N° d'ordre : 699 ANNEE 1986

LIFL U.A. 369 du C.N.R.S.

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

THÈSE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

FLANDRES ARTOIS

pour obtenir

Le titre de DOCTEUR ES SCIENCES

par

Przemyslaw BAKOWSKI

CONTRIBUTION A LA CONCEPTION DE L'ARCHITECTURE MATERIELLE DES ORDINATEURS

Projet LIDO
Préprocesseur pour la compilation en silicium

Exemplaire corrigé après avis du jury

Thèse soutenue le 11 Juillet 1986 devant la Commission d'Examen

Membres du Jury	Président	C. CARREZ
	Rapporteur	V. CORDONNIER
	Examinateurs	G. JOURNEL
		D. ETIEMBLE
		R. MARCZINSKI

UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS
U.F.R. d'I.E.E.A. Bât. M3 - 59655 VILLENEUVE D'ASCQ CEDEX
Tél. 20 43 44 83

P R O F E S S E U R S CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	I.E.E.A.
M. FOURET René	Physique
M. GABILLARD Robert	I.F.E.A.
M. MONTREUIL Jean	Biologie
M. PARREAU Michel	Mathématiques
M. TRIDOT Gabriel	Chimie
M. VIVIER Emile	Biologie
M. WERTHEIMER Raymond	Physique

P R O F E S S E U R S 1ère classe

M. BACCHUS Pierre	Mathématiques
M. BEAUFILS Jean-Pierre (dét.)	Chimie
M. BIAYS Pierre	G.A.S.
M. BILLARD Jean (dét.)	Physique
M. BOILLY Bénoni	Biologie
M. BOIS Pierre	Mathématiques
M. BONNELLE Jean-Pierre	Chimie
M. BOUGHON Pierre	Mathématiques
M. BOURIQUET Robert	Biologie
M. BREZINSKI Claude	I.E.E.A.
M. CELET Paul	Sciences de la Terre
M. CHAMLEY Hervé	Biologie
M. COEURE Gérard	Mathématiques
M. CORDONNIER Vincent	I.E.E.A.
M. DEBOURSE Jean-Pierre	S.E.S.
M. DYMENT Arthur	Mathématiques

PROFESSEURS 1ère classe (suite)

M. ESCAIG Bertrand	Physique
M. FAURE Robert	Mathématiques
M. FOCT Jacques	Chimie
M. GRANELLE Jean-Jacques	S.E.S.
M. GRUSON Laurent	Mathématiques
M. GUILLAUME Jean	Biologie
M. HECTOR Joseph	Mathématiques
M. LABLACHE COMBIER Alain	Chimie
M. LACOSTE Louis	Biologie
M. LAVEINE Jean Pierre	Sciences de la Terre
M. LEHMANN Daniel	Mathématiques
Mme LENOBLE Jacqueline	Physique
M. LHOMME Jean	Chimie
M. LOMBARD Jacques	S.E.S.
M. LOUCHEUX Claude	Chimie
M. LUCQUIN Michel	Chimie
M. MIGEON Michel Recteur à Grenoble	E.U.D.I.L.
M. MIGNOT Fulbert (dét.)	Mathématiques
M. PAQUET Jacques	Sciences de la Terre
M. PROUVOST Jean	Sciences de la Terre
M. ROUSSEAU Jean-Paul	Biologie
M. SALMER Georges	I.E.E.A.
M. SEGUIER Guy	I.E.E.A.
M. SIMON Michel	S.E.S.
M. STANKIEWICZ François	S.E.S.
M. TILLIEU Jacques	Physique
M. VIDAL Pierre	I.E.E.A.
M. ZEYTOUNIAN Radyadour	Mathématiques

P R O F E S S E U R S 2ème classe

M. ANTOINE Philippe	Mathématiques (Calais)
M. BART André	Biologie
Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mathématiques
M. BELLET Jean	Physique
M. BERZIN Robert	Mathématiques
M. BKOUCHE Rudolphe	Mathématiques
M. BODARD Marcel	Biologie
M. BOSQ Denis	Mathématiques
M. BRASSELET Jean-Paul	Mathématiques
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie
M. CARREZ Christian	I.E.E.A.
M. CAYATTE Jean-Louis	S.E.S.
M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
M. CROSNIER Yves	I.E.E.A.
M. CURGY Jean-Jacques	Biologie
Mle DACHARRY Monique	Géographie
M. DAUCHET Max	I.E.E.A.
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DELORME Robert	S.E.S.
M. DE MASSON D'AUTUME Antoine	S.E.S.
M. DEMUNTER Paul	C.U.E.E.P.

PROFESSEURS 2ème classe (Suite 1)

M. DENEL Jacques	I.E.E.A.
M. DE PARIS Jean-Claude	Mathématiques (Calais)
Mle DESSAUX Odile	Chimie
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
Mme DHAINAUT Nicole	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique (Calais)
M. DUBUS Jean-Paul	I.E.E.A.
M. FAKIR Sabah	Mathématiques
M. FONTAINE Hubert	Physique
M. FOUQUART Yves	Physique
M. FRONTIER Serge	Biologie
M. GAMBLIN André	G.A.S.
M. GLORIEUX Pierre	Physique
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel (dét.)	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREGORY Pierre	I.P.A.
M. GREMY Jean-Paul	S.E.S.
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HENRY Jean-Pierre	E.U.D.I.L.
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JEAN Raymond	Biologie
M. JOFFRE Patrick	I.P.A.

PROFESSEURS 2ème classe (suite 2)

M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LANGRAND Claude	Mathématiques
M. LATTEUX Michel	I.E.E.A.
Mme LECLERCQ Ginette	Chimie
M. LEFEVRE Christian	Sciences de la Terre
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques
M. LEMAIRE Jean	Physique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	C.U.E.E.P.
M. LOUAGE Francis(dét.)	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
M. MESSELYN Jean	Physique
M. MESSERLIN Patrick	S.E.S.
M. MONTEL Marc	Physique
Mme MOUNIER Yvonne	Biologie
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RAOULT Jean François	Sciences de la Terre
M. RICHARD Alain	Biologie

PROFESSEURS 2ème Classe (suite 3)

M. RIETSCH François	E.U.D.I.L.
M. ROBINET Jean-Claude	E.U.D.I.L.
M. ROGALSKI Marc	Mathématiques
M. ROY Jean-Claude	Biologie
M. SCHAMPS Joël	Physique
Mme SCHWARZBACH Yvette	Mathématiques
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STAROSWIECKI Marcel	E.U.D.I.L.
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole
Mme TJOTTA Jacqueline (dét.)	Mathématiques
M. TOULOTTE Jean-Marc	I.E.E.A.
M. TURRELL Georges	Chimie
M. VANDORPE Bernard	E.U.D.I.L.
M. VAST Pierre	Chimie
M. VERBERT André	Biologie
M. VERNET Philippe	Biologie
M. WALLART Francis	Chimie
M. WARTEL Michel	Chimie
M. WATERLOT Michel	Sciences de la Terre
Mme ZINN JUSTIN Nicole	Mathématiques

CHARGES DE COURS

M. ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M. BAFCOP Joël	I.P.A.
M. DUVEAU Jacques	S.E.S.
M. HOFLACK Jean	I.P.A.
M. LATOUCHE Serge	S.E.S.
M. MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M. NAVARRE Christian	I.P.A.
M. OPIGEZ Philippe	S.E.S.

A ma femme

Przemyslaw BAKOWSKI

CONTRIBUTION A LA CONCEPTION DE L'ARCHITECTURE MATERIELLE DES ORDINATEURS

Projet LIDO
Préprocesseur pour la compilation en silicium

* * *

A CONTRIBUTION TO THE DESIGN OF COMPLEX COMPUTER ARCHITECTURE

LIDO Project
A Silicon Compiler Preprocessor

REMERCIEMENTS

Je tiens à exprimer toute ma reconnaissance à Monsieur V. Cordonnier, Professeur à l'Université Lille I, qui m'a accueilli dans son Laboratoire de Recherche et qui m'a accordé une confiance exigeante.

Je tiens à remercier:

Monsieur C. Carrez, Professeur à l'Université Lille I, de m'avoir fait l'honneur de présider le jury de cette thèse.

Monsieur D. Etiemble, Professeur à l'Université Paris VI, d'avoir accepté d'être rapporteur de cette thèse et d'avoir bien voulu me faire profiter de ses remarques constructives.

Monsieur G. Journel, Professeur à l'Université Lille I, d'avoir accepté, malgré des responsabilités multiples, de faire partie du jury de cette thèse.

Monsieur R. Marczynski, Professeur à l'Académie des Sciences de Pologne, d'avoir accepté d'être rapporteur de cette thèse et d'avoir franchi les frontières afin d'assister à la soutenance.

Je tiens également à remercier:

Tous mes collègues du Laboratoire d'Informatique Fondamentale qui, par leur soutien et leur aide ont contribué à ce travail.

TABLE DES MATIERES GENERALE

I. INTRODUCTION
1. Modèlisation globale de l'architecture du matériel
2. Description de l'architecture du matériel
3. Analyse et synthèse de l'architecture du matériel

II. LA "FORMATIQUE" DU MATERIEL INFORMATIQUE - Une approche systémique et morphologique de la complexité du matériel informatique
1. Introduction
2. Formes (systèmes) élémentaires
3. Formes (systèmes) de deuxième ordre
4. Formes (systèmes) de troisième ordre
5. Formes (systèmes) de quatrième ordre
6. Remarque finale

III. LE LANGAGE LIDO - Langage Interprétable pour la Description des Ordinateurs
1. Introduction
2. Présentation du langage LIDO
3. Description comparative des processeurs de type RISC
4. Remarques finales

IV. LE SYSTEME LIDO - Préprocesseur pour la compilation en silicium
1. Introduction
2. Les fonctions du système LIDO
3. Construction du système LIDO
4. Les modes de travail et les commandes du système LIDO

V. EN CONCLUSION

VI. BIBLIOGRAPHIE

VII. APPENDICES - La syntaxe du langage LIDO et deux exemples de travail sur système LIDO

I. INTRODUCTION

L'arrivée des systèmes VLSI a bouleversé profondément les méthodes traditionnelles de conception du matériel informatique et a apporté des moyens permettant de réaliser des structures et des fonctions d'un nouvel ordre de complexité. Toute méthodologie et tout outil informatique permettant de maîtriser et de concevoir des systèmes de plus en plus complexes ou évolués (évolué ne doit pas forcément signifier plus complexe) contribuent ainsi au développement du matériel informatique.

Dans ce mémoire nous présentons une approche et une méthodologie de conception dites "systémiques"* qui permettent de concevoir l'architecture du matériel orientée VLSI. L'approche systémique consiste à comprendre et à modéliser les structures matérielles de façon globale en tenant compte du caractère hiérarchique de ces structures. Cette approche implique un certain raisonnement particulier propre à la dualité information ⇔ structure matérielle, étalée sur plusieurs niveaux "systémiques".

L'approche systémique est à la base de la méthodologie mise en oeuvre pour concevoir l'architecture du matériel selon le langage et le système LIDO (Langage Interprétable pour la Description des Ordinateurs). L'ensemble langage-système permet de décrire, d'analyser et de synthétiser les architectures complexes (a fortiori les plus simples) des ordinateurs afin de pouvoir les utiliser à l'entrée d'un compilateur en silicium de type GENESIL.

Ce mémoire est composé de trois parties correspondant aux trois étapes suivantes:

- modélisation globale
- description (langage LIDO)
- analyse et synthèse (système LIDO)

Ces trois étapes étant basées sur les mêmes concepts nous présentons ci-dessous une introduction générale et commune.

** systémique - (angl.systemic) Adj. Qui se rapporte ou affecte un système dans son ensemble. N.f. La systémique, technique des systèmes complexes. LE PETIT ROBERT 1983.*

1. Modélisation de l'architecture du matériel

Il est évident qu'une conception dite raisonnée de l'architecture du matériel doit être établie à partir de certaines "images" ou "formes" générales représentant les traits essentiels de l'aspect structurel et de l'aspect fonctionnel de cette architecture. Ces traits devraient être dépourvus de tous les détails techniques et technologiques intervenant exclusivement de façon quantitative (p.e. nombre de bits traités par UAL ou nombre de mots de mémoire centrale). Ce type de modélisation, bien qu'il nous semble irremplaçable pour la compréhension des phénomènes et des structures du matériel, est relativement rare, ou limité, malgré les efforts considérables faits dans cette direction [Gil83]. Parmi les exemples les plus connus nous trouvons le découpage traditionnel d'un système matériel complexe en partie contrôle et en partie opérative. Cette décomposition enrichie par le facteur quantitatif donne la fameuse classification de Flynn [Fly68]: SISD, SIMD, MIMD et (MISD). Une autre approche relative à la modélisation de la partie contrôle (automate) permet de découper la fonctionnalité de cette partie en fonctions de transition et en fonctions de sortie (automate de Mealy). Prises ensemble les fonctions de transition (séquencement), les fonctions de sortie (décodage) et les fonctions de la partie opérative donnent trois niveaux de complexité fonctionnelle pour un système matériel relativement simple. Dans la même optique on peut facilement ajouter de nouvelles couches fonctionnelles telles que les fonctions de synchronisation (horloge) ou les fonctions de contrôle dans un système multi-processeur.

Il reste quand même à expliquer la nature des liens et des interactions entre les différents niveaux structurels et fonctionnels d'un système complexe. On peut se demander si les signaux de synchronisation, ou encore "signaux" d'alimentation peuvent être considérés comme de l'information. Par ailleurs nous voudrions par exemple savoir si l'introduction de la microprogrammation augmente ou non le nombre de niveaux de complexité d'un système donné.

De telles questions se posent immédiatement si on veut introduire un certain ordre ou établir une classification. Depuis quelques années l'attention d'un certain nombre de chercheurs est attirée par la recherche de concepts relatifs

aux fonctions d'ordre supérieur capables de refléter la complexité opérationnelle du matériel informatique. Ces traveaux portent sur l'aspect global (architecture) [Bou83] et sur l'aspect détaillé (logique) [Han85],[Gor83] de ce matériel. Parallèlement l'efficacité, malgré ses limites, de la classification de Flynn et la validité des concepts proposés par Giloi [Gil83] ("computer architecture is defined by ordred pair <operational principle,hardware structure>") semblent tracer de nouvelles voies vers la compréhension plus cohérente et plus profonde de la nature du matériel informatique.

Notre méthode de modélisation [Bak83], restant entièrement dans le cadre des concepts proposés par Flynn et Giloi, emprunte de plus quelques concepts reconus et largement employés en systémique [Dur81],[Moi78]. Parmi ces derniers celui de systèmes énergétiquement et structurellement ouverts (voir von Bertalanffy, "General System Theory") nous a été très utile.

La méthode proposée dans la première partie du mémoire consiste "en gros" à voir l'architecture des ordinateurs comme superposition de couches structurelles (formes) animées par des flux d'informations. A chaque niveau systémique correspond son niveau d'information. L'influx d'énergie (alimentation) est considéré comme l'information de base indispensable pour l'animation de chaque autre niveau d'information.

La dualité information ⇔ structure (forme) matérielle est un couplage qui permet de résoudre le problème d'interfaçage dynamique entre différents niveaux systémiques. Le passage d'un état d'information d'un niveau superieur à celui du niveau inférieur provoque des modifications structurelles du niveau subordonné. De cette façon à chaque flux d'informations peut être associé un ou plusieurs "flux de structures" subordonnées qui par la suite impliquent à nouveau des flux d'informations, qui eux mêmes provoquent de nouveaux "flux de structures" etc... En réalité, il est difficile de trouver des systèmes complexes dont l'agencement est exclusivement descendant. Dans la plupart des cas on rencontre des systèmes qui disposent de structures impliquant des rétroactions ascendantes qui sont par ailleurs indispensables pour l'enchaînement de plusieurs instructions (éxécution de programmes ou microprogrammes) ou pour des modifications dynamiques concernant l'éxécution d'instructions singulières.

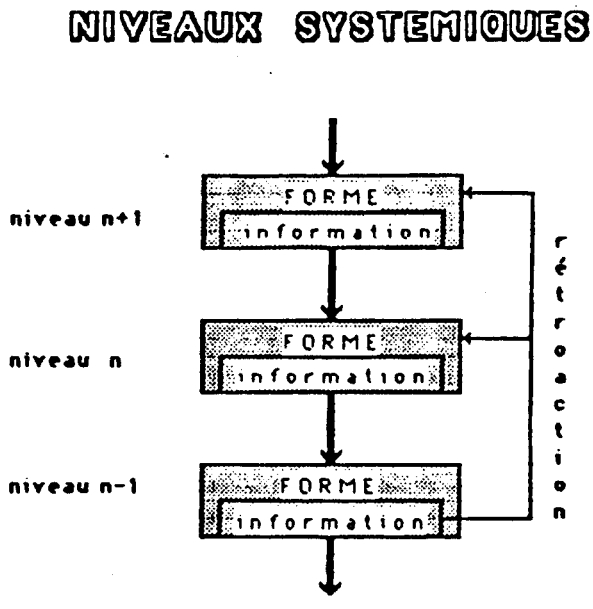


Fig.1.1. La superposition et l'agencement des niveaux systémiques

L'approche systémique offre des moyens de compréhension et de classification pour les différentes architectures matérielles par rapport à leur ordre de complexité interne. Cet ordre correspond au nombre de niveaux d'informations traitées par une architecture donnée.

Remarque terminologique

Afin de rendre les modèles systémiques plus cohérents dans le domaine terminologique nous employons le terme "forme" dans le sens d'une structure matérielle interne acheminant les flux d'informations.
L'opposition classique forme ⇔ matière peut ainsi être remplacée par une autre, mieux adaptée à nos besoins: **forme ⇔ information**.

2. Description de l'architecture du matériel

Les langages pour la description du matériel (angl. Computer Hardware Description Languages - CHDLs) sont aussi vieux que le matériel informatique [Mar79], et plus vieux même que le mot informatique*. La deuxième génération des CHDLs commence par l'introduction des langages CDL [Chu65], DDL [Du168] et AHDL [Hil171], et par la nouvelle possibilité liée à ces notations: la simulation. Parmi les successeurs européens les plus proches nous trouvons RTS I [Pil73], CASSANDRE [Mer71], OSM [Mar75]. Une liste plus détaillée et complètée par des remarques relatives à l'implémentation est donnée par Shiva [Shi79] (voir la page suivante Tab.1.1.).

Une évolution rapide des CHDLs et l'afflux des nouvelles notations ("Hardware description languages: voices from the Tower of Babel" [Lip77]) ont conduit à l'organisation d'un symposium international biennal appelé "Computer Hardware Description Languages and Their Applications"(1971), suivi d'un premier effort important de standardisation des CHDLs, réalisé par le groupe CONLAN [Pil83]. Indépendamment du projet CONLAN nous assistons à la création de nouveaux outils descriptifs de plus en plus variés et spécifiques [Mar79], [Paw81], [Bak83].

L'arrivée de la technologie VLSI, évènement capital pour l'évolution des CHDLs (surtout dans le domaine de la compilation en silicium) a provoqué de nouveaux efforts dont les résultats peuvent influencer sérieusement la conception du matériel informatique [Lieb83],[Dew83]. Dans ce cadre le langage VHDL (VHSIC Hardware Description Language) sa méthodologie et son environment représentent une entreprise sans précédent. Le tableau Tab.1.2. donne les informations caractèrisant les ambitions du projet VHDL par rapport aux quelques langages CHDL les plus connus aux Etats Unis. Dans ce tableau nous apercevons plusieurs paramètres d'un langage CHDL. Nous essaierons d'expliquer les plus signicatifs.

* néologisme introduit en 1962 par Philippe Dreyfus et construit à partir des mots "information" et "automatique" [angl. data processing]

IMPLEMENTATION DETAILS OF HDL's

Language	Reference	Adapted from	Implemented Machine	Implementation Language
ACDL	9	–	–	
AHPL	10-12	APL	CDC-6400	SNOBOL
			DEC-10	Fortran
ALERT	13-14	APL	IBM 7094	–
APDL	15-16	ALGOL	CDC-G20	ALGOL-60
APL	7	–	many	Assembly
APL•DS	17	APL	–	–
CASD	18	PL/1	IBM 360	PL/1
CASSANDRE	19	ALGOL	IBM 360,370	Assembly
CDL	20-29	ALGOL	IBM 370	Fortran Assembly
CSL	30	ALGOL	IBM 370/155	BCPL
DDL	31-39	–	Harris 6024/3	–
DIGITEST II	40	P1/1; petry Nets	–	–
DIDL	41	–	–	–
DSDL	42	DDL	IBM 360	XPL
FLOWWARE	43	flowcharts	IBM 360/50	PL/1
		CDL	NOVA-800	Nova Assembly
FST	44	–	IBM 360	Fortran IV
GLIDE	45	–	–	–
HARGOL	46	ALGOL	–	–
HILO	47	–	ICL 1900	–
ISP				
ISPL	48-54	ALGOL	PDP-10	BLISS
ISPS				
LALSD	55-57	–	IBM 360/91	PL/1
			CDC 6400	SNOBOL
LASCAR	58	CASSANDRE	–	–
LCD	59	PL/1	–	–
LDT	60	RTL	Burroughs	ALGOL-58
LOGAL	61	RTL	UNIVAC 1108	Fortran IV
LOTIS	62	ALGOL	–	–
MDL	63	APL	–	–
MODEL/LINDA	64	–	–	–
OSM	65		ODRA-1305	PLAN
PMS	49	ALGOL	PDP-10	SNOBOL
RTL	6	–	CDC1604	ALGOL
RTS I	66	ALGOL	Siemens 4004/151	Fortran
RTS II	67-68	RTS-I	–	–
RTS III	69	CDL,RTS II	–	–
SDL	70	RTL	–	–
SDL II	71	ALGOL	–	–
SDL	72	–	–	–
SMITE	73	–	CYBER 174	–
SSM	74	–	–	–
VDL	75	–	–	–
V-PMS	76	PMS	–	–

Notes: 1) "–" Indicates that the detail is either not available or not known.
2) No accuracy is claimed for the contents of this table.

Tab.1.1. Les langages "classiques" CHDLs et leurs implémentations

Features supported	IDL	TI-HDL	CDL	AHPL	ZEUS	CON-LAN	TEGAS	ISPS	VHDL
Scope—range of hardware:									
Digital-systems design	x	x		x	x	x	x	x	x
Gate-level design	x	x	x	x	x	x	x	x	x
Digital-circuit design									
Combinational design		x		x		x	x	x	x
Synchronous design	x	x	x	x	x	x	x	x	x
Asynchronous design		x		x		x	x	x	x
Mixed-mode design		x					x		x
Management of design:									
Hierarchy	x	x		x	x	x	x	x	x
Modularity	x	x		x	x	x	x	x	x
Incremental compile		x			x	?		-	x
Arbitrary decompose	x	x		x	x	x	x	x	x
Libraries	x	x		x	x	x	x	x	x
Data abstraction					x	x			x
User type conv									x
Alternate description									x
Reusable designs	x	x		x	x	x	x	x	x
Timing description:									
Timing at all levels	x	x	na	x	x	x	x	x	x
Specify timing data		x		x		x	x	x	x
User-defined data		x		x		x	x		x
Timing constraints		x					x		x
Propagation delay	x	x		x		x	x	x	x
Inertial and transport									x
Architectural description:									
Algorithmic description	x	x		x	x	x	x	x	x
Architectural description	x		x	x	x	x	x	x	x
Parallelism	x		x	x	x	x	x	x	x
Control and data separate	x		x						x
Descriptive continuum					x	x			x
User assertions						x		x	x
Explicit structure		x		x	x	x			
Implicit structure	x	x	x	x	x	x	x	x	x
Generic components		x			x	x	x		x
Regular structures	x	x	x	x	x	x	x	x	x
Recursive structures				x	x	x			
Interface description:									
Explicit interface		x		x	x	x	x	x	x
Defined at all levels	na	x	na	x	x	x	x	x	x
Supports functional equiv									x
Strongly typed interface	na				x	x			x
Design environment:									
Environment info								x	x
Language extensibility:									
User-defined data types					x	x			x
Design tool support						x		x	x
Multiple technologies	x	x	x	x	x	x	x	x	x
Multiple methodologies	x	x		x		x		x	x

na—Not applicable: This feature is based upon another feature which the language does not support.
? —We could not determine the language support for this feature.

Tab.1.2. Les paramètres du langage VHDL

Niveaux de description (scope - range of hardware)

Etant donné que les ordinateurs peuvent être conçus sur différents niveaux d'abstraction les langages CHDL peuvent être classés par rapport au niveau principal visé par leur utilisation:

- niveau des instructions
- niveau du transfert de registres
- niveau logique

Tout de suite il faut souligner qu'il n'existe pas de notation limitée exclusivement au niveau des instructions ou du transfert de registres. Un tel langage serait peu utile pour la description complète des fonctions et/ou des structures du matériel reél.

Types de description

Traditionnellement on distingue deux types de description [Bar75]:

- description procédurale
- description non-procédurale

Une description procédurale définit le fonctionnement d'une architecture du matériel de façon algorithmique. L'enchaînement des opérations y est indiqué par l'ordre typographique de la description. Les langages dits procéduraux sont dans la plupart des cas derivés de langages de programmation (p.e. AHDL), ou prévus pour la description du matériel au niveau des instructions (p.e. ISPS, LASCAR). Les langages non-procéduraux permettent de décrire un système matériel de façon typographiquement libre. Les liens entre les objets décrits sont établis à partir d'étiquettes d'activation (reste à résoudre le problème des opérations asynchronnes dont le déroulement réél est impliqué par l'ordre topologique). Ce type de description est préféré pour la spécification proche de la structure

matérielle des ordinateurs (p.e. CDL, RTS I, CASSANDRE, OSM ...), ou pour la spécification des circuits au niveau logique (p.e. LALSD).

Aspects de description

Indépendamment du niveau de description ou du type de description un langage CHDL peut être adapté à un ou aux deux aspects de description:

- aspect fonctionnel
- aspect structurel

L'aspect fonctionnel est lié aux fonctions et aux opérations du matériel vues à partir des différents niveaux de conception. L'aspect structurel concerne la description des structures qui supportent différentes fonctions d'un système (p.e. registres, connecteurs, bus, portes logiques). Les structures décrites en langage de haut niveau de conception peuvent être dépourvues de fonctionnalités précises et interprétables, tel est le cas de la notation P-M-S (Processor-Memory-Switch) [Bell71], qui représente exclusivement des liens structurels entre les unités globales d'une organisation matérielle. L'aspect structurel est plus souvent lié aux spécifications de type non-procédural concernant le niveau transfert de registres et le niveau logique.

VLSI et CHDLs

L'arrivée des systèmes VLSI, intégrant dans un seul composant des dizaines ou des centaines de milliers d'éléments actifs, ouvre une nouvelle époque pour les CHDLs.
La nécessité d'une vérification fonctionnelle détaillée d'un système integré avant sa fabrication fait appel a des moyens symboliques tels que les langages CHDLs. Au même titre les langages pour la description deviennent des langages pour la conception du matériel intégré. Leur utilisation est obligatoire dans le

processus de synthèse automatique liée aux problèmes de compilation en silicium. Il existe deux manières d'attaquer ces nouveaux problèmes. La première consiste à adapter les anciennes notations en ajoutant de nouveaux niveaux de conception (p.e. niveau topologique). La deuxième solution c'est le développement d'une notation originale exploitant l'expérience accumulée par les concepteurs des systèmes VLSI. La première méthode à été utilisée par un certain nombre de chercheurs disposant d'une base logicielle (langages, simulateurs) datant des années soixantes et soixante-dix. Un des développements de cette méthode est l'extension du langage LALSD II vers un système multiniveaux permettant la simulation au niveau transfert de registres et au niveau logique [Su85]. Une autre extension multiniveaux beaucoup plus vaste est représentée par le système CASCADE [Bor85] dont les origines peuvent être facilement retrouvées dans les langages CASSANDRE et LASCAR (voir le tableau 1.1.). Parmi les développements plus récents, au premier plan se trouve le langage VHDL, au deuxième le langage ZEUS. Il faut toutefois souligner que l'arrivée de compilateurs en silicium, tels que GENESIL [Sil85], réalisés par les spécialistes de circuits integrés impose une nouvelle approche de la conception du matériel integré qui consiste à générer les descriptions topologiques directement à partir du niveau transfert de registres sans passer par le niveau logique et le niveau circuit.

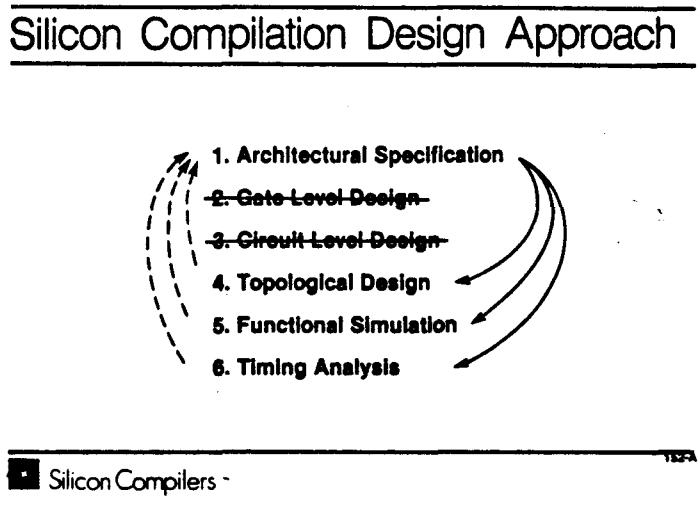


Fig.1.2. L'approche "directe" de la compilation en silicium utilisée dans le compilateur GENESIL.

Argument principal utilisé en faveur de cette approche est le suivant:

> **A Typical Function Built from Gates or Unstructured Transistors Takes 2 to 10 Times More Transistors and Area than a Structured Design**

Notre proposition s'aligne sur cette dernière tendance et par ailleurs trouve ses origines dans la suite des langages CHDLs issue de la notation CDL. Cette suite de développements est illustrée ci-dessous.

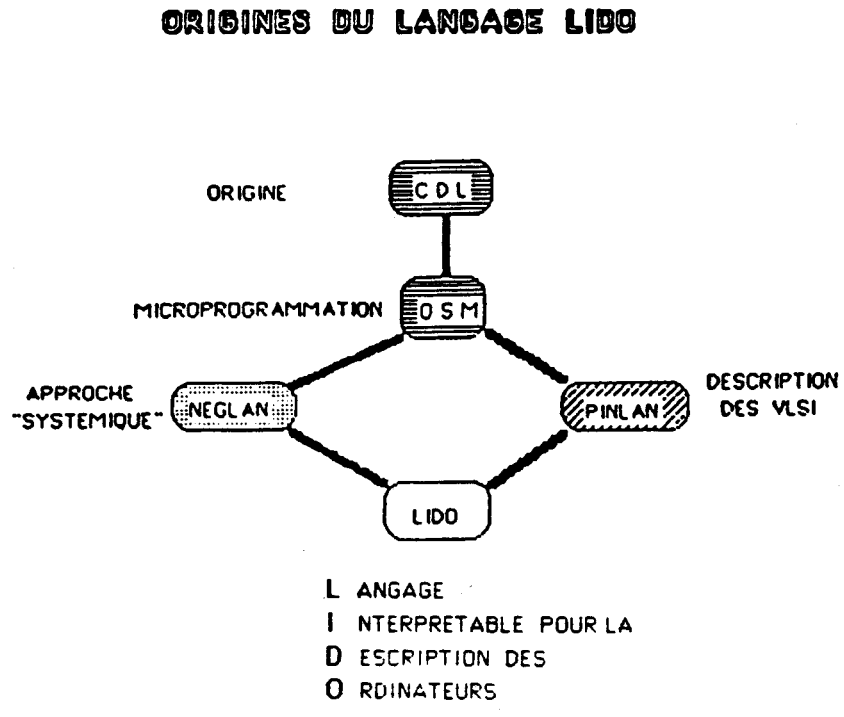


Fig.1.3. Les origines du langage LIDO (Langage Interprétable pour la Description des Ordinateurs).

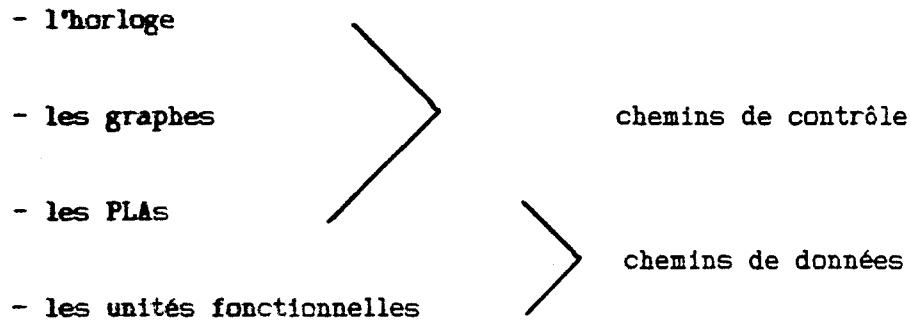
Le langage CDL est reconu comme prototype des langages RTL non-procéduraux [Chu65]. L'un de ses descendants, le langage OSM [Mar75] à été développé à l'Académie Polonaise des Sciences afin de modéliser de gros ordinateurs au contrôle microprogrammé (p.e. IBM 370/158). Les descriptions en langage OSM sont simulées par les systèmes de simulation 3SM [Mar75] et 3SM-K [Bak78]. Par rapport au CDL, le langage OSM a une structure modulaire et se caractèrise par l'utilisation intensive de graphes de contrôle. Le développement du langage NEGLAN [Bak83] marque la première étape vers la description systémique du matériel informatique. Son but principal est de décrire le fonctionnement du matériel de façon "libre" et naturellement active. Cette possibilité est liée à l'introduction du contrôle négatif qui permet d'inhiber les opérations "normalement actives". Le langage NEGLAN est simulé par le système MAMAYA implémenté par les étudiants de l'Ecole Polytechnique de Varsovie (implémentation en PASCAL). Une autre forme de développement a été exprimée par le langage PINLAN [Mar79] destiné à la description comportementale (entrées ⇔ sorties, PINS) des systèmes LSI.
La richesse des résultats et des expériences acquises pendant le développement et l'implémentation des langages cités a permis de concevoir et de réaliser le langage et le système LIDO, qui font respectivement l'objet de la deuxième et de la troisième partie de ce mémoire. Ici nous nous limiterons à quelques remarques essentielles concernant le langage LIDO qui permet:

- la description systémique du matériel
- la description orientée VLSI
- la description des architectures évoluées

Description systémique

Le langage LIDO permet de décrire le matériel informatique de façon largement non-procédurale et basée sur le concept de systèmes ouverts emprunté à la systémique. Dans ce cadre conceptuel le matériel à décrire est considéré comme l'ensemble des objets "normalement actifs" ou "normalement inactifs". De plus on impose une hiérarchie fonctionnelle et structurelle découpant des systèmes

complexes en plusieurs couches, chacune correspondant à un niveau d'informations traitées par le système. Les couches (strata) de caractère global sont exprimées par différentes constructions telles que:



A l'exclusion de l'horloge, qui est un mécanisme unique pour chaque description, les autres mécanismes sont multiples et étalés sur plusieurs niveaux locaux. De cette façon on dispose d'"arrays" de graphes, d'"arrays" de matrices logiques de différentes sortes, et d'"arrays" d'unités fonctionnelles.
La communication entre différents niveaux systémiques et à l'interieur d'un même niveau peut se faire à l'aide de différents types de porteurs d'informations tels que:

- les connecteurs
- les terminaux
- les registres

Les connecteurs permettent de définir la communication asynchronne (endochronne), les registres la communication synchronne et/ou contrôlée, les terminaux peuvent participer dans les deux types de communication.
Tous ces moyens pris ensemble donnent au concepteur de matériel la possibilité de "bijection" entre la description d'un système et le système lui même de telle manière qu'à chaque unité décrite en langage LIDO correspond une unité topologique d'un système réél. De plus, chaque description peut être modifiée

(recablée) ou étendue d'une manière totalement libre, correspondant à la manière dont on recompose ou augmente une structure matérielle réelle.

