Synthèse d'architecture pour les circuits de communication : application, projet Prometheus

Hakim Saheb

THESE

présentée par

Hakim SAHEB

pour obtenir le grade de

Docteur de l'Institut National Polytechnique de Grenoble

(arrêté ministériel du 23 Novembre 1988)

Spécialité Microélectronique

Synthèse d'Architecture pour les Circuits de Communication

Application : Projet PROMETHEUS

Thèse préparée au sein du Laboratoire de Génie Informatique (**LGI**)
soutenue le 26 Novembre 1993 devant le Jury d'examen:

MM.	Guy	**Mazaré**	*Président*
	Thierry	**Maurin**	*Rapporteur*
	Denis	**Rouquier**	*Rapporteur*
	Hoang	**Nguyen**	*Examinateur*
	Michel	**Dang**	*Directeur de Thèse*

PRESIDENT DE L'INSTITUT
Monsieur Maurice RENAUD

INPG

Année 1993

PROFESSEURS DES UNIVERSITES

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

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

SITUATION PARTICULIERE

PROFESSEURS D'UNIVERSITE

DETACHEMENT

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

PERSONNES AYANT OBTENU LE DIPLOME

D'HABILITATION A DIRIGER DES RECHERCHES

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

DIRECTEURS DE RECHERCHE CNRS

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

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

à ma mère et mon père

Mes remerciements

A mon directeur de thèse, Michel Dang, Professeur de l'INPG, pour son suivi et ses conseils et aussi pour l'ambiance amicale qu'il a su créer au sein de son équipe de recherche.

Aux membres de la Commission d'examen :

Monsieur Guy Mazaré, Professeur de l'INPG et Directeur de l'ENSIMAG, pour m'avoir fait l'honneur de présider le Jury de cette thèse ;

Messieurs Denis Rouquier, Adjoint au Chef du Groupement CCI (Conception de Circuits Intégrés) du CNET/CNS, et Thierry Maurin, Professeur de l'Université Paris XI, pour avoir accepté d'être rapporteurs et pour leurs conseils qui me furent très précieux ;

Monsieur Hoang Nguyen, Ingénieur au Centre de Recherches de la RNUR, chargé du suivi des actions PRO-CHIP dans PROMETHEUS pour avoir accepté de faire partie de ce jury.

Je tiens à exprimer ma reconnaissance à tous mes amis et collègues du Laboratoire de Génie Informatique (LGI) avec qui il fut si agréable de travailler.

Je remercie également toutes les personnes que j'ai côtoyées pendant la durée de cette thèse. Grâce à leur aide, à leur amabilité et à leur générosité, j'ai pu mener à bien ce travail dans de bonnes conditions.

Table des Matières Générale

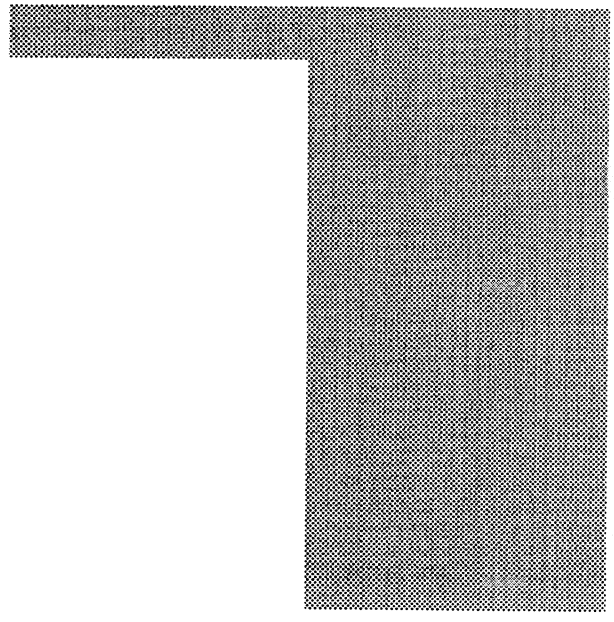


Table des Matières Générale

Introduction

Chapitre 1

Chapitre 2

Chapitre 3

Chapitre 4

Conclusion Générale

Bibliographie

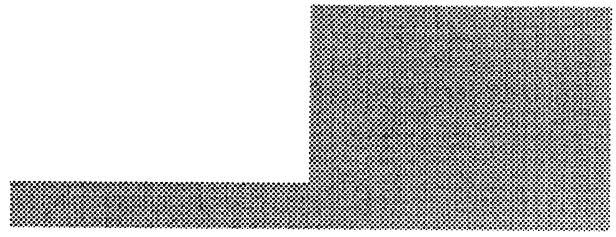
Annexe A

Annexe B

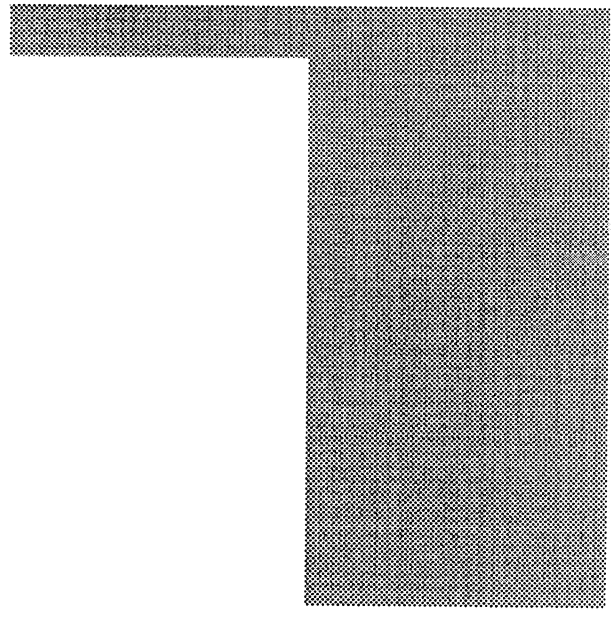
Annexe C

Annexe D

Annexe E



INTRODUCTION



— INTRODUCTION —

La Conception Assistée par Ordinateur (CAO) est devenue dans beaucoup de domaines tels que l'architecture, la mécanique et l'électronique, une démarche automatique de développement et de conception rapide et "sûre" en diminuant au maximum le temps qui sépare le cahier des charges de la réalisation du projet.

Dès qu'on parle de CAO, on ne peut s'empêcher d'évoquer ce qu'elle a apporté dans le domaine de conception des Circuits Intégrés (la microélectronique). Avec l'intégration à très large échelle (VLSI) , le nombre de transistors qu'on peut intégrer dans une puce a augmenté très rapidement en quelques années. Du fait de la complexité des circuits, les concepteurs ont vite abandonné la conception à la main (*full custom*) pour se tourner vers la conception assistée en automatisant les différentes étapes du processus de conception.

Le terme Compilation de Silicium est apparu pour désigner des méthodologies de conception partant de spécifications externes pour aboutir à la génération du dessin des masques d'un circuit.

La synthèse d'architecture (parfois appelée synthèse comportementale ou encore synthèse de haut-niveau) est une étape de conception permettant le passage d'une description comportementale du fonctionnement d'un circuit à une description structurelle (RTL, ...).

Dans le domaine de la communication, la nécessité du traitement rapide des messages et l'augmentation du débit de plus en plus élevé dans les réseaux ont nécessité l'implémentation en circuits des protocoles de communication de niveau bas. Cette nécessité devient évidente dans le domaine des réseaux industriels où le temps-réel et le coût bas des équipements sont des critères d'implémentation à respecter.

L'objectif de notre étude est d'implémenter d'une manière automatique les protocoles de communication de niveau MAC (*Medium Access Control*) à partir d'une spécification comportementale du protocole afin de réaliser un circuit appelé : Circuit de Communication. Le niveau MAC joue un rôle déterminant dans le domaine des réseaux locaux et les réseaux industriels. L'analyse des différents protocoles a été faite dans notre équipe, dans le cadre de la thèse de Mr Imad Sabouni.

Cette étude est menée dans le cadre de la participation de notre équipe au projet européen EURÊKA PROMETHEUS (*PROgraM for European Traffic with Highest Efficiency and Unprecedented Safety*) qui nous a servi de champ d'application.

Dans le projet PROMETHEUS se regroupent 14 constructeurs automobile européens (dont PSA et RNUR en France) ainsi que plusieurs sociétés d'électronique et laboratoires universitaires.

Notre participation dans ce projet s'insère dans le cadre de l'action de recherche de base PRO-CHIP (*Custom Hardware for Intelligent Processing*). Nous sommes responsables essentiellement de la spécification des interfaces, de la spécification des protocoles de contrôle-commande, ainsi que de l'implantation matérielle de l'ensemble de la communication dans le cadre de l'étude de la faisabilité technique d'un système d'aide à la navigation dans une voiture intelligente (projet de démonstration "Détection d'Obstacles").

Ce mémoire de thèse se présente de la manière suivante:

— Le chapitre 1 présente la spécification de protocoles de communication en vue de leur implémentation. Il permet de présenter la démarche de spécification en général et puis la spécification des protocoles de communication en particulier.

— Dans le chapitre 2, nous présentons l'architecture cible permettant d'implémenter les couches basses des réseaux de contrôle-commande avec les différentes fonctionnalités du circuit de communication.

— Dans le chapitre 3, nous décrivons les différentes étapes de la démarche de synthèse en partant de la description d'un protocole dans un langage (appelé MACS) pour générer l'architecture du circuit de communication correspondant.

— Le chapitre 4 permet de mettre en évidence la démarche de conception d'un circuit de communication en prenant comme exemple d'application un protocole de communication des réseaux intra-véhicule.

— Nous terminons par une conclusion générale et par les perspectives de cette étude.

Une première étude des différentes techniques de synthèse d'architecture a été réalisée et présentée dans la partie annexe A afin d'alléger la lecture du rapport. Cette étude présente un panorama des démarches en synthèse de haut niveau.

Chapitre 1

Spécification de Protocoles de Communication en Vue de leur Implémentation

Chapitre 1

SPECIFICATION DE PROTOCOLES DE COMMUNICATION EN VUE DE LEUR IMPLEMENTATION

Introduction

Dans ce chapitre, une première approche de formalisation des différentes caractéristiques des protocoles de communication sera présentée. Celle-ci nous permet de faire une synthèse générale de notre domaine d'application sous forme d'une représentation formelle des objets des protocoles de communication pour réaliser un modèle de représentation interne.

A partir de ce modèle de représentation interne, le concepteur pourra bâtir tout un environnement de développement pour la spécification, la validation, la simulation et l'implémentation des protocoles de communication :

. la spécification : Le concepteur peut spécifier son protocole de communication sous forme d'un langage de description (ou graphique) des protocoles de communication. De cette description, une représentation interne est construite, correspondant aux différents objets de communication qui interviennent dans la spécification du protocole.

