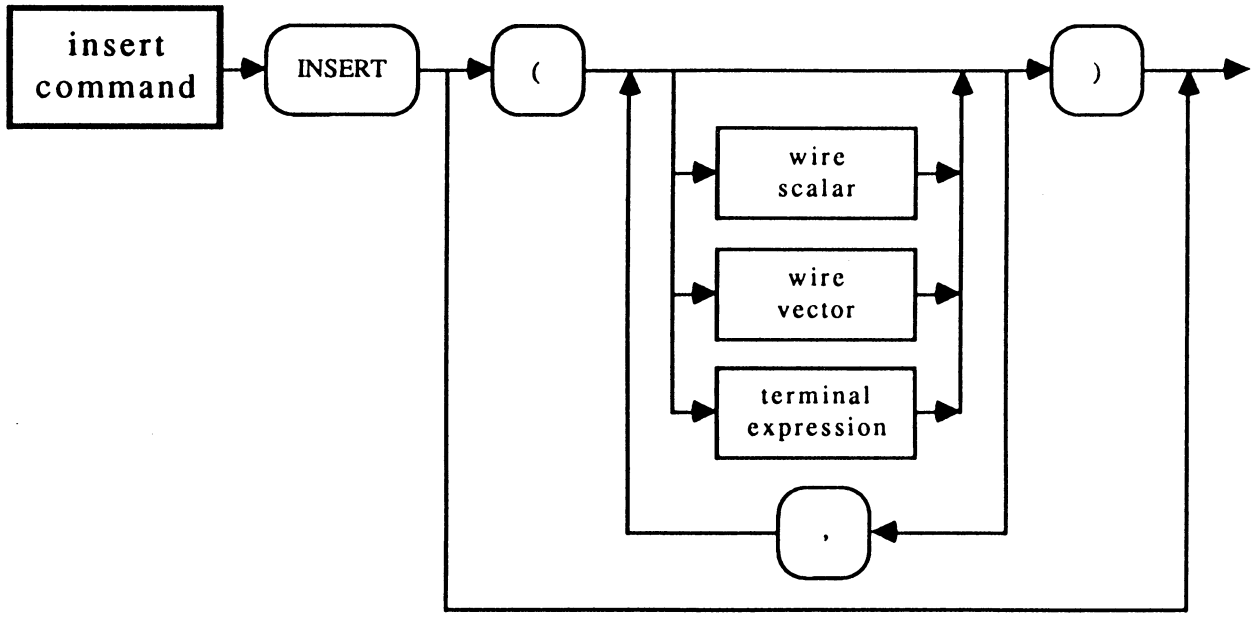

by repeat block
BY
number
time variable
REPEAT
number
integer variable
state assignment
time or integer assignment
block
command

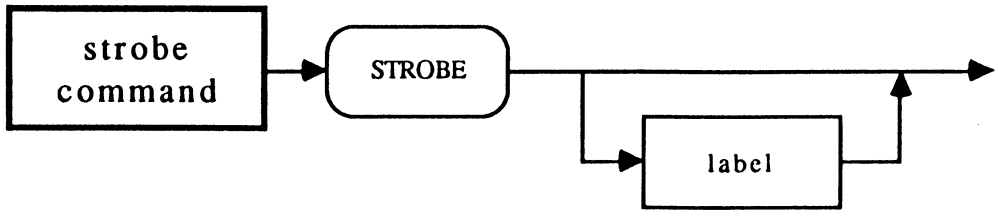

by to block
BY
number
time variable
TO
number
time variable
state assignment
time or integer assignment
block
command