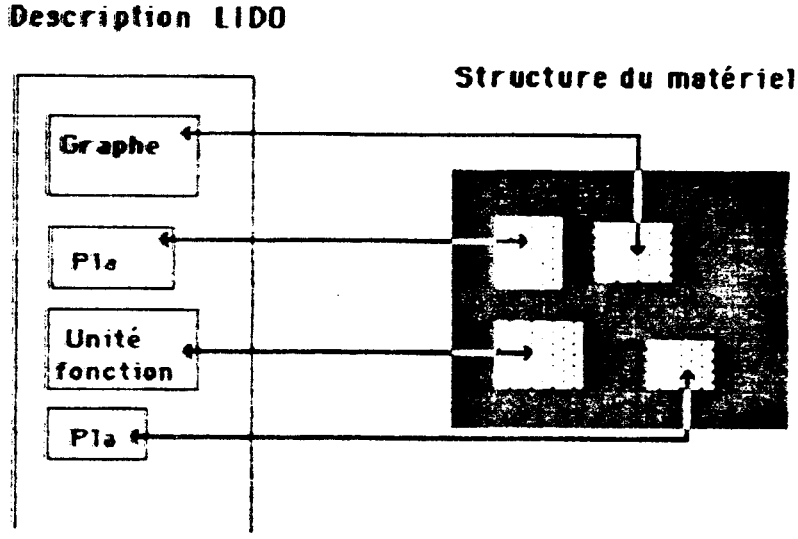


Fig.1.4. La "bijection" entre une description en langage LIDO et la structure materielle réelle.

Description orientée VLSI

L'utilisation du langage pour la conception de l'architecture moderne dans le cadre de la compilation en silicium oblige à introduire des mécanismes correspondant aux sous-systèmes topologiques des systèmes VLSI. Parmi ces mécanismes on trouve toutes les structures géométriquement régulières, telles que les mémoires ROM, RAM, les matrices PLA, SLA et MGA. Le langage LIDO offre ces structures comme constructions de base.

Description des architectures évoluées

La comlexité des systèmes VLSI et bientôt des systèmes ULSI dépassa largement la complexité d'une UAL (l'exemple usé d'echantillonage de langages CHDLs) ou d'un processeur simple de type von Neumann. Les systèmes à plusieurs unités de traitement, multiprocesseurs, architecture pipe-line et data-flow, deviennent accessibles aux concepteurs de circuits intégrés.
Un langage qui suit, ou mieux, anticipe de nouvelles architectures, doit permettre leur description. L'utilisation de mécanismes multiples ("arrays" de mécanismes - graphes, matrices logiques, unités fonctionnelles) en langage LIDO assure cette possibilité. De plus, l'introduction de mémoires associatives comme porteurs primitifs d'informations facilite les spécifications des machines disposant de mémoires virtuelles et des machines data-flow dont l'unité "matching unit" nécessite une recherche associative des arguments prêts à éxécuter.

3. L'analyse et la synthèse de l'architecture du matériel

Les modèles et ensuite les langages pour la description de l'architecture des ordinateurs constituent les points de départ pour l'analyse et la synthèse du matériel informatique. L'analyse fonctionnelle du matériel décrit en CHDLs représente l'un des principaux objectifs pour la conception de nouvelles architectures. Malgré le nombre important de langages CHDL le nombre d'implémentations (compilateurs, simulateurs, interpréteurs) est relativement limité. Parmi les implémentations les plus connues on trouve celles des langages CDL [Bara75], DDL [Diet75], AHPL [Swa77], ISPS [Bar81], CASSANDRE [Mer73], OSM [Mar75], RTS I [Pil74], et plus récamment LALSD II [Su85], LASCAR [Bor78], HSL-FX [Hos85].
La grande majorité des simulateurs est basée sur la compilation du langage source, ou sur l'interprétation d'une forme intermédiaire, précompilée, de description. Presque tous ont été conçus avant l'époque des systèmes VLSI.

L'arrivée de nouvelles possibilités techniques liées à l'intégration à très haute échelle, a imposé aux concepteurs des CHDLs de nouveaux objectifs et de nouvelles contraintes. Un de ces objectifs est la synthèse automatisée ou automatique des circuits VLSI à partir d'une description architecturale. Cette synthèse, qui implique une nouvelle dimension à savoir la topologie, est radicalement différente de celle ancienne limitée à la transformation architecture → portes logiques. Malgré ce changement d'objectif la plupart des systèmes visés pour le développement du matériel intégré restent dans le cadre traditionnel et sont complétés par deux niveaux de conception : circuit et topologie.

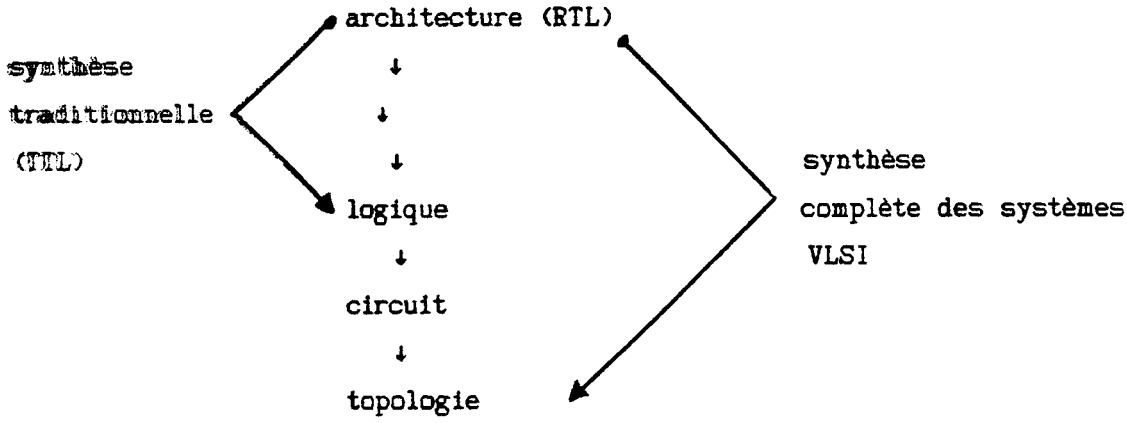


Fig.1.5. La synthèse traditionnelle et moderne du matériel informatique.

Les systèmes CASCADE, VLSI-DE, HSL-FH peuvent servir d'exemples. Une stratégie alternative a été adoptée et réalisée par les concepteurs du système GENESIL. Elle consiste à générer la topologie d'un système VLSI directement à partir d'une description détaillée au niveau transfert de registres. La "compilation directe" des descriptions détaillées RTL a permis de simplifier le processus de synthèse en évitant une grande partie des problèmes liés à l'interfaçage entre différents niveaux de conception. Il est quand même nécessaire de souligner que

la description initiale d'un circuit à compiler doit être très précise et détaillée.

description structurelle, détaillée
au niveau transfert de registres

↓

↓

COMPILATION en SILICIUM

↓

↓

sous-systèmes topologiques d'un système VLSI

Fig.1.6. La compilation "directe" d'une description RTL en silicium.

Ci-dessous nous présentons une liste d'objets (fonctions) utilisés à l'entrée du GENESIL (Genesil Function Set).

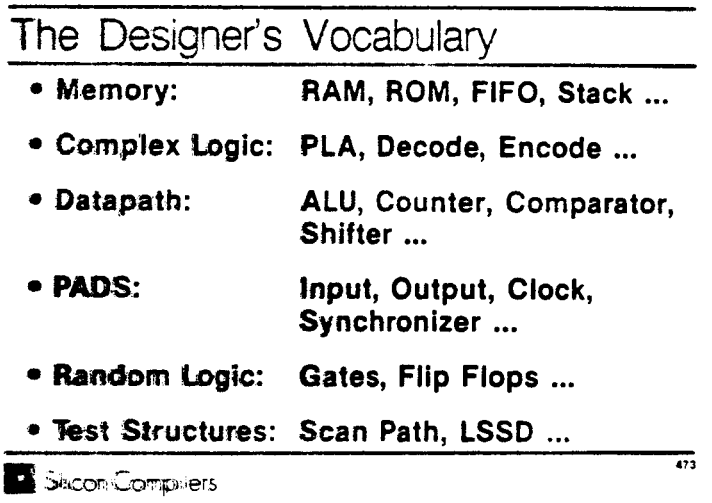


Fig.1.7. La liste d'objets (fonctions) disponibles à l'entrée du GENESIL

la description initiale d'un circuit à compiler doit être très précise et détaillée.

description structurelle, détaillée
au niveau transfert de registres

↓
↓

COMPILATION en SILICIUM

↓
↓

sous-systèmes topologiques d'un système VLSI

Fig.1.6. La compilation "directe" d'une description RTL en silicium.

Ci-dessous nous présentons une liste d'objets (fonctions) utilisés à l'entrée du GENESIL (Genesil Function Set).

Fig.1.7. La liste d'objets (fonctions) disponibles à l'entrée du GENESIL

La nécessité, pour un compilateur en silicium de type GENESIL, de fournir une description structurelle et détaillée au niveau transfert de registres, restreinte aux primitifs fortement typés, limite de façon significative l'espace et la flexibilité de conception offerts à l'utilisateur. D'où la nécessité, soulignée même par les réalisateurs de compilateur GENESIL, d'un préprocesseur travaillant à partir d'une spécification textuelle et assurant un grand espace de conception.
Le système LIDO a été conçu afin de remplir cette tâche. Il permet de concevoir une vaste gamme d'architectures incluant des machines SISD, SIMD, MIMD, pipeline et data-flow, orientées vers les systèmes VLSI.
Les trois rôles principaux du système LIDO sont les suivants:

- interprétation *symbolique* (analyse symbolique)
- interprétation *"profonde"* (analyse détaillée)
- synthèse automatique des chemins de contrôle

L'interprétation symbolique, dans le contexte systémique, consiste à générer des traces d'activation des chemins de données ("flux des structures") sans la visualisation des contenus des porteurs d'informations.
Ce style d'interprétation libère le concepteur de l'analyse détaillée des résultats du traitement des données [Mil85]. L'interprétation "profonde", seule possibilité de travail envisagée dans les simulateurs traditionnels (CDL, CASSANDRE, OSM,...), implique l'éxécution détaillée des opérations activées pendant les processus d'interprétation (simulation), ce qui permet, mais en même temps rend obligatoire, la génération de traces d'interprétation pour différents jeux de données. Il nous semble important de souligner, que seul le choix de l'interprétation directe", mode de travail du système LIDO, permet de résoudre de façon simple le problème d'accès à la description source, qui est indispensable dans le processus de l'interprétation symbolique.
La synthèse automatique des chemins de contrôle consiste à transformer leur description fonctionnelle exprimée par les graphes de contrôle en description structurelle exprimée par les matrices logiques. Etant donné que les graphes et les matrices logiques appartiennent aux constructions de base du langage LIDO,

les transformations graphes → matrices logiques préservent l'intégrité des descriptions, ce qui assure la possibilité d'interprétation des descriptions structurelles et des descriptions fonctionnelles. La synthèse automatique des descriptions structurelles optimisées est l'objectif principal visé dans le développement des futurs versions du système LIDO.

L'un de nos soucis principaux, lors de l'élaboration du système LIDO, a été d'offrir à l'utilisateur un outil simple et intégré, dont l'apprentissage soit une question de quelques heures. D'un autre côté nous voulions utiliser de la façon la plus large possible l'approche systémique, qui à notre avis, peut faciliter la tâche de compréhension et de création d'architectures matérielles de plus en plus complexes.

Une réflexion sur l'approche systémique

Il nous semble nécessaire de remarquer que l'approche systémique ne doit pas être considerée comme une "science", une "thèorie" ou une "discipline", mais comme une méthodologie permettant d'englober la totalité des éléments du système étudié, ainsi que leurs interactions et leurs interdépendances. L'approche systémique n'a rien avoir avec une approche systématique, qui consiste à aborder un problème ou à effectuer une série d'actions de manière séquentielle et détaillée.
Il faut également souligner que l'approche systémique et le terme systémique utilisés dans notre mémoire n'impliquent ni de nouveaux éléments ni de nouvelles architectures, ni même de nouveaux domaines de recherche. Ils permettent exclusivement d'organiser notre connaissance et d'exprimer notre point de vue de façon plus efficace et mieux adaptée à notre but principal, qui est la maîtrise de la complexité fonctionnelle et structurelle du matériel informatique.

LA FORMATIQUE DU MATERIEL INFORMATIQUE

Une approche systémique et morphologique

de l'architecture des systémes matériels

Cette partie du mémoire a été présentée pendant la conference: IFIP WG10.1 Working Conference on Methodologies for Computer System Design, Lille, 15-17 Septembre 1983

TABLE DES MATIERES

1. Introduction
2. Formes élémentaires - composants de premier ordre
   2.1. Composants transitifs - ^1F_T
   2.2. Composants mémorisants - ^1F_M
   2.3. Composants générateurs - ^1F_G
   2.4. Composants mémorisants-transitifs - $^1F_{\langle MT \rangle}$
3. Formes de deuxième ordre
   3.1. Formes transitives - $^2F_{2T}$
   3.2. Formes transitives-mémorisantes - $^2F_{TM}$
   3.3. Formes transitives/mémorisantes-transitives - $^2F_{T\langle MT \rangle}$
   3.4. Formes génératrices
   3.5. Formes transitives-génératrices/mémorisantes-transitives - $^2F_{\langle TG \rangle \langle MT \rangle}$
4. Formes de troisième ordre
   4.1. Formes $^3F_{2G\langle MT \rangle}$
   4.2. Formes $^3F_{2GT}$
   4.3. Formes $^3F_{G2\langle MT \rangle}$
   4.4. Formes $^3F_{G\langle MT \rangle T}$
   4.5. Formes $^3F_{GT\langle MT \rangle}$
   4.6. Formes à rétroaction $^3F_{G2\langle MT \rangle}$ et $^3F_{GT\langle MT \rangle}$
5. Formes de quatrième ordre
   5.1. Formes $^4F_{2G\langle MT \rangle T}$
   5.2. Formes $^4F_{2GT\langle MT \rangle}$
   5.3. Formes $^4F_{2G2\langle MT \rangle}$
   5.4. Formes $^4F_{G\langle MT \rangle T\langle MT \rangle}$
6. Formes de cinquième ordre
   6.1. Formes $^5F_{2G2T\langle MT \rangle}$
   6.2. Formes $^5F_{\langle GT \rangle \langle MT \rangle 2T\langle MT \rangle}$
7. Remarque finale

II. APPROCHE SYSTEMIQUE ET "MORPHOLOGIQUE" DE LA COMPLEXITE DE L'ARCHITECTURE MATERIELLE DES ORDINATEURS

1. Introduction

Le chapitre présent est consacré à la modélisation globale et systémique de l'architecture matérielle. Cette approche orientée vers le problème de la complexité fonctionnelle et structurelle du matériel nous donne la base conceptuelle utilisée par la suite pour la description, l'analyse et la synthèse des architectures detaillées. A côté de termes tels que : fonction, structure, système, et hiérarchie, nous utilisons très fréquemment les termes "forme" et "sélection". La forme signifie pour nous la structure matérielle qui conduit, mémorise ou "trans-forme" l'"in-form-ation" présente à son entrée. Ces trois fonctions se réalisent, dans notre approche, à l'aide d'une seule opération : la sélection.

La "super-position" des formes simples est étroitement liée à la superposition ou à la hiérarchisation des différentes sortes d'information (l'alimentation, les données, les adresses, les signaux de contrôle, les signaux de synchronisation). D'où les notions de "complexité hiérarchisée", de "formes d'ordre supérieur" et de "niveaux systémiques".

Dans ce chapitre, à partir de formes simples nous établirons plusieurs modèles complexes de l'architecture du matériel. Toutefois, ces modèles réprésentent les structures et les fonctionnements des systèmes digitaux de façon largement globale sans donner leur interprétation détaillée. La spécification détaillée des mêmes modèles globaux fera l'objet du chapitre suivant dans lequel nous présenterons le langage LIDO et ses applications. Ici nous allons d'abord présenter les formes ou composants élémentaires du matériel informatique. Ces formes de base transforment, mémorisent et génèrent des informations de nature différente: données, adresses, codes opératifs, phases, etc...

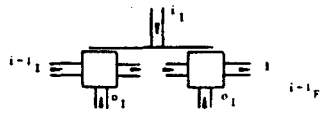
Chaque sous-chapitre est successivement consacré à la présentation de systèmes à deux, trois, quatre puis cinq niveaux de complexité systémique. Les systèmes à

plusieurs niveaux de complexité peuvent exécuter un grand nombre de processus (instructions) basés sur un même ensemble de formes de traitement. Les systèmes (formes d'ordre supérieur à un) choisis comme exemples sont représentatifs des principaux schémas fonctionnels et structurels du matériel informatique.

NOTATIONS

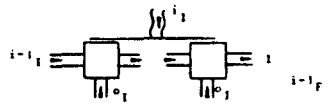
Sélection positive globale (contrôle positif) :

$i_I \Rightarrow\!| \; {}^{i-1}I \Rightarrow {}^{i-1}F \Rightarrow I$



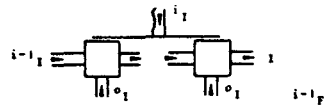
Sélection négative globale (contrôle négatif) :

$i_I \Rightarrow\!| \; {}^{i-1}I \Rightarrow {}^{i-1}F \Rightarrow I$



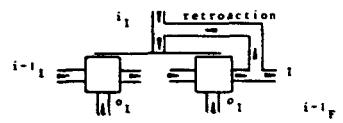
Sélection positive/négative globale (contrôle positif/négatif) :

$i_I \Rightarrow\!| \; {}^{i-1}I \Rightarrow {}^{i-1}F \Rightarrow I$



Rétroaction du niveau i-1 vers le niveau i :

$i_I \Rightarrow\!| \; {}^{i-1}I \Rightarrow {}^{i-1}F \Rightarrow I$



^0I - Information globale d'ordre zéro (l'influx d'énergie)

^iI - Information globale d'ordre i

^0F - Forme globale d'ordre zéro (connecteurs)

^iF - Forme globale d'ordre i

$j/^iI$ - Information d'ordre global i et d'ordre local j

$j/^iF$ - Forme d'ordre global i et d'ordre local j

$i_I \Rightarrow\!| \; {}^{i-1}I \Rightarrow {}^{i-1}F \Rightarrow I \quad \Rightarrow \quad {}^iI \Rightarrow {}^iF \Rightarrow I$

Emergence de forme d'ordre de complexité supérieur

2. Formes élémentaires - composants de premier ordre

Les processus et les systèmes que nous allons étudier sont composés de différents types de composants ou formes élémentaires. Tous ces composants sont considérés comme des éléments de premier ordre fonctionnel et structurel, même si dans leur réalisation il est possible d'utiliser des fonctions et des circuits logiques d'ordre supérieur à un. Cela signifie que l'on distingue deux ordres d'informations : l'information d'ordre zéro - ^{0}I (influx d'énergie), et l'information d'ordre un - ^{1}I, traitée par les composants. Le caractère global de ces composants signifie d'autre part que nous allons considérer l'information qui entre et qui sort des composants comme une information codée le plus souvent sur plusieurs bits.

Il y a trois types de composants :

- formes élémentaires transitives - composants transitifs,
- formes élémentaires mémorisantes - composants mémorisants,
- formes élémentaires génératrices - composants générateurs.

2.1. Composants transitifs - $^{1}F_{T}$

Les composants transitifs sont des formes de premier ordre qui transforment l'information à l'entrée - ^{1}I par l'intermédiaire de l'information d'ordre zéro - ^{0}I en information à la sortie - I_S

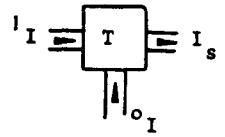
Représentation textuelle :

$^{1}F_{T}[^{1}I, ^{0}I] = {}^{1}I \nRightarrow {}^{0}I \Rightarrow {}^{0}F \Rightarrow I_S$

après émergence :

$^{1}F_{T}[^{1}I] = {}^{1}I \Rightarrow F_T \Rightarrow I_S$ avec $^{1}I \neq I_S$

Représentation graphique :



Interprétation :

Les composants transitifs représentent une grande classe de

fonctions et de circuits combinatoires utilisés pour la transformation de l'information numérique et non-numérique. Parmi eux nous pouvons distinguer des structures spécialisées comme les additionneurs et les comparateurs, et des structures universelles comme les"mémoires" ROM et les matrices logiques PLA. Les structures de l'additionneur et du comparateur exécutent une seule fonction. Ce sont les formes irrégulières et non redontantes qui "performent" les fonctions régulières. Par contre les structures ROM et PLA sont destinées à exécuter une fonction parmi plusieurs possibles. Le choix des fonctions est limité par les dimensions des formes régulières, mais la fonction choisie peut être totalement irrégulière.

2.2. Composants mémorisants - $^{1}F_{M}$

Les composants mémorisants sont des formes de premier ordre qui mémorisent l'information introduite à l'entrée - ^{1}I par l'intermédiaire de l'influx d'énergie qu'est l'information d'ordre zéro - ^{0}I. Les composants mémorisants ne transforment pas l'information, ils se contentent de la conserver. Cette conservation est due aux boucles rétroactives positives alimentées par l'influx d'énergie.

Représentation textuelle :

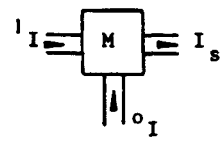
$$^{1}F_{M}[^{1}I,^{0}I] = {}^{1}I \Rightarrow {}^{0}I \Rightarrow {}^{0}F \Rightarrow I_S$$

(rétroaction)

après émergence :

$$^{1}F_{M}[^{1}I] = {}^{1}I \Rightarrow F_M \Rightarrow I_S \qquad \text{avec } {}^{1}I = I_S$$

Représentation graphique :



Interprétation :

Les composants mémorisants constituent une grande classe d'éléments-mémoire dans lesquels on enregistre une information soit à un bit, soit à plusieurs bits. Comme composants mémorisants existent les bascules et les registres, les deux étant des formes de premier ordre. Les contenus des registres et des bascules représentent les états de l'information, c'est-à-dire les états de fonctionnement du système informatique.

2.3. Composants générateurs - $^{1}F_{G}$

Les composants générateurs sont des formes de premier ordre qui génèrent l'information exclusivement à partir de l'influx d'énergie (ou information d'ordre zéro - ^{0}I). La génération de l'information est due aux boucles rétroactives négatives. Le délai de la rétroaction implique la fréquence des oscillations caractéristique d'un générateur donné.

Représentation textuelle :

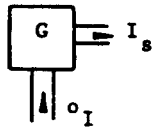
$^{1}F_{G}[^{0}I] = \Rightarrow {}^{0}I \Rightarrow {}^{0}F \Rightarrow I_{S}$ (rétroaction)

$\Rightarrow$ - sélection négative (contrôle négatif)

après émergence :

$^{1}F_{G} \Rightarrow I_{S}$

Représentation graphique :



Interprétation :

Tous les circuits générateurs de signaux de synchronisation, autrement dit tous les circuits générateurs d'impulsions de cadencement, sont des composants générateurs. Ils contiennent des boucles de rétroaction négatives alimentées par un influx d'énergie. La fréquence des oscillations

dépend soit des capacités et des résistances internes des circuits, soit d'éléments spéciaux comme le quartz.

2.4. Composants mémorisants-transitifs - $^{1}F_{MT}$

Les connections des composants mémorisants-transitifs forment la base des transformations de l'information mémorisée. Ce type de transformation est largement utilisé dans les systèmes informatiques car il permet d'enchaîner en séquences plusieurs fonctions, ce qui donne les processus. Les connections des composants mémorisants et transitifs se retrouvent sur tous les niveaux de complexité qui possèdent leurs propres états de fonctionnement.

Représentation textuelle :

$$^{1}F_{MT}[^{1}I, ^{0}I] = {}^{1}I \Rightarrow| \; {}^{0}I \Rightarrow {}^{0}F \Rightarrow {}^{1}I \Rightarrow| \; {}^{0}I \Rightarrow {}^{0}F \Rightarrow I \Rightarrow| .. \Rightarrow I_S$$

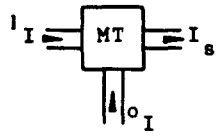
après émergence :

$$^{1}F_{MT}[^{1}I] = {}^{1}I \Rightarrow F_T \Rightarrow {}^{1}I \Rightarrow F_M \Rightarrow .. \Rightarrow I_S$$

après contraction :

$$^{1}F_{MT}[^{1}I] = {}^{1}I \Rightarrow F_{MT} \Rightarrow I_S$$

Représentation graphique :



Interprétation :

Les connections des formes mémorisantes-transitives constituent des ensembles opérateurs qui transforment et mémorisent l'information. Au niveau d'une horloge multiphase les composants sont utilisés pour l'enchaînement et la mémorisation de plusieurs phases dans un cycle d'horloge. Au niveau d'un séquenceur les composants mémorisants et transitifs sont connectés afin de produire différentes séquences de signaux. Enfin, au niveau terminal de traitement les composants mémorisants-transitifs enchaînent les flux de données traitées de façon séquentielle.

3. Les formes de deuxième ordre

Les formes de deuxième ordre de complexité possèdent deux niveaux systémiques : le contenu, état fonctionnel de niveau supérieur qui par les opérateurs-sélecteurs définit la structure de niveau inférieur. Ces deux niveaux systémiques sont construits à partir de différents types de composants, ce qui donne différents systèmes de deuxième ordre qui "performent" différents types de prosessus.

3.1. Formes transitives de deuxième ordre - $^{2}F_{TT}$

Dans les formes transitives de deuxième ordre les composants transitifs sont superposés. On obtient ainsi deux niveaux systémiques de complexité et de multiplexité. Le niveau supérieur est géré par l'information d'ordre global deux - ^{2}I, le niveau inférieur par l'information d'ordre global un - ^{1}I.

a)Représentation textuelle après émergence (suppression de l'information-énergie ^{0}I) :

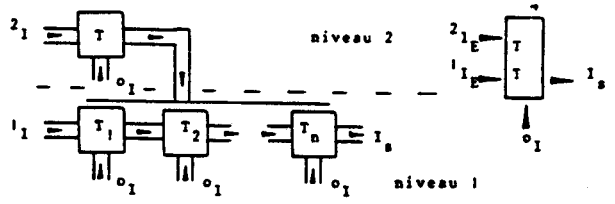
$$^{2}F_{TT}[^{2}I,^{1}I] = {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I \Rightarrow| {}^{1}I \Rightarrow {}^{1}F_{T} \Rightarrow I_{S}$$

b)Représentation textuelle avec macro-sélection (contrôle) négative et positive ($\Rightarrow|$) :

$$^{2}F_{TT}[^{2}I,^{1}I] \Rightarrow {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I \Rightarrow| {}^{1}I \Rightarrow {}^{1}F_{T} \Rightarrow I_{S}$$

La formule a ne concerne que la sélection "positive", c'est-à-dire celle dont les valeurs d'information de deuxième ordre spécifient les formes de premier ordre à sélectionner. Losqu'on veut utiliser la sélection "négative" il est nécessaire d'introduire la formule b qui comporte des sélecteurs négatifs.

Représentation graphique :



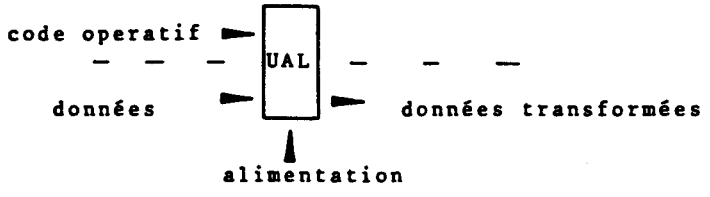
où α - ensemble des opérateurs de sélection positifs et/ou négatifs.

Interprétation :

Les formes transitives de deuxième ordre constituent une classe de circuits combinatoires complexes qui éxécutent différentes fonctions choisies par le niveau supérieur de complexité. Simultanément peuvent être exécutées plusieurs fonctions parallèles ou enchaînées. Le parallèlisme, ou consécutivité des fonctions exécutées, dépend de la disposition spatiale des structures sélectionnées.

Parmi les formes transitives de deuxième ordre les plus connues nous trouvons les unités arithmétiques et logiques dans lesquelles le même ensemble de composants transitifs est utilisé pour exécuter plusieurs fonctions différentes. Chaque unité fonctionnelle qui ne contient pas de composant mémorisant et qui "donne" plusieurs fonctions, constitue une structure transitive de deuxième ordre.

Unité arithmétique et logique :



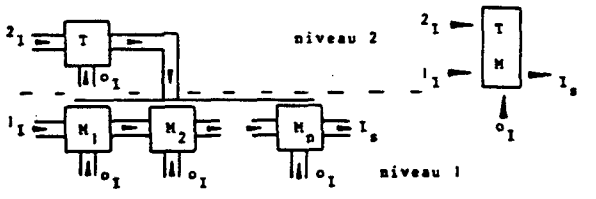
3.2. Formes transitives/mémorisantes de deuxième ordre - $^2F_{TM}$

Les formes $^2F_{TM}$ de deuxième ordre constituent de "vraies" mémoires dans lesquelles on peut stocker et extraire de l'information de premier ordre - ^1I. Les composants mémorisants forment le niveau inférieur de complexité, les composants transitifs le niveau supérieur. L'information de niveau supérieur - ^2I - sélectionne les composants mémorisants de niveau inférieur.

Représentation textuelle :

$$^2F_{TM}[^2I, ^1I] = {}^{1/2}I \Rightarrow {}^{1/2}F_T \Rightarrow {}^2I \Rightarrow {}^1I \Rightarrow {}^1F_M \Rightarrow I_S$$

Représentation graphique :



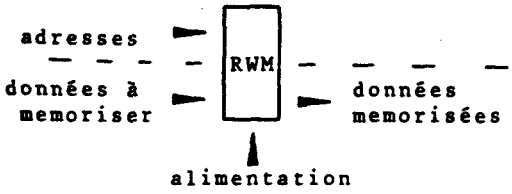
où $\underline{M}$ - ensemble des opérateurs-sélecteurs qui sélectionnent les structures subordonnées.

Interprétation :

Les formes transitives/mémorisantes de deuxième ordre constituent des ensembles de composants qui mémorisent plusieurs valeurs de variables considérées comme l'information de premier ordre. L'information de deuxième ordre peut être conçue comme des adresses de composants mémorisants formant le niveau inférieur de complexité. Dans le cas de la mémoire à écriture et à lecture les composants mémorisants ($M_1..M_n$) sont intégrés et on ne peut accéder qu'à un seul de ces composants à la fois, en conséquence qu'à une seule information. Dans le cas des registres ou des bascules les composants mémorisants sont distribués

et il est possible de sélectionner, par une seule valeur d'information de deuxième ordre, plusieurs registres et/ou bascules simultanément.

Dans ces deux cas nous utilisons la sélection spatiale des composants mémorisants.



où RWM - Read Write Memory.

3.3. Formes transitives/mémorisantes-transitives de deuxième ordre - $^{2}F_{T(MT)}$

Les formes $^{2}F_{T(MT)}$ de deuxième ordre forment deux niveaux de complexité. Au niveau inférieur nous trouvons les composants mémorisants-transitifs, au niveau supérieur les composants transitifs.L'information de deuxième ordre sélectionne aussi bien les composants mémorisants que les composants transitifs. Ce qui permet d'enregistrer l'information transformée par le système.

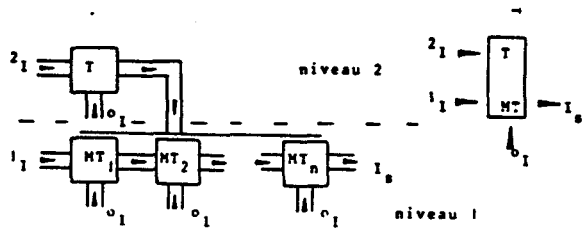
Représentation textuelle :

$$^{2}F_{T(MT)}[^{2}I,^{1}I] = {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I \Rightarrow| {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

avec la sélection (contrôle) positive et négative :

$$^{2}F_{T(MT)}[^{2}I,^{1}I] = {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I \Rightarrow| {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

Représentation graphique :



3.4. Formes génératrices de deuxième ordre - $^{2}F_{2G}$

Les formes génératrices de deuxième ordre forment deux niveaux de complexité. Le niveau inférieur comporte les composants mémorisants-transitifs, et le niveau supérieur un composant générateur. L'information générée par ces systèmes de deuxième ordre n'est influencée par aucune information entrante, sa-f par l'influx d'énergie, c'est-à-dire l'information d'ordre zéro.

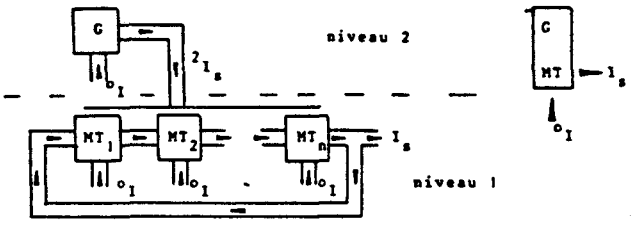
Il faut souligner que la sélection des formes mémorisantes-transitives a un caractère temporel du fait qu'elle est utilisée pour la production des séquences de fonctions.

Représentation textuelle :

$^{2}F_{2G} = {}^{1/2}F_{G} \Rightarrow {}^{2}I_{S} \Rightarrow | \circlearrowright {}^{1}F_{MT} \Rightarrow I_{S}$

(pas d'information entrante au niveau supérieur à zéro)

Représentation graphique :



où ^2I_S - l'information générée par le composant générateur au niveau 2,

I_S - l'information sortante recyclée au niveau 1.

Interprétation :

Les formes génératrices de deuxième ordre sont des systèmes multiphases utilisés comme horloges dans le matériel informatique. En principe ces systèmes ne sont pas contrôlés par des signaux externes, la seule entrée étant l'alimentation.