· la conception de l'architecture cible: A partir des objets de communication (trames, champs,...), une architecture d'implémentation associée peut être définie (conçue) afin d'implémenter d'une manière systématique les protocoles de communication. L'architecture cible peut être de type matériel ou logiciel. Celle-ci dépend d'une part des différents traitements à réaliser et d'autre part des performances attendues (contraintes temporelles, coût, ...).

· la simulation : De cette architecture, le concepteur peut effectuer des simulations comportementales afin de réaliser des évaluations de performance de l'architecture générée. Pour qu'on puisse réaliser des simulations, il faut que le code généré, si code généré il y a, soit exécutable dans le sens de la simulation. Un exemple type est un langage de programmation (ex: ADA, C, ...) ou un langage de description de matériel (ex: VHDL, langages HDL).

· la validation : L'approche de validation par simulation permet à l'utilisateur d'avoir une vue plus opérationnelle du système qu'il modélise (le fonctionnement est-il bien celui attendu ?). Le choix en faveur de la validation par simulation vient du fait que les systèmes de preuve ne valident qu'une représentation de l'algorithme réel décrit dans leur propre formalisme. Cette représentation ne peut être très précise car il existe généralement une grande distance entre la sémantique d'un modèle théorique et l'environnement ou le contexte d'implantation pour lequel le protocole est conçu.

· l'implémentation : Une fois la conception validée, les objets de communication qui auront été ainsi spécifiés de manière exécutable, sont interprétés et traduits automatiquement dans une implémentation ciblée.

1 Spécification et Formalisation des Systèmes

Le but de cette section n'est pas de présenter une taxonomie des différents types de spécifications ni des méthodes formelles employées pour la spécification des systèmes dans le domaine du génie-logiciel. Une présentation de l'état de l'art dans le domaine de spécification de système et des méthodes formelles utilisées est présentée afin d'avoir une vue générale des approches formelles de description des systèmes complexes.

1.1 Langage de Spécification

L'étape de spécification est universellement reconnue comme étant cruciale dans le processus de développement de logiciel ou de matériel [ChGa88]. L'utilisateur et le concepteur doivent employer le même vocabulaire pour décrire un système. Cette description est une spécification qui peut servir de contrat, de documentation et de moyen de communication entre un client, l'implémenteur et le concepteur qui définit la spécification [Wing90].

1.1.1 Spécification

Aujourd'hui, on peut trouver quatre classes de spécifications selon leur forme [Brac88] :

- les spécifications informelles, c'est-à-dire rédigées en toute liberté par leur auteur,

- les spécifications standardisées, c'est-à-dire utilisant toujours essentiellement la langue naturelle comme forme d'expression, mais se pliant à un plan type et éventuellement à divers standards méthodologiques : schéma imposés, notations normalisées, glossaire, index, ...

- les spécifications semi-formelles, qui utilisent un "langage de spécification" textuel ou graphique. C'est un langage doté d'une syntaxe qui est en général précise, et d'une sémantique souvent assez faible, mais suffisante pour permettre l'automatisation de certaines vérifications, ou l'aide à la consultation; des exemples de telles techniques sont: SADT[Liss], SA [Marc79] aux états-unis.

- les spécifications formelles, exprimées dans un langage à syntaxe et sémantique précises, construit sur une base théorique et qui permettent des validations automatisées. Les réseaux de Pétri, les grammaires formelles, les automates à états finis, la logique formelle, l'algèbre ... sont autant d'exemples de telles techniques.

Les deux premiers types de spécification sont aujourd'hui largement majoritaires, avec un transfert important depuis quelques années des spécifications informelles vers les spécifications standardisées.

Cependant, dans certains domaines d'activité, les spécifications formelles ont acquis un droit de cité incontestable:

. les réseaux de Pétri sont largement utilisés pour spécifier et valider les protocoles de communication,

. les grammaires formelles sont utilisées pour définir et analyser les langages, en particulier les langages de programmation,

. les automates à états finis permettent la modélisation de mécanismes critiques dont la validation formelle est estimée nécessaire (moniteurs temps réel, postes de commande, ...).

1.1.2 Spécification Formelle

Selon [ChGa88], une spécification est formelle si elle est écrite en utilisant une notation (un langage, un graphisme, ...) complètement définie, précise, non ambiguë. Plus formellement, cette notation a non seulement une syntaxe mais aussi une sémantique. Cette sémantique établit quelles sont les implémentations possibles pour une spécification donnée.

1.2 Classification des Spécifications et des Méthodes Formelles

1.2.1 Classification des Spécifications:

Toujours selon [ChGa88], il y a trois grandes classes de spécifications formelles:

- *les spécifications opérationnelles*

Ils permettent de décrire l'enchaînement des actions à réaliser au moyen d'algorithmes, d'automates ou autres systèmes de transition. L'implémentation doit simplement assurer que le système aura le même comportement, selon les critères d'observation donnés, que sa spécification. Les réseaux de Pétri sont un exemple de spécification opérationnelle.

- *les spécifications axiomatiques*

Ils décrivent l'ensemble des propriétés que doivent satisfaire les implémentations. La signification d'une telle spécification est la classe de toutes les implémentations qui satisfont ces propriétés et éventuellement

quelques hypothèses propres à la méthode de spécification. Dans les langages de spécification algébrique comme PLUSS[Gaud83] [BiGa85] et LARCH [GuHo83] [Horn85], on écrit des spécifications axiomatiques.

- *les spécifications relationnelles*

Un univers est décrit sous forme d'un ensemble d'objets et de relations entre ces objets, de règles décrivant la façon dont cet univers peut évoluer par certaines actions ainsi que des conditions sous lesquelles elles peuvent être invoquées. C'est une approche qui a été choisie dans le système GIST [BaGW82] [CoSB82].

Selon ses auteurs, cette classification est discutable car les spécifications relationnelles peuvent être aussi considérées comme des spécifications axiomatiques. Ainsi, des spécifications axiomatiques ou opérationnelles sont obtenues avec un langage comme VDM [Jone86] selon la manière dont il est utilisé.

1.2.2 Classification des Méthodes Formelles

Une proposition de classification des méthodes formelles en deux grandes classes par [Wing90] sont :

- *Les méthodes orientées modèle*

Le comportement du système est spécifié en construisant le modèle de ce système en termes de structures mathématiques (suites, fonctions, ...). VDM (Vienna Development Method), Z [Abri84] font partie de cette classe.

Il existe d'autres langages qui permettent de spécifier les systèmes concurrents et distribués : les réseaux de Pétri, CCS (a Calculus of Communicating Systems) [Miln80], ESTELLE (Extended State Transition Language) [Cour89], ESTEREL [BeCo85] [BeCG87], logique temporelle [AuFC87], RAISE [NHWG89].

- *Les méthodes orientées propriété*

Le comportement du système est dans ce cas spécifié indirectement en représentant un ensemble de propriétés que le système doit satisfaire, généralement sous forme d'un ensemble d'axiomes.
Cette classe peut être divisée en deux catégories appelées méthodes axiomatiques et méthodes algébriques.

Les premières sont basées sur les preuves d'implémentation. La logique de premier ordre, les pré-conditions et les post-conditions sont utilisées pour la spécification de chaque opération. LARCH , OBJ [Futa85], Anna [Luck85] sont des langages qui supportent ces méthodes.

Les méthodes algébriques sont spécialement adaptées à la description de classe d'implémentation. La représentation d'une spécification algébrique définit une classe d'algèbre. CLEAR [BuGo86], Act One [EhMa85] et PLUSS sont des langages algébriques.

1.3 Les Langages Visuels

Les langages visuels sont obtenus en ajoutant au domaine syntaxique des éléments graphiques [Wing90]. On peut citer comme méthodes visuelles les réseaux de Pétri pour spécifier le comportement des systèmes concurrents. Un langage visuel basé sur les "*States Charts*" a été utilisé pour spécifier des transitions d'états dans des systèmes réactifs. Le formalisme des *States Charts* créé par HAREL est dérivé des diagrammes de Venn et du formalisme des Higraphs [Hare88].

1.4 Spécification Formelle Exécutable

Si une spécification formelle est exécutable, elle permet d'obtenir rapidement un prototype, une maquette, du système à développer. Une spécification est exécutable si elle est exprimée dans un langage pour lequel il existe un interprète ou le fait de pouvoir simuler la réponse du modèle en suivant tous les scénarios possibles ou si, par une suite de transformations automatiques, elle peut être transformée en une implémentation [Smol82] [ChGa88].

Cependant, un langage de spécification exécutable doit tenir compte de la machine sur laquelle il est exécuté et devient pour cela, plus restrictif dans son pouvoir d'expression. Il peut jouer un rôle important dans la phase de

conception. Le concepteur peut à tout moment revenir en arrière dans la spécification au vue de son exécution.

La spécification a l'avantage par rapport au prototype d'être indépendante de toute implémentation afin de garder un niveau d'abstraction plus élevé.

2 Les Spécifications dans le Domaine de la Communication

Dans le domaine de la communication, plusieurs langages de spécification ont été utilisés pour décrire les protocoles de communication. On se réfère souvent à ces langages comme étant des langages pour la spécification des systèmes concurrents où plusieurs processus communiquent pour échanger des messages d'informations ou de synchronisation. Parmi ces langages, il y a: Act One, CCS (*a Calculus of Communicating Systems*), CSP (*Communicating sequencial processes*), ESTELLE (*Extended State Transition Language*), ESTEREL, LOTOS et SDL.

L'ISO et le CCITT (Comité Consultatif International Télégraphique et Téléphonique) ont étudié depuis quelques années des techniques de description formelle (TDF) pour la spécification de protocoles. Ce travail était motivé par le fait que les ambiguïtés dans la description des normes conduisent souvent à des mises en œuvre incompatibles, un résultat en contradiction avec la normalisation.

L'ISO a retenu deux types de techniques. Une est LOTOS, l'autre est ESTELLE. La première proposition de norme pour ESTELLE et LOTOS a été produite en 1985. Ces langages sont ainsi destinés à être utilisés pour la description formelle des protocoles normalisés.

ESTELLE est basé sur les machines à états étendus dans lesquelles la communication entre deux modules se fait à travers un canal reliant deux ports bidirectionnels. ESTELLE permet de décrire l'interaction non déterministe qui résulte de l'échange de messages entre deux modules grâce à une queue gérée en FIFO [Dia87].

LOTOS, basé sur CCS [Miln80], fait intervenir des processus (eux-même composés d'autres processus) qui communiquent avec l'environnement grâce à des interactions dont la forme est le Rendez-vous. Il est fondé sur la description

de l'ordonnancement temporel des interactions. Il peut être vu comme une extension de CCS supportant la notion du type abstrait [Loto87].

• Implémentation Matérielle à Partir d'une Spécification de Protocole de Communication

Des travaux sur l'implémentation automatique en partant d'une spécification d'un protocole de communication pour arriver à une implémentation logicielle sont partiellement réalisés et d'autres sont en cours. Généralement, ils utilisent un langage de description de type ESTELLE car il permet de faire une description structurelle du système en introduisant des modèles d'automates pour décrire le comportement interne.