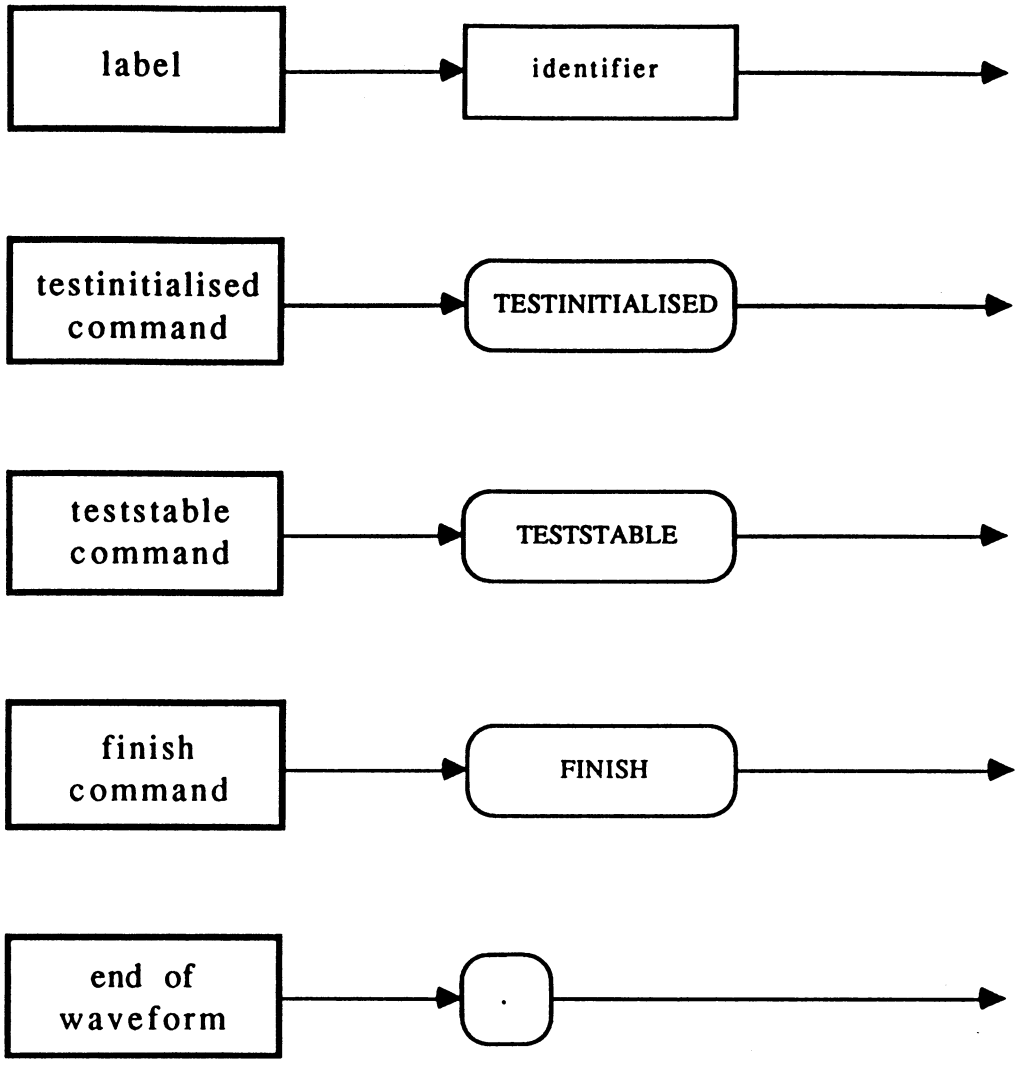

command
check command
ignore command
insert command
strobe command
testinitialised command
teststable command
finish command


check command
CHECK
(
)
wire scalar
wire vector
terminal expression
,


ignore command
IGNORE
(
)
wire scalar
wire vector
terminal expression
,


insert command
INSERT
(
)
wire scalar
wire vector
terminal expression
,


strobe command
STROBE
label


label
identifier
testinitialised command
TESTINITIALISED
teststable command
TESTSTABLE
finish command
FINISH
end of waveform
.

Annexe C

Résultats de différentes méthodes de partitionnement

C.1. Méthodes et Critères

C.1.1. Méthodes de partitionnement

◊ Méthode du placement aléatoire

Cette méthode consiste à placer, en travaillant sur une description de circuit à plat, les éléments du circuit sur les unités de simulation d'une manière aléatoire.

◊ Méthode du découpage linéaire

La description du circuit est sous forme de liste d'éléments interconnectés. Le partitionnement consiste à découper cette liste en N segments de taille identique ou approchée et à les distribuer sur les unités de simulation, N étant le nombre d'unités de simulation.