3.5. Formes transitives-génératrices/mémorisantes-transitives $^2F_{(TG)(MT)}$

Les formes $^2F_{(TG)(MT)}$ de deuxième ordre alors même qu'elles sont identiques aux formes génératrices ^2F_G par leur type de composants, sont complètement différentes par la nature de l'information qu'elles traitent. On peut les considérer comme les systèmes les plus simples de traitement de l'information du fait qu'elles emploient à la fois la sélection temporelle et la sélection spatiale des formes qui constituent le niveau inférieur du système.

a)Représentation textuelle sans rétroaction :

$$^2F_{(TG)(MT)}[^2I, {}^1I] = {}^{1/2}I \Rightarrow \begin{matrix} {}^{1/2}F_T \Rightarrow {}^2I_{Sp} \Rightarrow \\ {}^{1/2}F_G \Rightarrow {}^2I_{Te} \Rightarrow \end{matrix} \Big| \; {}^1I \Rightarrow {}^1F_{MT} \Rightarrow I_S$$

où $^2I_{Sp}$ - information de la sélection spatiale,

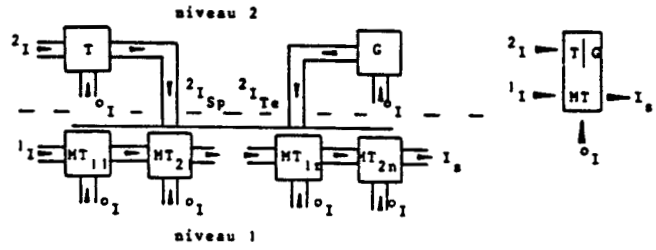
$^2I_{Te}$ - information de la sélection temporelle.

b)Représentation textuelle avec rétroaction (du niveau 1 vers le niveau 2) :

$$^2F_{(TG)(MT)}[^2I, {}^1I] = \begin{array}{l} {}^{1/2}F_G \Rightarrow {}^2I_{Te} \Rightarrow \\ {}^{1/2}I \Rightarrow {}^{1/2}F_T \Rightarrow {}^2I_{Sp} \Rightarrow \end{array} {}^1I \Rightarrow {}^1F_{MT} \Rightarrow I_S$$

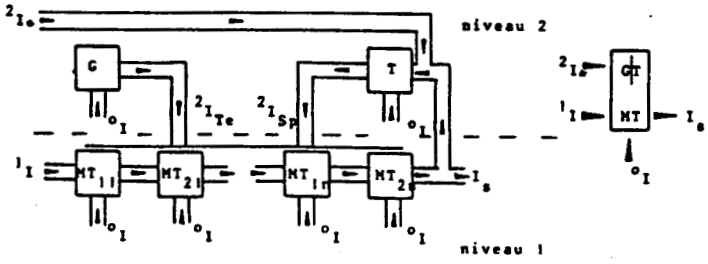
rétroaction

a)Représentation graphique de $^2F_{(TG)(MT)}$ sans rétroaction :



où $\underline{\quad}^{2}I_{Sp}\,|\,|\,|\,^{2}I_{Te}$ - la sélection combinée par l'information entrante pour la sélection spatiale, et par l'information générée pour la sélection temporelle.

b)Représentation graphique de $^2F_{(TG)(MT)}$ avec rétroaction :



où la boucle de rétroaction introduit l'"autosélection" par le contenu des composants subordonnés $MT_{11}..MT_{2n}$.

Interprétation:

Les formes transitives-génératrices/mémorisantes-transitives sont capables d'exécuter plusieurs processus différents. Les rudiments de séquencement (deux états) sont assurés par le composant générateur et la sélection temporelle. La base de la variété fonctionnelle est donnée par la sélection spatiale, précédée éventuellement par la transformation des signaux de la sélection spatiale $^{2}I_{Sp}$.

La forme transitive-génératrice/mémorisante de type a permet d'exécuter plusieurs processus différents, chacun étant une composition de plusieurs fonctions. Chaque séquencement interne est réduit à deux phases. Le choix des fonctions, et par là même des processus, est effectué par l'information extérieure de deuxième ordre. Ce type de forme n'est pas capable de "mémoriser" les séquences de fonctions. Il n'est pas "auto-commandé".

La forme $^{2}F_{(TG)(MT)}$ de type b comporte une boucle de rétroaction. Cette boucle transporte l'information de niveau inférieur afin de sélectionner, à la phase suivante du fonctionnement, un ou plusieurs composants mémorisants-transitifs du même niveau. Cette forme est "auto-commandée", ce qui permet de construire des processus à plusieurs phases à l'intérieur du système. Ces processus peuvent être "modifiés" ou "choisis" par l'information extérieure de deuxième ordre.

La nature des processus peut être différente. Nous ici en évoquer deux types :

1) les processus de séquencement caractéristiques de la partie contrôle des processeurs informatiques,
2) Les processus de traitement caractéristiques de la totalité des processeurs informatiques.

Dans le premier cas l'information sortante constitue les signaux de séquencement qui ensuite peuvent activer les différentes opérations de traitement (opérations terminales). Les systèmes d'un tel type forment le séquenceur. Le séquenceur de la représentation structurelle a produit plusieurs séquences différentes de contrôle, toutes définies entièrement par l'information externe (les instructions). La forme b avec rétroaction permet de modifier dynamiquement, c'est-à-dire pendant

le déroulement du processus, la suite de ce processus initialement défini par l'information externe.

Une forme transitive-génératrice/mémorisante-transitive de deuxième ordre peut également exécuter directement le traitement des données. Dans ce cas les formes subordonnées mémorisantes-transitives mémorisent et transforment les données. Elles "performent" les fonctions de traitement (fonctions terminales). Il faut souligner que ce type de traitement exige un séquencement des fonctions défini extérieurement au système ou prévu au niveau des données (niveau inférieur) et transmis par une boucle rétroactive au niveau supérieur.

4. Formes de troisième ordre - ^{3}F

Les formes de troisième ordre possèdent trois niveaux de complexité : niveaux supérieur, médian et inférieur. Le niveau supérieur est constitué du composant générateur, les niveaux médian et inférieur des composants transitifs et/ou mémorisants. Les formes étudiées dans ce sous-chapitre se distinguent essentiellement par la nature des deux niveaux subordonnés au niveau supérieur générateur.

Par rapport aux formes de deuxième ordre de complexité, les formes de troisième ordre contiennent un niveau de multiplexité de plus qui donne soit la nouvelle dimension de la sélection temporelle, soit celle de la sélection spatiale.

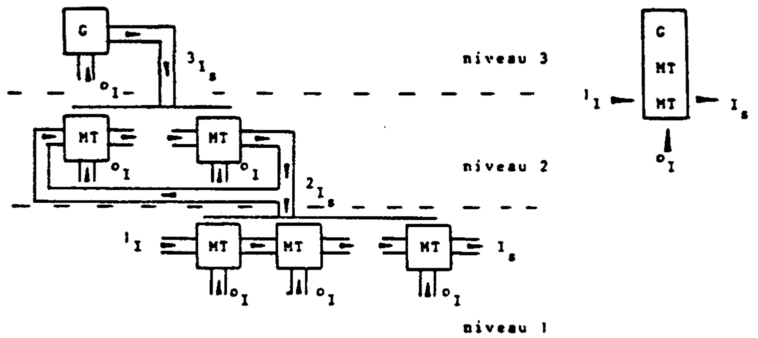
4.1. Formes de troisième ordre $^{3}F_{2G(MT)}$

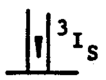
Les niveaux de complexité donnent trois niveaux d'états superposés. Le niveau supérieur comporte un composant générateur, les deux autres niveaux des composants mémorisants-transitifs. La structure génératrice de deuxième ordre est formée de deux niveaux : le supérieur et le médian. Le niveau inférieur est construit à partir des composants mémorisants-transitifs ; il transforme et mémorise l'information entrante.

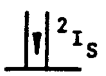
Représentation textuelle :

$$^3F_{2G(MT)}[{}^1I] = {}^{1/3}F_G \Rightarrow {}^3I_s \Rightarrow\!| \; {}^{1/2}F_G \Rightarrow {}^2I_s \Rightarrow\!| \; {}^1I \Rightarrow {}^1F_{MT} \Rightarrow I_S$$

Représentation graphique :



où  - les sélecteurs des composants structurels MT de niveau médian,

 - les sélecteurs des composants mémorisants-transitifs du niveau inférieur (terminal).

Interprétation :

La forme $^3F_{2G(MT)}$ est utilisée soit comme séquenceur, soit comme processeur. Les niveaux supérieur. et médian forment l'horloge multiphase qui effectue :

- soit la sélection temporelle des composants mémorisants-transitifs du séquenceur, celui-ci produisant des séquences à plusieurs pas, chacun de plusieurs phases ; cette possibilité donne des séquenceurs produisant une grande variété de processus de séquencement.
- soit le cadencement multiphase des opérations exécutées sur les données stockées dans les composants mémorisants et transformées par les composants transitifs ; possibilité qui ne permet d'exécuter qu'un seul processus multiphase.

4.2. Formes de troisième ordre $^{3}F_{2GT}$

Par rapport aux formes précédentes les formes $^{3}F_{2GT}$ ne possèdent au niveau inférieur que des composants transitifs. Ces forment donnent deux niveaux d'états : les états générés par le niveau supérieur, et ceux générés par le niveau médian.

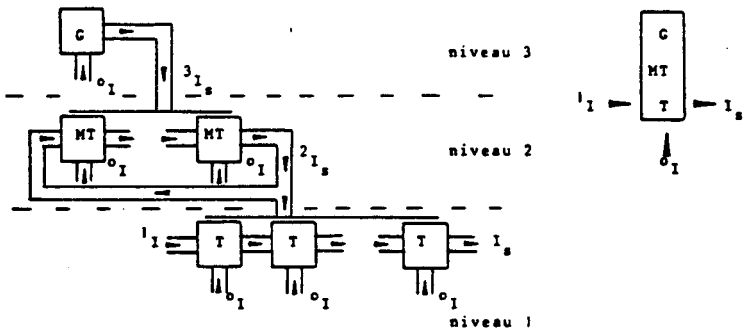
Représentation textuelle :

$$^{3}F_{2GT}[^{1}I] = {}^{1/3}F_{G} \Rightarrow {}^{3}I_{s} \Rightarrow| \; {}^{1/2}F_{G} \Rightarrow {}^{2}I_{s} \Rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

ou de façon contractée pour les formes génératrices :

$$^{3}F_{2GT}[^{1}I] = {}^{2/3}F_{G} \Rightarrow {}^{2}I_{s} \Rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

Représentation graphique :



Interprétation :

La forme $^{3}F_{2GT}$ peut servir comme partie de commande du système informatique. Les niveaux supérieur et médian génèrent les signaux de synchronisation et de séquencement, le niveau inférieur donne des ramifications des états de séquencement (ici phases d'horloge multiphase). Ce type de forme est largement utilisé pour le contrôle microprogrammé.

4.3. Formes de troisième ordre $^{3}F_{G2(MT)}$

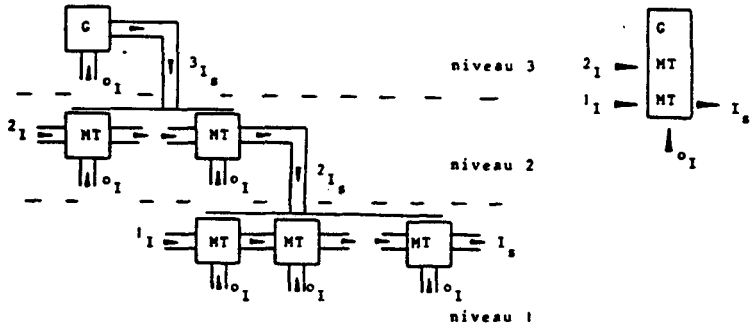
Les niveaux médian et inférieur des formes $^{3}F_{G2(MT)}$ sont construits à partir de composants mémorisants-transitifs. Chacun de ces deux niveaux a son information entrante : information de deuxième ordre pour le niveau médian, et information de premier ordre pour le niveau inférieur.

Représentation textuelle :

$^{3}F_{G2(MT)}[^{2}I, ^{1}I] =$

$^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow| \; ^{1/2}I \Rightarrow {}^{1/2}F_{MT} \Rightarrow {}^{2}I_{S} \Rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$

Représentation graphique :



Interprétation :

Les formes $^{3}F_{G2(MT)}$ servent pour l'exécution de différents processus. L'information ^{2}I joue le rôle des codes opératifs, l'information ^{1}I celui des données à traiter par le niveau inférieur. La sélection des composants de traitement ne peut être effectuée pendant le déroulement de ce processus. Il faut souligner qu'à chaque sélection de composants terminaux correspond un état de fonctionnement du niveau médian.

4.4. Formes de troisième ordre $^{3}F_{G(MT)T}$

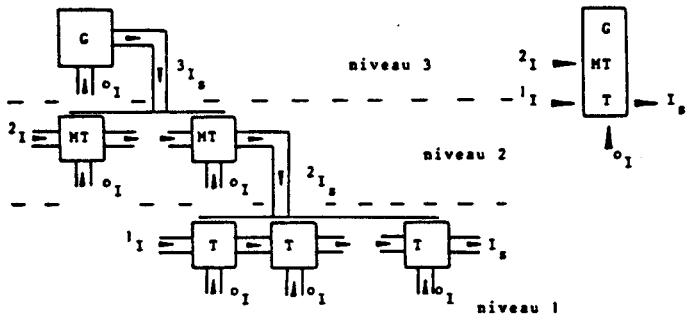
Le niveau supérieur des formes $^{3}F_{G(MT)T}$ est, comme pour les formes précédentes , constitué d'un composant générateur. Le niveau médian est construit à partir de composants mémorisants-transitifs. Le niveau inférieur est constitué de composants transitifs. Dans les formes de ce type existent donc deux sortes d'états superposés : les états générateurs (états de synchronisation), et les états du niveau médian représentés par les contenus des composants mémorisants.

Représentation textuelle :

$^{3}F_{G(MT)T}[^{2}I, ^{1}I] =$

$^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow| \; ^{1/2}I \Rightarrow {}^{1/2}F_{MT} \Rightarrow {}^{2}I_{S} \Rightarrow| \; ^{1}I \Rightarrow {}^{1}F_{T} \Rightarrow I_{S}$

Représentation graphique :



Interprétation :

La forme $^{3}F_{G(MT)T}$ constitue une machine à plusieurs états de fonctionnement enregistrés au niveau médian, et à plusieurs sorties pour chaque état, chacune conditionnée au niveau inférieur. Cette forme peut produire plusieurs séquences différentes et pour chaque pas d'une certaine séquence il peut y avoir plusieurs sorties. Elle remplit le rôle de partie de commande

d'un ordinateur. Il faut toutefois souligner qu'elle ne peut pas générer de séquences conditionnées intérieurement. Toutes les séquences et leurs sorties sont complètement définies par l'information entrante ^{2}I et ^{1}I. Les informations ^{2}I spécifient le type de séquence, et les informations ^{1}I définissent les sorties de chaque pas de la séquence sélectionnée.

4.5. Formes de troisième ordre $^{3}F_{GT(MT)}$

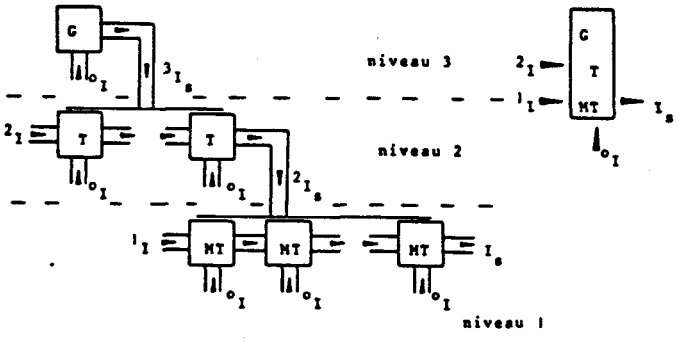
Le niveau supérieur des formes $^{3}F_{GT(MT)}$ est bien sûr générateur, le niveau médian transitif, et le niveau inférieur mémorisant-transitif. Par leur composition ces formes constituent l'"inversion" des formes $^{3}F_{G(MT)T}$ décrites précédemment.

Représentation textuelle :

$$^{3}F_{GT(MT)}[^{2}I, ^{1}I] =$$

$$^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow| \; ^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I_{S} \Rightarrow| \; ^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

Représentation graphique :



Interprétation :

Les formes $^{3}F_{GT(MT)}$ constituent un certain type de processeurs capables d'exécuter des processus définis extérieurement. Ces

formes assurent seulement les séquences à un pas générées au niveau supérieur par le composant générateur F_G. Par ailleurs ce type de forme peut exécuter une grande variété de pas donnés par le niveau médian transitif, lui-même conditionné par l'information de deuxième ordre. Le niveau médian sélectionne les composants mémorisants-transitifs subordonnés. Les sorties ^2I_S peuvent donc être interprêtées comme des codes opérateurs ou adresses.

4.6. Formes $^3F_{G2(MT)}$ et $^3F_{GT(MT)}$ à rétroaction

Toutes les formes de troisième ordre présentées jusqu'à présent produisaient des processus plus ou moins complexes définis totalement par l'information entrante, extérieure. Par l'introduction de la rétroaction interne il devient possible de modifier et d'enchaîner les processus de base au moyen de l'information interne.

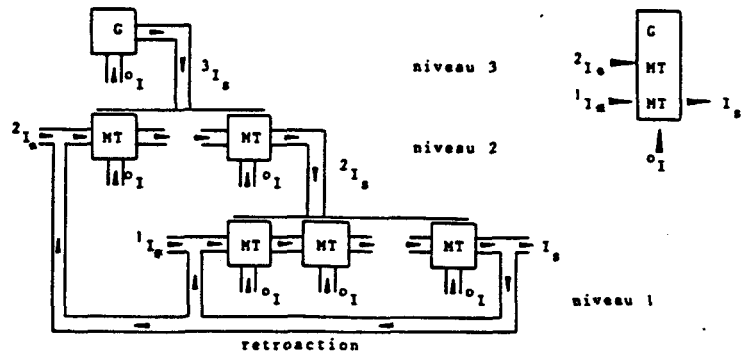
Représentation textuelle de $^3F_{G2(MT)}$ à rétroaction :

$$^3F_{G2(MT)}[^2I_{\#}, {}^1I_{\#}] =$$

$$1/3F_G \Rightarrow {}^3I_S \Rightarrow | \; 1/2I \Rightarrow {}^{1/2}F_{MT} \Rightarrow {}^2I_S \Rightarrow | \; {}^1I \Rightarrow {}^1F_{MT} \Rightarrow I_S$$

où : $^2I_{\#}, {}^1I_{\#}$ - informations entrantes et rétroactives.

Représentation graphique :



Interprétation :

La forme ${}^{3}F_{G2(MT)}$ à boucle rétroactive permet d'exécuter des processus de traitement modifiés ou composés par l'information de niveau inférieur. Les contenus des composants mémorisants au niveau inférieur peuvent être interprêtés comme de l'information d'"auto-contrôle". Cette information, après sa transformation et sa mémorisation, sélectionne les composants du même niveau inférieur. Cela permet d'enchaîner plusieurs processus définis par le niveau médian, et, en conséquence, d'exécuter plusieurs fois le même processus ou des processus modifiés par rapport à ceux définis extérieurement. Dans le cas du système ${}^{3}F_{G2(MT)}$ chaque valeur différente de l'information ${}^{2}I_{S}$ correspond à un état fonctionnel différent du niveau médian.

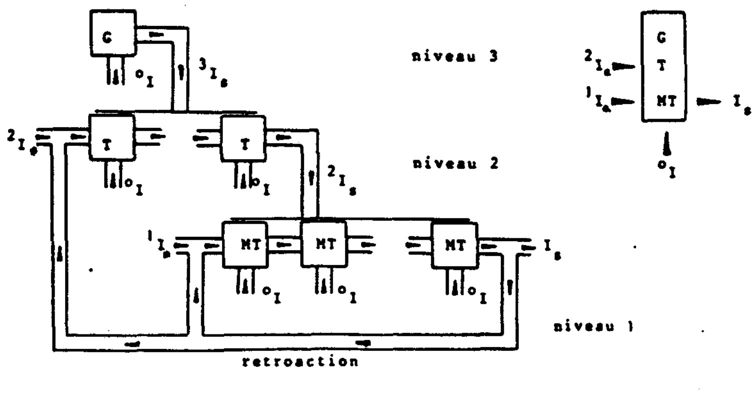
Représentation textuelle de ${}^{3}F_{GT(MT)}$ à rétroaction :

$${}^{3}F_{GT(MT)}[{}^{2}I_{*}, {}^{1}I_{*}] =$$

$${}^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow\!| \; {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I_{S} \Rightarrow\!| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

où ${}^{2}I_{*}, {}^{1}I_{*}$ - informations entrantes rétroactives.

Représentation graphique :



Interprétation :

La forme $^{3}F_{GT(MT)}$ qui possède une boucle de rétroaction permet d'exécuter plusieurs processus enchaînés ou modifiés par l'information du niveau inférieur. Chaque processus de base consiste en un seul pas. Par la boucle de rétroaction on obtient la possibilité de construire des processus à plusieurs pas.

5. Formes de quatrième ordre - ^{4}F

Les formes de quatrième ordre possèdent quatre niveaux systèmiques*. Le niveau 4 est constitué, comme dans les formes de troisième ordre, d'un composant générateur de signaux primaires de synchronisation. Le niveau 3, dans toutes les formes de quatrième ordre présentées, est construit à partir de composants mémorisants-transitifs. Ce niveau nous donne les états de synchronisation multiphase ou les états de séquencement. Les niveaux 2 et 1 peuvent être formés soit de composants transitifs, soit de composants mémorisants-transitifs.

Dans ce sous-chapitre nous présenterons cinq formes différentes de quatrième ordre. Quatre d'entre elles sont des formes capables d'exécuter plusieurs processus différents, arbitrairement longs.

5.1. Formes de quatrième ordre $^{4}F_{2G(MT)T}$

Les niveaux 2, 3 et 4 des formes $^{4}F_{2G(MT)T}$ sont des niveaux de séquencement basés sur la sélection temporelle. Le niveau 1 comporte les composants transitifs qui ramifient chaque état de séquencement en plusieurs sorties.

* A partir de ce degré de complexité nous introduisons une numérotation pour désigner les différents niveaux :

niveau inférieur - niveau 1
niveaux médians - niveaux 2, 3,.., n-1
niveau supérieur - niveau n.

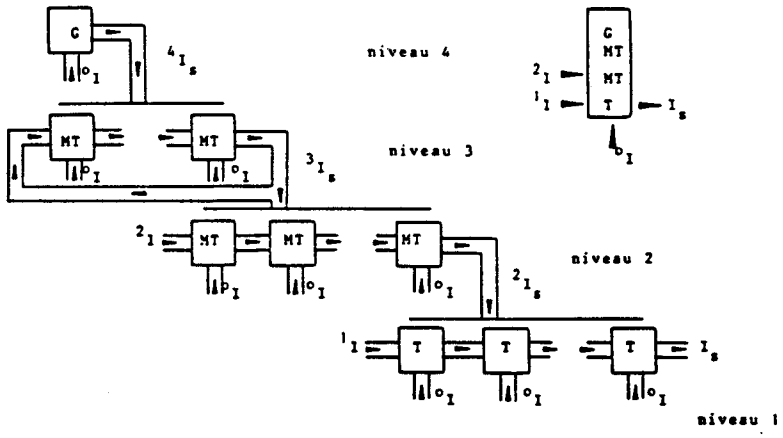
Représentation textuelle :

$$^{4}F_{2G(MT)T}\,[^{2}I,{}^{1}I] =$$

$$^{1/4}F_{G} \Rightarrow {}^{4}I_{S} \Rightarrow| \; {}^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow| \; {}^{1/2}I \Rightarrow {}^{1/2}F_{MT} \Rightarrow {}^{2}I_{S} \Rightarrow|$$

$$^{1}I \Rightarrow {}^{1}F_{T} \Rightarrow I_{S}$$

Représentation graphique :



Interprétation :

Une forme $^{4}F_{2G(MT)T}$ de quatrième ordre représente une structure "riche" de séquencement et de ramification des signaux de contrôle. Les niveaux 2, 3 et 4 produisent plusieurs séquences différentes, chacune à plusieurs pas, et chaque pas à plusieurs phases. Par ces propriétés cette forme peut être considérée comme une unité de commande très simple. La flexibilité de cette unité est totalement conditionnée par les informations entrantes au niveau 2 et au niveau 1. La forme représentée ci-dessus ne possède pas de boucle de rétroaction.

5.2. Formes de quatrième ordre $^{4}F_{2GT(MT)}$

Les formes $^{4}F_{2GT(MT)}$ avec leurs quatre niveaux de complexité donnent la possibilité d'exécuter différents processus, tous de même longueur, c'est-à-dire ayant le même nombre de phases. Les niveaux 3 et 4 génèrent plusieurs états de synchronisation. Le niveau 2 comporte des composants transitifs qui eux ne possèdent pas leurs propres états. Le niveau inférieur est construit à partir de composants mémorisants-transitifs qui transforment et mémorisent l'information à traiter.

Représentation textuelle :

$$^{4}F_{2GT(MT)}[^{2}I, ^{1}I] =$$

$$^{1/4}F_{G} \Rightarrow {}^{4}I_{S} \Rightarrow| \; ^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow|$$

$$^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I_{S} \Rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

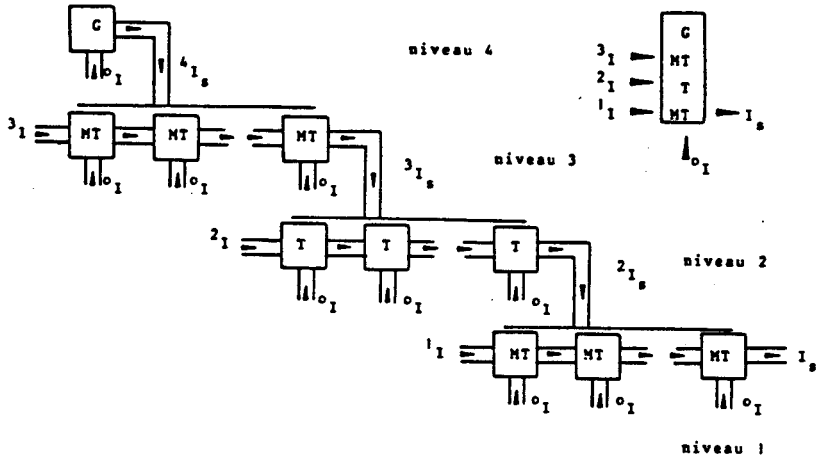
et avec rétroaction :

$$^{4}F_{2GT(MT)}[^{2}I_{\bullet}, ^{1}I_{\square}] =$$

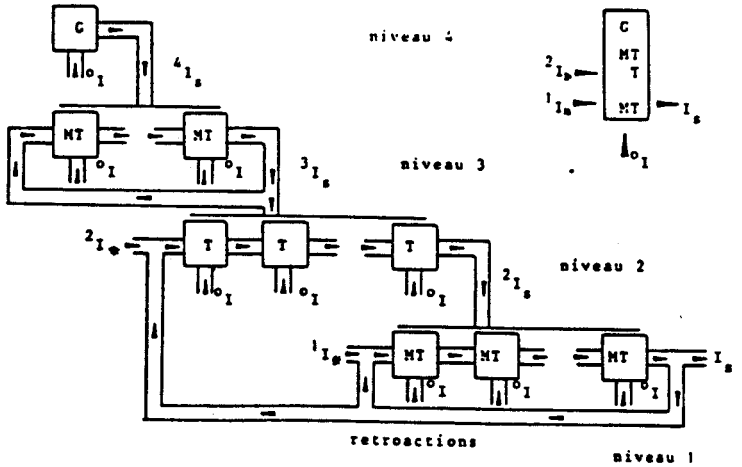
$$^{1/4}F_{G} \Rightarrow {}^{4}I_{S} \Rightarrow| \; ^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \Rightarrow|$$

$$^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I_{S} \Rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

Représentation graphique de ${}^{4}F_{2GT(MT)}[{}^{2}I, {}^{1}I]$:



Représentation graphique de ${}^{4}F_{2GT(MT)}[{}^{2}I_{\phi}, {}^{1}I_{\psi}]$:



où ${}^{2}I_{\phi}, {}^{1}I_{\psi}$ - informations entrantes et rétroactives.

Interprétation :

La forme $^{4}F_{2GT(MT)}$ peut exécuter plusieurs processus, chacun d'un même nombre de phases, générés par les niveaux 3 et 4 (horloge). La variété des processus est assurée par la sélection spatiale effectuée au niveau systémique 2. L'information entrante ^{2}I est transformée par les composants transitifs afin de sélectionner dans d'"espace" du niveau 1 les composants mémorisants-transitifs. Ces composants, alors activés, exécutent certaines opérations sur l'information de premier ordre.

La première structure, sans boucle rétroactive, requiert la définition de tous les processus l'un après l'autre à l'extérieur de la forme. La "programmation", ou plutôt la "microprogrammation", c'est-à-dire l'enchaînement des processus de base, doit être effectuée par les mécanismes extérieurs.

La même forme, mais avec boucle de rétroaction, permet d'interprêter au niveau 2 l'information mémorisée au niveau 1. Ce mécanisme peut enchaîner un certain nombre de processus de base (microinstructions) afin d'exécuter des microprogrammes simples. Il donne aussi la possibilité de modifications dynamiques des processus de base pendant leur exécution. Cette forme peut donc être considérée comme une machine informatique complète. Toutefois son rôle est limité par l'absence de mémoire de lecture et d'écriture : elle ne peut exécuter que des programmes fixés dans la mémoire "morte" (à lecture seule).

5.3. Formes de quatrième ordre $^{4}F_{2G2(MT)}$

Le niveau 4 des formes $^{4}F_{2G2(MT)}$ génère la phase principale de synchronisation. Le niveau 3 génère plusieurs phases-états de séquencement dérivées de la phase principale. Le niveau 2 produit les signaux-états de séquencement enchaînés dans différentes séquences de différentes longueurs.

Représentation textuelle :

$${}^{4}F_{2G2(MT)}[{}^{2}I, {}^{1}I] =$$

$${}^{1/4}F_{G} \Rightarrow {}^{4}I_{S} \rightarrow| \; {}^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \rightarrow|$$

$${}^{1/2}I \Rightarrow {}^{1/2}F_{MT} \Rightarrow {}^{2}I_{S} \rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

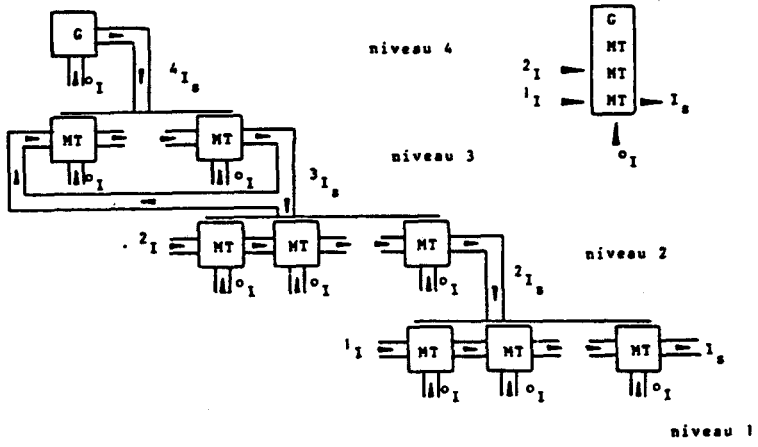
avec rétroaction :

$${}^{4}F_{2G2(MT)}[{}^{2}I_{*}, {}^{1}I_{*}] =$$

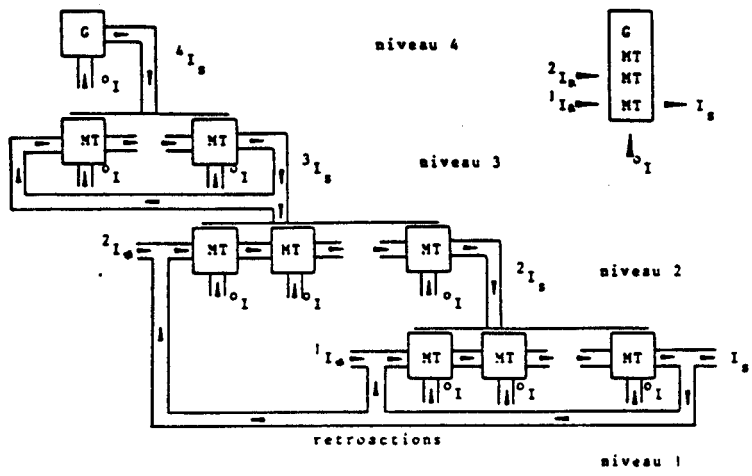
$${}^{1/4}F_{G} \Rightarrow {}^{4}I_{S} \rightarrow| \; {}^{1/3}F_{G} \Rightarrow {}^{3}I_{S} \rightarrow|$$

$${}^{1/2}I \Rightarrow {}^{1/2}F_{MT} \Rightarrow {}^{2}I_{S} \rightarrow| \; {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

Représentation graphique de ${}^{4}F_{2G2(MT)}[{}^{2}I, {}^{1}I]$:



Représentation graphique de ${}^{4}F_{2G2(MT)}[{}^{2}I_{\#}, {}^{1}I_{\#}]$:



Interprétation :

Les formes ${}^{4}F_{2G2(MT)}$ donnent quatre niveaux différents d'états. Les niveaux 3 et 4 mémorisent et transforment les états de l'horloge multiphase. Le niveau 2 mémorise et transforme les états du processus de séquencement. Enfin le niveau inférieur mémorise et transforme l'information à traiter (les données). Ces formes assurent des jeux d'états de séquencement extrêmement riches auxquels nous pouvons subordonner la même richesse de composition des formes terminales qui transforment et mémorisent les données. A chaque état du niveau 2 correspondent un ou plusieurs composants mémorisants-transitifs qui exécutent une ou plusieurs fonctions de traitement simultanément. La forme ${}^{4}F_{2G2(MT)}$ avec boucle rétroactive permet d'enchaîner ou de modifier les processus de base exécutés par le système. L'information rétroactive sert de code opérateur qui choisit ou influe sur le déroulement des processus de séquencement.

5.4. Formes de quatrième ordre ${}^4F_{G(MT)T(MT)}$

Les formes ${}^4F_{G(MT)T(MT)}$ constituent une vaste classe de systèmes à quatre niveaux de complexité que l'on trouve sous la forme de processeurs à contrôle "câblé". Le niveau 4 comporte un composant générateur. Le niveau 3 est construit à partir de composants mémorisants-transitifs qui produisent plusieurs séquences différentes. Le niveau 2 ramifie les états de séquencement en plusieurs sorties. Enfin le niveau 1 est constitué de composants mémorisants-transitifs afin de mémoriser et transformer l'information de premier ordre.

