Définition, étude et conception d'un microprocesseur autotestable spécifique: COBRA

Adham Osseiran

THESE

présentée par

Adham OSSEIRAN

pour obtenir le titre de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(arrêté ministériel du 5 Juillet 1984)

(Spécialité : Microélectronique)

DEFINITION, ETUDE ET CONCEPTION D'UN MICROPROCESSEUR AUTOTESTABLE SPECIFIQUE

COBRA

Date de soutenance : 12 Mai 1986

Composition du Jury :	M.	MAZARE	Guy	Président
	MM.	COURTOIS	Bernard	
		GEFFROY	Jean-Claude	Examinateurs
		LARCHER	Simon	
		LETRUNG	Bao	

Thèse préparée au sein du laboratoire TIM3 / INPG.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH
Vice-Présidents : B. BAUDELET
R. CARRE
H. CHERADAME
J.M. PIERRARD

Année universitaire 1985-1986

Professeurs des Universités

E.N.S.E.E.G.

BEAUFILS	Jean-claude	LOUCHET	François
BESSON	Jean	PARIAUD	Jean-Charles
BONNETAIN	Lucien	RAMEAU	Jean-Jacques
BONNIER	Etienne	SOHM	Jean-Claude
DURAND	Françis	SOUQUET	Jean-Louis
GUYOT	Pierre		

E.N.S.E.R.G.

BARIBAUD	Michel	COUMES	André
BLIMAN	Samuel	GENTIL	Pierre
BUYLE BODIN	Maurice	GUERIN	Bernard
CHENEVIER	Pierre	POUPOT	Christian
COHEN	Joseph	SERMET	Pierre
COUMES	André		

E.N.S.I.E.G.

BARRAUD	Alain	JOUBERT	Jean-Claude
BAUDELET	Bernard	JOURDAIN	Geneviève
BLOCH	Daniel	LACOUME	Jean-Louis
BRISSONNEAU	Pierre	LONGEQUEUE	Jean-Pierre
BRUNET	Yves	MASSELOT	Christian
CAVAIGNAC	Jean-François	MORET	Roger
CHARTIER	Germain	PAUTHENET	René
CHERUY	Arlette	PERRET	René
DURAND	Jean-Louis	PERRET	Robert
FOULARD	Claude	POLOUJADOFF	Michel
GAUBERT	Claude	SABONNADIERE	Jean-Claude
IVANES	Marcel	SCHLENKER	Claire
JALINIER	Jean-Michel	SCHLENKER	Michel
JAUSSAUD	Pierre		

E.N.S.H.G.

BOIS	Philippe	LESPINARD	Georges
BOUVARD	Maurice	MOREAU	René
CAILLERIE	Denis	OBLED	Charles
LESIEUR	Marcel	PIAU	Jean-Michel
TROMPETTE	Philippe		

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

FONLUPT	Jean	ROBERT	François
MAZARE	Guy	SAUCIER	Gabrielle
MOSSIERE	Jacques	VEILLON	Gérard

U.E.R.M.C.P.P.

CHERADAME	Hervé	RENAUD	Maurice
CHIAVERINA	Jean	ROBERT	André
GANDINI	Alessandro	SILVY	Jacques

Professeurs Associés

BLACKWELDER	Ronald	ENSHG
HAYASHI	Hirashi	ENSIEG
PURDY	Gary	ENSEEG

Professeurs à l'Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S.

Directeurs de recherche :

CAILLET	Marcel
CARRE	René
FRUCHART	Robert
JORRAND	Philippe
LANDAU	Ioan

Maître de recherche :

ALLIBERT	Colette	GIVORD	Dominique
ALLIBERT	Michel	EUSTATHOPOULOS	Nicolas
ANSARA	Ibrahim	JOUD	Jean-Charles
ARMAND	Michel	KAMARINOS	Georges
BINDER	Gilbert	KLEITZ	Michel
BONNET	Roland	LEJEUNE	Gérard
BORNARD	Guy	MERMET	Jean
CALMET	Jacques	MUNIER	Jacques
DAVID	René	SENATEUR	Jean-Pierre
DESPORTES	Jacques	SUERY	Michel
DRIOLE	Jean	WACK	Bernard

Personnalités agréées à titre permanent à diriger des travaux de recherche
(Décision du conseil scientifique)

E.N.S.E.E.G.

BERNARD	Claude	MALMEJAC	Yves (CENG)
CAILLET	Marcel	MARTIN GARIN	Régina
CHATILLON	Catherine	NGUYEN TRUONG	Bernadette
CHATILLON	Christian	RAVAINE	Denis
COULON	Michel	SAINFORT	Paul (CENG)
DIARD	Jean-Paul	SARRAZIN	Pierre
FOSTER	Panayotis	SIMON	Jean-Paul
GALERIE	Alain	TOUZAIN	Philippe
HAMMOU	Abdelkader	URBAIN	Georges(ODEILLO)

E.N.S.E.R.G.

BOREL	Joseph	DOLMAZON	Jean-Marc
CHOVET	Alain	HERAULT	Jeanny

E.N.S.I.E.G.

BORNARD	Guy	LEJEUNE	Gérard
DESCHIZEAUX	Pierre	MAZUER	Jean
GLANGEAUD	François	PERARD	Jacques
KOFMAN	Walter	REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine	MICHEL	Jean-Marie
BOIS	Daniel	ROWE	Alain
DARVE	Félix	VAUCLIN	Michel

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

BERT	Didier	DELLA DORA	Jean
CALMET	Jacques	FONLUPT	Jean
COURTIN	Jacques	SIFAKIS	Joseph
COURTOIS	Bernard		

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean	NIFENECKER	Hervé
COEURE	Philippe (LETI)	PERROUD	Paul
DELHAYE	Jean-Marc (STT)	PEUZIN	Jean-Claude(LETI)
DUPUY	Michel (LETI)	TAIEB	Maurice
JOUVE	Hubert (LETI)	VINCENDON	Marc
NICOLAU	Yvan (LETI)		

Laboratoires extérieurs

C.N.E.T.

DEMOULIN	Eric	MERCKEL	Gérard
DEVINE	R.A.B.	PAULEAU	Yves
GERBER	Roland		

****

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: M. M. MERMET
Directeur des Etudes et de la formation	: M. J. LEVASSEUR
Directeur des Recherches	: M. J. LEVY
Secrétaire Général	: Mle M. CLERGUE

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique-Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

AVANT-PROPOS

Je tiens à exprimer ma reconnaissance à :

Monsieur Guy MAZARE, Professeur à l'ENSIMAG et rapporteur de cette thèse, pour l'honneur qu'il me fait en présidant le jury de cette thèse.

Monsieur Bernard COURTOIS, Chargé de recherche au CNRS et Directeur de l'Equipe de recherche en Architecture d'Ordinateurs dans laquelle ce projet a évolué dans les meilleures conditions.

Monsieur Jean-Claude GEFFROY, Professeur à l'INSA de Toulouse pour avoir accepté d'être rapporteur de ce travail.

Monsieur Simon LARCHER, Directeur scientifique à la Compagnie des Signaux et d'Entreprises Electriques (CSEE) à Paris, pour avoir accepté d'être membre du jury de cette thèse.

Monsieur Bao LETRUNG, Directeur de recherche à l'Institut National de Recherche sur les Transports et leur Sécurité (INRETS, précédemment Institut de Recherche des Transports, IRT), pour avoir accepté d'être membre de ce jury, et surtout pour les conseils et les remarques qui m'ont orienté au cours des discussions que nous avons pu avoir.

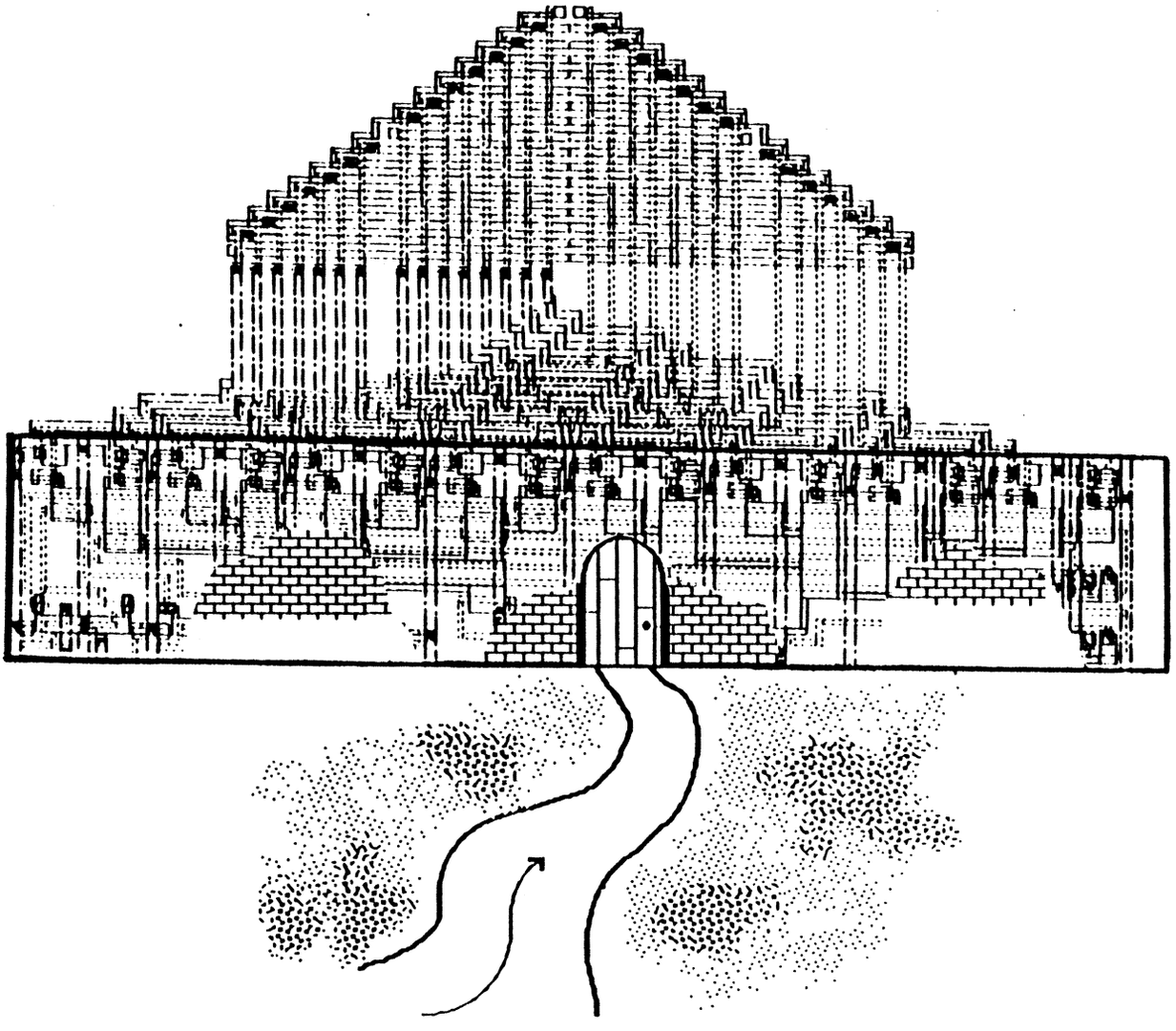
Je tiens à remercier Monsieur Dominique BIED-CHARRETON de l'INRETS de m'avoir aidé, par ses remarques pertinentes et la maîtrise parfaite de son rôle, à faire avancer le projet. Je ne saurais oublier dans ces remerciements Madame STUPARU de l'INRETS et Monsieur DAVID, Directeur du CRESTA.

Je tiens particulièrement à remercier mon collègue et ami Mihaïl NICOLAIDIS, chargé de recherche au CNRS, pour son aide et sa collaboration enrichissante.
Tous les membres de l'Equipe de recherche en Architecture d'Ordinateurs sont cités, plus loin, pour les remercier ainsi de leurs diverses collaborations pendant la durée de mon projet. Merci à Sandrine, Anne Marie, Isabelle qui ont contribué à la frappe de cette thèse.

Merci à Daniel IGLESIAS, responsable du service de reprographie qui assure un travail d'excellente qualité.

EQUIPE DE RECHERCHE EN ARCHITECTURE D'ORDINATEURS

ALIOUAT	Makhlouf	
AMIELH	Isabelle	(secrétaire de l'équipe)
ANTONINO	Christian	
BAILLE	Gérard	
BASCHIERA	Daniel	
BEKKARA	Noureddine	
BERGER SABBATEL	Gilles	
BOSC	Bernard	(CIME)
BOURCIER	Emile	
CAISSO	Jean-paul	
COURTOIS	Bernard	(Directeur de l'équipe)
DANG	Weidong	
DELORI	Hubert	(CMP)
DREVILLE	Fabienne	
DUPRAT	Jean	
FERNANDES	Antonio	
GUIGUET	Isabelle	
GUYOT	Alain	
HOCHET	Bertrand	
HORNIK	Armand	
JAMIER	Robert	
JERRAYA	Ahmed Amine	
LAURENT	Jacques	
MARCHAL	Pierre	
MARINE	Souheil	
MHAYA	Noureddine	
MICOLLET	Dominique	
NICOLAIDIS	Mihaïl	
PAILLOTIN	Jean-François	(CMP)
ROUGEAUX	François-René	
SOUAI	Mohamed	
VARINOT	Patrice	
ZYSMAN	Eytan	



Il était une fois MS le registre, DECO, MUX et son frère jumeau. Ce jour là, il se dirigeaient vers leur Palais ...

D'aprés les contes de "1001" nuits

AIDE AUX LECTEURS

Le nom donné au circuit décrit dans cette thèse est COBRA, il correspond aux initiales de l'expression, en anglais, qui décrit son vrai rôle et qui est :

COntroller with **B**uilt-in-self-checking for **R**eal-time **A**pplications

qui signifie contrôleur autotestable pour des applications temps réel.

Ce circuit a été défini et conçu, et il est à ce jour en cours de fabrication.

Les trois premiers chapitres qui le décrivent sont totalement indépendants les uns des autres, le quatrième chapitre est basé sur les explications données dans ces trois précédents, pour décrire la conception et la réalisation de COBRA.

Les termes "partie opérative/partie **contrôle**" (les deux composantes principales d'un microprocesseur), sont remplacés ici par les termes "partie opérative/partie **commande**", afin d'éviter toute confusion possible avec les notions de "contrôle" couramment employées dans cette thèse, et qui sont à la base des circuits autotestables. Le terme "partie contrôle" tel qu'il est connu, est en fait la traduction de l'anglais de "control part" qui signifie (d'aprés les dictionnaires "puristes") "partie commande" ...

RESUME

Cette thèse décrit les différentes étapes de la conception d'un microprocesseur spécifique au contrôle des automatismes de sécurité, et plus particulièrement à la surveillance des automatismes embarqués des systèmes de transports en sécurité. Ce microprocesseur est autotestable, i.e., capable de détecter instantanément ses propres erreurs.

La conception d'un tel circuit est basée sur les hypothèses de pannes au niveau analytique, c'est à dire en considérant les mécanismes électriques élémentaires, dans le cadre de la technologie dans laquelle est réalisée le circuit, qui est ici le N-MOS. Les blocs fonctionnels "Strongly Fault Secure" auxquels sont associés des contrôleurs "Strongly Code Disjoint" sont à la base des circuits "Self-Checking", dits autotestables.

Le circuit réalisé ici, nommé COBRA, démontre la faisabilité d'un microprocesseur autotestable. COBRA gère indépendamment 19 signaux différents : date des évènements externes, mesure des fréquences, surveille 14 entrées logiques, et possède 7 sorties indépendantes. Le programme d'application de COBRA est contenu dans une PROM externe de 16 Koctets, adressés par 14 bits multiplexés sur le bus interne de 8 bits. COBRA contient également une liaison série, une RAM de 64 octets, et 3 temporisateurs 14 bits indépendants, ainsi qu'une unité arithmétique et logique de 8 bits. Un jeu de 43 instructions est exécuté par COBRA.

MOTS-CLEFS

Conception VLSI ; circuits autotestables ; outils de C.A.O. ; microprocesseurs ; hypothèses de pannes ; codes ; contrôleurs ; assemblage de cellules ; partie opérative ; partie commande.

ABSTRACT

This thesis deals with the design of a microprocessor dedicated to safety applications and especially to Automatic Train Control. This microprocessor is self-checking, i.e., able to detect its own errors. In the design of this circuit, low-level fault hypotheses for the N-MOS technology are considered, i. e., elementary electrical mechanisms are taken into account. "Self-Checking" circuits are based on "Strongly Fault Secure" functional circuits and "Strongly Code Disjoint" checkers.

The circuit we describe in this thesis, is called COBRA for COntroller with Built-in-self-checking for Real-time Applications. It processes independently 19 different signals : date out external events, measure frequencies, supervise 14 logic inputs, and has 7 independent outputs. COBRA communicates with an external PROM, used as program storage. This PROM is addressed with 14 bits, multiplexed over the 8-bit address/data internal bus of the data path. It contains also one serial I/O, a 64 byte RAM, and 3 independent 14 bits counters. An 8 bit ALU could be used for arithmetical and logical operations. A set of 43 instructions processes all these operations.

KEY-WORDS

VLSI Design ; Self-Checking ; CAD tools ; microprocessors ; faults hypotheses ; codes ; checkers ; cells assembling ; data path ; control part.

TABLE DES MATIERES

CHAPITRE 3 - METHODE ET MOYENS DE CONCEPTION

CHAPITRE 4 - DESCRIPTION DE LA REALISATION DE *COBRA*

INTRODUCTION

INTRODUCTION

Les circuits intégrés logiques poursuivent leur évolution vers la réalisation de fonctions de plus en plus complexes. Chargés de missions telle que la régulation de processus, ils ont également comme fonctions importantes le maintien de la sécurité en cas d'incident survenant lors du déroulement d'un processus. Il est donc fondamental de se donner les moyens de s'assurer à tout moment du bon fonctionnement de ces circuits, ainsi que des systèmes logiques qu'ils composent.

La motivation principale de ce travail est l'étude et la réalisation d'un circuit spécifique pour les systèmes de transports ferroviaires, dans lesquels la sécurité est la première condition de leur existence. Dans ce circuit, tous les moyens sont donnés pour s'assurer à tout moment de son bon fonctionnement. Ces moyens de contrôle, nommés le "test en ligne", consistent en une surveillance du fonctionnement d'un circuit pendant qu'il accomplit sa mission et donc pendant que le système est actif. En cas d'anomalies, une alarme est donnée en activant le système de sorte que la sécurité reste assurée.

Jusqu'à nos jours, la sécurité dans les systèmes critiques (systèmes de transports, automatismes industriels de sécurité, nucléaire) était assurée par des techniques diverses basées, soit sur des composants discrets dits de sécurité, soit sur des techniques de codage des informations dans le cas de systèmes contrôlés par des circuits logiques ou microprocesseurs. Dans le premier cas, il était question de sécurité intrinsèque (fail-safe) qui consistait à polariser les pannes dans un sens restrictif. Exemple : une panne qui provoquait une survitesse d'un train, devait également provoquer la limitation du seuil de vitesse admise, voire l'arrêt du train. Mais l'application de cette technique impliquait la nécessité d'établir une liste exhaustive des comportements d'un composant en cas de panne. Une telle liste n'est possible que dans le cas de composants simples tels que transistors, capacités, résistances, relais, circuits imprimés, etc...

La sécurité intrinsèque consistait également à réduire l'intervalle de risque de façon qu'il soit négligeable par rapport au temps de réaction du système : une panne qui reste dormante pendant un temps très court, durant lequel le train parcourt une distance négligeable, est considérée comme de sécurité intrinsèque.

Mais les automatismes de trains, comme tous les automatismes en général, devenant de plus en plus complexes, la puissance de traitement des techniques basées sur la sécurité intrinsèque, est vite limitée. Il est maintenant indispensable d'avoir recours aux microprocesseurs pour assurer la continuité dans ce domaine.

Le premier chapitre de cette thèse analyse les automatismes de transports en mettant l'accent dans sa première partie, sur les techniques à base de microprocesseurs commerciaux utilisant les redondances et le codage des informations. La littérature spécialisée dans les systèmes de transports de sécurité montre que ces techniques ne présentent pas toujours toutes les conditions autorisant leur emploi. Dans certains cas la démonstration de sécurité n'est pas satisfaisante, dans d'autres cas la fonctionnalité est alourdie par ces techniques à un point tel que le système est handicapé. La deuxième partie de ce chapitre décrit un exemple sur lequel l'étude de fonctionnalité du circuit faisant l'objet de cette thèse est partiellement basée. C'est le système connu sous le nom de VAL, du métro de Lille. La description de la chaîne de surveillance de ce système donne une idée précise du cahier des charges d'un système de transport en général. Ce fut donc un des points de départ qui ont permis d'engager la présente étude. D'autres systèmes de transports ont été étudiés, afin de définir un cahier des charges fonctionnelles plus général, aboutissant à un circuit capable de gérer la surveillance de systèmes importants et divers.

Le deuxième chapitre traite des techniques d'autotest utilisées dans la conception du circuit COBRA. Pour être plus précis, il aurait fallu utiliser l'expression "self-checking" qui signifie auto-contrôle. La plupart des références, écrites en langue française, dans ce domaine emploie cette expression. La notion de "circuits autotestables" est une notion plus large que la notion de circuits "self-checking". Les autres expressions anglo-saxonnes, décrivant des notions particulières des circuits autotestables, trouvent leurs définitions dans ce chapitre.

Le test des circuits intégrés logiques est un problème très vaste et qui évolue très vite, grâce au grand nombre d'études qui y sont consacrées. Deux types de test peuvent être distingués : le test hors-ligne et le test en-ligne.

Le test hors-ligne s'applique en dehors de l'application du circuit à tester, et peut être envisagé pour toutes les phases de la vie d'un circuit :

- conception (validation des spécifications, vérification des masques, etc...)
- fabrication (tri des "puces" et contrôle sous pointes des tranches de silicium, vérification des soudures, test de fin de fabrication, etc...)
- utilisation (test d'entrée, maintenance, etc...)

Deux grandes familles constituent le test hors-ligne :

- Le test paramétrique : c'est une mesure des différents paramètres qui caractérisent le circuit testé : paramètres statiques (valeurs des tensions de seuil, courants, impédances, etc...) et les paramètres dynamiques (temps de commutations, délais, etc...).

-Le test logique (ou fonctionnel): vérifie si le circuit accomplit correctement la fonction logique pour laquelle il a été conçu.

Le test en-ligne s'applique pendant le fonctionnement du circuit. Deux types de test en-ligne peuvent être distingués :

- Le test en-ligne continu, qui n'altère pas la fonctionnalité du circuit testé mais qui est toujours disponible.

- Le test en-ligne discontinu, qui consiste à élaborer des procédures de test spécifiques que le circuit exécute quand il n'a aucune tâche fonctionnelle à exécuter.

La technique utilisée dans la conception du circuit de cette étude est le test en-ligne continu, qui n'exige donc pas d'interruption périodique du programme d'application pour le lancement d'un programme de test (cas du test en-ligne discontinu).

Les explications données dans ce chapitre situent les pannes dans les circuits intégrés de la technologie MOS canal N, dans laquelle le circuit COBRA sera réalisé. Ensuite, les hypothèses de pannes de cette technologie sont classifiées pour définir, en fonction de ces hypothèses, la classe qui est retenue pour la conception de circuits autotestables. Dans la seconde partie de ce chapitre, quelques définitions situent sommairement la théorie de base des circuits "self-checking". Ensuite, les codages internes utilisés pour ces circuits, sont présentés, avec les contrôleurs qui leurs sont associés (code de parité, code de berger et le code double-rail). Enfin, les règles de conception de circuits SFS (Strongly Fault Secure) sont présentées.

Le troisième chapitre présente la méthode de conception adoptée pour la réalisation du circuit COBRA. Cette méthode part des spécifications initiales, qui sont les besoins de l'utilisateur, et aboutit progressivement à une architecture adaptée qui permet la réalisation physique du circuit. Cette méthode de conception progressive s'appelle la démarche descendante. Elle est particulièrement adaptée à l'évolution technologique et aux circuits V.L.S.I., en raison des affinements successifs des spécifications du système à concevoir, à partir des spécifications initiales relatives au cahier des charges. Cette démarche permet de décomposer le processus de conception selon plusieurs niveaux. Ainsi le choix de chemin des données, ou la description d'une partie opérative est distingué du choix de la structure de contrôle, ou la description des séquencements gérés par la partie commande.

Cette méthode de conception appliquée à ce circuit est étroitement liée à un certain nombre d'outils d'aide à la conception. Et enfin, est présentée la méthode utilisée pour réaliser le circuit, en tant que circuit autotestable, et donc différent des circuits classiques que réaliseraient ces outils. Une nouvelle stratégie est nécessaire pour adapter l'emploi de ces outils à la structure particulière d'un circuit autotestable. Une proposition est formulée à la fin de ce chapitre pour adapter certains outils à la réalisation des circuits autotestables à l'aide de la C.A.O.

Le quatrième chapitre est consacré à la réalisation matérielle de COBRA. Tous les blocs fonctionnels qui composent ce circuit y sont présentés, ainsi que la façon de les utiliser lorsqu'ils sont accessibles par des instructions, et également les structures de contrôle qui leurs sont associées. Ce quatrième chapitre s'appuie sur les trois premiers : le premier chapitre présente le cahier des charges fonctionnelles, le deuxième présente le cahier des charges du contrôle self-checking et le trosième résume les moyens et la méthode de conception du circuit. Enfin, le dessin des masques destiné à la fabrication du circuit, est montré à la fin de ce chapitre. Ce dessin est le résultat de l'assemblage des dessins de tous les blocs fonctionnels munis de leurs structures de contrôle "self-checking". Lors de cet assemblage, tous les problèmes de topologie et de conformité aux règles de dessin N-MOS et aux règles de conception de cirucits autotestables, ont été considérés.

CHAPITRE 1

AUTOMATISMES DE TRANSPORTS

1- INTRODUCTION

L'objectif de ce chapitre est de présenter l'environnement de l'étude du circuit COBRA tel qu'il s'est présenté au début de l'étude, c'est à dire, la définition en partie du rôle de COBRA dans un automatisme de transport en le situant à la place d'un équipement déjà existant.L'exemple du VAL (métro de Lille) décrit plus loin dans ce chapitre, illustre le fonctionnement d'un automatisme de transport dans lequel COBRA pourrait jouer le rôle important de surveillance de sécurité de l'ensemble de l'automatisme.

Le cahier des charges de la présente étude est décrit globalement dans ce même chapitre.

2- POSITION DU PROBLEME

Cette étude se place dans le contexte d'une évolution du concept de sécurité dans les automatismes de transport terrestre (métro, train, ...). La sécurité des voyageurs est la condition sine qua non de l'existence de ces moyens de transports. Cette évolution se caractérise par un glissement du concept de sécurité intrinséque vers le concept de sécurité probabiliste, qui est celui des circuits intégrés VLSI. Cette évolution est justifiée par deux raisons :

* Les progrés réalisés dans le domaine technologique des composants semi-conducteurs : SSI -> MSI -> LSI -> VLSI. La limite supérieure de la sécurité intrinsèque se situe au niveau du MSI. Par ailleurs, les équipements des automatismes faits avec cette technique n'ont qu'une puissance limitée de traitement.

* Les performances de plus en plus grandes, demandées aux moyens de transports en site propre : vitesse élevée, intervalles entre trains réduits, souplesse d'exploitation, etc. Ces performances ont pour corollaire une puissance de traitement (rapidité, complexité), telle que seules les techniques nouvelles puissent assurer, sans compter le bon niveau de fiabilité de ces techniques, ce qui assure une bonne exploitabilité des systèmes qu'elles réalisent.

2.1 Architecture générale d'un pilote automatique

Un pilote automatique de métro ou de train est un ensemble d'équipements au sol et d'équipements embarqués. Il se compose en général de deux chaînes (figure 1). Ces deux chaînes sont à la base de toute structure d'un automatisme de pilotage automatique. La sécurité est assurée par la chaîne de surveillance qui doit vérifier constamment le bon fonctionnement de la chaîne de commande et s'assurer de l'exécution de ses ordres.

Ceci illustre donc l'importance du rôle de la chaîne de surveillance dans un tel système de sécurité. Les différents constructeurs améliorent régulièrement les systèmes qu'ils proposent en diversifiant les études et les techniques, dans le but d'atteindre le niveau de sécurité maximale.

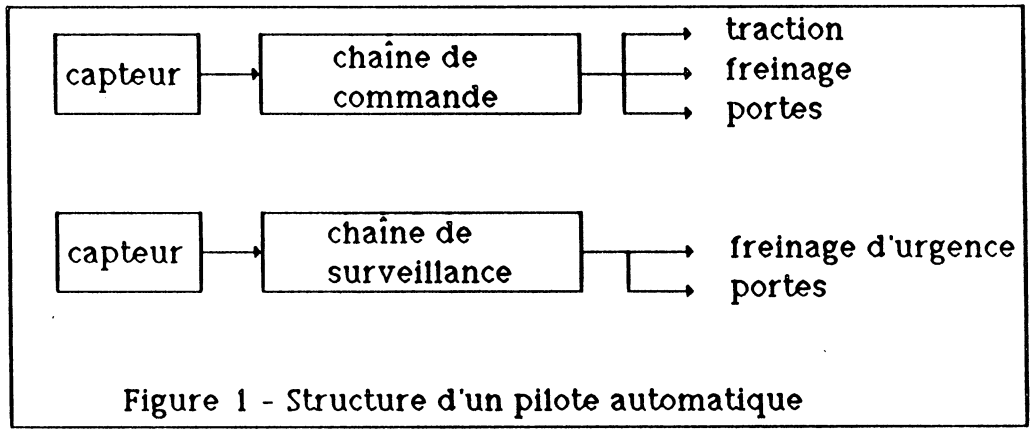


Figure 1 - Structure d'un pilote automatique

Dans ce qui suit, on ne s'intéresse qu'à la chaîne de surveillance.

2.2 Techniques à l'étude

Compte tenu de l'utilisation de plus en plus répandue des microprocesseurs dans les automatismes ferroviaires de sécurité, plusieurs études ont été développées et ont donné naissance aux filières techniques étudiées dans les systèmes ferroviaires actuels. Toutefois, ce chapitre traite exclusivement de l'utilisation des microprocesseurs dans les automatismes remplissant des fonctions de sécurité dans les transports terrestres. Ceci ne concerne donc pas les circuits logiques cablés utilisés encore dans quelques systèmes.

Trois filières peuvent être distinguées dans l'emploi des microprocesseurs dans les automatismes de sécurité :

* Filière de la boîte noire : un microprocesseur du commerce est utilisé ici, indépendamment de sa structure interne. La sécurité est assurée par les techniques de redondances matérielles ou de redondance par codage.

* Filière de la boîte grise : un microprocesseur spécifique dans lequel la panne matérielle n'est pas considérée. Par contre, les moyens de commandabilité et d'observabilité nécessaires, y sont donnés.

* Filière de la boîte blanche : un microprocesseur spécifique dans lequel la panne matérielle est prise en compte dés le départ de la conception du circuit : DFT (Design For Testability). C'est cette Filière qui fait l'objet de la présente thèse.

2.2.1 Utilisation de la redondance (boîte noire)

Il n'est pas possible de satisfaire aux exigences de la sécurité ferrovaire, avec un système à microprocesseur unique du commerce, puisqu'un microprocesseur du commerce a été par définition conçu pour être utilisé dans les applications les plus générales et les plus diverses. Or pour pouvoir utiliser un microprocesseur du commerce dans les équipements de sécurité, plusieurs techniques sont possibles:

- redondance au niveau du matériel : on utilise alors deux systèmes avec un comparateur, ou plusieurs systèmes avec un circuit de logique majoritaire. On parle alors de "redondance spatiale".

- redondance au niveau du logiciel : on peut alors utiliser deux logiciels différents dans deux systèmes différents, ou bien deux logiciels qui fonctionnent alternativement dans un système unique. Cette dernière technique plus utilisée que la première est généralement dénommée "redondance temporelle".

- la redondance par codage mathématique: le traitement est fait suivant une loi mathématique. Par conséquent, la démonstration de sécurité est purement mathématique.La redondance est aussi fréquemment utilisée dans les transmissions de messages entre l'équipement au sol et l'équipement embarqué, qui doivent être aussi sures que le reste du système : la répétition des messages ou des mots permet d'accroître la sécurité de la transmission.

2.2.2.1 Structures générales utilisées

Structure Monoprocesseur

On utilise alors une redondance temporelle, c'est à dire que le système basé sur un microprocesseur unique, contient 2 programmes différents qui remplissent alternativement la même fonction et dont les résultats sont comparés : c'est la redondance logicielle.

Par ailleurs, certaines techniques ont recours au codage de l'information issue des sorties sans se soucier des pannes possibles, comme le processeur codé ou monoprocesseur [VAL 84], où il s'agit de coder les logiciels et vérifier les déroulements des algorithmes et des séquencements, et de coder les opérandes et les données avec un code de la forme :

$$\underset{\text{donnée codée}}{X} = \underset{\text{donnée non codée}}{2^k x} + \underset{\text{code de contrôle}}{C}$$

Un tel codage mathématique est indépendant du type du microprocesseur, mais le temps de codage augmente rapidement avec le niveau de sécurité requis.L'aspect "temps réel" pourrait être critique dans certaines applications. Par ailleurs, un tel codage est non tolérant aux fautes, ce qui peut être dans certains cas contraire à la sécurité.

Structure à deux chaînes avec comparateur (Système 2 de 2)

On utilise deux processeurs distincts. Les commandes en sortie sont validées si les deux systèmes donnent des résultats identiques. Dans le cas contraire l'équipement met ses sorties dans un état de sécurité et s'arrête. Il s'agit donc à priori d'un système non tolérant aux fautes. Le comparateur peut ne tester que les sorties, ou bien également des variables internes des deux systèmes. Cette deuxième solution a l'avantage de permettre la détection des erreurs au plus tôt. Elle est utilisée dans des sytèmes où tous les bits de données et d'adresses sont testés par le comparateur :

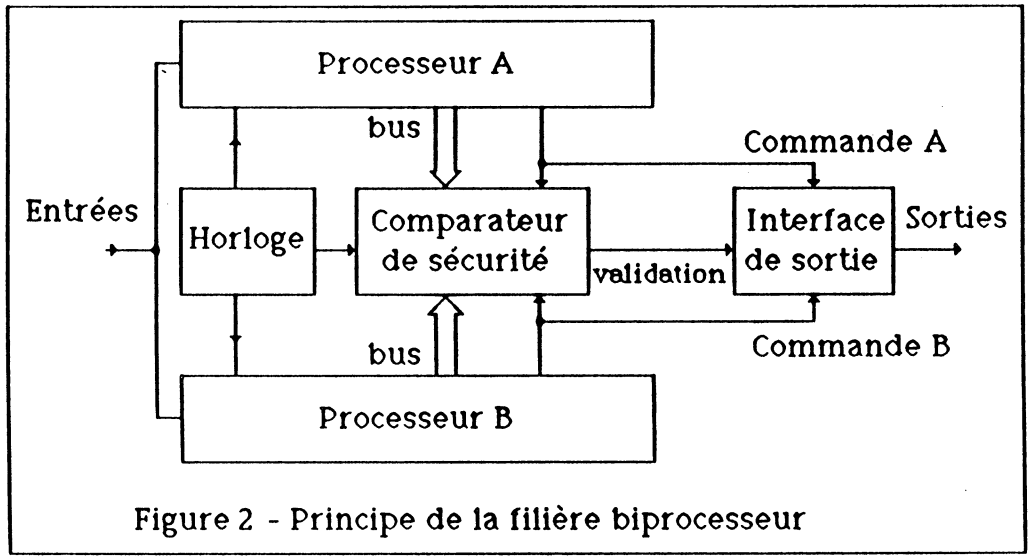


Figure 2 - Principe de la filière biprocesseur

Il est fréquent que la commande de sortie soit fournie par un seul processeur : l'interface de sortie est alors alimenté par le comparateur .

Structure à 3 chaînes avec logique majoritaire (Système 2 de 3)

On utilise trois processeurs distincts. Les commandes en sortie sont validées si deux systèmes au moins donnent des résultats identiques. Les sorties sont donc reliées à un circuit de vote majoritaire.

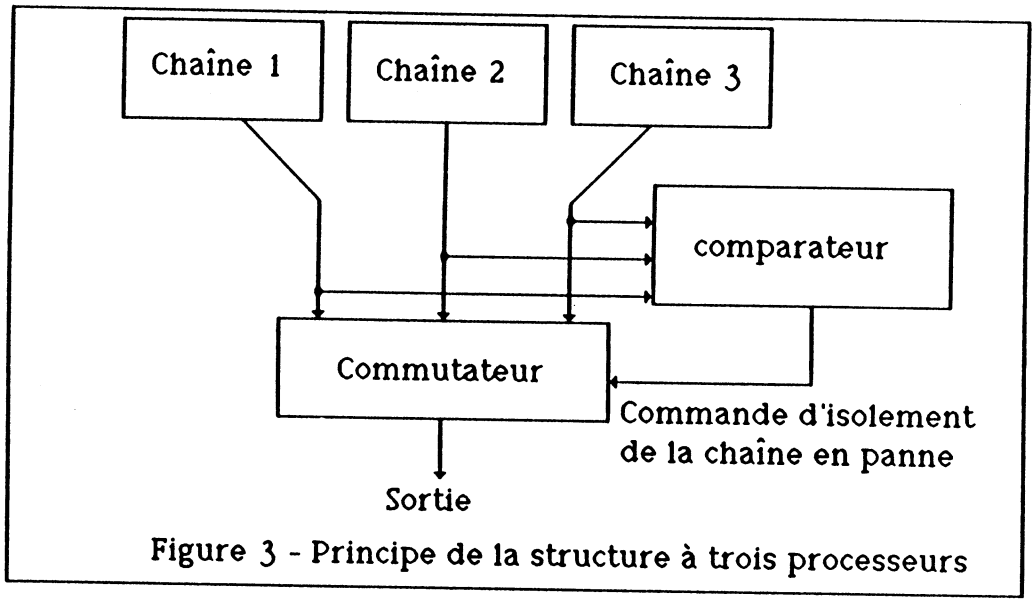


Figure 3 - Principe de la structure à trois processeurs

Cette structure est généralement reconfigurable ce qui signifie que lorsqu'un système est en désaccord avec les deux autres, il est mis hors circuit et les deux autres continuent à fonctionner dans une structure 2 de 2. Pour cette raison on considère que la strucuture 2 de 3 est plus fiable que la structure 2 de 2 puisqu'elle tolère une faute. Lorsque l'une des trois chaînes commet des erreurs, il importe donc que :

- l'on puisse la détecter et l'isoler pour reconfigurer le reste automatiquement
- une alarme soit donnée
- l'on puisse réparer la chaîne en panne alors que les deux autres sont en fonctionnement (par simple échange de cartes ou modules)
- et réinsérer cette chaîne dans l'ensemble pour revenir à la structure 2 de 3 normale.

Une séquence de réinsertion de chaîne qui permet de revenir à la triple redondance comprend les étapes suivantes :

- Enclenchement et initialisation de la chaîne réparée
- Synchronisation du programme de cette chaîne
- Enseignement, c'est à dire transfert du contenu de la Mémoire RAM d'une chaîne en fonctionnement avec vérification de cette copie par comparaison avec mémoire de la troisième chaîne
- Reconfiguration en structure 2 de 3 de l'ensemble

2.2.1.2 Applications dans les systèmes et problèmes rencontrés

Pour atteindre l'objectif de sécurité voulue, chaque constructeur a étudié le problème des systèmes à base de microprocesseurs, en le divisant en deux parties principales [IRT 82] :

- la sécurité par le matériel

- la sécurité par le logiciel

Les trois structures définies ci-dessus sont principalement appliquées dans l'équipement de sécurité.

Les problèmes rencontrés sont souvent dûs aux codages des logiciels : on applique alors les règles de l'art dans la réalisation des logiciels :

- simplifier les programmes et utiliser le langage Assembleur car il est le plus proche du code machine.
- écrire des programmes modulaires et structurés, en évitant les boucles.
- utiliser le moins possible les interruptions, en les remplaçant par des scrutations cycliques des entrées.
- choisir la structure du logiciel (en fonction du matériel choisi) : soit deux logiciels identiques se déroulant alternativement en les comparant, soit deux logiciels différents conçu par des équipes différentes.

Mais le problème de test du matériel par logiciel demeure difficile. Ainsi, certains projets en cours, utilisant les filières matèrielles citées ci-dessus, (microprocesseurs standards du commerce) se trouvent limités par les restrictions posées, tout en ayant un système global complexe (soit par le matériel soit par le logiciel). En effet, le problème se trouve dans la structure même du microprocesseur qui est inconnue de l'utilisateur, et il est donc incapable de prévoir ou de s'affranchir des pannes possibles.

D'autres techniques utilisant la filière biprocesseur sont en cours d'étude [GAZ 84], mais en plus des inconvénients de la méthode du monoprocesseur (non tolérance aux fautes, et aspect "temps réel" critique), cette méthode amène des problèmes du type : deux pannes différentes et simultanées arrivent dans les deux circuits, d'où une panne résultante indétectable : l'efficacité des test utilisés est difficilement démontrable, et la démonstration de sécurité n'est pas rigoureuse.

2.2.2 - Filière boîte grise

C'est généralement le cas de microprocesseurs spécifiques, mais qui portent le nom de "microprocesseurs testables", i.e., les tests ont lieu pendant les intervalles d'oisiveté, par des entrées et sorties convenables. Les procédés généralement appliqués sont le compactage des informations. Toutefois, aucune démonstration de sécurité n'a été donnée (une telle démonstration conduirait en fait à la filière boîte blanche mais avec de mauvaises caractéristiques par rapport à cette filière).

2.2.3 - Conclusions sur les filières précédentes

Ces conclusions proviennent des rapports établis par les utilisateurs des systèmes de transport de sécurité actuels :

* Il n'est jamais sûr que les tests soient exhaustifs dans les structures à redondance matérielle.

* Le temps consacré aux tests ou au codage risque d'être prohibitif pour certains systèmes de transports et plus particulièrement les systèmes à venir, qui ne supporteront plus de telles contraintes.

* Dans tous les cas, la sécurité n'est pas transparente à l'utilisateur.

D'où le projet de l'étude d'une filière "boîte blanche".

2.2.4 - Une autre filière (boîte blanche)

L'objet de la présente étude est le développement d'une voie différente des filières précédentes. Cette voie consiste à réaliser la chaîne de surveillance à l'aide d'un circuit spécifique à test intégré dans le matériel. Une généralité sera recherchée, en ce sens que les entrées d'un tel circuit soient banalisées de façon à ce que le circuit soit versatile.

Les contraintes de test intégré sont des contraintes de détection d'occurrence de pannes au cours du fonctionnement du circuit : c'est le test en-ligne intégré décrit dans le chapitre 2.

Les caractéristiques "utilisateur" de cette filière sont :

- la sécurité est transparente à l'utilisateur, la démonstration de sécurité étant faite à la conception.
- l'aspect non-tolérant aux fautes est identique aux autres filières si le circuit est utilisé seul.

L'exemple suivant du Métro de Lille, le VAL, est celui d'un des systèmes ayant servi pour référence afin de développer et faire démarrer cette étude.

3- EXEMPLE DU METRO DE LILLE (LE VAL)

3.1 Les automatismes du VAL

Les automatismes du VAL se composent essentiellement :

- d'équipements fixes qui détectent la présence des trains et renvoient des informations vers les trains
- d'un ensemble de boucles conductrices contenues dans un "tapis" situé entre les pistes de roulement et qui permet les échanges sol-train et inversement
- d'un équipement embarqué qui se compose (voir figure 5) :

  * d'un Dispositif de Conduite Automatique (DCA) (sans système de sécurité), qui réalise l'asservissement de vitesse du train
  * de deux Chaînes de Sécurité qui commandent le Freinage d'Urgence (FU), et autorisent l'ouverture des portes. Les sorties de commande de FU sont normalement mises en "ET" et peuvent être mises en "OU" en cas de panne de l'une des deux chaînes.

La ligne est découpée en cantons fixes. Lorsqu'un canton est libre, le canton précédent émet le Mode Normal MN. S'il est occupé, le Mode Perturbé MP est émis sur le canton précédent et le train suit une courbe de ralentissement pour s'arrêter avant la fin de ce canton (voir figure 6).

3.2 Informations reçues par le train

• Le train ne peut avancer que s'il reçoit la Fréquence de Sécurité FS. Cette porteuse est modulée en fréquence par deux types d'informations.

- le sens de la marche : avant MAV ou arrière MAR
- le mode : Normal MN, perturbé MP ou accostage MAC.

• La boucle FS est munie de croisements qui sont détectés par l'antenne du train (figure 4). En Mode Normal le DCA asservit la vitesse du train de sorte que l'intervalle de temps entre 2 croisements soit égal à 0,3 seconde.

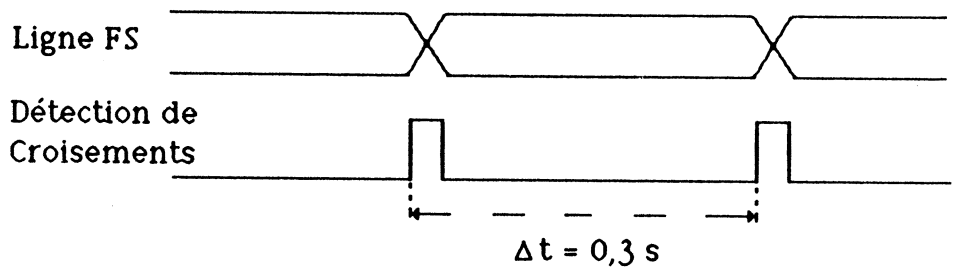


Figure 4 - Signal reçu par le train en mode normal

En Mode Perturbé le DCA prend en compte les croisements du Programme Perturbé qui émet une fréquence différente de la FS.

• Lorsqu'un train ne peut plus avancer par ses propres moyens il est accosté puis poussé par le train précédent. On utilise pour cela le Mode Accostage MAC qui autorise la progression du train accosteur à faible vitesse (0,8 m/s) et sa pénétration sur le canton aval occupé.

3.3 Constitution de la chaine de sécurité embarquée

Elle se compose :

- d'une partie analogique qui capte les information sécuritaires du sol, les décode et les met en forme
- d'une partie logique qui traite ces informations et fournit la commande de FU et la commande des portes. C'est cette partie logique qui pourrait être réalisée au moyen d'un microprocesseur spécialisé avec le "Test intégré".

Les fonctions remplies par la partie logique sont les suivantes :

- Sécurité survitesse FS (Mode Normal)
- Sécurité survitesse PP (Mode Perturbé)
- Sécurité survitesse Accostage (Mode Accostage)
- Sécurité dépassement d'un point d'arrêt
- Sécurité sens de Marche

Toutes ces sécurités déclenchent le Freinage d'Urgence FU.
De plus la chaîne de sécurité autorise l'ouverture des portes à droite ou à gauche.

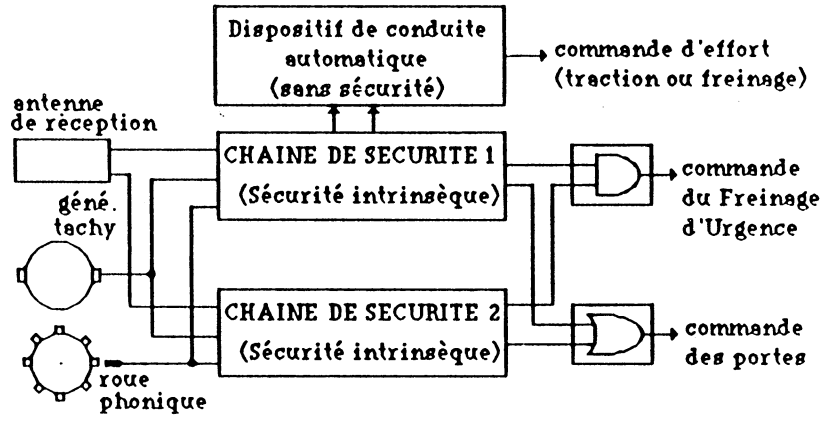
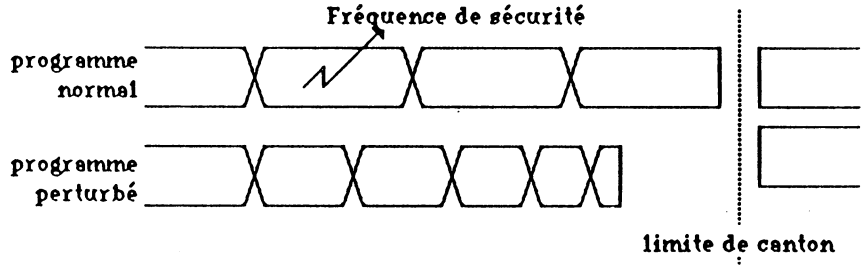


Figure 5 - Synoptique des automatismes embarqués



Le DCA asservit la vitesse du train de sorte que l'intervalle de temps Δt entre deux croisements soit égal à 0,3 s.
Si le canton aval est libre, le train reçoit le mode normal MN et suit le programme normal.
Si le canton aval est occupé, le train reçoit le mode perturbé MP et suit la courbe de freinage et d'arrêt du programme perturbé.

Figure 6 - Principe de l'anticollision

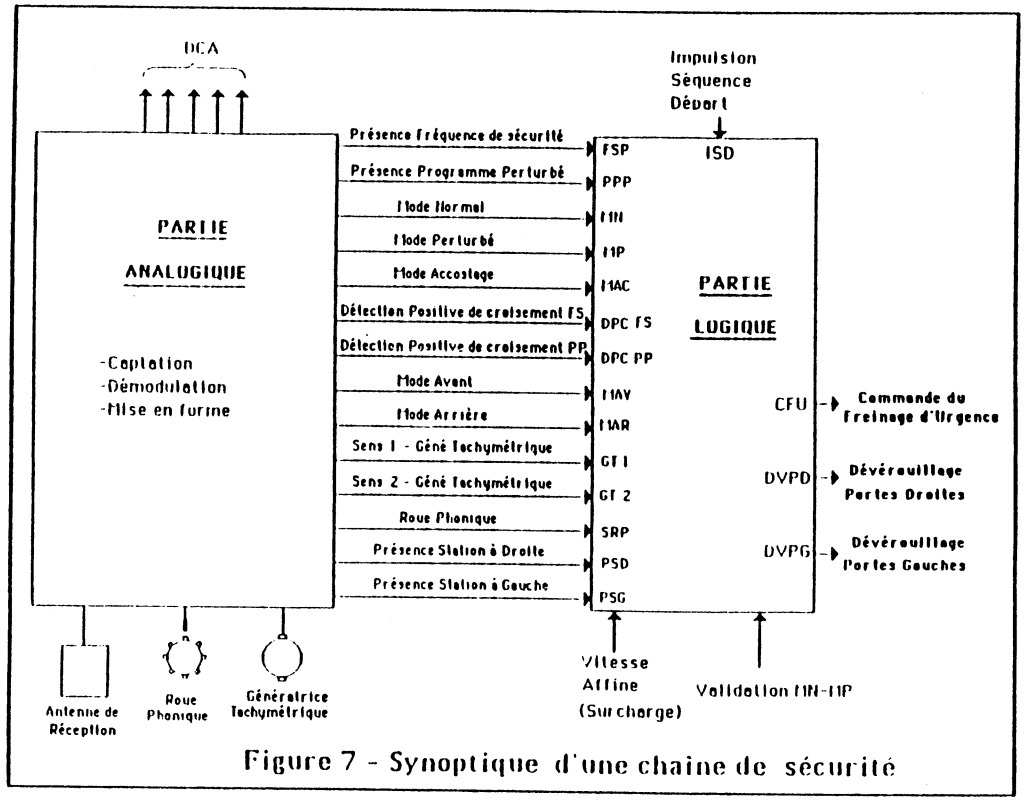


Figure 7 - Synoptique d'une chaine de sécurité

3.4 Description des fonctions remplies par la partie logique

* Sécurité Survitesse FS

La partie analogique détecte les croisements de la ligne FS et élabore à chaque croisement une impulsion DPC-FS (Détection Positive de Croisement). Tant que le train n'est pas en survitesse, l'intervalle de temps entre 2 DPC est supérieur à 0,27 seconde. Dans le cas contraire le train est en survitesse ce qui déclenche le Freinage d'Urgence irréversible jusqu'à l'arrêt.