Par contre, en ce qui concerne les implémentations matérielles, jusqu'à aujourd'hui (à notre connaissance), un seul travail a été réalisé par [KrKS87] [KrKS89] [Kris92] de AT & T Bell Laboratories permettant d'implémenter d'une façon statique les protocoles de communication. Pour cela, deux approches qui se suivent dans le temps ont été adoptées pour décrire et implémenter les protocoles.

Une première approche [KrKS87] consistait à utiliser un éditeur graphique pour la spécification du format des messages (trames) du protocole. Ceci permet d'extraire les différents champs ("*tokens*") de la trame pour exécuter les divers traitements associés. L'implémentation est réalisée sur une architecture cible où certaines parties susceptibles à la modification sont maintenues programmables. Ces parties concernent le contrôle de l'analyse des formats des trames et sont implémentées en machine d'état fini. Les informations relatives à la spécification du format de la trame sont stockées dans une RAM et sont gérées par un séquenceur. Un exemple de spécification d'une trame du protocole LAPD est donné dans la figure 1.

La deuxième approche [Kris92] qui est utilisée pour la synthèse des implémentations matérielles, consiste à employer un langage de spécification formelle APSL (*Augmented Protocol Specification Language*). Selon ses auteurs, APSL est similaire dans ses modèles de base au langage de spécification ESTELLE avec des différences importantes. Il a les caractéristiques qui sont basées sur:

. un modèle de machine d'état fini étendue : ensemble d'états et variables de données de contextes,

. des communications synchrones (pas de queues implicites entre processus),

. pas de notion d'hiérarchie (une structure éclatée),

. pas de création et de destruction dynamique des processus.

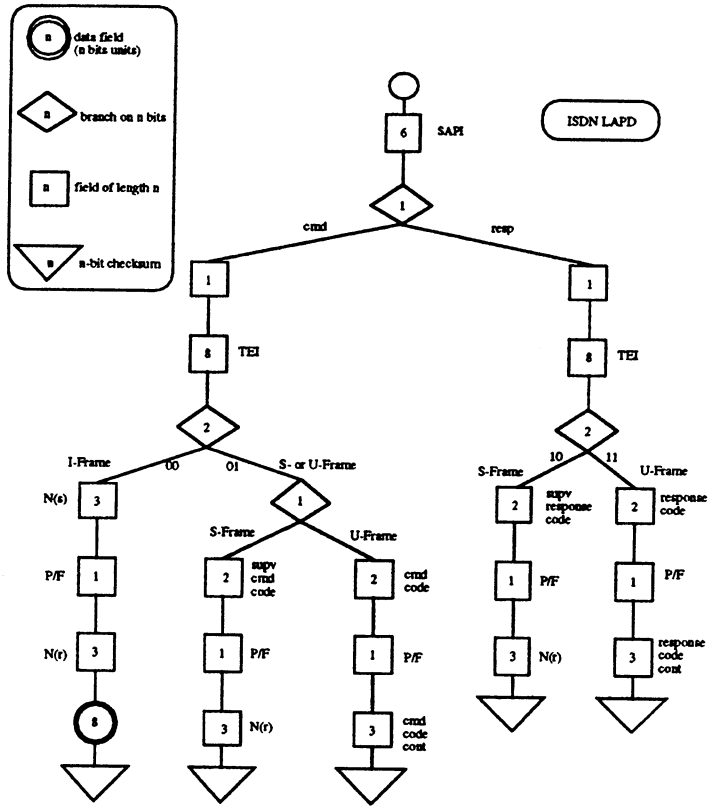


figure 1: exemple de spécification du format d'un message

D'après les auteurs, ces simplifications n'ont pas restreint l'utilisation de APSL en pratique puisque plusieurs protocoles standards comme X.25, FDDI, LAPD etc. ont été spécifiés en APSL. En plus, des directives pour contrôler le processus de synthèse ont été ajoutées.

Une spécification d'un processus en APSL est présentée comme suit:

EXERNAL *proc1, proc2, ... , proc#m ;*
PROCESS *process_name ;*
Declarations
TRANSITIONS

(*Transition statements*).
(*Subroutines of this process*).
(*Other PROCESS definitions*).

Les déclarations sont de type :

STATES *no—n1, n2—n3, n4, ..., nk—nl ;*
MINOR *n2—n3, n4 ;*
CONTEXT *name0, name2, name3(i), ..., namek ;*
REND *src?msg1, src?msg2, ... dst!msga, dst2!msgb ;*
INITIAL STATE *N0 ;*

Il n'est pas nécessaire d'utiliser toutes les instructions pour définir un processus; les déclarations *EXTERNAL* et *MINOR* sont optionnelles. La signification des différentes instructions est :

STATES : définit les noms des états du processus. Un intervalle n1—n2 indique tous les nombres compris entre n1 et n2 inclus.

MINOR : indique les états du processus qui doivent être traités comme des pseudo-états.

CONTEXT : définit toutes les variables d'état utilisée dans la spécification.

REND : (pour RENDEZ-VOUS) définit les différents message d'entrée/sortie qui sont visibles extérieurement.Ce sont les signaux à travers lesquels le processus communique avec d'autres processus. Les messages sont de la forme *a?m1* ou *b!m2* où *a?m1* indique la réception du message *m1* de la machine *a* et *b!m2* indique l'émission du message *m2* vers la machine *b*.

INITIAL STATE : cette instruction permet d'indiquer l'état initial qui ne doit pas être un MINOR.

TRANSITIONS : la forme générale d'une transition est :

```
i:j WHEN {condition} Input*Output [update-ops]
```

la notation `i:j` indique une transition de l'état i vers l'état j. la condition `{condition}` est un test utilisant une expression conditionnelle contenant des variables internes. La mise à jour des variables internes est effectuée à travers `[update-ops]`.

En plus, il y a quelques déclarations spécifiques au processus de synthèse permettant de contrôler de le contrôler:

```
.HOLD name0, name2, name3[1], ..., namek ;
.INPUT name0, name2, name3[1], ..., namek ;
.SAME common_name  dst1!out1, dst2!out2, ... ;
```

Directive *HOLD* : permet de rendre une variable interne visible à l'extérieur. Elle sera vue comme étant un signal externe.

Directive *INPUT* : indique qu'une variable déclarée dans le CONTEXT est réellement un signal d'entrée.

Directive *SAME* : Pour vérifier que les entrées d'un processus et les sorties d'un autre sont confondues.

La syntaxe et la définition des différentes instructions et directives sont décrites dans [Kris92].

3 Modèle pour une Spécification Exécutable des Protocoles de Communication

3.1 Domaine d'Application

Afin de réaliser le processus d'implémentation des protocoles de communication de niveau bas à partir d'une description de haut niveau, une approche similaire à celle des techniques déjà annoncées a été abordée. Pour spécifier un protocole de communication en vue de son implémentation et pour être indépendant de la manière dont le protocole sera implémenté, on s'est intéressé à la spécification des différents objets qui interviennent dans un protocole de communication plutôt qu'à la façon de les implémenter. Pour cela, on est parti de l'élément essentiel dans un protocole de communication qui est la "trame".

Ainsi, une analyse globale des fonctionnalités d'un protocole de communication est réalisée en les décrivant d'une manière abstraite tout en faisant surtout référence aux fonctionnalités de niveau MAC (Medium Access Control) des normes IEEE 802 [IEEE85] car il s'agit bien de notre domaine d'application.

3.2 Analyse Globale

Les réseaux de communication, en particulier les réseaux locaux, permettent d'interconnecter des équipements informatiques, bureautiques et/ou industriels au sein d'une zone géographique limitée (usine, société, laboratoire, ...). Le système de communication réparti comporte alors un ensemble de sites qui sont vus comme étant un ensemble d'entités communicantes. Chaque entité est organisée d'une manière hiérarchique dans un modèle à couches afin de séparer les différentes fonctions que le système doit accomplir. Ce concept a été utilisé par l'ISO (*International Standardization Organization*) pour le développement des réseaux "normalisés" afin d'intégrer une variété de systèmes informatiques hétérogènes. Ceci a abouti au modèle OSI (*Open System Interconnect*) comportant sept couches dont chacune est indépendante des voisines.

Un service représente les informations échangées entre deux couches consécutives. Les échanges entre des couches de même niveau N sont appelées protocole niveau N. Les normes OSI spécifient uniquement ces deux éléments : Services et protocoles.

3.2.1 Protocole de Communication

Dans un réseau de communication, deux ou plusieurs entités du même niveau communiquent à travers le niveau ci-dessous en utilisant un ou plusieurs types de trames. L'exemple de la figure 2 présente un échange de trames entre deux entités de niveau MAC.

Le protocole de communication entre entités de même niveau N est spécifié par la description des processus d'émission et de réception des différentes trames. On distingue trois types de communication : communication avec le niveau N+1 (couche du dessus), communication avec le niveau N-1 (couche du dessous) et la communication entre entités (entité de même niveau N). Ainsi, chaque processus se décompose en trois parties : partie interface avec le niveau haut, partie traitement du protocole et la partie interface avec le niveau bas (figure 3).

Dans le cas d'une entité de niveau MAC, le niveau ci-dessous est la couche physique et le niveau du dessus est la couche LLC ou l'application.

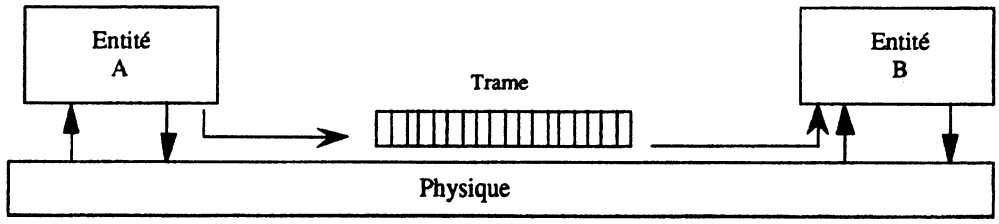


figure 2: entités communicantes

La communication entre les entités est réalisée à travers des trames d'informations. Chaque trame regroupe un certain nombre de données envoyées à l'entité de destination. Différents types de trames peuvent être utilisés dans un protocole de communication pour établir une communication, demander des données, envoyer un accusé de réception, terminer la communication... .

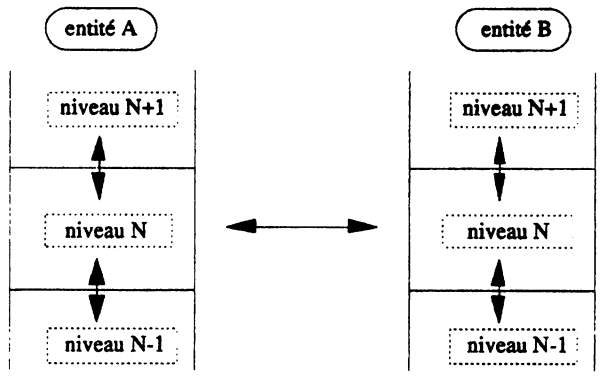


figure 3: protocole et service dans une entité

La complexité d'un protocole de communication dépend essentiellement de ses différents types de trames et des traitements associés.