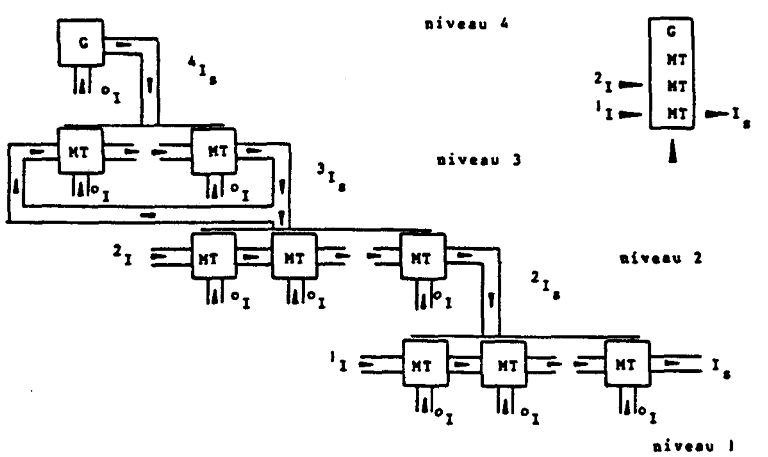
Représentation textuelle :

$${}^4F_{G(MT)T(MT)}[{}^3I, {}^2I, {}^1I] = {}^{1/4}F_G \Rightarrow {}^4I_S \Rightarrow|$$
$${}^{1/3}I \Rightarrow {}^{1/3}F_{MT} \Rightarrow {}^3I_S \Rightarrow| \; {}^{1/2}I \Rightarrow {}^{1/2}F_T \Rightarrow {}^2I_S \Rightarrow| \; {}^1I \Rightarrow {}^1F_{MT} \Rightarrow I_S$$

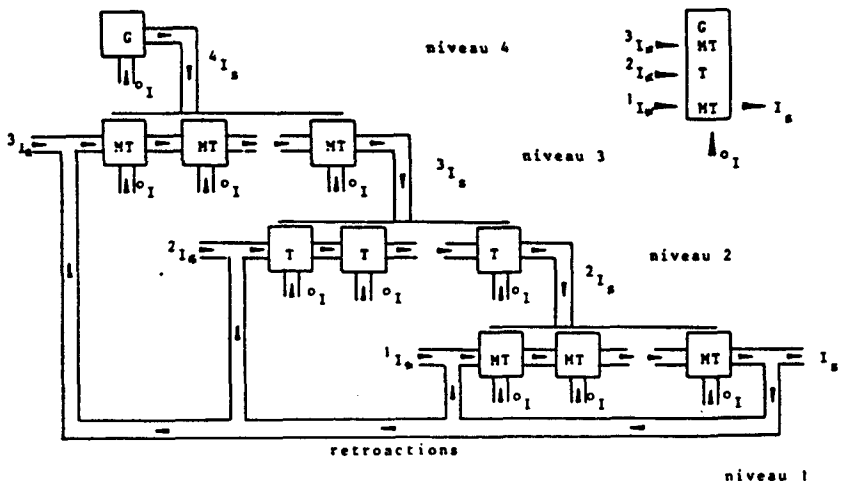
avec rétroaction :

$${}^4F_{G(MT)T(MT)}[{}^3I_*, {}^2I_*, {}^1I_*] = {}^{1/4}F_G \Rightarrow {}^4I_S \Rightarrow|$$
$${}^{1/3}I \Rightarrow {}^{1/3}F_{MT} \Rightarrow {}^3I_S \Rightarrow| \; {}^{1/2}I \Rightarrow {}^{1/2}F_T \Rightarrow {}^2I_S \Rightarrow| \; {}^1I \Rightarrow {}^1F_{MT} \Rightarrow I_S$$

Représentation graphique de ${}^4F_{G(MT)T(MT)}[{}^3I, {}^2I, {}^1I]$:



Représentation graphique de ${}^{4}F_{G(MT)T(MT)}[{}^{3}I_{e}, {}^{2}I_{e}, {}^{1}I_{e}]$:



Interprétation :

Les deux structures présentées ci-dessus appartiennent au même type de forme ${}^{4}F_{G(MT)T(MT)}$ et sont des processeurs complets. Les niveaux 3 et 4 génèrent les signaux de synchronisation et de séquencement. Ces signaux sont utilisés pour la sélection temporelle des composants des niveaux 1 et 2 subordonnés.

Le niveau 2 a soit l'un soit l'autre des deux rôles suivants :

- Il exécute les fonctions de sortie pour différents états de séquencement du niveau 3 ; dans ce cas il est associé au niveau 3,
- Il exécute directement les fonctions de sélection des composants mémorisants et/ou transitifs du niveau terminal.

L'intégration du niveau 2 au niveau 1 donne d'une part des formes mémorisantes de deuxième ordre : les mémoires de lecture et d'écriture, d'autre part des formes transitives de deuxième ordre : les unités multifonctionnelles (UAL - Unités Arithmétiques Logiques).

Dans ce deuxième cas les informations de deuxième ordre ^{2}I constituent des codes opérateurs et des adresses. L'enchaînement et la modification dynamique des processus de base, qui correspondent ici aux instructions, peuvent être effectués par les boucles rétroactives qui vont du niveau 1 au niveau 3 et au niveau 2. La première boucle rétroactive permet de transporter les instructions mémorisées au niveau 1 jusqu'au niveau 3 où elles sont interprétées comme des codes opérateurs. La deuxième boucle rétroactive permet de transporter les adresses et les codes opérateurs "locaux" des unités fonctionnelles du niveau 1 jusqu'au niveau 2 où ils sont interprétés afin de sélectionner les formes mémorisantes et transitives du niveau inférieur, c'est-à-dire les composants terminaux.

6. Formes de cinquième ordre - ^{5}F

Les cinq niveaux de complexité des formes de cinquième ordre leur confèrent une très grande flexibilité et multiplexité fonctionnelle. Le niveau supérieur génère les signaux de synchronisation. Le niveau inférieur est constitué des composants mémorisants et transitifs terminaux. Le séquencement est assuré par le niveau 4. Les niveaux intermédiaires 2 et 3 effectuent la ramification générale et fine des états de séquencement. La ramification fine correspond à la sélection spatiale fine des composants mémorisants (mémoires vives) et des composants transitifs (unités plurifonctionnelles) qui forment le niveau inférieur (terminal).

6.1. Formes de cinquième ordre $^{5}F_{2G2T(MT)}$

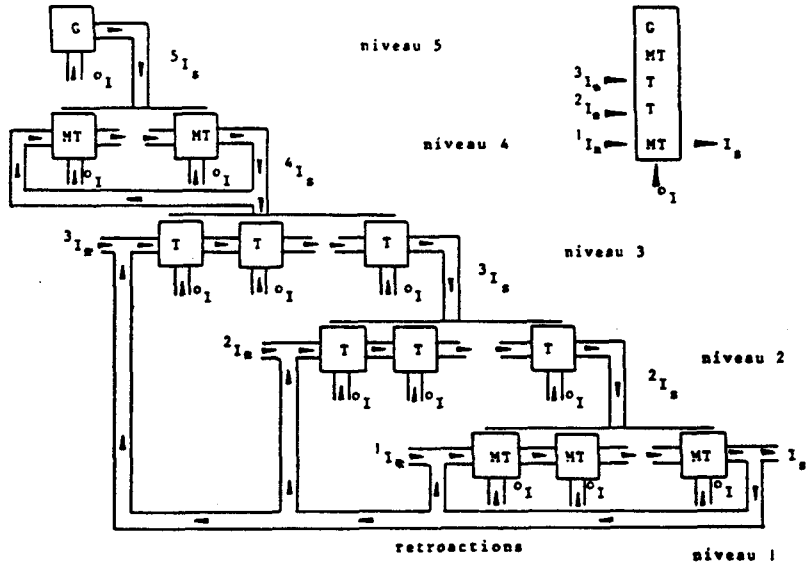
Les niveaux 3 et 4 de ces formes produisent les états signaux de synchronisation. Les composants du troisième niveau sont déterminés par la sélection temporelle. La sélection spatiale des composants terminaux au niveau inférieur est effectuée par les formes transitives du niveau 2 (sélection fine) et du niveau 3 (sélection générale).

Représentation textuelle :

$$^{5}F_{2G2T(MT)}[^{3}I_{\#}, ^{2}I_{\#}, ^{1}I_{\#}] = {}^{1/5}F_{G} \Rightarrow {}^{5}I_{S} \Rightarrow| \; {}^{1/4}F_{G} \Rightarrow {}^{4}I_{S} \Rightarrow|$$

$$^{1/3}I \Rightarrow {}^{1/3}F_{T} \Rightarrow {}^{3}I_{S} \Rightarrow| \; {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I_{S} \Rightarrow| \; {}^{1}I \Rightarrow F_{MT} \Rightarrow I_{S}$$

Représentation graphique :



— Interprétation :

Nous pouvons considérer les processus de base exécutés par les formes $^{5}F_{2G2T(MT)}$ comme des instructions "puissantes". Toutes les instructions sont des processus ayant le même nombre de phases générées au quatrième niveau systémique. Les niveaux 2 et 3 sont construits à partir des formes transives qui permettent de ramifier d'abord généralement, puis plus finement les états phases de séquencement ; ils effectuent la sélection spatiale des formes terminales afin d'exécuter

les opérations sur les données.

Les boucles de rétroaction présentées sur le schéma structurel permettent d'enchaîner les processus de base (ou microinstructions) en microprogrammes. L'ascension de l'information du niveau 1 au niveau 2 aboutit à l'"auto-sélection" spatiale et fine des formes mémorisantes (mots de la mémoire vive) et transitives (sous-circuits d'une unité fonctionnelle). L'ascension de l'information du niveau 1 au niveau 3 permet la sélection spatiale générale des blocs de mémoire vive et des unités fonctionnelles.

6.2. Formes de cinquième ordre $^{5}F_{(GT)(MT)2T(MT)}$

Ces formes possèdent une importante flexibilité aussi bien dans le domaine de la sélection temporelle (séquencement) que dans celui de la sélection spatiale (décodage). Au niveau supérieur de complexité nous avons deux types de formes : la forme génératrice qui produit l'information de synchronisation, et les formes transitives qui transforment l'information extérieure de cinquième ordre afin de contrôler l'ensemble des quatre niveaux systémiques subordonnés. Le niveau 4 est construit à partir de formes mémorisantes-transitives. Les états fonctionnels de ce niveau constituent les états de séquencement pour toute la forme. Les niveaux 1, 2 et 3 sont identiques aux niveaux correspondants de la forme $^{5}F_{2G2T(MT)}$ présentée dans le paragraphe précédent.

Représentation textuelle :

$$^{5}F_{(GT)(MT)2T(MT)}[^{5}I, ^{4}I_{*}, ^{3}I_{*}, ^{2}I_{*}, ^{1}I_{*}] = \begin{array}{l} {}^{1/5}F_{G} \Rightarrow {}^{5}I_{Te} \Rightarrow \\ {}^{1/5}I \Rightarrow {}^{1/5}F_{T} \Rightarrow {}^{5}I_{Sp} \Rightarrow \end{array}$$

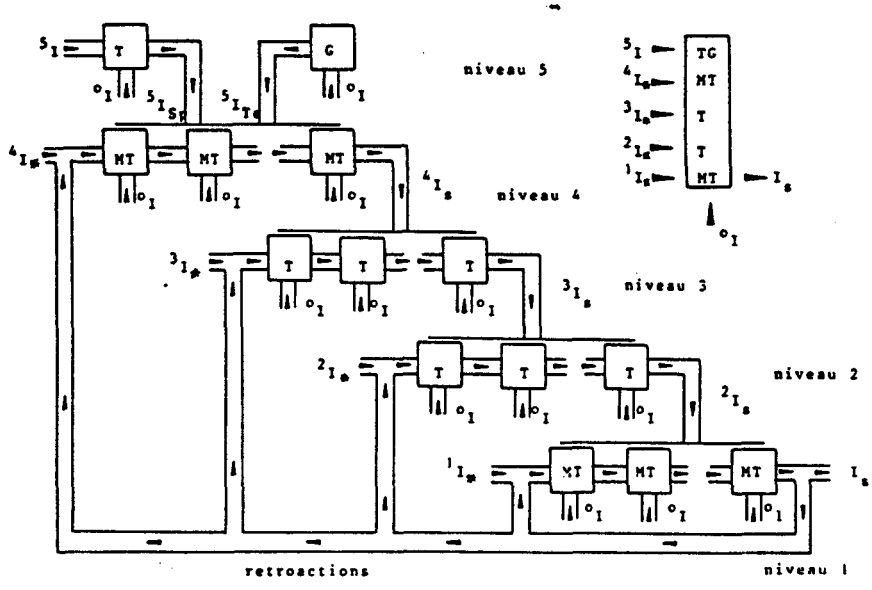
$$^{1/4}I \Rightarrow {}^{1/4}F_{MT} \Rightarrow {}^{4}I_{S} \Rightarrow | {}^{1/3}I \Rightarrow {}^{1/3}F_{T} \Rightarrow {}^{3}I_{S} \Rightarrow | {}^{1/2}I \Rightarrow {}^{1/2}F_{T} \Rightarrow {}^{2}I_{S} \Rightarrow | {}^{1}I \Rightarrow {}^{1}F_{MT} \Rightarrow I_{S}$$

où $^{4}I_{*}, ^{3}I_{*}, ^{2}I_{*}, ^{1}I_{*}$ - les informations entrantes et rétroactives,

$^{5}I_{Te}$ - l'information de la sélection temporelle,

$^{5}I_{Sp}$ - l'information de la sélection spatiale.

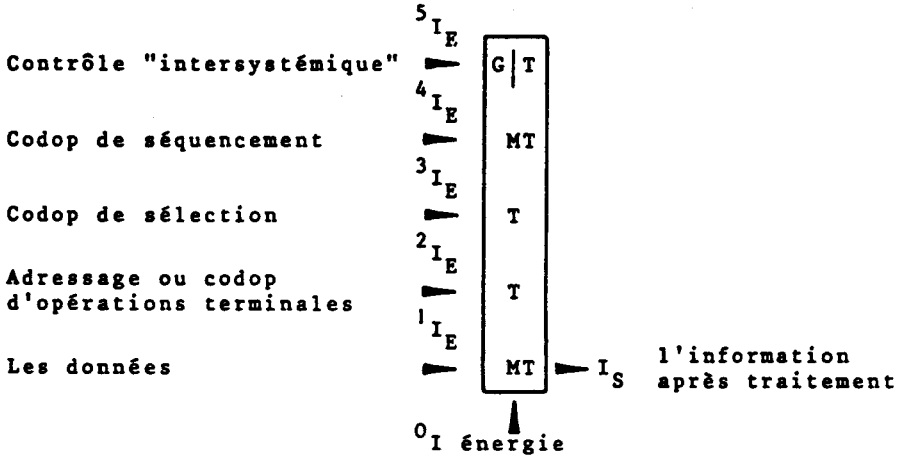
Représentation graphique :



Interprétation :

Les formes $^{5}F_{(GT)(MT)2T(MT)}$ exécutent les instructions des processeurs SISD (Single Instruction stream, Single Data stream). Le niveau supérieur joue ici deux rôles : celui de générateur de l'impulsion de synchronisation et celui de contrôle général de tous les niveaux subordonnés. Le contrôle général peut servir, par exemple, de mécanisme de coopération entre plusieurs systèmes de même type. L'enchaînement des processus de base (instructions) sur tous les niveaux de complexité (codes de séquencement, codes générateurs généraux, codes opérateurs et adresses des composants terminaux) est assuré par les boucles de rétroaction. Ce mécanisme d'enchaînement est celui de la programmation, c'est-à-dire de l'enchaînement de la complexité fonctionnelle de matériel.

Ci-dessous nous présentons le schéma de la forme $^{5}F_{(GT)(MT)2T(MT)}$ avec interprétation des informations entrantes :



7. Remarque finale

L'approche systémique et "morphologique" de la complexité hiérarchique de l'architecture du matériel constitue une base conceptuelle pour les développements consécutifs tels que le langage LIDO et le système LIDO. Cette base nous a permis de préparer un ensemble d'outils fortement intégrés qui imposent au concepteur de l'architecture du matériel la méthode et l'ordre hiérarchique de conception, à partir des niveaux systémiques les plus élevés jusqu'au niveaux les plus inférieurs.

Langage L I D O

Langage

Interprétable
pour la

Description
des

Ordinateurs

TABLE DES MATIERES

1. Introduction

2. Présentation du langage LIDO

2.1. Spécification fonctionnelle et structurelle du matériel au niveau transfert de registres
2.2. L'approche systémique

2.3. La conception de l'architecture du matériel évolué
2.3.1. L'architecture SIMD
2.3.2. L'architecture MIMD
2.3.3. L'architecture pipe-line
2.3.4. Les architectures basées sur les mémoires associatives

2.4. L'utilisation des mécanismes fonctionnels et structurels correspondant aux sous-systèmes topologiques
2.5. La spécification des entrées/sorties

3. Description comparative des processeurs de type RISC

3.1. Partie opérative du np32
3.2. Partie contrôle du np32 - description fonctionnelle
3.3. Partie contrôle du np32 - description mixte
3.4. Partie contrôle du np32 - descriptions structurelles
3.5. Le bi-processeur de type RISC - np32x2

4. Remarques finales

III. Langage LIDO

1. Introduction

Il est connu que le niveau le mieux adapté à la conception des ordinateurs est le niveau de transfert de registres.

"Computer system can be best designed based on registers. It is difficult to design them based on a level higher or lower than registers." [Bell76]

Par ailleurs une description structurelle et detaillée au niveau transfert de registres est le meilleur point de départ pour la compilation en silicium [Sil85]. Il nous semble aussi qu'une notation bien adaptée à la compilation des systèmes VLSI doit être essentiellement différente de celles utilisées pour la description du matériel existant avant l'époque de ces systèmes (p.e. CDL [Chu65], DDL [Dul68], AHPL [Hil171], CASSANDRE [Mer73], RTS I [Pil74], OSM [Mar75].

Un outil moderne conçu pour la conception des architectures évoluées et prévues pour l'intégration devrait permettre:

- La spécification fonctionnelle et structurelle des chemins de contrôle et des chemins de données.
- La spécification "systémique" incluant;
  - les systèmes hautement structurés et hiérarchisés
  - les systèmes à contrôle négatif et résiduel
- Un grand espace de conception incluant les machines SISD, SIMD, MIMD, les architectures pipe-line et data-flow.
- L'utilisation des mécanismes fonctionnels et structurels correspondants aux sous-systèmes topologiques des systèmes VLSI.
- La spécification des lignes et signaux d'entrée/sortie.

Ces cinq points constituent les lignes directrices qui ont présidé à la création du langage LIDO présenté dans les paragraphes suivants; et compte tenu de leur importance nous les développons ci-dessous un à un.

2. Présentation du langage LIDO.

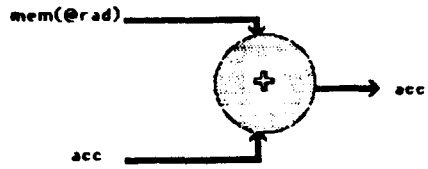
2.1. La spécification fonctionnelle et structurelle du matériel au niveau transfert de registres.

Avant de présenter l'utilisation du langage LIDO dans le contexte de la spécification fonctionnelle et/ou structurelle nous nous permettons de citer deux definitions données par A.Parker [Par81]:

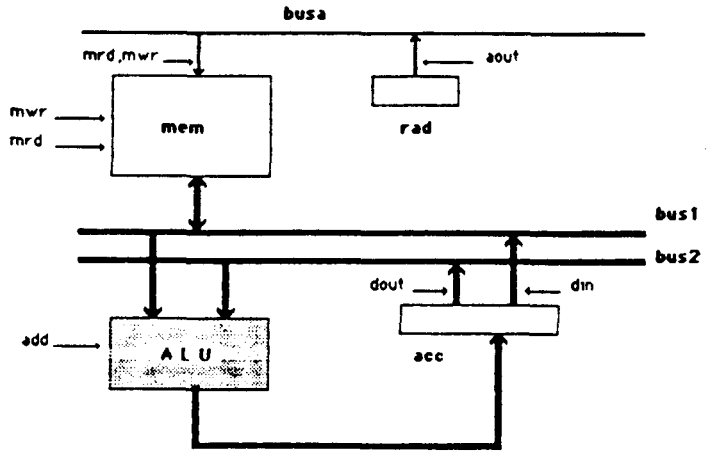
" *register transfer behavior - (description fonctionnelle):*
The operations of hardware are specified at register transfer level, but only abstract and visible transfers and registers are described; interactions between registers and functional units are usually implicit.
Control signals and overall control flow are usually implicit and/or expressed at higher level.

register transfer structure - (description structurelle):
Hardware implementation at register transfer level ; registers, functional operators and their interconnections are specified.
The control signals and overall control flow is embeded in the hardware as a part of the overall specification."

Le langage LIDO nous permet de décrire les deux aspects du matériel. La description fonctionnelle beaucoup plus abstraite et flexible offre un moyen de description très efficace mais relativement éloigné de la structure matérielle. Par contre la description structurelle beaucoup plus précise et contraignante permet d'exprimer la structure matérielle détaillée, ce qui convient mieux à la compilation en silicium. Le processus de transformation d'une description fonctionnelle en une description structurelle constitue la phase cruciale de la synthèse du matériel [Bar84]. L'objectif à atteindre: l'automatisation de cette transformation apporterait au concepteur un gain de temps considérable et ainsi lui permettrait de concentrer son effort sur la spécification fonctionnelle.
Ci-dessous nous présentons deux fragments de description en langage LIDO caractéristiques de la partie opérative d'un système complexe: l'une fonctionnelle et l'autre structurelle.




```
/* functional vs structural style of data path description */
/*** functional description of one ALU operation ***/
Fun 1,1;[1];                /* ALU functional unit */
/add/ acc + mem(@rad) => mem(@rad).
```

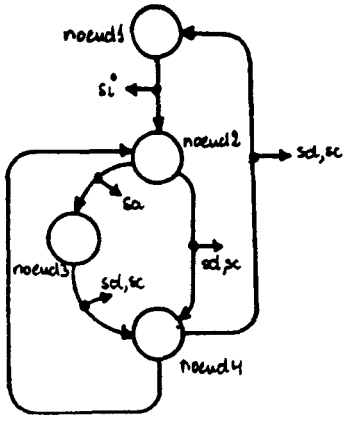



```
/*** structural description of one ALU operation ***/
Fun 1,1;[1];                /* ALU + transfer operations */
/dout/ acc => bus2;         /* accumulator to bus2 */
/aout/ rad => busa;         /* address to busa */
/mrd/  mem(@rad)=>bus1;     /* memory read */
/add/  bus1+bus2=>acc;      /* ALU add operation */
/din/  acc => bus1;         /* result to bus1 */
/mwr/  bus1=>mem(@rad).     /* memory write */

Connectors; bus1(32);bus2(32);busa(10).
Registers; rad(10);acc(32).
Memories; mem(1024,32).
```

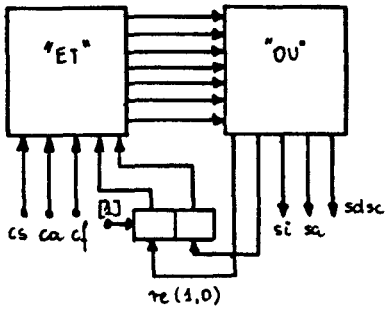
Fig 3.1. La description fonctionnelle (a), et la description structurelle (b) d'un fragment du chemin de données

La description fonctionnelle exprime exclusivement la fonction à exécuter associée aux arguments d'entrée et de sortie. Ces arguments sont des variables de type registre ou mémoire. La description structurelle, beaucoup plus riche en éléments matériels implique une certaine structure des connections qui vehiculent les signaux réprésentants différentes variables enregistrées dans les éléments mémoire. L'apparente complexité des descriptions structurelles disparait si on réalise une description complète et "systémique" exécutant un nombre important d'instructions.

La spécification de la partie contrôle constitue une tâche non-négligeable dans la conception du matériel. Il convient ici de pouvoir disposer en même temps des moyens d'expression fonctionnels et structurels, ce qu' offre le langage LIDO au travers des graphes pour la spécification fonctionnelle et des différents types de matrices logiques (+mémoires RAM et ROM) pour la spécification structurelle.
Ci-dessous nous présentons un exemple comparatif de ces deux types de description pour la partie contrôle d'un circuit de multiplication de deux nombres binaires.




```
/* functional vs structural control path description */
/*** functional description of multiplier control ***/
Graph 1,1;
node1:/cs!/ -> node1,          /* wait for start signal */
      /cs / si -> node2;       /* initialization */
node2:/ca!/ sd,sc -> node4,    /* shift and increment counter */
      /ca / sa -> node3;       /* add */
node3: sd,sc -> node4;
node4:/cf!/ -> node2,          /* next step */
      /cf / sd,sc -> node1.    /* last step */

Connectors;cs;ca;cf;si;sd;sc;sa.
End.
where: node1,..,node4 - functional states
       cs,ca,cf - external conditions
       si,sd,sc,sa - output control signals
```

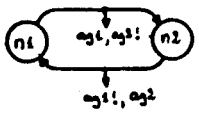


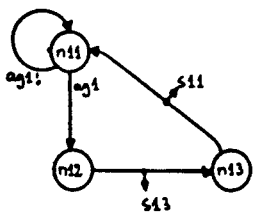

```
Register; sr(2).
Pla 1,1;[1]

cs,ca,cf,sr(1),sr(0) -
 sr(1),sr(0),si,sd,sc,sa;
(0xx 00) - (00 0000);
(1xx 00) - (01 1000);
(x0x 01) - (10 0110);
(x1x 01) - (11 0001);
(xxx 11) - (10 0110);
(xx0 10) - (01 0000);
(xx1 10) - (00 0110).
```

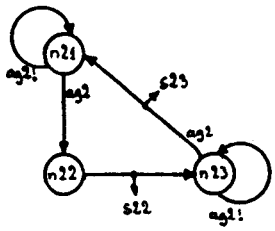

Fig.3.2. La description fonctionnelle (a), et la description structurelle (b) d'un séquenceur du circuit de multiplication.

2.2. L'approche "systémique"

L'approche "systémique" de la conception et de la description de l'architecture du matériel est l'une des caractèristiques les plus importantes non seulement du langage LIDO mais aussi du projet LIDO en général.
Cette approche, étant en opposition, à celle basée sur la vision algorithmique du fonctionnement du matériel, nécessite plusieurs moyens descriptifs inexistants dans les langages procéduraux et même non-procéduraux.
Un système matériel décrit de façon systémique est fortement stratifié en plusieurs sous-systèmes, celui de niveau supérieur contrôlant, en principe, le fonctionnement d'un sous-système soubordonné. Une telle description donne les moyens permettant de concevoir des systèmes de plus en plus complexes de façon rigoureuse et structurée.
Le langage LIDO possède plusieurs constructions conçues dans le cadre de l'approche systémique - hiérarchisée. Pratiquement chaque type de mécanisme de base du langage LIDO peut être stratifié en plusieur niveaux. De cette façon nous pouvons concevoir les graphes multiniveaux, les matrices logiques multiniveaux, et même les unités fonctionnelles multiniveaux, à l'exclusion de l'horloge qui est unique dans chaque description.
Ci-dessous nous présentons une description fonctionnelle qui comporte trois graphes étalés sur deux niveaux systémiques. Le graphe G 1,1; (pour le niveau 1 et le numéro 1) contrôle le fonctionnement de deux graphes subordonnés G 2,1; et G 2,2; (pour le niveau 2 et les numéros 1 et 2 sur ce niveau). Ce contrôle se fait à l'aide des signaux (variables) ag1 et ag2 positionnés par le graphe supérieur G 1,1; et testés par les graphes subordonnés G 2,1; et G 2,2; .



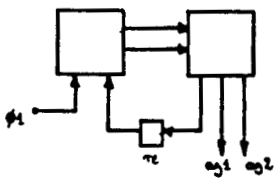




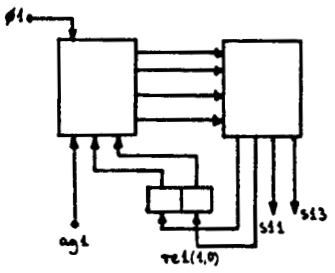

```
/* level 1 number 1 */
G 1,1;
n1: ag1,ag2! -> n2;
n2: ag1!,ag2 -> n1.

/* level 2 number 1 */
G 2,1;
n11:/ag1/ -> n12,
    /ag1!/ -> n11;
n12:s13 -> n13;
n13:s11 -> n11.

/* level 2 number 2 */
G 2,2;
n21:/ag2/ -> n22,
    /ag2!/ -> n21;
n22:s22 -> n23;
n23:/ag2/s23 -> n21,
    /ag2!/ -> n23.
```

Fig.3.3. La description fonctionnelle-multiniveaux d'un sous-système de séquencement

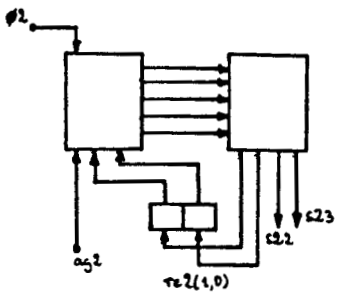
A la description fonctionnelle ci-dessus, exprimée par les graphes, correspond la description structurelle suivante. Cette fois-ci nous utilisons les séquenceurs conçus à partir de matrices logiques. Ces matrices sont stratifiées sur deux niveaux systémiques. La matrice P 1,1; (pour le niveau 1 et numéro 1 sur ce niveau) contrôle le fonctionnement de deux matrices subordonnées P 2,1; et P 2,2; (pour le niveau 2 et numéros 1 et 2 sur ce niveau).



```
/* level 1 number 1 */
 Reg;sr1(1).
P 1,1;[1]
sr1(0) -
sr1(0),ag1,ag2:
( 0 ) - ( 1 10);
( 1 ) - ( 0 01).
```



```
/* level 2 number 1 */
 Reg;sr21(2).
P 2,1;[2]
ag1,sr21(1),sr21(0) -
sr21(1),sr21(0),s13,s11;
(1 00 ) - ( 01 00);
(0 00 ) - ( 00 00);
(x 01 ) - ( 10 10);
(x 10 ) - ( 00 01).
```



```
/* level 2 number 2 */
 Reg;sr22(2).
P 2,2;[2]
ag2,sr22(1),sr22(0) -
sr22(1),sr22(0),s22,s23;
(1 00 ) - ( 01 00);
(0 00 ) - ( 00 00);
(x 01 ) - ( 10 10);
(1 10 ) - ( 00 01);
(0 10 ) - ( 10 00).
```


Fig.3.4. La description structurelle-multiniveaux d'un sous-système de séquencement.

Afin d'illustrer l'aproche systémique et morphologique de l'architecture des ordinateurs de façon plus détaillée et concrète nous présentons ci-dessous trois exemples qui lient les modèles globaux introduits dans le chapitre précedent avec leur expression detaillée en langage LIDO. Ce sont:

- le modèle et la description d'une unité arithmétique et logique (forme transitive de deuxième ordre)
- le modèle et la description d'une mémoire (forme transitive-mémorisante de deuxième ordre)
- le modèle et la description du circuit de multiplication (forme de troisième ordre)

DESCRIPTION D'UNE FORME TRANSITIVE DE DEUXIEME ORDRE

```
        File name : ilual
/* niveau 2 - transitif */
Pla 1,1;coop(1),coop(0);
(00) - dadd;            /* addition */
(01) - dmult;           /* multiplication */
(10) - dand;            /* and */
(11) - dwor.            /* wide-or */
/* niveau 1 - transitif , unit UAL sans registres */
Fun 1,1;;
/dadd/ bus1 + bus2 => cact;
/dmult/bus1 * bus2 => dact;
/dand/ bus1 & bus2 => cact;
/dwor/ bus1/; => tact.
Connectors; coop(2);dadd;dmult;dand;dwor;
            bus1(8);bus2(8);cact(9);dact(16);tact. /* ALU outputs */
End.
```

Fig.3.5. Description d'une forme transitive de deuxième ordre (unité arithmetique-logique)

DESCRIPTION D'UNE FORME TRANSITIVE/ MEMORISANTE DE DEUXIEME ORDRE

```
        File name : ilnen
/* niveau 2 , formes transitives, decodeur */
Pla 1,1;adr(1),adr(0);
(00) - dadr1;      /* register 1 */
(00) - dadr2;      /* register 2 */
(10) - dadr3;      /* register 3 */
(11) - dadr4.      /* register 4 */
/* niveau 1 , formes memorisantes , registres */
Fun 1,1;;
/dadr1/ r1 => bus;   /* selection of registers */
/dadr2/ r2 => bus;
/dadr3/ r3 => bus;
/dadr4/ r4 => bus;
Register; r1(8);r2(8);r3(8);r4(8).
Connectors;adr(2);bus(8).
End.
```

```
 commands: c,t,m,s,h,p,f,v ?  c

@r1=1111111;r2=1.

 commands: c,t,m,s,h,p,f,v ?  p

++    TRACE     cycle 2   phase 1    last phase 1 sec.
$ r1=>bus
$ r2=>bus
 adr         00
 dadr1       1
 dadr2       1
 dadr3       0
 dadr4       0
 r1          01111111
           ) 01111111
 r2          00000001
           ) 00000001
 r3          00000000
           ) 00000000
 r4          00000000
           ) 00000000
 bus         00000001
```

Fig.3.6. Description d'une forme transitive-mémorisante de deuxième ordre (mémoire)

DESCRIPTIONS D'UN ADDITIONNEUR/ SOUSTRACTEUR SERIEL (forme de troisième ordre)

```
      File name :  ilads
Graph 1,1;
n1:/st!/ -> n1,
   /st / sini -> n2;
n2:sadd -> n3;
n3:/end!/sshi -> n2,
   /end / -> n1.

Fun 1,1;C1];
/sini/ 0 => count;
/sini&coop!/ 0 => c;
/sini&coop / 1 => c.

Fun 1,2;C1];
/sadd&coop!/c^a(0)^b(0)  => s(7);
/sadd&coop /c^a(0)^b(0)! => s(7);
/sadd/ a(0)&b(0)=>ab,a(0)&c=>ac,b(0)&c=>bc,ab!ac!bc=>c,
      count+1=>count,count/&=>end;
/sshi/ a(7,1)=>a(6,0),b(7,1)=>b(6,0),s(7,1)=>s(6,0).

Reg;a(8);b(8);s(8);count(3);c.
Ter;end;coop.
Conn;st;sini;sadd;sshi;ab;ac;bc.
End.
```

NIVEAUX FONCTIONNELS-STRUCTURELS D'UN ADDITIONNEUR/SOUSTRACTEUR SERIEL

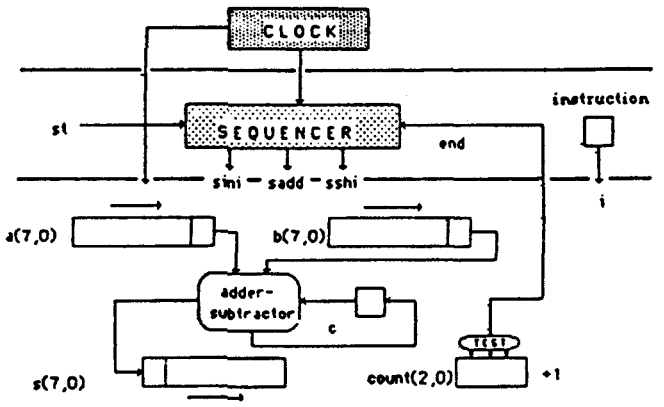
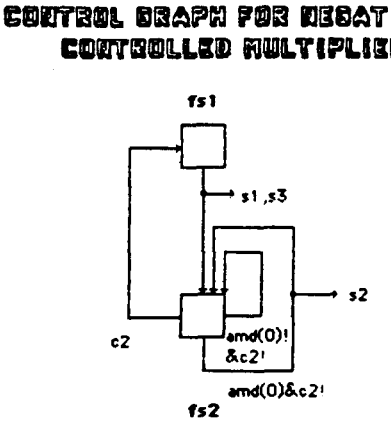


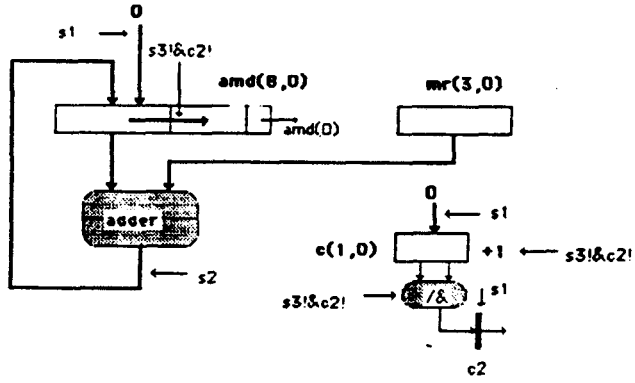
Fig.3.7. Description d'une forme de troisième ordre (circuit de multiplication)

Un autre aspect de l'approche systémique est constitué par le contrôle négatif. La spécification des sous-systèmes hautement autonomes, concurrents et contrôlés de façon négative [Bak83] implique l'existence d'unités fonctionnelles actives en l' absence de signaux de contrôle positif. Un signal de contrôle négatif (p.e. un signal de suspension ou d'inhibition) peut arrêter ou suspendre la ou les opérations normalement actives. Ci-dessous nous donnons la description en langage LIDO d'un simple exemple utilisant le principe du contrôle négatif. Cet exemple représente le fonctionnement du circuit de multiplication, dont la partie contrôle émet les signaux de contrôle interprétés au niveau opératif de façon positive et négative.




```
        File name :  neglan
/* negative control description */
/*** multiplier ***/
H;2.                        /* two phases */
/*** control path ***/
G 1,1;
fs1: s1,s3 -> fs2;      /* init.,shift and increment suppression! */
fs2:/amd(0)!&c2!/  -> fs2,     /* addition */
    /amd(0) &c2!/s2 -> fs2,
    /c2/ -> fs1.
```

Fig.3.8.a. Un exemple de l'utilisation du contrôle négatif décrit en langage LIDO : partie contrôle.




```
/*** data path ***/
F 1,1;[1];                      /* positively controlled data path */
/s1/ 0=> amd(8,4), 0=>c, 0=>c2 ;
/s2/ mr +amd(7,4) => amd(8,4).
F 1,2;[2];                      /* negatively controlled data path */
/s3!&c2!/ amd(8,1) => amd(7,0),    /* normally active operations */
      c+1 => c, amd(0)=>c1, c/&=>c2.
R; mr(4);amd(9);c(2).
C; s1;s2;s3.
T; c1;c2.
End.
```

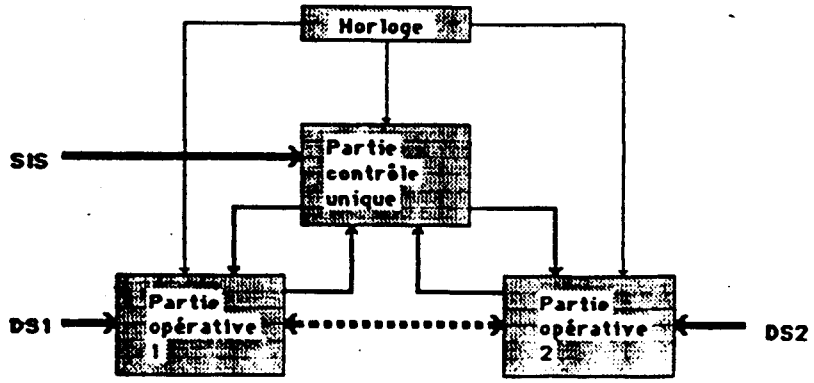

Fig.3.8.b. Un exemple de l'utilisation du contrôle négatif décrit en langage LIDO : partie opérative.