* Sécurité Survitesse Accostage

En mode Accostage la vitesse est limitée à 1,2 m/s ce qui correspond à une fréquence maximale de roue phonique de 310 Hz.

(La roue phonique est un organe électromécanique qui fournit un signal dont la fréquence est proportionnelle à la vitesse du train.

* Sécurité Dépassement d'un point d'arrêt

Lorsque le canton aval est occupé le train ne doit pas dépasser l'extrêmité du Programme Perturbé.

* Sécurité Sens de Marche SSM

Le sens de Marche est contrôlé à partir du moment où l'information "Vitesse Nulle" VNT tombe à 0, c'est à dire au dessus de 0,5 m/s qui correspond à une fréquence de roue phonique de 120 Hz. Cette fonction est réalisée en comparant le sens de marche donné par une Génératrice Tachymétrique (qui informe sur le sens effectif) avec le Mode Avant ou Arrière transmis par les équipements au sol : SSM = MAV. Sens 1 + MAR. Sens 2

* Autorisation de dévérouillage des Portes (Droite ou Gauche)

Cette autorisation n'est délivrée que si :

- la vitesse du train est inférieure à 0,5 m/s c'est à dire que VNT = 1 ce qui correspond à une fréquence de roue phonique de 120 Hz
- le train reçoit la porteuse de présence de la station
  . à droite soit PSD
  . à gauche soit PSG

* Cas de surcharge du train

Lorsque le train est en surcharge, la vitesse maximale autorisée est réduite de 10%. Le temps minimal entre impulsions DPC est donc de 0,3 s au lieu de 0,27 s.

* Synthèse des sécurités et commande du FU

Le Freinage d'Urgence n'est levé que si toutes les sécurités sont à 1. Cette synthèse est réalisée au moyen d'une fonction combinatoire comprenant une trentaine de portes logiques (ET, OU, NON).

A chaque arrêt, lorsque la vitesse tombe en dessous du seuil de la SSM, le FU doit se déclencher et être mémorisé. Il ne peut être réarmé que si l'on reçoit l'Impulsion SD à vitesse nulle (v< 0,5 m/s). Cette fonction séquentielle est réalisée au moyen d'une bascule.

3.5 Types de fonctions nécessaires

En résumé les fonctions nécessaires au fonctionnement de la Chaîne de Sécurité du VAL sont les suivantes :

- fonctions combinatoires nombreuses
- 1 fonction séquentielle (la bascule de sortie)
- 2 mesures de fréquence (comparaison à des seuils :120 et 310 Hz)
- 2 mesures d'intervalles de temps entre impulsions :0,27 et 0,3 s.
- 2 temporisations 1 et 5 secondes

Le système comporte 17 entrées et 4 sorties, toutes comprises entre 0 et 5V. Toutes les entrées varient lentement (f<4 Hz) sauf la roue phonique dont la fréquence peut atteindre 6 k Hz.

Un autre système de transport a été étudié dans le but de réaliser un circuit qui puisse au moins jouer le rôle des chaînes de sécurité des deux systèmes étudiés. Cet automatisme de transport est le métro de Lyon, d'autres systèmes ont été analysés dans le but de couvrir une plus large gamme, mais qui n'interviennent que pour certains détails. La figure 8 représente le cahier des charges du circuit COBRA.

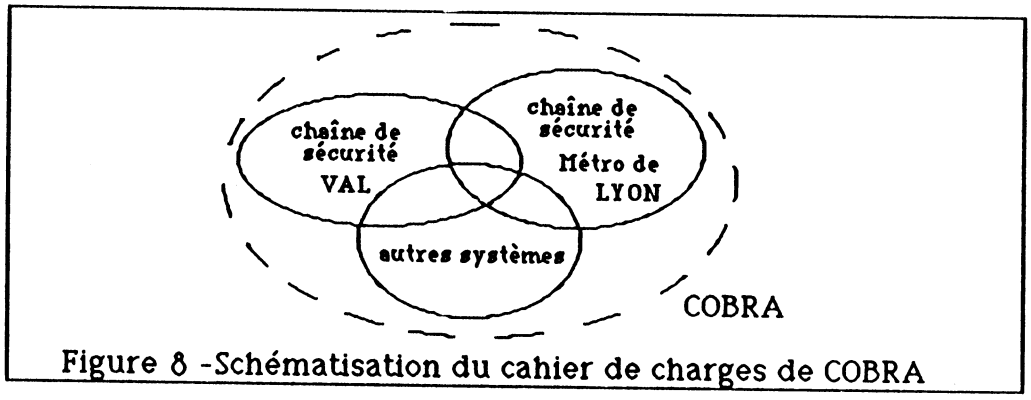


Figure 8 -Schématisation du cahier de charges de COBRA

On se propose de réaliser une structure souple de COBRA afin de pouvoir couvrir l'ensemble des possibilités fonctionnelles que possède chacun de ces automatismes sensiblement différents les uns des autres. C'est la raison pour laquelle nous avons choisi un circuit programmable de type microprocesseur.

L'étude des documents concernant ces automatismes , a permis de faire une synthèse visant à déterminer les spécifications externes d'un circuit programmable qui pourrait remplacer la chaîne de surveillance de chacun de ces automatismes. Les spécifications externes d'un circuit se résument à la spécification du nombre de broches d'Entrées/Sorties, et plus particulièrement à leurs types. En effet, on remarque lors de la synthèse des systèmes existants, que l'on peut classer les signaux parcourant les différents circuits selon deux types et en fonction de leurs fréquences : *signaux lents*, dont la partie information est le niveau logique "0" ou "1", et les *signaux rapides*, qui sont plus souvent associés à la notion de vitesse.

Nous allons donc définir pour les chapitres suivants, deux types de signaux que nous allons associer à certaines Entrées/Sorties des circuits. Ces deux types sont :

NIVEAU : associé aux signaux lents. On note parmi ces signaux les informations provenant de certains capteurs, comme les signaux "Mode Avant" ou "Mode Arrière", dans le cas du VAL ou encore "Présence station à droite" ou "Présence station à gauche".

HORLOGE : ce type d'E/S (Entrées/Sorties) est associé aux signaux rapides, ou plutôt à des informations de fréquence. La roue phonique permet de générer une information de type HORLOGE au contrôle de la vitesse afin de surveiller la vitesse maximale autorisée, dont dépend l'action du FU (Freinage d'Urgence). La définition de ces deux types d'E/S nous permet de prévoir un traitement logique de chaque information relativement à son type.

4 - CONCLUSION

L'analyse de différents systèmes ferroviaires européens [IRT 82] montre que les différents avis convergent vers les quelques méthodes à base de microprocesseurs citées dans ce chapitre en se regroupant toutes autour de quelques idées principales sans pour autant pouvoir éviter les compromis complexité/sécurité. La totalité des articles analysés dans [IRT 82] traite les problèmes en essayant d'améliorer l'aspect logiciel du fonctionnement car ne pouvant intervenir au niveau matériel que de loin (redondance matérielle). Le codage des informations en considérant le microprocesseur comme étant une boîte noire est la seule méthode en cours de développement en France pour les équipements ferroviaires.

La solution COBRA diffère des méthodes citées dans ce chapitre puisqu'elle est basée sur le test au niveau matériel principalement, et au niveau du transistor même (voir chapitre 2). Toutefois, les précautions prises au niveau du logiciel dans ces méthodes peuvent aussi être prises dans le cadre de l'utilisation de COBRA, ainsi que les structures "2 parmi 3" ou "le biprocesseur" avec COBRA, qui peuvent dans certains cas améliorer la sécurité de l'automatisme et la qualité de la tolérance aux fautes.

Du point de vue pratique, l'aspect extérieur de COBRA et les rôles des principales parties qui le composent, proviennent principalement des analyses des systèmes des métros de Lille et Lyon. On trouvera dans le chapitre 4 les exemples d'aspects externes qui ont conduit aux spécifications extérieures de COBRA. Les spécifications internes de COBRA, trés inspirées du point de vue fonctionnel des systèmes de transports actuels, en font un circuit spécifique aux systèmes de transports de sécurité, d'un côté de par sa fonctionnalité spécifique qui garantit la sécurité de fonctionnement du processus qu'il surveille : tout organe externe est surveillé par COBRA (programme d'application, E/S spécifiques). De l'autre côté, le test intégré dans COBRA garantit la détection de toute panne interne à sa structure, et par extension des pannes externes au circuit.

CHAPITRE 2

TECHNIQUES D'AUTOTEST

1-INTRODUCTION

Le but de ce chapitre est de présenter les techniques de test utilisées dans la conception du circuit COBRA.

Quelques généralités permettront de situer ces techniques selon leur historique. Nous présenterons ensuite une classification des pannes définies par B. COURTOIS [COU 81], à partir des mécanismes de pannes de la technologie utilisée, qui est pour COBRA le MOS-Canal N ou NMOS.

Ensuite, nous donnerons quelques définitions et règles de conception de circuits intégrés autotestables. Ces définitions et règles sont reprises principalement de [NIC 84] et [CRO 78].

Nous terminerons ce chapitre en présentant les circuits de contrôle nécessaires à l'observabilité des circuits autotestables.

Notons, que la méthode de test présentée dans ce chapitre est une méthode de test analytique de circuits, qui se reporte à la structure matérielle des circuits. Le test fonctionnel (cas des structures matérielles inconnues) n'est pas abordé dans ce chapitre. Toutefois, le test fonctionnel n'est pas absent de l'étude de COBRA, car il est utilisé pour le test de la mémoire morte externe utilisée comme support des programmes qu'exécutera le circuit COBRA.

2 - GENERALITES

2.1 Bref historique-collage

Une des motivations principales de la définition et de la création des techniques de test et de contrôle des circuits logiques, est la prévention des erreurs dans ces circuits.

L'évolution rapide des circuits intégrés ces dernières années, grâce aux progrès effectués dans le domaine de la technologie, a fait de sorte que les techniques de prévention utilisées sont devenues insuffisantes pour couvrir l'ensemble des pannes possibles dans les circuits intégrés logiques.

En effet, on avait cherché depuis longtemps à modéliser les pannes dans les circuits logiques afin de pouvoir aborder le problème de surveillance de leur comportement, sans avoir à établir d'avance la liste des pannes possibles dans chaque circuit.

Ainsi aux débuts des années 70, des études de modèlisation de pannes dans les circuits logiques ont été développées. Friedman a étudié dans [FRI 71] les modèles de collage des entrées et des sorties des éléments logiques de base de la technologie bipolaire.

Ces modèles de collage logique se sont avérés insuffisants pour représenter les défauts réels pouvant survenir dans ces circuits logiques. Le niveau d'étude de ces modèles de collage était celui des portes logiques. Des recherches ont alors été dirigées vers un niveau d'analyse plus profond : l'application de "vecteurs de test", ou le modèle de collage ouvert/fermé (stuck-open, stuck-on) pour les transistors MOS, [GAL 80].

Dans l'étude actuelle, ce sont les techniques de tests analytiques intégrés au niveau du transistor qui sont rappelées, et qui sont utilisées pour la conception du circuit COBRA.

2.2 Mécanismes de pannes

Le fonctionnement anormal d'un système lors d'une panne a été exprimé dans [ANC 76] par le terme de "catastrophe algorithmique". F. ANCEAU a ensuite proposé l'implantation d'un chien de garde dans ce système, pour détecter l'apparition du désordre résultant, ainsi qu'un système de détection de codes opération illégaux (cas d'Unité Centrale).

Ces suggestions ont été développées et étudiées dans [COU 79], pour contribuer à une étude plus large donnant naissance à un classement d'hypothèses de pannes [COU 81], détaillé plus loin. En effet, pour concevoir des circuits autotestables dans une certaine technologie, on doit connaître et maîtriser cette technologie, et aussi établir un concept solide d'une théorie de circuits autotestables.

Pour cela B. COURTOIS s'est basé sur un niveau bas d'analyse qui est le niveau du transistor, et en adaptant certaines études de "International Classification for Physics" [ICSU/AB], dans [COU 81], il a établi une première classification de mécanismes de pannes dues à des phénomènes physiques de la matière (transport de particules, croissance de surface, claquage de diélectrique,...).

Cette analyse des mécanismes de pannes fait partie de la théorie des circuits autotestables, puisqu'elle aboutit à une distribution des mécanismes de pannes pouvant être résumée dans des tableaux donnant des pourcentages de pannes sur les pastilles, ou des pannes relatives aux boîtiers.

2.3 Modes de défaillances électriques

La table I résume les modes de défaillance électrique consécutifs à l'étude des mécanimes de pannes. Toutefois, on remarque que cette étude ne concerne que les pannes matérielles survenant pendant la vie utile d'un circuit, et non pas les pannes relatives au processus de fabrication auxquelles des procédures de test de fin de fabrication conviendraient plus naturellement.

Localisation		Mécanisme physique pouvant être invoqué	Mode de défaillance électrique
coupures de contacts inter-niveaux	contact Al-Si poly contact Al-diffusion précontact Si poly diff	phénomène de transport (faiblesse mécanique)	circuit ouvert circuit ouvert (circuit ouvert)
coupures de lignes d'inter-connexions	Al Si poly Diffusion	claquage (faiblesse mécanique-électromigration) contamination ionique	circuit ouvert (circuit ouvert-grille flottante) circuit ouvert
courts-circuits entre des éléments appartenant à un même niveau	Al Si poly Diffusion	phénomènes électro-chimiques fabrication seulement (contamination ionique)	court-circuit (court-circuit)
court-circuit entre les niveaux proches l'un de l'autre sur la même verticale	Al-Si poly Diffusion	fabrication seulement claquage par overstress (périphérie) contamination ionique (intérieur)	MOS s-on, s-open
court-circuit entre un élément vertical et un autre niveau passant à proximité	contact Al-Diffusion et Si poly proche	fabrication seulement	
coupures ou court-circuit de connexions pastille-extérieur	soudures fils (wires) connexions externes	inter-Diffusion fatigue thermique corrosion	circuit ouvert circuit ouvert circuit ouvert
court-circuit entre connexions pastille-extérieur	fils, connexions externes	particules conductrices	court-circuit

Table 1 - Modes de défaillance électriques N-MOS(H-MOS) [COU 81]

3 - CLASSES D'HYPOTHESES DE PANNES

Trois classes de pannes possibles seront considérées dans la suite, en fonction du nombre de défauts physiques et de leurs types. Dans la suite on ne considèrera que la technologie NMOS utilisée pour COBRA. On admettra aussi que les pannes modélisées par ces classes ne provoquent que des erreurs logiques (à une tension sera associée un niveau logique 0 ou 1).

Classe 0

Cette classe est celle des défauts simples physiques, tels par exemple qu'un contact défaillant, un MOS défaillant, un conducteur coupé. Les court-circuits ne font pas partie de cette classe : on considère que deux "défauts" sont nécessaires. En cas de grille flottante, on considère que le mode de panne associé est un MOS s-on ou s-open. La notion de défaut simple est naturellement relative à un circuit de référence.

Classe 1

Par rapport à la classe 0, on admet les court-circuits entre Al, uniquement et entre un Al et l'autre Al le plus proche géographiquement. Bien que le cas similaire entre deux diffusions soit sans doute moins probable, on le considèrera néanmoins, car la difficulté du problème est comparable.

Classe 2

Cette classe contient naturellement ce qui n 'a pas été inclus dans les classes 0 et 1, tels que les court-circuits entre un Al et un autre Al quelconque dans le circuit. Ces court-circuits sont également étendus à plusieurs participants : 3, 4 .... .

Le cas des diffusions est étendu de façon semblable. Les défauts sont considérés comme multiples, au sens classique du terme. Seuls les défauts classés "fabrication seulement" restant exclus (court-circuits Al-diffusion ...)

classe 0	classe 1	classe 2
un défaut physique simple 1contact, 1précontact, 1MOS s-on ou s-open, 1Alu coupé ... grille flottante MOS s-	classe 0 plus : 1 court-circuit entre un Alu et l'autre Alu le plus proche géographiquement Idem pour la Diffusion	classe 1 plus : court-circuits entre Alu quelconques (idem pour la Diffusion) défauts multiples

Table 2 - Classes d'hypothèses de pannes [COU 81]

C'est principalement la classe 1 qui est retenue pour la conception des circuits autotestables. C'est donc cette classe qui nous intéresse pour la conception de COBRA.

4 - DEFINITIONS DE BASE

4.1. Critères de test

4.1.1 Test en-ligne/hors-ligne

Le test en-ligne est la détection de pannes dans un circuit, au cours du déroulement d'un programme d'application. Cette notion de test en-ligne est celle qui sera appliquée pendant le fonctionnement de COBRA. De la même façon, le test hors ligne est défini par : la détection de pannes, en dehors de toute application, en faisant dérouler un programme de test exécuté par le circuit même.

4.1.2 Test continu/discontinu

Il existe d'autres critères servant à préciser un type de test, comme les critères de test continu ou discontinu pour qualifier la notion de test en-ligne : dans le test en-ligne continu, la simultanéité du test et du déroulement de l'application est parfaite, tandis qu'une alternance d'une phase de test et des phases d'application a lieu pour le test en-ligne discontinu.

4.1.3 Test in-situ/ex-situ

On définit aussi le critère de test in-situ/ex-situ pour qualifier un test hors-ligne. Dans le cas d'un test hors-ligne in-situ, l'unité centrale intégrée ne peut qu'exécuter des programmes, spécifiques en vue de test. Bien que des mécanismes matériels puissent être actifs durant le déroulement de ces programmes, aucun moyen n'est supposé tel que l'on puisse à tout instant observer les valeurs des signaux transitant sur les bus adresses, données ou contrôle.

Dans le cas d'un test hors-ligne ex-situ, au contraire, ces valeurs sont supposées observables. Ce cas est celui des machines de test classiques, i.e, celles dont le schéma de fonctionnement consiste à appliquer des vecteurs de test, puis à observer si les sorties correspondent ou non aux valeurs normales (sans pannes).

4.2 Circuits autotestables

Le propos de ce paragraphe est d'introduire les notions élémentaires donnant lieu à la définition générale de "circuits autotestables". Le circuit autotestable, par définition, se compose d'un bloc fonctionnel et d'un contrôleur (figure 1)

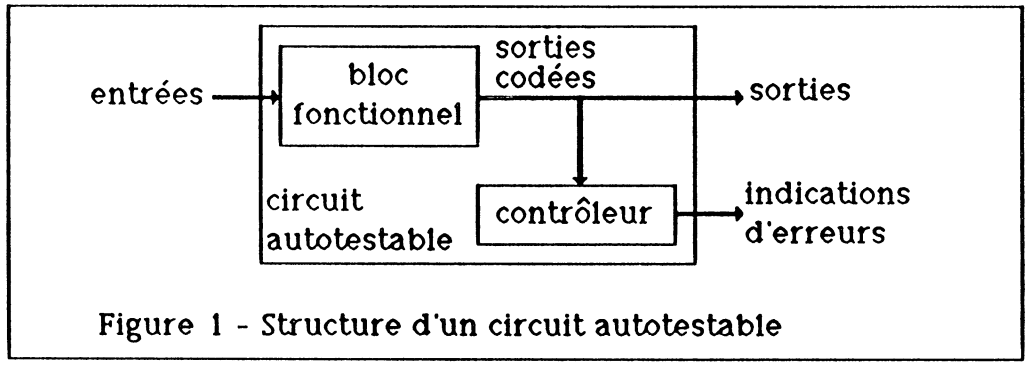


Figure 1 - Structure d'un circuit autotestable

Les définitions données dans la suite, concernent justement ces deux éléments du circuit autotestable :

Les notions de *selft-testing* (ST) et de *fault secure* (FS) concernent le bloc fonctionnel [AND 71]

Ces deux mêmes notions ainsi que la notion de *code disjoint* (CD) concernent le contrôleur.

Notons que la notion de codage des informations est nécessaire à la détection des erreurs dans un bloc fonctionnel. Ainsi le principe de fonctionnement de base des circuits autotestables (figure 1), est le suivant :

Le bloc fonctionnel transforme les valeurs encodées sur ses entrées, en des valeurs encodées (selon sa fonctionnalité) sur ses sorties. Le contrôleur transforme ces valeurs de sorties en signaux d'indication d'erreur détectant immédiatement les manifestations de défauts dans le circuit.

Les conventions qui suivent sont reprises de SMITH dans [SMI 78] :

soit G un bloc fonctionnel combinatoire possédant r entrées primaires et q sorties. Les 2^r vecteurs binaires forment l'ensemble X des vecteurs d'entrées. L'ensemble Y des vecteurs de sorties est formé par les 2^q vecteurs binaires (figure 2).

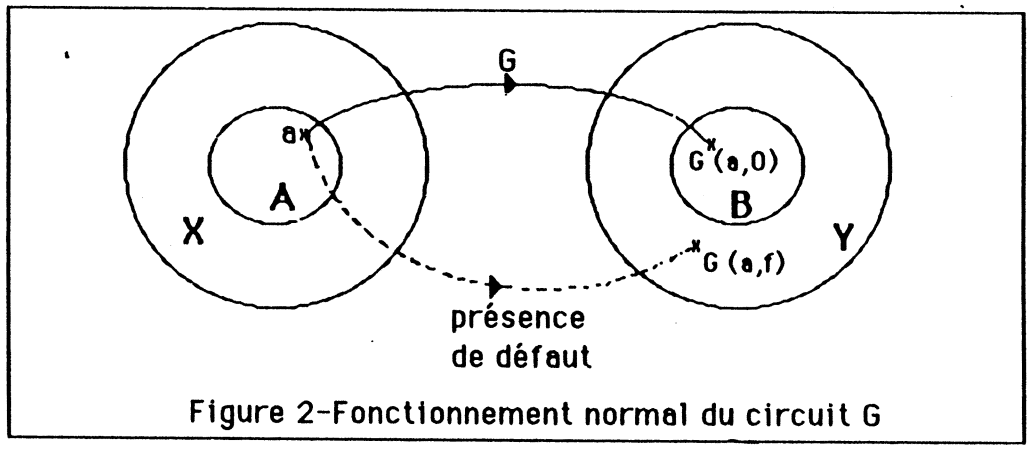


Figure 2-Fonctionnement normal du circuit G

En fonctionnement normal (avant l'occurrence d'un défaut), le bloc G reçoit sur ses entrées un sous-ensemble A de vecteurs de X, appelé code d'entrée et il produit un sous-ensemble B de vecteurs de Y, appelé code de sortie. Le bloc G est conçu de sorte qu'en présence de défauts, il produit des valeurs en dehors du code de sortie B. On notera b = G (a, o), la valeur de la sortie de G recevant a en son entrée, en absence de défaut. En présence d'un défaut f, on notera G(a, f) la valeur de la sortie, f étant un élément de l'ensemble de défauts F.

Cet ensemble de notations est dû à ANDERSON [AND 71]. Ceci explique, par ailleurs, la terminologie anglo-saxonne utilisée pour les définitions de circuits autotestables.

Dans ce qui suit, nous conserverons cette terminologie afin d'éviter toute confusion.

Définition D1

G est "self testing" (ST) pour un ensemble F de défauts si :

$\forall f \in F, \exists a \in A$ **tel que** $G(a, f) \notin B$

Autrement dit, le défaut peut être détecté si on balaye l'ensemble des vecteurs de A.

Définition D2

G est "fault secure" (FS) pour un ensemble de défauts F si :

- Pour tout défaut f de F la sortie est soit juste, soit hors du Code de Sortie B (figure 3)

$\forall f \in F, \exists a \in A$ **on a soit** $G(a, f) = G(a, 0)$

soit $G(a, f) \notin B$

Autrement dit en présence d'un défaut ou bien la sortie est juste, ou bien elle est fausse mais on détecte le défaut.

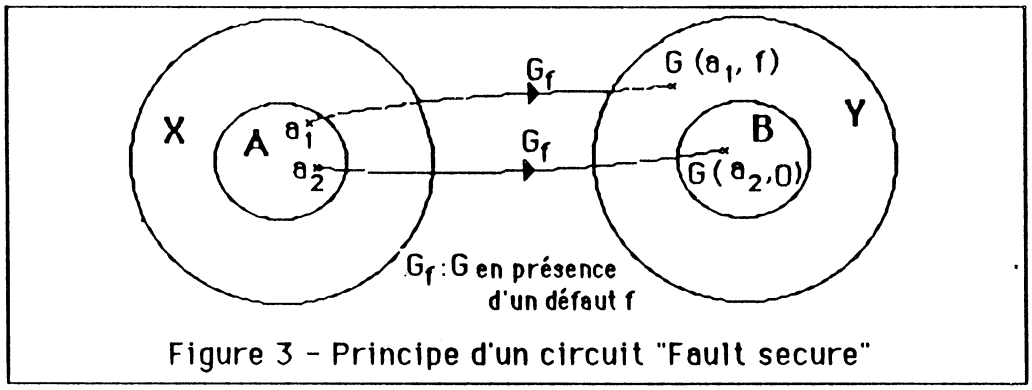


Figure 3 - Principe d'un circuit "Fault secure"

Définition D3

G est "Totally Self Checking" (TSC) pour un ensemble de défauts F si le circuit est "Self Testing" et "Fault Secure" pour l'ensemble F.

De cette dernière définition est déduite une propriété qui est qu' "un bloc fonctionnel G accomplit le "Totally Self Checking Goal" (TSC Goal), si la première sortie erronée due aux défauts de l'ensemble F est en dehors du code de sortie".

Le TSC goal peut naturellement être atteint par un circuit TSC si l'hypothèse suivante est respectée :

Hypothèse H1

Entre l'occurrence de deux défauts quelconques appartenant à l'ensemble F, il s'écoule un laps de temps suffisant pour que tous les vecteurs du code d'entrée A soit appliquées aux entrées du circuit G.

Définition D4

Un circuit est à "codes disjoints" (CD) si :

- Tout mot du Code d'entrée est transformé en un mot du Code de Sortie (figure 4)

- Tout mot hors du Code d'entrée est transformé en un mot hors du Code de Sortie.

$$\forall\ a \in A,\ G(a, 0) \in B$$

$$\forall\ x \notin A,\ G(x, 0) \notin B$$

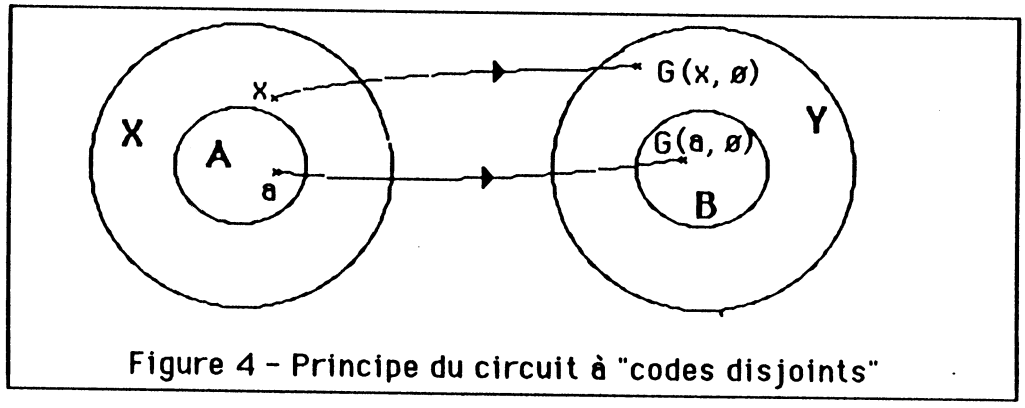


Figure 4 - Principe du circuit à "codes disjoints"

Définition D5

Un circuit G est un contrôleur TSC s'il est "Totally Self Checking" et à "Codes disjoints".

Il va de soi qu'un contrôleur doit être à codes disjoints puisque son rôle est :

- de détecter les mots issus du bloc fonctionnel et qui n'appartiennent pas au code B
- et de les signaler en émettant un signal d'erreur

Pour atteindre le TSC Goal malgré les pannes qui peuvent affecter le Contrôleur, on pourrait utiliser des contrôleurs Totally Self Checking c'est à dire Self Testing et Fault Secure.

L'examen de la littérature spécialisée ([JAN 85], [DAV 78], [DIA 79], [RED 74],...) montre que de nombreux auteurs se sont interessés à ce type de contrôleurs.

En fait ces propriétés ne sont pas nécessaires, en particulier la propriété Fault Secure. L'important est que le contrôleur reste à codes disjoints même en présence de défauts internes; lorsqu'une séquencc de pannes fait que le contrôleur ne peut plus être à codes disjoints, certaines propriétés doivent être assurées.

Ces remarques ont amené à introduire les contrôleurs Strongly Code Disjoint, ou SCD [NIC 84]. Ils représentent la plus large classe de contrôleurs qui associés à des circuits SFS permet d'atteindre le TSG Goal.

Pour éviter l'utilisation de listes énormes de défauts individuels, la définition des circuits SFS a été modifiée pour couvrir une classe des hypothèses de pannes (notée C_f).

L'introduction de quelques définitions est nécessaire avant de donner une définition de circuits SFS pour une classe C_f.

Définition 6

Un circuit G est redondant pour un défaut f si la fonction qu'il réalise n'est pas modifiée en présence du défaut f [BRE 76].

Cette dernière définition peut être affinée, étant donné que G peut être redondant pour l'ensemble X ou seulement pour le code d'entrée A.

Définition D7

Un circuit G est redondant pour un défaut f et pour l'ensemble des vecteurs d'entrée X si:

$$\forall\ x \in X\ ,\ G(x, f) = G(x, 0)$$

Définition D8

Un circuit G est redondant pour un défaut f et pour le code d'entrée A si :

$$\forall\ a \in A\ ,\ G(a, f) = G(a, 0)$$

Il est évident que :

- si un circuit est redondant pour un défaut f et pour l'ensemble X, il est redondant pour le même défaut f et pour le code d'entrée A.
- un circuit peut être redondant pour un défaut f et pour le code d'entrée A, sans être redondant pour le défaut f et pour l'ensemble X.

Définition D9

Pour une séquence des défauts <f1, f2, ...fn> soit k le plus petit entier pour lequel :

$$\exists\ a \in A\ ,\ G(a, \bigcup_{j=1}^{k} f_j) \neq G(a, 0)$$

Si un tel k n'existe pas, posons k=n. Alors G est "strongly fault secure" (SFS) pour la séquence <f1, f2, ...fn> si :

$$\forall\ a \in A \quad \text{soit } G(a, \bigcup_{j=1}^{k} f_j) = G(a, 0)$$

$$\text{soit } G(a, \bigcup_{j=1}^{k} f_j) \notin B$$

Définition D10

Un circuit G est "strongly fault secure" pour un ensemble de défauts F s'il est "strongly fault secure" pour chaque séquence constituée d'éléments appartenant à l'ensemble F.

Il est évident que chaque circuit TSC est SFS (k=1 dans la définition D9)

Définition D11

Un circuit est "Strongly redundant" pour une séquence de défauts <fl,f2,...fn> et pour le code d'entrée A (resp. pour l'ensemble de vecteurs d'entrée X), si le circuit est redondant pour les n séquences <fl>, <fl, f2>, ...<fl, f2, ...fn> et pour le code d'entrée A (resp. pour X).

Dans cette dernière définition l'ordre des défauts dans la séquence <fl, f2, ...fn> est utilisé pour définir l'ordre d'occurrence des défauts c'est à dire le défaut fl survient le premier, le défaut f2 le deuxième etc... On sait que la propriété de redondance est dynamique [FRI 67], [BRE 76], un circuit peut être strongly redundant pour la séquence <fl, f2, fi, fj...fn> sans être strongly redundant pour la séquence <fl, f2, fj, fi...fn>. Un circuit peut être redondant pour un défaut individuel fi selon l'existence d'autres défauts dans le circuit.

Définition D12

Un circuit est "fault secure" pour une classe C_f d'hypothèses de pannes s'il est "fault secure" pour tous les défauts dûs à la classe C_f.

Définition D13

Un circuit G est "Strongly Fault Secure" pour une classe C_f d'hypothèses de pannes si :

- **avant l'occurrence de défauts, G est "fault secure" pour la classe C_f.**

- **en présence de chaque séquence <fl, f2...fn> de défauts dûs à la classe C_f telle que le circuit G est "strongly redundant" pour la séquence <fl,f2...fn> et pour le code d'entrée A, le circuit G reste "fault secure" pour la classe C_f.**

En assumant l'hypothèse H1 et pour une classe C_f, un circuit SFS accomplit le TSC goal.

Définition D14

Un contrôleur est "Strongly Code Disjoint" (SCD) pour une classe C_f d'hypothèses de pannes s'il est à codes disjoints et si pour chaque séquence de défauts <f1, f2, ..., fn> due à la classe C_f :

Soit il existe k tel que :

$$\exists\ b \in B\ ,\ \text{tel que}\quad G(b, \bigcup_{j=1}^{k} f_j) \notin C$$

- le contrôleur est "strongly redundant" pour la séquence <f1, f2,..., f_{k-1}>

Soit le contrôleur est "strongly redundant" pour la séquence <f1, f2, ...,fn>.

La figure 5 montre le fonctionnement d'une cellule de base d'un circuit, (figure 1), qui atteint le TSC goal avec un bloc fonctionnel SFS et un contrôleur SCD [IRT 84]

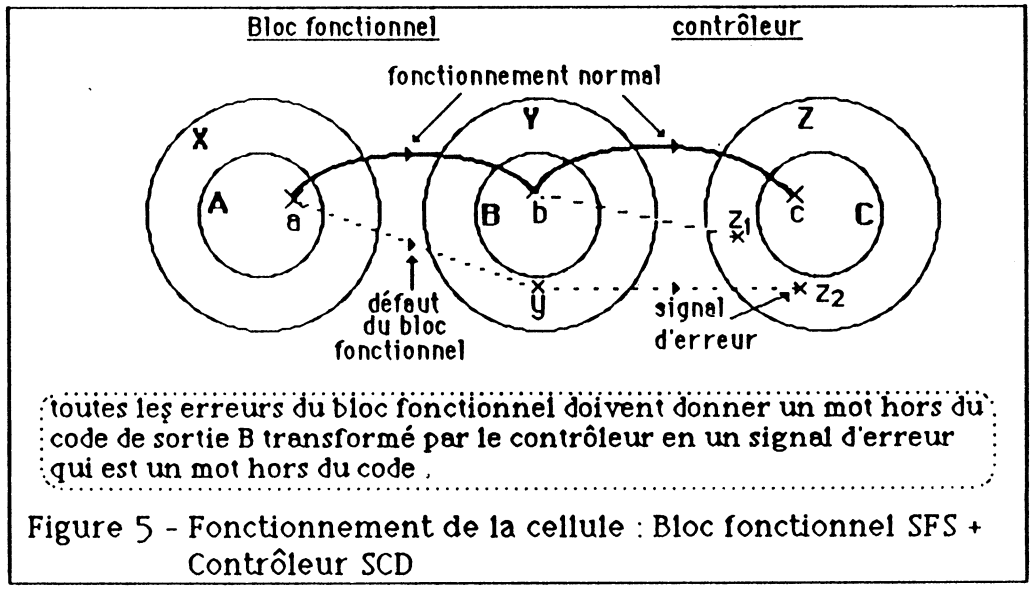


Figure 5 - Fonctionnement de la cellule : Bloc fonctionnel SFS + Contrôleur SCD

Il est nécessaire de poser l'hypothèse H2 suivante concernant l'occurrence de défauts dans un système composé d'un bloc fonctionnel et d'un contrôleur. Ce système accomplirait le TSC goal si H2 est respectée, avec un bloc fonctionnel SFS et un contrôleur SCD :

Hypothèse H2

Aprés l'occurrence d'un défaut dans le bloc fonctionnel, il s'écoule un laps de temps suffisant pour que tous les vecteurs du code d'entrée A soient appliqués dans le bloc fonctionnel, avant qu'un deuxième défaut survienne au bloc fonctionnel ou au contrôleur.

Aprés l'occurrence d'un défaut dans le contrôleur, il s'écoule un laps de temps suffisant pour que tous les vecteurs du code d'entrée B soient appliqués au contrôleur, avant qu'un deuxième défaut survienne dans le contrôleur ou dans le bloc fonctionnel.

Justification simplifiée de H2

- Si une panne arrive dans le bloc fonctionnel

* ou bien elle ne modifie pas la fonction de ce circuit (il s'agit d'une panne latente)
* ou bien elle provoque des erreurs qui seront en dehors du code de sortie B.Le contrôleur étant à codes disjoints fournira un signal d'erreur avant la panne suivante en application de H2

- Si une panne arrive dans le contrôleur

* ou bien le contrôleur reste à codes disjoints (panne latente)
* ou il existe au moins une valeur d'entrée du contrôleur qui provoquera un signal d'erreur avant la panne suivante.

La figure 6 schématise l'hypothèse H2.

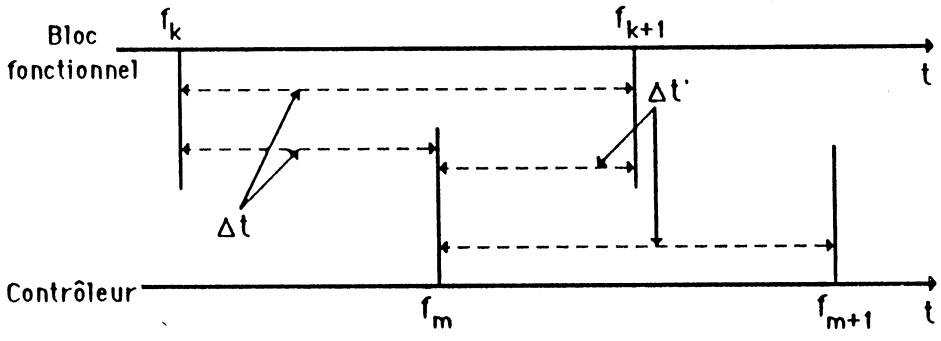


Figure 6 - Hypothèse H2 sur la répartition temporelle des pannes dans une cellule

1 - Aprés l'occurrence d'un défaut dans le bloc fonctionnel, il s'écoule un laps de temps Δt suffisant pour que tous les vecteurs du code d'entrée A soient appliqués dans le bloc fonctionnel avant qu'un deuxième défaut survienne au bloc fonctionnel ou au contrôleur

2-Aprés l'occurrence d'un défaut dans le contrôleur, il s'écoule un laps de temps $\Delta t'$ suffisant pour que tous les vecteurs du code d'entrée B soient appliqués au contrôleur avant qu'un deuxième défaut survienne dans le contrôleur ou dans le bloc fonctionnel

4.3. Interconnexion de circuits SFS

Dans un cas pratique il existe plusieurs blocs fonctionnels reliés entre eux de façon à former un circuit complet. A ces blocs SFS, on associe des contrôleurs SCD pour former un bloc global SFS. La figure 7 montre le cas de deux blocs consécutifs.

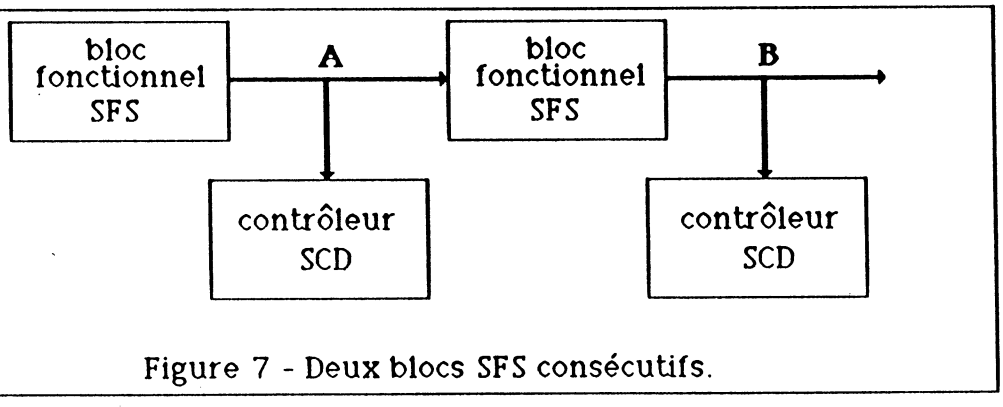


Figure 7 - Deux blocs SFS consécutifs.

Pour que le bloc global de la figure 7 soit SFS, il faut que les défauts des interconnexions soient signalés par un contrôleur.

Or le premier contrôleur assure la surveillance jusqu'au nœudA. Mais il existe des cas différents dans lesquels un contrôleur est nécessaire à chaque nœud (interconnexion de deux blocs consécutifs), ou bien un seul contrôleur suffit pour surveiller l'ensemble de deux blocs. Ces différents cas sont détaillés dans [NIC 84]. Le but en fait, est d'utiliser le moins possible de contrôleurs tout en veillant à assurer la propriété SFS globale.

4.4 Cas des circuits séquentiels

La définition des circuits SFS est valable pour les circuits combinatoires seulement. Il existe pour les circuits séquentiels des hypothèses de pannes qui leurs sont particulières, ainsi que des définitions de base de circuits "sequentially self-checking" données par Viaud et David[VIA 80].

Une famille de contrôleurs "Strongly Language Disjoint" (SLD) définie dans [JAN 85], est associée aux circuits séquentiels. Les définitions appliquées pour les circuits combinatoires ont été adaptées aux circuits séquentiels, et les différentes définitions concernant les contrôleurs des circuits séquentiels sont détaillées dans [JAN 85].

L'état actuel des connaissances permet, toutefois, de réaliser un microprocesseur autotestable selon les définitions données dans le présent chapitre, et ceci en respectant les règles de conception données dans la suite.

4.5 Codes de sorties

Les erreurs possibles dans un circuit, pour la classe 1 de pannes peuvent être classées selon trois types :

- erreurs simples : n'affectent qu'un seul bit dans un mot.
- erreurs unidirectionnelles : n'affectent les bits que dans un seul sens (1 vers 0 par exemple).
- erreurs multiples : peuvent modifier un nombre quelconque de bits, et dans les deux sens.

A ces trois types d'erreurs correspondent trois types de codes qui permettent de les détecter.

4.5.1 Code de parité

C'est un code séparable simple. Il permet de détecter les erreurs simples. La génération du code de parité de n variables e_i est obtenue si la relation suivante est vérifiée :

$$e_1 \oplus e_2 \oplus \ldots \oplus e_n = a$$

où a = 0 pour un code de parité paire
a = 1 pour un code de parité impaire

L'application de cette relation d'un point de vue pratique réalise le contrôle de parité des sorties codées d'un bloc fonctionnel. Ces sorties doivent être codées en parité, et le bloc fonctionnel doit être conçu de sorte que toute panne ne provoque que des erreurs simples aux sorties du bloc fonctionnel, et ceci grâce aux règles de conception de ce bloc.

4.5.2 Code de Berger [BER 61]

C'est un code séparable non ordonné. Il permet de détecter les erreurs unidirectionnelles. Il est obtenu par la concaténation à l'information I (de n bits), de la représentation P du complément du nombre de "1" contenus dans I. k repésente le nombre des bits du code concaténé à l'information I à coder: $k = \lceil Log_2 (n + 1) \rceil$ le premier entier supérieur à $Log_2(n+1)$, (figure 8).

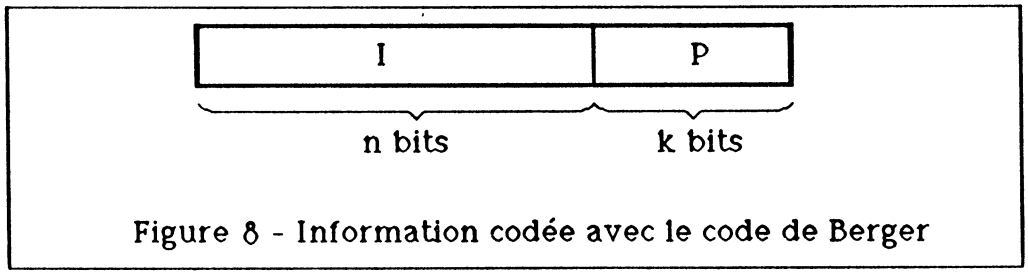


Figure 8 - Information codée avec le code de Berger

C'est le code séparable optimal pour la détection des erreurs unidirectionnelles, en ce sens qu'il utilise le moins de bits de contrôle pour un nombre donné de bits d'informations. Le bloc fonctionnel susceptible d'avoir en ses sorites des erreurs unidirectionnelles doit avoir ses sorties codées avec le code de Berger pour permettre la détection des erreurs par le contrôleur. Ce dernier est composé du point de vue pratique de deux éléments :

- un circuit combinatoire constitué d'additionneurs 3 bits, qui a pour rôle de regénérer à partir de l'information I, les bits de contrôle P complémentés.

- un comparateur, ou plutôt des contrôleurs double-rail (voir 4-5-3) qui comparent les bits de contrôle P des sorties codées avec les bits générés dans les additionneurs.

La figure 9 donne le schéma de principe d'un contrôleur du code de Berger. La figure 10 donne un exemple de générateur des bits de contrôle, (C : retenue, S : sortie).

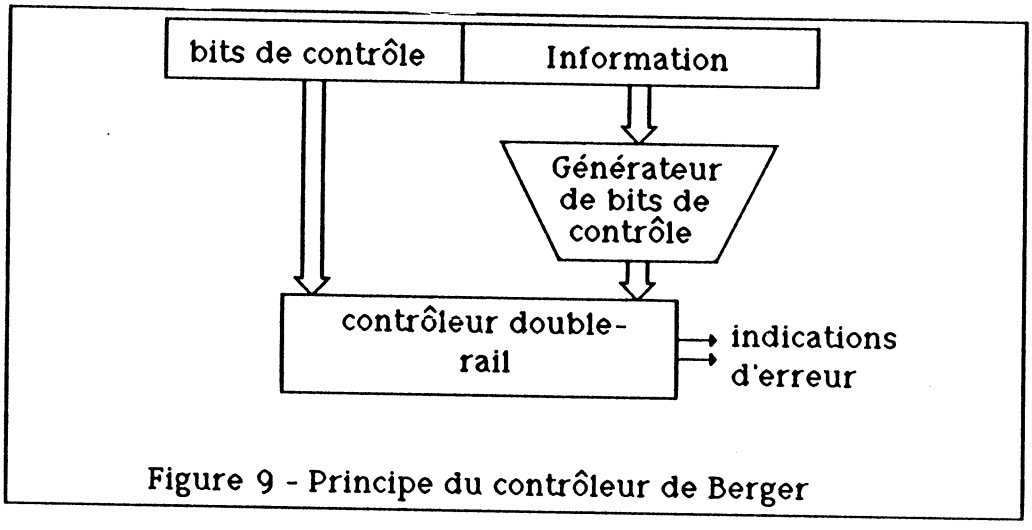


Figure 9 - Principe du contrôleur de Berger

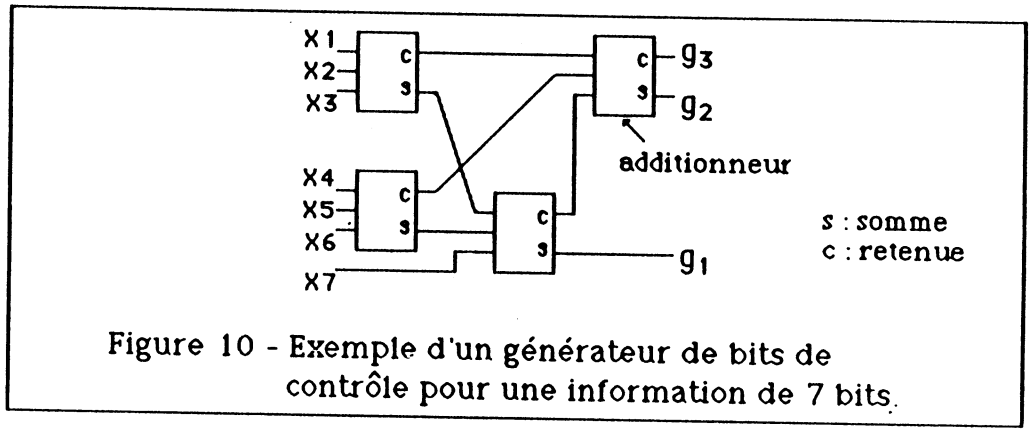


Figure 10 - Exemple d'un générateur de bits de contrôle pour une information de 7 bits.

4.5.3 Codes à duplication

Deux principales méthodes appliquent le principe des codes à duplication, adaptées à la détection des erreurs multiples [GEF 76] :

- méthode à duplication directe, qui consiste à faire correspondre au bloc fonctionnel B1, un bloc identique B2 fonctionnant en parallèle, ensuite à comparer les sorties des deux blocs.

- méthode à duplication et à complémentation , appelée "double-rail", qui consiste à faire correspondre à B1 un bloc dual B2, fonctionnant en parallèle, et à comparer les sorties des deux blocs avec un contrôleur "double-rail".

Pour le contrôle "double-rail" les vecteurs d'entrées qui appartiennent au code sont des paires (0, 1) ou (1, 0), alors que (0, 0) et (1, 1) sont en dehors du code. Le cas inverse est celui du code à duplication. Dans les deux cas on peut utiliser un contrôleur double-rail.

Un contrôleur double-rail est T.S.C., et il transforme les 2n groupes d'entrées (a_1,....., a_n) et (b_1, ..., b_n) organisés en n paires (a_i, b_i), en une paire de sortie, soit (f_n, g_n). Durant le fonctionnement normal (sans panne), on doit avoir $f_n = \neg g_n$. Mais dès que l'une des paires d'entrées (a_i, b_i) ne vérifie pas la relation $a_i = \neg b_i$ alors il y a indication d'erreur par $f_n = g_n$.

La cellule de base d'un contrôleur double-rail est donné à la figure 11. La figure 12 donne la structure en arbre pour quatre paires d'entrées

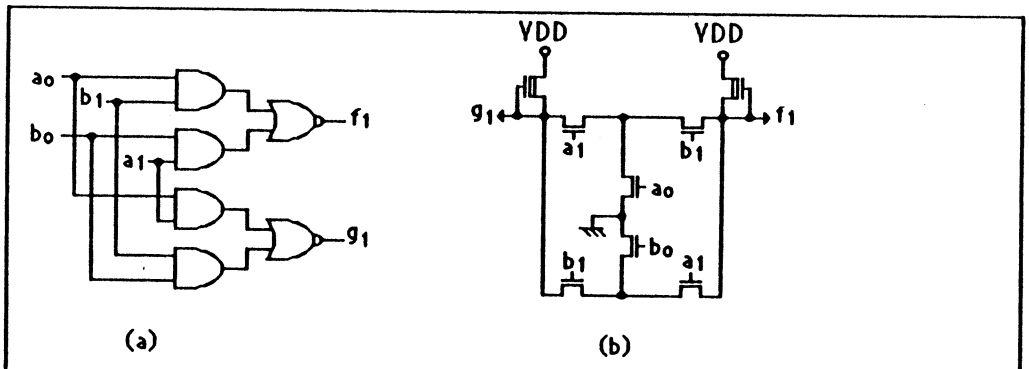


Figure 11 - Cellule de base d'un contrôleur double-rail
(a) modèle logique
(b) solution électrique à 8 transistors [NIC 84]

Le code à duplication est utilisé lorsque la protection offerte par les autres méthodes est insuffisante. Son inconvénient est la surface importante qu'il occupe (bloc dual + contrôleur + interconnexions en plus du bloc fonctionnel d'origine).