3.2.2 Emission

Une trame est générée en émission par la concaténation d'un ensemble de champs. Ceux-ci sont choisis selon le type de la trame à envoyer. L'ensemble de ces champs définit la structure interne de la trame. Le processus d'émission consiste alors à générer des trames à partir d'un ensemble de champs.

La figure 2 présente le processus d'assemblage des champs pour construire une trame. La source d'un champ peut être soit:

. l'interface haute (niveau N+1),
. ou le protocole de communication du même niveau (N).

figure 4: assemblage de trames

Dans la figure 4, la trame n'est pas nécessairement composée de tous les champs et ne sont pas fournis nécessairement d'une manière parallèle. Cette représentation "éclatée" ne sert qu'à illustrer le processus d'assemblage d'un ensemble de champs.

3.2.3 Réception

Une trame est reconnue et traitée en réception par une entité réceptrice dès son arrivée. Le processus de réception consiste à analyser la trame par la reconnaissance et le traitement de ses différents champs. La reconnaissance d'une trame consiste à délimiter et à extraire les différents champs dont elle est composée.

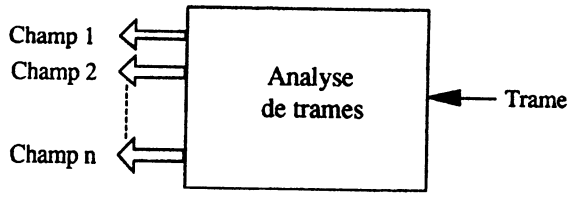


figure 5: analyse d'une trame

La figure 5 représente le processus qui permet de désassembler une trame en plusieurs champs. Comme dans le cas de l'émission, les différents champs de la trame peuvent avoir deux destinations : la première est celle de l'interface haute (N+1), et la seconde est celle du protocole de communication du même niveau (N).

3.2.4 Gestion d'une trame

Dans le cas de l'émission, le processus de génération de trames est appelé processus d'assemblage de trames. Par contre dans le cas de la réception, le processus d'extraction de champs d'une trame est appelé processus d'analyse de trames. Ces processus sont tous les deux basés sur deux types d'informations: la syntaxe et la sémantique interne des trames.

- *La syntaxe interne des trames* :

Elle permet de représenter une trame sous la forme de son squelette. C'est un ensemble de champs représentant le format ou la structure d'une trame. La connaissance de la syntaxe de la trame permet d'indiquer les limites des différents champs dont elle est composée.

- *La sémantique interne des trames* :

Elle correspond aux différentes fonctions activées durant le traitement d'une trame que ce soit en émission ou en réception. Chaque champ peut être traité par une ou plusieurs fonctions de traitement. Ainsi, la connaissance sémantique de la trame permet de préciser l'ensemble des actions associées aux différents champs de la trame.

La gestion de la trame en émission et en réception est représentée dans la figure 6 . Chaque processus est décomposé en deux parties interdépendantes: une partie qui s'occupe de la gestion syntaxique de la trame permettant d'indiquer les différents traitements associés à chaque champ, et une partie sémantique réalisant le traitement demandé et signalant à l'analyseur syntaxique les résultats et l'état du traitement courant.

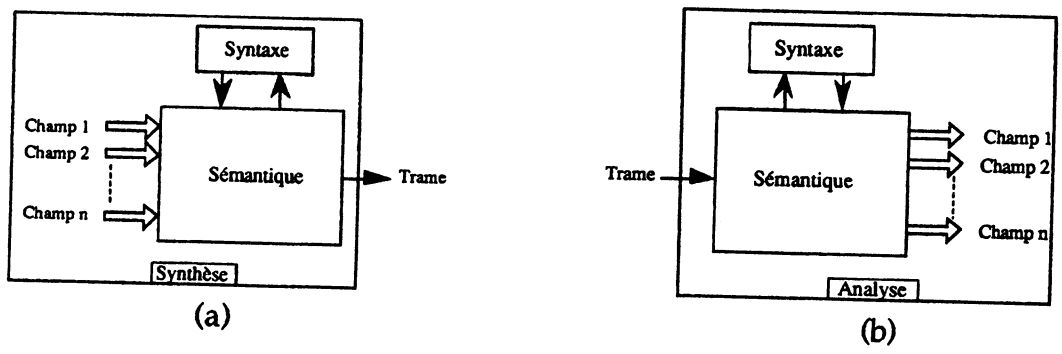


figure 6 : (a) traitement syntaxique et sémantique du processus d'émission.
(b) traitement syntaxique et sémantique du processus de réception.

On se propose dans ce qui suit, de spécifier la gestion syntaxique et sémantique des différentes trames du protocole.

3.3 Gestion Syntaxique

L'élément essentiel considéré durant le traitement d'un protocole de communication est l'objet de base, traité en émission et en réception, qui est la trame. Plusieurs types de trames peuvent être traités dans un protocole de communication.

Soit :

$\mathcal{T} = \{ T_1, T_2, \ldots, T_n \}$ l'ensemble des trames d'un protocole

La différence entre deux trames Ti et Tj est indiquée par la différence des champs dont elles sont composées ou par la différence des traitements associés.

3.3.1 La Trame

Une trame est définie par sa structure (ou format), c'est-à-dire par l'ensemble des champs Ci dont elle est composée.

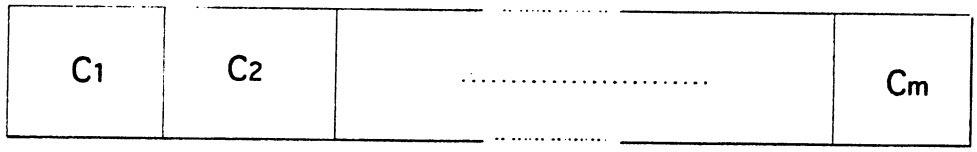


figure 7 : une trame = ensemble de champs

Un ou plusieurs champs peuvent être utilisés dans plusieurs trames différentes. On définit alors l'ensemble $\mathcal{C}$, qui est l'ensemble de tous les champs de toutes les trames d'un protocole de communication donné.

Alors :

$$\mathcal{C} = \{ C_1, C_2, \ldots, C_m \}$$

Une trame Tk est représentée donc par:

$$T_k = \{ C_i \} \quad \text{avec} \quad T_k \in \mathcal{T} \quad \text{et} \quad C_i \in \mathcal{C}.$$

3.3.2 Le Champ

Définition:

Un champ est défini comme étant un ensemble d'unités d'éléments binaires (*eb*) . Toute unité de *eb* est caractérisée par son type, sa valeur et sa durée. Par exemple, dans le cas d'un champ de niveau MAC ou supérieur, l'unité est un *eb* de valeur égale à "0" ou "1".

Dans le cas d'un champ de niveau physique, un codage Manchester, NRZ ou autre, est un exemple d'autres types d'unités *eb* dont les caractéristiques dépendent du codage physique choisi et dont la durée est d'une unité *eb* [Macc87].

3.3.2.1 Source et Destination d'un Champ

Un champ peut être :

· du niveau supérieur: dans ce cas, il existe un champ dans la trame qui est fourni par la couche supérieure (interface haute). Ce champ est inséré dans la trame et envoyé vers la couche inférieure.

· du même niveau de traitement du protocole: ce champ est produit par l'entité elle même et inséré dans la trame pour quelle soit envoyée vers la couche inférieure.

Dans le cas de la couche MAC, le champ envoyé est un vecteur d'éléments binaires (*eb*), envoyé par le niveau MAC ou par la couche supérieure (LLC ou application). Sa taille, constante ou variable, s'exprime en nombre d'éléments binaires. Il est envoyé vers la couche physique ensuite codé par celle-ci pour qu'il soit émis sur la ligne de communication.

Les champs générés par la couche physique sont appelés des champs de synchronisation dont le codage peut être une violation de code de l'élément binaire. C'est un champ de taille constante qui permet généralement de délimiter les différents champs de type vecteur de *eb* . Sa taille s'exprime en nombre de *eb* , et sa valeur dépend du codage utilisé.

3.3.2.2 Caractéristiques d'un champ

On peut distinguer trois paramètres permettant de caractériser un champ:

- son type, sa taille et sa valeur.

3.3.2.2.1 Type d'un champ

Généralement, un champ Ci est de type vecteur d'unités de *eb* :

$$Ci = \{ b_0, b_1, b_2, ..., b_n \}$$

Les valeurs des éléments binaires *bi* sont "0" ou "1" lorsque le champ Ci appartient aux protocoles des couches au-dessus de la couche physique. Dans le cas contraire, les éléments binaires *bi* du champ Ci ont une codification qui n'est pas nécessairement binaire.

A) Soit $\mathcal{M}$, l'ensemble des champs de niveau MAC de type vecteur de *eb* , on peut dire alors que :

si $Ci \in \mathcal{M}$, alors $Ci = \{ b_0, b_1, b_2, ..., b_n \}$ avec $b_i \in \{ '0', '1' \}$

B) soit $\mathcal{P}$, l'ensemble des champs de niveau physique où :

$$\mathcal{P} = \{ S_0, S_1, S_2, ..., S_m \}$$

avec Si : est un champ de synchronisation ou de données de niveau physique dont le codage peut être non-binaire et de taille un multiple d'unités de *eb* .

Un champ de niveau physique $Si \in \mathcal{P}$ est vu au niveau Mac comme étant un champ dont la taille est un multiple d'unités de *eb* .

3.3.2.2.2 Taille d'un champ

C'est le nombre d'unités *eb* que comprend ce champ. Dans le cas d'un champ physique, la taille d'un champ s'exprime en nombre de *eb* équivalents.

figure 8 : taille d'un champ
(a) champ vecteur de *eb*
(b) champ physique

Selon la taille et la valeur d'un champ, Trois types de champs peuvent être définies:

a) Un champ constant :

Un champ constant est un champ dont la taille et la valeur sont bien définies.

$$\text{Taille}(C_i) = \text{Constante} \quad \text{et} \quad \text{Valeur}(C_i) = \text{Constante}$$

$$\exists\, C_i \in \{\mathcal{P} \text{ ou } \mathcal{M}\} \quad \text{tel que : } \text{Taille}(C_i) = \text{Constante et } \text{Valeur}(C_i) = \text{Constante}$$

Par exemple, les champ de synchronisation marquant le début et la fin d'une trame sont générés par le niveau physique et sont considérés comme étant des champs constants par le niveau Mac.

b) Un champ statique:

Un champ est dit statique si c'est un champ de taille constante et dont la valeur (le contenu) est variable. Dans ce cas, la taille du champ est invariable d'une trame à une autre.

$$\text{Taille}(C_i) = \text{Constante} \quad \text{et} \quad \text{Valeur}(C_i) = \text{Variable}$$

Un exemple de champ statique est le champ adresse de destination qui peut prendre plusieurs valeurs selon l'entité destinataire et dont la taille du champ est toujours constante.

c) Un champ variable:

Un champ est dit variable si sa taille est variable d'une trame à une autre et par conséquent sa valeur l'est aussi. La délimitation de ce champ est connue pendant le traitement de la trame. Le nombre de *eb* que peut constituer ce champ n'est donc pas connu pendant la spécification du protocole.

$$\text{Taille}(C_i) = \text{Variable} \quad \text{et} \quad \text{Valeur}(C_i) = \text{Variable}$$

Un exemple de champ variable est le champ représentant les données utilisateurs où la valeur et la taille du message ne sont généralement pas connues au préalable.