2.3. La conception de l'architecture du matériel évolué

Les systèmes VLSI et ULSI commencent à dépasser les étroites limites de l'architecture conventionnelle de von Neumann et évoluent vers les structures multi-processeurs, les architectures pipe-line et les architectures data-flow. En conséquence pour la conception de ces systèmes de nouveaux moyens de description et de modèlisation deviennent nécessaires. Etant donné que les langages classiques (CDL,DDL,RTS,CASSANDRE,OSM,...) de la description de l'architecture ont évolué avant l'introduction des systèmes intégrés VLSI, il est difficile d'en trouver un adapté à la conception de l'architecture évoluée.
Un des objectifs de développement du langage LIDO est de combler cette lacune. Afin d'illustrer les différentes possibilités d'utilisation du langage LIDO pour la description (conception) des systèmes évolués nous présenterons quelques schémas globaux et systémiques basés sur les mécanismes du langage LIDO. Ces schémas se limitent à la spécification des mécanismes sans le détail de leurs opérations ou/et structures internes. Un exemple complet et détaillé de l'architecture MIMD sera présenté dans le chapitre suivant.

2.3.1. L'architecture SIMD

Ci-dessous nous présentons le schéma d'une structure SIMD et le schéma global-systémique de la même structure en LIDO. Cette structure possède trois niveaux globaux: Horloge, Partie Contrôle et Partie Opérative. La Partie Contrôle et la Partie Opérative contiennent chacune deux niveaux locaux.

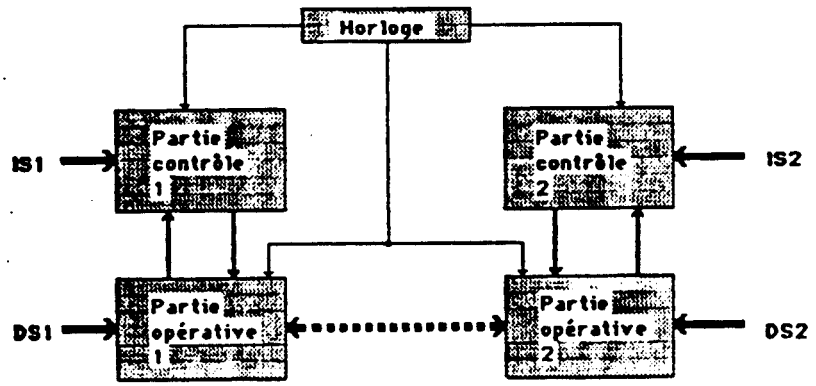



```
     Horloge;2,                /* niveau horloge */
-------------------------------------------------------------------
(a)
    Graphe 1,1;          /* niveau local 1 du contrôle - desc. fonctionnelle */
     ...
    Graphe 2,1;          /* niveau local 2 du contrôle - desc. fonctionnelle */
(b)
    Graphe 1,1;          /* niveau local 1 du contrôle - desc. fonctionnelle */
     ...
    Pla 2,1;...          /* niveau local 2 du contrôle - desc. structurelle  */
(c)
    Pla 1,1;...          /* niveau local 1 du contrôle - desc. structurelle  */
     ...
    Pla 2,1;...          /* niveau local 2 du contrôle - desc. structurelle  */
-------------------------------------------------------------------
Fun 1,10;...  Fun 1,14;                Fun 1,20;...  Fun 1,24;...
   /* niveau local 1 de parties opératives 1 et 2 */
Fun 2,10;...  Fun 2,14;                Fun 2,20;...  Fun 2,24;...
   /* niveau local 2 de parties opératives 1 et 2 */
```

Fig.3.9. Schéma global-systémique d'une architecture SIMD à deux unités opératives: (a) description fonctionnelle de la partie contrôle, (b) description mixte de la partie contrôle; (c) description structurelle de la partie contrôle.

2.3.2. L'architecture MIMD

Ci-dessous nous présentons le schéma d'une structure MIMD et le schéma global de la même structure en LIDO. Cette architecture possède les mêmes Parties Opératives que celles présentées pour l'architecture SIMLD, cette fois-ci chacune d'elles est contrôlée par une Partie Contrôle séparée.

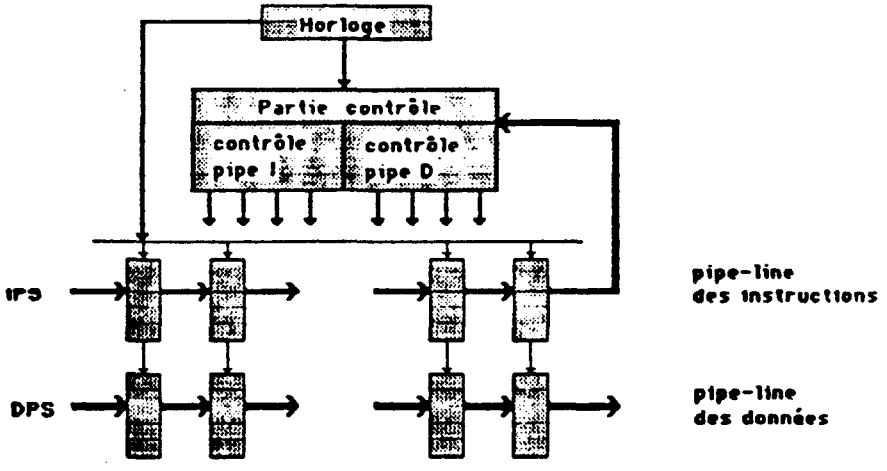



```
    Horloge;2,                  /* niveau horloge */
-------------------------------------------------------------------------------
    Graphe 1,1;                 /* niveau de "super-contrôle" */
-------------------------------------------------------------------------------
Pla 1,10;...                      Pla 1,20;...
Pla 2,10;...                      Pla 2,20;...

/* unité de contrôle 1 */        /* unité de contrôle 2 */
-------------------------------------------------------------------------------
Fun 1,10;...                      Fun 1,20;...
Fun 2,10;...                      Fun 2,20;...

/* unité opérative 1   */        /* unité opérative 2   */
```

Fig.3.10. Schéma global-systémique d'une architecture MIMD à deux unités de contrôle et à deux unités opératives.

2.3.3. L'architecture de type pipe-line

Ci-dessous nous donnons le schéma d'une structure pipe-line et le schéma global en LIDO correspondant. La Partie Contrôle est composée de deux structures : l'une destinée au contrôle du pipe-line des instructions, l'autre au contrôle du pipe-line des données.



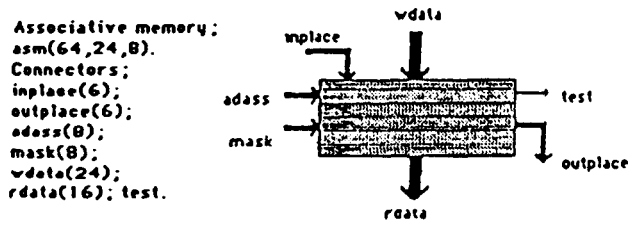

```
    Horloge;6,
-------------------------------------------------------------------------------
    Graphe 1,1;              /* contrôle global */
-------------------------------------------------------------------------------
Pla 1,10;...                                  Pla 1,20;...
...                                           ...
Pla 2,10;...                                  Pla 2,20;...
/* contrôle du pipe-line des instructions */  /* contrôle du pipe-line des données */
```

Fig.3.11. Schéma global-systémique d'une architecture pipe-line à deux pipe-lines contrôlés par deux sous-unités de contrôle spécialisées.

2.3.4. Les architectures basées sur les mémoires associatives

L'utilisation de la mémoire associative devient nécessaire si on veut concevoir certains systèmes de mémoire virtuelle ou l'architecture data-flow (p.e. matching-box). On ne trouve pratiquement pas de primitives "mémoire associative" dans les langages classiques pour la description du matériel. Un tel mécanisme est disponible en langage LIDO. Une mémoire associative déclarée dans une description LIDO possède une fonction d'écriture et trois fonctions de lecture. Nous les illustrons ci-dessous par un exemple.

DECLARATION ET UTILISATION DE LA MEMOIRE ASSOCIATIVE



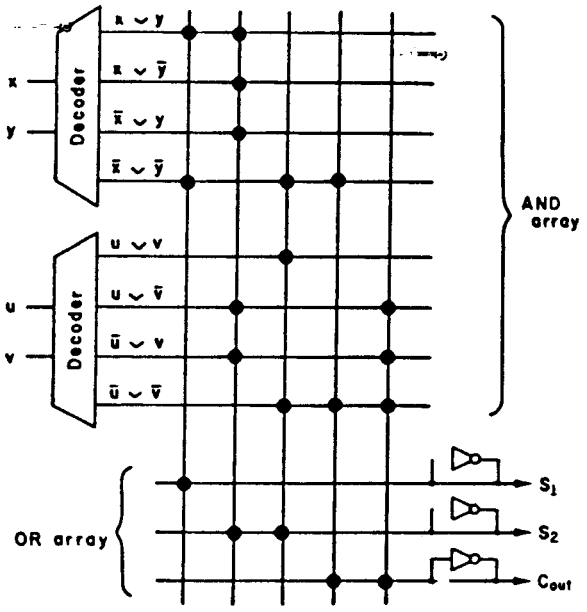

```
Fun 1,1;
/asswr/ wdata=>asm(@inplace);       /*write data*/
/assrd/ asm(adass&mask,$)=>rdata;  /*associative read data*/
/assrp/ asm(adass&mask,@)=>outplace; /*associative read placement*/
/assrt/ asm(adass&mask,?)=>test.    /*associative read test*/
```

Fig.3.12. L'exemple de la déclaration et de l'utilisation d'une mémoire associative.

2.4. L'utilisation des mécanismes fonctionnels et structurels correspondants aux sous-systèmes topologiques.

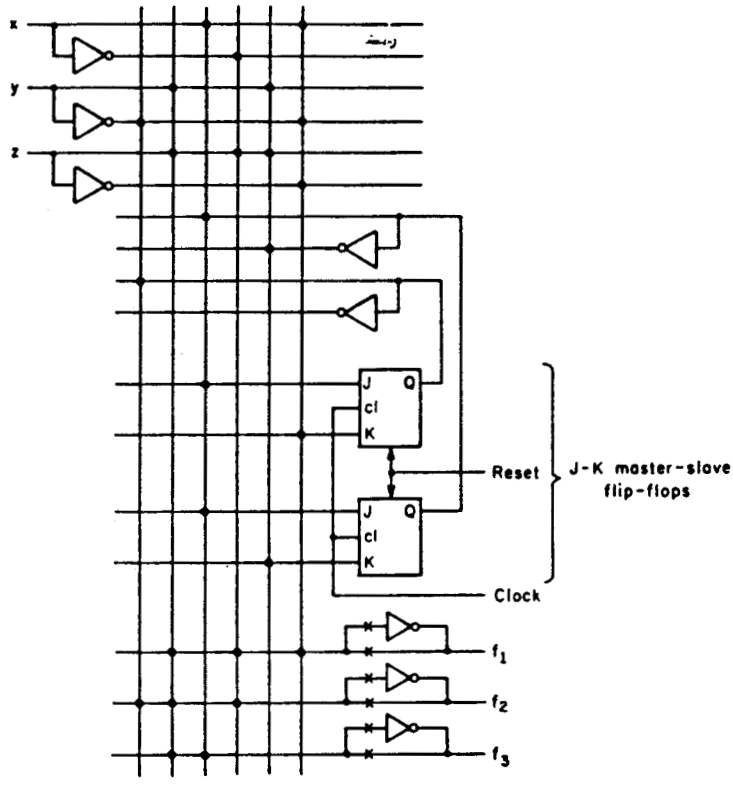
Une des plus importantes caractéristiques d'un langage pour la conception de l'architecture du matériel orientée vers les systèmes VLSI est la disponibilité des mécanismes fonctionnels et structurels conformes aux sous-systèmes topologiques. Les sous-systèmes topologiques tels que les mémoires vives RAM, les mémoires mortes ROM et les différents types de matrices logiques PLAs, SLAs, MGAs constituent les composants de base pour la conception et la réalisation des systèmes VLSI. Leur utilisation actuelle, qui est de l'ordre de 60% de la surface du circuit intégré, dépassera dans les années à venir 70% de cette surface [Mur81]. D'où la nécessité évidente de la présence de telles structures dans un langage moderne pour la conception du matériel. Il est important de noter que les matrices logiques ne sont pas introduites dans les langages classiques RTL tels que : CDL, DDL, CASSANDRE, RTS I ou OSM, en tant que mécanisme de base.
Afin de donner quelques utilisations possibles de matrices logiques disponibles en langage LIDO nous présentons trois exemples :

- matrice logique PLA (Programmable Logic Array), additionneur deux bits
- matrice logique SLA (Storage Logic Array)
- matrice logique MGA (Multilevel Gate Array)



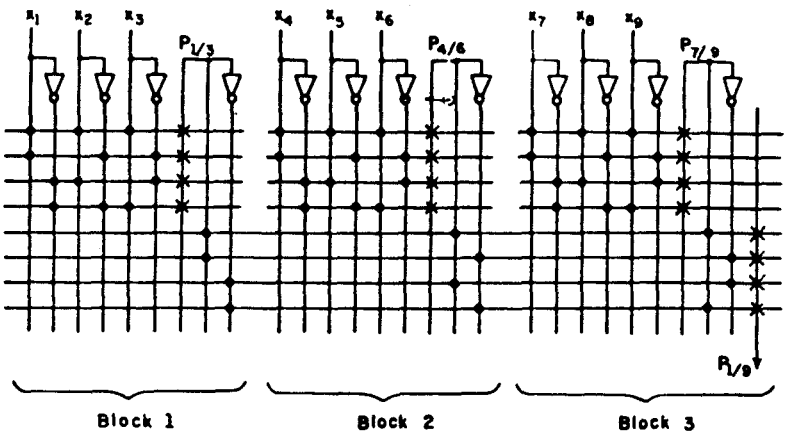

```
/* a 2-bit adder realized in decodec PLA */

Pla 1,1;[1] x,y;                /* xy decoder */
(00) - t(0);
(01) - t(1);
(10) - t(2);
(11) - t(3).

Pla 1,2;[1] u,v;                /* uv decoder */
(00) - z(0);
(01) - z(1);
(10) - z(2);
(11) - z(3).

Pla 2,1;[1] t(0),t(1),t(2),t(3),z(0),z(1),z(2),z(3);
(1xx1xxxx) - s1;
(111x11xx) - s2;
(xxx11xx1) - s2;
(xxxxx111) - cout!.

Connectors;x;y;u;v;t(4);z(4);s1;s2;cout.
End.
```

Fig.3.13. Matrice logique programmable PLA, exemple d'un additionneur deux bits [Mur81].




```
/* BLA  - Pla sequencer */

Pla 1,1;212 x,y,z,sr(1),sr(0);
(x0xx1) - f2;
(x11xx) - f1,f2,f3;
(1xx1x) - sr(1),sr(0),f2;
(0x1xx) - f1,f2;
(x110x) - sr(0);
(100xx) - sr(1),f1;
```

Fig.3.14. Matrice logique avec mémoire, exemple d'un séquenceur [Mur81].




```
/* MGA with three associative matrices */

Pla 1,1;[1] x1,x2,x3;
(111) - p1;
(100) - p1;
(010) - p1;
(001) - p1.

Pla 1,2;[1] x4,x5,x6;
(111) - p4;
(100) - p4;
(010) - p4;
(001) - p4.

Pla 1,3;[1] x7,x8,x9;
(111) - p7;
(100) - p7;
(010) - p7;
(001) - p7.

Pla 2,1;[1] p1,p4,p7;
(111) - p9;
(100) - p9;
(010) - p9;
(001) - p9.

Connectors; x1;x2;x3;x4;x5;x6;x7;x8;x9;
            p1;p4;p7;p9.
```

Fig.3.15. Matrice multiniveaux de portes logiques MGA, exemple d'une structure répétitive [Mur81].

2.5. La spécification des entrées et des sorties.

La spécification structurelle des systèmes VLSI doit tenir compte des différents types de signaux qui sortent ou entrent dans un système intégré [Par81]. Ces signaux peuvent être véhiculés par différents types de porteurs associés ou non aux capacités de mémoire. Par exemple les signaux qui doivent être "ratrappés" par le système intégré peuvent être verrouillés par un terminal équipé d'un verrou, pendant que le signal sortant et autorisant l'adresse d'une mémoire externe peut être concidéré, de la part du circuit, comme un signal-impulsion sans mémoire.
Ci-dessous nous présentons comme exemple un fragment de la structure entrée/sortie d'un processeur intégré. Le langage LIDO offre trois types de porteurs d'information simples:

- les connecteurs (porteurs sans mémoire)
- les terminaux (porteurs dotés d'une capacité de mémoire transitive)
- les registres (porteurs dotés de points mémoire double de type maître-esclave)

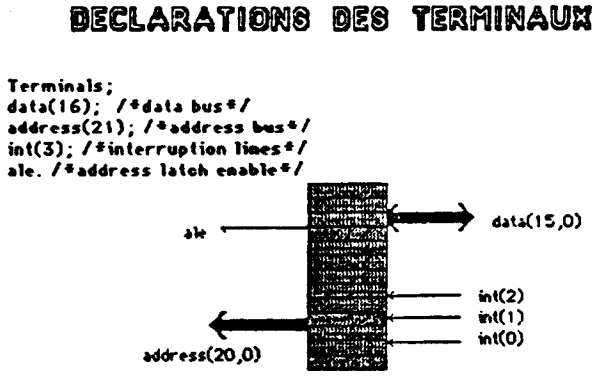


Fig.3.16. Un exemple de spécification des entrées/sorties d'un système intégré

3. Description comparative des processeurs de type RISC

Les processeurs de type RISC (Reduced Instruction Set Computer) [Patt85] en raison de leur architecture simple et régulière qui permet d'accélérer la vitesse d'exécution des instructions simples, commencent à envahir le marché du matériel informatique. Par ailleurs ces processeurs ayant une structure interne extrêmement régulière se prêtent très bien à la conception de VLSI réalisée par compilation en silicium [Sil85].
Afin d'illustrer de façon complète et variée les possibilités descriptives du langage LIDO nous allons présenter successivement la conception et la description de plusieurs versions d'un processeur RISC et d'un bi-processeur du même type. Imaginons un processeur "np32" basé sur une seule unité de traitement dont la largeur est de 32 bits. Les différentes varietés de ce processeur sont dues aux différentes versions de la partie contrôle.

```
SYNCHRONISATION  - Horloge
-------------------------------------------------

SEQUENCEMENT     - Graphes, PLAs, microcode
-------------------------------------------------

DECODAGE         - Graphes, Plas, microcode
-------------------------------------------------

TRAITEMENT       - Unités fonctionnelles
```

Fig.3.17. Parties principales (niveaux globaux) du processeur np32.

Les différentes versions de notre processeur - np32 vont être présentées dans "l'ordre de synthèse". La première description de la partie contrôle est fonctionnelle alors que les dernières sont structurelles. Les versions

intermédiaires sont de caractère mixte. Les différentes parties contrôle du np32 seront décrites après le paragraphe suivant, lequel est reservé à la présentation de la partie opérative unique pour chaque description complète.

3.1. Partie opérative du np32

La partie opérative du np32 a été conçue afin de pouvoir traiter des données de 32 bits et des adresses de 20 bits. Le jeu d'instructions fortement réduit permet de simplifier la structure des chemins de données et d'utiliser un seul format d'instructions.
La structure de la partie opérative est décomposée en quatre unités fonctionnelles :

- unité des transferts par bus; Fun 1,2;
- unité arithmétique et logique; Fun 1,3;
- unité de test arithmétiques; Fun 1,4;
- unité d'accumulateur temporaire; Fun 1,5;

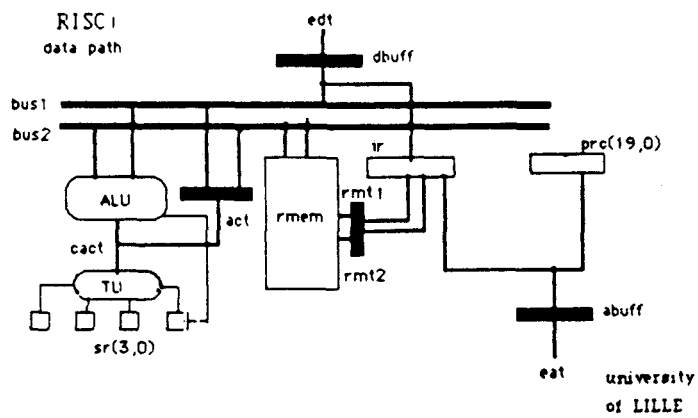


Fig.3.18.a. Schéma graphique de la partie opérative du np32.

```
/* Description de Partie Operative  NIVEAU 00  */

Fun 1,3; [2];       /* unite arithmetique et logique */

/dadd/ bus1 + bus2 => cact;
/dadr/ bus1 + bus2 + r => cact;
/dsub/ bus1 - bus2 => cact;
/dsur/ bus1 - bus2 - r => cact;
/dou/  bus1 ! bus2 => cact(31,0);
/det/  bus1 & bus2 => cact(31,0);
/dex/  bus1 ^ bus2 => cact(31,0);
/ddif/ bus1 - bus2 => cact;
/dded/ r => cact(31), bus1(31,1) => cact(30,0), bus1(0) => r;
/ddeq/ r => cact(0), bus1(30,0) => cact(31,1), bus1(31) => r;
/ddar/ bus1(31) => cact(31), bus1(31,1) => cact(30,0), bus1(0) => r;
/dcoa/ bus1(31,0)! => cact(31,0);
/dinc/ bus1 + 1 => cact;
/ddec/ bus1 - 1 => cact;
/dpas/ bus1 => cact(31,0).

Fun 1,4; [2];       /* unite de tests arithmetiques */

/dzer/ 0 => r, 0 => s, 0 => d, 0 => z ;
/dtzs/ cact(31) => s, cact /! => z!;
/dtd/  bus1(31) & bus2(31) & cact(31)! => cdn,   /* debord. negatif */
       bus1(31)!& bus2(31)!& cact(31)  => cdp,   /* debord. positif */
       cdn ! cdp => d;
/dtr/  cact(32) => r.

Fun 1,5; [2];       /* chargement de l'accumulateur temporaire act(31,0) */

/sual/ cact(31,0) => act.

Fun 1,2; [2];       /* unite des transferts par bus */

/sa/   prc => eat(19,0), emem(@eat) => ir, prc + 1 => prc;
/sual/ ir(23,20) => rat1, ir(19,16) => rat2,
       rmem(@rat1) => bus1, rmem(@rat2) => bus2;
/schr & ddif!/ act => bus1, bus1 => rmem(@rat1);
/scon/ ir(19,0) => prc;                    /* adresse de saut */
/ses1 & ir(24)!/ 1 => erd, ir(19,0) => eat(19,0), emem(@eat) => bus1;
/ses1 & ir(24) / ir(23,20) => rat1, rmem(@rat1) => bus1;
/ses2 & ir(24)!/ ir(23,20) => rat1, bus1 => rmem(@rat1);
/ses2 & ir(24) / 1 => ewr, ir(19,0) => eat(19,0), bus1 => emem(@eat).

Memoires; rmem(16,32);      /* memoire de registres interne */
          emem(32,32).      /* memoire de programme externe */

Registres; ir(32);          /* registre d'instructions */
           re(2);           /* registre d'etats        */
           s;d;r;z;         /* reg. drapeaux           */
           prc(20).         /* compteur de programme   */

Terminaux;  bus1(32); bus2(32);        /* bus internes */
            act(32);          /* accumulateur temporaire */
            rat1(4); rat2(4);  /* adresses de registres */
            eat(20);    /* external terminal d'adresses */
            sa;sual;scon;ses1;ses2;schr;
            dadd;dadr;dsub;dsur;dou;det;
            dex;ddif;dded;ddeg;ddar;dcoa;
            dinc;ddec;dpas;dzer;dtzs;dtr;
            dtd.

Connecteurs; dcp;
             erd;       /*external externe de lecture*/
             ewr;       /*terminal externe d'ecriture*/
             cact(33);      /*sortie de UAL */
             cdn;cdp.

End.
```

Fig.3.18.b. La description de la partie opérative du np32 en langage LIDO avec les déclarations des porteurs .

3.2. Partie contrôle du np32 - description fonctionnelle

La description fonctionnelle de la partie contrôle est exprimée à l'aide d'un graphe de contrôle. Ce graphe génère tous les signaux de contrôle destinés à la partie opérative. Il englobe les fonctions du séquencement et du décodage des instructions.

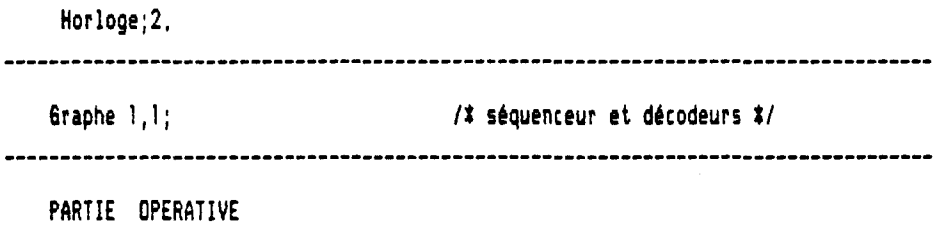


Fig.3.19. Schéma global-systémique de la description fonctionnelle de la partie contrôle du np32.

```
Graph 1,1;      /*functional description of sequencer and decoders*/
n1:sf->n2;                                      /*fetch*/
n2:/ir(29)!&ir(28)&ir(26)!/sc(0)->n1,           /*jump */
   /ir(29)!&ir(28)&ir(26)&ir(25)!&ir(24)!/sc(1)->n1,
   /ir(29)!&ir(28)&ir(26)&ir(25)!&ir(24) /sc(2)->n1,
   /ir(29)!&ir(28)&ir(26)&ir(25) &ir(24)!/sc(3)->n1,
   /ir(29)!&ir(28)&ir(26)&ir(25) &ir(24) /sc(4)->n1,
   /ir(29)!&ir(28)!&ir(27)!&ir(26)!&ir(25)!&ir(24)!/sa(0),st1->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26)!&ir(25)!&ir(24) /sa(1),st1->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26)!&ir(25) &ir(24)!/sa(2),st1->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26)!&ir(25) &ir(24) /sa(3),st1->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26) &ir(25)!&ir(24)!/sa(4),st2->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26) &ir(25)!&ir(24) /sa(5),st2->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26) &ir(25) &ir(24)!/sa(6),st2->n3,
   /ir(29)!&ir(28)!&ir(27)!&ir(26) &ir(25) &ir(24) /sa(7),st1->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26)!&ir(25)!&ir(24)!/sa(8),st2->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26)!&ir(25)!&ir(24) /sa(9),st2->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26)!&ir(25) &ir(24)!/sa(10),st2->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26)!&ir(25) &ir(24) /sa(11),st2->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26) &ir(25)!&ir(24)!/sa(12),st1->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26) &ir(25)!&ir(24) /sa(13),st1->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26) &ir(25) &ir(24)!/sa(14)   ->n3,
   /ir(29)!&ir(28)!&ir(27) &ir(26) &ir(25) &ir(24) /sa(15),sstz->n3,
   /ir(29) &ir(28)!&ir(27)!/slo(0)->n4,     /*load instruction 1st step*/
   /ir(29) &ir(28)!&ir(27) /sto(0)->n4;
n3:slr->n1;                                     /*load ALU result*/
n4:/ir(27)!/slo(1)->n1,
   /ir(27) /sto(1)->n1.
Terminals;sc(5);sa(16);sf;st1;st2;slr;slo(2);sto(2).
Registers;ir(32).
End.
```

Fig.3.20. La description fonctionnelle de la partie contrôle du np32 en langage LIDO.

3.3. Partie contrôle du np32 - description mixte

Ci-dessous nous présentons un schéma global et une description fonctionnelle-structurelle de la partie contrôle du np32. Le séquenceur est décrit fonctionnellement à l'aide du graphe de contrôle, le niveau de décodage est exprimé par une structure PLA.

```
Horloge;2.
-------------------------------------------------------------------------------
Graphe 1,1;                          /* séquenceur */
-------------------------------------------------------------------------------
Pla 1,1;...                           /* décodeur de UAL */
Pla 1,2;...                           /* décodeur des sauts */
-------------------------------------------------------------------------------
PARTIE OPERATIVE
```

Fig.3.21. Schéma global-systémique de la description mixte de la partie contrôle du np32.

```
      Graphe 1,1;

node1:/am1/ sa -> node2,        /* acquisition d'une instruction */
      /am1!/ -> node1;           /* "wait" */
node2:/ir(29)! & ir(28)!/ sual -> node3,  /* activation de UAL */
      /ir(29)! & ir(28) / scon -> node1,  /* act. sauts cond. */
      /ir(29) & ir(28)! & am1 / ses1 -> node4,  /* act. E/S */
      /ir1(29) & ir1(28)! & am1!/ -> node21;        /*  " wait" */
node3:/am1/ schr,sa -> node2, /* charg.d'un reg.& "fetch overlap"*/
      /am1!/schr -> node1;    /* charg.d'un reg. sans "fetch" */
node4:/am1/ses2 -> node1,     /* act. E/S */
      /am1!/ -> node4.

   /* Matrice logique - decodage des operations UAL */
                                                    /* Pla simple code */
    Pla 1,1; [1] /sual/ ir(27),ir(26),ir(25),ir(24);

    (0000) - dadd,dtzs,dtr,dtd;       /* signal d'addition */
    (0001) - dadr,dtzs,dtr,dtd;       /* - d'addition avec retenue */
    (0010) - dsub,dtzs,dtr,dtd;       /* - de soustraction */
    (0011) - dsur,dtzs,dtr,dtd;       /* - de soustr. avec retenue */
    (0100) - dou,dtzs;                /* - de Ou */
    (0101) - det,dtzs;                /* - de ET */
    (0110) - dex,dtzs;                /* - de EXOU */
    (0111) - dcif,dtzs,dtr,dtd;       /* - de difference */
    (1000) - ddec,dtzs;               /* - de decalage a droite */
    (1001) - ddeg,dtzs;               /* - de decalage a gauche */
    (1010) - ddar,dtzs;               /* - de decalage arith. a dr.*/
    (1011) - dcom,dtzs;               /* - de complementation */
    (1100) - dinc,dtzs,dtr,dtd;       /* - d'incrementation */
    (1101) - ddec,dtzs,dtr,dtd;       /* - de decrementation */
    (1110) - dpas;                    /* - de transfert */
    (1111) - dzer.                    /* mise a zero des indicateurs */
  /* dtzs -test zero-signe, dtr -test retenue, dtd -test debordement */

  /* Matrice logique - decodage des codes de sauts conditionnels */

    Pla 2,1;[1] /scon/ ir(26),ir(25),ir(24),r,s,d,z - dcp;        /* Pla double code */

  (10xxxxxx) - (1);                 /* saut inconditionnel */
  (11001xxx) - (1);                 /* saut si retenue = 1 */
  (1101x1xx) - (1);                 /* saut si signe = 1   */
  (1110xx1x) - (1);                 /* saut si debordement = 1 */
  (1111xxx1) - (1).                 /* saut si zero = 1 */
```

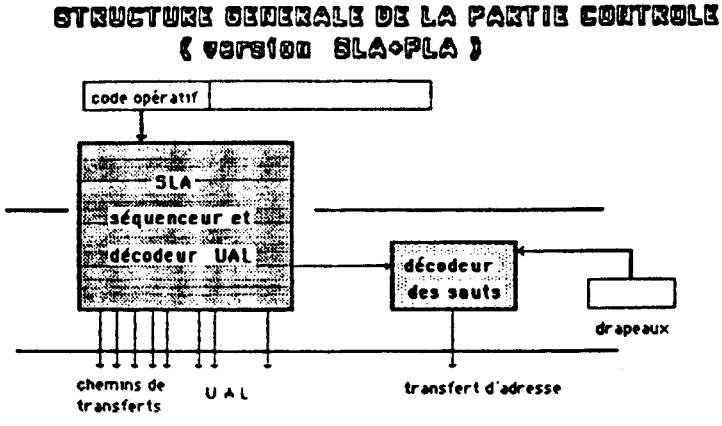
Fig.3.22. La description mixte de la partie contrôle du np32 en langage LIDO.