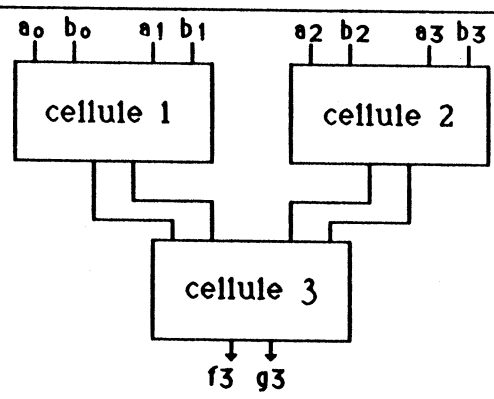


Figure 12 - Contrôleur double-rail pour 4 paires d'entrées.

Notons que les codes présentés dans ce chapitre font partie d'un ensemble de codes divers et variés [CRO 78]. Leur choix était orienté par le classement des erreurs dues à la classe 1 d'hypothèses de pannes. En effet, il est possible de regrouper les erreurs dues à la classe 1 en vue d'obtenir les règles de conception des circuits SFS.

5 - ERREURS DUES A LA CLASSE 1

On considère que les lignes d'alimentation sont faites soit en Aluminum (Alu), soit en diffusion, [NIC 84] :

1 - un contact, un précontact, un MOS s-on ou un MOS s-open -> une erreur simple sur la ligne concernée ou la sortie de la porte concernée par le défaut.

2 - une ligne de diffusion coupée :
  2a - une ligne de signal -> une erreur simple sur la ligne concernée ou sur la porte concernée par la coupure.
  2b - une ligne d'alimentation->une erreur multiple unidirectionnelle aux sorties des portes alimentées par la ligne.

3 - une ligne de poly coupée -> une erreur simple sur cette ligne.

4 - une ligne d'Alu coupée :
  4a - une ligne de signal -> une erreur simple sur cette ligne
  4b - une ligne d'alimentation->une erreur multiple unidirectionnelle aux sorties des portes alimentées par ces lignes.

5 - un court-circuit entre une ligne d'Alu et une autre ligne d'Alu la plus proche géographiquement, ou un court-circuit entre une ligne de diffusion et une autre ligne de diffusion la plus proche géographiquement :
  5a - un court-circuit entre deux lignes VDD ou deux lignesVSS -> pas d'erreur.
  5b - un court-circuit entre une ligne VDD et une ligne VSS -> on considère que la destruction du circuit est totale.
  5c - un court-circuit entre une ligne de signal et une ligne d'alimentation -> une erreur simple sur la ligne de signal (collage à la valeur logique VSS ou VDD).
  5d - un court-circuit entre une ligne interne d'une porte et une ligne d'alimentation -> une erreur simple à la sortie de la porte.
  5e - un court-circuit entre deux lignes de signal -> une erreur qui peut concerner les deux lignes mais seulement l'une des deux à chaque fois -> une erreur simple.
  5f - un court-circuit entre deux lignes internes de deux portes différentes -> une erreur qui peut concerner les sorties de deux portes mais seulement l'une des deux à chaque fois -> une erreur simple .

5g- un court-circuit entre une ligne de signal et une ligne interne d'une porte -> une erreur qui peut concerner la ligne de signal et la sortie de la porte mais l'une des deux à chaque fois, alors une erreur simple.

5h- un court-circuit entre deux lignes internes d'une porte -> une erreur simple à la sortie de la porte.

Tous ces cas peuvent être regroupés en quatre groupes listés ci-dessous :

A - Hypothèses de pannes qui peuvent provoquer des erreurs simples concernant une ligne de signal ou une sortie d'une porte (1, 2a, 3, 4a, 5c, 5d, 5h).

B - Hypothèses de pannes qui peuvent provoquer des erreurs concernant deux lignes signal/sorties de portes, mais l'une des deux à chaque fois->une erreur simple (5e,5f,5g).

C - Hypothèses de pannes qui peuvent provoquer des erreurs multiples unidirectionnelles (2b, 4b).

D - Hypothèses de pannes ne provoquant pas d'erreur (5a).

Les types d'erreurs qui seront produits aux sorties primaires d'un circuit fonctionnel dépendent de la propagation des erreurs depuis l'origine de leur apparition jusqu'aux sorties primaires du bloc fonctionnel.

Les propriétés de propagation d'erreurs sont utilisées dans [SMI 77] et [CRO 78], mais ces propriétés sont dérivées au niveau des portes logiques, en considérant par exemple des portes OR, NAND etc...

Par conséquent, on doit transposer ces propriétés au niveau du transistor car on considère des hypothèses de pannes à ce niveau.

Par la suite, nous définissons le chemin entre une ligne interne et les sorties primaires des circuits NMOS, le degré de divergence d'une ligne, le degré de divergence maximal d'un bloc et la parité d'inversion d'un chemin, toutes les définitions sont faites à un niveau général.

Leurs applications au niveau du transistor sont données dans [NIC 83].

Définition d'un chemin

Un chemin est un ensemble ordonné de lignes entre deux points d'un bloc.

Si dans l'ensemble $< I_{i1}, I_{i2}, \ldots I_{in} >$ de lignes, la ligne $I_{i\,j+1}$ est le successeur de I_{ij} pour j=1 jusqu'à n-1 alors cet ensemble ordonné est un chemin.

On considère deux lignes, la ligne I_j successeur de la ligne I_i :

Définition du couple inversant

Un couple de lignes (I_i, I_j) sera appelé inversant, si une erreur de type D "niveau bas à la place du niveau haut" (respectivement $\neg$D "niveau haut à la place du niveau bas") sur la ligne Ii ne peut produire qu'une erreur de type $\neg$D (respectivement D) sur le ligne I_j.

Définition du couple non inversant

Un couple de lignes (I_i, I_j) sera appelé non inversant, si une erreur de type D (respectivement $\neg$D) sur la ligne I_i ne peut produire qu'une erreur de type D (respectivement $\neg$D) sur la ligne I_j.

Définition des couples inversants/non inversants pour les transistors MOS.

- le couple (grille, drain) d'un transistor signal MOS est un couple inversant.
- le couple (grille, drain/source) d'un transistor MOS utilisé comme interrupteur n'est pas défini par rapport à la propriété inversant/non inversant en raison des positions relatives réversibles du drain et de la source. A travers un interrupteur MOS parfois c'est le niveau 0 qui est transmis (dans ce cas la parité d'inversion est 1 comme pour un MOS signal), et parfois c'est le niveau 1 qui est transmis (dans ce cas la parité d'inversion est 0).
- tous les autres couples sont non inversant.

Définition de la parité d'inversion

La parité d'inversion IP (p) d'un chemin p, dans lequel la propriété d'inversion est définie pour tous les couples de deux lignes successives, est le nombre binaire IP (p) {0, 1}, tel que IP(p) = n modulo 2, n étant le nombre de couples inversants de lignes successives du chemin p.

Définition du degré de divergence

Le degré de divergence d'une ligne $I_{i,1}$ interne d'un bloc fonctionnel est le nombre des sorties primaires I_{in} du bloc pour lesquelles il existe des chemins $< I_{i1}, I_{i2}, \ldots I_{in} >$.

Définition du degré de divergence maximal

Le degré de divergence maximal d'un bloc fonctionnel, pour un ensemble de lignes internes $I_1, I_2, \ldots I_k$ est égal au maximum des degrés de divergences des lignes $I_1, I_2, \ldots I_k$.

Ces quelques définitions sont utilisées dans la suite, par les règles de conception des circuits SFS.

6 - REGLES DE CONCEPTION DES CIRCUITS SFS POUR LA CLASSE1

Il s'agit des règles de dessin similaires à celles destinées à la conception de circuits testables par [GAL 80].

La différence est que ces règles sont moins restrictives et qu'elles sont destinées à la conception de circuits SFS. Lorsqu'il s'agira de vérifier la validité de la conception d'un système SFS, on se reportera à ces règles pour vérifier la conformité de telle ou telle autre conception, et ceci en fonction des codes de sorties des blocs, s'ils détectent les erreurs simples, unidirectionnelles ou multiples.

Ces règles sont données en deux groupe, en accord avec la définition D13 des circuit SFS pour une classe C_f : le premier groupe introduit la propriété "Fault Secure" avant l'occurrence de défaut. Le second garantit que le circuit conserve cette propriété en présence de séquences de défauts pour lesquelles le circuit est "Strongly Redundant". Nous donnerons ici les règles seules, sachant qu'une analyse complète est donnée dans [NIC 84].

6.1 Erreurs simples

On définit ici les règles de conception des circuits SFS dont le code de sortie détecte les erreurs simples.

6.1.1 Circuits "Fault Secure"

Règle R1

Le degré de divergence maximal du bloc fonctionnel, pour l'ensemble de toutes les lignes du bloc, est égal à 1.

La règle R1 est donnée pour assurer que toutes les erreurs simples internes au circuit seront propagées en erreurs simples aux sorties primaires du circuit.

Cette règle est au niveau logique ; si elle est combinée avec la "réduction" des circuits [SMI 78] (élimination des redondances), l'ensemble est suffisant pour concevoir des circuits TSC pour le modèle du collage logique. Pour les modèles de pannes au niveau du transistor, quelques autres règles de conception sont nécessaires pour concevoir des circuits FS et SFS.

Règle R2

Une seule sortie primaire du bloc fonctionnel peut être erronée à cause d'une erreur multiple unidirectionnelle provoquée dans le circuit par un défaut du typeC (paragraphe5).

La règle R2 étant très générale, il est trés difficile de la mettre en œuvre. Nous donnons ci-après des règles afin de faciliter cette tâche.

Règle R2'

Chaque ligne d'alimentation (diffusion ou aluminium) du bloc est utilisée pour alimenter des groupes de portes qui sont liées (par l'intermèdiaire de chemins) à une seule sortie primaire du bloc fonctionnel.

L'application de la règle R2' est possible étant donné que chaque porte du circuit est reliée à une seule sortie primaire (règle R1).

Règle R2''

Une seule ligne VSS (diffusion ou aluminium) et une seule ligne VDD (diffusion ou aluminium) sont utilisées. L'extrémité de ces lignes est utilisée pour alimenter des portes d'un contrôleur.

6.1.2 Circuits "strongly fault secure"

Soit un circuit G conçu de façon à ce que son code de sortie détecte les erreurs simples et que les règles R1 et R2 soient vérifiées (R2 est assurée par l'intermèdiaire de R2' ou R2''). Par la suite nous présenterons les règles qui peuvent garantir que G soit SFS. Selon la définition D13, ces règles doivent garantir que la propriété "fault secure" validée par les règles R1 et R2 (R2' et R2'') ne sera pas invalidée en présence des séquences pour lesquelles le circuit G est "strongly redundant". Les quatre groupes d'hypothèses de pannes données au paragraphe 5 sont affinés ci-après:

A - hypothèses de pannes qui peuvent provoquer des erreurs simples concernant une ligne de signal ou une sortie d'une porte (1, 2a, 3, 4a, 5c, 5d, 5h).

B - hypothèses de pannes qui peuvent provoquer des erreurs concernant deux lignes de signal/sorties de portes, mais une seule des deux à la fois (5e, 5f, 5g).

B1 - hypothèses de pannes B, telles que les deux lignes concernées sont liées, par l'intermédiaire de chemins, avec la même sortie primaire du circuit.

B2 - hypothèses de pannes B, telles que les deux lignes concernées sont liées, par l'intermédiaire de chemins, avec deux sorties primaires différentes.

C - hypothèses de pannes qui peuvent provoquer des erreurs multiples unidirectionnelles (2b, 4b).

D - hypothèses de pannes ne provoquant pas d'erreurs (5a).

D1 - hypothèses de pannes D telles que les deux lignes d'alimentation concernées sont utilisées pour alimenter des portes dont les sorties sont connectées, par l'intermédiaire des chemins, avec la même sortie primaire du circuit.

D2 - hypothèse de pannes D telles que les deux lignes d'alimentation concernées sont connectées, par l'intermédiaire de chemins, avec deux sorties primaires différentes.

Proposition

Si des séquences $<f_1, f_2, \ldots, f_{k-1}>$ contenant les défauts de type B2 et pour lesquelles le circuit est "strongly redundant", ne peuvent pas survenir, et si des défauts du type D2 ne peuvent pas non plus survenir, alors un circuit conçu de façon à ce que son code de sortie détecte les erreurs simples et de façon à ce que les règles R1 et R2' soient vérifiées, est SFS pour la classe 1.

Règle R3

Pour éliminer tous les défauts du type B2 d'un circuit, il est nécessaire de répertorier tous les court-circuits entre deux lignes (de diffusion ou d'aluminium) qui sont liées (par l'intermédiaire des chemins) avec deux sorties primaires différentes. Pour de tels court-circuits, il faut soit déplacer les lignes respectives, soit les implanter avec des matériaux non court-circuitables. Par exemple, on peut implanter une ligne en diffusion et l'autre en aluminium ou en polysilicium.

Règle R4

Pour éliminer tous les défauts du type D2, il faut répertorier tous les court-circuits entre deux lignes d'alimentation du même type (VSS ou VDD), servant à alimenter des portes dont les sorties sont liées par l'intermédiaire de chemins avec deux sorties primaires différentes. Ensuite on doit éliminer tous ces court-circuits soit en déplaçant quelques lignes, soit en utilisant des matériaux non court-circuitables.

6.2 Erreurs unidirectionnelles

On définit ici les règles de conception des circuits SFS dont le code de sortie détecte les erreurs unidirectionnelles.

6.2.1 Circuits "Fault secure"

Règle R5

Tous les chemins entre une ligne divergente et les sorties primaires du circuit ont la même parité d'inversion (0 ou 1)

Règle R6

Pour assurer la propagation des erreurs multiples unidirectionnelles, provoquées par les défauts du type C, en erreurs unidirectionnelles aux sorties primaires du circuit, tous les chemins entre les lignes d'occurrences de ces erreurs et les sorties primaires, doivent avoir la même parité d'inversion.

Dans la pratique, la règle R6 est assurée par les règles suivantes :

Règle R6'

Chaque ligne d'alimentation du bloc est utilisée pour alimenter des groupes de portes dont les sorties sont liées aux sorties primaires du bloc par l'intermédiaire de chemins qui ont la même parité d'inversion.

Règle R6''

Une seule ligne VSS et une seule ligne VDD sont utilisées pour alimenter le bloc, les extrémités de ces lignes sont utilisées pour alimenter des portes d'un contrôleur.

6.2.2 Circuits "Strongly Fault Secure"

Soit un circuit G conçu de façon à ce que son code détecte des erreurs unidirectionnelles et de façon à ce que les règles R5 et R6 (R6' et R6") soient vérifiées. Les quatre groupes d'hypothèses de pannes donnés au paragraphe 5 sont affinés ci-après :

A - hypothèses de pannes qui peuvent provoquer des erreurs simples concernant une ligne du circuit (1, 2a, 3, 4a, 5c, 5d, 5h).

B - hypothèses de pannes qui peuvent provoquer des erreurs concernant deux lignes du circuit, mais une seule des deux à chaque fois (5e, 5f, 5g).

B1'- hypothèses des pannes B, telles que les deux lignes concernées soient regroupées dans le même groupe de lignes (étant donné que la règle R5 est vérifiée, deux groupes de lignes sont définis selon des parités d'inversion qui peuvent être égales à 0 ou à 1).

B2' - hypothèses de pannes B, telles que l'une des lignes concernées appartienne au groupe des lignes défini par des parités d'inversion égales à 0 et l'autre ligne appartient au groupe défini par des parités d'inversion égales à 1.

C - hypothèses de pannes qui peuvent provoquer des erreurs multiples unidirectionnelles (2b, 4b).

D - hypothèses de pannes ne provoquant pas d'erreurs (5a).

D1'- hypothèses de pannes de type D telles que les deux lignes concernées appartiennent au même groupe (étant donné que la règle R6 est vérifiée ; deux groupes de lignes d'alimentation sont définis selon des parités d'inversion qui peuvent être égales à 0 ou à 1).

D2' - hypothèses de pannes de types D telles qu'une ligne concernée appartient au groupe des lignes d'alimentation défini par des parités d'inversion qui peuvent être égales à 0 et l'autre ligne appartient au groupe des lignes défini par des parités d'inversion égales à 1.

Proposition

Si des séquences $<f_1, f_2, \ldots, f_{k-1}>$ contenant les défauts de type B2' et pour lesquelles le circuit est "strongly redundant", ne peuvent pas survenir, et si des défauts du type D2' ne peuvent pas non plus survenir, alors un circuit conçu de façon à ce que son code de sortie détecte les erreurs unidirectionnelles et les règles R5 et R6' (ou R6") sont vérifiées, est SFS pour la classe 1.

Règle R7

Pour éliminer tous les défauts du type B2' il faut répertorier tous les court-circuits entre deux lignes (de diffusion ou d'aluminium) tels que : les parités d'inversion des chemins entre l'une des deux lignes et les sorties primaires sont égales à 0 et les parités d'inversion des chemins entre l'autre ligne et les sorties primaires sont égales à 1. Pour de tels court-circuits, il faut soit déplacer les lignes respectives, soit les implanter avec des matériaux non court-circuitables. Par exemple une ligne peut être implantée en diffusion et l'autre en aluminium ou en polysilicium.

Règle R8

Pour éliminer tous les défauts du type D2' on doit lister tous les court-circuits entre deux lignes d'alimentation du même type (VDD ou VSS) telles que : l'une des lignes alimente des portes dont les sorties sont liées avec les sorties primaires par l'intermédiaire de chemins ayant des parités d'inversion égales à 0 et l'autre ligne alimente des portes dont les sorties sont liées avec les sorties primaires par l'intermédiaire de chemins ayant des parités d'inversion égales à 1. Ensuite tous ces court-circuits doivent être éliminés, soit en déplaçant quelques lignes, soit en utilisant des matériaux non court-circuitables.

6.3 Erreurs multiples

Dans cette section on propose des règles de conception des circuits SFS dont le code de sortie détecte les erreurs multiples. Deux types de code de sortie sont considérés : l'un est le code de duplication, l'autre est le code "double-rail".

Pour les deux codes, les sorties primaires des circuits sont partitionnées en deux groupes. Chaque sortie primaire d'un groupe est associée avec une sortie primaire dupliquée (resp. "double-rail") de l'autre groupe pour le code de duplication (resp. pour le code "double-rail"). On peut considérer que chaque sortie de porte est associée ou non avec une sortie de porte dupliquée (resp. complémentaire) pour le code de duplication (resp. pour le code "double-rail"). De plus, on peut considérer que chaque ligne interne est associée ou non à une ligne interne dupliquée pour le code de duplication.

6.3.1 Circuits "fault secure"

Règle R9

Chaque ligne est liée, par l'intermédiaire de chemins, avec des sorties primaires qui appartiennent au même groupe.

Règle R10

Le circuit doit être conçu de façon à ce que les erreurs multiples unidirectionnelles provoquées dans le circuit par les défauts du types C soient partagées aux sorties primaires en erreurs détectables par le code de duplication (ou par le code double-rail).

La règle R10 étant trés générale, il est difficile de la mettre en œuvre:

Règle R10'

Chaque ligne d'alimentation est utilisée pour alimenter des portes dont les sorties sont liées (par l'intermédiaire de chemins) avec des sorties primaires du même groupe.

Règle R10''

Une ligne VDD et une ligne VSS sont utilisées. L'extrémité de chacune de ces lignes est utilisée pour alimenter deux portes dont les sorties sont des sorties primaires complémentaires. On considère que le code de sortie est le code double rail.

Règle R10 '''

Une seule ligne VSS et une seule ligne VDD sont utilisées pour alimenter le circuit. L'extrémité de ces lignes est utilisée pour alimenter les portes logiques d'un contrôleur.

6.3.2 Circuits "Strongly fault secure"

Les quatre groupes d'hypothèses de pannes donnés au paragraphe 5 sont affinés ci-après :

A - hypothèses de pannes qui peuvent provoquer de erreurs simples concernant une ligne de signal ou une sortie d'une porte (1, 2a, 3, 4a, 5c, 5d, 5h).

B - hypothèses de pannes qui peuvent provoquer des erreurs concernant deux lignes mais une seule des deux à chaque fois (5e, 5f, 5g).

B1"- hypothèses de pannes B telles que les deux lignes concernées sont liées avec des sorties primaires du même groupe.

B2"- hypothèses de pannes B telles que les deux lignes concernées sont liées avec des sorties primaires des deux groupes.

C - hypothèses de pannes qui peuvent provoquer des erreurs unidirectionnelles multiples dans les circuits (2b, 4b).

D - hypothèses de pannes ne provoquant pas d'erreur (5a).

D1"- hypothèses de pannes D telles que les deux lignes d'alimentation concernées sont utilisées pour alimenter des portes dont les sorties sont liées avec le même groupe des sorties primaires.

D2"- hypothèses de pannes D telles que les deux lignes d'alimentation concernées sont utilisées pour alimenter des portes dont les sorties sont liées avec des sorties primaires des deux groupes.

Règle R11

Pour éliminer tous les défauts du type B2" d'un circuit, il est nécessaire de répertorier tous les court-circuits entre deux lignes qui sont liées avec des sorties primaires appartenant à deux groupes différents. Pour de tels court-circuits, il faut, soit éloigner les lignes respectives, soit les implanter avec des matériaux non court-circuitables. Par exemple, une ligne peut être implantée en diffusion et l'autre en aluminium ou en polysilicium.

Règle R12

Pour éliminer tous les défauts du type D2", il faut répertorier tous les court-circuits entre deux lignes d'alimentation du même type (VSS ou VDD), servant à alimenter des portes dont les sorties sont liées avec des sorties primaires appartenant à deux groupes différents. Ensuite il faut éliminer tous ces court-circuits soit en déplaçant quelque lignes, soit en utilisant des matériaux non court-circuitables.

7. CONCLUSION

Ce chapitre était destiné à présenter brièvement les techniques self-checking, et principalement les hypothèses de pannes dans les circuits intégrés NMOS, les codes de contrôle des sorties et les contrôleurs qui leurs sont associés, et enfin les principales règles de conception de circuits intégrés qui doivent atteindre le "T.S.C. goal".

Le but de ce chapitre est de décrire, pour le chapitre 4 concernant la conception de COBRA, les raisons pour lesquelles un élément a été dupliqué (registre, opérateur, etc), ou bien pourquoi un codage de parité a été utilisé pour coder les bus, etc...

Les références données en fin de ce chapitre permettent en outre, d'approfondir l'étude générale des circuits autotestables.

Dans la première section de ce chapitre, nous avons cité le modèle de collage logique jugé insuffisant pour modéliser toutes les pannes possibles dans un circuit intégré logique. Ensuite nous avons présenté les mécanismes de pannes/modes des défaillances électriques en tant que complément du modèle de collage dans la base de l'étude et des techniques de l'autotestabilité.

La deuxième section présentait les hypothèses de pannes dans les circuits logiques selon trois classes 0, 1 et 2. C'est la classe 1 qui a été considérée dans la conception des circuits autotestables.

La troisième section contenait les définitions élémentaires permettant de situer les critères de test utilisés dans l'étude présente. Les codes de sorties nécessaires au test des éléments de COBRA ont été définis en fin de cette section, ainsi que les principes des contrôleurs qui leur sont associés.

La quatrième section était une sorte d'extension de la section précédente puisqu'elle contenait les définitions des erreurs dues à la classe 1 et les codes nécessaires à leurs détections.

La cinquième section s'appuyait sur la section précédente pour donner les règles de conception des circuits autotestables en fonction des erreurs dues à la classe 1 d'hypothèses de pannes.

La sécurité de fonctionnement dans une application, avec ces techniques de contrôle, repose en grande partie sur le respect des règles de conception, mais aussi sur une bonne utilisation du circuit de sorte que l'on puisse étendre ces techniques dans l'environnement du circuit autotestable utilisé et de garantir ainsi le respect total de la sécurité de fonctionnement au niveau du système global et non pas seulement au niveau du circuit seul. Il n'est pas exclu par exemple d'étudier une structure de redondance matérielle (biprocesseur) en utilisant COBRA comme circuit de base.

Il est important en outre de bien connaître le langage de programmation et de ne pas "mal-utiliser" les instructions associées qui, par ailleurs, sont volontairement peu nombreuses et directes (spécifiques aux applications) en ce sens qu'elles agissent directement sur les parties physiques concernées de COBRA, afin d'éviter toute complexité.

CHAPITRE 3

METHODE ET MOYENS DE CONCEPTION

1. INTRODUCTION

De plus en plus, la conception d'un circuit intégré requiert une méthode de travail appropriée et une approche structurée. La conception consiste, partant d'une volonté de faire réaliser quelques fonctions par un circuit, à obtenir les dessins des masques de ce circuit. La méthode de conception ainsi adoptée conditionne directement la manière avec laquelle ce circuit réalisera ces fonctions. Il s'agit donc d'adopter la meilleure méthode possible. Or, il est évident qu'une méthode universelle n'existe pas en raison du nombre important de paramètres en jeu. La méthode dépend principalement du concepteur et des moyens mis en oeuvre.

Il y a quelques années les concepteurs avaient pour tâche d'intégrer quelques transistors, voire quelques centaines de transistors sur quelques millimètres carrés de silicium. A l'époque, leurs principaux soucis étaient d'ordre électrique et fonctionnel. L'expérience des concepteurs s'est accrue mais il s'agit actuellement de tenir compte de contraintes comme la limitation de la surface et l'accroissement de la vitesse. Les méthodes de conception des circuits intégrés durent être revues afin de les adapter à ces nouveaux paramètres désormais capitaux. Mais il y a eu en parallèle de l'évolution des concepteurs celle de la technologie de circuits intégrés.

Les technologues de circuits intégrés déterminent les "lois" que devrait respecter tout concepteur de circuits intégrés, faute de quoi n'importe quelle panne est possible. Ces lois se traduisent par des règles de dessin des rectangles de métal, de polysilicium ou de diffusion, leurs dimensions minimales ainsi que les distances entre les différents rectangles.

Ces lois demandent aux concepteurs de plus en plus d'imagination dans l'agencement des rectangles et dans l'organisation de la circulation du courant électrique entre les différents rectangles et les différents niveaux technologiques disponibles. Ces phénomènes deviennent d'autant plus complexes que le nombre de fonctions par millimètre carré est plus grand : actuellement le nombre de transistors par circuit pour le VLSI est de quelques centaines de milliers.

La quantité d'informations que doit manipuler le concepteur devient de plus en plus considérable. Mais la C.A.O. peut rendre un trés grand service au concepteur puisqu'elle l'affranchit de manipuler en détail ces informations en lui permettant de traiter des entités d'un niveau plus élevé que celui des rectangles et même des transistors, à savoir celui des blocs de transistors.

La C.A.O. a donc profondément amélioré la méthode de conception de circuits intégrés, mais de plus elle devient le seul moyen pour continuer à faire de la conception de circuits intégrés, si on suit l'évolution dans ce domaine. La courbe III-1 montre qu'il serait inimaginable de réaliser des circuits intégrés sans le recours à la C.A.O..

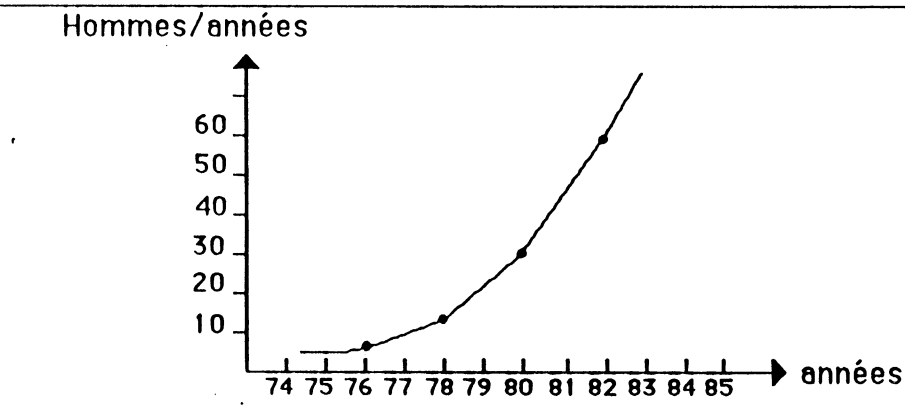


Figure 1 - Coût en hommes/années de réalisation de circuits intégrés sans outils informatiques [OBR 82]

Le futur proche de la conception de circuits intégrés en est à ce qu'on appelle la "Compilation de Silicium" qui est une discipline basée sur un certain nombre d'outils informatiques sous formes de programmes. Ces programmes représentent les modèles des différentes étapes de conception d'un circuit intégré allant de la conception architecturale jusqu'au dessin des masques. La "Compilation de Silicium" est une discipline de C.A.O. résultant de l'expérience accrue des concepteurs de circuits intégrés. La contribution de l'informatique dans ce domaine n'est pas nouvelle. En effet, plusieurs outils informatiques sont apparus au fur et à mesure de la complexité croissante de la conception et, de la validation des circuits intégrés. La conception du circuit COBRA en a profité au sein de l'équipe de recherche en architecture des ordinateurs, où plusieurs outils informatiques ont été créés et continuent à évoluer dans le but de réaliser la conception de circuits indépendamment de leur compléxité (degré d'intégration), dans des délais trés courts et surtout avec une fiabilité croissante grâce à l'emploi de l'intelligence artificielle. Nous présenterons dans ce chapitre les quelques outils qui ont contribué à la conception de COBRA, et ensuite la méthode suivie pour l'emploi des outils dans l'étude et la conception de COBRA, qui est légèrement différente de celle d'un circuit classique de type microprocesseur, et ceci en raison de la propriété d'auto-contrôle (Self-checking) intégrée dans le matériel, à laquelle il faut associer une "méthode dans la méthode" de conception.

2- OUTILS DE C.A.O.

2.1 Généralités

Afin de présenter et justifier l'emploi de tel ou tel outil de C.A.O. au cours de la conception de COBRA, nous allons d'abord montrer les différentes étapes globales de conception de COBRA, qui sont celles de tout circuit de type microprocesseur.

En effet, un microprocesseur est composé de deux parties : partie opérative ou P.O., et partie commande ou P.C.. La partie opérative est un ensemble d'éléments de mémorisations et d'opérateurs reliés entre eux par des bus. La partie commande gère les séquencements des transferts entre les éléments de mémorisations, via les bus, et gère aussi les séquencements d'éxecution des opérations dans les opérateurs. Comment allons nous définir ces deux parties en fonction du cahier des charges ? C'est une description algorithmique et fonctionnelle aboutissant à un jeu d'instructions qui distingue un microprocesseur d'un autre. La figure 2 montre le processus classique de conception d'un microprocesseur. La partie opérative est modélisée par des actions (transfert de registres, activation d'un opérateur), et la partie commande par des séquencements (séquencements de validations des actions de la partie opérative).

A partir de la description fonctionnelle, les deux chemins de la partie opérative et de la partie commande se séparent. Mais il arrive souvent, lors de la conception de la partie opérative de revenir à la description fonctionnelle afin de réduire la charge de la partie opérative en intensifiant le rôle de la partie commande dans la mesure du possible. Ceci est le cas d'une partie commande simple (peu de séquences) et d'une grosse partie opérative où il y aurait besoin de beaucoup d'élements mémoires et de beaucoup d'actions. Cet aspect de rétroaction est valable entre chaque étape et l'étape précédente, à chaque fois qu' une contrainte doit être respectée vis à vis du cahier des charges.

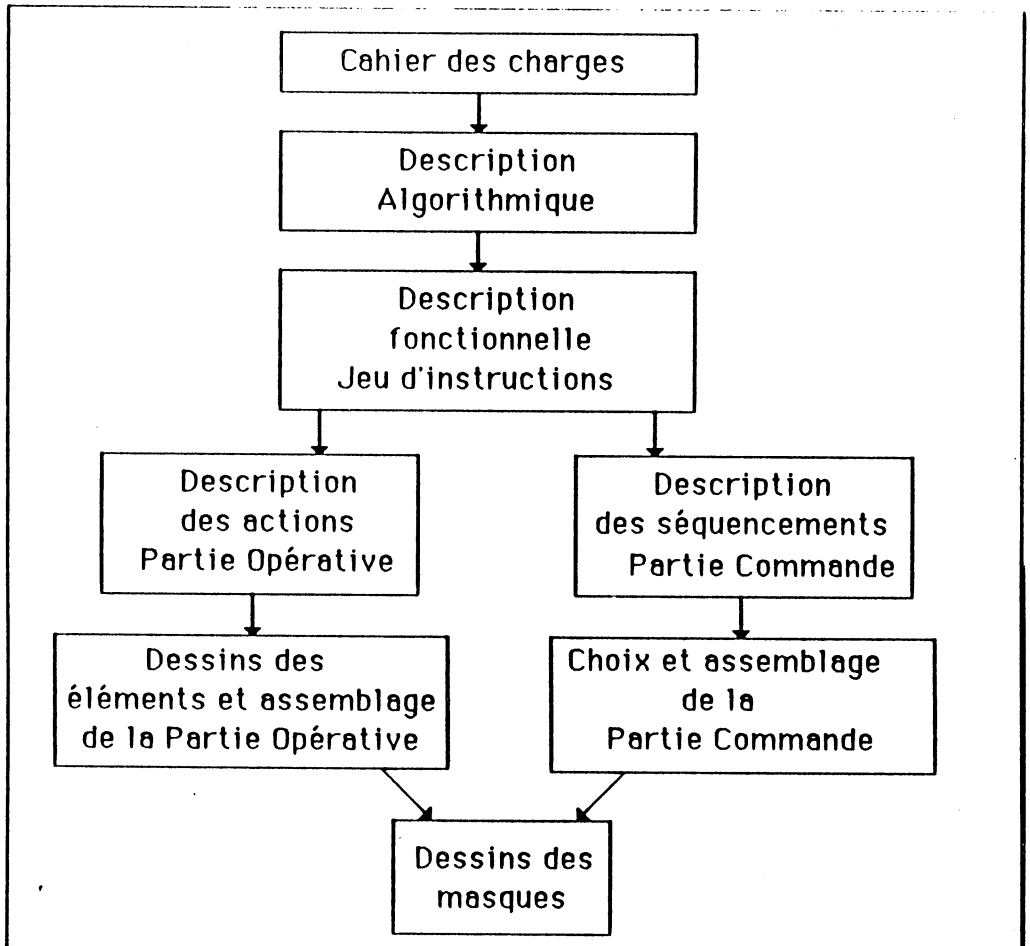


Figure 2 - Processus de conception d'un microprocesseur

La figure 3 montre la place qu'occupe chacun des outils dans le processus de conception d'un microprocesseur. En comparant avec la figure 2, il est possible de déduire le rôle de chacun de ces outils dans le processus de conception. Une description sommaire de chacun de ces outils sera donnée dans la suite de ce chapitre.

Tous ces outils (à l'exception du simulateur électrique SPICE) ont été réalisés dans l'équipe de recherche en architecture des ordinateurs.

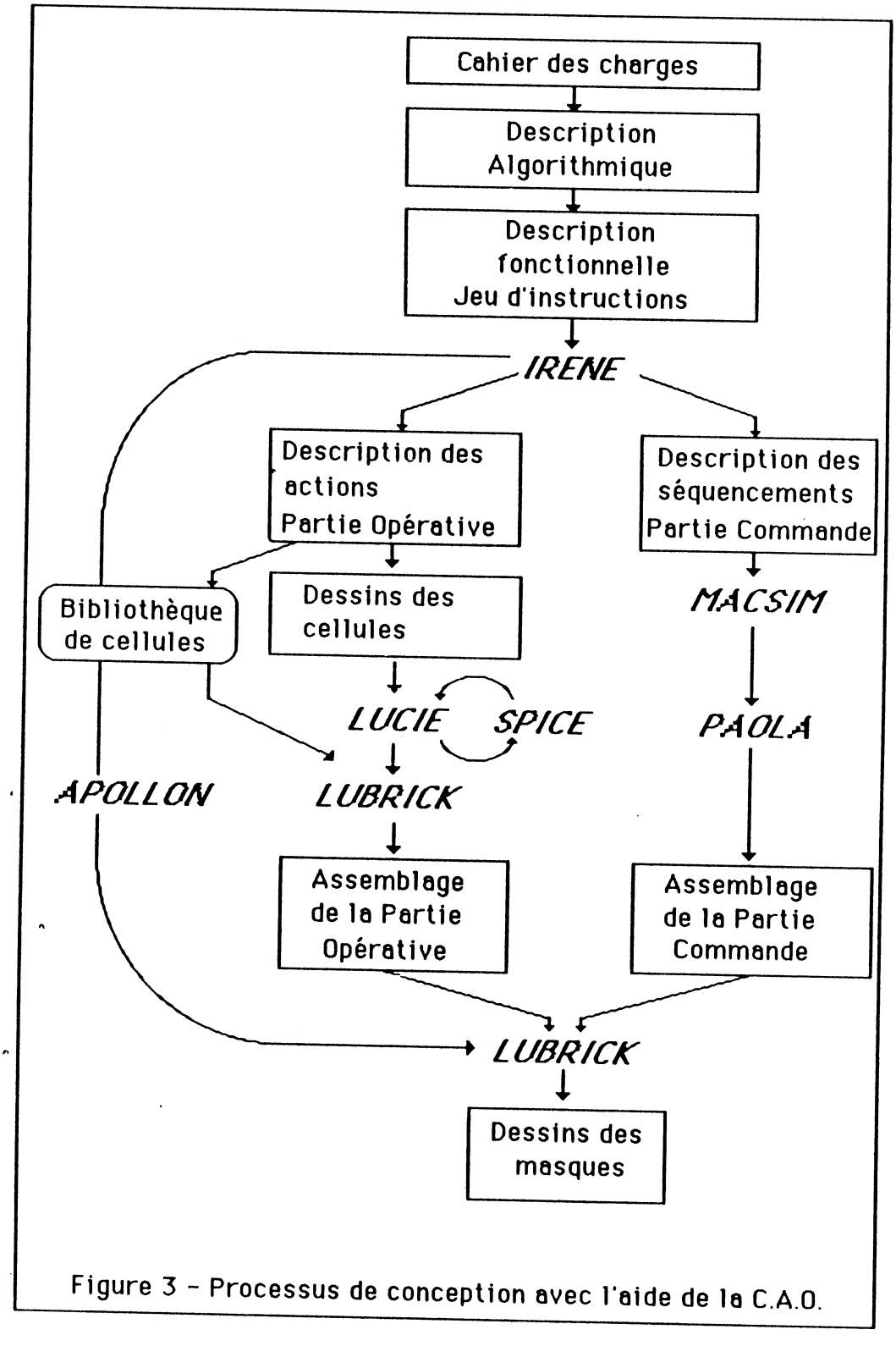


Figure 3 - Processus de conception avec l'aide de la C.A.O.

2.2 IRENE [MAR 85]

La place d'IRENE dans le graphe de la figure 3, montre que cet outil établit une liaison entre le cahier des charges du matériel VLSI à réaliser, et les descriptions de la partie opérative et la partie commande. IRENE offre une synthèse claire et concise de la structure à réaliser, pour les besoins des concepteurs et pour les outils de C.A.O. qui l'utilisent (MACSIM, APOLLON,...).

Ainsi, IRENE permet de spécifier à la fois le comportement et la structure d'une machine en cours de réalisation. L'utilisateur peut modifier sa description IRENE, si les résultats (simulation layout généré) s'avèrent insatisfaisants.

2.2.1 IRENE-C

C'est la description comportementale à partir de laquelle une forme intermédiaire serait extraite. A partir d'une forme intermédiaire on peut :

- soit simuler la machine avec un simulateur fonctionnel [BOU 84]
- soit accéder aux outils de synthèse du matériel pour des générations automatiques de parties opératives (APOLLON) ou de parties contrôles (MACSIM et d'autres outils en cours de développement).

Avant de passer aux formes intermédiaires, il faut compiler la description IRENE-C de la machine avec le compilateur d'IRENE qui détecte les erreurs sémantiques statiques et les erreurs de syntaxe, et génère les fichiers de codes nécessaires à la simulation [MAR 86]. (Une description IRENE de COBRA est donnée en Annexe A en fin de thèse). Une fois la compilation terminée, on passe à la simulation fonctionnelle ou algorithmique de la description compilée de la machine.

La relation entre la description IRENE et la structure matérielle de la machine à réaliser est donnée par les types de variables utilisées. En effet, il existe deux classes de variables dans le langage IRENE : les variables physiques et les variables algorithmiques :

- **a** - les variables physiques sont de deux types : mémorisées comme les registres, les "latches" ou les RAM, qui conservent l'information jusqu'à l'affection suivante par le langage de commandes du simulateur. Cette durée correspond à l'unité de base de temps, et est définie par le concepteur pour chaque circuit. De plus elle est transparente à la description, e.g. : elle peut aller de quelque nanosecondes à quelques millisecondes, l'important étant qu'elle soit non nulle puisqu'elle caractérise la propriété de mémorisation de l'information.

La déclaration des variables mémorisées dans la description IRENE se fait dans les premières lignes, en précédant la liste des éléments de ces types par les expressions 'reg', 'latch' ou 'ram' respectivement. Dans la description de COBRA, seul le type latch est utilisé pour les variables physiques mémorisées, bien que l'on parle de registres dans le chapitre suivant. La différence entre registre et latch dans IRENE réside en la sensibilité au niveau (latch) ou au front (registre) du signal de chargement des valeurs des ports d'entrées dans ces éléments.

L'autre type est celui des variables physiques non mémorisées, comme les bus ou les RAM dynamiques. Leur déclaration se fait aprés l'expression mot-clé 'inst' qui signifie instantané. Ces variables représentent donc des objets matériels où l'information stockée est volatile. Deux attributs peuvent être associés à ces variables lors de leur déclaration :

* un attribut dynamique qui attache à la variable instantanée un entier représentant le nombre d'unités de temps pendant lesquelles la variable, connectée à aucune source, pourra garder sa valeur.

* L'autre attribut est l'attribut de majorité qui est spécifique au type bus, il permet de définir lors de la connexion d'une variable instantanée à plusieurs sources la valeur qu'elle va prendre, en évitant ainsi les conflits possibles entre plusieurs valeurs. Dans la description de COBRA, seul l'attribut dynamique est utilisé pour certains bus et fils d'interconnexion.

- b - Les variables algorithmiques identiques à celles des langages de programmation, en ce sens qu'elle n'ont aucune correspondance matérielle. Elles peuvent représenter des entiers relatifs (signés) ou des booléens.

Il existe aussi dans la description IRENE la possibilité de déclarer des constantes. Elles peuvent être de deux types :

- les constantes matérielles, qui représentent les ROMs et les PLAs. Ces constantes peuvent donc être structurées en tableau. Les dimensions du tableau sont représentées par la longueur du mot d'adressage et la longueur du mot de sortie, données dans la déclaration.

- Les constantes algorithmiques. Elles sont soit explicitées par leur valeur directement dans le texte IRENE, ou, comme pour PASCAL, explicitées dans la partie de déclaration.

Toutes les variables et constantes qui viennent d'être citées sont les opérandes du langage IRENE.

Plusieurs opérateurs orientent le langage IRENE vers la structure de microprocesseurs :

- les opérateurs arithmétiques : mult (multiplication), div (division), mod (modulo) et l'addition +.

- les opérateurs logiques : and, nand, or, nor, exor, exand et not. Ces opérateurs requièrent que les opérandes soient de la même taille.

- les opérateurs relationnels : =, >, <, >>, << et <> respectivement égalité, plus grand, plus petit, beaucoup plus grand, beaucoup plus petit et différence.

- les opérateurs de concaténation, qui ne s'appliquent pas aux variables algorithmiques.

- les opérateurs d'adressage qui permettent la sélection d'un mot dans une variable algorithmique ou physique et/ou la sélection d'un bit ou d'un groupe de bits dans une variable physique.

- les opérateurs d'assignation qui permettent d'assigner une valeur à une variable destination, ce qui détermine une partie gauche (destinataire) et une partie droite (source). Ces opérateurs sont le chargement de registre (<--), l'assignation d'un latch (:=) et la connexion bidirectionnelle (:=:). Seuls les deux derniers sont utilisés pour la description de COBRA.

Il est possible par ailleurs de créer de nouveaux opérateurs en définissant une fonction comme dans un langage de programmation. Il est possible également dans IRENE d'introduire dans la description une ou plusieurs conditions appelées expressions complexes, et qui sont : IF-THEN- ELSE et CASE-OF. Ces expressions peuvent être imbriquées et elles s'appliquent à quatre sortes d'objets :

- les expression simples utilisant les opérateurs arithmétiques et logiques
- les instructions simples, assignant à une variable le résultat d'une expression
- la partie gauche d'une assignation (cas utilisé dans la description de COBRA pour les branchements conditionnels)
- les instructions de séquencement simples qui permettent de changer l'ordre d'éxécution d'une description IRENE-C :
  goto (label), call (routine) et return if (condition vraie)

2.2.2 Structure générale d'une description IRENE-C

Une description comportementale décrit le développement dans le temps d'un ensemble d'éléments matériels ou logiciels. Le comportement décrit l'algorithme d'interprétation de la réalisation décrite. La structure d'une description IRENE est une hiérarchie de MODULES. Un MODULE est une entité décrivant le comportement d'une partie du système, en fonction du type des variables manipulées (algorithmiques ou physiques). Dans le cas des variables physiques le MODULE décrit l'évolution d'un ensemble d'objets matériels dans le temps.

Un MODULE est structuré en cinq parties :

1) L'entête : définissant le nom du module et les variables d'interface (entrée, sortie et bidirectionnelle d'un MODULE). Ces variables servent à créer les liaisons physiques entre les différents MODULES à l'intérieur de leur hiérarchie.

2) La partie déclaration permet de définir les variables et les constantes physiques, les variables et les constantes algorithmiques et les fonctions à utiliser dans le bloc de la description.

3) La spécification du séquencement. Une séquence est une suite d'états dont la durée de chacun est égale au pas élémentaire ou à un multiple de celui-ci. Une séquence décrit un automate avec ses états.

On peut avoir une suite de définitions de séquences à l'intérieur de la définition d'une séquence. Il existe, par ailleurs, deux types de séquencement spécifiés par les mots-clés "rigid" ou "général". Dans le premier cas tous les états de séquencement doivent être exécutés dans l'ordre où ils apparaissent dans la description de séquencement, dans l'autre cas l'exécution des états de séquencement peut être altérée par des instructions de séquencement (branchements conditionnels par exemple) comme les instructions "goto" ou "call".

Dans la déclaration de séquencement de la description de COBRA, il y a trois niveaux de séquencement qui sont :

séquence instruction
séquence cycle
séquence phase

Ce qui signifie que chaque séquence 'instruction' est une suite de séquences 'cycle', chacune d'elles étant une suite de séquences "phase".

La séquence de plus bas niveau : *phase*, est celle qui correspond à l'unité de temps, déterminant ainsi le cycle horloge de la machine. Le type de séquencement *phase* est le type 'rigid', et le nombre de *phases* par *cycle* est de trois. Alors que le type des séquencements *instruction* et *cycle* est le type 'général'. La description de COBRA en annexe A illustre ces propos.

4) La définition hiérarchique de MODULES. La description de COBRA est un MODULE unique.

5) Le bloc de description d'un MODULE est une liste d'étapes séparées par des ';'. Chaque étape commence par au moins une étiquette, faisant référence à un état de séquencement d'automate reconnu ou défini dans ce MODULE.

Une étape contient plusieurs actions (en terme d'actions de partie opérative) en parallèle séparées par des '/'. Pour COBRA, une étape correspond à une séquence 'phase' qui contient plusieurs actions opératives (affectations) ou des actions de séquencement ou d'activation.

Nous donnons ici un exemple de description d'une instruction de COBRA dans le but de faciliter la présentation du simulateur mais aussi pour MACSIM qui utilise cette description pour générer les commandes de la partie opérative. C'est l'exemple de l'instruction CIM (Copy Inputs to MSB). On remarque les trois niveaux de séquencements : instruction : CIM, cycle : C1et C2 et phase : PHI1, PHI2 et PHI3.

Toutes les instructions sont décrites de la même façon. Le nombre de cycles varie de 2 à 7, mais il peut atteindre 12 pour les instructions exécutées lors d'une interruption (non accessibles aux utilisateurs).

Exemple

```
        ! CODOP1 = 01101111 ; !

$ CIM :   $C1 :     $ PHI 1 : prchg := '1;
                    $ PHI 2 : IB := PCL/HB := MSB7 ;
                    $ PHI 3 : EB := IB/INC 1 := PCL/ ACCA := HB/IINC 1 :='1;

          $C2 :     $ PHI 1 : prchg := '1;
                    $ PHI 2 : HB := SINC1/OE:='1/IB := EB;
                    $PHI 3 : IR := IB/ PCL := HB/ bitv := '0/
                    bitn := ALU 2<6> / IT := '1;
```

Dans le niveau de séquencement le plus bas (phase PHIi), on retrouve les commandes à exécuter. Celles qui sont en parallèle sont séparées par des '/'.

Le caractère "!" est le signe limitant un commentaire pour IRENE, alors que c'est un signe d'attribut pour MACSIM. Les caractères "(*" et "*)" limitent le commentaire pour MACSIM. Ceci est utilisé par l'analyse de MACSIM de la description IRENE, pour la génération de la partie commande.

2.2.3 Le simulateur

Le simulateur est l'interface de la description IRENE-C compilée avec l'utilisateur. En effet, l'utilisateur a besoin de savoir si sa description correspond bien au circuit à réaliser. Ainsi, il existe un langage de commande interactif appelé ILIS (Interactive Language for IRENE Simulation) [MAR 84], qui comprend plusieurs sortes d'actions, permettant de suivre dans le temps l'évolution d'une machine à partir de sa description IRENE-C. Les résultats de la simulation, obtenus étape par étape, renseignent le concepteur sur l'éventuelle nécessité de reprendre la description de la machine.