3.3.3 Séquencement des champs

Une trame est en réalité un ensemble de champs concaténés dans un ordre donné. Le séquencement des champs permet de définir l'ordre dans lequel les différents champs sont traités. Il est déterminé par le type de la trame à traiter, c'est-à-dire par le protocole.

Soit la trame suivante :

$$T = [\, C1 \mid C2 \mid C3 \mid .... \mid Cm \,]$$

A la déclaration des différentes trames du protocole, nous prenons pour hypothèse que tout champ C_i est spécifié avant la déclaration du champ qui le suit (C_{i+1}).

Deux types de trames peuvent être distingués:

- Trame "*statique* " et trame "*dynamique*" .

a) Trame *Statique*

Une trame est dite *statique* si la présence du champ suivant C_{i+1} dans une trame ne dépend pas des champs précédents (C_0, C_1, ..., C_i). Dans ce cas, une seule structure de trame est définie par le protocole de communication. Chaque champ de la trame apparaît obligatoirement dans l'émission ou la réception de la trame.

b) Trame *Dynamique*

Dans ce cas, il existe au moins un champ C_{i+1} dans la trame dont la présence dépend au moins d'un champ parmi les champs précédents (C_0, C_1, ..., C_i).

Dans le cas d'un protocole donné, on peut y extraire plusieurs trames qui partagent un ou plusieurs champs en commun.

Soit un ensemble de trames $\mathcal{T}$ et un ensemble de champs $\mathcal{C}$ tels que :

$$\mathcal{T} = \{ T1, T2, T3, T4 \}$$

et $$\mathcal{C} = \{ C1, C2, C3, C4, C5, C6, C7, C8, C9 \}$$

avec :

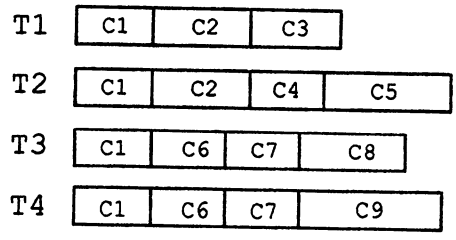


figure 9: les différents champs des trames Ti

La relation entre les différentes trames est représentée par le séquencement des champs suivants :

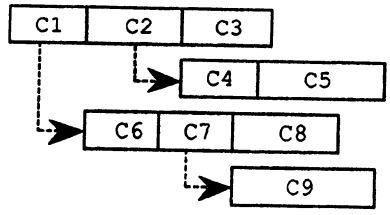


figure 10: relation entre les trames

La structure unifiée des différentes trames forme un arbre dont chaque noeud représente un champ et dont l'arc indique le champ suivant (arc de séquencement). La représentation sous forme d'un graphe est la suivante :

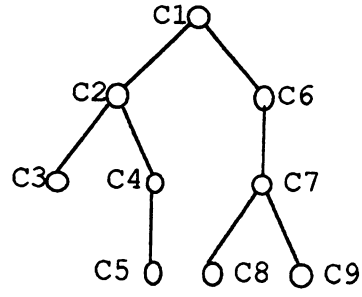


figure 11: représentation graphique (Arbre)

Dans cet exemple, trois conditions sont nécessaires pour reconnaître les champs suivants de C1, C2 et C7. Ces conditions sont positionnées selon les résultats des traitements associés à ces champs.

- **Traitement du Séquencement des Champs**

a) Cas d'une trame statique

Dans ce cas, le début du champ suivant dépend uniquement de la fin du champ courant. La taille du champ courant permet de déterminer cette fin dans le cas où cette taille est constante c'est-à-dire dans le cas d'un champ de type constant ou statique. Par contre, dans le cas d'un champ variable, une condition doit être satisfaite pour indiquer la fin du champ courant. Le test de cette condition est réalisé par un traitement particulier sur le champ qui doit être spécifié par le protocole.

b) Cas d'une trame dynamique

Dans le cas d'une trame dynamique, un champ peut aussi être de taille constante ou variable; ceci représente le même problème que dans le cas d'une trame statique.

Pour savoir quel est le champ suivant dans le cas où l'on a plusieurs champs qui dérivent à partir du champ courant, une ou plusieurs conditions sont associées à chaque dérivation dont le test permet la sélection du champ suivant.

3.4 Gestion Sémantique

La gestion sémantique d'une trame s'occupe essentiellement des différents traitements qui doivent être activés sur chaque champ de la trame.

3.4.1 Fonctions de Traitement d'un Champ

Ce sont les différents traitements à appliquer sur chaque champ de la trame que ce soit en émission ou en réception (Analyse ou Assemblage). Chaque champ

est associé à un ensemble de fonctions internes définies par le protocole. Celles-ci dépendent de la trame dans laquelle le champ est utilisé (type de trame).

Soit $\mathcal{F}$ = { F_1, F_2, ... } l'ensemble des fonctions associées aux champs de l'ensemble $\mathcal{C}$. Alors, pour chaque champ C_i de la trame T_j on a :

$T_j.C_i \Rightarrow \{F_k\}$ avec $T_j \in \mathcal{T}$, $C_i \in \mathcal{C}$ et $F_k \in \mathcal{F}$

Le signe "⇒" indique les différentes fonctions de traitement activées pour le champ associé.

Une même fonction F_k peut être appliquée à plusieurs champs C_i d'une même ou de différentes trames.

3.4.2 Caractéristiques d'une fonction de traitement

Une fonction de traitement d'un champ permet de réaliser un traitement spécifique pendant la durée du champ en question. La durée de traitement dépend de la taille du champ qui s'exprime donc en nombre d'unités de *eb* .

Dans le cas d'un champ constant ou statique, la durée de traitement du champ est connue à la déclaration du champ. Par contre dans le cas d'un champ variable, la durée de traitement dépend de la fin de traitement du champ signalée par un événement.

L'émission de données utilisateur à partir du niveau Mac est un exemple d'un traitement d'un champ de taille variable. Dans ce cas, la fin du champ de données est signalée par l'interface haute pour passer au champ suivant.

On distingue deux types de traitements : traitement en émission et traitement en réception.

3.4.1.1 Traitement en Emission de Champ

Le traitement en émission consiste soit :

- à générer un champ vers l'interface bas pendant l'émission,
- à réaliser des traitements spécifiques sur un champ à émettre,
- ou à réaliser des traitements spécifiques sur des paramètres internes ou externes.

Dans le cas du niveau Mac, le traitement qui consiste à sérialiser les différents champs de données reçus à travers l'interface haut est un traitement de génération de champs en série.

Un traitement spécifique interne (comme l'exemple du traitement CRC) est appliqué sur des champs en émission. Il permet aussi de générer un champ (champ CRC) à la fin de ce traitement.

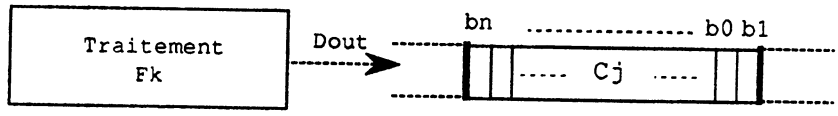


figure 12: traitement en émission

3.4.1.2 Traitement en Réception de Champ

Le traitement en réception consiste soit à :

. recevoir un champ de l'interface bas et de l'envoyer vers l'interface haut,
. réaliser des traitements spécifiques sur un champ en réception,
. réaliser des traitements spécifiques sur des paramètres internes ou externes.

Par exemple, un traitement consistant à envoyer les champs de données vers l'interface permet de produire un ou plusieurs champs désérialisés dans le cas du MAC. Le traitement CRC de décodage est une fonction interne qui réalise un traitement sur un ou plusieurs champs. La reconnaissance d'une adresse est aussi un traitement interne.

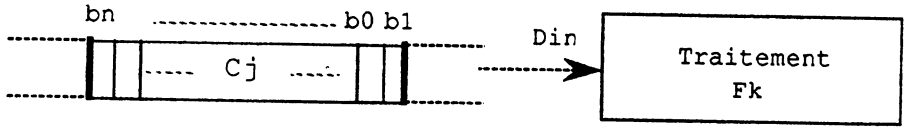


figure 13: traitement en réception

Que ce soit dans le cas d'un traitement d'émission ou de réception, une fonction de traitement est donc caractérisée par :

- son début d'activation,
- les résultats générés,
- ses paramètres
- et sa fin d'activation.

3.4.3 Activation d'une Fonction de Traitement

Le début de traitement d'un champ correspond à l'activation des fonctions associées. Ceci est réalisé par un événement d'activation du début de traitement généré pour le traitement du champ à partir du premier *eb* . Le traitement peut être initialisé avec des paramètres si nécessaire. Dans ce cas E est accompagné de paramètres de traitement.

$$Ti.Cj \Rightarrow \{ Fk \rightarrow E \}$$

Le signe "→" désigne l'événement d'activation.

3.4.4 Résultat de Traitement

Le compte rendu du traitement d'un champ est signalé par un événement résultat R à la fin du traitement qui correspond au dernier *eb* du champ. Plusieurs résultats R peuvent être fournis à la fin du traitement.

$$Ti.Cj \Rightarrow \{ Fk \rightarrow R \}$$

3.4.5 Traitement d'un Champ

Un champ Cj peut être éventuellement considéré comme un paramètre pour un traitement donné. Din et Dout sont les données qui représentent le vecteur de *eb* du champ Cj respectivement en réception ou en émission.

$$Ti.Cj \Rightarrow \{ Fk \rightarrow ([Din], [Dout]) \}$$

Dans le cas du protocole de niveau Mac, les traitements des champs sont réalisés en série.

3.4.6 Représentation d'un Traitement

Une fonction de traitement associée à un champ est exprimée par:

$$Ti.Cj \Rightarrow \{ Fk \rightarrow (E , R , [Din], [Dout]) \}$$

avec E un événement de validation de début de traitement (accompagné éventuellement avec des paramètres),

R le résultat du traitement qui peut regrouper plusieurs résultats,

et éventuellement Din ou Dout représentant le contenu du champ Cj.