3.4. Partie contrôle du np32 - descriptions structurelles

La dernière étape de la synthèse (dans le cadre du langage LIDO) de la partie contrôle consiste à développer la structure complète de cette partie. Ci-dessous nous présentons deux structures développées à partir de la description fonctionnelle et de la description mixte. La première structure complète de la partie contrôle du np32 est une implémentation en matrice unique PLA du séquenceur et du décodeur de l'UAL. Les fonctions de décodage des sauts conditionnels sont implémentées en une matrice logique supplémentaire.

```
Horloge;2,
-------------------------------------------------------------------------------

Pla 1,1;...                    /* Pla séquenceur et décodeur de l'UAL */
Pla 2,1;...                    /* Pla décodeur des sauts */
-------------------------------------------------------------------------------

PARTIE OPERATIVE
```

Fig.3.32. Schéma global-systémique de la description structurelle de la partie contrôle du np32; version PLA unique.



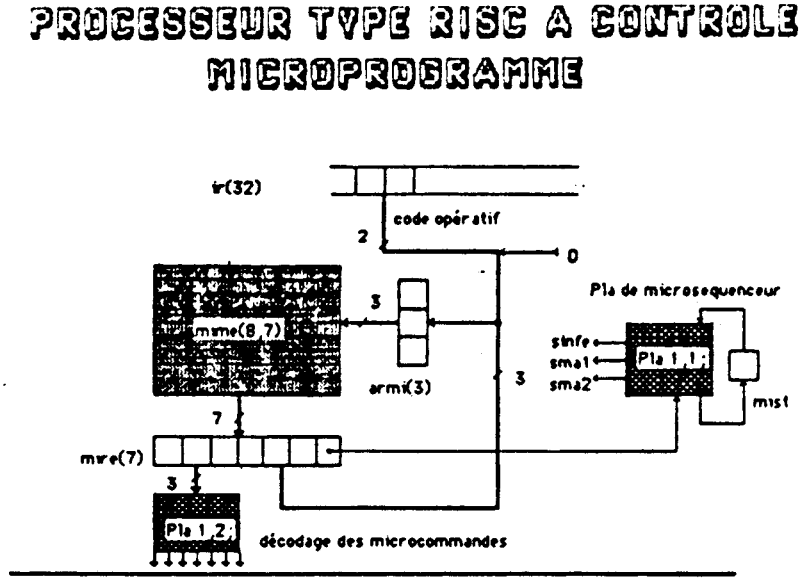
```
Horloge; 2.          /* horloge a deux phases  #NIVEAU 3# */

/* description structurelle de sequencement  #NIVEAU 2# */

Pla 1,1;[1] ir(29),ir(28),ir(27),ir(26),ir(25),ir(24),re(1),re(0) -
   re(1),re(0),  sa,sual,  dtzs,dtr,dtd,dzer,  dadd,dadr,dsub,dsur,
   dou,det,dex,ddif,  dded,ddeg,ddar,dcom,  dinc,ddec,dpas,dzer,
   schr,scon,ses1,ses2;
  (xx xxxx 00) - (01 10 0000 0000 0000 0000 0000 0000);  /* "fetch"       */
  (00 0000 01) - (10 01 1110 1000 0000 0000 0000 0000);  /* addition      */
  (00 0001 01) - (10 01 1110 0100 0000 0000 0000 0000);  /* add avec ret. */
  (00 0010 01) - (10 01 1110 0010 0000 0000 0000 0000);  /* soustraction  */
  (00 0011 01) - (10 01 1110 0001 0000 0000 0000 0000);  /* sous avec ret */
  (00 0100 01) - (10 01 1000 0000 1000 0000 0000 0000);  /* OU            */
  (00 0101 01) - (10 01 1000 0000 0100 0000 0000 0000);  /* ET            */
  (00 0110 01) - (10 01 1000 0000 0010 0000 0000 0000);  /* EXOU          */
  (00 0111 01) - (10 01 1110 0000 0001 0000 0000 0000);  /* difference    */
  (00 1000 01) - (10 01 1000 0000 0000 1000 0000 0000);  /* decal.a dr.   */
  (00 1001 01) - (10 01 1000 0000 0000 0100 0000 0000);  /* decal.a g.    */
  (00 1010 01) - (10 01 1000 0000 0000 0010 0000 0000);  /* decal.aritha. */
  (00 1011 01) - (10 01 1000 0000 0000 0001 0000 0000);  /* complement.   */
  (00 1100 01) - (10 01 1110 0000 0000 0000 1000 0000);  /* incrementation*/
  (00 1101 01) - (10 01 1110 0000 0000 0000 0100 0000);  /* decrementation*/
  (00 1110 01) - (10 01 0000 0000 0000 0000 0010 0000);  /* transfert     */
  (00 1111 01) - (10 01 0000 0000 0000 0000 0001 0000);  /* 0=>drapeaux   */
  (00 xxxx 10) - (01 10 0000 0000 0000 0000 0000 1000);  /* overlapping   */
  (01 xxxx 01) - (00 00 0000 0000 0000 0000 0000 0100);  /* sauts cond.   */
  (10 xxxx 01) - (11 00 0000 0000 0000 0000 0000 0010);  /* E/S 1-er pas  */
  (10 xxxx 11) - (00 00 0000 0000 0000 0000 0000 0001).  /* E/S 2-eme pas */

/* Pla - decodage des codes de sauts conditionnels  #NIVEAU 1# */

Pla 1,2;[2] /scon/ ir(26),ir(25),ir(24),r,s,d,z - dcp; /*Pla double code*/
  (0xx xxxx) - (1);      /* saut inconditionnel */
  (100 1xxx) - (1);      /* saut si retenue = 1 */
  (101 x1xx) - (1);      /* saut si signe = 1   */
  (110 xx1x) - (1);      /* saut si debord. = 1 */
  (111 xxx1) - (1).      /* saut si zero = 1    */
```

Fig.3.33. La description structurelle de la partie contrôle du np32 en langage LIDO; version "PLA unique".

La deuxième structure représente une partie contrôle microprogrammée basée sur une mémoire de microprogrammes - mime(8,7) et deux matrices logiques: Pla 1,1; - microséquenceur et Pla 1,2; - décodeur des microcommandes.

```
Horloge;2,
-------------------------------------------------------------------------------

Pla 1,1;...                    /* micro-séquenceur */
Pla 1,2;...                    /* décodeur des microcommandes */
Pla 2,1;...                    /* décodeur de l'UAL */
Pla 2,2;...                    /* décodeur des sauts conditionnels */
-------------------------------------------------------------------------------

Fun 1,1;...
          /* transferts des micro-instructions et des micro-adresses */

PARTIE OPERATIVE
```


Fig.3.34. Schéma global-systémique de la description structurelle de la partie contrôle du np32; version microprogrammée.



```
Horloge!2.

Pla 1,1! mire(0),mist - mist,sinfe,sma1,sma2!
                ( x0 )    -   ( 1100 )!
                ( 01 )    -   ( 0010 )!
                ( 11 )    -   ( 0001 ).

Pla 1,2! [2] mire(6),mire(5),mire(4)!
(xxx) - sa!,sual!,scon!,schr!,ses1!,ses2!!
(000) - sa!
(001) - sual!
(010) - schr!
(011) - scon!
(100) - ses1!
(101) - ses2.

Fun 1,1!!
/sinfe/ mime(àarmi) => mire(6,0)!
/sma1/ mire(3,1) => armi(2,0)!
/sma2/ ir(29,28) => armi(2,1), 0 => armi(0).


Registers!ir(32)! /* instruction register*/
    mist !   /* microstate */
    armi(3). /* microinstr. address register */

Memory!mime(8,7).   /* micro-memory 8 words 7 bits */

Connectors! sinfe!sma1!sma2.

Terminals!armi(3)!      /* microaddress latch */
          mire(7)!  /* microinstruction latch */
          sa!sual!scon!schr!ses1!ses2.


  /* matrice logique - décodage des operations UAL */

    Pla 2,1! [2] /sual/ ir(27),ir(26),ir(25),ir(24)!    /*  Pla simple code */

(0000) - dadd,dtzs,dtr,dtd!                 /* signal d'addition */
(0001) - dadr,dtzs,dtr,dtd!                 /*   - -  d'addition avec retenue */
(0010) - dsub,dtzs,dtr,dtd!                 /*   - -  de soustraction */
(0011) - dsur,dtzs,dtr,dtd!                 /*   - -  de soustraction avec ret */
(0100) - dou,dtzs!                          /*   - -  de OU */
(0101) - det,dtzs!                          /*   - -  de ET */
(0110) - dex,dtzs!                          /*   - -  de EXOU */
(0111) - ddif,dtzs,dtr,dtd!                 /*   - -  de difference */
(1000) - dded,dtzs!                         /*   - -  de decalage a droite */
(1001) - ddeg,dtzs!                         /*   - -  de decalage a gauche */
(1010) - ddar,dtzs!                         /*   - -  de decalage arith. a dr. */
(1011) - dcom,dtzs!                         /*   - -  de complementation */
(1100) - dinc,dtzs,dtr,dtd!                 /*   - -  d'incrementation */
(1101) - ddec,dtzs,dtr,dtd!                 /*   - -  de decrementation */
(1110) - dpas!                              /*   - -  de transfert */
(1111) - dzer.                              /*  mise a zero des indicateurs */
/* dtzs -test de zero-signe, dtr -test de retenue, dtd -test de debordement */


     /* matrice logique - décodage des codes de sauts conditionnels */

 Pla 2,2! [2] /scon/ ir(26),ir(25),ir(24),r, s,d,z - dcp!  /* Pla double code */

    (10xxxxxx) - (1)!                       /* saut inconditionnel */
    (11001xxx) - (1)!                       /* saut si retenue = 1 */
    (1101x1xx) - (1)!                       /* saut si signe = 1   */
    (1110xx1x) - (1)!                       /* saut si débordement = 1 */
    (1111xxx1) - (1).                       /* saut si zero = 1 */
```

Fig.3.35. La description structurelle de la partie contrôle du np32 en langage LIDO; version microprogrammée.

3.5. Le bi-processeur de type RISC - np32x2

La description d'un bi-processeur RISC (closely coupled processors) a été développée afin de présenter la description en langage LIDO des architectures de type MIMD. Nous avons modifié la partie contrôle du processeur de base np32 de telle façon quelle puisse être influencée de l'exterieur par le niveau de "super-contrôle", qui dans ce cas devient l'arbitre d'accès à la mémoire commune emem(32,32). Le fonctionnement de cet arbitrage, étant extrêmement simple, permet d'organiser l'accès à la mémoire en "temps partagé" en générant une fois le signal am1 (autorisation d'accès au processeur 1) l'autre fois le signal am2 (autorisation d'accès au processeur 2).

```
   Horloge;2.
-------------------------------------------------------------------------------

   Graphe 1,1;          /* super-contrôle, l'arbitrage d'accès à la mémoire commune */
-------------------------------------------------------------------------------

Graphe 2,1;          /* + séquenceurs 1 et 2 + */          Graphe 2,2;
Pla 1,11;...         /* + décodeurs UAL1 et UAL2 + */       Pla 1,12;...
Pla 2,11;...         /* + décodeurs des sauts 1 et 2 + */   Pla 2,12;...

/* partie contrôle 1 */                                    /* partie contrôle 2 */
-------------------------------------------------------------------------------

Fun 1,11;...      /* + transferts par bus + */              Fun 1,12;...
Fun 1,21;...      /* + unités arithmétiques et logiques + */ Fun 1,22;...
Fun 1,31;...      /* + unités de test arithmétiques + */    Fun 1,32;...
Fun 1,41;...      /* + unités d'accumulateur temporaire + */ Fun 1,42;...

/* partie opérative 1 */                                   /* partie opérative 2 */

                  /* [emem] - mémoire commune emem */
```

Fig.3.36. Schéma global-systémique de bi-processeur np32x2

a.

```
        Horloge; 2.          /* horloge a deux phases */

     /* contr. d'acces a la memoire commune emem - acces temps partage */

        Graphe 1,1;

   cn1: am1,am2! -> cn2;     /* autorisation processeur 1 */
   cn2: -> cn3;              /* attente */
   cn3: am2,am1! -> cn4;     /* autorisation processeur 2 */
   cn4: -> cn1.              /* attente */

 /* Description fonctionnelle de sequencement   #NIVEAU 2# */

        Graphe 2,1;

 node11:/am1/ sa1 -> node21,       /* acquisition d'une instruction */
        /am1!/ -> node11;          /* "wait" */

 node21:/ir1(29)! & ir1(28)!/ sual1 -> node31,  /* activation de UAL */
        /ir1(29)! & ir1(28) / scon1 -> node11,  /* act. sauts cond. */
        /ir1(29) & ir1(28)! & am1 / ses11 -> node41,  /* act. E/S */
        /ir1(29) & ir1(28)! & am1!/ -> node21;        /*  " wait" */
 node31:/am1/ schr1,sa1 -> node21, /* charg.d'un reg.& "fetch overlap"*/
        /am1!/schr1 -> node11;     /* charg.d'un reg. sans "fetch" */
 node41:/am1/ses21 -> node11,      /* act. E/S */
        /am1!/ -> node41.          /* "wait" */


 /* Description fonctionnelle de sequencement  processeur 2 */

        Graphe 2,2;

  node12:/am2/ sa2 -> node22,      /* acquisition d'une instruction */
         /am2!/ -> node12;         /* "wait" */
  node22:/ir2(29)! & ir2(28)!/ sual2 -> node32,  /* act. UAL */
         /ir2(29)! & ir2(28) / scon2 -> node12,  /* act. sauts cond. */
         /ir2(29) & ir2(28)!& am2 / ses12 -> node42,/* act. E/S */
         /ir2(29) & ir2(28)!& am2!/ -> node22;      /* "wait" */
  node32:/am2/ schr2,sa2 -> node22,/* charg.d'un reg.& fetch overlap*/
         /am2!/ schr2 -> node12;   /* pas d'"overlapping" */
  node42:/am2/ ses22 -> node12,    /* activation de E/S */
         /am2!/ -> node42.         /* "wait" */
```

b.

```
Horloge: 2.          /* horloge a deux phases */

/* contr. d'acces a la memoire commune emem - acces temps partage */


              Regsra(2).
          Pla 1,1;D10
          sra(1),sra(0) -
          sra(1),sra(0),am1,am2;
          ( 00 ) - ( 01 10);
          ( 01 ) - ( 10 00);
          ( 10 ) - ( 11 00);
          ( 11 ) - ( 00 00).
```

```
     Regsr1(2).
Pla 2,4;D12
am1,ir1(27),ir1(22),sr1(1),sr1(0) -
sr1(1),sr1(0),sa1,sual1,scom1,ses1,scm1,ses2;
(1xx 00 ) - ( 01 100000);
(0xx 00 ) - ( 00 000000);
(x00 01 ) - ( 10 010000);
(x01 01 ) - ( 00 001000);
(110 01 ) - ( 11 000100);
(010 01 ) - ( 01 000000);
(1xx 10 ) - ( 01 100010);
(0xx 10 ) - ( 00 000010);
(1xx 11 ) - ( 00 000001);
(0xx 11 ) - ( 11 000000).
```

```
     Regsr2(2).
Pla 2,2;D12
am2,ir2(27),ir2(28),sr2(1),sr2(0) -
sr2(1),sr2(0),sa2,sual2,scom2,ses12,sc12,ses22.
(1xx 00 ) - ( 01 100000);
(0xx 00 ) - ( 00 000000);
(x00 01 ) - ( 10 010000);
(x01 01 ) - ( 00 001000);
(110 01 ) - ( 11 000100);
(010 01 ) - ( 01 000000);
(1xx 10 ) - ( 01 100010);
(0xx 10 ) - ( 00 000010);
(1xx 11 ) - ( 00 100001);
(0xx 11 ) - ( 11 000000).
```

Fig.3.37. Les descriptions du contrôleur d'accès à la mémoire commune et des séquenceurs du bi-processeur np32x2 en langage LIDO; (a) version fonctionnelle, (b) version structurelle.

4. Remarques finales

Le langage LIDO, à part son orientation systémique, a déjà révélé une grande souplesse et facilité d'utilisation. Des exemples de plus en plus nombreux (processeurs, circuits logiques, circuits arithmétiques, circuits interfaces,...) permettent de confirmer son grand champs d'application et sa simplicité. Toutefois, pour illustrer de façon plus "évoluée" et complète les capacités du langage LIDO nous prévoyons la préparation d'un catalogue des descriptions en commençant par les circuits simples et finissant par les systèmes complexes tels que les architectures MIDM, pipe-line et data-flow.

Système L I D O

Préprocesseur pour la Compilation

en Silicium

TABLE DES MATIERES

1. Introduction

2. Les fonctions du système LIDO

2.1. Description de l'architecture du matériel

2.2. Vérification fonctionnelle de l'architecture du matériel

2.2.1. L'interprétation symbolique

2.2.2. L'interprétation profonde

2.3. Synthèse automatisée des matrices logiques

3. Construction du système LIDO

4. Les modes de travail et les commandes du système LIDO

4.1. Les modes du travail

4.2. Les commandes

5. Les performances du système LIDO

IV. SYSTEME L I D O - Préprocesseur pour la compilation en silicium

1. Introduction

Le système LIDO* a été conçu afin de pouvoir vérifier et transformer différentes architectures du matériel préalablement décrites en langage LIDO. Notre but principal a été de créér un outil moderne permettant d'explorer un très vaste espace de conception en exploitant les mécanismes conformes aux sous-systèmes VLSI et futurs ULSI (Ultra Large Scale of Integration).
Nous avons adopté une philosophie de conception proche de celle utilisée par les réalisateurs de compilateur en silicium GENESIL [Sil85]. Cette philosophie consiste a générer des sous-systèmes topologiques (PLAs de contrôle, UALs, blocs de registres,...) directement à partir d'une description détaillée de l'architecture du matériel. Dans ce cadre le système LIDO fonctionne comme un préprocesseur de compilateur en silicium fournissant un interface simple et "intelligent" entre l'architecte et le compilateur en silicium, ce qui permet de développer rapidement les architectures complexes en utilisant des descriptions fonctionnelles et/ou structurelles du matériel.
Il nous semble nécessaire de souligner que les termes "préprocesseur" et "preprocessing" sont liés essentiellement au rôle préliminaire de la compilation en silicium tel que la vérification fonctionnelle et transformation des descriptions fonctionnelles en descriptions structurelles corespondantes. Dans ce cadre le système LIDO permet de concevoir différentes architectures dont l'optimisation est principalement fonctionnelle et structurelle au niveau transfert de registres. Il n'est pas question ici d'optimisation topologique, laquelle est considerée comme une des tâches principales du compilateur en silicium.

*Le système LIDO est implémenté en langage C sur machines VAX (VMS) et IBM PC/XT/AT (MS/DOS)

Indépendamment de la philosophie de conception du matériel le projet LIDO a été réalisé sous l'angle de l'approche systémique et hiérarchique de la complexité du matériel. Cette approche est présente partout, aussi bien dans la conception du langage LIDO que dans l'implémentation du système LIDO et dans la méthodologie d'utilisation de ce système.

Grâce à l'approche systémique et hiérarchisée il est possible au concepteur de développer l'architecture de façon hautement stratifiée. Cette stratification ou décomposition "horizontale" des systèmes informatiques nous semble être la meilleure méthodologie pour la maîtrise de leur complexité.

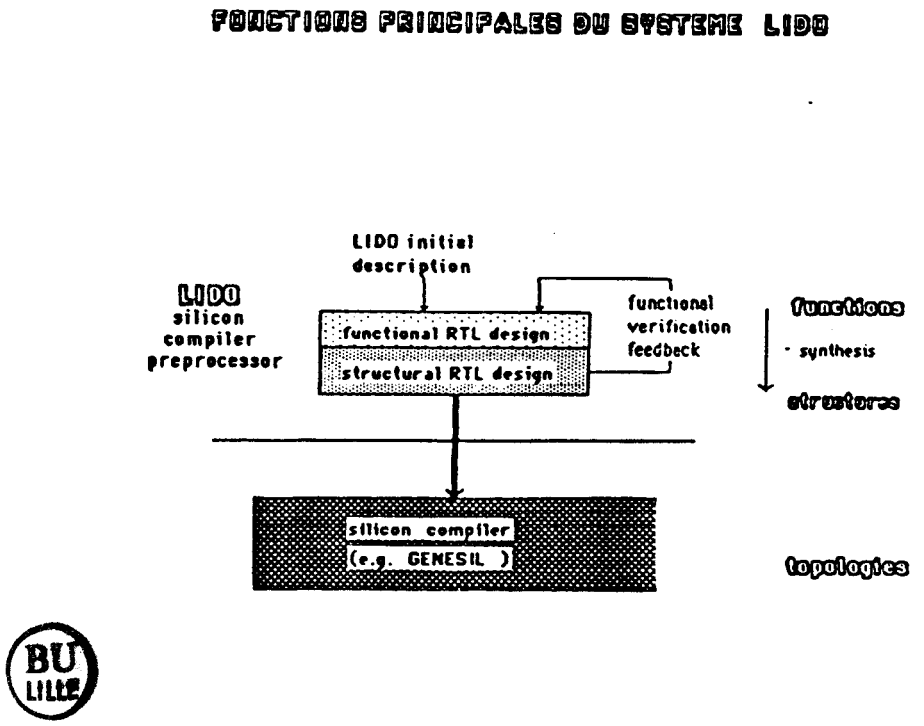


Fig.4.1. Le système LIDO - préprocesseur pour la compilation en silicium.

2. Les fonctions du système LIDO

Les trois fonctions principales du système LIDO sont les suivantes:

- description de l'architecture du matériel
- vérification fonctionnelle symbolique et profonde
- synthèse automatisée

2.1. Desription de l'architecture du matériel

Les langage source pour le système LIDO est le langage LIDO. Ce langage, présenté dans le chapitre précedant, permet de définir l'architecture du matériel evolué et orienté VLSI de façon fonctionnelle et/ou de façon structurelle.

2.2. Vérification fonctionnelle de l'architecture du matériel

Une des parties les plus importantes du système LIDO est l'interpréteur des descriptions LIDO. L'interpréteur anime de façon globale et "systémique" toutes les opérations spécifiées dans la description donnée. Il permet de faire fonctionner des descriptions fonctionnelles aussi bien que des descriptions structurelles fournissant deux types de traces :

- traces symboliques (interprétation symbolique)
- traces profondes (interprétation profonde)

2.2.1. L'interprétation symbolique.

Cet type d'interprétation permet d'analyser le fonctionnement de l'architecture sans entrer dans les détails d'exécution des opérations transfert de registres propres à la partie opérative. La trace symbolique visualise exclusivement la structure de chemins de données activés pendant le processus d'interprétation. On peut dire qu'à travers une trace symbolique nous observons le "flux de structure" en ignorant le contenu de cette structure. De cette façon la trace

symbolique permet de vérifier globalement le fonctionnement d'une architecture pour tous les jeux de données possibles. Par contre elle ne permet pas d'analyser les transformations des contenus des chemins de données. Ce rôle est propre à l'inteprétation profonde.

2.2.2. L'interprétation profonde.

L'interprétation profonde assure une analyse complète de tous les événements intervenant dans l'architecture interprétée. On y observe le flux de données et l'éxécution des opérations au niveau de la partie opérative.

En système LIDO les deux types de traces peuvent être générés concurremment. Ci-dessous nous présentons un fragment (un pas) de trace mixte (interactive); symbolique et profonde, réalisée pour le processeur np32. Nous y voyons les opérations activées - présentation symbolique, et les contenus des porteurs sélectionnés à visualiser - présentation profonde.

```
commands: c,t,a,s,h,p,f,v ? p

!!   TRACE     cycle 10   phase 2    last phase 1 sec.
$ ir(23,20)=>rat1,ir(19,16)=>rat2,raea(@rat1)=>bus1,raea(@rat2)=>bus2
$ bus1+bus2=>cact
$ cact(31)=>s,cact/!=>z!
$ bus1(31)&bus2(31)&cact(31)!=>cdn,bus1(31)!&bus2(31)!&cact(31)=>cdp,cdn!cdp=>d
$ cact(32)=>r
$ cact(31,0)=>act
prc        0000  0000  0000  0000  0101
         > 0000  0000  0000  0000  0101
ir         0000  0000  0000  0000  0000  0000  0000  0000
         > 0000  0000  0000  0000  0000  0000  0000  0000
bus1       0111  1111  1111  1111  1111  1111  1111  1111
bus2       0111  1111  1111  1111  1111  1111  1111  1111
act        1111  1111  1111  1111  1111  1111  1111  1110
```

Fig.4.2. Un fragment de la trace mixte (symbolique et profonde sélective) en mode interactif.

L'interpréteur du langage LIDO travaille sur deux modes : le mode interactif et le mode "batch", ce qui permet d'effectuer l'analyse fonctionnelle d'une façon convenant à l'utilisateur. Les traces réalisées en mode "batch" ont un caractère plus complet que les traces interactives, ces dernières restant hautement sélectives. Un fragment (un pas) de la trace type "batch", réalisée pour le processeur np32, est présenté ci-dessous. Nous y voyons indiquées les activations des différentes unités intervenant dans la description du processeur np32.

```
$$  Cycle 2   Phase 1

 Graphe activé   niveau 1  nr 1
 Graphe activé   niveau 2  nr 2
 Pla activée   niveau 2   nr 2
 srg      s 01
          m 11
 srg      s 01
          m 10
 sal        0
 sual1      1
 scon1      0
 ses11      0
 schr1      0
 ses21      0
 Fun. activé   niveau 1   nr 11
$ ir1(23,20)=>rat11,ir1(19,16)=>rat21,rmem1(@rat11)=>bus11,rmem1(@rat21)=>bus21

 rat11      0000
 rat21      0000
 bus11        00000000  00000000  00000000  00000000
 bus21        00000000  00000000  00000000  00000000
 Fun. activé   niveau 1   nr 12
 Fun. activé   niveau 1   nr 21
 Fun. activé   niveau 1   nr 22
 Fun. activé   niveau 1   nr 31
 Fun. activé   niveau 1   nr 32
 Fun. activé   niveau 1   nr 41
$ cact1(31,0)=>act1
 act1         00000000  00000000  00000000  00000000
 Fun. activé   niveau 1   nr 42
```

Fig.4.3. Un fragment de la trace mixte en mode "batch"

2.3. Synthèse automatisée des matrices logiques

La synthèse des structures matérielles est l'objectif important de l'utilisation du système LIDO comme préprocesseur pour la compilation en silicium. A cette fin le système LIDO dispose d'un synthétiseur des matrices logiques à partir des descriptions fonctionnelles exprimées par des graphes de contrôle. Les matrices logiques résultantes sont formulées dans les termes du langage LIDO, ce qui permet de conserver l'intégrité du processus de conception.
Ci-dessous nous présentons un exemple de la transformation d'un graphe de contrôle en matrice SLA (PLA+registre d'état) sans aucune optimisation sauf la triangularisation.

```
      Graph 2.1;

node11:/am1/ sa1 -> node21,       /* fetch */
      /am1!/ -> node11;           /* "wait" */

node21:/ir1(29)! & ir1(28)!/ sual1 -> node31,  /* ALU activation */
      /ir1(29)! & ir1(28) / scon1 -> node11,  /* jumps activation*/
      /ir1(29) & ir1(28)! & am1 / ses11 -> node41,  /* I/O */
      /ir1(29) & ir1(28)! & am1!/ -> node21;        /*  " wait" */
node31:/am1/ schr1,sa1 -> node21, /* load result & "fetch overlap"*/
      /am1!/schr1 -> node11;      /* load result without fetch */
node41:/am1/ses21 -> node11,      /* I/O */
      /am1!/ -> node41.


    Give output file name  :    a:risc2ap

    Name of graph to transform - ex: Graph 1,1;    Graph 2,1;

    Name of pla with its phase - ex: Pla 1,1;C1)   Pla 2,1;C1)

    Name of state register      sr


Register; sr(2).
Pla 2,1;C1)

am1,ir1(29),ir1(28),sr(1),sr(0) -
 sr(1),sr(0),sa1,sual1,scon1,ses11,schr1,ses21;
(1xx 00) - (01 100000);
(0xx 00) - (00 000000);
(x00 01) - (10 010000);
(x01 01) - (00 001000);
(110 01) - (11 000100);
(010 01) - (01 000000);
(1xx 10) - (01 100010);
(0xx 10) - (00 000010);
(1xx 11) - (00 000001);
(0xx 11) - (11 000000).
```

Fig.4.4. Un exemple de la synthèse automatique d'une matrice logique.

Le développement actuel du système LIDO est orienté vers l'optimisation fonctionnelle, basée sur le compactage et la décomposition fonctionnelle des graphes de contrôle. Le processus de compactage permet de "synthétiser" les sous-graphes librement exprimés par le concepteur (p.e. un sous-graphe → une instruction) en un graphe compacté, dans lequel toutes les séquences du même type (longueur, boucles...) passent par les mêmes noeuds. La matrice logique générée à partir d'un graphe compacté posséde moins d'entrés/sorties d'état, qu'une matrice générée à partir d'un graphe original. Un exemple de compactage est présenté sur le Fig.4.5.

```
        File name :  gt1
 Gr 1,1;                    /* graphe initial */
 n1:/a!&b!/ s1 -> n2,
    /a!&b / s2 -> n3,
    /a &b!/ s3 -> n4;
 n2: s5 -> n5;              Register; sr(3).
 n3:/c!/ s6 -> n5,          Pla 1,1;
    /c / s7 -> n6;
 n5: s8 ->n6;               a,b,c,sr(2),sr(1),sr(0) -
 n4: s7 ->n6;                sr(2),sr(1),sr(0),s1,s2,s3,s5,s6,s7,s8;
 n6: -> n1.                 (00x 000) - (001 1000000);
                            (01x 000) - (010 0100000);
                            (10x 000) - (011 0010000);
                            (xxx 001) - (100 0001000);
                            (xx0 010) - (100 0000100);
                            (xx1 010) - (101 0000010);
                            (xxx 100) - (101 0000001);
                            (xxx 011) - (101 0000010);
                            (xxx 101) - (000 0000000).
                           * EOF *

Gr 1,2;                   /* graphe compacté */
n1:/a!&b!/ s1 -> n2,
   /a!&b / s2 -> n2,
   /a &b!/ s3 -> n2;         Register; sr(2).
n2:/a!&b!/ s5 -> n3,         Pla 1,2;
   /a!&b &c!/ s6 -> n3,
   /a!&b &c / s7 -> n4,      a,b,c,sr(1),sr(0) -
   /a &b!/ s7 -> n4;          sr(1),sr(0),s1,s2,s3,s5,s6,s7,s8;
n3: s8 -> n4;                (00x 00) - (01 1000000);
n4: -> n1.                   (01x 00) - (01 0100000);
End.                         (10x 00) - (01 0010000);
 * EOF *                     (00x 01) - (10 0001000);
                             (010 01) - (10 0000100);
                             (011 01) - (11 0000010);
                             (10x 01) - (11 0000010);
                             (xxx 10) - (11 0000001);
                             (xxx 11) - (00 0000000).
                            * EOF *
```

Fig.4.5. Un exemple du compactage d'un graphe de contrôle et les matrices logiques en résultant.

3. Construction du système LIDO

Le système LIDO possède une structure interne modulaire. Chaque module réalise une commande. Le travail de la totalité est contrôlé par le module principal - main. Ce module accepte des commandes et des paramètres et les transmet aux autres modules concernés. Sur le schéma ci-dessous nous voyons les modules suivants:

- module principal de contrôle
- module de synthèse PLA
- module chargeur des descriptions et des contenus des porteurs d'information
- module interpréteur (interpréteur symbolique et profond)
- module traceur et "mapper"
- module de visualisation

ainsi que les tableaux suivants:

- tableau du système où est enregistré l'état du système
- tableau de description LIDO où est conservée la description LIDO à interpréter
- tableau de contenu où sont enregistrés les contenus des porteurs d'informations
- tableaux des porteurs où sont conservés les pointeurs vers les mécanismes utilisés en description LIDO et les pointeurs vers les contenus des porteurs d'informations

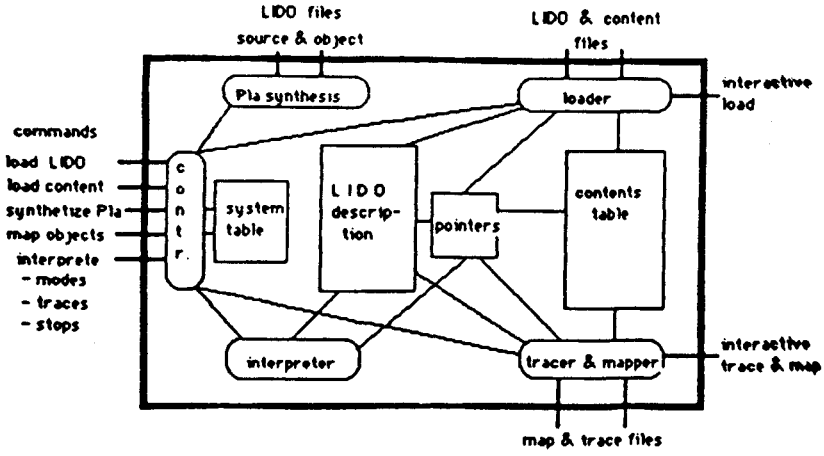


Fig.4.6. Schéma global de la structure du système LIDO

Remarque:

Il faut souligner que l'interpréteur interpréte à l'aide des pointeurs la description source ce qui permet dans les traces d'interprétation d'illustrer par la description source les chemins des données activés pendant les processus d'interprétation (trace symbolique). Cette possibilité, très importante pour l'analyse fonctionnelle n'existe pas dans les simulateurs travaillant en mode compilé.

4. Les modes de travail et les commandes du système LIDO

4.1. Les modes de travail

Le système LIDO a été conçu afin de permettre une grande souplesse de travail. Cette souplesse est, en grande mesure, possible grâce à deux modes de travail:

- mode interactif (on-line)
- *mode "batch" ou mode "fichier" (off-line)*

Le mode interactif permet d'analyser à vif le fonctionnement d'une description fonctionnelle ou structurelle. Ce mode est très utile pour une analyse préalable et immédiate, mais peut aussi bien servir pour une expérimentation focalisée sur un fragment de description grâce à ses capacités sélectives (chargement, traces et maps sélectifs). De plus le mode interactif offre à côté des traces profondes d'interprétation la possibilité de traces purement symboliques ou mixtes.

Le mode "batch" permet d'effectuer une analyse fonctionnelle complète et detaillée d'une description fonctionnelle et/ou structurelle et de produir des fichiers et des listings facilitant une réflexion approfondie et "off-line" sur des résultats d'interprétation. Il y est possible de générer la trace symbolique associée à la trace profonde.