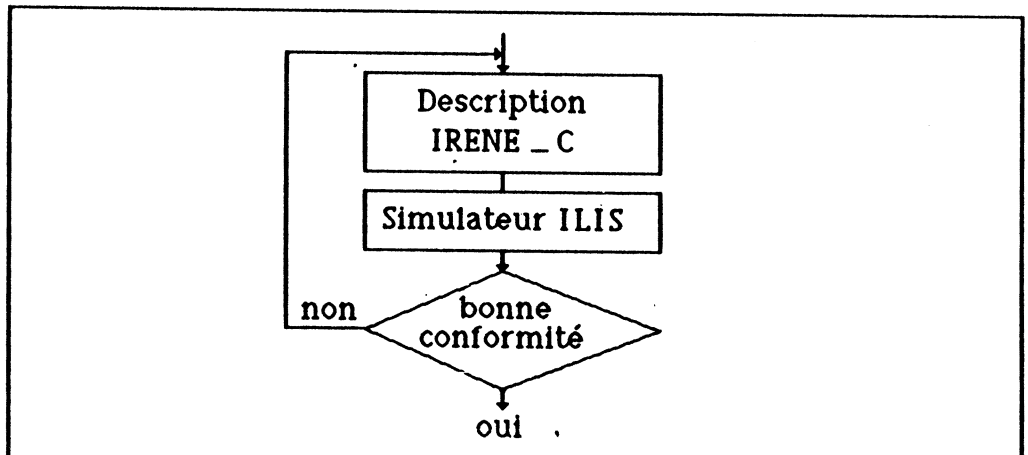


Figure 4 - Validation de la conception par simulation

Le langage de commande ILIS permet les actions suivantes :

- assignation d'une valeur à une variable (assign, Keep)
- la mise à 0,1 ou X (valeur indéfinie) d'une variable (set, reset, undef).
- le forçage d'une variable à une valeur (0,1 ou X) jusqu'à l'occurence d'une commande contraire (init 0,1 ou X)

- blocage d'une (ou plusieurs) variable(s) à une valeur tant qu'il n'y a pas de commande de déblocage (lock, release)
- la lecture de la valeur en cours d'une variable et son impression sur l'écran ou dans un fichier (print)
- l'impression de toutes les variables qui ont changé de valeur durant la dernière évolution du modèle (prmv. prav)
- l'avance d'un pas dans la simulation (step)
- l'affichage du contenu du compteur des pas de simulation (date)

Il est possible, par ailleurs, de sauvegarder (savctx) ou de restaurer (res ctx) un contexte du modèle simulé, à un état précis de la simulation.

De plus, il existe quelques primitives supplémentaires du langage de commandes inspirées du PASCAL, et qui contribuent au contrôle de l'éxécution de la simulation : (write, writeln, read, readln, repeat until, while do, if then else). Il est possible d'introduire une commande 'stop' à l'intérieur d'une boucle (comme repeat, ...while) pour procéder par pas.

Enfin, pour sortir d'une simulation, il faut utiliser la commande 'bye'.

Le circuit COBRA a été simulé à plusieurs reprises lors des premières descriptions. L'annexe B donne un aperçu de la simulation d'une partie de la description IRENE de COBRA.

IRENE a été réalisé dans l'équipe de recherche en architecture des ordinateurs. Les différents programmes qui le composent (compilateur, simulateur et langage de commandes) ont été écrits en PASCAL, et sont implantés sur VAX/VMS. La description de COBRA par IRENE, a contribué avec plusieurs autres projets (description du M6800, MC 68000 et d'un circuit de synthèse de la parole, etc...) à la validation et à l'adaptation des programmes d'IRENE (à différents niveaux) aux problèmes posés aux concepteurs.

2-3 LUCIE

La place attribuée à LUCIE dans le graphe de la figure 3, montre que c'est un outil d'aide à la conception et au dessin des éléments de circuits intégrés.

Le système LUCIE est construit autour d'un langage textuel de description hiérarchisée de masques de circuits intégrés : le langage LUCIE (Langage Universitaire de Conception de circuit Intégrés pour l'Enseignement).

Les autres logiciels du système LUCIE sont :

- **Traducteurs** : logiciel de passage d'une description en langage LUCIE, à une forme intermédiaire, interprétable par les logiciels intéractifs, comme l'éditeur graphique LUCIE.

- **Editeur** : logiciel interactif d'édition graphique permettant l'affichage de tout ou d'une portion d'un circuit sur un écran graphique. Il permet en outre diverses opérations de correction et d'extraction de paramètres du dessin, ainsi que le tracé du dessin sur table traçante (électrostatique, type BENSON).

- **Digitaliseur** (ou Digiteck) : logiciel d'entrée d'un dessin en le digitalisant. Ce logiciel renvoie une description en langage LUCIE.

La figure 5 donne une vue d'ensemble du système LUCIE.

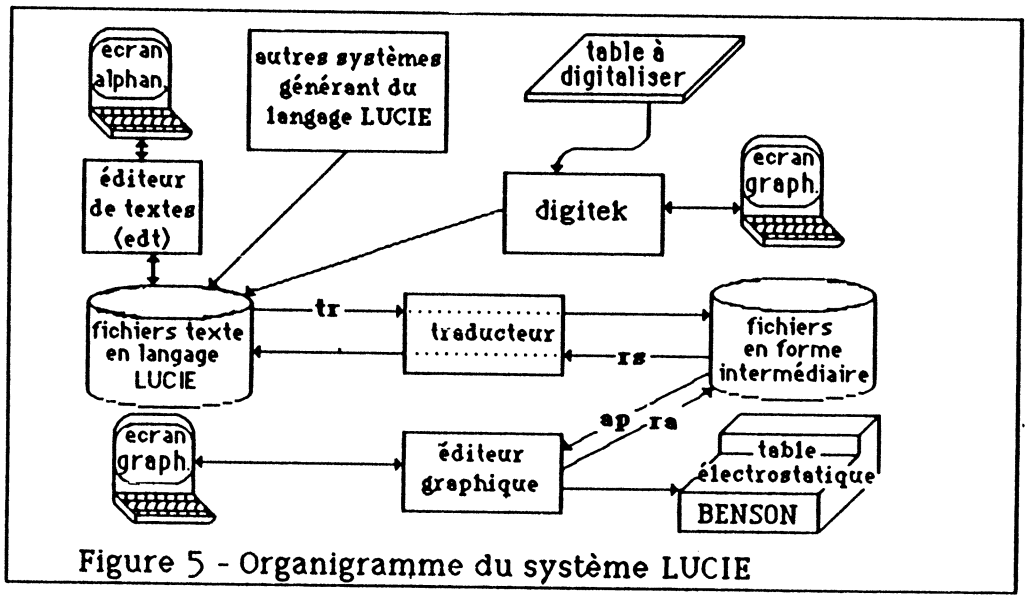


Figure 5 - Organigramme du système LUCIE

2.3.1 Description LUCIE

Les descriptions LUCIE sont hiérarchisées. L'élément de base de ces descriptions LUCIE est le rectangle.

Il est décrit par sa position, sa taille et le niveau de masque (niveau technologique) auquel il appartient. Le rectangle est décrit par :

rec (x, y, dx, dy, n)

- x et y sont les coordonnées du coin inférieur gauche du rectangle
- dx et dy sont les dimensions du rectangle
- n est le niveau de masque du rectangle (polysilicium, métal, diffusion,...) en fonction de la technologie utilisée.

Remarque :

Pour le circuit COBRA, comme pour les autres circuits réalisés dans le cadre du projet national CMP (Circuits Multi Projets), les niveaux technologiques sont normalisés ; par exemple le CMP 85 :

pour le NMOS : md, mc, mp, mm, mg, mi, me
pour le CMOS : md, mb, mc, mp, mm, ms, me, mv, mh, mg

mb : diffusion P
mc : point de contact
md : diffusion N
me : précontact (ou contact enterré)
mg : passivation
mh : deuxième niveau métal
mi : implantation ionique
mm : niveau métal
mp : polysilicium
ms : caisson P
mv : contact entre mh et mm.

Il existe un autre élément hiérarchique dans toute description LUCIE qui est la figure : FIG. Une figure est un ensemble nommé d'éléments (rectangles, figures) placés par des opérateurs (répétition, translation, symétrie, rotation).

Une figure déclarée peut être utilisée comme un élément de bibliothèque, elle est alors appelée par la commande "figext" (figure extérieure). L'ensemble des commandes et la syntaxe de la description LUCIE sont explicités dans [LUC 85]. L'annexe C montre un exemple d'une description d'un bloc de la partie opérative.

Les fichiers de description textuelle LUCIE des masques des circuits intégrés sont envoyés au projet national CMP [CMP 85]. Les fichiers LUCIE sont ensuite traduits dans la forme intermédiaire. Puis les circuits ainsi décrits seront assemblés en macrocircuits. Un macrocircuit est un carré de dimensions fixes dans lequel seront placés plusieurs circuits à réaliser par le CMP : c'est justement l'idée de base du projet CMP qui est de fabriquer collectivement des circuits différents, le coût de fabrication étant ainsi partagé.

Pour chaque macrocircuit complété il sera généré une bande en format Calma GDS2. La génération des commandes pour le masqueur électronique sera effectuée sur le système Calma permettant enfin la fabrication des masques des circuits. La gestion du projet CMP est faite par l'équipe de recherche en architecture des ordinateurs.

Le système LUCIE a été développé par l'équipe de recherche en architecture des ordinateurs. Ses différents programmes (traducteur, éditeur graphique et digitaliseur) sont écrits en Fortran77. Il est implanté sur VAX/VMS, sur SM90/UNIX et sur HB68/MULTICS.

2-4 MACSIM

MACSIM (pour MAChines SIMulation) est un outil de C.A.O. à double but. Le premier est, comme l'indique son nom, de décrire une machine de type unité centrale pour en réaliser sa simulation à un niveau de transfert de registres, et ensuite extraire de la description même deux caractéristiques : le chemin de données et un modèle de partie commande. C'est dans ce dernier cas que MACSIM est utilisé pour le circuit COBRA (voir figure 3).

Le deuxième but est de préciser le cahier des charges du système à réaliser dans le cadre du système global d'IRENE et autour du langage de description IRENE.

Le modèle de la partie commande synthétisée par MACSIM est celui de la partie commande à un niveau d'interprétation et un générateur d'instants (ou générateur de temps). le principe de cette solution, décrit plus loin, est basé sur une structure multi-PLAs associée à une génération de séquences par le générateur de temps. Le choix d'une telle solution est justifié par plusieurs raisons :

- les PLAs sont des structures régulières. Par rapport aux solutions de la logique anarchique (ou aléatoire), les avantages sont : la conception facile grâce à la C.A.O., le taux d'erreurs de conception nettement diminué, et il est plus facile de "relire" une structure régulière au cas ou il est nécessaire d'apporter des modifications.

- Cette solution a de bonnes performances globales par rapport aux autres solutions existantes. Une étude complète a été réalisé par M. OBREBSKA [OBR 82] pour comparer les solutions de choix de partie opérative.

- Les bonnes performances de cette solution ont motivé son choix pour générer automatiquement les PLAs par MACSIM.

- Les dessins de masques des PLAs (fichiers LUCIE traduits) sont ainsi générés automatiquement par un autre outil de C.A.O. : PAOLA.

Par ailleurs il n'est pas nécessaire de créer une entrée spéciale pour MACSIM, puisqu'il peut recevoir en entrée la description source d'IRENE-C (à quelques petites modifications près.)

2.4.1 Principe de la partie commande avec générateur de temps

Le rôle d'une partie commande est de générer à partir des instructions et de leurs opérandes un ensemble de commandes pour la partie opérative. A chaque instruction correspond un code opération qui transite vers la partie commande via le registre d'instructions (RI ou IR). Ce registre envoie le code opération à un opérateur de décodage qui détermine à quelle famille d'instructions appartient l'instruction correspondante. Pour la partie commande avec générateur de temps, l'opérateur de décodage est un "PLA de propriétés" (ou aussi des attributs), puisqu'il extrait du code opération de l'instruction décodée un ensemble d'attributs statiques (propriétés).

Les codes opérations de toutes les instructions sont écrits de sorte que les propriétés soient scindées en deux parties. La première partie contient les codages des instants d'une famille d'instructions, l'autre partie est le code distinguant une instruction d'une autre dans cette famille. Ces propriétés étant extraites dans le PLA de propriétés, les sorties de ce PLA vont d'un côté vers le générateur de temps (réalisé en logique anarchique) pour décoder les instants, et d'un autre côté vers un deuxième PLA nommé le PLA de commandes, qui a en ses sorties les commandes destinées à la partie opérative. Ce PLA de commandes reçoit aussi en entrée, les sorties du générateur de temps. La figure 6 illustre le principe d'une telle partie commande.

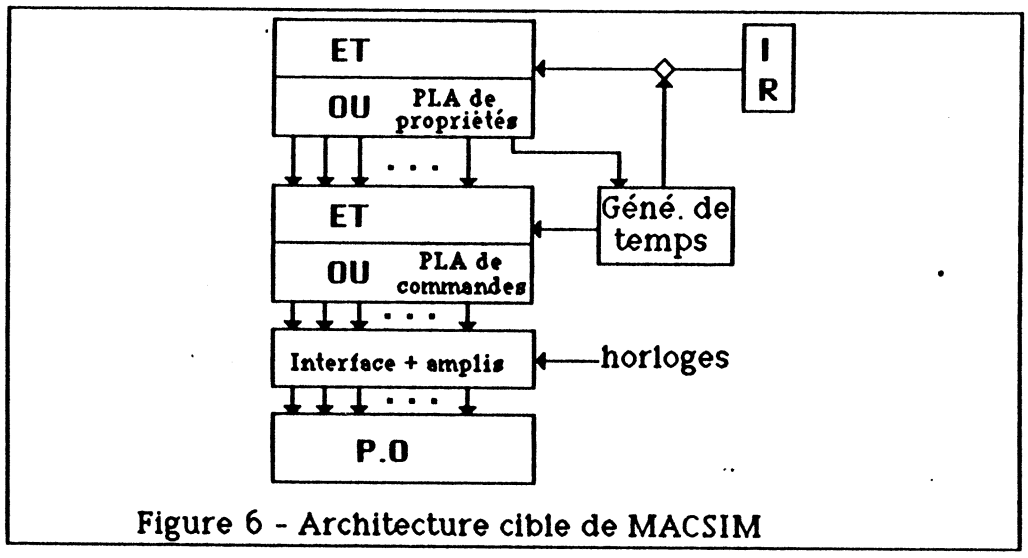


Figure 6 - Architecture cible de MACSIM

Le PLA de commandes réalise la somme de produits des propriétés statiques du code opération décodé, avec les instants séquencés par le générateur de temps (à partir du même code opération décodé). Ainsi, une commande est active lorsque la somme de produits d'un instant par une propriété est vraie, par exemple : on considère les instants Ti et les propriétés Pj. La commande IB:=ACCA (bus IB lit le contenu de l'accumulateur ACCA) équivaut à :

(P1 et T1) ou (P3 et T2) ou (P6 et T12)
Cette commande est donc activée :

à l'instant	T1 pour les instructions qui ont la propriété		P1
"	T2	"	P3
"	T12	"	P6

Remarque :

- Une instruction a généralement plusieurs propriétés.
- Dans COBRA le nombre d'instants est plafonné à 16, ce qui permet de les coder sur 4bits.

2.4.2 Fonctionnement de MACSIM

MACSIM est un ensemble de trois programmes correspondant à trois phases de traitement de la description IRENE. Ces trois phases sont : l'analyse, l'expansion et la synthèse. Elles correspondent à trois programmes distincts, mais qui communiquent entre eux par des fichiers intermédiaires.

Le premier programme, ANALYSE, reçoit en entrée le fichier texte de description comportementale IRENE-C, auquel certaines modifications ont été apportées. Par exemple, le code opération de chaque instruction décrite, est présenté sous forme de commentaire pour IRENE, mais pour MACSIM il correspond au contenu du registre d'instruction (voir exemple 2-2-2). Ainsi il faut rajouter à la fin de la description de chaque instruction une commande supplémentaire permettant le retour vers un début d'analyse d'une autre instruction par MACSIM : c'est la commande "goto" <début>. Cette phase d'analyse permet de dresser une table des commandes activées dans chaque instruction en indiquant dans quelle phase et quel cycle, chacune est activée (dans la description IRENE de COBRA, chaque instruction a un nombre variable de cycles allant de 2 à 12, et chaque cycle a trois phases). Cette table est utilisée par le deuxième programme de MACSIM, EXPANSION, qui effectue l'expansion des codes opérations de chaque famille d'instructions : le codage des commandes et leurs instants d'activation (dans les tables du programme ANALYSE) est étendu à une représentation sur 256 bits (pour un code opération de 8 bits), d'où le nom expansion.

Cette représentation ensembliste va permettre de réaliser des unions entre toutes les familles d'instructions qui doivent générer la même commande, et pour toutes les instructions, un autre fichier est généré, et sera exploité par le dernier programme de MACSIM : SYNTHESE. Cette dernière phase associe des sommes de produits d'instants par des propriétés. Les "instants" sont les instants d'activation des commandes, et les "propriétés" sont extraites des étapes précédentes, et caractérisent les familles d'instructions et les instructions au sein des familles d'instructions.

L'explication complète de MACSIM est donnée par son auteur, J.P. SCHOELLKOPF dans [SCH 85].

MACSIM fournit finalement, quatre tableaux sous formes de matrices, représentant les contenus de deux PLAs :

- le PLA de propriétés (matrice ET et matrice OU)
- le PLA de commandes (idem)

A ces matrices correspondent les fichiers de sorties de MACSIM directement exploitables par le programme PAOLA qui en déduit les descriptions des masques. MACSIM a été réalisé dans l'équipe de recherche en architecture des ordinateurs. la version qui a servi pour COBRA, est écrite en PASCAL, et elle est implantée sur VAX/VMS.

2.5 PAOLA

PAOLA (pour Pla Automatic Optimization LAyout) est un outil de C.A.O. destiné à l'optimisation topologique et à la génération automatique de la description des masques de PLAs complexes [CHU 84].

Un PLA est l'implantation sous forme canonique (somme de produits) d'une ou plusieurs fonctions logiques. D'où la description possible de ces fonctions selon deux matrices : ET et OU. En technologie MOS, le PLA est implanté sous forme NOR-NOR, [MEA 80].

Les avantages des PLAs par rapport à la logique aléatoire pour la réalisation des fonctions logiques complexes sont :

- réduction du temps de conception
- génération et dessin automatiques possibles grâce à leur régularité
- diminution du risque d'erreurs, et éventuellement correction facile de ces erreurs.

Mais l'inconvénient des PLAs est la perte importante de surface par rapport à la logique aléatoire, pour les mêmes fonctions à réaliser. Pour en réduire l'effet, il existe deux méthodes complémentaires :

- la minimisation logique qui consiste à trouver la forme logique minimale pour réaliser une fonction logique donnée.
- l'optimisation topologique réduisant directement la surface de silicium occupée par le PLA en agissant de plusieurs façons : modifications des emplacements des connecteurs entrées/sorties, découpage des matrices, placement latéral des connecteurs, triangularisation, linéarisation de la matrice OU et optimisation en lignes brisées.

PAOLA réalise l'optimisation logique avec la méthode des lignes brisées. L'optimisation se fait en quatre phases :

- l'ordonnancement des monômes par diagonalisations des matrices
- la duplication des monômes qui permet de modifer la forme du PLA par allongement de celui-ci
- le compactage des matrices : étape correspondant à l'optimisation proprement dite. Son but est de regrouper le plus de segments possible dans un minimum de colonnes.
- les traitements complémentaires qui préparent les matrices pour la génération des masques et pour le tracé des connexions internes.

La figure 7 montre la structure du système PAOLA.

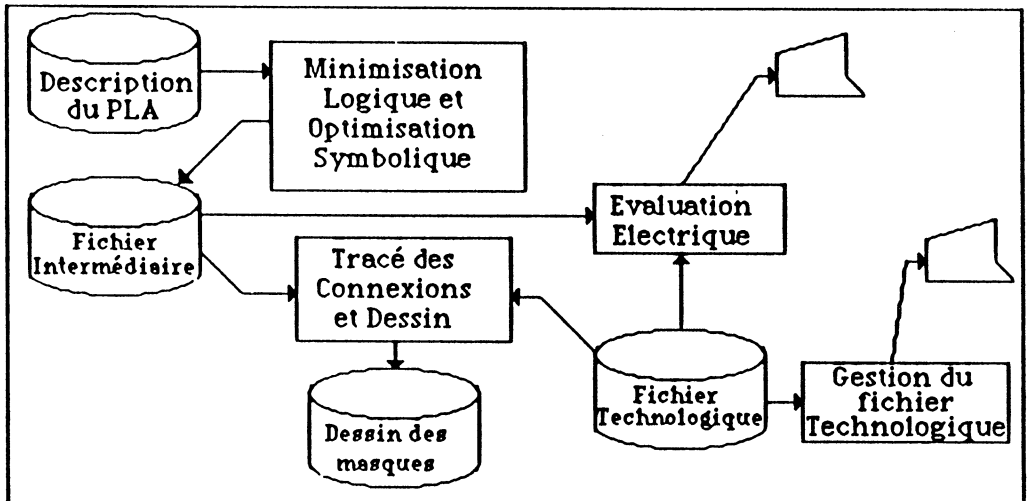


Figure 7 - Structure du système PAOLA [PER 85]

Le système PAOLA est organisé en plusieurs modules. Chacun de ces modules est composé de plusieurs programmes spécifiques, et l'optimisation topologique est un des modules de PAOLA. Mais pour le circuit COBRA, c'est principalement le module de tracé des connexions et de dessins des masques des PLAs qui est utilisé. Ce module accepte en entrée les fichiers de sortie de MACSIM directement et sans aucune modification. Les descriptions LUCIE des PLAs sont obtenues en sortie de ce module.

PAOLA est écrit en langage PASCAL, et a été réalisé dans l'équipe de recherche en architecture des ordinateurs. Il est implanté sur plusieurs machine dont le VAX/VMS.

2.6 SPICE

C'est un programme de simulation électrique. Son abréviation signifie "Simulation Program with Integrated Circuit Emphasis". Il permet la simulation du fonctionnement de circuits pour les analyser : en régime continu (non linéaire) en donnant une fonction de transfert du circuit en continu, en régime transitoire (non linéaire) en déterminant la réponse du circuit simulé en fonction du temps et dans un intervalle spécifié, et également en régime alternatif (linéaire) en permettant de tracer la réponse en fréquence du circuit simulé. Un circuit peut contenir des résistances, des capacités, des inductions, des sources de tension et de courant ainsi que quatre dispositifs à semi-conducteurs : diodes, transistors bipolaires, MOS et JFET. Tous ces éléments se trouvent sous forme de modèles dans le programme et peuvent être paramètrés. De plus, les analyses peuvent être faites à différentes températures.

Les modèles de simulation SPICE sont établis en fonction de la technologie avec laquelle sera fabriqué le circuit à simuler. Pour COBRA, ce sont les modèles de simulation SPICE établis pour le CMP-NMOS (Circuit Multi Projets NMOS).

L'annexe D montre les modèles utilisés au CMP 1985 ainsi que l'exemple de simulation d'un bloc de transistors (partie d'un circuit).

La simulation électrique en général a un rôle important dans la conception. Toutefois son emploi systématique peut devenir lourd et coûteux. Alors plutôt que de calculer les caractéristiques électriques des cellules d'un circuit, en utilisant un simulateur, il faut avoir recours à une standardisation des cellules (bibliothèque). Ceci permet une simplification du calcul avec l'assurance de rester dans un domaine de fonctionnement électrique correct.

Des ensembles de portes précalculées et simulées servent comme éléments de base d'une bibliothèque de cellules logiques. Toutefois, la place de SPICE dans la figure 3 autour de LUCIE est justifiée par la nécessité de simuler à nouveau chaque cellule nouvelle décrite avec le langage LUCIE plusieurs fois jusqu'à l'obtention de la conformité électrique, en modifiant à chaque simulation : les longueurs et/ou largeurs des grilles des transistors, ou bien en déplaçant certaines interconnexions afin de supprimer des capacités parasites, etc. Il faut également tenir compte dans une simulation des problèmes d'entrées/sorties des blocs en adaptant électriquement la sortie d'un circuit vis à vis de l'ensemble des entrées des circuits qu'il doit attaquer (ensemble de capacités équivalentes à charger).

SPICE est un programme qui peut simuler des circuits comprenant au maximum une centaine de composants actifs (transistors MOS par exemple). Il donne en résultat 16 variables de sortie présentées soit sous forme de tableau, soit sous forme de courbes. SPICE a été développé à l'université de Berkeley-Californie, mais plusieurs versions continuent à être développées afin de l'adapter à l'évolution et la complexité rapides des circuits intégrés. La version de SPICE utilisée pour la simulation des éléments de COBRA est disponible sur VAX/VMS.

2.7 APOLLON

Le système APOLLON est un compilateur de parties opératives. Il constitue la première partie d'un compilateur de silicium SYCO [JER 86]. Ce qui caractérise APOLLON c'est qu'il utilise un modèle standard (architecture cible) qui est celui de la partie opérative du MC 68000.

A partir d'une description comportementale, il est possible d'obtenir une liste de registres, opérateurs et signaux de contrôle. Tel que présentée dans la figure 3, la description comportementale est celle d'IRENE. Par ailleurs, la liste d'éléments obtenue à partir de cette description doit être structurée en une description d'actions opératives. Ces actions opératives peuvent être soit des transferts de registres, soit des opérations logiques ou arithmétiques. Cette description d'actions opératives est indépendante de la technologie, de même que la structure du système APOLLON qui reçoit en entrée cette description et génère une partie opérative correspondante.

2-7-1 Modèle de partie opérative d'APOLLON

C'est un modèle standard constitué par un ensemble de sous-parties opératives qui exécutent les actions en parallèle. Exemple : une sous-partie opérative pour les données et une autre pour les adresses. L'architecture de chaque sous-partie opérative est organisée autour d'une structure à double bus bifilaire communiquant avec les autres sous-parties opératives par des interrupteurs de liaison placés sur les bus.

Le modèle topologique est celui de la partie opérative du MC68000, il a été choisi pour sa structure élémentaire basée sur le principe de "bit slice" (tranche d'un bit). Cette tranche d'un bit a une hauteur fixe définie pour une technologie donnée. L'assemblage vertical de n blocs constitue un bloc fonctionnel de n bits.

Une sous-partie opérative est construite en alignant des blocs fonctionnels (registres, opérateurs,...). La partie opérative est, enfin l'assemblage par alignement de toutes les sous-parties opératives ainsi construites. D'où l'intérêt de ce modèle standard qui est facile à formaliser et à implanter. La figure 8 montre la structure de bus pour une tranche d'un bit qu'utilise APOLLON. Cette structure est également celle de la partie opérative de COBRA.

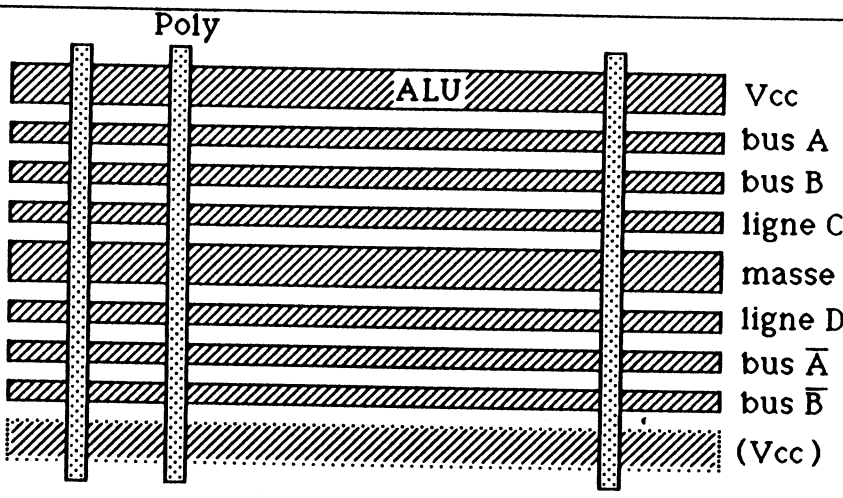


Figure 8 - Structure de bus utilisé par APOLLON pour un bit slice N-MOS

La caractéristique de cette structure est que les bus en aluminium sont horizontaux et les lignes de commandes en polysilicium sont verticales, ce qui leur permet de traverser par transparence autant de cellules qu'il est nécessaire.

APOLLON fournit une forme de partie opérative globale (dessin fonctionnel), ainsi que la description du fonctionnement interne en détaillant l'exécution de chaque étape (ou groupe d'actions opératives). APOLLON fournit le dessin des masques de la partie opérative (en utilisant LUCIE et LUBRICK pour l'assemblage). Mais avant de demander la génération du dessin des masques par APOLLON, il est nécessaire de faire un certain nombre de choix :

- nombre de bits
- technologie
- bibliothèque de cellules à utiliser pour le dessin des masques.

La bibliothèque de cellules est constituée actuellement d'une cinquantaine de cellules en technologie NMOS [GAL 84]. Toutes ces cellules ont été conçues pour être assemblées directement en utilisant LUBRICK. Elle sont construites selon la structure de bus de la figure 8.

La bibliothèque contient principalement :

-des éléments de mémorisation

-des opérateurs arithmétiques et logiques : UAL complète à quatre étages possibles.

* demi-additionneur
* retenue
* indicateurs (Flags)
* décaleurs (shifters)

-le reste de l'infrastructure électrique :

* connecteurs (bus-bus, bus-registre, tampons-UAL)
* éléments de précharge
* peignes d'alimentation
* amplificateurs.

2.7.2 Applications pour COBRA

APOLLON a été utilisé dans le projet de COBRA principalement dans le but de validation. En effet, la partie opérative de COBRA avait une certaine structure à une étape de la conception. Il est évident que si une autre personne avait pour tâche de trouver une structure de partie opérative à partir du même cahier des charges, elle aurait trouvé une structure différente de celle proposée dans la présente étude. Ce fut donc le rôle d'APOLLON de générer à partir de la même description IRENE une solution parallèle qui a été comparée avec la solution manuelle classique, et la solution définitive étant un mélange des deux, en conservant de chacune ses avantages :

- structure des bus standard (double bus bifilaire)

- conformité à la description et au jeu d'instructions en conservant un parallèlisme maximal dans les exécutions des commandes.

- conservation pour certains blocs de leurs structures non classiques : la spécificité de COBRA fait que certains des éléments de la partie opérative sont eux mêmes spécifiques et ne correspondent pas à des éléments prévus pour APOLLON, car ils n'existent pas dans la bibliothèque de cellules à partir de laquelle il construit les parties opératives.

Par ailleurs, le séquencement des étapes associé à toute partie opérative construite par APOLLON (modèle du MC68000) est en quatre phases T1, T2, T3 et T4, tandis que le séquencement des étapes dans COBRA est en trois phases PHi1, PHi2 et PHi3 (voir figure 9).

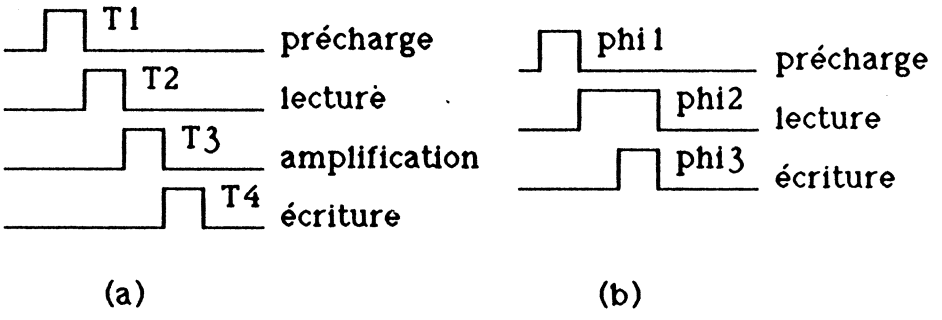


Figure 9 - Séquencement des étapes dans une partie opérative
(a) générée par APOLLON
(b) dans COBRA

Certaines des fonctions d'assemblage de blocs utilisées par APOLLON ont été également utilisées pour générer les dessins de masques de quelques blocs de la partie opérative de COBRA conformes au choix de la structure définitive (l'unité arithmétique et logique, l'incrémenteur, et le registre à décalage).

APOLLON est écrit en LISP. Il a été réalisé dans l'équipe de recherche en architecture des ordinateurs [JAM 85]. Il est implanté sur SM90/UNIX.

2.8 LUBRICK

Le système LUBRICK est un ensemble de programmes qui permettent l'assemblage de cellules décrites avec LUCIE (le nom de LUBRICK provient de LUCIE briques). A chaque cellule ou bloc décrit avec le langage LUCIE, il faut associer un fichier de description LUBRICK, pour lui permettre d'être assemblé automatiquement. En effet, il est possible d'assembler entièrement tous les blocs d'un circuit avec le langage LUCIE, mais ce serait un travail laborieux et long, et surtout une source d'erreurs de manipulation. LUBRICK est basé sur un principe simple et adapté à la structure de "bit slice". En effet, l'assemblage de plusieurs cellules entre elles doit assurer les interconnexions entre les cellules, ainsi que le placement des cellules de façon à respecter la garde qui est la distance minimale entre les niveaux technologiques (des deux cellules) rapprochés par l'assemblage. Pour cela, LUBRICK s'est basé sur le principe de cellule LUCIE définie également par ses "frontières" et ses "connecteurs" probables. Ces informations sur les frontières et les connecteurs d'une cellule sont contenues dans le fichier de description LUBRICK déterminé par le concepteur selon ses choix (un exemple de la description d'un bloc de COBRA avec LUBRICK est donné dans l'annexe E).

Ce fichier de description LUBRICK doit contenir toutes les informations sur l'environnement topologique de la cellule. d'où la notion de frontière (boundary) : nord, est, sud ou ouest.

De même pour les sens des connecteurs : nord, est, sud ou ouest

2.8.1 Syntaxe de description LUBRICK

Soit une frontière "sud" décrite par les trois lignes suivantes :

bs	2		(signifie frontière sud décrite en 2 lignes)	
0	0	15	3	0
5	15	10	1	0
offset	position	largeur	niveau	type (pour les connecteurs seulement)

Et soient les deux connecteurs "ouest" et "est" décrits par :

cw	1		(signifie : un connecteur ouest)	
0	12	2	3	7
ce	1			
8	10	3	4	11
offset	position	largeur	niveau	type

Ces informations sont illustrées dans la figure 10.

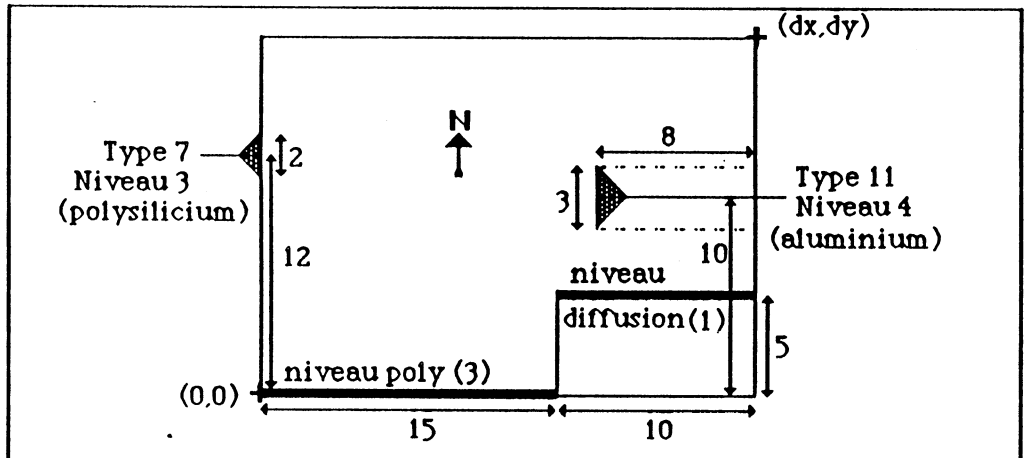


Figure 10 - Exemple d'informations pour une description LUBRICK

Les paramètres offset, position et largeur fixent les coordonnées et les dimensions des frontières et des connecteurs. Le paramètre "niveau" définit le niveau technologique correspondant : pour le NMOS les niveaux diffusion, contact, polysilicium, aluminium et précontact sont codés respectivement avec 1, 2, 3, 4 et 5.

Le paramètre "type" (pour les connecteurs seulement) caractérise un ou plusieurs connecteurs. Ainsi, un connecteur sud de type 8 ne peut se connecter qu'avec un connecteur de type 8.

La description d'une cellule sous la forme textuelle ainsi définie, est facile à modifier. Elle peut être :

- entrée à la main avec un éditeur de texte,
- entrée à la main avec le digitaliseur, comme pour une figure LUCIE,
- générée par un programme LUBRICK comme description de la cellule résultante de l'assemblage demandé.

La syntaxe d'une description LUBRICK est la suivante :

FIG **< nom de figure >**
dx **dy** (dimensions de la figure)
cn **< nombre de connecteurs >** (idem pour ce, cs et cw)
< offset position largeur niveau type > (n fois)
bn **< nombre de frontières >** (idem pour be, bs et bw)
< offset position largeur niveau type > (n fois)
FFIG

EXEMPLE :

```
FIG     RGC
28      54
cs      3
        9       5       2       3       20
        0       11      2       1       0
        9       17      2       3       5
cn      3
        0       5       2       3       20
        0       11      2       1       0
        0       17      2       3       5
be      1
        0       0       28      1       0
bw      1
        0       0       28      3       0
FFIG
```

2.8.2 GESTION DES DESCRIPTIONS LUBRICK

Le système LUBRICK est basé sur le langage PASCAL. Il utilise des fonctions et des procèdures prédéfinies pouvant être appelées depuis n'importe quel programme d'assemblage.

Les descriptions LUBRICK des circuits à assembler sont appelées par LUBRICK et gérées par ses fonctions et procédures selon le programme d'assemblage écrit en PASCAL.

De plus, il est possible de créer n'importe quelle fonction ou procédure PASCAL, et de la considérer ensuite comme prédéfinie.

Les fonctions et procédures prédéfinies les plus utilisées sont les suivantes :

* manipulation des descriptions des figures :

- *openfig* : fonction d'ouverture d'une figure. Elle équivaut à la génération pour la description LUCIE de : fig < nom >.
  syntaxe : **i: = openfig ('nom') ;** (i est le pointeur sur la figure courante)

- *getfig* : fonction d'appel d'une figure générée par LUBRICK ou par le digitaliseur. La figure appelée doit être fermée et externe.
  syntaxe : **i: = getfig ('nom') ;**

- *closefig* : fonction de fermeture d'une figure ouverte et non vide. Elle génère pour LUCIE l'expression : ffig.
  syntaxe : **i: = closefig (i);**

- *putfig* : procédure qui permet de fermer une figure interne principale (si elle était ouverte). Elle génère une figure fermée non modifiable et externe.Elle génère pour LUCIE l'expression : ffig (si la figure était ouverte).
  syntaxe : **putfig (i);**

*manipulations internes à une figure :

- *callrec* : procédure de génération d'un rectangle à l'intérieur d'une figure ouverte. Elle génère pour LUCIE l'expression : rec (x, y, dx, dy, niveau)
  syntaxe : **callrec (x, y, dx, dy, niveau) ;**

- *supconn* : fonction de suppression d'un ou plusieurs connecteurs à l'intérieur d'une figure en précisant leurs orientations (nord, est, sud, ouest), leurs types LUBRICK, ainsi que le rang dans la description LUBRICK de la figure ouverte et enfin le nombre de connecteurs à supprimer.

Cette fonction permet de masquer des connecteurs seulement pendant le déroulement du programme d'assemblage dans lequel elle est appelée, elle ne modifie pas la description LUBRICK source.

syntaxe : **i: = supconn (i, ccn , typ , rang, nombre) ;**
cce
ccs -1
ccw

(remarque : la valeur -1 indique n'importe quel type).

-*addconn*: fonction d'ajout d'un connecteur LUBRICK à l'intérieur d'une figure en fournissant tous les paramètres d'un connecteur, i.e., orientation, offset, position, largeur, niveau et type. Cette fonction rajoute un connecteur à la figure appelée et ne modifie pas sa description source.

syntaxe : **i:=addconn(i,ccn,offset,position,largeur,niveau,type);**
cce
ccs
ccw

-*setbound* : fonction de modification des frontières d'une figure. Il faut lui donner la taille en x et y, les offsets et les niveaux des frontières nord, est, sud, ouest. La valeur -1 pour les niveaux indique que l'on souhaite conserver l'ancien niveau.

syntaxe :
i: = setbound (i, dx, dy, offn, nivn, offe, nive, offs, nivs, offn,nivn);

* opérateurs géometriques

Ces opérateurs s'appliquent sur des figures fermées, soit externes (accédées par Getfig), soit internes (après un Closefig).

- *symx, symy, rotp, rotm* : fonctions de symétrie (par rapport à l'axe des x ou l'axe des y) ou de rotation (de + 90° ou de - 90°), d'une figure fermée. Les frontières et les connecteurs de la figure résultante sont calculés.

Chaque fonction génère son équivalent pour LUCIE :

symx --> sym (x, dx) ... fsym
symy --> sym (y, dy) ... fsym
rotp --> rot (+, dx) ... frot
rotm --> rot (-, dy) ... frot

syntaxe : **i: = symx (i) ;** (idem pour symy, rotp et rotm).

- *repx, repy* : fonctions de répétition suivant l'axe des x ou l'axe des y de la figure fermée pointée par le paramètre. Les règles technologiques sont respectées par le calcul des pas de répétition. De même les frontières et les connecteurs de la figure résultante sont calculés. Chaque fonction génère son équivalent pour LUCIE :

repx --> rep (x, nombre)

repy --> rep (y, nombre)

syntaxe : **i: = repx (i, nombre);**

* assemblage des figures

Les fonctions d'assemblage et de placement des figures sont basées sur le principe d'assembler chaque figure appelée à droite ou au dessus de la figure précédente.

Toutes les possibilités de positionnement existent (en combinant les fonctions et procédures de manipulations internes des figures et des opérateurs géométriques). De plus, ces fonctions respectent les règles de garde, et assurent l'interconnexion des figures soit directement, soit par des routages.

- *right, up* : fonctions de placement d'une ou plusieurs figures fermées à droite (right) ou au dessus (up) de la figure ouverte pointée par i. Ce pointeur peut s'appeler "figure 1" et la (les) figure(s) à assembler "figure 2" : **i: = right (figure 1, figure 2, nombre, offset) ;** dans ce cas "figure 2" serait répétée à droite de "figure1" un certain nombre de fois avec un certain "offset" (voir figure 11).

Cette fonction "right" génère pour LUCIE :

```
rep (x, nombre, pas)
    fig figure 1 (abcisse, ordonnée)
frep
< éventuellement rectangles de raccord >
```

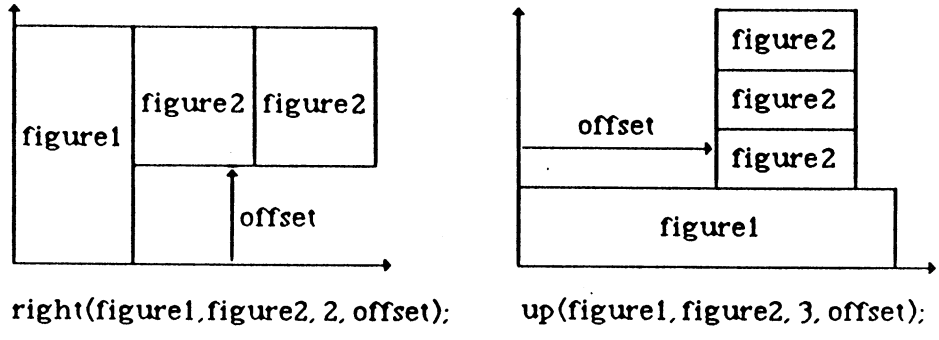


Figure 11 - Assemblage à droite et au dessus

- *cright, cup* : ces fonctions remplissent les mêmes rôles que les deux précédentes. Mais de plus, elles assurent les interconnexions des figures par routage : c'est le cas où les connecteurs des deux figures à assembler nécessitent pour les raccorder, d'amener des lignes coudées ou en escalier. Les croisements des lignes de raccord ne sont pas prévus par ces fonctions. Par ailleurs, un tableau associé à chaque fonction établit la correspondance entre la liste des connecteurs de la figure de gauche (ou du bas) avec les connecteurs de la (les) figure(s) de droite (ou du haut).

Deux autres fonctions réalisent les mêmes opérations que les deux précédentes, mais de plus elles autorisent les croisements des lignes de raccord grâce à deux niveaux d'interconnexions : le polysilicium et l'aluminium. Elles nécessitent également d'établir les tableaux de correspondance des connecteurs : ces deux fonctions sont :

crright et crup (pour "channel routing" right et up).

syntaxe : **i:= cright (i, j, offset bas, offset haut, tableau);**
i:= crup (i, j, offset gauche, offset droite, tableau);

idem pour crright et crup.

Ces fonctions génèrent pour LUCIE les mêmes expressions que right et up.

LUBRICK a été développé dans l'équipe de recherche en architecture des ordinateurs [SCH 85]. Il est écrit en PASCAL, et il est implanté sur VAX/VMS et sur SM90/UNIX.

3 - Méthode appliquée pour la conception de COBRA

L'architecture particulière de COBRA nécessite pour l'utilisation des outils de CAO une certaine stratégie. Sa particularité est due à une architecture non standard, car certains des éléments de la partie opérative ne sont pas divisibles en "bit slice" : cas des contrôleurs et de quelques opérateurs spécifiques. En effet, les contrôleurs (double-rail ou de parité) doivent être construits et montés avant d'être assemblés dans la partie opérative. La raison est qu'il n'existe pas une cellule de base pour un contrôleur d'un bit d'information. De plus, l'information à contrôler ici est sur 8 bits, et donc la structure adéquate est un contrôleur 8 bits, composé de 7 cellules de base (pour un contrôle double-rail), comme celle montrée à la figure 11 du chapître 2. De même pour le contrôleur de parité 8 bits qui est composé de 6 cellules de base, construit d'abord et ensuite inclu dans la structure de la partie opérative. Certains opérateurs sont également montés avant d'être assemblés, comme le "shifter" et le décodeur DECO.

Mais certains des outils sont insensibles à ces problèmes, toutefois, leur emploi dans la conception de COBRA a nécessité quelques opérations supplémentaires.

Le cas d'IRENE est typique, car ce langage est souple par définition, puisqu'il décrit le comportement de tout circuit de type microprocesseur. Toutefois, la description IRENE-C de COBRA a été faite à deux reprises : d'abord directement à partir du jeu d'instructions correspondant à un circuit COBRA non autotestable. De cette façon le circuit a été fonctionnellement validé et sa structure a pu être définie. A partir de cette étape, les différents blocs fonctionnels ont pu être étudiés, et à chacun de ces blocs a été associé une structure de contrôle de sorte que tous les blocs sont devenus conformes un par un et dans leur ensemble aux règles SFS présentées dans le chapitre précédent. La deuxième description IRENE était due à la modification de la structure de la partie opérative de COBRA par la structure de test. Cette deuxième description a pris en compte les commandes opératives des blocs rajoutés (contrôleurs, blocs dupliqués, ...). Cette deuxième description est celle qui a servi à la génération des contenus des PLAs de la partie commande.

Le système LUCIE fonctionne dans tous les cas. En effet, tous les opérateurs fonctionnels et de contrôle ont été décrits avec LUCIE en respectant dans chaque cas, indifféremment de leurs structures, les règles électriques du CMP-NMOS.

La génération des contenus des PLAs (de la partie commande) avec MACSIM a été faite une seule fois, à partir de la deuxième description IRENE. Le PLA de commandes génère donc toutes les commandes nécessaires, i.e., à la fois les commandes fonctionnelles occasionnées par l'exécution d'une instruction décodée, et les commandes (supplémentaires) de la structure de contrôle des blocs exécutant cette instruction, ces commandes se déroulant en parallèle.

Mais MACSIM ne différencie pas une structure classique d'une structure autotestable, puisqu'il transforme une description IRENE en des tableaux de remplissage de PLAs. PAOLA transforme ces tableaux de remplissage en dessin de masques de PLA, également sans distinguer les structures. Mais par contre, il faut rajouter une étape intermédiaire entre MACSIM et PAOLA pour coder les sorties des PLAs avec le code de Berger (voir chapitre II 4-5-2). Des contrôleurs de Berger détecteront les erreurs (unidirectionnelles) que peut avoir chaque PLA en ses sorties. Ce codage est rajouté avec un programme PASCAL, qui transforme le tableau de la matrice OU d'un PLA (matrice de sorties), en un tableau dont chaque ligne est codée. Ce programme rajoute en bout de chaque ligne de la matrice OU le complément à 1 du nombre binaire que contient la ligne. Les matrices ainsi transformées seront données à PAOLA.

Le simulateur électrique SPICE ne nécessite aucune opération supplémentaire. Il permet de simuler n'importe quel circuit (contrôleur et blocs fonctionnels) à assembler dans la structure globale.

Le rôle d'APOLLON dans le cas de COBRA fut celui de la validation de la conception de sa partie opérative, avant de rajouter les blocs de contrôle. Toutefois, il serait intéressant de rajouter à APOLLON les fonctions lui permettant de compiler également des parties opératives autotestables, soit entièrement, soit par sous-parties opératives. En effet, la structure d'APOLLON se prête à ce type d'opération, puisqu'à partir d'une description comportementale (IRENE par exemple), le compilateur APOLLON procède à une construction progressive de la partie opérative en empruntant une démarche identique à celle d'un concepteur.

L'assemblage des éléments d'une sous-partie opérative ainsi que des sous-parties opératives entre elles se fait avec LUBRICK en utilisant une bibliothèque de cellules. Ainsi il serait possible de réaliser avec APOLLON la compilation par blocs (sous-parties opératives) d'une partie opérative munie de sa structure de contrôle, en rajoutant (sans modifier), quelques fonctions au compilateur APOLLON. La bibliothèque de cellules devrait être augmentée des cellules de contrôle de codes les plus classiques (parité, Berger, double-rail, etc...), d'autant plus que les contrôleurs correspondant aux nouvelles techniques de test unifié, ont une structure modulable et s'assemblent donc facilement avec un programme comme LUBRICK.

Il faudrait également rajouter, dans la bibliothèque de cellules les opérateurs duaux, réalisant les fonctions complémentaires des opérateurs classiques pour le codage double-rail, ainsi que des éléments comme les générateurs de parité. L'assemblage des cellules se ferait alors de la même manière que pour les circuits classiques, avec des programmes LUBRICK, mais en respectant rigoureusement les règles de conception des circuits SFS (voir chapitre II) qui doivent être appliquées, d'un côté lors du dessin des cellules, et de l'autre côté au moment de l'assemblage des cellules (ce qui peut remettre en cause l'architecture des cellules).

La description comportementale serait faite en deux étapes également : la première étape serait celle d'un circuit classique sans contrôle. Ensuite, APOLLON et ses nouvelles fonctions (qui pourraient être interactives et optionnelles) construiraient l'ensemble des sous-parties opératives ainsi que leurs structures de contrôle associées. Ces mêmes fonctions, fourniraient en sortie les commandes supplémentaires à rajouter à la description comportementale (deuxième étape) afin de réitérer la description, mais cette fois celle de la machine respectant les règles SFS. A partir de cette dernière description, il serait possible de réaliser la partie commande (du modèle de MACSIM par exemple), et de rajouter aux PLAs le codage de Berger, avec un programme simple. La méthode proposée pour la généralisation du rôle d'APOLLON aux circuits SFS, est la même méthode utilisée pour la conception de COBRA.

4-CONCLUSION

Ce chapitre présentait les outils de CAO utilisés dans la conception de COBRA. Leur présentation était parfois volontairement détaillée, afin de permettre aux lecteurs et concepteurs débutants de se familiariser avec le fonctionnement de ces outils. Les références données pour chacun des outils permettent d'approfondir l'étude de leur utilisation et de leur fonctionnement.