◊ Méthode par la connectivité d'entrée

Cette méthode minimise les communications entre les unités de simulation. Les unités de simulation sont affectées d'un nombre identique ou approché d'éléments. La sélection des éléments tient compte de la connectivité des éléments. Lorsqu'un élément est placé sur une unité de simulation, les éléments connectés à ses sorties y sont également placés, jusqu'à ce que le nombre d'éléments souhaité soit atteint.

◊ Méthode par la connectivité de sortie

Cette méthode est basée sur le même principe que la méthode précédente. Mais la prise en compte de la connectivité se fait au niveau des sorties d'éléments.

◊ Méthode par la hiérarchie du circuit

La description du circuit reste hiérarchique. Au lieu de mettre à plat tout le circuit, on le décompose en sous-circuits qui seront distribués sur les unités de simulation :

• Si le sous-circuit contient trop d'éléments, alors il doit être décomposé en ses sous-circuits.

• Sinon il peut être placé entièrement sur une unité de simulation.

Le partitionnement s'effectue en deux étapes :

• On descend dans la hiérarchie à partie du circuit principal pour trouver un certain nombre de sous-circuits et d'éléments primitifs en décomposant le circuit et les sous-circuits. Cette descente s'arrête au moment où le partitionnement devient possible.

• On distribue les sous-circuits et les primitives sur les unités de simulation en utilisant la méthode linéaire.

C.1.2. Critères de comparaisons

◊ Déviation de nombres d'éléments

La déviation de nombres d'éléments définit l'écart relatif de l'équilibrage des nombres d'éléments sur les unités de simulation.

◊ Déviation de poids d'éléments

La déviation de poids d'éléments définit l'écart relatif de l'équilibrage des charges d'éléments sur les unités de simulation.

Le poids d'un élément est calculé d'une manière simpliste :

Entrance + Sortance + Nombre d'instructions fonctionnelles

◊ Déviation de nombres d'équipotentielles

La déviation de nombres d'équipotentielles définit l'écart relatif de l'équilibrage des nombres d'équipotentielles sur les unités de simulation.

◊ Nombre d'équipotentielles globales

Les équipotentielles globales sont celles qui traversent plusieurs unités de simulation. Un partitionnement doit minimiser le nombre d'équipotentielles globales qui nécessitent des communications entre les unités de simulation pendant la simulation.

◊ Somme de distances logiques

La distance logique définit le nombre d'unités de simulation à traverser par une équipotentielle globale. Un partitionnement idéal doit éviter des connexions à longue distance.

La somme de distances logiques est celle des distances logiques de toutes les équipotentielles globales du circuit.

C.2. Résultats et comparaisons

C.2.1. Circuits sélectionnés

Quatre circuits de test sont sélectionnés, pour tester et comparer les résultats de différentes méthodes :

- un additionneur 32 bits;
- une unité arithmétique et logique 32 bits;
- un circuit généré par un outil automatique;
- le microprocesseur MC68000 décrit en fonctionnel.

Les principales caractéristiques de ces circuits sont les suivantes :

caractéristiques / circuit	équipotentielles	éléments structurels	éléments fonctionnels	instructions fonctionnelles	structure hiérarchique
additionneur 32 bits	357	288	0	0	oui
UAL 32 bits	1701	1342	515	10051	oui
circuit généré par outil	8387	7058	4125	141605	non
MC68000 fonctionnel	3558	137	1261	46766	non

Trois configurations matérielles sont prises en compte, qui sont respectivement les cas de 4, 16 et 64 unités de simulation.

C.2.2. Additionneur 32 bits

La déviation de nombres d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	2,8%	0,0%	0,0%	0,0%	0,0%
16	7,0%	0,0%	0,0%	0,0%	0,0%
64	17,2%	0,0%	12,5%	12,5%	0,0%

La déviation de poids d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	8,3%	0,0%	0,0%	0,0%	0,0%
16	6,9%	0,0%	0,0%	0,0%	0,0%
64	16,1%	0,0%	11,5%	11,5%	0,0%

La déviation de nombres d'équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	4,2%	2,1%	3,1%	4,5%	0,0%
16	4,8%	5,2%	6,2%	4,1%	0,0%
64	13,8%	21,3%	12,6%	6,4%	26,7%