4.2. Les commandes du système LIDO

Les sept commandes principales du système LIDO sont:

- **chargement de description #d :**

Cette commande permet de charger une description préalablement préparée dans un fichier, le nom du fichier doit être fourni au système.
(give description name: <nom du fichier>)

fonctionnement:
La commande "chargement de description" charge les tableaux principaux du système et prépare les pointeurs vers les tableaux de contenu des porteurs d'information. L'analyse syntaxique globale indique les différentes erreurs relatives aux défauts dans les déclarations des mécanismes et des porteurs d'informations.

remarque:
Cette commande est unique pour les deux modes de travail du système LIDO.

- **chargement du contenu #c :**

Cette commande permet de charger le contenu binaire initial des porteurs d'information; c'est-à-dire: connecteurs, terminaux, registers, mémoires et mémoires associatives. De plus elle permet d'initialiser les états fonctionnels des graphes et de l'horloge.
Dans le mode "batch" le contenu doit être préparé dans un fichier dont le nom doit être fourni avec la commande.
Dans le mode interactif le contenu des éléments indiqués est chargé à partir du clavier.

fonctionnement:
La commande "chargement de contenu" charge les tableaux de contenus aux endroits indiqués par les pointeurs. Le chargement d'un porteur non-déclaré signale une erreur de chargement et le nom défectueux.

remarque:
Cette commande permet de restituer l'état d'une structure interprétée et dont le contenu a été sauvegardé par la commande map .

- **mapping *m :**

Cette commande permet de sauvegarder et de visualiser les contenus des porteurs d'information utilisés dans une description.
La commande map peut être utilisée en mode "batch" (sauvegarde) ou en mode interactif (visualisation). En mode "batch" il faut fournir le nom du fichier pour l'enregistrement de la "map".Le mapping en mode "batch" peut être: global - sous commande *a, partiel - sous commande *r, selectif - sous commande *s suivie de la liste d'éléments à mapper.
Le mapping en mode interactif est exclusivement sélectif, c'est-à-dire que la commande *m est suivie directement d'une liste d'éléments à visualiser.

remarque:
Cette commande peut être utilisée pour la sauvegarde complète de l'état de la structure interprétée.

- **synthèse des PLAs *s :**

Cette commande permet de transformer les graphes de contrôle en structures PLA. Il faut lui fournir les noms des fichiers de description source et objet et les paramètres de transformation.

p.e.

```
      Give input  file name : risc2
Give output file name : risc2p
Name of graph to transform - ex: Graph 1,1; :  Graph 2,1;
Name of PLA with its phase - ex: Pla 1,1;[1] :  Pla 2,1;[1]
Name of state register : sr
```

fonctionnement:
La commande "synthèse des PLAs" lit le fichier source LIDO (description fonctionnelle) et génère le nouveau fichier objet LIDO (description structurelle). Les transformations à effectuer sont indiquées par les paramètres fournis par le concepteur. Un mauvais paramètre (p.e. manque d'un graphe dans la description source) provoque une erreur de fonctionnement et un commentaire correspondant.

- **visualisation $v ;**

Cette commande permet de visualiser la totalité ou un fragment de la description d'une architecture en langage LIDO. La visualisation peut être effectuée aussi bien au niveau des commandes principales qu' au niveau des sous-commandes d'interprétation.

- **format $f ;**

Cette commande a un caractère supplémentaire et permet de définir le format unique des traces de l'interprétation profonde et des "maps". Le formatage se fait par le groupement et par l'espacement des positions binaires .

- **exit $e ;**

Cette commande provoque la sortie du système LIDO
(see you later LIDO)

- **interprétation *i :**

La commande d'interprétation est la commande principale du système LIDO. Elle permet d'animer toute sorte de descriptions "bien formées" en langage LIDO et de fournir différentes traces d'interprétation. Cette commande est suivie d'une chaine de sous-commandes initialisée par l'une des deux sous-commandes de mode: *c (mode continu ou "batch") et *s (mode interactif ou pas-à-pas).

MODE BATCH (off-line)

-sous-commandes de trace:

*a - cette sous-commande provoque l'enregistrement de toutes les affectations pendant le processus d'interprétation, elle donne les traces les plus détaillées possibles

*c - cette sous-commande provoque l'enregistrement des affectations propres aux chemins de contrôle

*d - cette sous-commande provoque l'enregistrement des affectations propres aux chemins de données

*n - cette sous-commande supprime toutes les traces

remarques:

α. Les sous-commandes de trace (*a,*c,*d) sont utilisées exclusivement en mode "batch". Il faut leur fournir le nom du fichier de la trace.

β. Les sous-commandes de trace sont suivies de la sous-commande de format et de la sous-commande de trace symbolique (opérations source activées).

- sous-commandes d'arrêt:

*p - cette sous-commande permet de spécifier le nombre de phases horloge d'interprétation

*c - cette sous-commande permet de spécifier le nombre de cycles horloge d'interprétation

Ce deux sous-commandes sont suivies par la demande du nombre phases/cycles:

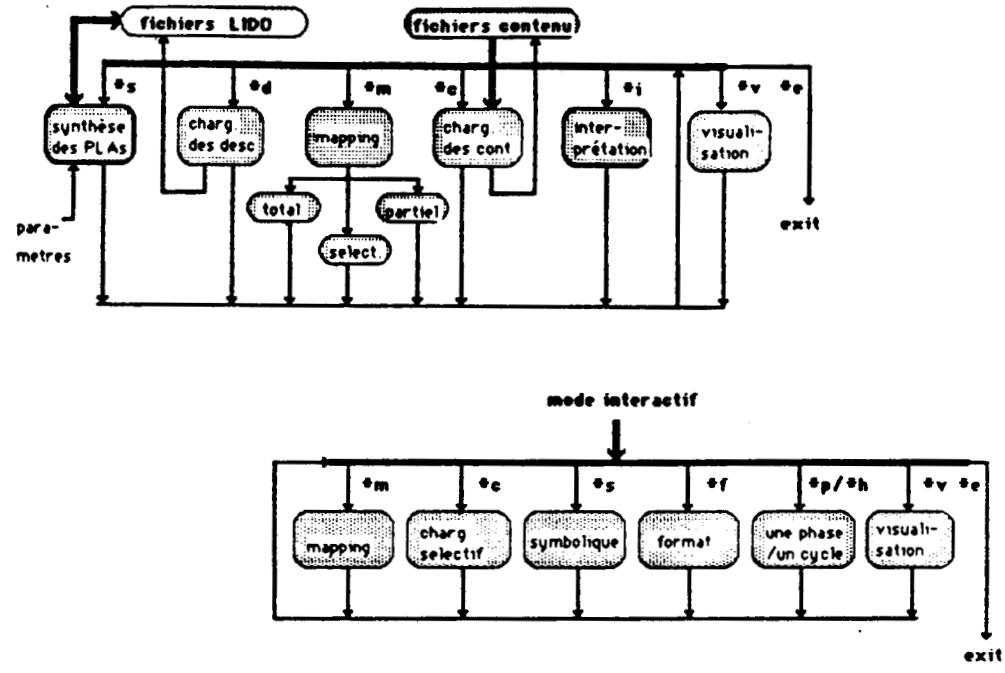
(give phases/cycles number: <nombre>)

*d - cette sous-commande d'arrêt permet d'arrêter le processus d'interprétation si la valeur de la "donnée d'arrêt" (p.e. contenu d'un registre) est égale à la valeur générée pendant le processus d'interprétation. Le nom d'élément et la donnée sont demandés par le système:
(give the breakpoint name & content: <nom>=<donnée binaire>)

remarques:
α. Les sous-commandes d'arrêt sont utilisées exclusivement en mode "batch".
β. La fin du processus de l'interprétation continue est suivie par le retour au niveau des commandes principales.

MODE INTERACTIF (on line)

En mode interactif après chaque pas de fonctionnement le système LIDO demande l'une des neuf sous-commandes suivantes:
*s - cette sous-commande provoque la visualisation de la trace symbolique comportant exclusivement les chemins de données activés, visualisés sous forme symbolique
*f - cette sous-commande installe un format de visualisation des contenus de porteurs pour la trace et le map sélectifs
*t - cette sous-commande doit être suivie d'une liste d'éléments porteurs (connecteurs,terminaux et registres) à visualiser dans chaque pas d'interprétation
*m - cette sous-commande doit être suivie d'une liste d'éléménts porteurs (connecteurs,terminaux,registres,mémoires,mémoires associatives) à visualiser à l'instant donné
*c - cette sous-commande doit être suivie d'une liste d'éléments porteurs et leurs contenus correspondants prévus à charger à l'instant donné
*v - cette sous-commande permet de visualiser la totalité ou un fragment de la description source de l'architecture interprétée
*p et *h - ces sous-commandes initialisent un pas d'interprétation: une phase horloge (*p), un cycle horloge (*h)
*e - cette sous-commande provoque le retour au niveau des commandes principales



INTERPRETATION

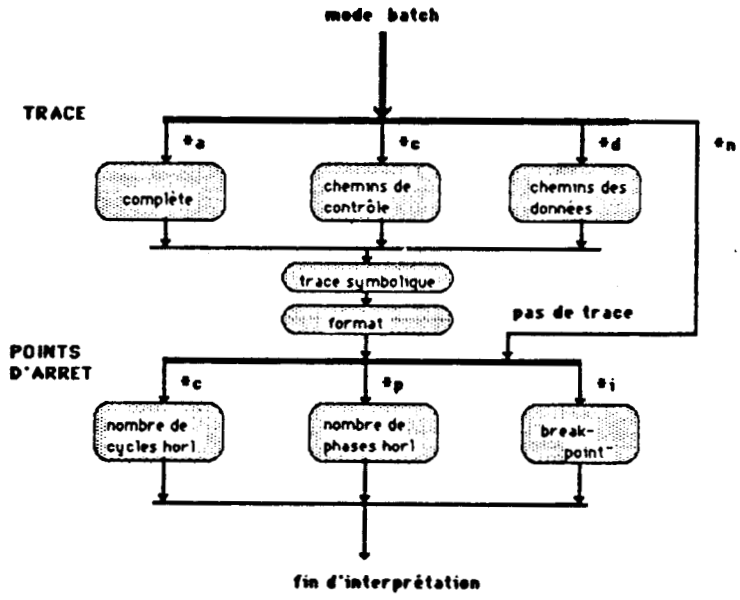


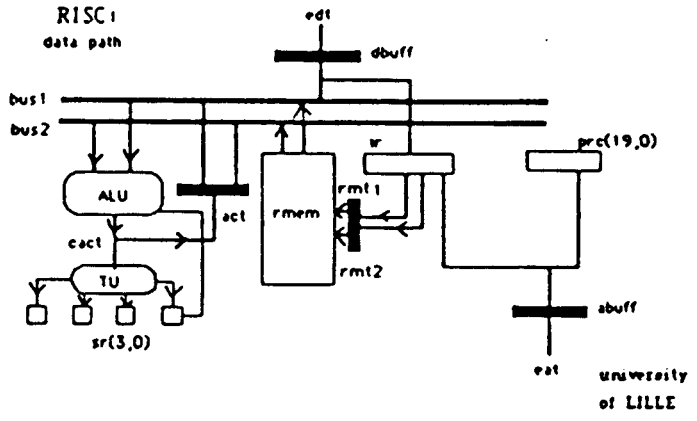
Fig.4.7. Le schéma synoptique des "chemins de travail" dans le système LIDO.

5. La performance du système LIDO

Le problème de la performance ou de la vitesse de simulation de l'architecture du matériel est connu de tous ceux qui ont implémenté un langage CHDL. La performance peut être largement influencée par l'implantation, dont nous évoqueront certains types ci-dessous:

- compilation; langage CHDL → code machine
- translation et compilation; CHDL → langage de programmation → code machine
- interprétation "indirecte"; CHDL → langage intermediaire → interpréteur
- interprétation "directe"; CHDL → interpréteur
  (problème: langages procéduraux et non-procéduraux)

Vouloir disposer de traces symboliques directement accessibles nous a imposé la dernière solution, c'est-à-dire, l'interprétation directe. Malgré les exigences de l'approche systémique, supposant toutes les opérations a priori actives, l'interpréteur LIDO permet par exemple de simuler le processeur np32 avec la vitesse approximative de 1 instruction/10 sec.* (1 instruction/2 sec.)*. Celle-ci dépend largement du type d'instruction et des opérateurs utilisés.
Ci-dessous nous présentons un exemple comparatif* concernant la vitesse de l'interprétation d'une phase horloge et relatif aux deux descriptions l'une d'un mono-processeur np32, l'autre d'un bi-processeur np32x2. Cet exemple montre que le temps de l'interprétation est, en quelque sorte, proportionnel à la taille de la description et à la localisation des arguments sur la liste des déclarations de porteurs.




```
commands: c,t,m,s,h,p,f,v ?  p

##    TRACE     cycle 2   phase 2    last phase 1 sec.
$ ir(23,20)=>rat1,ir(19,16)=>rat2,rmem(@rat1)=>bus1,rmem(@rat2)=>bus2
$ bus1+bus2=>cact
$ cact(31)=>s,cact/!=>z!
$ bus1(31)&bus2(31)&cact(31)!=>cdn,bus1(31)!&bus2(31)!&cact(31)=>cdp,cdn!cdp=>d
$ cact(32)=>r
$ cact(31,0)=>act
 bus1       0011  1111  1111  1111  1111  1111  1111  1111
 bus2       0011  1111  1111  1111  1111  1111  1111  1111
 act        0111  1111  1111  1111  1111  1111  1111  1110
 commands: c,t,m,s,h,p,f,v ?  p

##    TRACE     cycle 3   phase 1    last phase 3 sec.
```

Fig.4.8.a. Un fragment de la trace interactive (une phase horloge) pour processeur np32 (temps réel d'éxécution 3 sec.)

```
**    TRACE     cycle 4   phase 1    last phase 0 sec.
$ ir2(23,20)=>rat12,ir2(19,16)=>rat22,rmem2(@rat12)=>bus12,rmem2(@rat22)=>bus22
$ bus12+bus22=>cact2
$ cact2(31)=>s2,cact2/!=>z2!
$ bus12(31)&bus22(31)&cact2(31)!=>cdn2,bus12(31)!&bus22(31)!&cact2(31)=>cdp2,cdn
2!cdp2=>d2
$ cact2(32)=>r2
$ cact2(31,0)=>act2
 bus11      0000  0000  0000  0000  0000  0000  0000  0000
 bus12      0000  0000  0000  0000  0000  0000  0000  0011
 bus21      0000  0000  0000  0000  0000  0000 -0000  0000
 bus22      0000  0000  0000  0000  0000  0000  0000  0011
 act1       0000  0000  0000  0000  0000  0000  0000  0000
 act2       0000  0000  0000  0000  0000  0000  0000  0110
 commands: c,t,m,s,h,p,f,v ?  p

**    TRACE     cycle 4   phase 2    last phase 5 sec.

 **    TRACE     cycle 6   phase 1    last phase 0 sec.
 $ ir1(23,20)=>rat11,ir1(19,16)=>rat21,rmem1(@rat11)=>bus11,rmem1(@rat21)=>bus21
 $ bus11+bus21=>cact1
 $ cact1(31)=>s1,cact1/!=>z1!
 $ bus11(31)&bus21(31)&cact1(31)!=>cdn1,bus11(31)!&bus21(31)!&cact1(31)=>cdp1,cdn
 1!cdp1=>d1
 $ cact1(32)=>r1
 $ cact1(31,0)=>act1
  bus11      0000  0000  0000  0000  0000  0000  0000  1111
  bus12      0000  0000  0000  0000  0000  0000  0000  0110
  bus21      0000  0000  0000  0000  0000  0000  0000  1111
  bus22      0000  0000  0000  0000  0000  0000  0000  0011
  act1       0000  0000  0000  0000  0000  0000  0001  1110
  act2       0000  0000  0000  0000  0000  0000  0000  0110
  commands: c,t,m,s,h,p,f,v ?  p

**    TRACE     cycle 6   phase 2    last phase 6 sec.
```

Fig.4.A.b. Deux fragments de la trace interactive (une phase horloge) pour bi-processeur np32x2 (temps réel d'éxécution 5 sec. et 6 sec.)*

* implémentation IBM XT (MSDOS); * implémentation VAX750 (VMS)

Le deuxième exemple indique la différence entre le temps d'interprétation d'un opérateur simple (addition,soustraction,opérateur logique) et le temps d'interprétation d'un opérateur complexe - multiplication.

```
commands: c,t,m,s,h,p,f,v ?  p

**    TRACE     cycle 14   phase 1    last phase 1 sec.
$ a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d
 a          00000000000000000000011111111111
          ) 00000000000000000000011111111111
 b          00000000000000000011111111111111
          ) 00000000000000000011111111111111
 c          00000000000000000100111111111110
          ) 00000000000000000100111111111110
 d          00000000000000000000000000000000000000
          ) 00000000000011111111111011000000000001
 commands: c,t,m,s,h,p,f,v ?  p

**    TRACE     cycle 15   phase 1    last phase 2 sec.


$ a+b=>c,a+b=>c,a+b=>c,a+b=>c,a+b=>c,a+b=>c,a+b=>c,a+b=>c,a+b=>c,a+b=>c
$ a&b=>c,a&b=>c,a&b=>c,a&b=>c,a&b=>c,a&b=>c,a&b=>c,a&b=>c,a&b=>c,a&b=>c
$ a=>b,a=>b,a=>b,a=>b,a=>b,a=>b,a=>b,a=>b,a=>b,a=>b
$ a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d,a*b=>d
 a          00000000000000000000011111111111
          ) 00000000000000000000011111111111
 b          00000000000000000011111111111111
          ) 00000000000000000011111111111111
 c          00000000000000000000000000000000
          ) 00000000000000000000011111111111
 d          00000000000000000000000000000000000000
          ) 00000000000011111111111011000000000001
 commands: c,t,m,s,h,p,f,v ?  p

**    TRACE     cycle 27   phase 1    last phase 4 sec.
```

Fig.4.9. Un exemple comparatif de la vitesse d'éxécution d'un opérateur simple (addition) et l'opérateur complexe (multiplication).

V. EN CONCLUSION

Chaque jour l'utilisation du système LIDO pour de nouvelles descriptions et de nouvelles expérimentations vérifie et confirme la validité des concepts systémiques . Par ailleurs les résultats obtenus impliquent des modifications et des développements qui permettent d'augmenter la facilité et la souplesse d'exploitation du langage et du système LIDO. Parmi les nouveaux développements nous envisageons deux directions : la synthèse optimisée des structures matérielles et le méta-logiciel.

La synthèse optimisée des structures matérielles est un objectif important dans le cadre du "preprocessing" pour la compilation en silicium. Cette optimisation doit, dans notre cas, viser la génération des structures logiques (PLAs, SLAs, MGAs) optimisées globalement à partir des graphes de contrôle. Le travail sur ce sujet est déjà amorcé et les premiers résultats concrets sont attendus au cours de l'année 1987.

Parallèlement nous prévoyons la réalisation du module de synthèse automatisée des chemins de données exprimés initialement de façon fonctionnelle.

Le méta-logiciel de type méta-assembleur ou méta-micro-assembleur permettrait de concevoir les programmes et/ou microprogrammes destinés aux structures sous-jacentes tels que processeurs et micro-machines. La version actuelle du système LIDO n'assure que la programmation (microprogrammation) binaire.

Malgré toutes ces limitations, les versions actuelles du système LIDO (VAX/VMS - la recherche, IBM PC-XT-AT/MSDOS - l'enseignement) ont permis de concevoir et de vérifier un nombre important d'architectures du matériel. Parmi les plus intéressantes nous trouvons les processeurs de type RISC et les circuits-processeurs arithmétiques.

Dans l'avenir le plus proche nous prévoyons la préparation d'un catalogue des descriptions englobant un grand éventail d'exemples, des circuits combinatoires les plus simples aux architectures évoluées MIMD ,pipe-line et data-flow. Toutes ces descriptions ainsi que les différentes versions du système LIDO seront mises à la disposition des chercheurs et des étudiants.

BIBLIOGRAPHIE

[Anc85] Ançeau F.,"La compilation en silicium des circuits à structures de microprocesseurs", Bulletin INRIA, N°102, 1985

[Ayl86] Aylor J.H.,et al.,"VHDL - Feature Description and Analysis", IEEE D&T of Computers, April 1986

[Ayr83] Ayres R.F.,"VLSI-Silicon Compilation and the Art of Automatic Microchip Design", Prentice-Hall, 1983

[Ayr79] Ayres R.F.,"Silicon Compilation - A Hierarchical Use of PLAs", Proc.of 16th Design Automation Conf.,1979

[Bak83] Bakowski P.,"NEGLAN - A Language for Negative Control Design", Proc.of 6th Intern.Symposium on CHDLs and Their Applications, Pittsburgh, 1983

[Bak83] Bakowski P.,"A Unified Theory of Computative Forms", Proc.of the IFIP WG10.1 Working Conf. on Methodologies for Computer System Design, Lille, 1983

[Bak83] Bakowski P.,"Rudiments of Multilevel Finite State Machines", Univ.of Lille, Rap.ERA, 1983

[Bara75] Bara Y, Born R.,"A CDL compiler for Designing and Simulating digital systems at the register transfer level", Proc.of 2nd Intern.Symposium on CHDLs and Their Applications, New York, 1975

[Bar75] Barbacci M.R.,"A comparison of register transfer languages for describing computers and digital systems", IEEE Trans.on Comp., vol C-24, February 1975

[Bar77] Barbacci M.R., Naget A.W.,"An ISPS simulator", Dept.of CS and EE Carnegie-Mellon Univ., November 1977

[Bar81] Barbacci M.R.,"Instruction Set Processor Specification (ISPS): The Notation and its Applications", IEEE Trans.on Comp., vol C-30, January 1981

[Bar84] Barbacci M.R.,"Structural and Behavioral description of Digital Systems", in New Computer Architectures, ed. Tiberghien, Academic Press, 1984

[Bell76] Bell C.G.,"Computers - Past, Present and Future", IBM JRD, May 1976

[Bell71] Bell C.G., Newell R.,"Computer Structures: Readings and Examples", Prentice Hall, 1971

[Ber68] Bertalanffy L. von,"General System Theory", Brazilier, New York, 1968

[Bol85] Bolton M.J.P.,"Designing with programmable logic", IEE Proc., vol.132, March/April 1985

[Bor75] Borionne D.,"LASCAR: A Language for Simulation of Computer Architecture", Proc.of 2nd Intern.Symposium on CHDLs and Their Applications, New York, 1975

[Bor85] Borionne D., Le Faou C.,"Overview of the CASCADE Multi-level Description Language and its Mixed-mode simulation mechanisms", IFIP 7th Intern.Conf. on CHDLs and Their Applications, Tokyo, 1985

[Bou83] Boute R.T.,"Functional Description of Digital Systems", IFIP WG10.1 Work. Conf. on Methodologies for Computer System Design, Lille, 1983

[Bra85] Brayton R.K.,et al.,"Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1985

[Chu74] Chu Y.,"Why Do We Need Computer Hardware Description Languages?", COMPUTER (special edition on CHDLs), December 1974

[Chu65] Chu Y.,"An ALGOL-like Computer Design Language", Communications of ACM, October 65

[Chu72] Chu Y.,"Computer Organization and Microprogramming", Prentice Hall, 1972

[Diet75] Dietmeyer D.L., Duley J.R.,"Register Transfer Languages and their Translation", in "Digital System Design Automation", ed. Breuer M.A., Computer Science Press, 1975

[Dul67] Duley J.R.,"DDL - A Digital System Design Language", Ph.D. dissertation, University of Wisconsin, Madison, 1967

[Dur83] Durand P.,"La Systémique", PUF, "Que sais-je?", N°1795

[Est83] Estrin G.,"The Story of SARA", Proc.of the IFIP WG10.1 Working Conf. on Methodologies for Computer System Design, Lille, 1983

[Fly66] Flynn M.J.,"Very high-speed computing systems", Proc.IEEE, vol.54, December 1966

[For83] Forrest Y., Edwards M.D.,"The Automatic Generation of Programmable Logic Arrays from Algorithmic State Machines Descriptions", Proc.of VLSI83 Conf., North-Holland, 1983

[Gil83] Giloi W.K.,"Toward a Taxonomy of Computer Architecture Based on the Machine Data Type View", Proc.of 10th Intern.Symposium on Computer Architecture, Stockholm, 1983

[Gil83] Giloi W.K., Shriver B.D.,"An Overview of the Computer System Design Process", IFIP WG10.1 Working Conf. on Methodologies for Computer System Design, Lille, 1983

[Gor84] Gordon M.J.C.,"How to specify and verify hardware using Higher Order Logic", Lecture notes, Univ.of Cambridge Computer Laboratory, 1984

[Han85] Hanna F.K., Daeche N.,"Specification and Verification using Higher-Order Logic", IFIP 7th Intern. Conf.on CHDLs and Their Applications, Tokyo, 1985

[Hil175] Hill F.J.,"Updating AHPL", Proc.of 2nd Intern. Symposium on CHDLs and Their Applications, New York, 1975

[Hil184] Hill F.J. et al.,"Hardware Compilation from an RTL to a Storage Logic Array Target", IEEE Trans.on Computer-Aided Design, vol. CAD-3,N°3, July 1984

[Hos85] Hoshino et al.,"HSL-FX: A Unified Language for VLSI Design", Proc.of 7th Intern. Conf.on CHDLs and Their Applications, Tokyo, 1985

[Joh79] Johannsen D.,"Bristle Blocks: A Silicon Compiler", 16th Design Automation Conf., 1979

[LeM78] Le Moigne J.L.,"La thèorie du système général", PUF, 1978

[Lieb83] Lieberherr K.J.,"Zeus: A Hardware Description Language for VLSI", Proc.of 20th Design Automation Conf.,1983

[Lip77] Lipovski J.,"Hardware Description Languages: Voices from the Tower of Babel", IEEE Trans.on Comp., June 1977

[Lips86] Lipsett R.,et al.,"VHDL - The Language", IEEE D&T of Computers, April 1986 (special edition on VHDL)

[Log75] Logue J.C. et al.,"Hardware Implementation of a Small System in Programmable Logic Arrays", IBM Journal of R&D, March 1975 (special edition on PLAs)

[Mar79] Marczynski R., Bakowski P.,"What Do Computer Hardware Description Languages Describe?", Proc.of 4th Intern. Symposium on CHDLs and Their Applications, Palo Alto, 1979

[Mar81] Marczynski R.,et al.,"PINLAN - Language for digital integrated circuits description", ICS PAS Report N°453, Warszawa ,1981

[Mead80] Mead C., Convay L.,"Introduction to VLSI Systems", Addison-Wesley, 1980

[Mead83] Mead C.,"Structural and Behavioral Composition of VLSI", Proc.of VLSI83 Conf., North-Holland, 1983

[Mer73] Mermet Y.,"Etude méthodologique de la Conception Assistée par Ordinateur des Systèmes Logiques: CASSANDRE", Thèse d'Etat, USMG, Grenoble, 1973

[Mer85] Mermet Y.,"Vers un système integré de CAO de VLSI", Bulletin INRIA, N°102, 1985

[Mil85] Milne G.J.,"Simulation and Verification", IFIP 7th Intern. Conf.on CHDLs and Their Applications, Tokyo, 1985

[Nash84] Nash J.D.,"Bibliography of Hardware Description Languages", SIGDA, 1984

[New85] Newton A.R.,"Techniques for Logic Synthesis", Proc.of VLSI85, North-Holland, 1985

[Obr81] Obrebska M.,"Comparative Study of Control Units of Microprocessors using different Design Methodologies", Intern. Conf.of New Trends in Integrated Circuits, CMU Pittsburgh, April 1981

[Par81] Parker A.C.,"An I/O Hardware Description Language", IEEE Trans.on Comp., vol-30, June 1981

[Pat85] Patterson D.A.,"Reduced Instruction Set Computers", Communications of the ACM, vol-28, January 1985

[Paw81] Pawlak A., Jezewski J.,"MODLAN - A Language for Multilevel Description and Modeling of Digital Systems", 5th Intern. Symposium on CHDLs and Their Applications, Kaiserslautern, 1981

[Paw85] Pawlak A.,"A Tutorial Guide to Modern Hardware Description and Design Languages", Proc.of EUROMICRO Conf., 1985

[Pil74] Piloty R.,"RTS I (Registertransferschprache)", Institut für Nachrichtenverarbeitung, TH Darmstadt, 1974

[Pil83] Piloty R. et al.,"CONLAN Report", Lecture Notes in Computer Science N°151, Springer Verlag, New York, 1983

[Ros75] Rosnay J. de,"Le Macroscope - Vers une vision globale", Editions du Seuil 1975

[Sha85] Shahdad M., et al.,"VHSIC Hardware Description Language", COMPUTER, February 1985

[Shi79] Shiva S.G.,"Computer Hardware Description Languages - A Tutorial", Proc.of IEEE, vol-67, N°12, December 1979

[Sil85] Silicon Compilers Inc.,"A Guide to Silicon Compilation", 1985

[Su77] Su S.Y.H.,"Hardware Description Languages Applications", COMPUTER (special edition on CHDLs Applications), June 1977

[Su85] Su S.Y.H.,"A Multi-level Hierarchical Simulator based on LALSD II", IFIP 7th Intern. Conf.on CHDLs and Their Applications, Tokyo, 1985

[Sub83] Subrahmanyam P.A.,"Synthesizing VLSI Circuits from Behavioral Specifications: A Very High Level Silicon Compiler and its Theoretical Basis", Proc.of VLSI83 Conf., North-Holland, 1983

[Subr82] Subrata Dasgupta,"Computer Design and Description Languages", in Advances in Computers, vol-21, Academic Press, 1982

[Subr81] Subrata Dasgupta,"S*: A Language for Describing Computer Architectures", Proc.of 5th Symposium on CHDLs and Their Applications, Kaiserslautern, 1981

[Swa77] Swamson R.E.,et al.,"An AHPL Compiler-Simulator System", Proc.of 6th Texas Conf.on Computing Systems, 1977

APPENDIX I

LIDO language syntax

Notes:

1. The meta-symbols { .. } refer to zero or one occurence of the enclosed constructs.
2. The meta-symbols { .. }* refer to zero or more occurences of the enclosed constructs.
3. The meta-symbol ε refers to null production.

The LIDO description is composed from:

1. Memory elements: Memory, Associative memory, Register
2. Transient memory elements: Terminal, Connection
3. Complex combinational logic constructs: Pla, conditions
4. Sequential logic constructs: Horloge, Graph
5. Complex register-transfer constructs: Functional units

BNF for LIDO

<LIDO description>::={<clock description>}{<control graph description>}*
{<pla description>}*{<functional unit description>}*
{<r-t-c declaration>}*{<memory declaration>}*
{<associative memory declaration>}*End.