La méthode appliquée pour la conception de COBRA était présentée à travers la description de chaque outil, dans le même ordre que leur présentation. Cette méthode de conception est particulière à un circuit spécifique autotestable. Mais il est possible d'automatiser la conception de circuits autotestables en rajoutant quelques étapes supplémentaires dans le processus de conception assistée par les outils présentés dans ce chapitre : exemples d'APOLLON et de MACSIM. Actuellement, plusieurs outils de CAO sont en étude, et sont regroupés dans le cadre d'un grand projet national CAO de VLSI : SYCOMORE. La possibilité de réaliser, même partiellement, des circuits autotestables peut être intéressante, d'autant plus qu'en l'état actuel, elle ne serait pas coûteuse.

CHAPITRE 4

DESCRIPTION DE LA REALISATION DE COBRA

1- INTRODUCTION

Ce chapitre traite de la conception proprement dite du circuit COBRA. Les trois chapitres précédents introduisent naturellement ce dernier : le premier chapitre comporte le cahier des charges fonctionnelles, représentant ainsi l'aspect spécifique du circuit, le deuxième chapitre comporte les règles des circuits SFS, représentant l'aspect de sécurité du circuit (self-checking) appliquées lors de la conception de COBRA, et enfin le troisième chapitre présente les moyens mis en œuvre et la méthode de conception adoptée.

La présentation de la conception de COBRA, dans ce chapitre commence par une description des signaux externes et s'oriente alors vers une vue globale de l'intérieur du circuit. Les spécifications externes, avec leurs contraintes dues au besoin de contrôle des entrées/sorties, sont d'abord présentées. Ensuite, les spécifications internes, qui correspondent aux descriptions logiques des fonctions à réaliser. Toutes les fonctions sont décrites dans la suite, une par une et dans le détail, jusqu'aux réalisations matérielles, et jusqu'à l'obtention des dessins des masques.

2- LES SPECIFICATIONS DE COBRA

La définition du cahier des charges d'un circuit intégré spécifique (nommé plus tard COBRA) a été choisie après plusieurs présentations techniques. Les spécifications globales de ce circuit ont découlé de la définition du cahier des charges et d'un ensemble d'informations sur des systèmes réalisant de manière câblée (portes discrètes) les fonctions de surveillance de sécurité dans les systèmes de transport : en l'occurrence le métro de Lille et le métro de Lyon.

Les spécifications externes et fonctionnelles du circuit à réaliser ont été inspirées des plans de câblage des chaînes de surveillance de sécurité des P.A. (Pilote Automatique) des métros de Lille et de Lyon.

2.1 Les spécifications externes

Les spécifications externes d'un circuit correspondent :
- au nombre de broches nécessaires
- au type de chaque broche et de la nature du signal qu'elle transmet.

Les parties logiques de surveillance des systèmes de métro de Lille, le VAL, et du métro de Lyon, ont permi d'établir les spécifications externes de COBRA. La synthèse des plans électriques de chacun de ces deux systèmes a abouti à l'aspect extérieur d'un circuit capable de gérer la surveillance de sécurité du système correspondant. Aux entrées/sorties de chaque circuit correspondent des signaux réels spécifiques à chacun des systèmes, comme le signal d'entrée de "détection de présence de station" ou le signal de sortie actionnant le freinage d'urgence.

L'analyse de ces deux systèmes a permi également de distinguer deux types de signaux d'entrées/sorties (voir chapitre 1) :

- les signaux de type NIVEAU, où seul le niveau logique compte à un instant donné.

- les signaux de type HORLOGE, dont les transitions comptent soit pour le calcul d'une fréquence, soit pour le calcul d'un délai.

La synthèse du système de surveillance du pilote automatique embarqué du VAL a donné un circuit qui aurait : 13 entrées NIVEAU, 4 entrées HORLOGE, et 3 sorties NIVEAU. Alors que la synthèse du système de Lyon, a abouti à un circuit qui aurait : 12 entrées NIVEAU, 3 entrées HORLOGE, 4 sorties NIVEAU et 1 sortie HORLOGE. A partir de ces informations et de l'idée de réaliser un circuit dont la structure est souple, les spécifications externes de COBRA ont été fixées à : 14 entrées NIVEAU, 5 entrées HORLOGE, 4 sorties NIVEAU et 3 sorties pouvant être au choix, soit NIVEAU soit HORLOGE (dont la fréquence est fixée dans le circuit). Par ailleurs, le choix de stockage du programme d'application était fixé à celui d'une mémoire morte externe (type ROM ou REPROM), alors qu'il existait une autre possibilité à l'étude, qui était celle d'un circuit cœur auquel serait associée une ROM interne dont le contenu serait particulier à chaque application. Cette dernière solution supposait la fabrication d'un circuit spécial en générant sur demande les masques de la ROM interne, particuliers à chaque application.

La solution choisie a l'avantage d'être plus souple, car il suffit de changer la ROM pour changer d'application, tout en gardant le circuit cœur. Mais cette solution a l'inconvénient d'utiliser, pour le dialogue avec la ROM externe, 11 broches du circuit. Ceci a un impact sur la généralité du circuit, qui est limitée par le nombre de ses broches, par son jeu d'instructions et par la taille maximale de son programme d'application. La taille maximale de la ROM dans le cas de COBRA est de 16 koctets adressés en deux fois par multiplexage externe sur le bus adresses/données.

Les 11 broches se partagent alors en : 8 broches d'informations, 1 broche de sélection de la ROM en lecture, 1 broche de validation d'adressage poids fort en agissant sur un port d'adresse externe, la 11ème broche est la duplication de la dernière pour le contrôle double-rail. La figure 1 montre le brochage de COBRA. Par ailleurs, COBRA comprend une liaison série destinée au dialogue entre deux circuits COBRA. Trois broches servent à la liaison série, leur usage sera détaillé dans le paragraphe décrivant la liaison série.

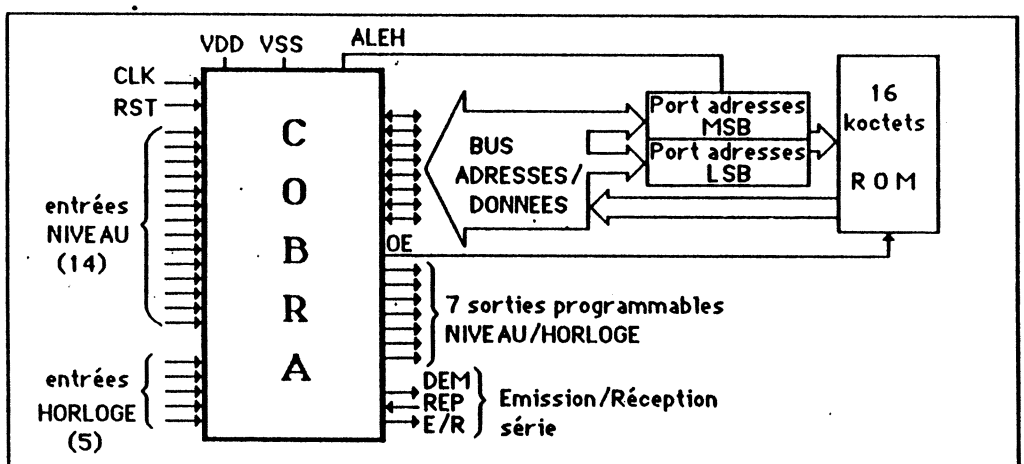


Figure 1 - brochage fonctionnel du circuit COBRA

Le nombre total de broches fonctionnelles est de 42. Les broches supplémentaires nécessaires au contrôle du circuit et de ses entrées/sorties sont au nombre de 22. Au total 64 broches utilisées de la façon suivante :

	fonctionnel	contrôle
entrées NIVEAU	14	2
entrées HORLOGE	5	5
sorties	7	7
busA/D	7	1
ALEH	1	1
OE	1	
liaison série	3	2
alimentation, masse	2	
horloge	1	1
reset	1	1
sorties des contrôleurs		2

2.2 Contrôles des Entrées/Sorties

2.2.1 Le contrôle des entrées NIVEAU

Les entrées NIVEAU sont contrôlées à l'intérieur du circuit. Elles sont codées en parité paire: un bit de parité pour un ensemble de 7 entrées. La parité doit être calculée à l'extérieur du circuit par deux générateurs de parité (figure 2). Les entrées des générateurs doivent être indépendantes de celles du circuit, afin que les pannes éventuelles soient toujours des pannes simples, détectables par le code de parité.

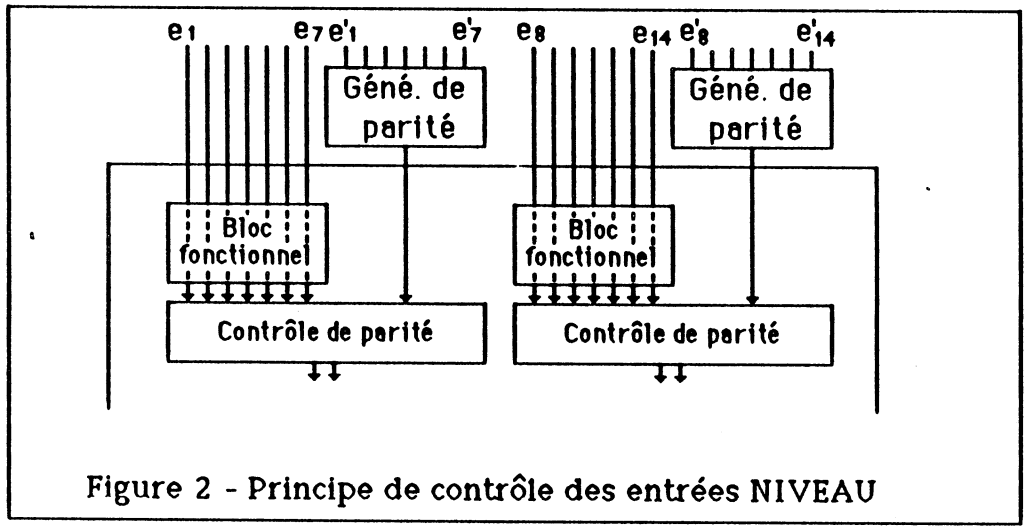


Figure 2 - Principe de contrôle des entrées NIVEAU

La présentation ainsi donnée des entrées NIVEAU est intéressante du point de vue fonctionnel, car il est possible de les considérer selon trois façons différentes :

a) 14 variables booléennes indépendantes (ou moins), représentant chacune une information analogique provenant généralement d'un capteur (présence d'une station, sens de la marche, etc.). Ces variables sont traitées individuellement dans une partie spécifique du circuit, mais leur contrôle de parité est permanent grâce aux 2 bits de parité qui se partagent chacun le codage de 7 variables.

b) 7 variables booléennes indépendantes (identiques aux précédentes, codées avec 1 bit de parité), et un bus de 8 bits (dont la parité), accédant via un registre tampon puis le bus interne du circuit au registre accumulateur d'entrée de l'unité arithmétique et logique 8 bits de la partie opérative (UAL). Une instruction spéciale est associée à cette opération.

c) Deux bus de 8 bits chacun (dont les parités), accédant chacun par une instruction spéciale au registre accumulateur d'entrée de l'UAL. Dans tous les cas, le contrôle de parité est permanent sur toutes les entrées NIVEAU.

2.2.2 Le contrôle des entrées HORLOGE

A la différence des entrées NIVEAU qui véhiculent des signaux logiques dont le codage en parité est intéressant et suffisant, les entrées HORLOGE transportent souvent des signaux de type fréquence dont les périodes sont par définition aléatoirement variables. Cette contrainte ne facilite pas l'emploi d'un code simple, comme le code de parité. Finalement, le choix fut celui du code à double-rail qui nécessite la duplication des entrées HORLOGE du circuit, et l'introduction des signaux direct et inversé pour chaque information de type HORLOGE. Les contrôleurs double-rail sont placés en sortie du bloc fonctionnel recevant les entrées HORLOGE (figure 3).

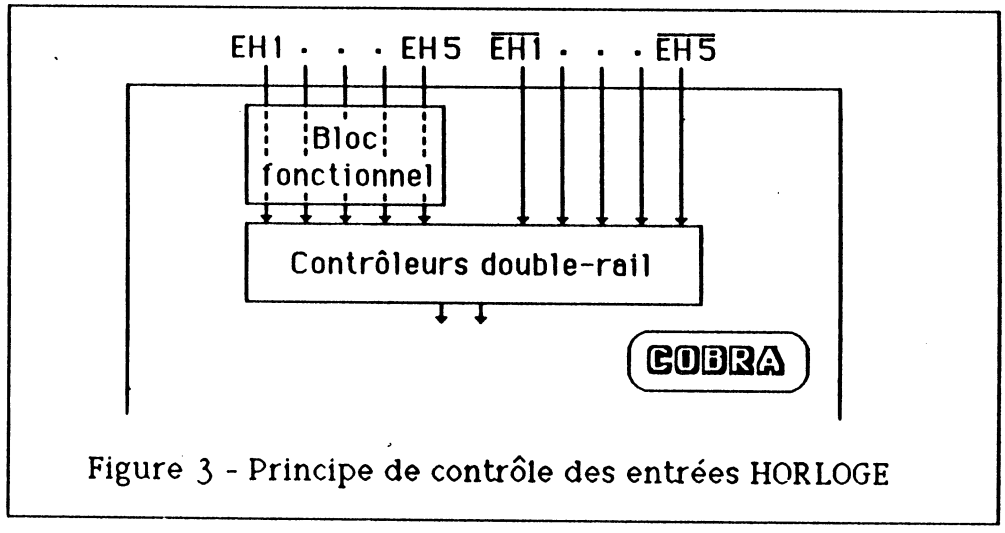


Figure 3 - Principe de contrôle des entrées HORLOGE

2.2.3 Le contrôle des sorties

Les sorties au nombre de 7 sont dupliquées et codées en double-rail. Le choix de ce codage est dû à l'aspect fonctionnel de ces sorties. En effet, trois de ces sorties peuvent être utilisées, soit comme sorties NIVEAU, soit comme sorties HORLOGE, en fonction de l'application du circuit. Pour cette raison l'emploi d'un codage autre que le code double-rail serait d'une lourde conséquence sur l'exploitation des sorties d'un point de vue fonctionnel.

Le contrôle double-rail de ces sorties doit se faire à l'extérieur du circuit avec des contrôleurs double-rail disposés selon la même structure que les contrôleurs à l'intérieur du circuit. Le fonctionnement de la structure des sorties ainsi que leur contrôle est détaillé dans le paragraphe correspondant.

2.2.4 Le contrôle du bus adresses/données et de la ROM externe

Le bus externe est la liaison entre le circuit et la ROM externe, contenant le programme d'application. Ce bus assure une communication bidirectionnelle sur 8 bits de l'ensemble des adresses et des données : 7 bits d'informations et un bit de codage de parité. Ce format est le même que celui de la partie opérative, dans laquelle un registre tampon assure le stockage des octets de données ou adresses entre la ROM et le circuit. La figure 4 montre la structure de la liaison avec la ROM [OSS 86], ainsi que les lignes de contrôles de cette liaison ALEH (Adress Latch Enable High) et OE (Output Enable).

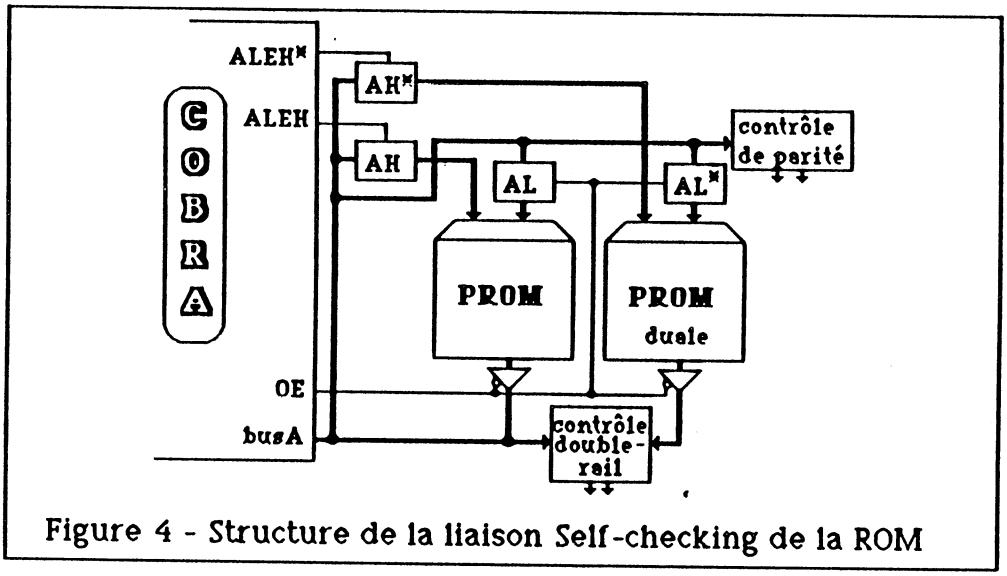


Figure 4 - Structure de la liaison Self-checking de la ROM

Le fonctionnement de cette structure est le suivant :

L'adressage de la ROM se fait avec 14 bits d'adresses (16 koctets de ROM à adresser), multiplexés à l'extérieur du circuit par deux ports d'entrées/sorties (de type 8212 Intel) l'un pour l'adresse poids faible, AL, et l'autre pour l'adresse poids fort, AH.

Le plan mémoire ROM peut donc atteindre 128 pages de 128 octets chacune. L'adressage d'un mot dans une page (figure 5) de 128 octets nécessite de changer seulement le poids faible de l'adresse, donc le contenu de AL seulement. Le changement de page se fait dans le port AH par une instruction de branchement.

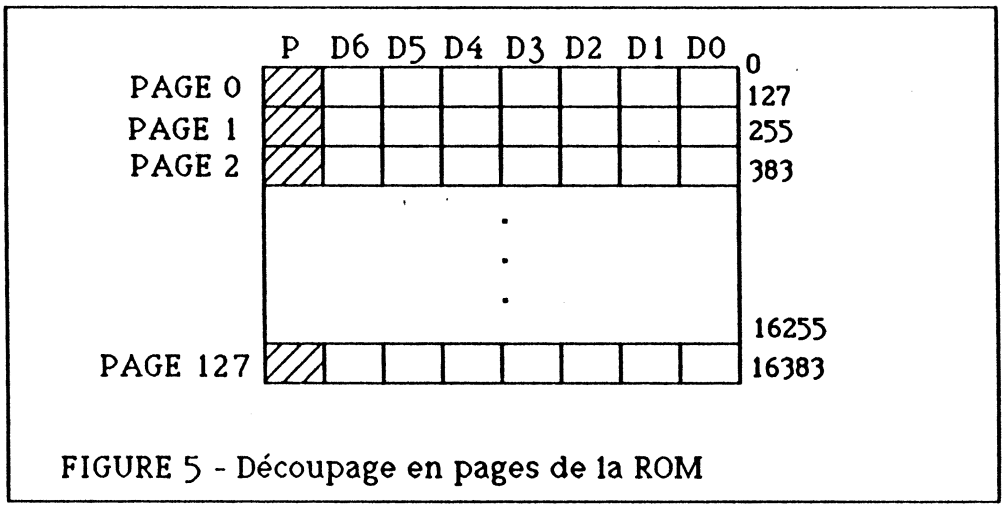


FIGURE 5 - Découpage en pages de la ROM

Cette structure du plan de la ROM permet de définir deux types de branchements : les branchements courts, dans la page courante, ce qui permet de ne modifier que le poids faible de l'adresse ; les branchements longs, qui permettent d'accéder à n'importe quel octet du plan mémoire. Les détails sont donnés dans le paragraphe du jeu d'instructions. La structure de la liaison avec la ROM nécessite pour des raisons de contrôle (self-checking) de dupliquer la ROM ainsi que les ports d'adresses AL et AH, et de rajouter un contrôleur de parité et un contrôleur double-rail (figure 4). La fonctionnalité n'est pas modifiée par cette structure particulière, mais la surveillance est totale. En effet, étant donné que la ROM du commerce, utilisée dans ce cas, possède une structure inconnue, pouvant avoir des pannes provoquant des erreurs multiples, il est nécessaire de contrôler les sorties des ROM en double-rail. Il est également nécessaire de stocker dans la ROM duale les octets complémentaires de ceux du programme d'application. Toute panne dans cette structure est détectée dans l'un des deux contrôleurs (parité ou double-rail).

2.2.5 Contrôle de la liaison série

La liaison série utilise 3 lignes pour son fonctionnement : une bidirectionnelle qui est la ligne série proprement dite, et deux pour le contrôle du fonctionnement de cette ligne. Le contrôle des pannes possibles sur la ligne bidirectionnelle est assuré par une duplication de l'information à émettre et par le contrôle en double-rail de cette information à la réception. Les deux lignes de contrôle du fonctionnement de la liaison série doivent être dupliquées et contrôlées dès qu'elles arrivent à leurs destinations. Le détail du fonctionnement de la communication série ainsi que de son contrôle est décrit dans le paragraphe associé à la liaison série.

2.2.6 Contrôle des lignes extérieures de contrôle et d'alimentation

Le signal d'horloge ainsi que le signal du Reset, sont dupliqués et sont contrôlés en double-rail à l'intérieur du circuit, comme le sont les entrées HORLOGE. Par ailleurs les lignes d'alimentaiton + 5V et la masse, ne nécessitent aucune structure supplémentaire pour leur contrôle (au niveau du câblage externe du circuit). La raison est que toute panne à ce niveau est détectée par tous les contrôleurs du circuit, et plus particulièrement le dernier contrôleur double-rail situé avant les sorties des contrôleurs du circuit, qui va indiquer la panne en ayant ses deux sorties soit à "zéro" (par rapport à l'alimentation globlale du système) soit à "un" (par rapport à la masse globlale du système) et ceci en fonction du lieu de la panne (alimentation + 5V ou masse).

2.2.7 Structure des contrôleurs

Les deux broches du circuit qui portent le nom de "sorties des contrôleurs", reçoivent le compte-rendu global de l'ensemble des contrôleurs de tous les blocs fonctionnels qui composent le circuit. Le signalement d'une panne sur ces deux broches (qui reçoivent les sorties d'un contrôleur double-rail) signifie la présence d'un panne à l'intérieur du circuit (dans un bloc fonctionnel, ou dans un contrôleur, ou bien au niveau d'une interconnexion). Les contrôleurs sont disposés dans le circuit selon un arbre. Le principe de leurs interconnexions est donné à la figure 7 du chapitre 2.

2.3 Les spécifications internes

Dans ce paragraphe, les spécifications internes fonctionnelles et de contrôle sont considérées. Ces spécifications représentent les matérialisations des fonctions que doit exécuter le circuit. Un aperçu des fonctions que réalise COBRA est donné dans les spécifications externes décrites ci-dessus. Les éléments et opérateurs qui réalisent ces fonctions dans le circuit COBRA ont les rôles suivants :

- Les traitements associés aux entrées NIVEAU, i.e. : l'acquisition d'une entrée NIVEAU (variable booléenne) ; l'application d'une fonction logique (ET, OU, OU exclusif et leurs compléments) à l'une quelconque des entrées avec une variable interne ; la copie d'une fois un mot de 7 bits dans le registre accumulateur de l'UAL.
- Les traitements associés aux entrées HORLOGE, i.e. : la détermination, pour une entrée, de la période séparant deux fronts montants sur cette entrée ; le signalement de l'évènement qui correspond à une période trop petite(cas de dépassement de vitesse du train par exemple).

- Les traitements associés aux sorties, i.e. : l'imposition d'une valeur logique (0 ou 1) ou d'une fréquence à une des sorties.
- La temporisation programmable pouvant décompter jusqu'à 22 secondes.
- La communication série pour transmettre des messages entre deux circuits COBRA.

Par ailleurs, le circuit comprend une unité arithmétique et logique 7 bits et une RAM interne de 64 octets.

2.3.1 Jeu d'instructions

Le jeu d'instruction donne une idée précise des possibilités d'un circuit programmable. Au départ de l'étude de COBRA, le nombre d'instructions a été volontairement limité au strict minimum pour une raison de simplification et surtout dans un but de contrôle et de prévention contre la complexité du circuit. Les microprocesseurs du commerce se comparent avec l'importance de leurs jeux d'instructions. Or selon les applications d'un circuit du commerce, son jeu d'instructions est utilisé à un certain pourcentage, et il est très rare que toutes les instructions soient utilisées. Ceci ne correspond pas aux applications de sécurité, car une instruction non utilisée correspond à un ou plusieurs blocs fonctionnels non activés, et donc peuvent avoir une ou plusieurs pannes latentes. Pour cela le jeu d'instructions de COBRA est limité à 43 instructions (tous modes d'adressage confondus). Elles sont distribuées de la manière suivante :

- 11 instructions associées aux entrées NIVEAU.
- 14 instructions de branchements, courts et longs.
- 10 instructions arithmétiques et logiques (sur 7 bits)
- 1 instruction d'activation de sortie
- 2 instructions d'écriture/lecture de la RAM
- 3 instructions de temporisations sur 14 bits
- 1 instruction associée à l'émission série
- 1 instruction de retour d'interruption

La page suivante montre l'ensemble des instructions, les opérations qu'elles effectuent, ainsi que les actions sur les bits indicateurs des états .

Mnémo	~	#	CODOP	CCR Z1	V	C	N	Z2	Opération effectuée
LZ	4	2	48	↓	.	.	.	.	Z := entrée Ei
AZ	"	"	4B	↓	.	.	.	.	Zj := (entrée Ei) ET (Zj-1)
OZ	"	"	4D	↓	.	.	.	.	Zj := (entrée Ei) OU (Zj-1)
XZ	"	"	4E	↓	.	.	.	.	Zj := (entrée Ei) OUEX (Zj-1)
NAZ	"	"	C9	↓	.	.	.	.	" " NAND "
NOZ	"	"	CA	↓	.	.	.	.	" " NOR "
NXZ	"	"	CF	↓	.	.	.	.	" " NOREX "
SXZ	2	1	CØ	.	.	.	.	.	X: = Z
SYZ	"	"	C3	.	.	.	.	.	Y: = Z
CIL	"	"	6C	.	R	.	↑	↑	ACCA := entrées (Ei), (i : 1 à 7)+P
CIM	"	"	6F	.	R	.	↑	↑	ACCA := entrées (Ei), (i : 8 à 14)+P
BAS	4	2	3Ø	.	.	.	.	.	Branchement court inconditionnel
BAL	7	3	3F	.	.	.	.	.	" long "
BSZ	4	2	DB	.	.	.	.	.	" court si Z1=1 (Z=0)
BLZ	7	3	53	.	.	.	.	.	" long " "
BSNZ	4	2	DD	.	.	.	.	.	" court si Z1 = 0 (Z=1)
BLNZ	7	3	55	.	.	.	.	.	" long " "
BEQ	4	2	59	.	.	.	.	.	" court si Z2 = 1
LBEQ	7	3	D1	.	.	.	.	.	" long " "
BMI	4	2	5A	.	.	.	.	.	" court si N = 1
LBMI	7	3	D2	.	.	.	.	.	" long " "
BLT	4	2	5C	.	.	.	.	.	" court si N ⊕ V = 1
LBLT	7	3	D4	.	.	.	.	.	" long " "
BCS	4	2	5F	.	.	.	.	.	" court si C = 1
LBCS	7	3	D7	.	.	.	.	.	" long " "
LOAD	4	2	12	.	R	.	↑	↑	ACCA := opérande
ADDA	"	"	9A	.	↑	↑	↑	↑	ACCA := ACCA + opérande
SUBA	"	"	9C	.	↑	↓	↑	↑	ACCA := ACCA - opérande
ANDA	"	"	18	.	R	.	↑	↑	ACCA := ACCA ET opérande
ORA	"	"	1B	.	R	.	↑	↑	ACCA := ACCA OU opérande
CMPA	"	"	99	.	↑	↓	↑	↑	ACCA - opérande
COM	2	1	ØØ	.	R	S	↑	↑	ACCA := $\overline{ACCA}$
DEC	"	"	Ø3	.	*	.	↑	↑	ACCA := ACCA - 1
LSL	"	"	81	.	**	↑	↑	↑	C <- [ACCA] <- 0
LSR	"	"	82	.	.	↑	R	↑	0 -> [ACCA] -> C
ACO	4	2	FØ	.	.	.	.	.	OR := opérande
STM	5	"	2D	.	R	.	↑	↑	RAM (opérande) : = ACCA
LDM	"	"	2E	.	R	.	↑	↑	ACCA := RAM (opérande)
TMPA	6	3	71	.	.	.	.	.	CNTA := opérande
TMPB	"	"	72	.	.	.	.	.	CNTB := "
TMPC	"	"	74	.	.	.	.	.	CNTC := "
SERT	3	1	EØ	.	.	.	.	.	TR := ACCA
RTI	6	"	FF	.	.	.	.	.	Retour d'interruption

~ : nombre de cycles machine
#: nombre d'octets nécessaires
R : mis à 0 S : mis à 1
↑: mis à 1 ou à 0 selon le résultat
↓: mis à 0 ou à 1 selon le résultat
. : inchangé
* : vaut 1 si l'opérande avant opération est $ 8Ø
** : V prend la valeur N⊕C après décalage

TABLE 1 - JEU D'INSTRUCTIONS

LZ

Acquisition d'une entrée NIVEAU <i> puis stockage de sa valeur dans le registre accumulateur 1 bit, Z. Son numéro, contenu dans l'opérande de LZ sera stocké dans un registre d'adresse, MS, pour être communiqué au multiplexeur 16 -> 1.

AZ, OZ, XZ, NAZ, NOZ, NXZ

Acquisition d'une entrée NIVEAU <i> de la même façon que LZ, puis réalisation de la fonction logique (respectivement ET, OU, OUEX, NAND, NOR, NOREX), avec le contenu de l'accumulateur Z. Chaque fonction logique est donnée dans le code opération de chaque instruction. Le résultat de l'opération est stocké dans Z.

SXZ, SYZ

Copie du contenu de Z dans un registre tampon (X ou Y),pour permettre de réaliser des fonctions combinatoires entre les différentes entrées NIVEAU. Ces deux registres sont considérés ensuite comme des entrées NIVEAU pour être rappelés.

CIL, CIM

Copie des 14 entrées NIVEAU dans deux registres 8 bits à cet effet, selon trois modes différents décrits plus loin, afin de les stocker ensuite dans le registre accumulateur ACCA, pour toute sorte d'opération 8 bits.

BAS

Branchement inconditionnel court, dans une page de 128 octets, autour dE l'adresse courante.

BAL

Branchement inconditionnel long, dans tout l'espace mémoire adressable

BSZ, BSNZ

Branchement conditionnel court, si le dernier résultat est égal à zéro (pour BSZ) ou différent de zéro (pour BSNZ), l'adresse de branchement est contenue dans l'opérande. Si la condition n'est pas vraie, alors passage à l'instruction suivante.

BLZ, BLNZ

Branchement conditionnel long,les conditions sont les mêmes que pour BSZ ou BSNZ, l'adresse de branchement est contenue sur deux octets dans l'opérande.

BEQ, BMI, BLT, BCS

Branchement conditionnel court, les conditions de branchement sont respectivement: Z2=1, N=1, N+V=1 et C=1. Tous ces résultats sont donnés dans l'UAL 8 bits.

LBEQ, LBMI, LBLT, LBCS

Branchement conditionnel long, les conditions de branchement sont les mêmes .

LOAD

Chargement dans l'accumulateur ACCA d'une valeur contenue dans son opérande

ADDA, SUBA, ANDA, ORA

Ces instructions (arithmétiques: addition et soustraction, et logiques: ET et OU) réalisent les fonctions sélectionnées, entre leurs opérandes et le registre ACCA. le résultat se trouve ensuite dans le même registre ACCA.

CMPA

Cette instruction a le même séquencement que les quatre précédentes, elle réalise la même opération que SUBA, sauf que le résultat n'est pas conservé, seul le bits indicateurs d'état (flags) contiennet l'information résultante, dans le but de branchement conditionnel en général.

COM, DEC

Instructions de complémentation et de décrémentation du contenu du registre ACCA. Le résultat se trouve ensuite dans ACCA.

LSL, LSR

Instructions de décalage à gauche ou à droite du contenu du registre ACCA. Le résultat se trouve ensuite dans ACCA.

ACO

Forçage d'une sortie, soit à une valeur logique (1 ou 0), soit à une fréquence. Le numéro de la sortie à activer ainsi que le choix du mode (logique ou fréquence) et la valeur à imposer sont donnés dans l'opérande, et stockés dans un registre approprié OREG.

STM, LDM

Chargement dans la RAM du contenu du registre ACCA (pour STM) et chargement dans le registre ACCA du contenu d'une case mémoire RAM (pour LDM). L'adresse de la case à sélectionner est donnée dans l'opérande.

TMPA, TMPB, TMPC

Chargement d'un compteur 14 bits , avec une valeur à partir de laquelle il va décompter, et ceci en deux fois 7 bits. Quand le décomptage est terminé, une interruption appellera un sous-programme correspondant au compteur choisi. Trois instructions correspondent à trois compteurs CNTA, CNTB et CNTC.

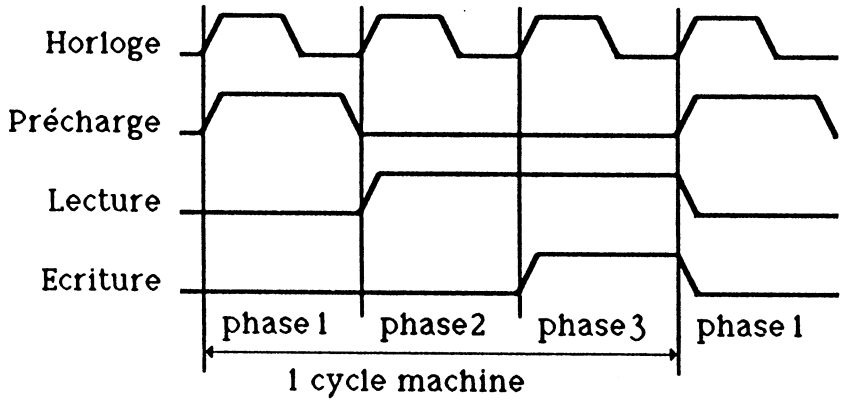
SERT

Instruction qui copie le contenu du registre ACCA dans un registre dit de "transmission série", TR, et ensuite donne la main à un PLA qui commande le transfert du TR vers un registre à décalage SR qui transforme le mot et le transmet vers l'extérieur du circuit en mode série.

RTI

Instruction de retour d'interruption, restaure les registres de travail après une interruption, et remet le processus dans le mode où il était avant l'interruption

Un cycle machine est décomposé en trois phases, comme le montre le diagramme suivant :



La durée d'un cycle est d'une microseconde. Donc les instructions nécessitent entre 2 microsecondes et 7 microsecondes chacune pour s'exécuter. Le diagramme ci-dessus représente l'état du bus interne de la partie opérative de COBRA. L'état de ce bus conditionne le séquencement de chaque instruction étant donné que ce bus est tantôt bus adresse, tantôt bus données. Donc les cycles de chaque instruction sont séquencés de sorte à pouvoir dialoguer avec la ROM via ce bus multiplexé. Le premier cycle de chaque instruction comprend la fin de décodage de son propre code opération, ce décodage ayant commencé à la fin de l'instruction précédente. Et naturellement, le dernier cycle est donc celui de l'acquisition du code opération de l'instruction suivante et le début de son décodage. L'utilisation d'un second bus interne permet l'exécution en parallèle des transferts de données propres à chaque instruction. L'exemple du paragraphe 2.2.1 du chapitre 3, illustre ces propos, ainsi que la description IRENE de toutes les instructions donnée à l'annexe A.

Une partie du jeu d'instructions est commune à tous les circuits de type microprocesseur : les instructions de branchement, les instructions arithmétiques et logiques, les instructions de la RAM et l'instruction de retour d'interruption. Les autres instructions sont spécifiques au circuit COBRA. Les branchements conditionnels dépendent des états des flags, qui sont au nombre de cinq :

Z1 : bit indicateur du zéro, associé à la structure des entrées NIVEAU. Ce "Flag" indique l'état de la dernière entrée ayant subi une opération dans cette structure.

V : bit de débordement (overflow), généré à partir de l'unité arithmétique et logique (UAL).

C : bit de retenue (carry), généré à partir de l'UAL.

N : bit de signe (négatif), généré à partir de l'UAL.

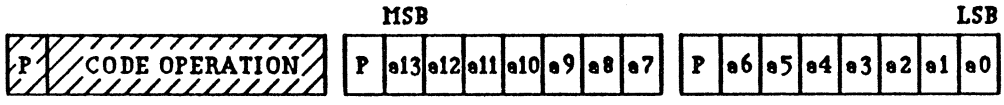
Z2 : bit de zéro, généré à partir de l'UAL.

Les branchements courts nécessitent deux octets, le premier étant le code opération de l'instruction correspondante, munie de son code de parité, et le deuxième octet est l'adresse de la case mémoire à sélectionner. Cette adresse est sur 7 bits, le 8ème étant le code de parité de l'adresse.

-Format de l'instruction de branchement court-

Les branchements longs nécessitent trois octets, le premier est toujours le code opération de l'instruction, le deuxième est le poids fort de l'adresse, le troisième est le poids faible de l'adresse.

Le format est le suivant :



-Format de l'instruction de branchement long-

Le format particulier des instructions de branchements longs est la raison pour laquelle elles sont décrites à part, ici. En effet, l'adresse effective est de 14 bits, mais l'adresse complète (avec les bits de parité) qui correspond à celle qui sera écrite dans les programmes est de 16 bits. Alors, une adresse effective qui s'écrivait, par exemple :

1B49 devient 36C9 avec le codage de parité

Ces deux écritures correspondent à l'adresse décimale : 6985, qui elle correspond à la 73 ème case mémoire de la 37 ème page de la ROM.

Ce problème est également posé dans le cas des instructions de temporisation 14 bits, mais d'une manière légèrement différente.

Toutefois, les explications de toutes les autres instructions sont données en même temps que les blocs fonctionnels qu'elles régissent.

2.3.2 Ressources matérielles

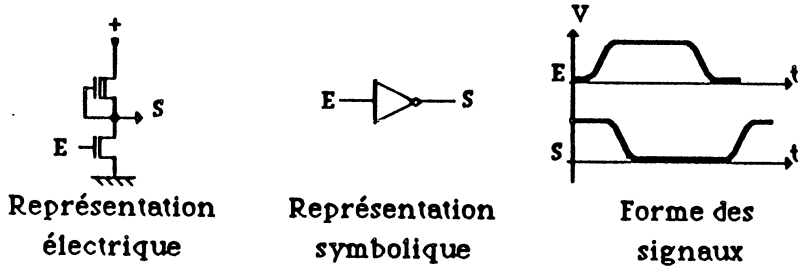
Les fonctions réalisées dans COBRA sont matérialisées par des opérateurs classiques ou spécifiques ainsi que des registres.

La plupart des éléments de la partie opérative particulièrement, sont construits à partir de cellules élémentaires. Ces cellules élémentaires constituent les ressources matérielles nécessaires à la réalisation de la machine physique. Une bibliothèque de cellules élémentaires standards a été utilisée pour le dessin du circuit. Elle contient des éléments comme les registres, les amplificateurs, un registre à décalage, et une unité arithmétique et logique, ainsi que plusieurs autres éléments. Elle permet dans certains cas standards, de réaliser une partie opérative complète avec les amplificateurs des commandes. C'est ce que fait APOLLON (voir chapitre 3) pour la compilation des parties opératives. Toutes les cellules de la partie opérative ont une structure basée sur le principe de bit-slice, et sont destinées à être assemblées par alignement, formant ainsi une partie opérative 1 bit, selon la structure de bus de la figure 8 du chapitre 3.

L'emploi de la bibliothèque de cellules dans le cas de COBRA a nécessité quelques précautions. En effet, chaque cellule a dû être vérifiée, pour s'assurer du respect des règles de conception des circuits SFS, et ceci avant et après son assemblage dans le circuit. Certaines cellules ont été reprises (cas du registre à décalage), et transformées en certains points afin qu'elles respectent les règles SFS. Généralement, les autres cellules de base ont été conservées puisqu'elles respectent les règles de conception. Toutefois, toutes les cellules de la bibliothèque n'ont pas été vérifiées, mais il serait intéressant pour la suite de le faire, afin de mettre sur pieds une bibliothèque de cellules pouvant être assemblées en vue de l'obtention d'un circuit respectant les règles de conception SFS. Mais pour que cela soit posssible, il faut d'abord vérifier le respect des règles SFS après assemblage.

Quelques éléments de base sont présentés ici. Les détails ne seront pas donnés ici, dans le souci de ne pas alourdir la présentation.

* L'inverseur NMOS



Représentation électrique | Représentation symbolique | Forme des signaux

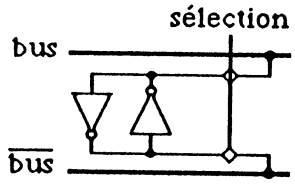
Pour les technologies MOS, l'inverseur est la porte logique de base à partir de laquelle serait faite toute interprétation d'une structure logique.

* L'interrupteur MOS

si la commande est vraie alors B et A sont reliés

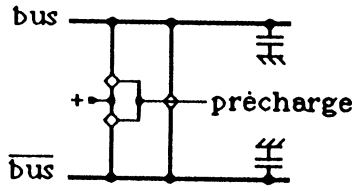
Le transistor MOS utilisé en tant qu'interrupteur (ou porte de passage) est très présent dans les circuits intégrés MOS.

* Le point mémoire NMOS



Le point mémoire est réalisé à partir de deux inverseurs, et deux interrupteurs (cas du bus bilifaire). La commande de sélection permet alternativement au BUS de lire le contenu du point mémoire ou d'y écrire une valeur binaire.

* Le circuit de précharge



Ce dispositif de précharge réalisé avec trois interrupteurs, permet de préparer les bus bifilaires (équivalents à deux condensateurs), à écrire ou lire sans perte de temps.

2.3.3 - Eléments architecturaux

La partie opérative de COBRA est divisée en plusieurs sous-parties opératives. Deux classes fonctionnelles regroupent ces sous-parties opératives :

a) Les fonctions communes à tout circuit de type microprocesseur :
   - L'unité arithmétique et logique
   - Les registres de travail : pointeur de pile, registre d'état, l'accumulateur, les registres compteurs de programme et le registre d'entrée/sortie des adresses/données.
   - La RAM interne (pour certains types de circuits) avec le registre de sélection correspondant.

b) Les fonctions spécifiques à l'application de COBRA :
   - La sous-partie opérative 1 bit, qui correspond aux entrées NIVEAU.
   - Les registres auto-incrémentés associés aux entrées HORLOGE
   - Les compteurs 14 bits pour la temporisation.
   - Le registre des sorties et les bascules des sorties.
   - Les registres de la liaison série ainsi que le registre à décalage.

Chacune de ces fonctions est décrite plus loin, en détaillant dans chaque cas sa fonctionnalité et la structure de contrôle qui lui est associée.

La partie commande est également présentée avec son codage et les contrôleurs qui lui sont associés. Mais dans ce qui suit, ce sont les éléments architecturaux correspondants aux fonctions spécifiques qui sont d'abord présentés, suivis par les autres éléments correspondants aux fonction communes.

2.3.4 - Structure de traitement des entrées NIVEAU

La structure nécessaire aux traitements des entrées NIVEAU, se compose d'un mutilplexeur, d'un opérateur, nommé unité logique et de quelques registres. La figure 6 montre le principe de cette structure.

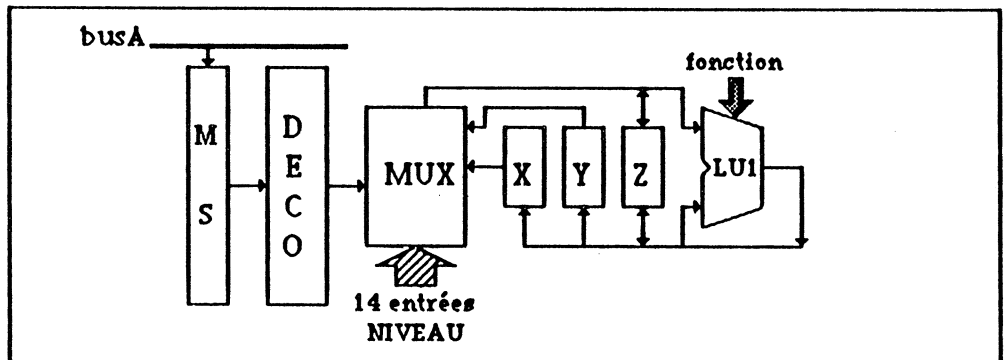
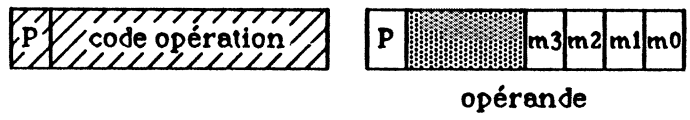


Figure 6 - Structure de traitement des entrées NIVEAU

Le registre MS reçoit le numéro de l'entrée NIVEAU à sélectionner (ce numéro est contenu dans l'opérande de l'instruction correspondante). Le décodeur 1 parmi 16 "DECO" sélectionne à l'entrée du multiplexeur "MUX" l'entrée correspondante et la stocke dans le registre 1 bit "Z", ou bien dans l'unité logique 1 bit "LU1". Ce choix est fait à partir du code opération de l'instruction employée. Les instructions utilisées dans cette structure, correspondant à la sous-partie opérative 1 bit, sont LZ,AZ,OZ,XZ,NAZ,NOZ,NXZ. La fonction que chacune réalise est décrite dan la table 1 montrant le jeu d'instructions. Le format de chacune des instructions est le suivant :



- Format d'une instruction : LZ, AZ, OZ, XZ, NAZ, NOZ, NXZ -

Le tableau 2 montre la correspondance entre l'opérande m_i et l'entrée correspondante à sélectionner :

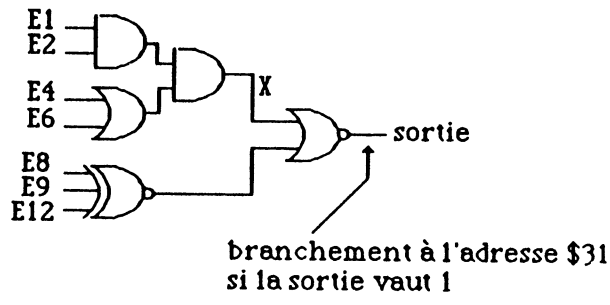
opérande				entrée sélectionnée
m3	m2	m1	m0	
0	0	0	0	x
0	0	0	1	e1
0	0	1	0	e2
0	0	1	1	e3
0	1	0	0	e4
0	1	0	1	e5
0	1	1	0	e6
0	1	1	1	e7
1	0	0	0	e8
1	0	0	1	e9
1	0	1	0	e10
1	0	1	1	e11
1	1	0	0	e12
1	1	0	1	e13
1	1	1	0	e14
1	1	1	1	y

- Tableau 2 : correspondance opérande/entrée à sélectionner -

L'existence des entrées sélectionnées x et y correspond à deux registres tampons X et Y. Deux instructions spéciales : SXZ et SYZ permettent de copier dans X ou Y le contenu de Z, afin de faire réaliser par programme une fonction combinatoire des entrées NIVEAU.

Exemples : 1- LZ $5 chargement dans Z de l'entrée E5
OZ $E OU avec l'enrée E14
BSZ $1F branchement à l'adresse 1F si le résultat est zéro

2- LZ $4
OZ $6
AZ $1
AZ $2
SXZ
LZ $8
NXZ $9
NXZ $C
NOZ $0
BSNZ $31



La structure de la figure 6 est contrôlée en double-rail sauf le registre MS et le décodeur DECO qui sont contrôlés en parité. La figure 7 montre la structure "self-checking" de cette sous-partie opérative 1 bit.

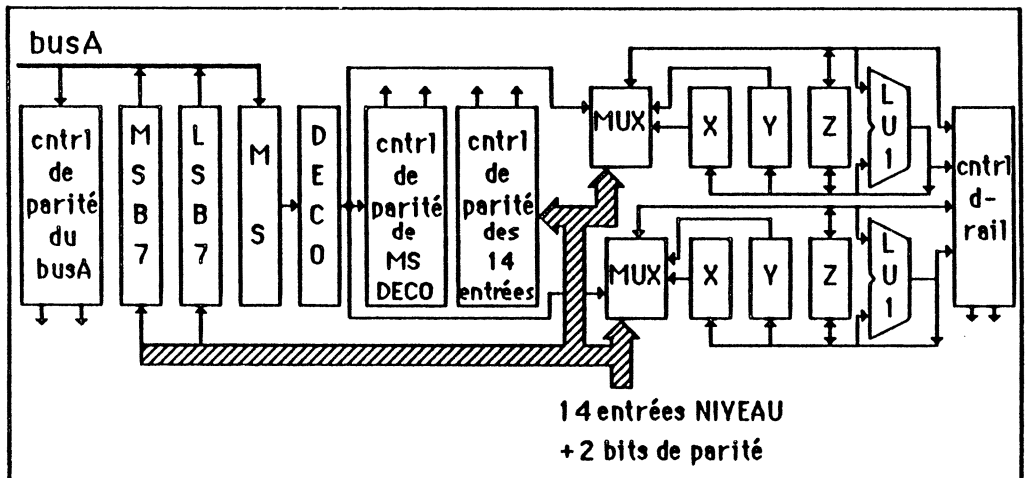


Figure 7 - Structure "Self-checking" de traitement des entrées NIVEAU

La figure 7 montre également deux registres : LSB7 et MSB7. Ces registres répondent aux instructions CIL et CIM respectivement. Chacune de ces instructions copient, via le registre tampon correspondant, 7 bits (7 entrées NIVEAU) et le bit de parité qui les code, dans l'accumulateur de l'unité arithmétique et logique (voir § 2.2.1), leur contrôle de parité se fait dans le contrôleur associé au bus interne busA.

L'ensemble MS, DECO, MUX est présenté à la figure 8 sous sa forme électrique. Le codage de MS et de DECO en parité est visible : un point mémoire supplémentaire pour MS, et une structure légèrement modifié pour DECO en additionnant un générateur de parité, pour reconstituer le code de parité des 4 bits d'entrée du décodeur, à sa sortie, et le comparer ensuite avec le bit de parité de MS.