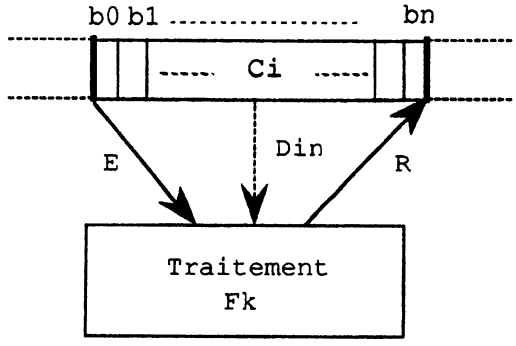


figure 14: traitement d'un champ en réception

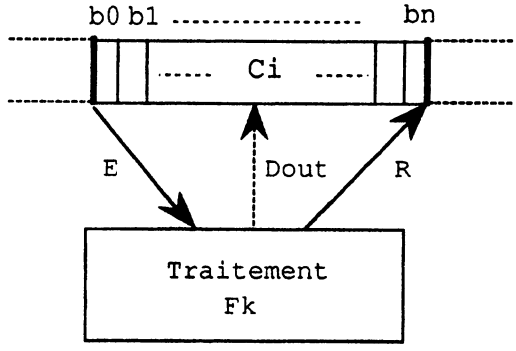


figure 15: traitement d'un champ en émission

3.4.7 Représentation Générale d'un Protocole de Communication

En conclusion, on peut dire qu'un protocole de communication est un ensemble de traitements associé à un ensemble de champs appartenant à un ensemble de trames.

Soit :

$\mathcal{T} = \{ T_1, T_2, ... \}$ un ensemble de trames ,

$\mathcal{C} = \{ C_1, C_2, ... \}$ un ensemble de champs ,

$\mathcal{F} = \{ F_1, F_2, ... \}$ un ensemble de fonctions de traitements.

Le protocole de communication définit alors une relation entre les différents ensembles par:

$$\text{Protocole} = \{ T_i.C_j \Rightarrow \{ F_k \rightarrow (E , R , [Din], [Dout]) \} \}$$

avec E un événement de validation de traitement,
R le résultat du traitement,
et éventuellement Din et Dout le contenu du champ Cj.

Cela signifie qu'un protocole définit un ensemble de trames et que dans chaque trame, les champs sont associés à un ensemble de traitements.

on a alors, la représentation suivante:

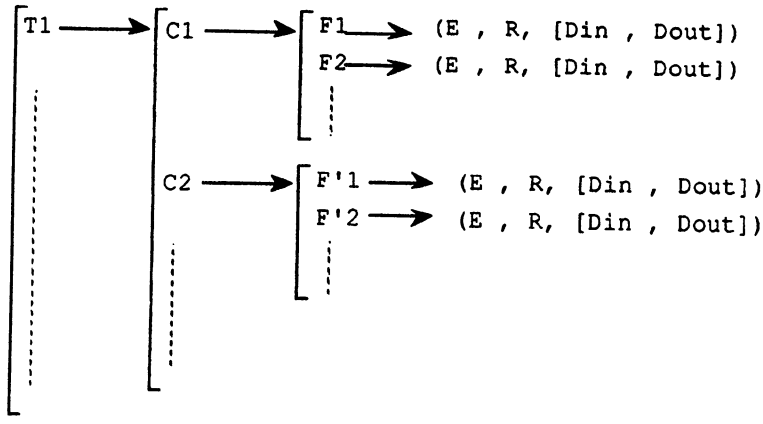


figure 16: un protocole de communication

3.5 Modèle de Représentation

3.5.1 Graphe de Représentation

La représentation interne d'un protocole de communication dépend de la représentation de :

$$Protocole = \{ Ti.Cj \Rightarrow \{ Fk \rightarrow (E , R , [Din], [Dout]) \} \}$$

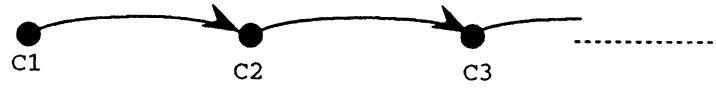
qui constitue la syntaxe et la sémantique des différentes trames du protocole.

3.5.1.1 La structure de la trame: la syntaxe

Chaque trame Ti est vue comme une succession de champs Cj :

$$Ti = [C1 \mid C2 \mid ...]$$

La séquence de champs est ordonnée:



Chaque trame Ti est alors représentée par un graphe où chaque noeud représente un champ Cj et chaque arc indique le champ suivant de Cj, c'est-à-dire Cj+1. Un protocole est représenté donc par un ensemble de graphes correspondant à chaque trame.

Le noeud représentant un champ est caractérisé par son *type* et sa *taille* . Il peut être de type *champ synchronisation (physique)* ou un *vecteur de eb* et de *taille constante ou variable.* Toutes les caractéristiques d'un noeud sont connues à la déclaration de la structure des différentes trames.

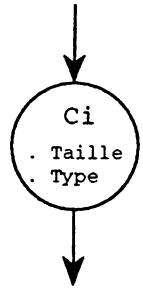


figure 17: représentation d'un champ Ti.Cj par un noeud

3.5.1.2 Traitement de la trame: La sémantique

Les traitements Fk associés à un champ donné Ci sont représentés par un ensemble de fonctions liées au noeud correspondant:

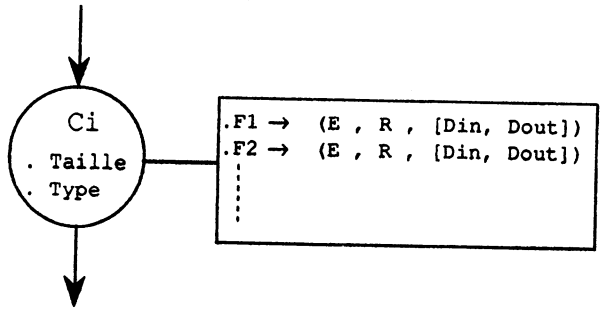


figure 18: Ti.Cj ⇒ { Fk → (E , R , [Din], [Dout]) }

Lorsque plusieurs trames ont un ou plusieurs champs équivalents dont la sémantique est la même, alors leur regroupement forme un arbre dont le sommet est un début de trame et les feuilles sont les différents champs de fin de trame.

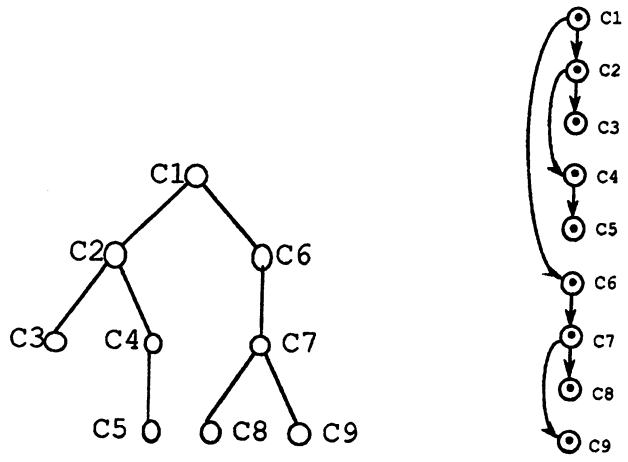


figure 19: graphe de structures des trames

Les parcours des noeuds des graphes à partir des différents sommets jusqu'aux feuilles de chaque graphe constituent l'ensemble des trames du protocole. Le noeud suivant est déterminé par un arc de séquencement des champs. Le choix du noeud approprié est déterminé par le résultat de traitement du champ courant. Pour cela, on a deux types d'arcs : arc sans condition et arc avec condition.

3.5.1.3 Caractéristiques d'un arc

Le séquencement des champs dans une trame peut être réalisé par deux types d'arcs:

a. Arc de séquencement sans condition:

Dans ce cas il n'y a aucune condition sur le séquencement des champs. Le champ suivant est défini par le protocole d'une façon statique.

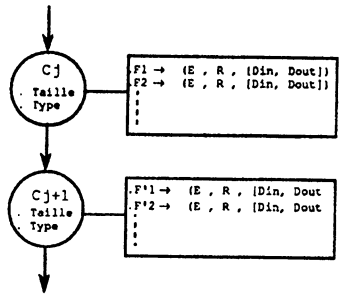


figure 20: arc de séquencement sans condition

b. Arc de séquencement avec condition:

Le champ suivant est déterminé par la satisfaction d'une ou plusieurs conditions associées à l'arc de séquencement. Une condition est un événement signalé par une fonction de traitement.

Généralement, on a trois types de conditions:

- Condition sur un champ dans la trame: dans ce cas, le champ suivant dépend de la valeur du champ courant.

- Condition sur le résultat d'un ou de plusieurs traitements: le champ suivant dans une trame peut être déterminé par un ou plusieurs événements correspondant à un résultat de traitement.

- Condition sur un événement d'interface: le champ suivant dans une trame peut être déterminé par un ou plusieurs événements correspondant à une signalisation de l'interface (haut ou bas).

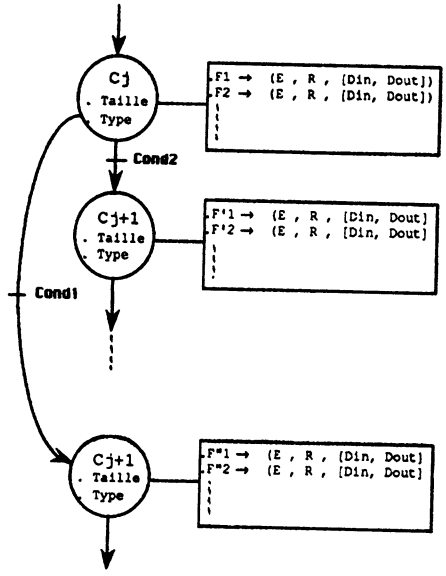


figure 21: arc de séquencement avec condition

3.5.2 Représentation Interne d'un protocole:

Un protocole de communication est donc représenté par deux graphes correspondant à un graphe d'émission et de réception. Ces deux graphes représentent donc les processus de traitement pour l'émission (Assemblage) et la réception (Analyse). La syntaxe et la sémantique sont inclues dans chaque graphe.

La représentation interne d'un tel graphe est réalisée sous forme d'un tableau à deux dimensions. Les lignes représentent les différents noeuds du graphe et les colonnes représentent les caractéristiques et les différents traitements ou fonctions qui sont associés à chaque noeud (figure 22). Le passage d'un noeud à un autre est indiqué par une colonne de *noeud suivant*. Si plusieurs noeuds sont les suivants d'un noeud donné, plusieurs colonnes de *noeuds suivants* sont ainsi créées.

Cette représentation interne donne une forme "*plane*" de la représentation graphique d'un protocole de communication.