<r-t-c declaration>::=<register declaration>|<terminal declaration>|<connection declaration>

<register declaration>::=<register header><r-t-c list>

<terminal declaration>::=<terminal header><r-t-c list>

<connection declaration>::=<connection header><r-t-c list>

<memory declaration>::=<memory header><memories>

<associative memory declaration>::=<ass.memory header><ass.memories>

<register header>::= R<string>;

<terminal header>::= T<string>;

<connection header>::= C<string>;

<memory header>::= M<string>;

<ass.memory header>::= A<string>;

<string>::= any string of characters excluding ";"

<r-t-c list>::=<r-t-c>{;<r-t-c>}*

<r-t-c>::=<r-t-c name><r-t-c parameters>

<r-t-c name>::=<name>

<r-t-c parameters>::=(<bitnum>)|£

<name>::= a string of letters, digits and # character

<bitnum>::= decimal number

<memories>::=<memory>{;<memory>}

<memory>::=<memory name><memory parameters>

<memory name>::=<name>

<memory parameters>::=(<wordnum>,<bitnum>)

<wordnum>::= decimal number

<ass.memories>::=<ass.memory name><ass.memory parameters>

<ass.memory name>::=<name>

<ass.memory parameters>::=(<wordnum>,<bitnum>,<compared bitnum>)

<compared bitnum>::= decimal number

<register affectation>::=<r-t-c affectation>

<terminal affectation>::=<r-t-c affectation>

<connection affectation>::=<r-t-c affectation>

<r-t-c affectation>::=<r-t-c name><complementation>|<r-t-c name>(<left bit>
{,<right bit>})<complementation>

<complementation>::= ! | £

<left bit>::= decimal number

<right bit>::= decimal number

<memory affectation>::=<memory name>(<memory affectation parameters>)

<memory affectation parameters>::= @<r-t-c affectation>|<constant>

<constant>::= binary number

<ass.memory affectation>::=<ass.memory read>|<ass.memory write>

<ass.memory write>::=<ass.memory name>(<memory affectation parameters>)

<ass.memory read>::=<ass.memory name>(<ass.memory read parameters>)

<ass.memory read parameters>::=<r-t-c affectation>{(<mask>),$|<r-t-c affectation>{(<mask>),@|<r-t-c affectation>{(<mask>),?
<mask>::= &<r-t-c affectation>|&<constant>

<clock description>::= H<string>;<number of phases>.
<number of phases>::= decimal number

<control graph description>::=<graph header><graph body>
<graph header>::= G<alpha-string><graph level number>,<graph number>;
<alpha-string>::= any string of letters
<graph level number>::= decimal number
<graph number>::= decimal number
<graph body>::={<node name>:<branch cluster>;}*<node name><branch cluster>
<branch cluster>::={<branch>,}*<branch>
<branch>::=<condition><signals> -> <node name>
<condition>::= /{<term>|}*<term>/
<term>::={<r-t-c affectation>&}*<r-t-c affectation>
<signals>::={<r-t-c affectation>,}*<r-t-c affectation>
<node name>::=<name>

<pla description>::=<pla header><pla body>
<pla header>::= P<alpha-string><pla level number>,<pla number>;{[<phases numbers>]}{<condition>}<pla input-output>;
<pla level number>::= decimal number
<pla number>::= decimal number
<phases numbers>::= {<phase number>|}*<phase number>
<phase number>::= decimal number
<pla input-output>::=<single input code>|<double input-output code>
<single input code>::={<r-t-c affectation>,}*<r-t-c affectation>
<double input-output code>::={<r-t-c affectation>}*<r-t-c affectation> - {<r-t-c affectation>}*<r-t-c affectation>
<pla body>::=<single code body>|<double code body>
<single code body>::={(<code*>) - <signals>}*(<code*>) - <signals>.
<double code body>::={(<code*>) - (<code>);}*(<code*> - <code>).
<code*>::= string of 0, 1 and x

```
<code>::= string of 0 and 1
<functional unit description>::=<functional unit header><functional unit body>
<functional unit header>::= F<alpha-string><functional unit level number>,
                    <functional unit number>;{[<phase number>]}{<condition>}
<functional unit level number>::= decimal number
<functional unit number>::= decimal number
<functional unit body>::={{{[phases numbers]}}{<condition>}<operations group>;}*
                          {{[phases numbers]}}{<condition>}<operations group>.
<operations group>::={<operation>,}*<operation>
<operation>::={<read argument affectation><operator>}*<read argument
               affectation><terminator><write argument affectation>
<read argument affectation>::=<r-t-c affectation>|<memory affectation>|
                               <ass.memory read affectation>
<write argument affectation>::=<r-t-c affectation>|<memory affectation>|
                               <ass.memory write affectation>
<operator>::= ! | & | ^ | + | - | * | < | >
<terminator>::= => | /<operator*>=>
<operator*>::= | | & | ^
```

APPENDIX II

Two examples of LIDO system interpretation traces:

- a fragment of interactive trace for RISC processor with the PLA based control unit (IBM-XT/MSDOS version)

- a fragment of "batch" trace for RISC processor with microprogrammed control unit (VAX750/VMS version)

a.

```
L I D O  system commands

description loading     d
contents loading        c
interpretation start    i
pla synthesis           s
mapping                 m
visualization           v

F10 to exit




continuous  / file      c
interactive / step      s

                        s


Step mode  sub-commands

load contents           c
trace                   t
mapping                 m
symbolic trace          s
clock cycle step        h
phase step              p
format                  f
visualization           v
```

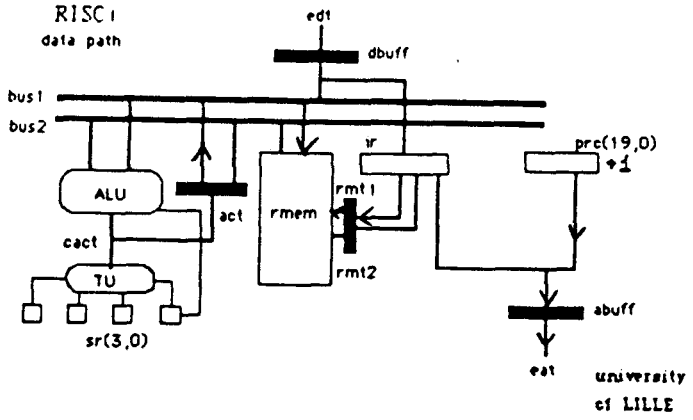
```
/* DESCRIPTION DE PROCESSEUR TYPE RISC MIPS */

Horloge; 2.          /* horloge a deux phases  NIVEAU 3# */

/* description structurelle de sequencement  NIVEAU 2# */

Pla 1,1;[1] ir(29),ir(28),ir(27),ir(26),ir(25),ir(24),re(1),re(0) -
   re(1),re(0),  sa,sual,  dtzs,dtr,dtd,dzer,  dadd,dadr,dsub,dsur,
   dou,det,dex,ddif,  dded,ddeg,ddar,dcon,  dinc,ddec,dpas,dzer,
   schr,scon,ses1,ses2;
  (xx xxxx 00) - (01 10 0000 0000 0000 0000 0000 0000);  /* "fetch"       */
  (00 0000 01) - (10 01 1110 1000 0000 0000 0000 0000);  /* addition      */
  (00 0001 01) - (10 01 1110 0100 0000 0000 0000 0000);  /* add avec ret. */
  (00 0010 01) - (10 01 1110 0010 0000 0000 0000 0000);  /* soustraction  */
  (00 0011 01) - (10 01 1110 0001 0000 0000 0000 0000);  /* sous avec ret */
  (00 0100 01) - (10 01 1000 0000 1000 0000 0000 0000);  /* OU            */
  (00 0101 01) - (10 01 1000 0000 0100 0000 0000 0000);  /* ET            */
  (00 0110 01) - (10 01 1000 0000 0010 0000 0000 0000);  /* EXOU          */
  (00 0111 01) - (10 01 1110 0000 0001 0000 0000 0000);  /* difference    */
  (00 1000 01) - (10 01 1000 0000 0000 1000 0000 0000);  /* decal.a dr.   */
  (00 1001 01) - (10 01 1000 0000 0000 0100 0000 0000);  /* decal.a g.    */
  (00 1010 01) - (10 01 1000 0000 0000 0010 0000 0000);  /* decal.aritha. */
  (00 1011 01) - (10 01 1000 0000 0000 0001 0000 0000);  /* complement.   */
  (00 1100 01) - (10 01 1110 0000 0000 0000 1000 0000);  /* incrementation*/
  (00 1101 01) - (10 01 1110 0000 0000 0000 0100 0000);  /* decrementation*/
  (00 1110 01) - (10 01 0000 0000 0000 0000 0010 0000);  /* transfert     */
  (00 1111 01) - (10 01 0000 0000 0000 0000 0001 0000);  /* 0=>drapeaux   */
  (00 xxxx 10) - (01 10 0000 0000 0000 0000 0000 1000);  /* overlapping   */
  (01 xxxx 01) - (00 00 0000 0000 0000 0000 0000 0100);  /* sauts cond.   */
  (10 xxxx 01) - (11 00 0000 0000 0000 0000 0000 0010);  /* E/S 1-er pas  */
  (10 xxxx 11) - (00 00 0000 0000 0000 0000 0000 0001).  /* E/S 2-eme pas */

/* Pla - decodage des codes de sauts conditionnels  NIVEAU 1# */

Pla 1,2;[2] /scon/ ir(26),ir(25),ir(24),r,s,d,z - dcp; /*Pla double code*/
  (0xx xxxx) - (1);      /* saut inconditionnel */
  (100 1xxx) - (1);      /* saut si retenue = 1 */
  (101 x1xx) - (1);      /* saut si signe = 1   */
  (110 xx1x) - (1);      /* saut si debord. = 1 */
  (111 xxx1) - (1).      /* saut si zero = 1    */

/* Description de Partie Operative  NIVEAU 0#  */

Fam 1,3; [2];       /* unite arithmetique et logique */

/dadd/ bus1 + bus2 => cact;
/dadr/ bus1 + bus2 + r => cact;
/dsub/ bus1 - bus2 => cact;
/dsur/ bus1 - bus2 - r => cact;
/dou/  bus1 ! bus2 => cact(31,0);
/det/  bus1 & bus2 => cact(31,0);
/dex/  bus1 ^ bus2 => cact(31,0);
/ddif/ bus1 - bus2 => cact;
/dded/ r => cact(31), bus1(31,1) => cact(30,0), bus1(0) => r;
/ddeg/ r => cact(0), bus1(30,0) => cact(31,1), bus1(31) => r;
/ddar/ bus1(31) => cact(31), bus1(31,1) => cact(30,0), bus1(0) => r;
/dcon/ bus1(31,0)! => cact(31,0);
/dinc/ bus1 + 1 => cact;
/ddec/ bus1 - 1 => cact;
/dpas/ bus1 => cact(31,0).
```

```
Fam 1,4; [2];       /* unite de tests arithoetiques */

/dzer/ 0 => r, 0 => s, 0 => d, 0 => z ;
/dtzs/ cact(31) => s, cact /! => z!;
/dtd/  bus1(31) & bus2(31) & cact(31)! => cdn,   /* debord. negatif */
       bus1(31)!& bus2(31)!& cact(31)  => cdp,   /* debord. positif */
       cdn ! cdp => d;
/dtr/  cact(32) => r.

Fam 1,5; [2];       /* chargeoent de l'accuoulateur temporaire act(31,0) */

/sual/ cact(31,0) => act.

Fam 1,2; [2];       /* unite des transferts par bus */

/sa/   prc => eat(19,0), eaea(@eat) => ir, prc + 1 => prc;
/sual/ ir(23,20) => rat1, ir(19,16) => rat2,
       raea(@rat1) => bus1, raea(@rat2) => bus2;
/schr & ddif!/ act => bus1, bus1 => raea(@rat1);
/scon/ ir(19,0) => prc;                    /* adresse de saut */
/ses1 & ir(24)!/ 1 => erd, ir(19,0) => eat(19,0), eaea(@eat) => bus1;
/ses1 & ir(24) / ir(23,20) => rat1, raea(@rat1) => bus1;
/ses2 & ir(24)!/ ir(23,20) => rat1, bus1 => raea(@rat1);
/ses2 & ir(24) / 1 => ewr, ir(19,0) => eat(19,0), bus1 => eaea(@eat).



Hemoires; raea(16,32);      /* oemoire de registres interne */
          eaea(32,32).      /* oemoire de programme externe */

Registres; ir(32);          /* registre d'instructions */
           re(2);           /* registre d'etats        */
           s;d;r;z;         /* reg. drapeaux           */
           prc(20).         /* compteur de programme   */

Terminaux;  bus1(32); bus2(32);        /* bus internes */
            act(32);        /* accumulateur temporaire */
            rat1(4); rat2(4);  /* adresses de registres */
            eat(20);    /* external terminal d'adresses */
            sa;sual;scon;ses1;ses2;schr;
            dadd;dadr;dsub;dsur;dou;det;
            dex;ddif;dded;ddeg;ddar;dcoe;
            dinc;ddec;dpas;dzer;dtzs;dtr;
            dtd.

Connecteurs; dcp;
             erd;       /*external externe de lecture*/
             ewr;       /*terminal externe d'ecriture*/
             cact(33);     /*sortie de UAL */
             cdn;cdp.

End.
```

```
 commands: c,t,a,s,h,p,f,v ?  h

!!    TRACE     cycle 3   phase 1    last phase 4 sec.
!!    TRACE     cycle 3   phase 2    last phase 1 sec.
$ prc=>eat(19,0),emem(@eat)=>ir,prc+1=>prc
$ act=>bus1,bus1=>rmem(@rat1)
 commands: c,t,a,s,h,p,f,v ?

!!    TRACE     cycle 5   phase 1    last phase 4 sec.
!!    TRACE     cycle 5   phase 2    last phase 1 sec.
$ prc=>eat(19,0),emem(@eat)=>ir,prc+1=>prc
$ act=>bus1,bus1=>rmem(@rat1)
 commands: c,t,a,s,h,p,f,v ?  p

!!    TRACE     cycle 6   phase 1    last phase 3 sec.
 commands: c,t,a,s,h,p,f,v ?  p

!!    TRACE     cycle 6   phase 2    last phase 1 sec.
$ ir(23,20)=>rat1,ir(19,16)=>rat2,rmem(@rat1)=>bus1,rmem(@rat2)=>bus2
$ bus1+bus2=>cact
$ cact(31)=>s,cact/!=>z!
$ bus1(31)&bus2(31)&cact(31)!=>cdn,bus1(31)!&bus2(31)!&cact(31)=>cdp,cdn!cdp=>d
$ cact(32)=>r
$ cact(31,0)=>act
 commands: c,t,a,s,h,p,f,v ?  p

!!    TRACE     cycle 7   phase 1    last phase 4 sec.
 commands: c,t,a,s,h,p,f,v ?  p

!!    TRACE     cycle 7   phase 2    last phase 1 sec.
$ prc=>eat(19,0),emem(@eat)=>ir,prc+1=>prc
$ act=>bus1,bus1=>rmem(@rat1)
```

Un fragment de la trace symbolique.

```
commands: c,t,a,s,h,p,f,v ?  p

!!    TRACE     cycle 9   phase 1    last phase 4 sec.
prc         0000  0000  0000  0000  0100
          » 0000  0000  0000  0000  0100
ir          0000  0000  0000  0000  0000  0000  0000  0000
          » 0000  0000  0000  0000  0000  0000  0000  0000
bus1        0000  0000  0000  0000  0000  0000  0000  0000
bus2        0000  0000  0000  0000  0000  0000  0000  0000
act         0000  0000  0000  0000  0000  0000  0000  0000
commands: c,t,a,s,h,p,f,v ?  p

!!    TRACE     cycle 9   phase 2    last phase 0 sec.
$ prc=>eat(19,0),emem(@eat)=>ir,prc+1=>prc
$ act=>bus1,bus1=>rmem(@rat1)
prc         0000  0000  0000  0000  0100
          » 0000  0000  0000  0000  0101
ir          0000  0000  0000  0000  0000  0000  0000  0000
          » 0000  0000  0000  0000  0000  0000  0000  0000
bus1        0000  0000  0000  0000  0000  0000  0000  0000
bus2        0000  0000  0000  0000  0000  0000  0000  0000
act         0000  0000  0000  0000  0000  0000  0000  0000
```



Un fragment de la trace profonde et de la trace mixte.

```
commands: c,t,m,s,h,p,f,v ?

mem
0000  0000  0000  0000  0000  0000  0000  0000    a: 0
0000  0000  0000  0000  0000  0000  0000  0000    a: 1
0000  0000  0000  0000  0000  0000  0000  0000    a: 2
0000  0000  0000  0000  0000  0000  0000  0000    a: 3
0000  0000  0000  0000  0000  0000  0000  0000    a: 4
0000  0000  0000  0000  0000  0000  0000  0000    a: 5
0000  0000  0000  0000  0000  0000  0000  0000    a: 6
0000  0000  0000  0000  0000  0000  0000  0000    a: 7
0000  0000  0000  0000  0000  0000  0000  0000    a: 8
0000  0000  0000  0000  0000  0000  0000  0000    a: 9
0000  0000  0000  0000  0000  0000  0000  0000    a: 10
0000  0000  0000  0000  0000  0000  0000  0000    a: 11
0000  0000  0000  0000  0000  0000  0000  0000    a: 12
0000  0000  0000  0000  0000  0000  0000  0000    a: 13
0000  0000  0000  0000  0000  0000  0000  0000    a: 14
0000  0000  0000  0000  0000  0000  0000  0000    a: 15

rmem
0111  1111  1111  1111  1111  1111  1111  1111    a: 0
0000  0000  0000  0000  0000  0000  0000  0011    a: 1
0000  0000  0000  0000  0000  0000  0000  0001    a: 2
0000  0000  0000  0000  0000  0000  0000  0000    a: 3

commands: c,t,m,s,h,p,f,v ?
```

Un ecran de la "map" sélective.

```
 commands: c,t,m,s,h,p,f,v ?  p

##    TRACE     cycle 11   phase 2    last phase 1 sec.
$ prc=>eat(19,0),emem(@eat)=>ir,prc+1=>prc
$ act=>bus1,bus1=>rmem(@rat1)
 sa         1
 sual       0
 dadd       0
 bus1       1111  1111  1111  1111  1111  1111  1111  1110
 bus2       0111  1111  1111  1111  1111  1111  1111  1111
 act        1111  1111  1111  1111  1111  1111  1111  1110
 prc        0000  0000  0000  0000  0101
          > 0000  0000  0000  0000  0110

 commands: c,t,m,s,h,p,f,v ?  h

##    TRACE     cycle 14   phase 1    last phase 3 sec.
 sa         0
 sual       1
 dadd       1
 schr       0
 bus1       1111  1111  1111  1111  1111  1111  1111  1100
 bus2       1111  1111  1111  1111  1111  1111  1111  1110
 act        1111  1111  1111  1111  1111  1111  1111  1100
##    TRACE     cycle 14   phase 2    last phase 1 sec.
$ ir(23,20)=>rat1,ir(19,16)=>rat2,rmem(@rat1)=>bus1,rmem(@rat2)=>bus2
$ bus1+bus2=>cact
$ cact(31)=>s,cact/!=>z!
$ bus1(31)&bus2(31)&cact(31)!=>cdn,bus1(31)!&bus2(31)!&cact(31)=>cdp,cdn!cdp=>d
$ cact(32)=>r
$ cact(31,0)=>act
 sa         0
 sual       1
 dadd       1
 schr       0
 bus1       1111  1111  1111  1111  1111  1111  1111  1100
 bus2       1111  1111  1111  1111  1111  1111  1111  1100
 act        1111  1111  1111  1111  1111  1111  1111  1000
```

Deux fragments de la trace mixte avec le display des signaux de contrôle (sa- fetch, sual- l'activation de l'UAL, dadr- l'addition avec la retenue, schr- chargement du résultat "overlapping" avec fetch)

b.

```
        /* mp32m - version microprogrammee */

          /* microprogrammed sequencer */

Horloge;2.

Pla 1,1; mire(0),mist - mist,sinfe,sma1,sma2;
                ( x0 )    -    ( 1100 );
                ( 01 )    -    ( 0010 );
                ( 11 )    -    ( 0001 ).

Pla 1,2; [2] mire(6),mire(5),mire(4);
(xxx) - sa!,sual!,scon!,schr!,ses1!,ses2!;
(000) - sa;
(001) - sual;
(010) - schr;
(011) - scon;
(100) - ses1;
(101) - ses2.

Fun 1,1;;
/sinfe/ mime(&armi) =) mire(6,0);
/sma1/ mire(3,1) =) armi(2,0);
/sma2/ ir(29,28) =) armi(2,1), 0 =) armi(0).


Registers;ir(32); /* instruction register*/
    mist ;   /* microstate */
    armi(3). /* microinstr. address register */

Memory;mime(8,7).   /* micro-memory 8 words 7 bits */

Connectors; sinfe;sma1;sma2.

Terminals;armi(3);     /* microaddress latch */
          mire(7);  /* microinstruction latch */
          sa;sual;scon;schr;ses1;ses2.


/* description structurelle du decodage des codes de controle  NIVEAU 1 */

   /* matrice logique - decodage des operations UAL */

     Pla 2,1; [2] /sual/ ir(27),ir(26),ir(25),ir(24);    /*  Pla simple code */

(0000) - dadd,dtzs,dtr,dtd;              /* signal d'addition */
(0001) - dadr,dtzs,dtr,dtd;              /*   - -  d'addition avec retenue */
(0010) - dsub,dtzs,dtr,dtd;              /*   - -  de soustraction */
(0011) - dsur,dtzs,dtr,dtd;              /*   - -  de soustraction avec ret */
(0100) - dou,dtzs;                       /*   - -  de OU */
(0101) - det,dtzs;                       /*   - -  de ET */
(0110) - dex,dtzs;                       /*   - -  de EXOU */
(0111) - ddif,dtzs,dtr,dtd;              /*   - -  de difference */
(1000) - dded,dtzs;                      /*   - -  de decalage a droite */
(1001) - ddeg,dtzs;                      /*   - -  de decalage a gauche */
(1010) - ddar,dtzs;                      /*   - -  de decalage arith. a dr. */
(1011) - dcom,dtzs;                      /*   - -  de complementation */
(1100) - dinc,dtzs,dtr,dtd;              /*   - -  d'incrementation */
(1101) - ddec,dtzs,dtr,dtd;              /*   - -  de decrementation */
(1110) - dpas;                           /*   - -  de transfert */
(1111) - dzer.                           /*  mise a zero des indicateurs */
/* dtzs -test de zero-signe; dtr -test de retenue; dtd -test de debordement */


      /* matrice logique - decodage des codes de sauts conditionnels */

 Pla 2,2; [2] /scon/ ir(26),ir(25),ir(24),r, s,d,z - dcp;  /* Pla double code */

    (10xxxxxx) - (1);                    /* saut inconditionnel */
    (11001xxx) - (1);                    /* saut si retenue = 1 */
    (11010x1xx) - (1);                   /* saut si signe = 1   */
    (11110xx1x) - (1);                   /* saut si debordement = 1 */
    (11111xxx1) - (1).                   /* saut si zero = 1 */
```

```
            /* description de Partie Operative  NIVEAU 0  */

               /* unite de transferts internes et externes */

   Fun 2,1; [2];
/sa/  prc => eat(19,0), emem(àeat) => ir, prc + 1 => prc;
/sual/ ir(23,20) => rat1; ir(19,16) => rat2;
       rmem(àrat1) => bus1; rmem(àrat2) => bus2;
/schr/ act => bus1; bus1 => rmem(àrat1);
/scon/ ir(19,0) => prc;                          /* adresse de saut */
/ses1 & ir(24)!/ 1 => erd; ir(19,0) => eat(19,0); emem(àeat) => bus1;
/ses1 & ir(24) / ir(23,20) => rat1; rmem(àrat1) => bus1;
/ses2 & ir(24)!/ ir(23,20) => rat1; bus1 => rmem(àrat1);
/ses2 & ir(24) / 1 => ewr; ir(19,0) => eat(19,0); bus1 => emem(àeat).

            /* unite arithmetique et logique */

   Fun 2,2; [2] ;                          /* unite fonctionnelle */

/dadd/ bus1 + bus2 => cact;
/dadr/ bus1 + bus2 + r => cact;
/dsub/ bus1 - bus2 => cact;
/dsur/ bus1 - bus2 - r => cact;
/dou/  bus1 ù bus2 => cact(31,0);
/det/  bus1 & bus2 => cact(31,0);
/dex/  bus1 ^ bus2 => cact(31,0);
/ddif/ bus1 - bus2 => cact;
/dded/ r => cact(31), bus1(31,1) => cact(30,0), bus1(0) => r;
/ddeg/ r => cact(0), bus1(30,0) => cact(31,1), bus1(31) => r;
/ddar/ bus1(31) => cact(31), bus1(31,1) => cact(30,0), bus1(0) => r;
/dcom/ bus1(31,0)! => cact(31,0);
/dinc/ bus1 + 1 => cact;
/ddec/ bus1 - 1 => cact;
/dpas/ bus1 => cact(31,0).

            /* unite de test arithmetiques */

   Fun 2,3; [2];

/dzer/ 0 => r, 0 => s, 0 => d, 0 => z ;
/dtzs/ cact(31) => s, cact /ù => z!;
/dtd/  bus1(31) & bus2(31) & cact(31)! => cdn,   /* deb. negatif */
       bus1(31)!& bus2(31)!& cact(31)  => cdp,   /* deb. positif */
       cdn ù cdp => d;
/dtr/  cact(32) => r.


            /* chargement d'accumulateur temporaire act(31,0) */

   Fun 2,4; [2];

/sual/ cact(31,0) => act .


Memoires; rmem(16,32);       /* memoire de registres interne */
          emem(32,32).       /* memoire de programme externe */

Registres; s;d;r;z;  /* reg. drapeaux */
           prc(20).          /* compteur de programme */

Terminaux; bus1(32); bus2(32);   /* bus internes */
           act(32);          /* accumulateur temporaire */
           rat1(4); rat2(4);     /* adresses de registres */
           eat(20).          /* external terminal d'adresses */

Connecteurs; dadd;dadr;dsub;dsur;dou;det;
             dex;ddif;dded;ddeg;ddar;dcom;
             dinc;ddec;dpas;dzer;dtzs;dtr;
             dtd;dcp;
             erd;        /*external externe de lecture*/
             ewr;        /*terminal externe d'ecriture*/
             cact(33);      /*sortie de UAL */
             cdn;cdp.


  End.
```

```
****  map file name :  mp32m.m  ****

***   map of carriers content ***

** map of memories content **

 mime
001  0010   a: 0
010  1110   a: 1
011  1110   a: 2
000  0000   a: 3
100  1010   a: 4
101  1110   a: 5
000  0000   a: 6
000  0001   a: 7

 rmem
0000  0000  0000  0000  0000  0000  0000  0011   a: 0
0000  0000  0000  0000  0000  0000  0000  1111   a: 1
0000  0000  0000  0000  0000  0000  0000  1010   a: 2
0000  0000  0000  0000  0000  0000  0000  0000   a: 3
0000  0000  0000  0000  0000  0000  0000  0000   a: 4
0000  0000  0000  0000  0000  0000  0000  0000   a: 5
0000  0000  0000  0000  0000  0000  0000  0000   a: 6
0000  0000  0000  0000  0000  0000  0000  0000   a: 7
0000  0000  0000  0000  0000  0000  0000  0000   a: 8
0000  0000  0000  0000  0000  0000  0000  0000   a: 9
0000  0000  0000  0000  0000  0000  0000  0000   a: 10
0000  0000  0000  0000  0000  0000  0000  0000   a: 11
0000  0000  0000  0000  0000  0000  0000  0000   a: 12
0000  0000  0000  0000  0000  0000  0000  0000   a: 13
0000  0000  0000  0000  0000  0000  0000  0000   a: 14
0000  0000  0000  0000  0000  0000  0000  0000   a: 15

 mmem
0000  0000  0000  0000  0000  0000  0000  0000   a: 0
0000  0000  0000  0000  0000  0000  0000  0000   a: 1
0000  0000  0000  0000  0000  0000  0000  0000   a: 2
0000  0000  0000  0000  0000  0000  0000  0000   a: 3
0000  0000  0000  0000  0000  0000  0000  0000   a: 4
0000  0000  0000  0000  0000  0000  0000  0000   a: 5
0000  0000  0000  0000  0000  0000  0000  0000   a: 6
0000  0000  0000  0000  0000  0000  0000  0000   a: 7
0000  0000  0000  0000  0000  0000  0000  0000   a: 8
0000  0000  0000  0000  0000  0000  0000  0000   a: 9
0000  0000  0000  0000  0000  0000  0000  0000   a: 10
0000  0000  0000  0000  0000  0000  0000  0000   a: 11
0000  0000  0000  0000  0000  0000  0000  0000   a: 12
0000  0000  0000  0000  0000  0000  0000  0000   a: 13
0000  0000  0000  0000  0000  0000  0000  0000   a: 14
0000  0000  0000  0000  0000  0000  0000  0000   a: 15
0000  0000  0000  0000  0000  0000  0000  0000   a: 16
0000  0000  0000  0000  0000  0000  0000  0000   a: 17
0000  0000  0000  0000  0000  0000  0000  0000   a: 18
0000  0000  0000  0000  0000  0000  0000  0000   a: 19
0000  0000  0000  0000  0000  0000  0000  0000   a: 20
0000  0000  0000  0000  0000  0000  0000  0000   a: 21
0000  0000  0000  0000  0000  0000  0000  0000   a: 22
0000  0000  0000  0000  0000  0000  0000  0000   a: 23
0000  0000  0000  0000  0000  0000  0000  0000   a: 24
0000  0000  0000  0000  0000  0000  0000  0000   a: 25
0000  0000  0000  0000  0000  0000  0000  0000   a: 26
0000  0000  0000  0000  0000  0000  0000  0000   a: 27
0000  0000  0000  0000  0000  0000  0000  0000   a: 28
0000  0000  0000  0000  0000  0000  0000  0000   a: 29
0000  0000  0000  0000  0000  0000  0000  0000   a: 30
0000  0000  0000  0000  0000  0000  0000  0000   a: 31
```

```
 ** map of associative memories content **

 ** map of registers content **

ir      0000  0000  0000  0000  0000  0000  0000  0000
mist    0
armi    000
s       0
d       0
r       0
z       0
prc     0000  0000  0000  0000  0000
 ** map of terminals content **

armi     000
mire     000  0000
sa       0
sual     0
scon     0
schr     0
ses1     0
ses2     0
bus1     0000  0000  0000  0000  0000  0000  0000  0000
bus2     0000  0000  0000  0000  0000  0000  0000  0000
act      0000  0000  0000  0000  0000  0000  0000  0000
rat1     0000
rat2     0000
eat      0000  0000  0000  0000  0000
 ** map of connectors content **

sinfe    0
sma1     0
sma2     0
dadd     0
dadr     0
dsub     0
dsur     0
dou      0
det      0
dex      0
ddif     0
dded     0
ddeg     0
ddar     0
dcom     0
dinc     0
ddec     0
dpas     0
dzer     0
dtzs     0
dtr      0
dtd      0
dop      0
erd      0
ewr      0
cact     0  0000  0000  0000  0000  0000  0000  0000  0000
cdn      0
cdp      0
```

```
***** start of interpretation trace *****


** clock cycle  1   clock phase  1 **

activated Pla level 1 , Pla nr 1
mist          slave :  0
             master :  1
sinfe     1
sma1      0
sma2      0
activated Fun-level 1 Fun nr 1
* source operations:  mime(àarmi)=)mire(6,0)
mire      001  0010

** clock cycle  1   clock phase  2 **

activated Pla level 1 , Pla nr 1
mist          slave :  1
             master :  0
sinfe     0
sma1      1
sma2      0
activated Pla level 1 , Pla nr 2
sa        0
sual      0
scon      0
schr      0
ses1      0
ses2      0
sual      1
activated Pla level 2 , Pla nr 1
dadd      1
dtzs      1
dtr       1
dtd       1
activated Fun-level 1 Fun nr 1
* source operations:  mire(3,1)=)armi(2,0)
armi          slave :  000
             master :  001
activated Fun-level 2 Fun nr 1
* source operations:  ir(23,20)=)rat1,ir(19,16)=)rat2,rmem(àrat1)=)bus1,rmem(àrat2)=)bus2
rat1          0000
rat2          0000
bus1          0000  0000  0000  0000  0000  0000  0000  0011
bus2          0000  0000  0000  0000  0000  0000  0000  0011
activated Fun-level 2 Fun nr 2
* source operations:  bus1+bus2=)cact
cact      0  0000  0000  0000  0000  0000  0000  0000  0110
activated Fun-level 2 Fun nr 3
* source operations:  cact(31)=)s,cact/ù=)z!
s             slave :  0
             master :  0
z             slave :  0
             master :  0
* source operations:  bus1(31)&bus2(31)&cact(31)!=)cdn,bus1(31)!&bus2(31)!&cact(31)=)cdp,cdnùcdp=)d
cdn       0
cdp       0
d             slave :  0
             master :  0
* source operations:  cact(32)=)r
r             slave :  0
             master :  0
activated Fun-level 2 Fun nr 4
* source operations:  cact(31,0)=)act
act           0000  0000  0000  0000  0000  0000  0000  0110

** clock cycle  2   clock phase  1 **

activated Pla level 1 , Pla nr 1
mist          slave :  0
             master :  1
sinfe     1
sma1      0
sma2      0
activated Fun-level 1 Fun nr 1
* source operations:  mime(àarmi)=)mire(6,0)
mire      010  1110
```

```
** clock cycle  2   clock phase  2 **

activated Pla level 1 , Pla nr 1
mist          slave :  1
              master :  0
sinfe      0
sma1       1
sma2       0
activated Pla level 1 , Pla nr 2
sa         0
sual       0
scon       0
schr       0
ses1       0
ses2       0
schr       1
activated Fun-level 1 Fun nr 1
* source operations:  mire(3,1)=)armi(2,0)
armi          slave :  001
              master :  111
activated Fun-level 2 Fun nr 1
* source operations:  act=)bus1,bus1=)rmem(àrat1)
bus1          0000  0000  0000  0000  0000  0000  0000  0110
rmem          0000  0000  0000  0000  0000  0000  0000  0110  à: 0
activated Fun-level 2 Fun nr 2
activated Fun-level 2 Fun nr 3
activated Fun-level 2 Fun nr 4

** clock cycle  3   clock phase  1 **

activated Pla level 1 , Pla nr 1
mist          slave :  0
              master :  1
sinfe      1
sma1       0
sma2       0
activated Fun-level 1 Fun nr 1
* source operations:  mime(àarmi)=)mire(6,0)
mire       000  0001

** clock cycle  3   clock phase  2 **

activated Pla level 1 , Pla nr 1
mist          slave :  1
              master :  0
sinfe      0
sma1       0
sma2       1
activated Pla level 1 , Pla nr 2
sa         0
sual       0
scon       0
schr       0
ses1       0
ses2       0
sa         1
activated Fun-level 1 Fun nr 1
* source operations:  ir(29,28)=)armi(2,1),0=)armi(0)
armi          slave :  111
              master :  001
armi          slave :  111
              master :  000
activated Fun-level 2 Fun nr 1
* source operations:  prc=)eat(19,0),emem(àeat)=)ir,prc+1=)prc
eat           0000  0000  0000  0000  0000
ir             slave :    0000  0000  0000  0000  0000  0000  0000  0000
              master :    0000  0000  0000  0000  0000  0000  0000  0000
prc            slave :    0000  0000  0000  0000  0000
              master :    0000  0000  0000  0000  0001
activated Fun-level 2 Fun nr 2
activated Fun-level 2 Fun nr 3
activated Fun-level 2 Fun nr 4
```

```
** clock cycle  4   clock phase  1 **

activated Pla level 1 , Pla nr 1
mist          slave :  0
             master :  1
sinfe      1
sma1       0
sma2       0
activated Fun-level 1 Fun nr 1
* source operations:  mime(àarmi)=)mire(6,0)
mire       001  0010

** clock cycle  4   clock phase  2 **

activated Pla level 1 , Pla nr 1
mist          slave :  1
             master :  0
sinfe      0
sma1       1
sma2       0
activated Pla level 1 , Pla nr 2
sa         0
sual       0
scon       0
schr       0
ses1       0
ses2       0
sual       1
activated Pla level 2 , Pla nr 1
dadd       1
dtzs       1
dtr        1
dtd        1
activated Fun-level 1 Fun nr 1
* source operations:  mire(3,1)=)armi(2,0)
armi          slave :  000
             master :  001
activated Fun-level 2 Fun nr 1
* source operations:  ir(23,20)=)rat1,ir(19,16)=)rat2,rmem(àrat1)=)bus1,rmem(àrat2)=)bus2
rat1          0000
rat2          0000
bus1          0000  0000  0000  0000  0000  0000  0000  0110
bus2          0000  0000  0000  0000  0000  0000  0000  0110
activated Fun-level 2 Fun nr 2
* source operations:  bus1+bus2=)cact
cact       0  0000  0000  0000  0000  0000  0000  1100
activated Fun-level 2 Fun nr 3
* source operations:  cact(31)=)s,cact/ù=)z!
s             slave :  0
             master :  0
z             slave :  0
             master :  0
* source operations:  bus1(31)&bus2(31)&cact(31)!=)cdn,bus1(31)!&bus2(31)!&cact(31)=)cdp,cdnùcdp=)d
cdn        0
cdp        0
d             slave :  0
             master :  0
* source operations:  cact(32)=)r
r             slave :  0
             master :  0
activated Fun-level 2 Fun nr 4
* source operations:  cact(31,0)=)act
act           0000  0000  0000  0000  0000  0000  0000  1100

** clock cycle  5   clock phase  1 **

activated Pla level 1 , Pla nr 1
mist          slave :  0
             master :  1
sinfe      1
sma1       0
sma2       0
activated Fun-level 1 Fun nr 1
* source operations:  mime(àarmi)=)mire(6,0)
mire       010  1110
```

```
** clock cycle  5   clock phase  2 **

activated Pla level 1 , Pla nr 1
mist          slave :  1
              master :  0
sinfe      0
sma1       1
sma2       0
activated Pla level 1 , Pla nr 2
sa         0
sual       0
scon       0
schr       0
ses1       0
ses2       0
schr       1
activated Fun-level 1 Fun nr 1
* source operations:  mire(3,1)=)armi(2,0)
armi          slave :  001
              master :  111
activated Fun-level 2 Fun nr 1
* source operations:  act=)bus1,bus1=)rmem(àrat1)
bus1          0000  0000  0000  0000  0000  0000  0000  1100
rmem          0000  0000  0000  0000  0000  0000  0000  1100  à: 0
activated Fun-level 2 Fun nr 2
activated Fun-level 2 Fun nr 3
activated Fun-level 2 Fun nr 4

** clock cycle  6   clock phase  1 **

activated Pla level 1 , Pla nr 1
mist          slave :  0
              master :  1
sinfe      1
sma1       0
sma2       0
activated Fun-level 1 Fun nr 1
* source operations:  mime(àarmi)=)mire(6,0)
mire       000  0001

** clock cycle  6   clock phase  2 **

activated Pla level 1 , Pla nr 1
mist          slave :  1
              master :  0
sinfe      0
sma1       0
sma2       1
activated Pla level 1 , Pla nr 2
sa         0
sual       0
scon       0
schr       0
ses1       0
ses2       0
sa         1
activated Fun-level 1 Fun nr 1
* source operations:  ir(29,28)=)armi(2,1),0=)armi(0)
armi          slave :  111
              master :  001
armi          slave :  111
              master :  000
activated Fun-level 2 Fun nr 1
* source operations:  prc=)eat(19,0),emem(àeat)=)ir,prc+1=)prc
eat           0000  0000  0000  0000  0001
ir             slave :    0000  0000  0000  0000  0000  0000  0000  0000
              master :    0000  0000  0000  0000  0000  0000  0000  0000
prc            slave :    0000  0000  0000  0000  0001
              master :    0000  0000  0000  0000  0010
activated Fun-level 2 Fun nr 2
activated Fun-level 2 Fun nr 3
activated Fun-level 2 Fun nr 4
* end of interpretation trace LIDO *
```