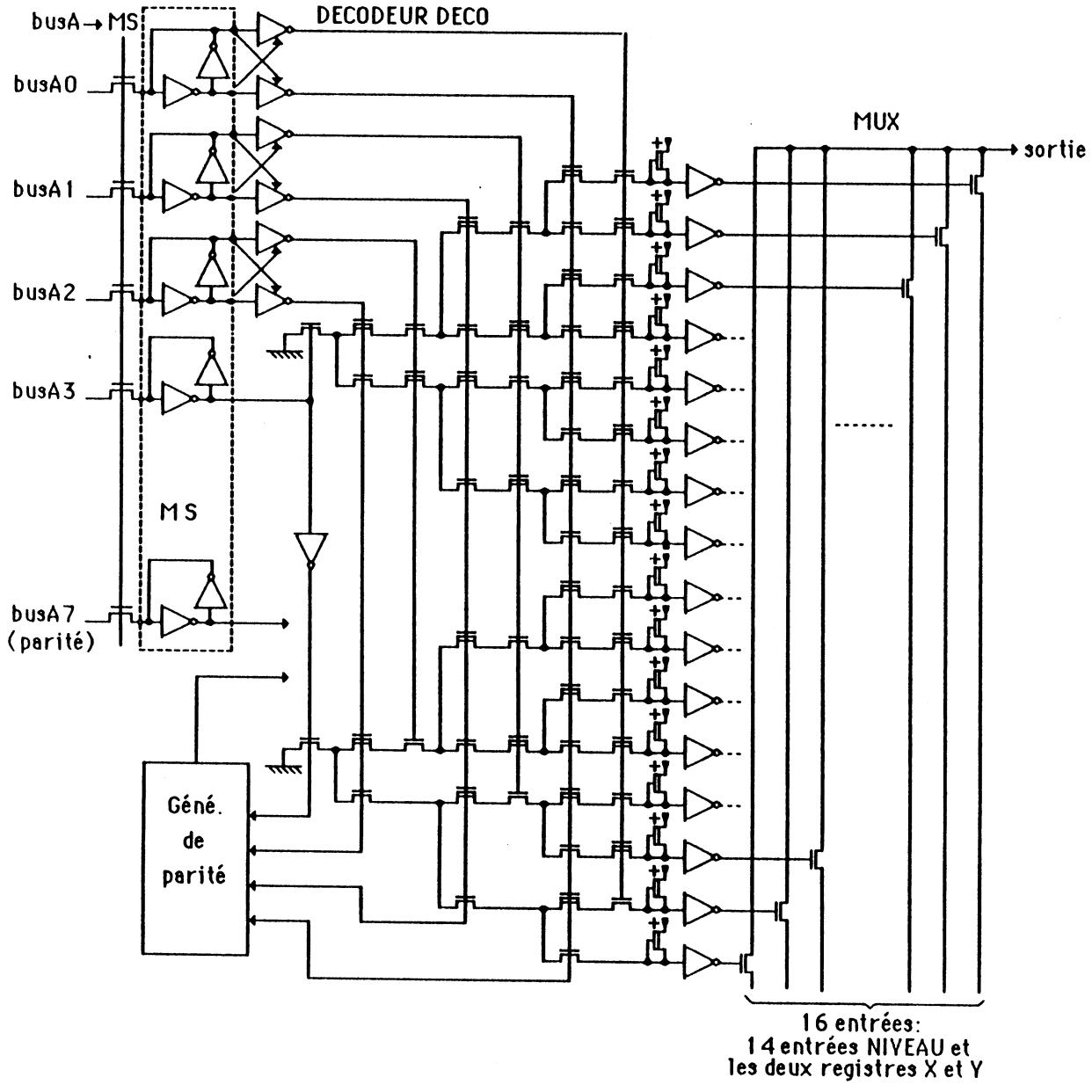


Figure 8 - Codage en parité de MS et de DECO et le codage de MUX en double-rai

Les registres X, Y et Z sont des points mémoires de 1 bits, et l'unité logique LU1 est une fonction combinatoire de ses deux entrées, mais également de 3 lignes de commande provenant de la partie commande, et qui permettent de choisir la fonction à appliquer aux deux entrées de LU1 (figure 9).

ⓐⓑⓒ	sortie	opération
0 0 0	$\overline{x \oplus y}$	norex
0 0 1	$x \oplus y$	ouex
0 1 0	$\overline{x + y}$	or
0 1 1	$x + y$	ou
1 0 0	$x . y$	and
1 0 1	$\overline{x . y}$	nand
1 1 0	1	non utilisés
1 1 1	0	

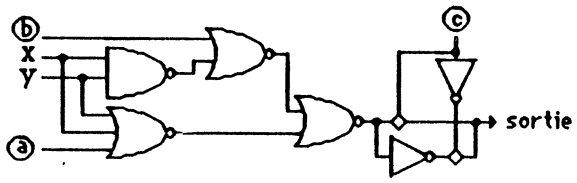


Figure 9 - Structure de l'unité logique LU1

L'opérateur LU1 à été réalisé et dessiné au micron par l'auteur, et a été rajouté à la bibliothèque de cellules (figure 10). L'annexe G explique les représentations des différents niveaux.

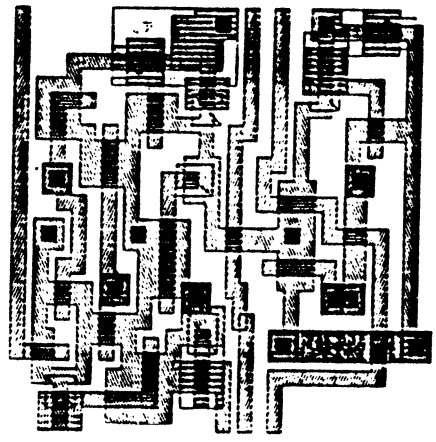


Figure 10 - Dessin au micron de LU1

2.3-5 Structure de traitement des entrées HORLOGE

Les problèmes spécifiques à résoudre par la structure des entrées HORLOGE se rapportent d'une manière générale à la détection d'un seuil de fréquence. la détection d'excés de vitesse est directement appliquée dans ce cas. Le principe de fonctionnement de cette structure est le suivant:

Les entrées HORLOGE sont traitées comme des entrées d'interruption à priorité. Quand une demande d'interruption est acceptée, alors la date correspondante est immédiatement comparée à la date de la demande d'interruption précédente (sur la même entrée), et le résultat de cette comparaison conditionne la suite du fonctionnement du processus.

Le datage des interruptions est effectué par les opérateurs CRi nommés registres auto-incrémentés. Ces opérateurs s'incrémentent à une fréquence choisie par l'utilisateur lors de l'initialisation du système. La figure 11 montre le principe de fonctionnement de cette structure.

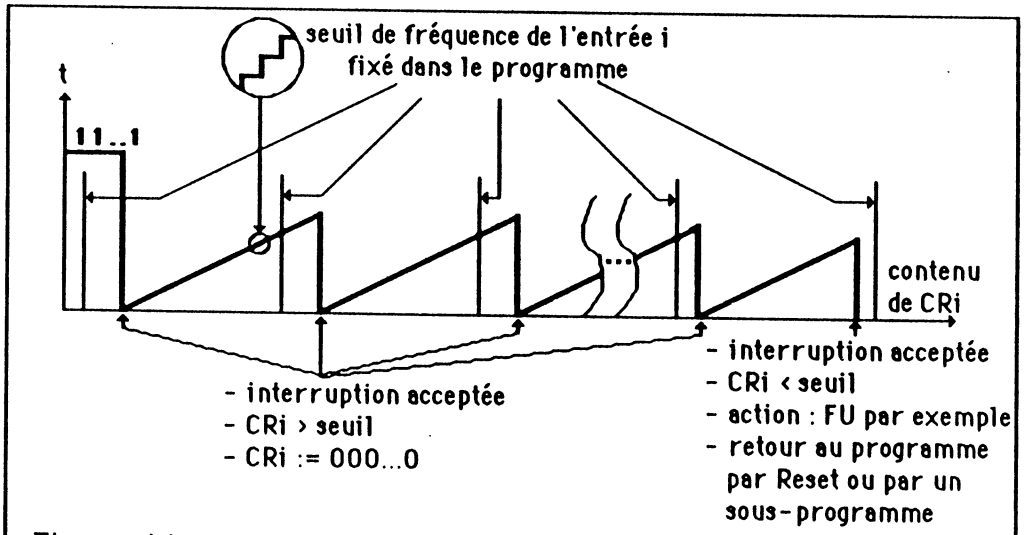


Figure 11 - Principe de fonctionnement des entrées HORLOGE

A l'initialisation du circuit, les 5 registres correspondant aux 5 entrées HORLOGE, sont mis à 11... 1. Mais dès qu'une entrée reçoit un front montant, alors le registre CRi correspondant est remis à zéro, puis commence à s'incrémenter à la fréquence demandée lors de l'initialisation. Si aucun front n'arrive à cette entrée, alors le registre CRi s'arrête de s'incrémenter, lorsqu'il atteint la valeur 11... 1 (d'après sa structure logique expliquée ci-dessous).

Le front suivant sur cette entrée provoque l'appel d'un sous-programme d'interruption, qui réalise la comparaison du contenu de CRi avec un seuil limite contenu dans ce sous-programme, et remet CRi à 00... 0. Le cycle recommence toujours, et pour chaque entée, indépendamment des autres.

Le sous-programme d'interruption peut être, pour un cas simple, le suivant :

```
CMP "seuil limite"
BLT "adresse"  (adresse du programme de freinage d'urgence par exemple)
RTI            (retour au programme principal si tout va bien)
```

Il existe des cas où il est nécessaire de prévoir plusieurs valeurs de seuil pour la même entrée, et ceci selon des modes précis (en fonction de la voie parcourue, ou à la suite d'un fonctionnnement anormal causé par un phénomène particulier). Pour cela, une certaine valeur d'un seuil doit être chargée en cours du fonctionnement, en vue de la comparer avec la valeur de la vitesse à cet instant (en terme de fréquence par le CRi correspondant).

Le mode en cours est déterminé par une entrée NIVEAU, et le choix du seuil en fonction du mode en cours serait fait dans chacun des 5 sous-programmes en testant les entrées NIVEAU et en utilisant les instructions de branchement BSZ et BSNZ, à l'intérieur du sous-programme d'interruption (avant le RTI). L'exemple suivant montre un cas où 3 seuils limites différents correspondent à une entrée HORLOGE:

```
-->     LZ E2       (premier mode déterminé par E2)
        BSZ adr 1
        LZ E4       (deuxième mode déterminé par E4)
        BSZ adr 2
        BSNZ adr 3
adr 1   CMP Seuil 1
        BLT adr FU
        BRA adr 4
adr 2   CMP Seuil 2
        BLT adr FU
        BRA adr 4
adr 3   CMP Seuil 3
        BLT adr FU
adr 4   RTI
```

Cet exemple est généralisable à n seuils et à m modes différents. La structure du registre CRi est montrée à la figure 11. Quand le contenu de CRi est 11... 1, alors la retenue sortante "carryout 7" est à "0", mais aprés l'inverseur rebouclé sur la retenue entrante "carry in 1", la retenue devient "1" et l'incrémentation est bloquée et le contenu de CRi reste à 11... 1 indépendamment des horloges H1, H2, et jusqu'à l'arrivée d'un second front sur l'entrée HORLOGE correspondante. L'intérêt de cette manœuvre est d'éviter le scénario suivant :

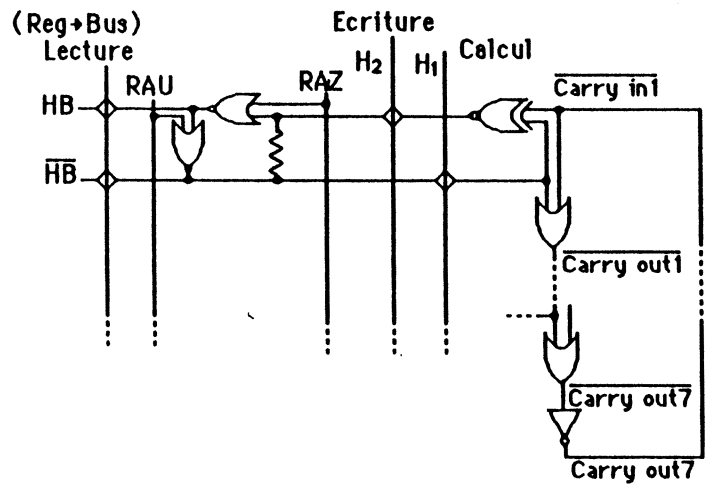


Figure 12 - Registre CRi avec le bloc d'incrémentation par le rebouclage de la retenue

Un front arrive sur une entrée HORLOGE. Le registre CRi correspondant est réinitialisé et s'incrémente normalement. Pas de second front jusqu'à la valeur 11... 1 dans CRi. S'il n'y a pas de blocage de l'incrémentation alors CRi passe à 00... 0 puis 00... 1 et continue son incrémentation. Si un front arrive, alors il va correspondre à une valeur bien inférieure à sa juste valeur, et la comparaison dans l'unité arithmétique et logique va indiquer un résultat faux, qui signifie dans le cas d'un train une survitesse alors que le phénomène est tout à fait normal. La figure 12 montre le dessin au micron de l'opérateur CRi, conçu et dessiné au micron par l'auteur, et rajouté dans la bibliothèque de cellules.

Les opérateurs CRi sont de 7 bits chacun donc s'incrémentent jusqu'à 128 fois (127 intervalles élémentaires). Or dans certains cas la gamme de fréquences à mesurer peut s'étendre de 4 Hz à 500 Hz, d'après le cahier des charges de COBRA. Alors une seule fréquence de l'horloge d'incrémentation ne suffit pas à couvrir cette gamme, et il est donc nécessaire de pouvoir changer de valeur de l'horloge en fonction des cas qui se présentent.

Le choix de 2 parmi 4 fréquences possibles, lors de l'initialisation du système (mot d'état dans la ROM à une adresse connue par la séquence d'initialisation, après un RESET ou lors de la mise sous tension), résoud ce problème. L'exemple suivant illustre ces propos :

Les quatre fréquences possibles sont :

11719, 2929, 732 et 183 Hz

ce qui correspond aux quatre intervalles élémentaires (entre deux incrémentations successives) suivants : 85, 341, 1366 et 5464 microsecondes.

Ces valeurs sont obtenues en divisant l'horloge de 3 MHz par :

256, 1024, 4096 et 16384, soit avec des bascules bistables au nombre de 8, 10, 12 et 14 bascules.

Il est possible de sélectionner deux de ces quatre valeurs, dont chacune pouvant être associée à une ou plusieurs des cinq entrées HORLOGE. Le mot d'état qui permet le choix de la fréquence est de la forme:

P	C1	C2	C3	C4	C5	Q1	Q2

P : parité

Q1, Q2 : choix de deux fréquences parmi quatre

C1, ..., C5 : choix de l'une des deux fréquences pour le registre 1,...,5.

Le tableau suivant montre la correspondance entre Q1, Q2 et le choix des fréquences

Q1	Q2	0		1
0	0	11719 Hz	et	732 Hz
0	1	11719 Hz	et	183 Hz
1	0	2929 Hz	et	732 Hz
1	1	2929 Hz	et	183 Hz

L'exemple suivant explique l'emploi de ce tableau :

soit le mot d'état :

1	1	0	0	1	0	0	1

Ceci signifie : CR2, CR3 et CR5 s'incrémentent à la fréquence d'horloge 11719 Hz, et CR1 et CR4 à la fréquence 183 Hz. Alors dans l'application, CR2, CR3 et CR5 permettent de mesurer des périodes allant de 85 microsecondes à 10,8 millisecondes, alors que CR1 et CR4 se compareront avec des périodes allant de 5,5 millisecondes à 694 millisecondes. La gamme totale converte ici, va de 85 microsecondes à presque 0,7seconde.

Le contrôle de la structure des opérateurs CRi est réalisé par le code de duplication en double-rail pour chaque incrémentation, et par le code de parité lors de la lecture de la valeur atteinte par le contrôleur de parité du bus B. la figure 14 montre la structure "self-checking" d'un opérateur telle que c'est intégré dans la partie opérative .

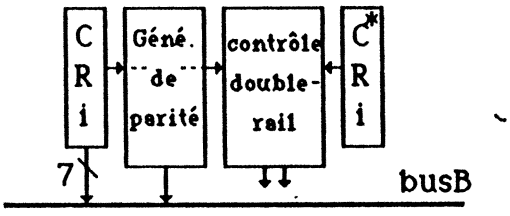


Figure 14 - Structure Self-checking d'un opérateur CRi

2.3.6 Structure de génération des sorties

7 broches de sortie sont disponibles et sont activées avec une instruction spéciale ACO (pour Activate Output). Ces sorties sont de type NIVEAU, si l'on fait la comparaison avec les entrées. Mais trois de ces sorties peuvent être de type HORLOGE, en présentant à l'extérieur une fréquence multiple de la fréquence d'horloge de COBRA. La figure 15 montre la structure des sorties, avec la logique combinatoire de choix NIVEAU/HORLOGE.

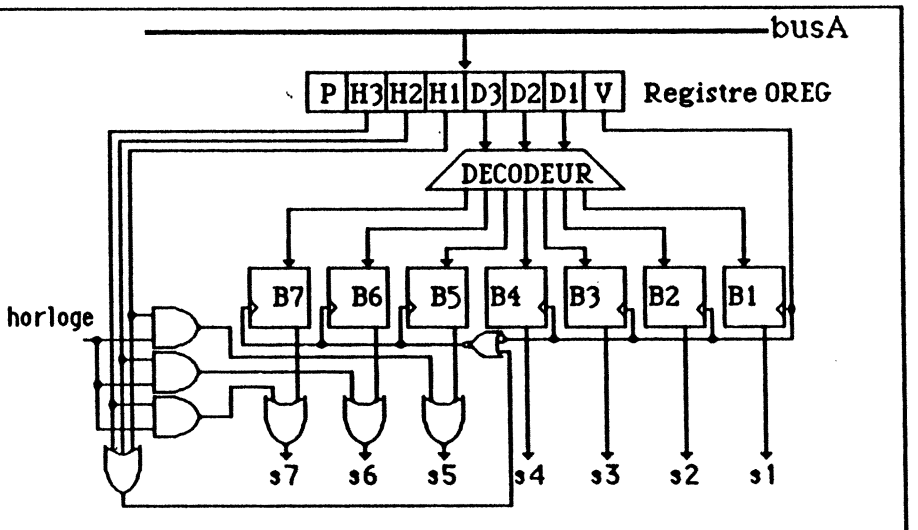


Figure 15 - Structure de génération des sorties

* V : valeur binaire (1 ou 0) à donner à la sortie sélectionnée
* H1, H2, H3 : validation en HORLOGE des sorties S5, S6 et S7.
  si H1 = H2 = H3 = 0 alors toutes les sorties sont de type NIVEAU.
* D1, D2, D3 : bits d'entrée du décodeur des sorties
* P : bit de parité

Le registre de sortie OREG reçoit l'opérande de l'instruction ACO qui représente un mot d'état contenant le numéro de la sortie à activer et la valeur d'activation (1 ou 0), pour les sorties de type NIVEAU. Le décodeur 3–>8 permet de choisir, une des 7 sorties à sélectionner, en validant une des bascules de sorties. Le bit V impose la valeur nouvelle à accorder à la sortie sélectionnée. L'exemple suivant décrit quelques possibilités de cette structure pour les opérandes suivants de ACO :

1	0	0	0	1	0	0	1	Forcer la sortie S4 à "1"
1	0	0	0	1	1	1	0	Forcer la sortie S7 à "0"
P	0	1	1	X	X	X	X	Forcer les sorties S5 et S6 à la fréquence d'horloge, indépendamment des valeurs des autres bits, X

La figure 16 montre la structure de génération des sorties munies de leurs codages et du contrôleur du registre de sortie OREG.

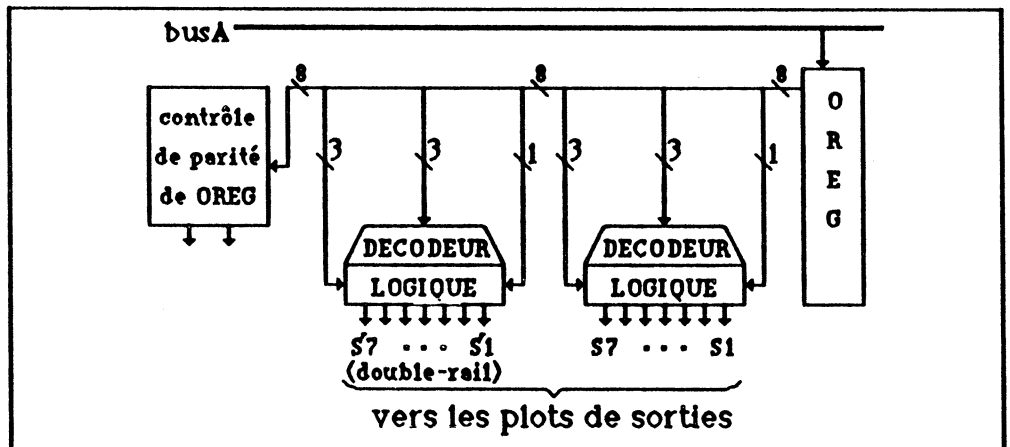


Figure 16 - Structure "Self checking" de génération des sorties

Le registre OREG n'est lu par aucun bus, son contrôle de parité n'est donc pas assuré. Alors il est nécessaire de rajouter un contrôleur de parité après la lecture de OREG, pour assurer la surveillance en cas de panne, tout au long des lignes de propagation de l'information. Par ailleurs les décodeurs de sortie et la logique associée, sont dupliqués et doivent être contrôlés en double-rail à l'extérieur.

2.3.7. Les Compteurs LFSR

Le LFSR (Linear Feedback Shift Register) est une structure basée sur un registre à décalage avec des rebouclages internes à travers des portes OU exclusif disposées d'une manière bien précise. Les LFSR sont souvent utilisés pour générer des vecteurs d'entrées d'un bloc à tester par l'analyse de la signature en sa sortie. Les LFSR sont également utilisés pour des fonctions de comptage, comme dans le cas présent. La figure 17 montre un exemple de fonctionnement d'un LFSR 3 bits :

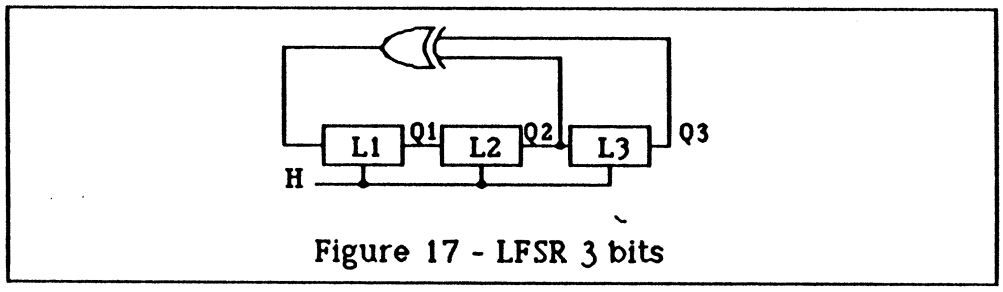


Figure 17 - LFSR 3 bits

L1, L2 et L3 sont trois bascules maître-esclave formant un registre à décalage 3 bits. En rebouclant à l'entrée de L1 la sortie du OU exclusif des sorties Q_2 et Q_3 de L_2 et L_3 on réalise un LFSR à 3 bits [WIL 80]. Cette structure permet de compter jusqu'à 7, la règle générale étant :

- pour n bascules on compte jusqu'à 2^n-1

l'état soustrait ici est appelé état absorbé, et il correspond au cas où la valeur initiale de L1, L2 et L3 est le zéro. Dans ce cas on reboucle indéfiniment la valeur zéro à chaque impulsion d'horloge H. Toutes les autres combinaisons permettent de compter des états au nombre de 7. Pour les LFSR de taille plus grande, il existe des tables qui permettent de déterminer la configuration et la structure des portes OU exclusif à réaliser.

Dans le cas du circuit COBRA, le LFSR qui correspond à ses spécifications est de 14 bits. Trois compteurs de ce type sont utilisés dans COBRA, et se nomment CNT. La figure 18 montre le cas du LFSR dans le cas de CNT :

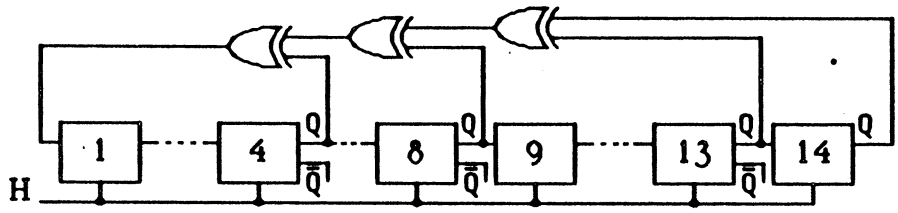


Figure 18 - LFSR 14 bits , cas des compteurs CNT

Dans ce cas, le LFSR compte jusqu'à 2^{14}-1 = 16383. Donc pour H=732Hz la temporisation peut aller jusqu'à 22 secondes.

Une instruction spéciale, TMP, permet de charger en deux fois la valeur à décompter dans les bascules du CNT correspondant. Quand la valeur correspondant à la fin de décomptage est atteinte, alors une demande d'interrruption est envoyée, et le vecteur d'interruption correspondant pointe sur l'adresse d'un sous-programme dans la ROM, qui est le sous-programme d'interruption correspondant au CNT.

Le sous-programme d'interrupiton pourrait être une succession d'instructions de temporisation et d'activation de sorties, ce qui permet de générer toute sorte de séquence, directement et sans interface particulier.

L'exemple suivant montre deux cas possibles :

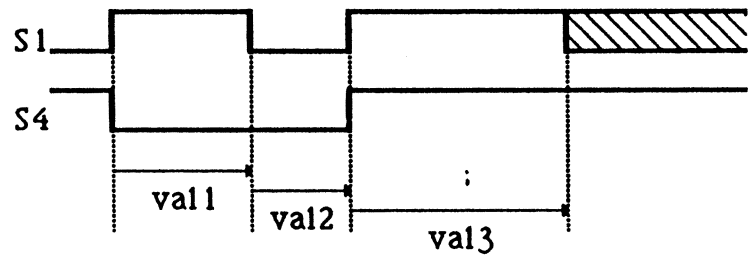
```
*     ACO   $83      la sortie S1 forcée à 1
      ACO   $88      la sortie S4 forcée à 0
      TMBP  "val1"   une temporisation de durée val1 est lancée avec CNTB
```

A la fin de la temporisation, un sous-programme d'interruption est appelé :

```
      ACO   $82      la sortie S1 forcée à 0
      TMPA  "val2"   une temporisation de durée val2 est lancée avec CNTA
      RTI
            et ensuite

      ACO   $83      S1 remis à 1
      TMPB  "val3"   une 3ème temporisation val3 est lancée avec CNTB
      ACO   $89      S4 remis à 1
```

Les signaux obtenus sont alors les suivants :



* le 2ème exemple est le bout de programme suivant :

```
        LZ      E3
        OZ      E9
        BSNZ    "adr1"      si E3+E9=1 alors un signal périodique est envoyé de la
                            sortie S5 vers l'extérieur après une attente de "val1".
adr1    TMPA    "val1"
        ACO     $90         Sortie S5 en sortie HORLOGE.
        RTI
```

Plusieurs autres cas sont possibles notamment le cas où un problème quelconque ne serait pas dangeureux s'il ne persiste pas plus qu'une certaine durée. Dans ce cas une temporisation de cette durée permet d'éviter de bloquer le système et ainsi de conserver sa disponibilité.

Les LFSR ont un inconvénient dans leur fonction de comptage car celui-ci ne s'incrémente pas à la manière du code binaire (001–010–011...), mais utilise un code particulier où la correspondance avec le code binaire (ou décimal) est donnée dans des tables, si la valeur de l'origine de comptage est fixé. Mais, une fois la correspondance entre ces codes est donnée, le problème deviendrait alors transparent et l'utilisation serait identique.

L'avantage important des LFSR, c'est qu'ils peuvent être contrôlés avec le code le plus économique, qui est le code de parité. Mais pour cela, la structure de la figure 18 serait augmentée de deux bascules, qui supporteront les codes de parité des 14 autres, ainsi que trois portes OU exclusif qui calculeront ces parités. Le montage de la figure 19 est expliqué dans la suite :

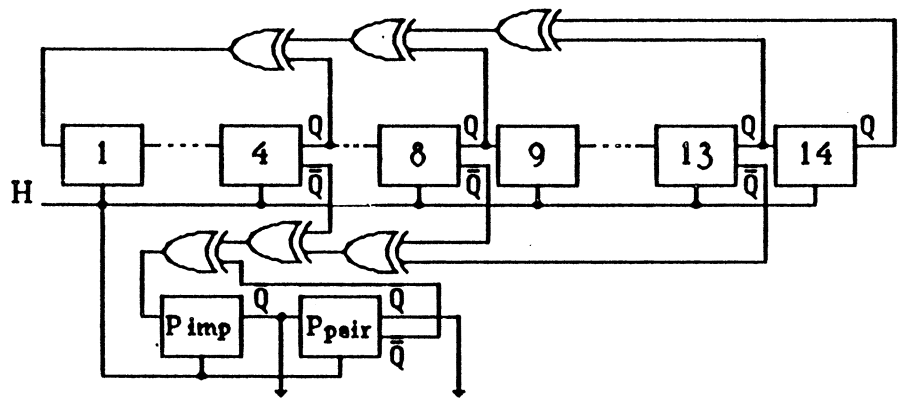


Figure 19 - LFSR 14 bits avec les deux bits de parité paire des bascules d'ordre pair et impair

Cette structure a été dessinée au micron par l'auteur, pour être implantée dans la partie opérative, selon le format standard de "bit slice", afin de permettre tout transfert entre le bus interne et le compteur type LFSR (ainsi que les codes de parité associés).

La bascule de base est rajoutée dans la bibliothèque de cellules. Il s'agit d'une bascule D maître-escalve (figure 20), dans laquelle l'écriture de 1 ou 0 (initialisation), se fait par l'une des entrées S (Set) ou R (Reset), tandis que la propagation de la valeur incrémentée se fait par l'entrée D.

Le contrôle de parité des LFSR serait suffisant à condition d'éviter les court-circuits entre deux bascules d'ordres consécutifs dans la structure du bus de la partie opérative dans laquelle les LFSR sont incorporés, et ceci en raison de la règle R3 de conception des circuits SFS. Pour cela, les cellules d'ordres pairs sont placées sur le bus en tant qu'un opérateur 8 bits (avec la parité), et les cellules d'ordres impairs sont placées en tant qu'un autre opérateur 8 bits.

La figure 21 montre le placement des cellules dans la partie opérative de COBRA. L'instruction relative au chargement d'un compteru CNT, chargera d'abord les bascules d'ordres impairs et ensuite les bascules d'ordres pairs.

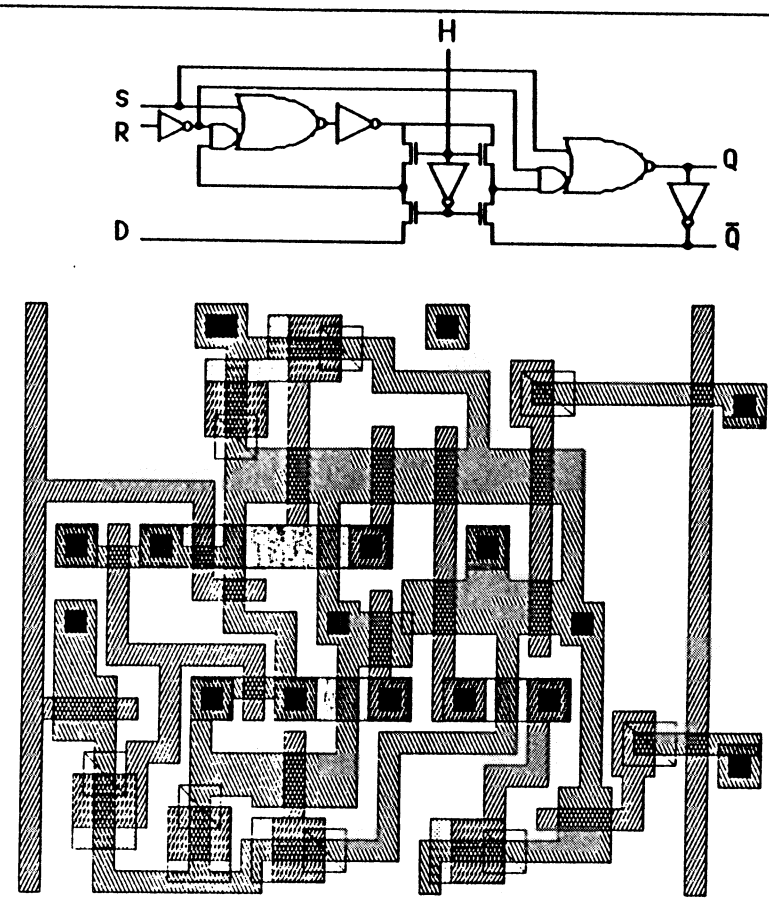


Figure 20 - Bascule D utilisée dans les compteurs CNT et son dessin au micron

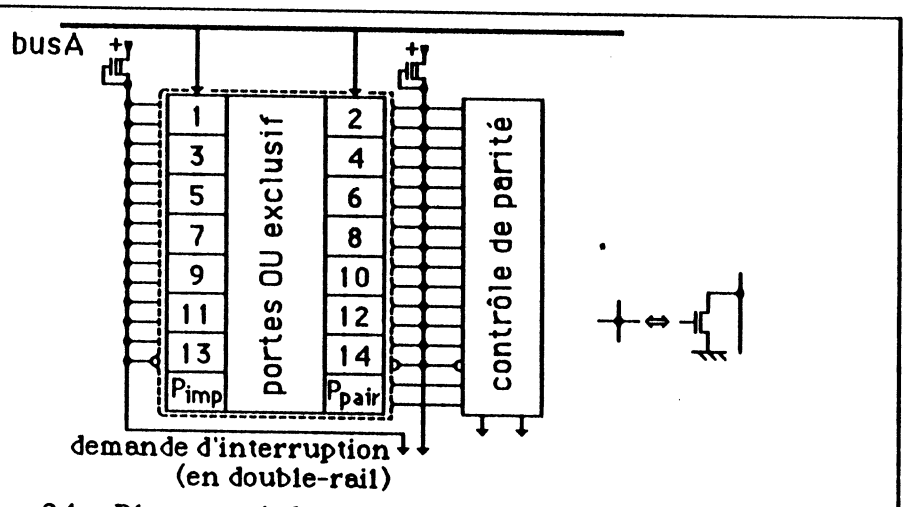


Figure 21 - Placement des cellules du LFSR 14 bits et contrôle de lecture et de la demande d'interruption

La demande d'interruption en fin de décomptage est faite par un décodeur simple (porte NOR) dupliqué pour contrôler cette demande. La figure 21 montre le cas de CNT avec la valeur 00...01 comme la valeur correspondant en fin de décomptage et donc à la demande d'interruption.

La validation du choix du contrôle des LFSR est détaillée dans [NIC 86-2].

2.3.8 - La mémoire vive "self-checking"

La RAM de 64 octets de COBRA possède deux fonctions dans COBRA :

- le stockage à tout moment de mots de 8 bits, et la lecture par deux instructions LDM (lecture de la RAM) et STM (écriture dans la RAM).
- la sauvegarde lors d'une interruption des registres de travail : fonction de pile LIFO (Last In First Out).

Dans le premiers cas, un registre de sélection (RS) communique l'adresse vers la RAM. Un pointeur de pile (SP) gère la pile. Le décodeur d'adresse de la RAM (RAMDEC) est divisé en deux parties : décodeur lignes et décodeur colonnes. Mais pour diminuer la taille des contrôleurs des décodeurs, la RAM sera découpée en 8 blocs matriciels (16 x 4 bits) chacun. Le décodeur ligne 4–>16 sélectionne une ligne parmi 16, et le décodeur colonne sélectionne une colonne parmi 4 dans chacun des 8 blocs. Les 8 bits ainsi pointés sont l'octet à lire ou à écrire dans le bus interne. Cette structure de la RAM (figure 22) est munie des contrôleurs des décodeurs et des adresses. En effet, les 4 bits d'adresse-ligne sont recomposés et contrôlés en double-rail après avoir sélectionné la ligne. Les 2 bits d'adresse-colonne sont contrôlés directement en double-rail. Ensuite le bit de parité des 6 bits d'adresse est reconstitué puis comparé avec le bit de parité contenu dans RS.

Cette structure de la RAM utilise un point mémoire de la bibliothèque des cellules, mais qui a été redessiné par l'auteur et compacté à 85 % de la surface de la cellule de départ.

Les contrôleurs de la RAM sont réalimentés par les lignes VDD et VSS qui assurent l'alimentation des blocs de la RAM par un peigne d'alimentation (figure 23), ce qui respecte les règles de circuits SFS pour le code détectant les erreurs simples.

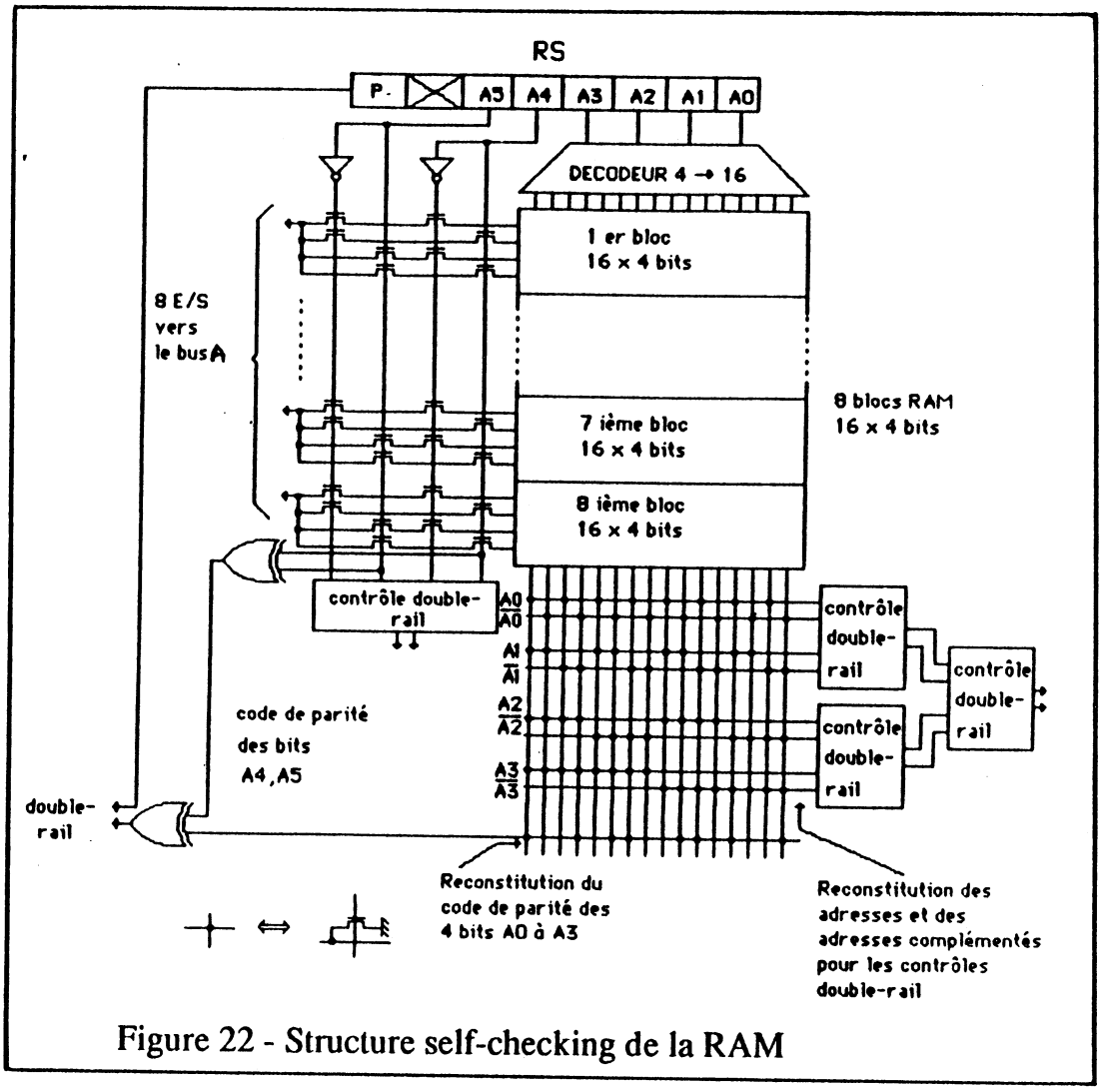


Figure 22 - Structure self-checking de la RAM

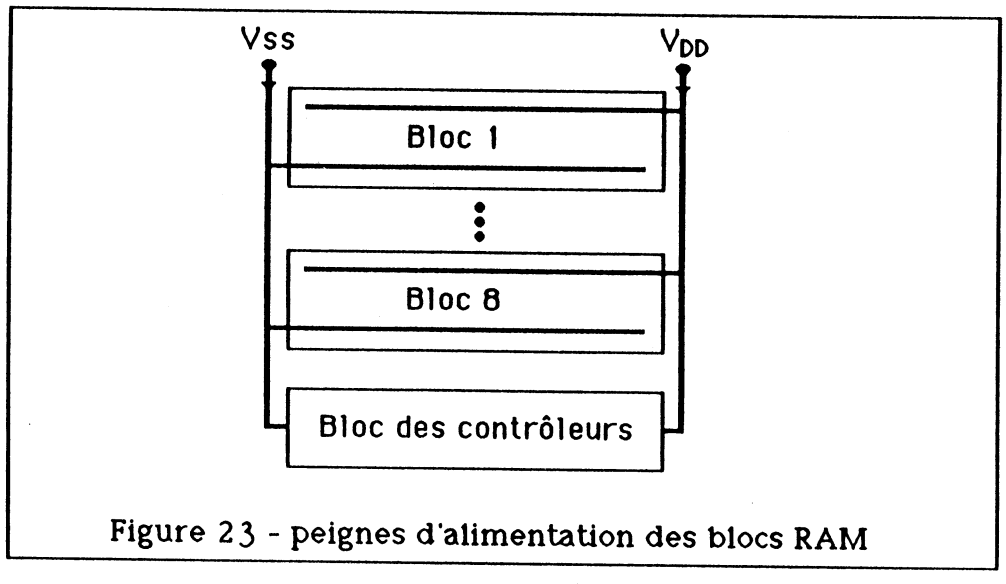


Figure 23 - peignes d'alimentation des blocs RAM

2.3.9 - La liaison série

L'émission et la réception série se font dans une sous-partie opérative dont le fonctionnement est indépendant du reste du circuit. Une instruction spéciale "SERT" demande l'émission d'un octet, et une demande d'interruption annonce la réception d'un octet. L'octet transmis est contenu dans le registre accumulateur (voir § 2.3.10.2).

La sous-partie opérative se compose : d'un registre de transmission, TR, un registre de réception RR et un registre à décalage SR.

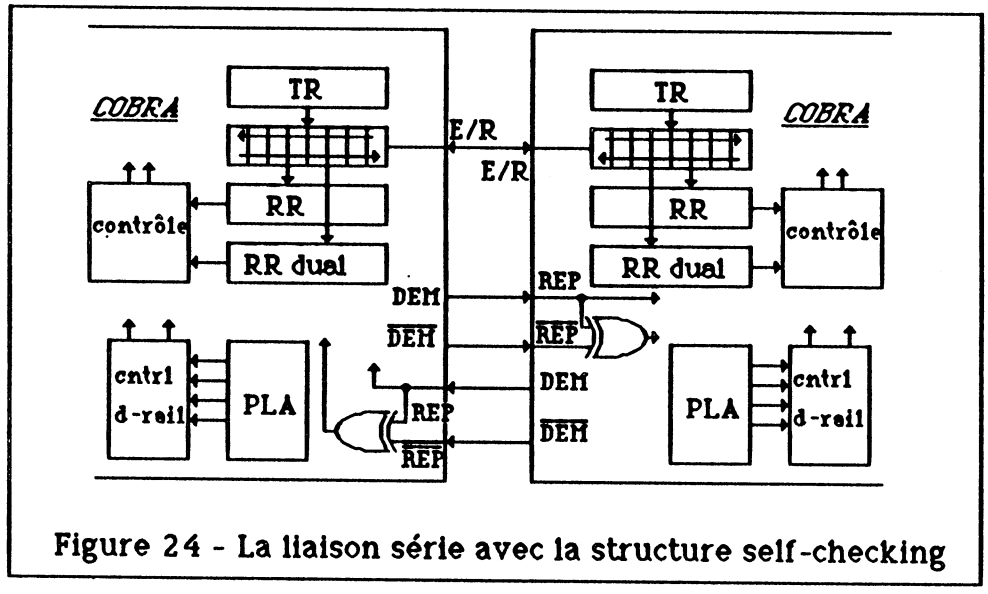
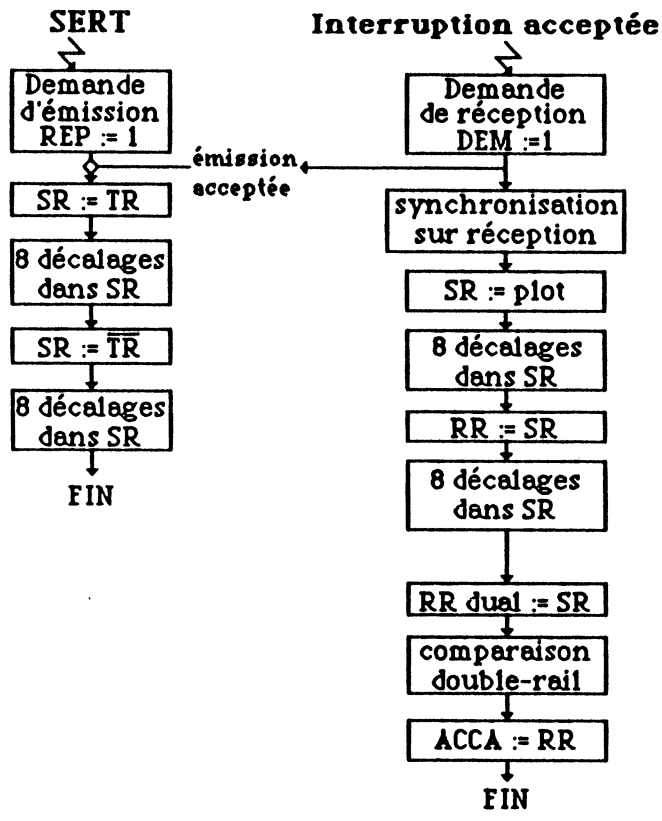


Figure 24 - La liaison série avec la structure self-checking

La figure 24 montre la structure utilisée pour la liaison série. Cette structure peut être assimilée à une machine relativement indépendante du fonctionnement du reste du circuit. Elle est activée lors de la validation par la partie commande principale de l'un des deux algorithmes, un pour l'émission, et l'autre pour la réception, contenus tous les deux dans un PLA. Les bascules l et tm indiquent les disponibilités de SR et TR pour les besoins des séquencements des transmissions. Elles sont dupliquées et contrôlées en double-rail.



- principe des émissions et réception série dans COBRA -

Le message transmis par la ligne unique est codé en parité paire, mais de plus il est complémenté et réémis, via SR, et stocké à la réception dans le registre RR puis RRdual et ensuite le contrôle double-rail indique s'il y a coupure de la ligne série.

Les paramètres de transmission série sont fixés à : format de 7 bits plus un bit de parité paire, la vitesse de transmission est de 62500 bits/sec.

La demande d'émission et la réponse à cette demande sont dupliquées sur deux lignes (DEM et ¬DEM), et contrôlées à la réception (REP et ¬REP) par une porte OU exclusif qui signale les pannes sur ces lignes.

Les pannes dans le registre TR sont signalées par le contrôleur de parité du bus à travers lequel se fait le transfert TR->SR. Les registres RR et RRdual sont contrôlés en double-rail, ce qui couvre donc toutes les pannes internes aux deux registres, ainsi que les pannes qui pourraient provenir du registre SR (lors des transferts SR->RR et SR->RRdual), ou bien les pannes de la ligne série même.

Le registre SR (shift registre) est un registre à décalage, qui doit conserver la propriété du codage en parité des 7 bits d'informations. En effet, un registre à décalage classique, n'est pas tenu de calculer la parité lors des décalages successifs qu'il subit.

Alors, un générateur de parité, prenant en compte le bit sortant lors du décalage, calcule la parité après chaque décalage, et permet ainsi de contrôler le registre à décalage en utilisant le code de parité. La figure 25 illustre le fonctionnement d'une telle structure. Le générateur de parité est une porte OU exclusif rebouclée sur un point mémoire qui contiendrait la parité réactualisée.

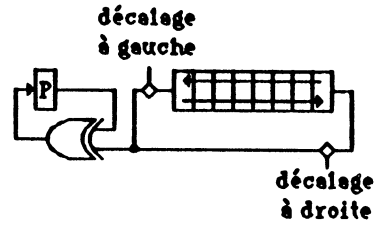
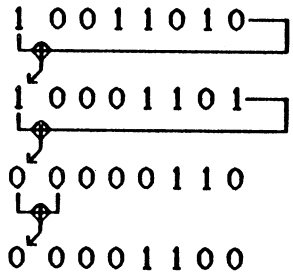


Figure 25 - Registre à décalage avec calcul de parité

. Le fonctionnement du registre à décalage avec parité est le suivant :
soit au départ, le mot copié dans le registre à décalage :



un décalage à droite donne:

un deuxième décalage à droite:

un décalage à gauche donne:

A chaque décalage, un contrôleur de parité, en sortie de SR, surveille le fonctionnement, et signale la première occurrence de panne.

Le registre à décalage utilisé dans COBRA provient de la bilbliothèque de cellules. L'adjonction de la structure de calcul de parité a occasionné l'adaptation de la structure du "shifter" de la bibliothèque pour constituer un registre à décalage avec calcul de parité immédiat, de 8 bits. Ce même registre à décalage est utilisé également à un autre endroit de la partie opérative de COBRA, pour répondre aux deux instructions de décalage à gauche, LSL, et à droite, LSR, (§2.3.10.1).

2.3.10 - Les opérateurs et registres de travail

Ce sont les opérateurs et registres suivants :

- l'unité arithmétique et logique
- le registre à décalage et l'accumulateur
- les registres compteurs de programme
- l'incrémenteur/décrémenteur
- le registre de branchements conditionnels
- le registre d'instructions.

2.3.10.1 - L'unité arithmétique et logique l'UAL

L'UAL exécute des opérations arithmétiques et logiques, 7 bits, et calcule la parité du résultat. Le bit de parité calculé devient le huitième bit de l'octet, au format standard utilisé dans COBRA.

Les opérations exécutées dans l'UAL sont :

- l'addition, la soustraction et la comparaison (ADDA, SUBA, et CMPA)
- les opérations logiques : ET, OU, fonction complément (ANDA, ORA et COM)
- la décrémentation (DECA).

Quatre lignes de commande sélectionnent l'opération à effectuer dans l'UAL. Ces quatre commandes sont déduites du code opération de l'instruction décodée, et sont générées par la partie commande.