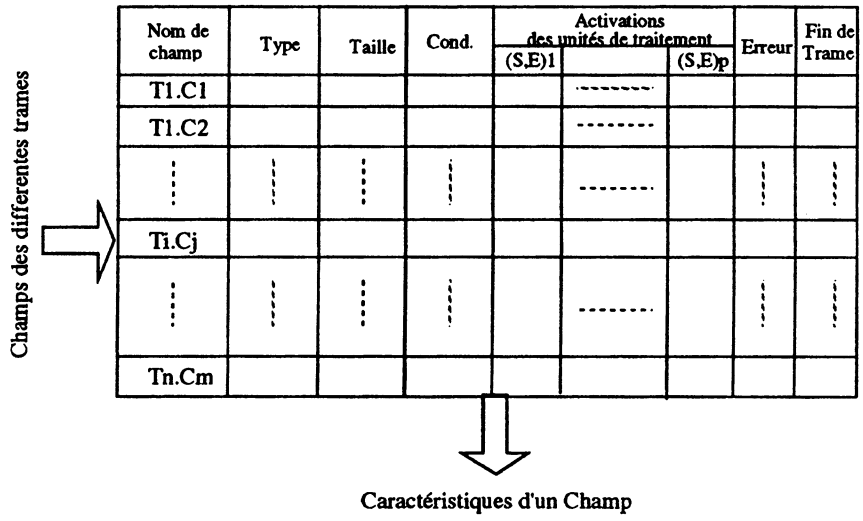


figure 22: représentation interne d'un graphe de protocole

3.5.3 Réponse dans la trame : Un cas particulier

Dans le cas des protocoles de niveau MAC, plus particulièrement des protocoles dits "bus de terrain", on peut trouver des cas particuliers où durant l'émission (ou la réception), on change de mode de transfert, éventuellement selon une condition, pour passer en réception (émission).

Par exemple, considérant deux entités A et B qui s'échangent des données à travers deux trames T1 et T2 de formats suivants:

$T1 = \{ C1, C2, C3 \}$ et $T2 = \{ C4, C5 \}$

A la fin d'émission de la trame T1 par l'entité A, cette dernière passe systématiquement en réception pour recevoir la trame T2 émise par l'entité B.
De même manière, mais un comportement inverse est réalisé par l'entité B.
Ce type de comportement est appelé en générale "la réponse dans la trame" (figure 23).

Le passage de l'émission en réception (et vice-versa) peut être vu comme un arc liant un état du processus d'émission à un état du processus de réception.

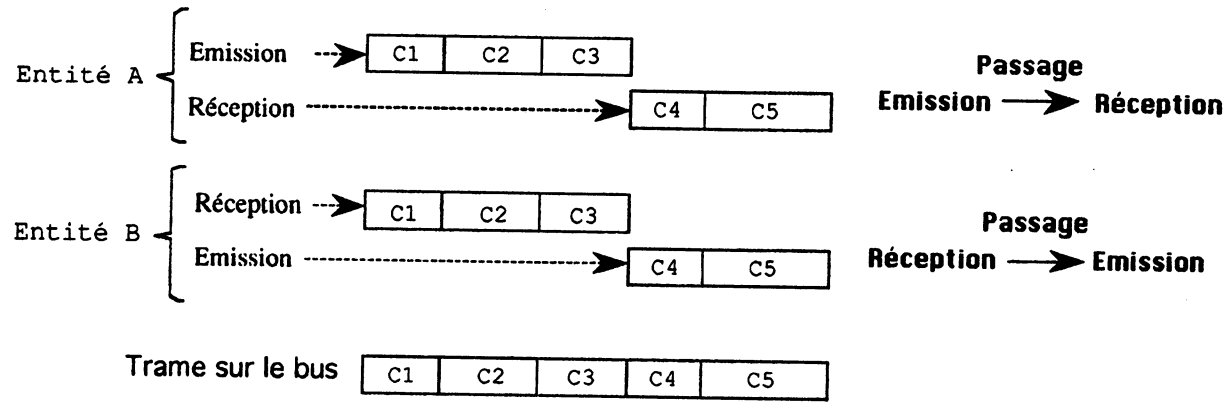


figure 23: réponse dans la trame

Pour le protocole de communication, il n'y a qu'une seule trame T' qui est vue sur la ligne, telle que:

$$T' = \{ C1, C2, C3, C4, C5 \}$$

Comme en émission ou en réception, le passage d'un mode de transfert à un autre (Emission vers la réception ou Réception vers Emission), est spécifié par un arc de passage en émission ou en réception. Ce passage peut être indiqué comme précédemment par un arc de condition sur un champ ou par un arc de condition sur un événement. Cet arc est échangé entre le processus d'émission et de réception. Il indiquera alors le champ à partir duquel le processus doit commencer son traitement.

Conclusion

Le protocole de communication est représenté finalement par deux principaux graphes :

a) Graphe représentant le processus d'émission
b) Graphe représentant le processus de réception

Ce sont des graphes d'états, où chaque état correspond à un champ d'une trame donnée. Le traitement de chaque champ est réalisé au niveau de chaque état.

Le passage d'un état à un autre correspond au passage d'un champ à un autre. La réponse dans la trame est le passage d'un graphe d'émission (réception) à un graphe de réception (émission).

Le but est d'utiliser le modèle de représentation pour :

. la spécification de protocoles
. la synthèse d'architecture : le modèle est une représentation intermédiaire d'implémentation.

On remarque que, aucune hypothèse n'a été émise sur la façon d'implémenter le protocole car, jusqu'ici on a fait abstraction des architectures d'implémentation (que ce soit logicielle ou matérielle).

Chapitre 2

Modèle d'Architecture Pour les Circuits de Communication

Chapitre 2

Modele d'Architecture Pour les Circuits de Communication

Introduction

Dans le présent chapitre, nous nous intéressons aux différentes fonctionnalités des circuits de communication et à leurs implémentations matérielles. L'analyse des caractéristiques de notre domaine d'application, qui est celui des couches basses d'un réseau local de contrôle-commande, nous permet de préciser les différentes fonctions que le circuit doit réaliser. Cette analyse a été faite dans notre équipe parallèlement avec l'analyse des différents types de circuits de communication qui implantent ces protocoles de contrôle-commande. Ces analyses ont été faites dans le cadre de la thèse de Mr Imad Sabouni [Sab92]. Elles ne seront rappelées que succinctement dans ce chapitre.

A partir de la représentation graphique d'un protocole de communication, un modèle d'architecture d'implémentation de protocole sera présenté. Ceci permettra d'implanter les protocoles de communication à partir de leurs descriptions comportementales. Un autre paramètre qui est pris en compte est celui de l'interfaçage du circuit avec l'extérieur que ce soit du côté de l'interface haute (station) ou du côté de l'interface basse (ligne).

1 La Conception des Circuits de Communication

Un circuit de communication est un circuit destiné à implanter en matériel un protocole de communication en intégrant ses différentes fonctionnalités. Dans ce qui suit, on présentera quelques circuits de communication, appelés souvent "contrôleurs de communication", en décrivant leur architecture interne et en dégageant quelques caractéristiques communes.

1.1 Quelques Circuits de Communication

Nous présentons rapidement les circuits implantant le protocole VAN, le FIP et le CAN, sachant que le lecteur intéressé pourra se repporter à [Sab92] pour avoir une analyse plus complète.

1.1.1 Les Circuits VAN

Ce sont des circuits qui sont destinés à effectuer des échanges de données dans un réseau intra-véhicule. Plusieurs circuits implantant le VAN ont été définis parallèlement avec les efforts de normalisation du protocole [VAN92].

1.1.2.1 Le circuit VAN-RCP

Le circuit VAN-RCP (Remote Controlled Process) est réalisé par la société RNUR en collaboration avec PHILIPS dans le cadre du projet PROMETHEUS [RCP91]. Il est destiné à implanter la partie contrôle de communication d'une station autonome. La figure 1 donne le schéma bloc du circuit.

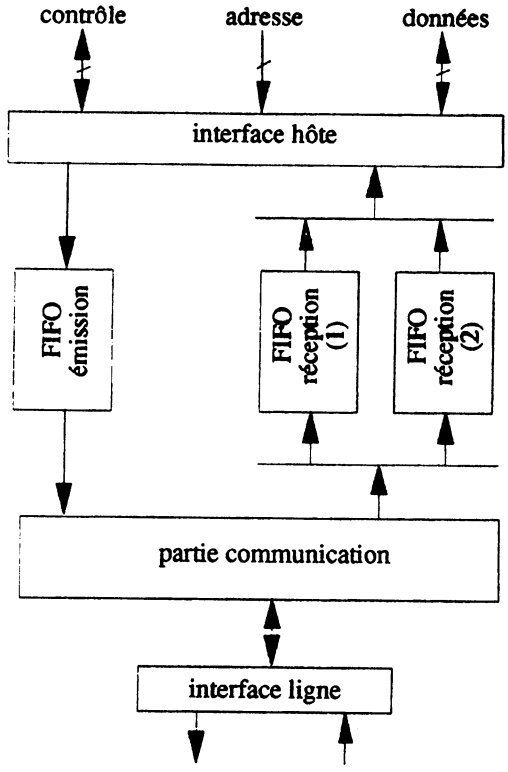


figure 1 : schéma général du circuit VAN-RCP

Il s'interface, côté station, avec un microprocesseur ou un micro-contrôleur (famille 8051 de INTEL). Il est vu comme un périphérique classique de type UART (Universal Asynchronous Receiver and Transmitter).

Le circuit est composé de trois parties : la partie interface ligne, la partie gestion de la communication et la partie interface avec le hôte.

L' interface ligne permet de connecter le circuit à un émetteur/récepteur de ligne, de réaliser le codage/décodage des bits et de résoudre les problèmes de synchronisation et d'échantillonnage.

La partie communication gère l'accès au médium et l'arbitrage, le filtrage d'adresse, le calcul de CRC en réception et en émission, la gestion d'acquittement et la gestion d'erreurs. Les échanges de données entre le processeur hôte et le circuit sont réalisées à travers trois FIFOs dont une qui est en émission et deux qui sont en réception. Chaque FIFO a une capacité de 10 octets permettant de recevoir 8 octets maximum de message, de l'identificateur et du champ COM de la trame VAN.

L'interface avec l'hôte est réalisé par un bus de 8 lignes de données, 4 lignes d'adresse et plusieurs lignes de contrôle. Les 4 lignes d'adresse permettent de sélectionner un registre interne parmi 10. Ces registres contiennent des paramètres de contrôle et des états de communication tels que : 2 registres pour mémoriser l'adresse de la station, 2 registres pour le contrôle et un pour l'état, un registre de contrôle d'interruption, ...

Pendant la réception, le circuit effectue la vérification de format et de CRC. Il positionne le registre d'état selon la validité du message reçu et lance ensuite une demande d'interruption vers l'hôte. Celui-ci vérifie l'état de la réception pour ensuite lire le message par le biais du registre de lecture de FIFO de réception.

Pour réaliser une émission d'un message, l'hôte écrit les données dans le registre d'écriture du circuit et positionne le registre de contrôle en émission. Après émission du message, l'hôte vérifie si la communication s'est bien déroulée.