Le nombre d'équipotentielles globales :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	535	22	372	121	8
16	593	94	512	181	32
64	728	479	606	318	256

La somme de distances logiques des équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	1456	70	424	216	8
16	7018	4126	6012	3608	32
64	157042	113554	70714	163966	256

C.2.3. Unité arithmétique et logique 32 bits

La déviation de nombres d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	1,2%	0,0%	0,0%	0,0%	1,7%
16	4,1%	0,0%	0,0%	0,0%	6,0%
64	7,2%	0,0%	0,0%	0,0%	0,0%

La déviation de poids d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	7,5%	12,1%	35,1%	35,7%	13,7%
16	8,5%	27,5%	57,5%	52,9%	27,6%
64	15,2%	43,4%	71,5%	58,6%	43,4%

La déviation de nombres d'équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	3,0%	11,3%	19,5%	28,6%	2,7%
16	3,1%	24,6%	27,7%	34,0%	32,4%
64	6,9%	30,8%	40,5%	40,7%	30,8%

Le nombre d'équipotentielles globales :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	1621	1091	988	957	152
16	2854	1710	1320	1450	592
64	3619	2289	1972	1900	2289

La somme de distances logiques des équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	3718	2838	2104	1942	316
16	69672	21232	18806	17246	1697
64	820896	285924	202232	128022	285924

C.2.4. Circuit généré par un outil automatique

La déviation de nombres d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	0,9%	0,0%	0,0%	0,0%	0,0%
16	1,8%	0,0%	0,1%	0,1%	0,0%
64	2,7%	0,0%	0,4%	0,4%	0,0%

La déviation de poids d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	2,6%	1,9%	18,0%	7,5%	1,9%
16	6,6%	18,0%	26,4%	24,2%	18,0%
64	9,0%	34,5%	41,1%	42,7%	34,5%

La déviation de nombres d'équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	1,1%	10,8%	13,6%	4,2%	10,8%
16	3,1%	18,3%	23,9%	30,1%	18,3%
64	5,5%	25,7%	33,4%	41,1%	25,7%

Le nombre d'équipotentielles globales :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	10120	2334	2520	3480	2334
16	13957	3683	3865	5237	3683
64	16191	4998	4891	5898	4998

La somme de distances logiques des équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	20622	3526	3730	5170	3526
16	181910	28852	27632	31284	28852
64	2172606	406874	291634	183164	406874

C.2.5. MC68000 fonctionnel

La déviation de nombres d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	1,8%	0,0%	0,1%	0,1%	0,0%
16	5,1%	0,0%	0,4%	0,4%	0,0%
64	8,3%	0,0%	0,4%	0,4%	0,0%

La déviation de poids d'éléments :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	25,0%	11,6%	12,0%	9,0%	11,6%
16	12,6%	19,9%	21,9%	20,0%	19,9%
64	22,2%	59,0%	27,3%	28,4%	59,0%

La déviation de nombres d'équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	7,2%	83,4%	83,1%	88,0%	83,4%
16	30,9%	94,1%	91,8%	91,9%	94,1%
64	56,0%	89,7%	88,2%	91,2%	89,7%

Le nombre d'équipotentielles globales :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	290	60	60	41	60
16	488	297	212	235	297
64	727	585	483	465	585

La somme de distances logiques des équipotentielles :

méthode / nombre d'unités	méthode aléatoire	méthode linéaire	méthode par entrée	méthode par sortie	méthode par hiérarchie
4	610	156	156	118	156
16	13198	5582	5370	5052	5582
64	277492	173412	172974	160710	173412

❄ ❄ ❄

NSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

INPG

Grenoble, le 28 Août 1990

ÉPARTEMENT DES ÉTUDES DOCTORALES

'aire suivie par
l : 76.57.

Réf. :

jet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'Arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

- **Monsieur F.ANDRE**
- **Monsieur G.CAMBON**

Monsieur WU Yang

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de **DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE**, spécialité

"Informatique"

Pour le Président de l'I.N.P. G.
et Par délégation
Le Vice-Président

C. GAUBERT

INPG