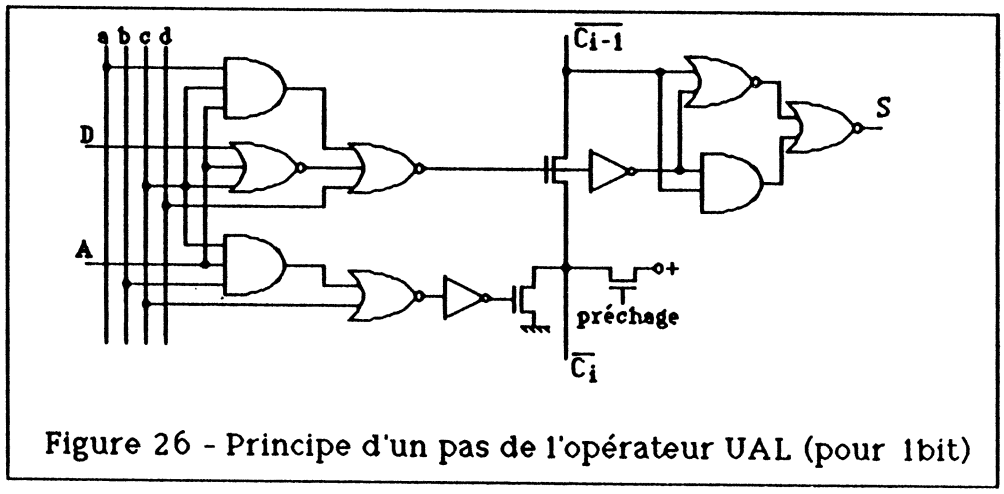


Figure 26 - Principe d'un pas de l'opérateur UAL (pour 1bit)

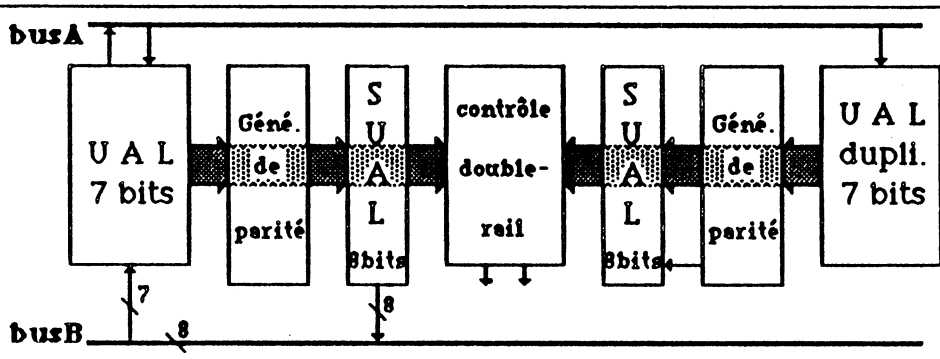


Figure 27 - Duplication de l'UAL et contrôle self-checking

Le générateur de parité est transparent aux lignes qui vont de l'UAL vers le contrôle double-rail (figure 27). Le registre de sortie de l'UAL, SUAL reçoit les 7 bits d'information provenant de l'UAL, et le bit résultant du calcul de la parité. La deuxième UAL est la duplication de l'UAL principale, et sert uniquement pour le contrôle.

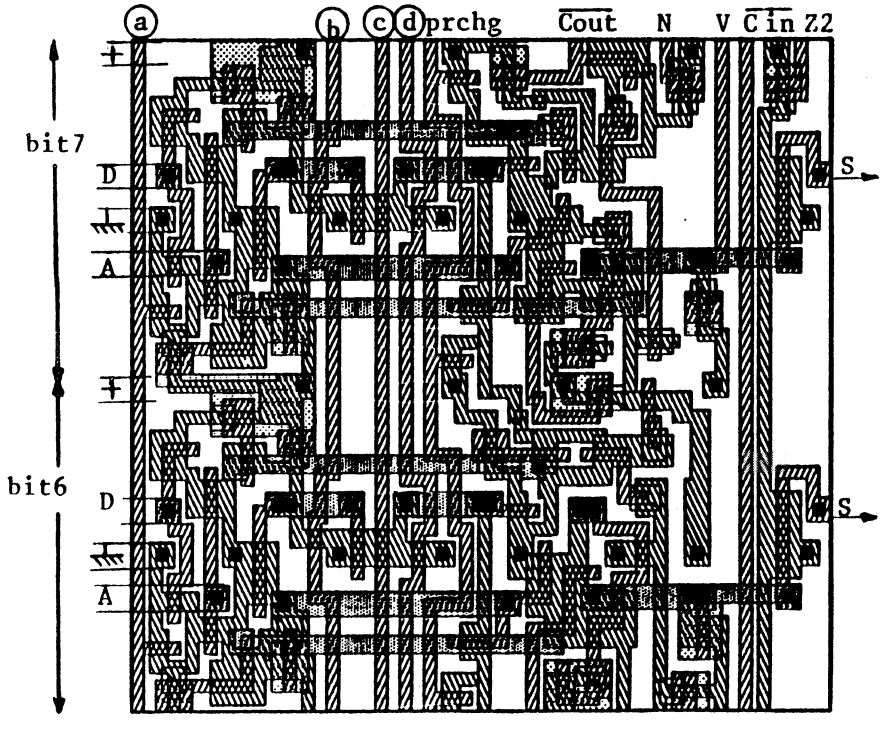


Figure 28 - Dessin au micron de deux "bit-slice" de l'UAL

2.3.10.2 - Le registre à décalage et l'accumulateur

Le registre à décalage est le même que celui de la liaison série (figure 25). Il répond aux instructions de décalage à gauche, LSL et à droite, LSR, du contenu du registre accumulateur, ACCA, (figure 29).

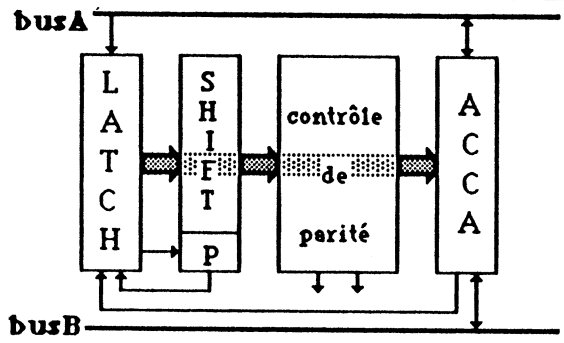


Figure 29 - Le registre à décalage, l'accumulateur et leur contrôle

En effet, le registre accumulateur ACCA, est celui dans lequel se trouvent tous les résultats de l'UAL. C'est dans ACCA que l'instruction d'émission série, SERT, cherche l'octet à émettre. C'est également dans ACCA que les instructions STM et LDM (dialogue avec la RAM), écrivent et lisent les octets dans la RAM. L'instruction, LOAD, charge dans ACCA le contenu de son opérande. L'instruction DEC, réalise la décrémentation du contenu de ACCA, en utilisant le LATCH de la figure 29 comme support de la valeur intermédiaire.

2.3.10.3 - L'incrémenteur et les compteurs de programme

Deux registres gèrent la liaison avec la ROM externe : PCH et PCL, compteur de programme poids fort, et poids faible. En total 14 bits pour adresser les 128 pages de 128 octets possibles (§ 2.2.4). Ces registres sont codés en parité comme tous les autres registres de la partie opérative. Mais, de plus le registre PCL, est dupliqué en raison de son accès en lecture par l'opérateur INC1, qui est un incrémenteur/décrémenteur. En effet, le transfert PCL vers les tampons d'entrée de INC1, est direct et ne passe pas par les bus. De ce fait les contrôleurs de parité des bus ne peuvent pas assurer le contrôle de PCL. Ce problème est posé également pour le registre pointeur de pile, SP. Alors les deux registres PCL et SP sont dupliqués et contrôlés en lecture par le contrôleur double-rail de la structre montrée à la figure 30.

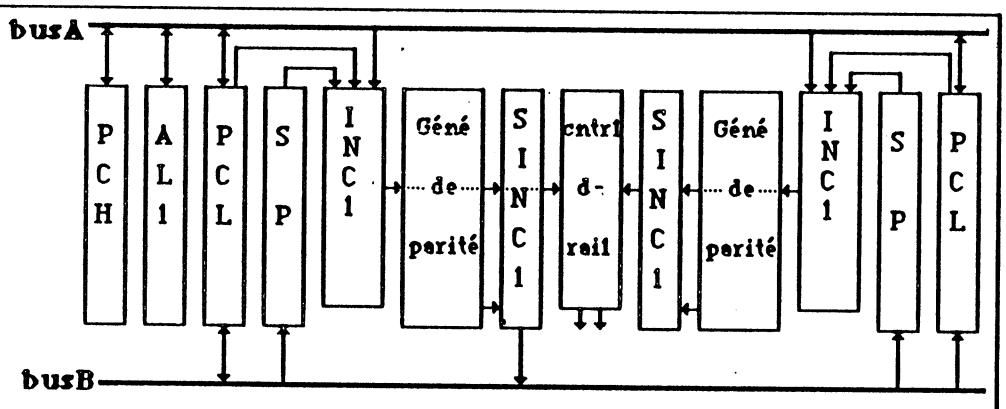


Figure 30 - Structure self-checking de l'incrémenteur et des registres qui l'utilisent

L'opérateur INC1 a deux fonctions, qui sont l'incrémentation et la décrémentation. L'incrémentation est appliqué pour :

- PCL, à chaque lecture d'un octet dans la ROM externe, à l'intérieur d'une page (§2.2.4), et pour les branchements courts.

- PCH, à chaque branchement long

- AL1, utilisée pour stocker le poids fort d'une adresse lors d'un branchement long conditionnel.

- SP, pointeur de pile, lors de l'empilement de registres à sauvegarder après une interruption.

La décrémentation est appliquée pour le retour d'interruption (instruction RTI) pour récupérer les registres sauvegardés dans la pile.

Mais l'opérateur d'incrémentation de la bibliothèque des cellules réalise uniquement l'opération d'incrémentation. Alors, des modifications ont été aportées aux cellules composant l'incrémenteur par l'auteur, en rajoutant trois transistors, permettant de choisir une des deux fonctions avec deux commandes. Cet opérateur est rajouté à la bibliothèque de cellules.

La représentation synoptique de la partie opérative autotestable est donnée dans l'annexe F.

2.3.11 - La partie commande

Le choix de la partie commande de COBRA est justifié dans le paragraphe 2.4, décrivant la génération des contenus des PLA de la partie commande de COBRA par l'outil MACSIM, selon le principe de la partie commande à générateur de temps. Les deux PLA (propriétés et commandes), le générateur de temps, et un PLA de paramétrisation (pour les branchements conditionnels), composent cette partie commande (figure 31).

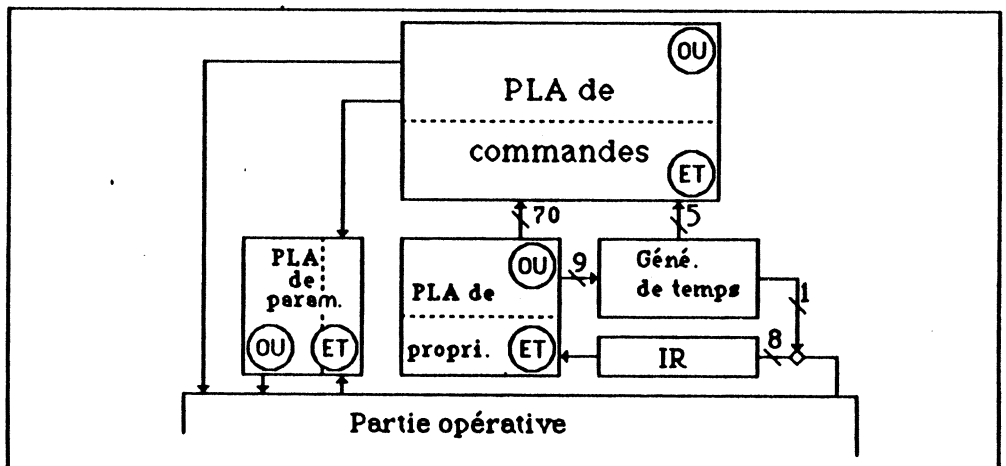


Figure 31 - Schéma fonctionnel de la partie commande

Le registre d'instruction IR reçoit, à la fin de l'exécution de chaque instruction, le code opération de l'instruction suivante, et le communique au PLA de propriétés, qui le décode et communique au PLA de commandes et au générateur de temps, les propriétés de ce code opération qui sont les propriétés statiques et les instants (§2.4.1). Le PLA de commandes (synchronisé par les séquences que lui fournit le générateur de temps à partir des instants) fournit les commandes correspondant à l'instruction décodée.

Le rôle du PLA de paramétrisation se limite aux instructions de branchements conditionnels. Ce PLA reçoit les compte-rendus de l'unité arithmétique et logique (Z2, V, N et C) ainsi que le compte-rendu de l'unité logique, 1 bit, qui est Z1. En fonction de ces bits indicateurs les conditions de branchement seront "vraies" ou "fausses".

L'exemple de BSZ de transfert conditionnel dans la description IRENE :
(busA := if bitZ1 =1 then AL1 else PCL endif) signifie

si Z1 = 0 alors le bus A prendra la valeur contenue dans PCL

si Z1 = 1 alors le bus A prendra la valeur contenue dans AL1.

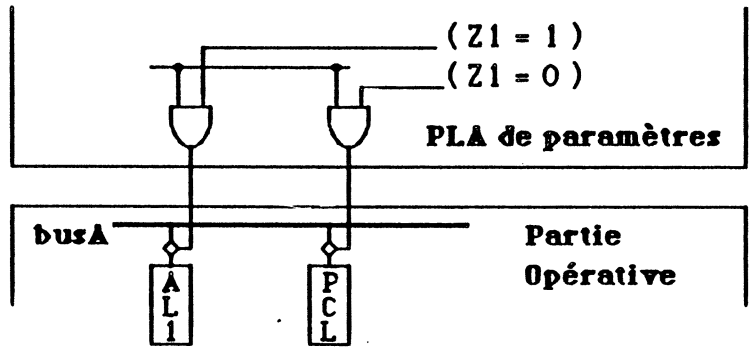


Figure 32 - Fonctionnement du PLA de paramétres

Le PLA de paramétrisation permet de diminuer le nombre de signaux que doit générer le PLA de commandes, ainsi que le nombre d'étapes de séquencement.

La détection des pannes de la partie commande est assurée par des codages différents de chacun de ces éléments. Le registre d'instruction IR est un des éléments de la partie opérative, en ce sens qu'il est au format du "bit slice" et qu'il est connecté sur le bus de la partie opérative, en lecture et écriture. IR est donc codé en parité, et son contrôle se fait lors de sa lecture, comme pour tous les autres registres de la partie opérative, dans les contrôleurs de parité des bus de transfert (bus A ou B). Mais IR est également lue par le PLA de propriétés : le code opération lu dans IR par la matrice ET de ce PLA est également codé en parité, alors un contrôleur de parité placé à la sortie de la matrice ET du PLA de propriétés assure le contrôle de parité de IR (figure33).

Le générateur de temps doit être dupliqué et contrôlé en double-rail à la sortie de la matrice ET du PLA de commandes, et après la commande de sélection de IR en écriture. Par ailleurs, le code d'entrée du générateur de temps est un code "1 parmi 9", en fonction duquel, le générateur de temps fournit les séquences "cycle" de chaque instruction décodée. Ce code est défini par MASCIM lors de la génération du contenu des PLA. Mais il se trouve que ce code peut également servir pour détecter les erreurs unidirectionnelles pouvant survenir sur les 9 lignes de code jusqu'à la sortie du générat"eur de tmps. En effet ce code est un code non ordonné (comme le code de Berger), et donc un contrôleur 1 parmi 9 permet de détecter les erreurs unidirectionnelles dans ce secteur de la partie commande.

Une structure identique est également appliquée pour la reconstitution et le contrôle des adresses de la RAM, en utilisant un code 1 parmi 16 (figure 22 du §2.3.8). La figure 34 montre le principe du contrôleur 1 parmi 9.

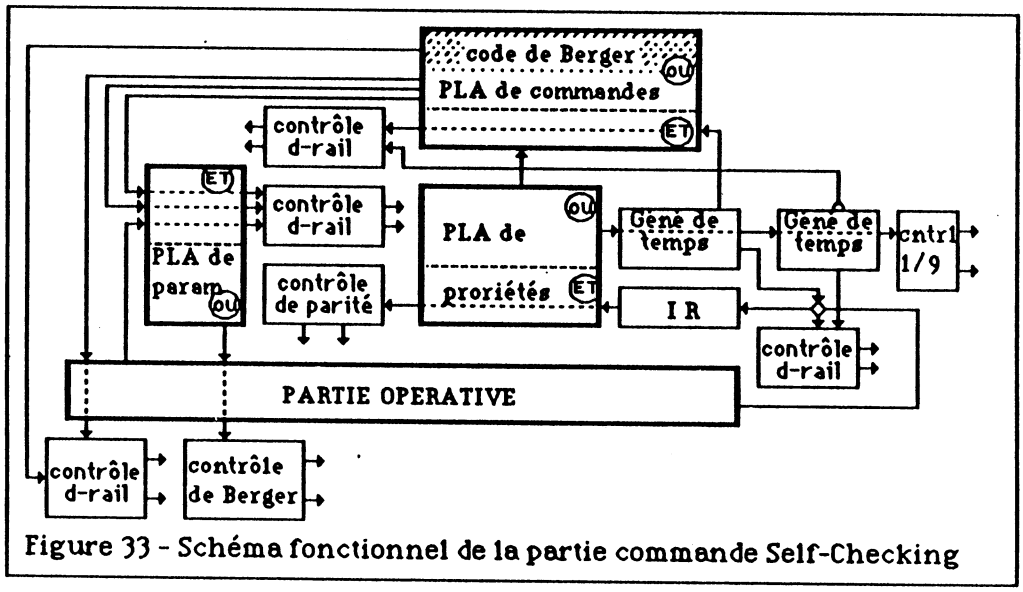


Figure 33 - Schéma fonctionnel de la partie commande Self-Checking

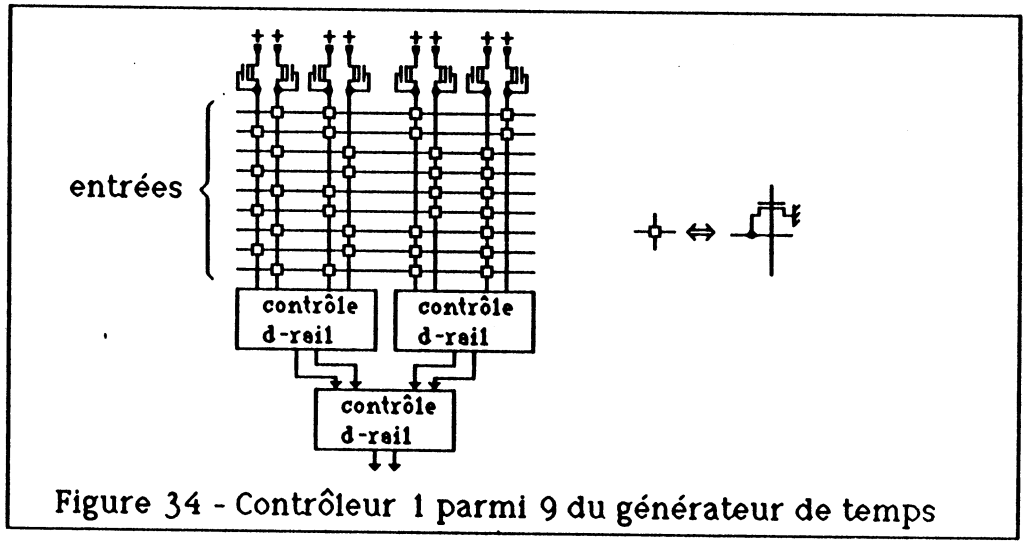


Figure 34 - Contrôleur 1 parmi 9 du générateur de temps

Les PLA sont codés avec le code de Berger et contrôlés par comparaison du code avec les sorties des contrôleurs de Berger. Le code pour un PLA est introduit au moment de l'établissement de la table de vérité de ce PLA. La table de vérité est la correspondance entre les vecteurs de sortie et les vecteurs d'entrée. Ainsi pour chaque vecteur d'entrée, le vecteur de sortie correspondant est augmenté de son code de Berger (§4.5.2 du chapitre 2). Le PLA généré à partir de la table de vérité augmenté du code de Berger est un PLA SFS. Seule la matrice OU du PLA est modifiée car c'est celle qui contient les vecteurs de sortie.

Pour MACSIM, qui génère les PLA de COBRA, cette étape de codage des matrices OU (qui contiennent les vecteurs des sorties), est facilitée. En effet, il est possible de rajouter le code de Berger à chacun des vecteurs de sortie de la matrice OU, un par un, lors de l'établissement de la table de vérité du PLA, dans le dernier programme de MACSIM (le programme SYNTHESE). Cette opération a été réalisée uniquement pour le PLA de commandes et pas pour les autres. La raison est que les PLA présentent une parité d'inversion, entre les entrées et les sorties primaires, égale à 0 (§5 chapitre 2). Or, MACSIM génère les deux PLA (commandes et propriétés), de sorte que le PLA de commandes reçoit les sorties du PLA de propriétés sans aucune inversion. La parité d'inversion est donc inchangée et le code de Berger reste valable, i.e., comme s'il s'agissait d'un bloc composant les deux PLA et dont les entrées primaires sont celles du PLA de propriétés et les sorties primaires sont celles du PLA de commandes.

Si cela n'avait pas été le cas, il aurait fallu augmenter le PLA de propriétés du code de Berger correspondant, et rajouter les contrôleurs de Berger de ce PLA à la sortie de la matrice ET et du PLA de commandes. Les contrôleurs dans ce cas auraient occupé une surface importante.

Le PLA de commandes doit générer également les commandes servant aux blocs dupliqués de la partie opérative. Ces commandes sont codées en double-rail et seront alors contrôlées en double-rail. Ceci pousse à définir deux types de champ des commandes pour la matrice OU du PLA de commandes : le champ des commandes codées en Berger et le champ des commandes codées en double-rail. La figure 33 montre les contrôleurs (double-rail et Berger) qui contrôlent les différents champs de commandes après avoir traverser la partie opérative.

2.3.12 - Gestion des interruptions

COBRA présente un seul niveau d'interruption, pour 10 origines possibles, prises en compte par ordre de priorité. Les demandes d'interruption sont soit d'origine externe : Reset, entrées HORLOGE, liaison série, soit d'origine interne : fin de décomptage dans un compteur CNT. La ligne externe Reset est la plus prioritaire, suivie des entrées HORLOGE(de 1 à 5) et des compteurs (CNTA, CNTB et CNTC) et enfin de la liaison série. Le déroulement normal d'un programme est interrompu à la fin de l'instruction en cours d'exécution lorsqu'une demande d'interruption est active. Dans le cas où plus d'une demande sont actives, la plus prioritaire est considérée, et les autres seront mémorisées dans des bascules, et ignorées jusqu'à la fin de l'exécution du sous-programme d'interruption en cours, et qui est signalée par l'instruction RTI.

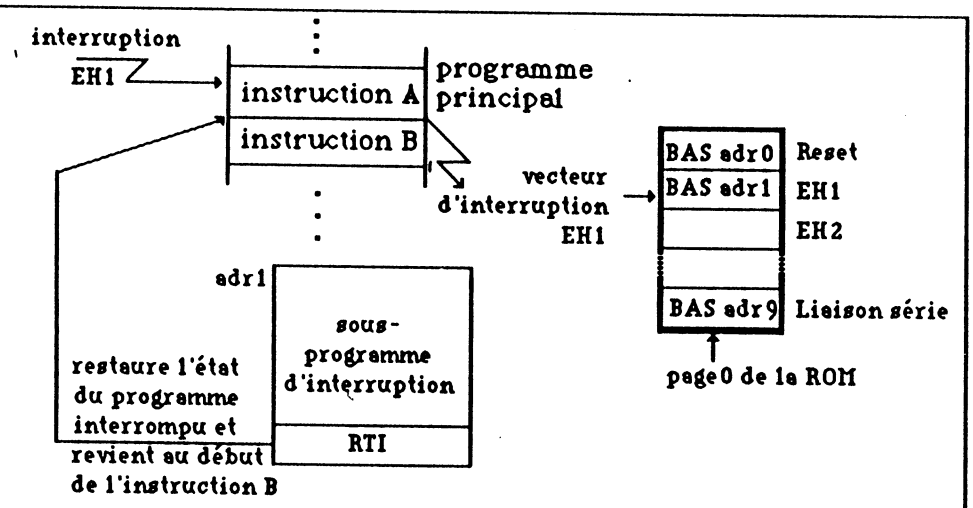


Figure 35 - Déroutement d'un programme suite à une interruption

A chacune des 10 origines d'interruption est associé un code opération, comme pour une instruction. Lorsqu'une interruption est admise par la fonction de priorité (implantée dans un PLA), alors le code opération correspondant (codé en parité) est présenté à l'entrée du PLA de propriétés, et à partir de là, une suite de séquences sera exécutée comme pour une instruction. La description IRENE de l'annexe A se termine par les descriptions de "pseudo-instructions" nommées sir0, sir1 jusqu'à sir9. Ces pseudo-instructions ne sont autres que les suites de séquencements correspondants aux opérations réalisées à chaque interruption et ceci en fonction de son origine. Elles commencent toujours par la sauvegarde des registres dans la pile, et ensuite le pointage d'un sous-programme d'interruption par un vecteur d'interruption. Les vecteurs d'interruption sont sur 16 bits chacun, et occupent les premiers octets de la page zéro de la ROM externe :

Reset	ØØ et Ø1
EH1	Ø2 et Ø3
EH2	Ø4 et Ø5
EH3	Ø6 et Ø7
EH4	Ø8 et Ø9
EH5	ØA et ØB
CNTA	ØC et ØD
CNTB	ØE et ØF
CNTC	1Ø et 11
série	12 et13

Comme c'est indiqué à la figure 35, ces adresses doivent contenir les instructions de branchement aux sous-programmes d'interruptions correspondants.

Ces choix de traitement des interruptions est particulièrement intéressant pour le contrôle, car la plupart des opérations se font dans une structure "self-checking" (partie opérative, RAM, PLA, etc...). Seules les fonctions de l'acquisition d'une interruption et de la priorité, doivent être augmentées des structures de contrôle.

Le PLA des interruptions réalise deux fonctions combinatoires, qui sont: la fonction de comparaison de priorités et la fonction qui fait correspondre un code opération (parmi dix) à l'interruption sélectionnée (la plus prioritaire). Ce PLA est contrôlé en double-rail pour ses entrées, qui sont dupliquées (entrées HORLOGE, entrée Reset, liaison série, CNT), à la sortie de la matrice ET. La matrice OU présente en sortie un code opération codé en parité comme le registre IR, mais pour couvrir les pannes unidirectionnelles, cette matrice est également codée avec le code de Berger.

2.3.13 - Signaux d'horloge

La fréquence d'horloge à l'entrée du circuit doit être de 3 MHz. Cette fréquence est ensuite mise en forme à l'intérieur du plot d'horloge, qui génère alors de façon interne deux signaux d'horloge non recouvrants et de la même fréquence.

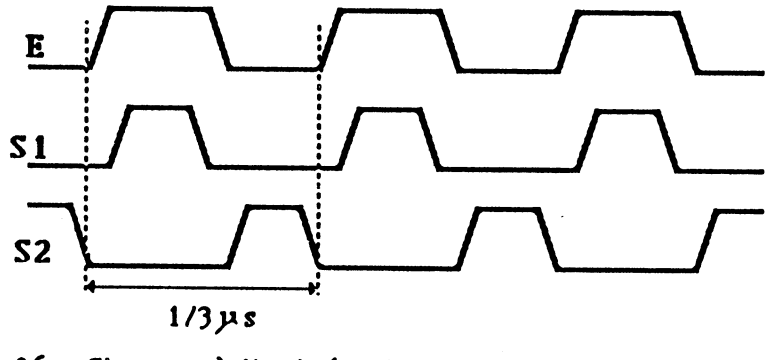


Figure 36 - Signaux à l'entrée et aux sorties du plot d'horloge

La génération des signaux d'horloge non recouvrants permet d'attaquer une chaîne de bascules D maître-esclave, montées en diviseur par deux. Ces bascules au nombre de 14, permettent de générer les fréquences nécessaires aux fonctionnements de :

- temporisation (732 Hz) avec les opérateurs CNT
- registres HORLOGE (11719 Hz, 2929 Hz, 732 Hz et 183 H z).
- génération des sorties HORLOGE (732 Hz)
- liaison série (46875 Hz)

La bascule D de base est représentée dans la figure 37 sous ses deux formes électrique et topologique.

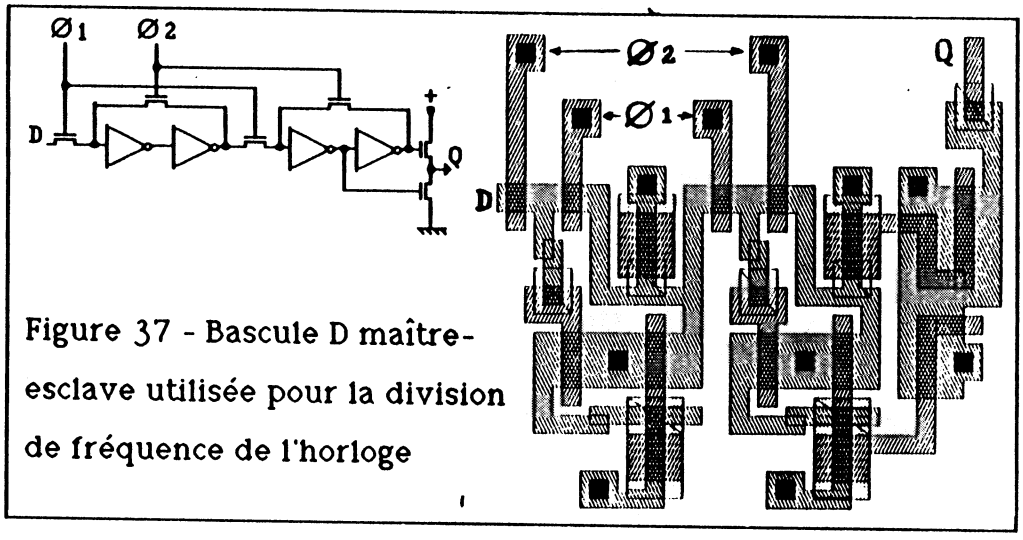


Figure 37 - Bascule D maître-esclave utilisée pour la division de fréquence de l'horloge

Le plot d'horloge est dupliqué, ainsi que toute la structure de division de fréquence. S'il y a une coupure interne ou externe au circuit d'une ligne d'horloge, ainsi qu'un glissement de fréquence, pouvant perturber le fonctionnement, une indication d'erreur serait de toute façon donnée, dans l'un des contrôleurs de blocs utilisant l'horloge. Toutefois, un contrôle aux sorties HORLOGE permet de surveiller d'une façon permanente (ou régulière) la fréquence de l'un des signaux d'horloge.

2.4 - Contraintes topologiques et implantation

Le problème d'implantation et d'assemblage des éléments composant un circuit VLSI devient synonyme d'un bon rendement de surface sans perte de performances. Plus le nombre d'éléments à assembler augmente, et plus les contraintes topologiques augmentent et nécessitent d'adopter des compromis d'assemblage. Le bon choix correspond à un minimun de compromis. Mais certaines contraintes nécessitent de déformer des éléments topologiques pour assurer une meilleure imbrication, comme les problèmes d'interconnexions ou les transparences. Alors les outils de C.A.O. sont très utiles pour aider à trouver la meilleure disposition des éléments et d'assurer leurs interconnexions, comme LUBRICK par exemple, bien qu'il soit souvent nécessaire de donner manuellement les grandes directives d'assemblage aux outils, qui s'occupent ensuite d'affiner les choix.

Dans le cas de COBRA, les éléments ont tous au départ des encombrements sous forme de rectangles, comme les PLA, la RAM et particulièrement la partie opérative, qui est plus longue que la partie commande dans un rapport de deux.

Les contrôleurs sont disposés autour des blocs qu'ils surveillent, comme l'exigent les règles des circuits autotestables. Leur implantation est donc difficilement optimisable, et pour la plupart, leur placement est intervenu après que la structure globale "partie opérative/partie commande" ait été définitive.

La partie opérative a donc été implantée autour de la partie commande. La figure 38 montre le plan de masse du circuit COBRA dans lequel le PLA de commandes figure au centre de la structure, ce qui facilite l'envoi des commandes vers la partie opérative, en parcourant des distances minimales. Le PLA de propriétés est situé à mi-chemin entre la partie opérative, contenant le registre d'instruction IR, et le PLA de commandes, de manière à ce que les interconnexions soient directes autant que possible. De même pour les générateurs de temps et le PLA de paramètres, ainsi que les autres éléments comme la RAM, les gestions des interruptions, la liaison série et les plots.

L'implantation de la partie opérative sous sa forme définitive a été préparée plusieurs étapes à l'avance. En effet, les éléments de la partie opérative ont été assemblés par modules. Chaque module pouvant être placé à n'importe quel endroit de la structure "bit slice" des bus de la partie opérative, puisqu'il comprend le bloc fonctionnel et la structure de contrôle associée. Ces modules sont ceux décrits dans les paragraphes précédents, et s'appellent : traitement des entrées NIVEAU, génération des sorties, ou encore unité arithmétique et logique. Pour la partie commande, seule la matrice OU du PLA des commandes a dû être remaniée. En effet, les commandes générées à partir de cette matrice doivent être amenées vers la partie opérative sans qu'ils se croisent et de la façon la plus directe possible. Alors les colonnes de cette matrice ont été déplacées une par une et par paquets, ce qui n'a modifié en rien la fonctionnalité de la partie commande, ni le codage de la matrice OU du PLA de commandes.

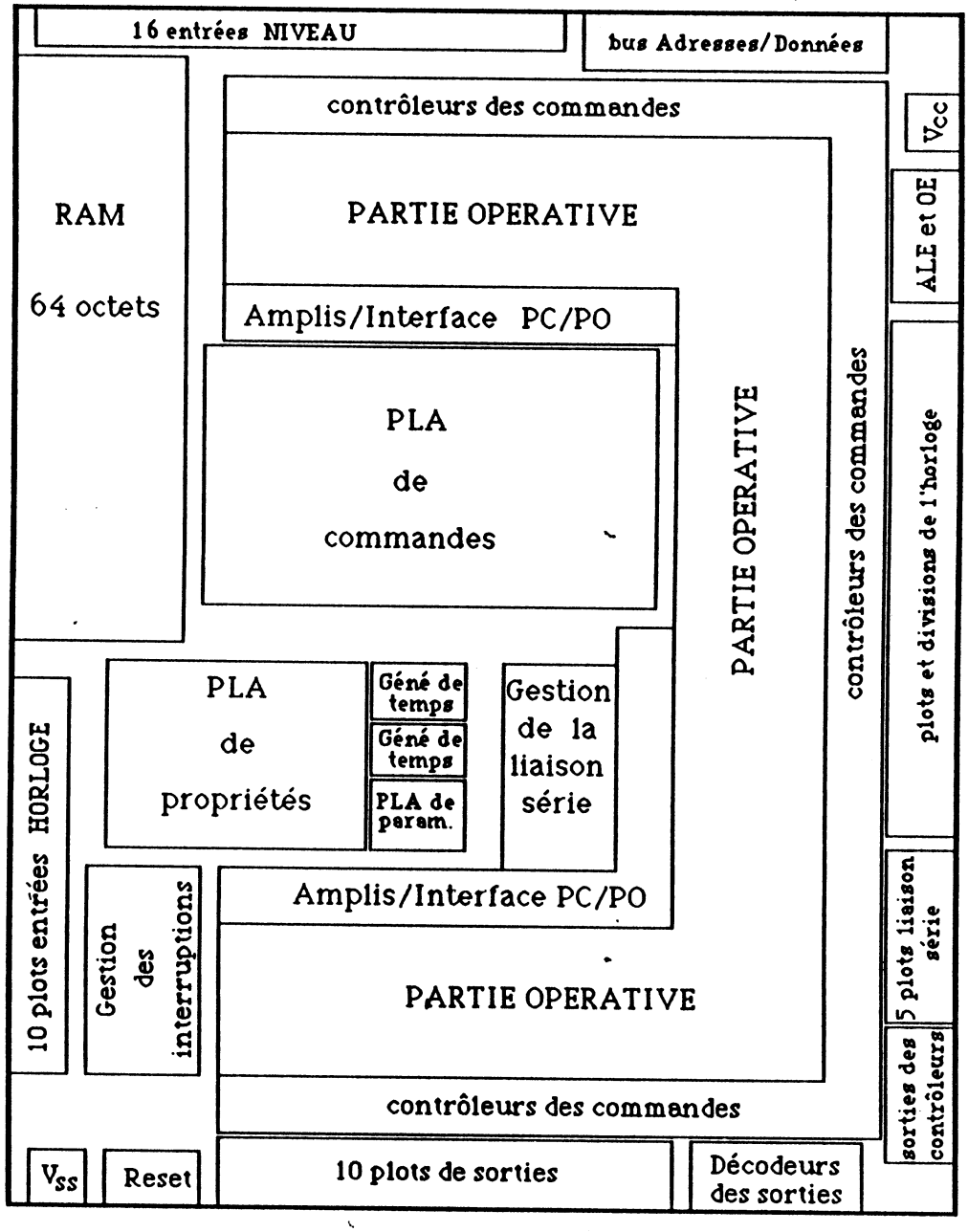


Figure 38 - Plan de masse de COBRA

3 - CONCLUSION

Ce chapitre présentait la réalisation du circuit COBRA en tant que phase principale du projet faisant l'objet de cette thèse. Cette réalisation se présente comme résultat de la méthode de conception descendante dans laquelle ont été incorporées les règles de conception de circuits autotestables.

Les spécifications externes ont été présentées en parallèle avec les contrôles des entrées/sorties, et des modifications que ces contrôles apportent par l'augmentation du nombre des broches externes : 22 broches de contrôle pour 42 broches fonctionnelles.

Les affinements successifs des spécifications externes, puis internes aboutissent à une structure globale de partie opérative et partie commande, composées d'éléments spécifiques à l'application de sécurité des transports, particulièrement dans la partie opérative. Cet aspect de spécificité a nécessité de concevoir et dessiner les masques de plusieurs éléments nouveaux. Les autres éléments étaient repris, et adaptés pour certains, dans une bibliothèque de cellules de l'équipe.

Les contrôleurs et les codes qu'ils surveillent, ont été introduits de manière simultanée autour des blocs fonctionnels à contrôler afin d'assurer la détection de toutes les pannes, selon les règles de conception de circuits autotestables rappelées dans le chapitre 2.

Le jeu d'instructions est présenté, avec ses 43 instructions au total, dont 16 pouvant être considérées comme particulièrement spécifiques à l'application de surveillance des automatismes comme les transports.

Les éléments du circuit sont présentés un par un, avec la structure de contrôle associée et leur mode de fonctionnement. Certains de ces éléments ont des architectures nouvelles comme les compteurs LFSR et la mémoire vive, RAM self-checking. La partie commande choisie ici est basée sur le principe de multi-PLA avec générateur de temps. Ce choix a plusieurs avantages : réalisation assistée par des outils de C.A.O., ici IRENE, MACSIM er PAOLA, et de plus sa structure se prête bien aux contrôles de façon peu coûteuse en surface.

Le plan de masse de COBRA, montre à la fin de ce chapitre, la disposition des éléments de COBRA, sur une surface de silicium de 7 mm x 9 mm.

CONCLUSION

CONCLUSION

La conception d'un "circuit intégré, de type microprocesseur, autotestable et spécifique à des applications de sécurité", nécessite une méthode de travail dans laquelle interviennent les notions suivantes :

- La spécificité de l'application du circuit. Dans le cas de COBRA, l'application dans les systèmes de transport de sécurité, exige des structures particulières et des opérateurs spéciaux pour sa fonctionnalité. Elle demande également une démonstration de sécurité et des preuves solides de la validité de la méthode de surveillance adoptée.

- La théorie et les techniques de circuits autotestables, qui introduisent des contraintes diverses lors du dessin et de l'implantation des blocs fonctionnels et de leurs contrôleurs, mais qui assurent la détection des pannes internes pendant que le circuit surveille le système qui le contient.

- La conception du circuit par assemblage des cellules élémentaires, à des niveaux différents. Les outils de C.A.O. contribuent à la fiabilité de la conception, puisqu'ils diminuent le risque d'erreurs manuelles. Mais, certains de ces outils nécessitent une adaptation aux structures non classiques pour lesquelles ils n'ont pas été conçus, comme ici les structures ayant les propriétés "self-checking". Une proposition dans ce but, a été faite ici.

Le cahier des charges du circuit COBRA a dû être révisé à plusieurs reprises et a finalement été fixé pour l'étude et la réalisation d'un circuit spécifique d'un environnement dans lequel son rôle est bien établi, et se nomme "partie logique de la chaîne de surveillance du pilote automatique embarqué". Ce circuit doit donc respecter un certain nombre de consignes, et de plus, comme il est programmable, il doit pouvoir fonctionner dans divers environnements de sécurité.

Le système du métro de Lille, le VAL, présenté en tant qu'exemple, est d'actualité, et représente bien le cahier des charges fonctionnellles de départ de la présente étude (à distinguer du cahier des charges de l'aspect "sécurité").

il est par ailleurs montré que les différentes techniques visant à assurer la surveillance de sécurité dans les systèmes de transports actuels sont insuffisantes, soit en raison de la limitation de leur puissance de traitement face aux systèmes complexes à venir, soit par l'insuffisance de la démonstration de sécurité, importante pour la certificaion de la qualité de la sécurité à accomplir.

La filière "test intégré" adoptée pour COBRA n'est pas nouvelle, beaucoup d'études ont été réalisées sur des méthodes de test et de surveillance intégrés. Les résultats de ces études poussent à raisonner autrement, face aux pannes possibles, dans les circuits intégrés et incitent à prévoir les pannes plutôt que de développer des méthodes de codage et de contrôle qui ne couvrent pas toutes les pannes parfois inconnues. Les techniques des circuits "self-checking" sont basées sur des hypothèses de pannes de bas niveau, et sur des classifications des pannes réelles. Ainsi toutes les pannes possibles sont connues (pour une technologie donnée, qui est ici le NMOS), et pour différentes classes de pannes il existe des techniques adaptées qui assureront la détection de la panne dès qu'elle apparaîtra. Les démonstrations de validité des techniques de "self-checking" s'appuient essentiellement sur des preuves mathématiques [NIC 84].

Quelques définitions et règles de conception des circuits logiques "self-checking" sont présentées dans le deuxième chapitre, résumant ainsi, avec les hypothèses de pannes de bas niveau, la théorie et les techniques de conception des circuits autotestables appliquées pour la conception de COBRA.

D'autres techniques de test sont en cours de développement, et correspondent à une amélioration sensible des méthodes actuelles, il s'agit du "test unifié" [NIC 86-2]. Cette amélioration ne signifie pas que les techniques de "self-checking" sont insuffisantes pour détecter les pannes, mais elles correspondent à un pas en avant dans la conception des circuits autotestables. Le test unifié est basé sur la conception de blocs fonctionnels "Strongly Fault Secure" (comme pour les techniques "self-checking") et sur une approche nouvelle de contrôleurs "Strongly Code Disjoint" incluant des générateurs internes de vecteurs de test. Cette génération nouvelle de contrôleurs n'est pas beaucoup plus coûteuse en surface que les contrôleurs SCD classiques, et elle est plus avantageuse lorsque le nombre d'informations à contrôler est important, et ceci grâce à une très bonne modularité de la structure de contrôle.

Dans le test unifié, les contrôleurs restent SCD (Strongly Code Disjoint) pour toutes les pannes (simples, multiples), et indépendamment du bloc fonctionnel contrôle et du programme d'application. Par ailleurs, la vitesse est conservée grâce au peu de couches logiques de contrôleurs à traverser et à la simplicité de la structure des contrôleurs. De plus, l'hypothèse H2 est assurée de façon sûre et simple grâce à une activation périodique d'une phase de test. Le test unifié a un autre avantage notable qui est de permettre à la fois et avec la même structure, le test en ligne et le test hors ligne, ce dernier assurant la détection des pannes simples et multiples en fin de fabrication et lors de la maintenance,impossible auparavant.

La méthode de conception adoptée est présentée parallélement aux outils de C.A.O. Elle consiste à faire progresser les descriptions par affinements successifs, jusqu'à atteindre les niveaux de description les plus proches de la machine physique (méthode de conception dite descendante). La méthode de conception est validée à chaque étape par des simulations logiques ou électriques.

Le dernier chapitre décrit la réalisation du circuit COBRA. La partie opérative a été partagée en plusieurs sous-parties opératives pour faciliter son implantation et pour mieux cerner les problèmes de contrôle et de respect des règles de conception des blocs SFS. La partie opérative est d'abord séparée en deux ensembles : les sous-parties opératives spécifiques et les sous-parties opératives de travail (UAL, incrémenteur, registres de travail,...) chacune de ces sous-parties opératives est décrite par sa structure, son utilisation par les instructions associées,sa structure logique modifiée pour le contrôle et dans certains cas les dessins au micron des éléments de base. La partie commande basée sur le principe du générateur de temps est réalisée à l'aide des outils IRENE–MACSIM–PAOLA. Dans le cas de COBRA, les PLA de décodage et de commande sont de tailles importantes mais, avec des outils du milieu industriel leurs tailles seraient nettement diminuées en raison de la simplicité des fonctions qu'elles réalisent.

La poursuite du projet COBRA est envisagée pour l'avenir en s'orientant vers un circuit en CMOS fabriqué en milieu industriel, et qui réaliserait les mêmes fonctions générales que le COBRA (temporisation, datage d'événements, liaison série, RAM, etc...) mais qui serait construit autour d'un circuit coeur standard, afin de faciliter des opérations comme l'émulation, mais surtout afin de s'ouvrir vers un marché en pleine croissance.

BIBLIOGRAPHIE

[ANC 76] **ANCEAU F.**
Sécurité dans les systèmes complexes : application aux autocommutateurs téléphoniques
Note interne IMAG, Décembre 1976.

[AND 71] **ANDERSON D.A.**
Design of self-checking digital networks using coding techniques.
Coordinated Science Laboratory, Report R/527. University of Illinois, Urbana, Septembre 1971.

[BER 61] **BERGER J.M.**
A note on error detection codes for assymetric channels, Information and control.
New York, Mars 1961.

[BOU 84] **BOURCIER E.**
Conception et réalisation du SIMULATEUR du langage de description de circuits intégrés IRENE-C.
Thèse d'ingénieur CNAM, Grenoble, Octobre 1984.

[BRE 76] **BREUER M.A., FRIEDMAN A.D.**
Diagnosis and reliable design of digital systems.
London, Pitman Publishing Limited 1976.

[CHU 84] **CHUQUILLANQUI S.H.**
Une nouvelle approche pour l'optimisation topologique et l'automatisation du dessin des masques de P.L.A. Complexes.
Thèse de docteur ingénieur, INPG, Octobre 1984.

[CMP 85] **DELORI H., GUYOT A.**
Le CMP Français en 1984
Rapport de Recherche RR479, Février 1985.

[COU 79] **COURTOIS B.**
Some results about the efficiency of simple mechanisms for the detection of microcomputer malfunctions.
Proceedings of the 9th Fault-Tolerant Computing Symposium, Madison, USA, Juin 1979.

[COU 81] **COURTOIS B.**
Failure mechanisms, fault hypotheses and analytical testing of LSI-NMOS (HMOS) circuits.
VLSI 81, University of Edinburgh, Grande Bretagne, Août 1981.

[CRO 78] **CROUZET Y.**
Conception de circuits à large échelle d'intégration totalement autotestables.
Thèse de Docteur-Ingénieur, Université Paul Sabatier, Toulouse, Novembre 1978.

[DAV'78] **DAVID R.**
A totally self-checking 1-out-of-3 checker,
IEEE Transactions on Computers. New York, Juin 1978.

[DIA 79] **DIAZ M., AZEMA P., AYACHE J.**
Unified design of self-checking and fail safe combinational circuits and sequential machines.
IEEE Transactions on Computers. New York, Mars 1979.

[FRI 67] **FRIEDMAN A. D.**
Fault detection in redundant circuits.
IEEE Transactions on Computers, Février 1967.

[FRI 71] **FRIEDMAN A. D., MEMON P. R.**
Fault detection in digital circuits.
Prenctice Hall Inc., 1971.

[GAL 84] GALLAIS R.
Participation à la réalisation d'un microprocesseur 16 bits interprétant le P.CODE.
Diplôme d'ingénieur CNAM, Mars 1984.

[GAL 80] GALIAY J., CROUZET Y., VERGNIAULT M.
Physical versus logical fault models MOS-LSI circuits : impact on their testability
IEEE Transactions on Computers, Juin 1980.

[GAZ 84] GAZET A.
Version biprocesseur de sécurité.
Journées " Sécurité des applications des automatismes numériques dans les transports ".
Villeneuve d'Ascq, 24-25 Octobre 1984.

[GEF 76] GEFFROY J-C
Test en ligne et sécurité des systèmes logiques
.Thèse de docteur d'état, Université Paul Sabatier,
Toulouse, Novembre 1976.

[ICSU/AB] Abstracting board on the International Council of Scientific Unions

[IRT 82] INSTITUT DE RECHERCHE DES TRANSPORTS
Utilisation des microprocesseurs dans les applications ferroviaires de sécurité.
Recheche bibliographique. Essais de synthèse.
Rapport MA.82.105, Août 1982.

[IRT 84] INSTITUT DE RECHERCHE DES TRANSPORTS
Introduction à la conception de circuits intégrés autotestables
Rapport CR/A.84.28 1ère édition . Septembre 1984

[JAM 85] **JAMIER R., JERRAYA A.A.**
APOLLON, A Data Path Silicon Compiler
ICCD'85, New York, October 1985.

[JAN 85] **JANSCH SCHREIBER I. E.**
Conception de contrôleur autotestables pour des hypothèses de pannes analytiques.
Thèse de docteur-ingénieur, INP de Grenoble, Janvier 1985.

[JER 86-1] **JERRAYA A.A., VARINOT P., JAMIER R., COURTOIS B.**
SYCO : A Silicon Compiler for VLSI Circuits Described by Algorithms.
VLSI Physical Design, IEEE, March 1986
Houston, TEXAS USA.

[JER 86-2] **JERRAYA A.A., VARINOT P., JAMIER R., COURTOIS B.**
The principles of the SYCO compiler
23^{d} D.A.C., LAS VEGAS USA. June 28th 1986

[LUC 85] **PAILLOTIN J.F.**
Le système LUCIE. Manuel d'utilisation. Version de Juillet 1985.

[MAR 84] **MARINE S.,BOURCIER E.,BRAZZI M., SCHOELLKOPF J.P.**
CVT subtask 1-6, six months technical report, Aôut 1984.

[MAR 85] **MARINE S., BOURCIER E.**
IRENE, a language for the Description of VLSI Digital Hardware.
IMAG/TIM 3, R.R. n° 536, Mai 1985.

[MAR 86] **S. MARINE**
IRENE, un langage pour la description, simulation et Synthèse automatique du matériel VLSI.
Thèse de Doctorat, INPG, Février 1986.

[MEA 80] **MEAD C., CONWAY L.**
Introduction to VLSI Systems Addison - Wesley Publishion Company, 1980.

[NIC 83] **NICOLAIDIS M., COURTOIS B.**
Design of self-checking systems based on analytical fault hypotheses.
IMAG, R.R. N° 353, Mars 1983.

[NIC 84] **NICOLAIDIS M.**
Conception de circuits intégrés autotestables pour des hypothèses de pannes analytiques.
Thèse de docteur-ingénieur, INP de Grenoble, Janvier 1984.

[NIC 86-1] **NICOLAIDIS M.**
Démonstration de sécurité du microcontrôleur COBRA
Rapport interne, Avril 1986

[NIC 86-2] **NICOLAIDIS M.**
A unified BIST approach using specific strongly code disjoint checkers' design
TIM3 IMAG, R.R. N° 599, Mars 1986.

[OBR 82] **OBREBSKA M.**
Etude Comparative de différentes méthodes de conception des parties contrôle des microprocesseurs.
Thèse DI, INPG, Juin 1982.

[OSS 86] **OSSEIRAN A. et al**
Design of a self-checking microprocessor for real time applications
Proc. of IFAC, 5th conference of security in transportation systems, Vienna, Austria ,July 1986

[PER 85] **PEREZ SEGOVIA T.**
PAOLA : un système d'optimisation topologique de P.L.A.
Thèse 3ème Cycle, INPG, Octobre 1985.

[SCH 85] **SCHOELLKOPF J.P.**
SILICIEL, Contributions à l'architecture des circuits intégrés et à la compilation du Silicium.
Thèse de docteur d'état, INPG, Avril 1985.

[SMI 77] **SMITH J. E., METZE G.**
The design of totally self-checking combinational circuits.
Proceedings of the 7th Fault tolerant computing Symposium, Los Angeles, CA, June 1977.

[SMI 78] **SMITH J. E., METZE G.**
Strongly fault secure logic networks.
IEEE Transactions on Computers, New York, June 1978.

[SPI 80] **SPICE Version 2G-1 user's guide**
Departement of Electrical Engineering and Computer Sciences.
University of California, Berkeley, CA., 94720.Octobre 1980.

[VAL 84] **VALENCOGNE J.**
Processeur codé ou monoprocesseur de sécurité.
Journées " Sécurité des applications des automatismes numériques dans les transports ".
Villeneuve d'Ascq, 24-25 Octobre 1984.

[VIA 80] **VIAUD J., DAVID R.**
Sequentially self-checking circuits.
10th International Symposium on Fault Tolerant Computing, Kyoto Japan, October 1980.

[WIL 80] **WILLIAMS T.W.**
Computer design aids for VLSI circuits.
Design For Testability, Urbino Italy ,Summer 1980

ANNEXE A

DESCRIPTION *IRENE* DE *COBRA*

```
MODULE COBRA;
INST
BUSA<7:0>,
BUSB<7:0>,
ALEH,
OE,
IT;
LATCH
LSB7<7:0>,
MSB7<7:0>,
EB<7:0>,
INCA<7:0>,
INCAB<7:0>,
SINC1<7:0>,
IINC1,
IINC1B,
DINC1,
DINC1B,
SP<7:0>,
SPB<7:0>,
PCL<7:0>,
PCLB<7:0>,
PCH<7:0>,
AL1<7:0>,
ACCA<7:0>,
SHIFT<7:0>,
SELS<1:0>,
CR1<7:0>,
CR2<7:0>,
CR3<7:0>,
CR4<7:0>,
CR5<7:0>,
CR1RST,
CR2RST,
CR3RST,
CR4RST,
CR5RST,
ALU2<7:0>,
ALU2B<7:0>,
SALU<7:0>,
EALUB<7:0>,
EALUBB<7:0>,
EALUA<7:0>,
EALUAB<7:0>,
EALU<7:0>,
BEALU<7:0>,
SEL2<4:0>,
SEL2B<4:0>,
CNTAL<7:0>,
CNTAH<7:0>,
CNTBL<7:0>,
CNTBH<7:0>,
CNTCL<7:0>,
CNTCH<7:0>,
OREG<7:0>,
RAM1<7:0>,
PILE[44:0]<7:0>,
RS<7:0>,
RAMDEC<7:0>,
IR<7:0>,
```

```
CCR<7:0>,
MS<7:0>,
DECO<7:0>,
IL1<9:0>,
TR<7:0>,
PRCHG,
MUXI,
TM,
X,
Y,
Z,
BITZ1,
BITZ2,
BITN,
BITC,
BITV,
BITNV,
ALU1,
ABC<2:0>,
BB,
BA,
BUS;


SEQUENCE INSTRUCTIONS<53> :$LZ,$AZ,$OZ,$XZ,$NAZ,$NOZ,$NXZ,
                           $BAS,$BAL,$BSZ,$BSNZ,$BEQ,$BMI,$BLT,$BCS,
                           $BLZ,$BLNZ,$LBEQ,$LBMI,$LBLT,$LBCS,
                           $TMPA,$TMPB,$TMPC,$ACO,$STM,$LDM,$LOAD,
                           $ADDA,$SUBA,$ANDA,$ORA,$CMPA,$CIL,$CIM,
                           $SXZ,$SYZ,$LSL,$LSR,$COM,$DEC,$SERT,$RTI,
                           $SIR0,$SIR1,$SIR2,$SIR3,$SIR4,$SIR5,
                           $SIR6,$SIR7,$SIR8,$SIR9;GENERAL

SEQUENCE CYCLE<12>  : $C1,$C2,$C3,$C4,$C5,$C6,$C7,$C8,$C9,
                      $C10,$C11,$C12;GENERAL

SEQUENCE PHASE<3>   : $PHI1,$PHI2,$PHI3;RIGID FIXED

  ENDSEQ
 ENDSEQ
ENDSEQ

BEGIN

!CODOP1=01001000;!
$LZ  :$C1:$PHI1:PRCHG:='1;
           $PHI2:BUSA:=PCL;
           $PHI3:EB:=BUSA/INCA:=PCL/
                 INCAB:=PCLB/IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
           $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
           $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
           $PHI2:BUSA:=PCL/DECO:=MS;
           $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                 IINC1:='1/IINC1B:='1/
                 Z:=BB/Z:=MUXI/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
           $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
```

```
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01001011;!
$AZ   :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/DECO:=MS/
                A3C:='100/BB:=Z/BA:=MUXI/
                ALU1:=BUS;
          $PHI3:E3:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                BB:=ALU1/Z:=BB/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01001101;!
$OZ   :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/DECO:=MS/
                A3C:='011/BB:=Z/BA:=MUXI/
                ALU1:=BUS;
          $PHI3:E3:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                BB:=ALU1/Z:=BB/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01001110;!
$XZ   :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/DECO:=MS/
                A3C:='001/BB:=Z/BA:=MUXI/
                ALU1:=BUS;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC13:='1/
                BB:=ALU1/Z:=BB/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11001001;!
```

```
$NAZ :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/DECO:=MS/
                ABC:='101/BB:=Z/BA:=MUXI/
                ALU1:=BUS;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                BB:=ALU1/Z:=BB/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11001010;!
$NOZ :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/DECO:=MS/
                ABC:='010/BB:=Z/BA:=MUXI/
                ALU1:=BUS;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                BB:=ALU1/Z:=BB/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11001111;!
$NXZ :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:MS:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/DECO:=MS/
                ABC:='000/BB:=Z/BA:=MUXI/
                ALU1:=BUS;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                BB:=ALU1/Z:=BB/BITZ1:=NOT Z;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=00110000;!
$BAS :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA;
```

```
      $C2:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=00111111;!
$BAL  :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/PCL:=BUSB/PCLB:=BUSB;
       $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/ALEH:='1;
          $PHI3:EB:=BUSA;
       $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11011011;!
$BSZ  :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC13:='1;
       $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
       $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITZ1 ='1 THEN AL1
                ELSE PCL ENDIF/
                BUSB:=IF BITZ1='1 THEN AL1
                ELSE PCL ENDIF;
          $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA/
                INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11011101;!
$BSNZ:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
```

```
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITZ1 ='0 THEN AL1
                ELSE PCL ENDIF/
                BUSB:=IF BITZ1 ='0 THEN AL1
                ELSE PCL ENDIF;
          $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA/
                INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01011001;!
$BEQ :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITZ2 ='1 THEN AL1
                ELSE PCL ENDIF/
                BUSB:=IF BITZ2 ='1 THEN AL1
                ELSE PCL ENDIF;
          $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA/
                INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01011010;!
$BMI :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITN ='1 THEN AL1
                ELSE PCL ENDIF/
                BUSB:=IF BITN ='1 THEN AL1
                ELSE PCL ENDIF;
          $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA/
                INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01011100;!
$BLT :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
```

```
                IINC1:='1/IINC13:='1;
        $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/CE:='1/BUSA:=EB;
            $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
        $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITNV ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITNV ='1 THEN AL1
                  ELSE PCL ENDIF;
            $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA/
                  INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
        $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/CE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01011111;!
$BCS :$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
        $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/CE:='1/BUSA:=EB;
            $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
        $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITC ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITC ='1 THEN AL1
                  ELSE PCL ENDIF;
            $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA/
                  INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
        $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/CE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01010011;!
$BLZ :$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC13:='1;
        $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/CE:='1/BUSA:=EB;
            $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
        $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
        $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/CE:='1/BUSA:=EB;
            $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
        $C5:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITZ1 ='1 THEN ACCA
                  ELSE PCH ENDIF/ALEH:='1;
            $PHI3:EB:=BUSA/PCH:=BUSA;
        $C6:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITZ1 ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITZ1 ='1 THEN AL1
                  ELSE PCL ENDIF;
```

```
            $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                  IINC1:='1/IINC1B:='1;
      $C7:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01010101;!
$BLNZ:$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C5:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITZ1 ='0 THEN ACCA
                  ELSE PCH ENDIF/ALEH:='1;
            $PHI3:EB:=BUSA/PCH:=BUSA;
      $C6:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITZ1 ='0 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITZ1 ='0 THEN AL1
                  ELSE PCL ENDIF;
            $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                  IINC1:='1/IINC1B:='1;
      $C7:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11010001;!
$LBEQ:$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL;
            $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C5:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITZ2 ='1 THEN ACCA
                  ELSE PCH ENDIF/ALEH:='1;
            $PHI3:EB:=BUSA/PCH:=BUSA;
      $C6:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF BITZ2 ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITZ2 ='1 THEN AL1
                  ELSE PCL ENDIF;
```

```
          $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                IINC1:='1/IINC1B:='1;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11010010;!
$LBMI:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITN='1 THEN ACCA
                  ELSE PCH ENDIF/ALEH:='1;
          $PHI3:EB:=BUSA/PCH:=BUSA;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITN ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITN ='1 THEN AL1
                  ELSE PCL ENDIF;
          $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                IINC1:='1/IINC1B:='1;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11010100;!
$LBLT:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
       $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITNV ='1 THEN ACCA
                  ELSE PCH ENDIF/ALEH:='1;
          $PHI3:EB:=BUSA/PCH:=BUSA;
       $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITNV ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITNV ='1 THEN AL1
                  ELSE PCL ENDIF;
```

```
          $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                IINC1:='1/IINC1B:='1;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11010111;!
$LBCS:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:AL1:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITC ='1 THEN ACCA
                  ELSE PCH ENDIF/ALEH:='1;
          $PHI3:EB:=BUSA/PCH:=BUSA;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IF BITC ='1 THEN AL1
                  ELSE PCL ENDIF/
                  BUSB:=IF BITC ='1 THEN AL1
                  ELSE PCL ENDIF;
          $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                IINC1:='1/IINC1B:='1;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01110001;!
$TMPA:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:CNTAH:=BUSA/PCL:=BUSB/PCLB:=BUSB;
       $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:CNTAL:=BUSA/PCL:=BUSB/PCLB:=BUSB;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
       $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01110010;!
```

```
$TMP8:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:CNTBH:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:CNTBL:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01110100;!
$TMPC:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:CNTCH:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:CNTCL:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11110000;!
$ACO  :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:OREG:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;
```

```
!CODOP1=00101101;!
$STM :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:RS:=BUSA/PCL:=BUSB/PCLB:=BUSB/RAMDEC:=RS;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA;
          $PHI3:RAM1:=BUSA/BITV:='0/BITN:=ACCA<6>/
                  BITZ2:=(ACCA=#00);
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=00101110;!
$LDM :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:RS:=BUSA/PCL:=BUSB/PCLB:=BUSB/RAMDEC:=RS;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=RAM1;
          $PHI3:ACCA:=BUSA/BITV:='0/BITN:=ACCA<6>/
                  BITZ2:=(ACCA=#00);
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=00010010;!
$LOAD:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:ACCA:=BUSA/PCL:=BUSB/PCLB:=BUSB;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                  BITV:='0/BITN:=ACCA<6>/
                  BITZ2:=(ACCA=#00);
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10011010;!
```

```
$ADDA:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                EALUB:=BUSB/EALUBB:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:EALUA:=BUSA/EALUAB:=BUSA/SEL2:='00100/SEL2B:='00100/
                PCL:=BUSB/PCLB:=BUSB/ALU2:=EALU/ALU2B:=BEALU;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SALU;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/ACCA:=BUSB/
                  BITN:=ALU2<6>/BITZ2:=(ALU2=#00)/
                  BITC:=(ALU2<6:0><<ACCA<6:0>);
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB/
                BITV:=BITC EXOR BITN;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10011100;!
$SUBA:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                EALUB:=BUSB/EALUBB:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:EALUA:=BUSA/EALUAB:=BUSA/SEL2:='00100/
                SEL2B:='00100/PCL:=BUSB/PCLB:=BUSB/
                ALU2:=EALU/ALU2B:=BEALU;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SALU;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/ACCA:=BUSB/
                  BITN:=ALU2<6>/BITZ2:=(ALU2=#00)/
                  BITC:=(ALU2<6:0><<ACCA<6:0>);
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB/
                BITV:=BITC EXOR BITN;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=00011000;!
$ANDA:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                EALUB:=BUSB/EALUBB:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:EALUA:=BUSA/EALUAB:=BUSA/SEL2:='10010/
                SEL2B:='10010/PCL:=BUSB/PCLB:=BUSB/
                ALU2:=EALU/ALU2B:=BEALU;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SALU;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/ACCA:=BUSB/
                  BITN:=ALU2<6>/BITZ2:=(ALU2=#00)/
                  BITV:='0;
      $C4:$PHI1:PRCHG:='1;
```

```
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=00011011;!
$ORA :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                EALUB:=BUSB/EALUBB:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:EALUA:=BUSA/EALUAB:=BUSA/SEL2:='01001/
                SEL2:='01001/PCL:=BUSB/PCLB:=BUSB/
                ALU2:=EALU/ALU2B:=BEALU;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SALU;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/ACCA:=BUSB/
                  BITN:=ALU2<6>/BITZ2:=(ALU2=#00)/
                  BITV:='0;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10011001;!
$CMPA:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                EALUB:=BUSB/EALUBB:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:EALUA:=BUSA/EALUAB:=BUSA/SEL2:='00100/
                SEL2B:='00100/PCL:=BUSB/PCLB:=BUSB/
                ALU2:=EALU/ALU2B:=BEALU;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SALU;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/ACCA:=BUSB/
                  BITN:=ALU2<6>/BITZ2:=(ALU2=#00)/
                  BITC:=(ALU2<6:0><<ACCA<6:0>);
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB/
                  BITV:=BITC EXOR BITN;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=01101100;!
$CIL :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=LSB7;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/ACCA:=BUSB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                BITV:='0/BITN:=ALU2<6>/
                BITZ2:=(ALU2=#00)/IT:='1/GOTO;

!CODOP1=01101111;!
$CIM :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=MSB7;
```

```
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/ACCA:=BUSB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                BITV:='0/BITN:=ALU2<6>/
                BITZ2:=(ALU2=#00)/IT:='1/GOTO;

!CODOP1=11000000;!
$SXZ :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BB:=Z/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                X:=BB/IT:='1/GOTO;

!CODOP1=11000011;!
$SYZ :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BB:=Z/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                Y:=BB/IT:='1/GOTO;

!CODOP1=10000001;!
$LSL :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                SHIFT:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB/SELS:='01/
                BITC:=ACCA<6>/BITN:=ALU2<6>;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/ACCA:=SHIFT/
                BITZ2:=(ALU2=#00)/
                BITV:=BITN EXOR BITC/IT:='1/GOTO;

!CODOP1=10000010;!
$LSR :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                SHIFT:=BUSB;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB/SELS:='10/
                BITC:=ACCA<6>/BITN:='0;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/ACCA:=SHIFT/
                BITZ2:=(ALU2=#00)/IT:='1/GOTO;

!CODOP1=00000000;!
$COM :$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=ACCA;
          $PHI3:EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1/
                SHIFT:=ACCA;
      $C2:$PHI1:PRCHG:='1;
```

```
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/
                  ACCA:=SHIFT/BITV:='0/BITC:='1/
                  BITN:=ALU2<6>/BITZ2:=(ALU2=#00)/IT:='1/GOTO;

!CODOP1=00000011;!
$DEC  :$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=ACCA;
            $PHI3:EALUB:=BUSB/EALUBB:=BUSB/
                  SEL2:='00000/SEL2B:='00000/
                  ALU2:=EALU/ALU2B:=BEALU;
         $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SALU/BUSA:=PCL;
            $PHI3:ACCA:=BUSB/EB:=BUSA/INCA:=PCL/INCAB:=PCLB/
                  IINC1:='1/IINC1B:='1;
         $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=11100000;!
$SERT:$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=ACCA;
            $PHI3:TR:=BUSB/INCA:=PCL/INCAB:=PCLB/
                  DINC1:='1/DINC1B:='1;
         $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IF TM='0 THEN PCL
                    ELSE SINC1 ENDIF;
            $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA;
         $C3:$PHI1:PRCHG:='1;
            $PHI2:OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/TM:='1/IT:='1/GOTO;

!CODOP1=11111111;!
$RTI  :$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SP;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/
                  INCAB:=SPB/DINC1:='1/DINC1B:='1;
          $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PILE[RAMDEC]/BUSB:=SINC1;
            $PHI3:CCR:=BUSA/RAMDEC:=BUSB/INCA:=SP/
                  INCAB:=SPB/DINC1:='1/DINC1B:='1;
          $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PILE[RAMDEC]/BUSB:=SINC1;
            $PHI3:ACCA:=BUSA/RAMDEC:=BUSB/INCA:=SP/
                  INCAB:=SPB/DINC1:='1/DINC1B:='1;
          $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PILE[RAMDEC]/BUSB:=SINC1/ALEH:='1;
            $PHI3:PCH:=BUSA/EB:=BUSA/RAMDEC:=BUSB/
                  INCA:=SP/INCAB:=SPB/
                  DINC1:='1/DINC1B:='1;
          $C5:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PILE[RAMDEC]/BUSB:=SINC1;
            $PHI3:PCL:=BUSA/PCLB:=BUSA/EB:=BUSA/
                  RAMDEC:=BUSB/INCA:=PCL/
                  INCAB:=PCLB/IINC1:='1/IINC1B:='1;
          $C6:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PILE[RAMDEC]/BUSB:=SINC1/
                  OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;
```

```
!CODOP1=10100000;!
$SIR0:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#00/ALEH:='1;
          $PHI3:EB:=BUSA/PCL:=BUSA/PCLB:=BUSA;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:INCA:=PCL/INCAB:=PCLB/IINC1:='1/
                IINC1B:='1/PCH:=BUSA;
      $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/BUSA:=PCL;
          $PHI3:PCL:=BUSB/PCLB:=BUSB/EB:=BUSA/
                INCA:=PCL/INCAB:=PCLB/
                IINC1:='1/IINC1B:='1;
      $C9:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1/ALEH:='1;
          $PHI3:EB:=BUSA/PCL:=BUSB/PCLB:=BUSB;
     $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
     $C11:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10110001;!
$SIR1:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
```

```
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
       $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=CR1/BUSA:=#00/ALEH:='1;
          $PHI3:ACCA:=BUSB/EB:=BUSA;
       $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#02;
          $PHI3:EB:=BUSA/CR1RST:='1/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
       $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/PCL:=BUSB/PCLB:=BUSB/EB:=BUSB;
       $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
      $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10110010;!
$SIR2:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
       $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
       $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
       $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=CR2/BUSA:=#00/ALEH:='1;
          $PHI3:ACCA:=BUSB/EB:=BUSA;
       $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#04;
          $PHI3:EB:=BUSA/CR2RST:='1/INCA:=BUSA/
                INCAB:=BUSA/IINC1:='1/IINC1B:='1;
       $C8:$PHI1:PRCHG:='1;
```

```
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/PCL:=BUSB/PCL3:=BUSB/EB:=BUSB;
      $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCL3:=BUSA;
     $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCL3:=BUSB/IT:='1/GOTO;

!CODOP1=10110100;!
$SIR3:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=CR3/BUSA:=#00/ALEH:='1;
          $PHI3:ACCA:=BUSB/EB:=BUSA;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#06;
          $PHI3:EB:=BUSA/CR3RST:='1/INCA:=BUSA/
                INCAB:=BUSA/IINC1:='1/IINC1B:='1;
      $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/PCL:=BUSB/PCL3:=BUSB/EB:=BUSB;
      $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCL3:=BUSA;
     $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10110111;!
$SIR4:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
```

```
                PILE[RAMDEC]:=BUSA;
        $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCH/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                  IINC1:='1/IINC1B:='1/
                  PILE[RAMDEC]:=BUSA;
        $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=ACCA/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                  IINC1:='1/IINC1B:='1/
                  PILE[RAMDEC]:=BUSA;
        $C5:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=CCR/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
        $C6:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=CR4/BUSA:=#00/ALEH:='1;
            $PHI3:ACCA:=BUSB/EB:=BUSA;
        $C7:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=#08;
            $PHI3:EB:=BUSA/CR4RST:='1/INCA:=BUSA/
                  INCAB:=BUSA/IINC1:='1/IINC1B:='1;
        $C8:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:PCH:=BUSA/PCL:=BUSB/PCLB:=BUSB/EB:=BUSB;
        $C9:$PHI1:PRCHG:='1;
            $PHI2:OE:='1/BUSA:=EB;
            $PHI3:PCL:=BUSA/PCLB:=BUSA;
       $C10:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
            $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10111000;!
$SIR5:$C1:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=IR/BUSB:=SP;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                  IINC1:='1/IINC1B:='1/
                  PILE[RAMDEC]:=BUSA;
        $C2:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCL/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                  IINC1:='1/IINC1B:='1/
                  PILE[RAMDEC]:=BUSA;
        $C3:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=PCH/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                  IINC1:='1/IINC1B:='1/
                  PILE[RAMDEC]:=BUSA;
        $C4:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=ACCA/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                  IINC1:='1/IINC1B:='1/
                  PILE[RAMDEC]:=BUSA;
        $C5:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=CCR/BUSB:=SINC1;
            $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
        $C6:$PHI1:PRCHG:='1;
            $PHI2:BUSB:=CR5/BUSA:=#00/ALEH:='1;
            $PHI3:ACCA:=BUSB/EB:=BUSA;
        $C7:$PHI1:PRCHG:='1;
            $PHI2:BUSA:=#0A;
```

```
          $PHI3:EB:=BUSA/CRSRST:='1/INCA:=BUSA/
                INCAB:=BUSA/IINC1:='1/IINC1B:='1;
     $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/PCL:=BUSB/PCLB:=BUSB/EB:=BUSB;
     $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
    $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10111011;!
$SIR6:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
     $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
     $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
     $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
     $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
     $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#00/ALEH:='1;
          $PHI3:EB:=BUSA;
     $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#0C;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
     $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/EB:=BUSB;
     $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
    $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/ALEH:='1;
          $PHI3:EB:=BUSA;
    $C11:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
    $C12:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;
```

```
!CODOP1=10111101;!
$SIR7:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILE[RAMDEC]:=BUSA;
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#00/ALEH:='1;
          $PHI3:EB:=BUSA;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#0E;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
      $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/EB:=BUSB;
      $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
     $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/ALEH:='1;
          $PHI3:EB:=BUSA;
     $C11:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
     $C12:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10111110;!
$SIR8:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILE[RAMDEC]:=BUSA;
      $C3:$PHI1:PRCHG:='1;
```

```
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILECRAMDEC]:=BUSA;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILECRAMDEC]:=BUSA;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILECRAMDEC]:=BUSA;
       $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#00/ALEH:='1;
          $PHI3:EB:=BUSA;
       $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#10;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
       $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/EB:=BUSB;
       $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
      $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/ALEH:='1;
          $PHI3:EB:=BUSA;
      $C11:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
      $C12:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

!CODOP1=10101111;!
$SIR9:$C1:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=IR/BUSB:=SP;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILECRAMDEC]:=BUSA;
       $C2:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILECRAMDEC]:=BUSA;
       $C3:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILECRAMDEC]:=BUSA;
       $C4:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=ACCA/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/INCA:=SP/INCAB:=SPB/
                IINC1:='1/IINC1B:='1/
                PILECRAMDEC]:=BUSA;
       $C5:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=CCR/BUSB:=SINC1;
          $PHI3:RAMDEC:=BUSB/PILECRAMDEC]:=BUSA;
```

```
      $C6:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#00/ALEH:='1;
          $PHI3:EB:=BUSA;
      $C7:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=#12;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
      $C8:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:PCH:=BUSA/EB:=BUSB;
      $C9:$PHI1:PRCHG:='1;
          $PHI2:OE:='1/BUSA:=EB;
          $PHI3:PCL:=BUSA/PCLB:=BUSA;
     $C10:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCH/ALEH:='1;
          $PHI3:EB:=BUSA;
     $C11:$PHI1:PRCHG:='1;
          $PHI2:BUSA:=PCL;
          $PHI3:EB:=BUSA/INCA:=BUSA/INCAB:=BUSA/
                IINC1:='1/IINC1B:='1;
     $C12:$PHI1:PRCHG:='1;
          $PHI2:BUSB:=SINC1/OE:='1/BUSA:=EB;
          $PHI3:IR:=BUSA/PCL:=BUSB/PCLB:=BUSB/IT:='1/GOTO;

END.
```

ANNEXE B

EXEMPLE DE SIMULATION *ILIS* DE *COBRA*

```
VAR %I,%J:INTEGER;
BEGIN
WRITELN ('**********AVANT LE PAS ********');
PRINT *;
WRITELN('**********APRES LE PAS**********');
STEP;
PRINT *;
%J:=1;
READ(%I);
ASSIGN PCL  45;
STEP;
WRITELN('***************ETAPE:',%J);
PRINT *;
READ(%I);
%J:=%J+1;
WRITELN('***************ETAPE:',%J);
STEP;
PRINT *;
READ(%I);
%J:=%J+1;
STEP;
WRITELN('**************ETAPE:',%J);
PRINT *;
READ(%I);
%J:=%J+1;
ASSIGN EB 32;
STEP;
WRITELN('****************ETAPE:',%J);
PRINT *;
END.
```

-Programme qui permet de simuler l'exécution de l'instruction étape par étape (une étape= une phase) ici on donne à PCL la valeur 45 puis on donne à EB la valeur 32, ce qui correspond au cas suivant: le programme qu'exécute le circuit est à l'adresse 45 et c'est l'opérande de BAS qui est pointé par PCL, cet opérande est lu, et c'est la valeur 32 de EB qui sera ensuite lu par IB, et ensuite le branchement est effectué.

```
*********AVANT LE PAS ********
PCL          : XxXXXXXX
ALEL         : X
RALU2        : XXXXXXXX
EB           : XXXXXXXX
IB           : XXXXXXXX
IR           : XXXXXXXX
OE           : X                                XX... :valeur indéfini
INC1         : XXXXXXXX
HB1          : XXXXXXXX
HB2          : XXXXXXXX
RINC1        : XXXXXXXX
RINC2        : XXXXXXXX
*********APRES LE PAS**********

PCL          : XXXXXXXX
ALEL         : X
RALU2        : 11111111
EB           : 11111111
IB           : 11111111
IR           : XXXXXXXX
OE           : X                  c'est la phase de précharge
INC1         : XXXXXXXX
HB1          : 11111111
HB2          : 11111111
RINC1        : 11111111
RINC2        : 11111111

*************ETAPE:          1
PCL          : 00101101     PCL:= 45
ALEL         : 1
RALU2        : 11111111
EB           : 11111111
IB           : 00101101     IB:= PCL
IR           : XXXXXXXX
OE           : X
INC1         : XXXXXXXX
HB1          : 11111111
HB2          : 11111111
RINC1        : 11111111
RINC2        : 11111111
*************ETAPE:          2

PCL          : 00101101     PCL= 45
ALEL         : 1
RALU2        : 11111111
EB           : 00101101     EB:= IB
IB           : 00101101     IB= PCL
IR           : XXXXXXXX
OE           : X
INC1         : XXXXXXXX
HB1          : 11111111
HB2          : 11111111
RINC1        : 11111111
RINC2        : 11111111

*************ETAPE:          3
PCL          : 00101101     PCL= 45
ALEL         : 1
RALU2        : 11111111
                            précharge
```

```
EB          : 11111111
IB          : 11111111
IR          : XXXXXXXX
OE          : X                    précharge
INC1        : XXXXXXXX
HB1         : 11111111
HB2         : 11111111
RINC1       : 11111111
RINC2       : 11111111

**************ETAPE:          4
PCL         : 00101101         PCL = 45
ALEL        : 1
RALU2       : 11111111
EB          : 00100000         EB := 32
IB          : 00100000         IB := EB
IR          : XXXXXXXX
OE          : 1
INC1        : XXXXXXXX
HB1         : 11111111
HB2         : 11111111
RINC1       : 11111111
RINC2       : 11111111
```

ANNEXE C

EXEMPLE DE DESCRIPTION *LUCIE*

```
NIV MD,MC,MP,MM,MG,MI,ME
FIG INCA
 FIG BOSS
  FIG RGD01
   REP(8,Y,54)
    FIGEXT RGD(0,0)
   FREP
   REP(7,Y,54)
    REC(5,54,2,9,MP)
    REC(17,54,2,9,MP)
    REC(29,54,2,9,MP)
   FREP
  FFIG
  FIG RGD01(0,4)
  FIG RGC02
   REP(8,Y,54)
    SYM(X,8)
     FIGEXT SBD(0,0)
    FSYM
    FIGEXT PMEM(4,0)
    FIGEXT SCDD(16,0)
   FREP
   REP(7,Y,54)
    REC(5,54,2,9,MP)
    REC(17,54,2,9,MP)
   FREP
  FFIG
  FIG RGC02(36,4)
  FIGEXT INC1(65,48)
  REC(32,75,36,3,MM)
  REC(60,88,7,3,MM)
  REC(32,129,36,3,MM)
  REC(60,142,7,3,MM)
  REC(32,183,36,3,MM)
  REC(60,196,7,3,MM)
  REC(32,237,36,3,MM)
  REC(60,250,7,3,MM)
  REC(32,291,36,3,MM)
  REC(60,304,7,3,MM)
  REC(32,345,36,3,MM)
  REC(60,358,7,3,MM)
  REC(32,399,36,3,MM)
  REC(60,412,7,3,MM)
 FFIG
 FIG BOSS(0,0)
 FIGEXT GENEP(156,4)
 FIG SO303
  REP(8,Y,54)
   FIGEXT SO3(0,0)
  FREP
  REP(7,Y,54)
   REC(17,54,2,7,MP)
   REC(29,54,2,8,MP)
  FREP
 FFIG
 FIG SO303(186,4)
 FIGEXT DOU8(224,0)
 REC(156,88,77,4,MM)
 REC(156,142,87,4,MM)
 REC(156,196,77,4,MM)
 REC(156,250,87,4,MM)
 REC(156,304,77,4,MM)
 REC(156,358,87,4,MM)
 REC(156,412,77,4,MM)
 FIG REG04
  SYM(X,18)
   FIGEXT REGR1(0,0)
  FSYM
 FFIG
 FIG REG05
  REP(8,Y,54)
   FIG REG04(0,0)
  FREP
  REP(7,Y,54)
  FREP
 FFIG
 FIG REG05(295,4)
 REC(142,74,52,4,MM)
 REC(144,128,50,4,MM)
 REC(144,182,50,4,MM)
 REC(144,236,50,4,MM)
 REC(144,290,50,4,MM)
 REC(144,344,50,4,MM)
 REC(143,398,51,4,MM)
 FIG GEN06
  SYM(X,30)
   FIGEXT GENEP(0,0)
  FSYM
 FFIG
 FIG GEN06(313,4)
 REC(302,34,16,4,MM)
 REC(302,88,41,4,MM)
 REC(302,142,41,4,MM)
 REC(302,196,41,4,MM)
 REC(302,250,41,4,MM)
 REC(302,304,41,4,MM)
 REC(302,358,41,4,MM)
 REC(302,412,41,4,MM)
 FIG BOS07
  SYM(X,156)
   FIG BOSS(0,0)
  FSYM
 FFIG
 FIG BOS07(343,0)
 REC(149,68,201,3,MM)
 REC(149,95,201,3,MM)
 REC(119,122,261,3,MM)
 REC(116,149,267,3,MM)
 REC(119,176,261,3,MM)
 REC(116,203,267,3,MM)
 REC(119,230,261,3,MM)
 REC(116,257,267,3,MM)
 REC(119,284,261,3,MM)
 REC(116,311,267,3,MM)
 REC(119,338,261,3,MM)
 REC(116,365,267,3,MM)
 REC(133,392,233,3,MM)
 REC(116,419,267,3,MM)
 REC(225,20,80,4,MM)
 REC(237,74,68,4,MM)
 REC(214,34,33,4,MM)
 REC(225,128,80,4,MM)
 REC(237,182,68,4,MM)
 REC(225,236,80,4,MM)
 REC(237,290,68,4,MM)
 REC(225,344,80,4,MM)
 REC(237,398,68,4,MM)
 FIGEXT COND(10,0)
 FIGEXT COND(46,0)
 FIGEXT COND(134,0)
 FIGEXT COND(239,0)
 FIGEXT COND(271,0)
 FIGEXT COND(301,0)
 FIGEXT COND(449,0)
 FIGEXT COND(485,0)
FFIG
```

ANNEXE D

EXEMPLE DE *SPICE* ET MODELES

```
******* 9-APR-9 ********  SPICE 2G.4 (22JUN81)  ********11:11:15******

INCREMENTEUR DU REGISTRE CR

****      INPUT LISTING                    TEMPERATURE =    27.000 DEG C

******************************************************************************


********** SIMULATION D'UN INCREMNTEUR AVEC UNE PORTE "OU" ET **********
********** UNE PORTE "NOR EXCLUSIF" (COBRA) ****************************
*
*
******************************************************************************
*
.MODEL ENR NMOS LEVEL=2
+VTO=0.65 KP=4.24E-5 GAMMA=0.284 PHI=0.6 CJ=1.38E-4 TOX=650E-10
+NSUB=6.84E14 XJ=0.5E-6 LD=0.35E-6 UO=800 UCRIT=6E4 UEXP=0.15
+UTRA=0.5 CGSO=1.35E-10 CGDO=1.85E-10 JS=1.0E-7 VMAX=2.5E4 DELTA=1
*
******************************************************************************
*
.MODEL DEP NMOS LEVEL=2
+VTO=-1.3 KP=4.64E-5 GAMMA=0.284 PHI=0.6 CJ=1.38E-4 TOX=650E-10
+NSUB=6.84E14 XJ=0.5E-6 LD=0.35E-6 UO=875 UCRIT=6E4 UEXP=0.15
+UTRA=0.5 CGSO=1.35E-10 CGDO=1.85E-10 JS=1.0E-7 VMAX=2.5E4 DELTA=1
*
******************************************************************************
.PLOT TRAN V(5) (0,5) V(4) (0,5) V(3) (0,5) V(2) (0,5)
.IC V(2)=5 V(3)=0 V(4)=0 V(5)=0 V(6)=-2.5
.TRAN 10NS 1300NS UIC
VDD 1 0 DC 5
V38 6 0 DC -2.5
VCARRY 4 0 PWL(0NS 0 50NS 5 100NS 5 500NS 5 550NS 0 1300NS 0)
VINPUT 3 0 PWL(0NS 0 50NS 0 550NS 0 600NS 4 1100NS 4 1150NS 0 1300NS 0)
M1 1 2 2 6 DEP W=2U L=5U AD=20P AS=20P
M2 2 3 4 6 ENR W=6U L=2U AD=20P AS=20P
M3 2 4 3 6 ENR W=6U L=2U AD=20P AS=20P
M4 1 3 5 6 ENR W=6U L=2U AD=20P AS=20P
M5 1 4 5 6 ENR W=6U L=2U AD=20P AS=20P
M6 5 0 0 6 DEP W=2U L=5U AD=20P AS=20P
.END
```

********** 9-APR-8 ********** SPICE 2G.4 (22JUN81)

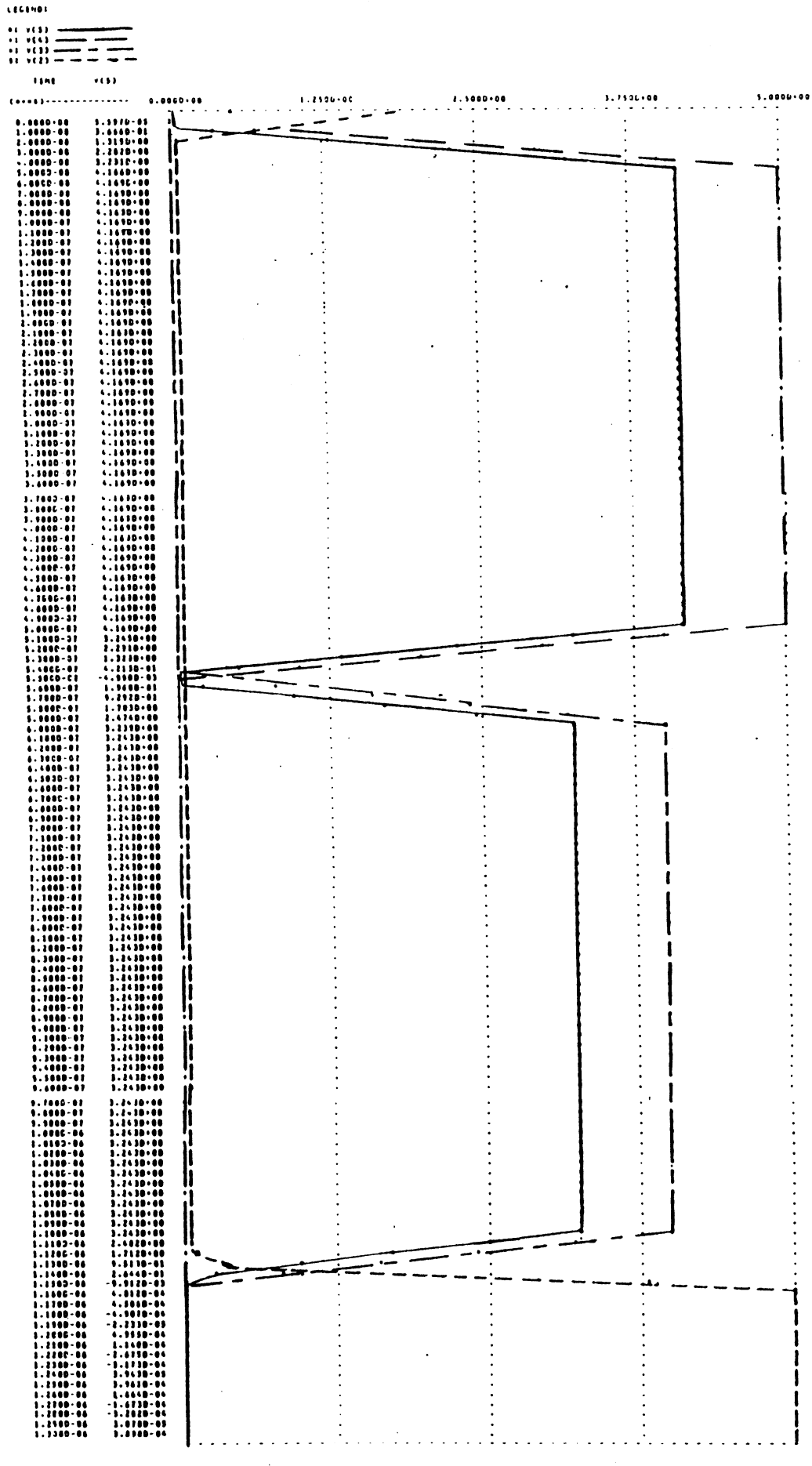
INCREMENTEUR DU REGISTRE CR

**** MOSFET MODEL PARAMETERS

****

	ENR	DEP
TYPE	NMOS	NMOS
LEVEL	2.000	2.000
VTO	0.650	-1.300
KP	4.24D-05	4.64D-05
GAMMA	0.284	0.294
PHI	0.600	0.600
CGSO	1.85D-10	1.85D-10
CGDO	1.85D-10	1.85D-10
CJ	1.38D-04	1.38D-04
JS	1.00D-07	1.00D-07
TOX	6.50D-08	6.50D-08
NSUB	6.84D+14	6.84D+14
TPG	1.000	1.000
XJ	5.00D-07	5.00D-07
LD	3.50D-07	3.50D-07
UO	800.000	975.000
UCRIT	6.00D+04	6.00D+04
UEXP	0.150	0.150
UTRA	0.500	0.500
VMAX	2.50D+04	2.50D+04
DELTA	1.000	1.000

ANNEXE D



JOB CONCLUDED

TOTAL JOB TIME 62.34

ANNEXE E

EXEMPLE DE DESCRIPTION *LUBRICK*

```
MODULE USERPROC (INPUT,OUTPUT);

CONST

%INCLUDE '[FELIN.AGUYOT.LUBRICK]CONSTLUB.INC'

TYPE

%INCLUDE '[FELIN.AGUYOT.LUBRICK]TYPELUB.INC'

%INCLUDE '[ARCHI.OSSEIRAN]PROCEDLUB.INC'

[GLOBAL]

PROCEDURE USER;

VAR I,J : INTEGER;

 BEGIN

 J:=OPENFIG('INCA  ');
 I:=OPENFIG('BOSS  ');
 I:=RIGHT(I,REPY(ADDCONN(GETFIG('RGD   '),CCE,4,17,3,4,0),8),1,4);
 I:=RIGHT(I,REPY(ADDCONN(GETFIG('RGC   '),CCE,4,30,3,4,0),8),1,4);
 I:=RIGHT(I,GETFIG('INC1  '),1,48);
 I:=CLOSEFIG(I);
 J:=RIGHT(J,I,1,0);
 J:=RIGHT(J,GETFIG('GENEP '),1,4);
 J:=RIGHT(J,REPY(GETFIG('SO3   '),8),1,4);
 J:=RIGHT(J,GETFIG('DOUB  '),1,0);
 J:=RIGHT(J,REPY(SYMX(ADDCONN(GETFIG('REGR1 '),CCW,11,30,4,4,0)),8),1,4);
 J:=RIGHT(J,SYMX(GETFIG('GENEP ')),1,4);
 J:=RIGHT(J,SYMX(I),1,0);
 J:=CLOSEFIG(J)

 END;

END.
```

- Exemple d'assemblage du module contenant les éléments PCL, SP, INC1, le générateur de parité, le registre de sortie de l'incrémenteur, le contrôleur double-rail et la duplication de tous ces éléments (sauf le contrôleur), voir figure 30 du 4ième chapitre.

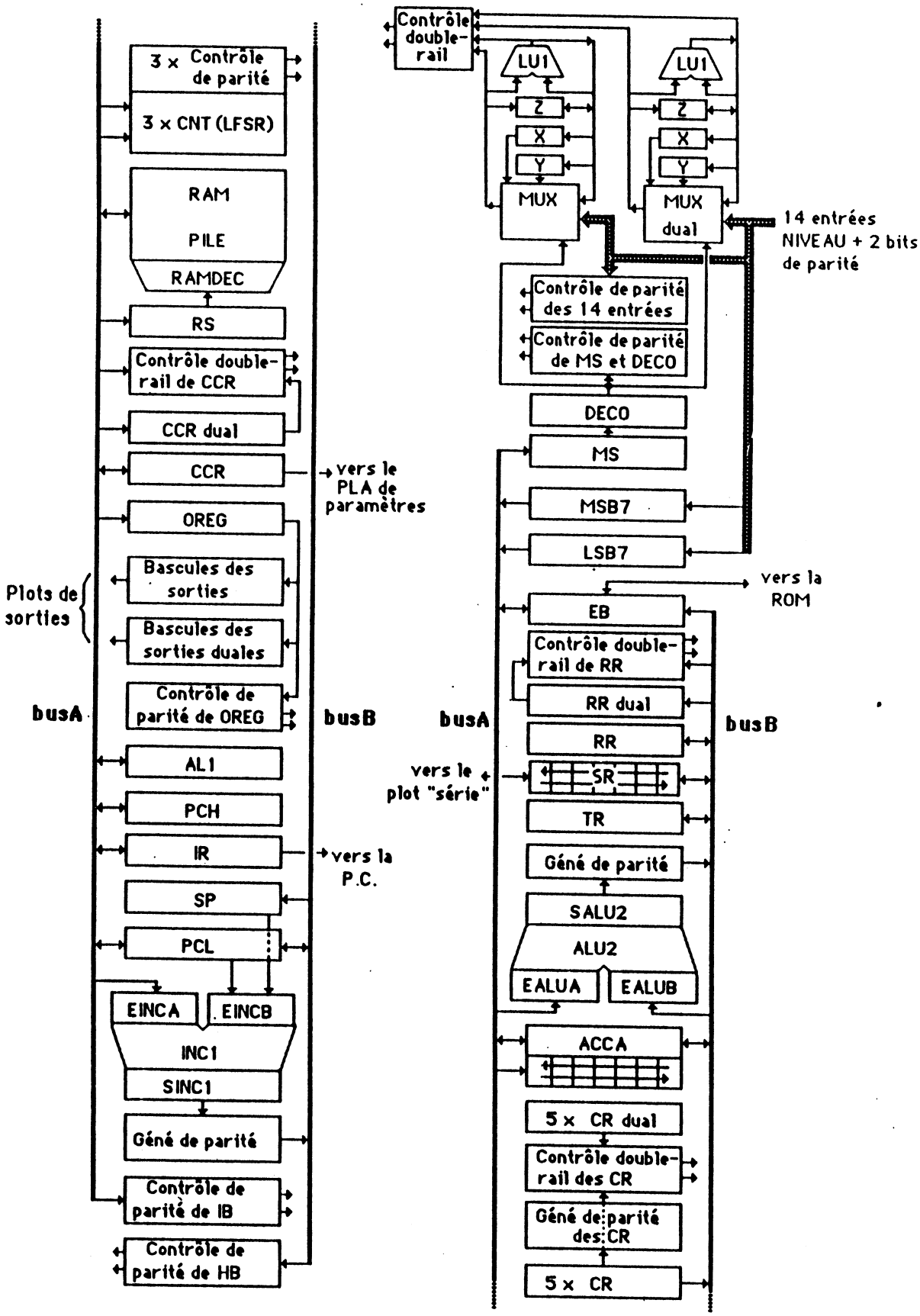
```
FIG INCA
  499  436
CW    14
    71    68     3     4     0
    71    95     3     4     0
    71   122     3     4     0
    71   149     3     4     0
    71   176     3     4     0
    71   203     3     4     0
    71   230     3     4     0
    71   257     3     4     0
    71   284     3     4     0
    71   311     3     4     0
    71   338     3     4     0
    71   365     3     4     0
    71   392     3     4     0
    71   419     3     4     0
CE    14
    71    68     3     4     0
    71    95     3     4     0
    71   122     3     4     0
    71   149     3     4     0
    71   176     3     4     0
    71   203     3     4     0
    71   229     3     4     0
    71   257     3     4     0
    71   284     3     4     0
    71   311     3     4     0
    71   338     3     4     0
    71   365     3     4     0
    71   392     3     4     0
    71   419     3     4     0
CS    16
    13     5     2     3    20
    13    17     2     3    20
    13    29     2     3     0
    13    41     2     3    20
    13    53     2     3     5
    58    65     2     3     0
    58    94     2     3     0
    11   203     2     3     5
    12   215     2     3     5
     0   298     2     3     9
     0   292     2     3     9
    13   444     2     3     5
    13   456     2     3    20
    13   468     2     3     0
    13   480     2     3    20
    13   492     2     3    20
CN    22
     0     5     2     3    20
     0    17     2     3    20
     0    29     2     3     0
     0    41     2     3    20
     0    53     2     3     5
     0    65     2     3     9
     0    94     2     3     8
     0    92     2     3     6
     0    96     2     3     2
     0   148     2     3     8
     0   203     2     3     5
     0   215     2     3     5
     0   349     2     3     8
     0   401     2     3     2
     0   405     2     3     6
     0   413     2     3     8
     0   432     2     3     8
     0   444     2     3     5
     0   456     2     3    20
     0   468     2     3     0
     0   480     2     3    20
     0   492     2     3    20
BE    1
     0     0     0     1     0
BW    1
     0     0     0     3     0
FFIG
```

- La description LUBRICK de l'incrémenteur INCA résultant de l'assemblage par le programme décrit à la page précédente. Cette description donne les renseignement sur les frontières et les connecteurs de INCA en vue de son intégration dans la structure de la partie opérative avec un programme LUBRICK.

ANNEXE F

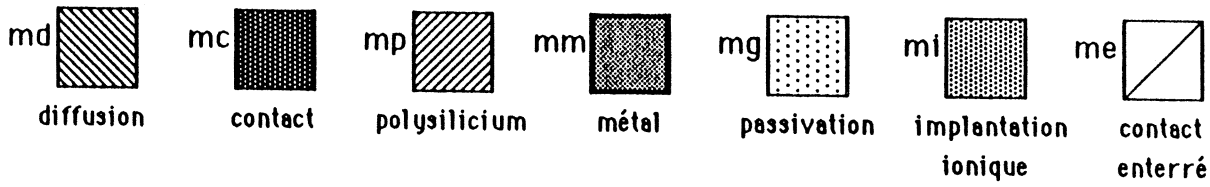
SYNOPTIQUE DE LA PARTIE OPERATIVE DE *COBRA*

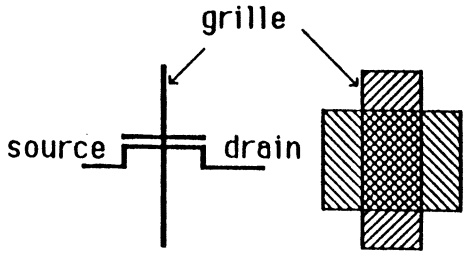
PARTIE OPERATIVE DE COBRA



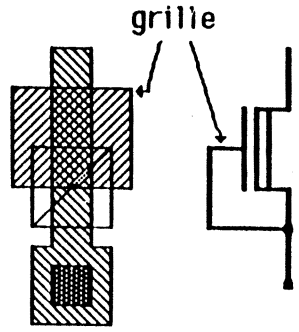
ANNEXE G

SIGNIFICATION DES NIVEAUX DES DESSINS AU MICRON

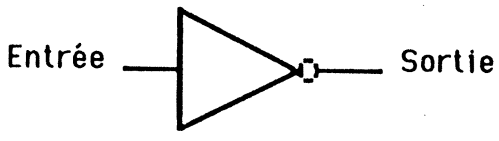


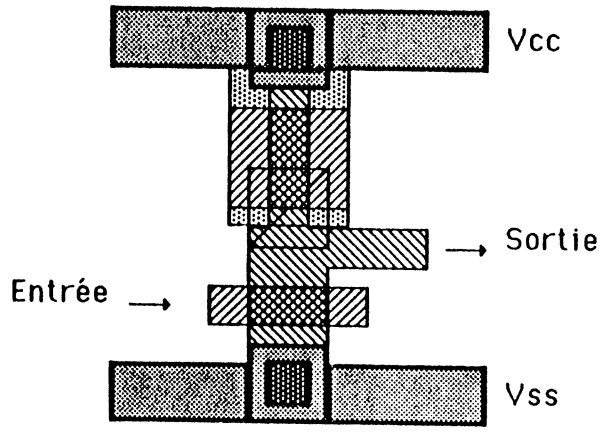


- Transistor signal -
(enrichi)



- Transistor de charge -
(déplété)





INPG département des études doctorales

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 15 TITRE III de l'arrêté du 5 Juillet 1984 relatif aux études doctorales,

VU les rapports de présentations de MM.

Guy MAZARE
Jean-claude GEFFROY

M. Adham OSSEIRAN

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE,

spécialité Microélectronique

Fait à GRENOBLE, le 29 avril 1986

Le Président de l'I.N.P.-G.
D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,