1.1.2.2 Le circuit VAN-DLC

Le circuit VAN-DLC (*Data-Link Control*) est un prototype conçu par la société MHS en collaboration avec PSA/RNUR [DLC92]. Il doit implémenter le protocole VAN et supporter les communicateurs autonomes et esclaves. Le schéma bloc du circuit est donné dans la figure 2.

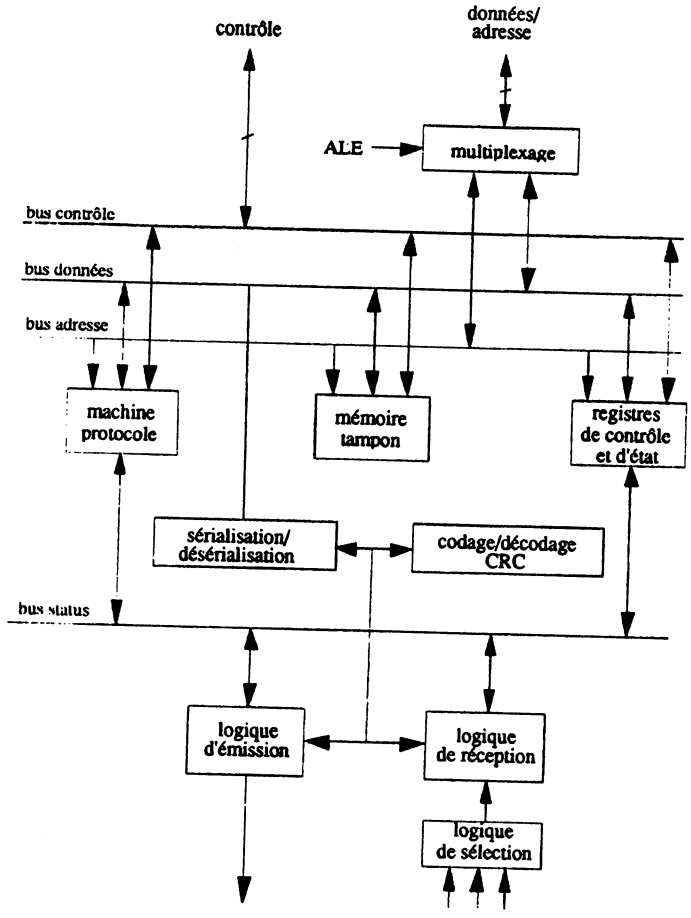


figure 2 : schéma général du circuit VAN-DLC

Le circuit doit s'interfacer avec plusieurs types de microprocesseurs par l'intermédiaire d'une mémoire partagée. L'accès à cette mémoire s'effectue en mode DMA (direct). Le bus d'adresse et le bus de données sont multiplexés et les signaux de lecture et d'écriture sont séparés, ce qui favorise beaucoup l'utilisation de circuits INTEL en particulier les contrôleurs de la famille 8051.

Un signal en émission et trois signaux en réception sont utilisés pour communiquer avec le bus VAN. Les trois signaux de réception sont utilisés pour comparer les différentes valeurs reçues et d'en décider sur le choix d'une.

Les procédures d'émission et de réception et de gestion du médium sont exécutées par deux blocs logiques. Ils contrôlent aussi une unité de codage et de décodage CRC et un bloc de sérialisation et de désérialisation des données.

Trois espaces d'adressage sont utilisés par le circuit:

- 12 registres d'interruption, de commande, de contrôle et d'état,

- 14 registres d'identificateurs de 8 octets chacun où sont mémorisés essentiellement un identificateur (12 bits), le champ commande (4bits), un masque de comparaison et adresse de début du message associé.

- 128 octets constituant la mémoire tampon où sont stockés les messages.

Pour envoyer un message, le processeur hôte doit programmer auparavant un idenficateur, écrit le message dans le tampon correspondant et positionne le registre de commande. Le circuit commence alors à envoyer le message d'une façon autonome.

Lors de la réception d'une trame, le circuit reçoit l'identificateur et le compare à l'ensemble des identificateurs du circuit. Dans le cas où un identificateur est reconnu, le message est écrit dans la mémoire tampon et les registres d'état et d'interruption sont positionnés pour signaler au processeur hôte la réception d'une trame.

Dans le cas d'une réponse dans la trame, lorsque le circuit reçoit une trame de question, il ne décide d'envoyer le message qui se trouve dans le tampon correspondant à l'identificateur reçu que si les données du message sont à jour. Dans le cas contraire, la réponse est différée dans une autre trame.

1.1.2.3 Le Circuit FVC

Le circuit FVC (*Full-VAN Controller*) est un contrôleur microprogrammable qui est conçu pour implémenter le standard VAN par la société TEXAS INSTRUMENT [FVC92]. Il permet d'intégrer dans le circuit une partie ou la totalité du logiciel application, ceci afin de diminuer la charge du processeur hôte. Le schéma bloc du circuit est présenté à la figure 3.

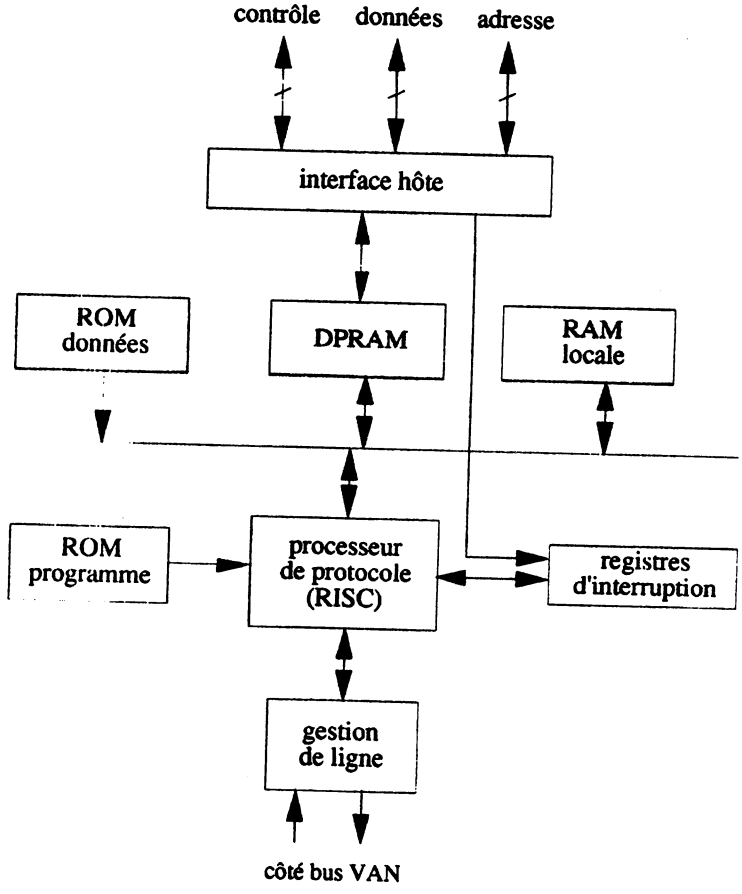


figure 3: schéma général du circuit FVC

Le corps du circuit permet d'implanter le protocole MAC et le niveau LLC en logiciel par le processeur de traitement de protocole RP2 (*RISC Protocol Processor*) qui possède un jeu d'instructions optimisé spécialement pour le domaine de communication. Le processeur RP2 a le bus programme et le bus données qui sont séparés et une architecture RISC pipelinée. Il possède un espace mémoire qui se divise en deux: espace de programmes et espace de données. Le premier est implémenté par une ROM qui contient le microcode à exécuter par le microprocesseur, et le second est organisé en une RAM locale et une ROM de données uniquement accessibles par le RP2, d'une pseudo DPRAM de communication qui est partagée entre le RP2 et le processeur hôte

et du registre de contrôle d'interruption et de TAS (*Test and Set*) pour la protection d'accès à la DPRAM.

Du côté du bus VAN, un module de gestion de ligne réalise la synchronisation des bits, le codage/décodage, la sérialisation/désérialisation et la gestion des collisions.

Du côté du processeur hôte, une interface est utilisée pour dialoguer avec le RP2. Elle comporte 8 lignes de données, 9 lignes d'adresse et un ensemble de lignes de contrôle telles que: ligne de lecture/écriture, lignes de sélection et d'interruption, ...

1.1.2 Le Circuit Full-FIP

La société CEGELEC RED a réalisé le circuit Full-FIP qui est un contrôleur de communication et implémentant le niveau liaison de données du standard FIP [CEG89]. Quelques aspects du protocole de niveau application tels que les mécanismes de rafraîchissement et de synchronisation peuvent être également pris en compte.

Le circuit possède un bus privé sur lequel est disposée une mémoire externe au circuit dans laquelle seront stockés les objets de communication (mémoire de données) ainsi que le programme à exécuter (mémoire de programmes). Il dispose également d'une interface avec le bus système à travers laquelle il communique avec le processeur hôte. Le schéma de l'environnement d'utilisation du circuit Full-FIP est présenté dans la figure 4 et le schéma bloc du circuit est donné dans la figure 5.

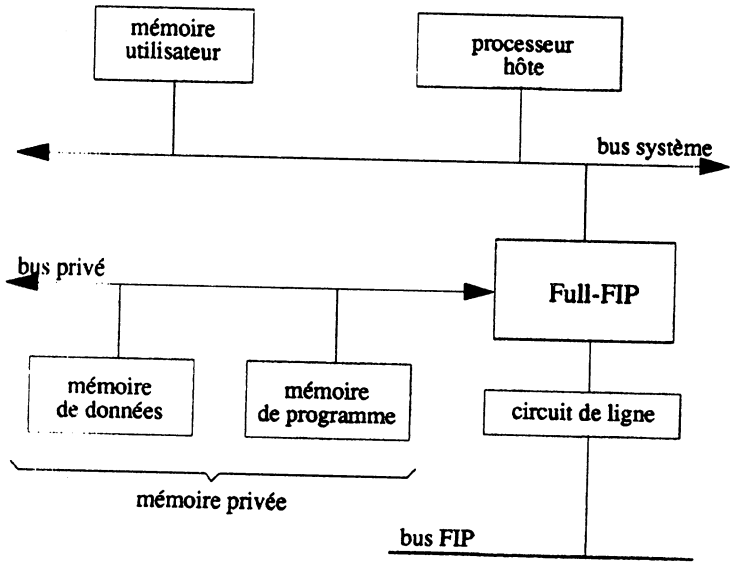


figure 4: schéma d'utilisation du circuit Full-FIP

Un processeur associé à une mémoire locale est intégré dans le circuit et permet de traiter le protocole de niveau LLC et une partie du protocole d'application. Le bus mémoire privée est composé de 16 lignes de données/adresses multiplexées et 4 lignes d'adresses supplémentaires. L'interface hôte qui permet les échanges de données avec le processeur hôte est réalisée en utilisant un protocole d'échange asynchrone de type 68000.

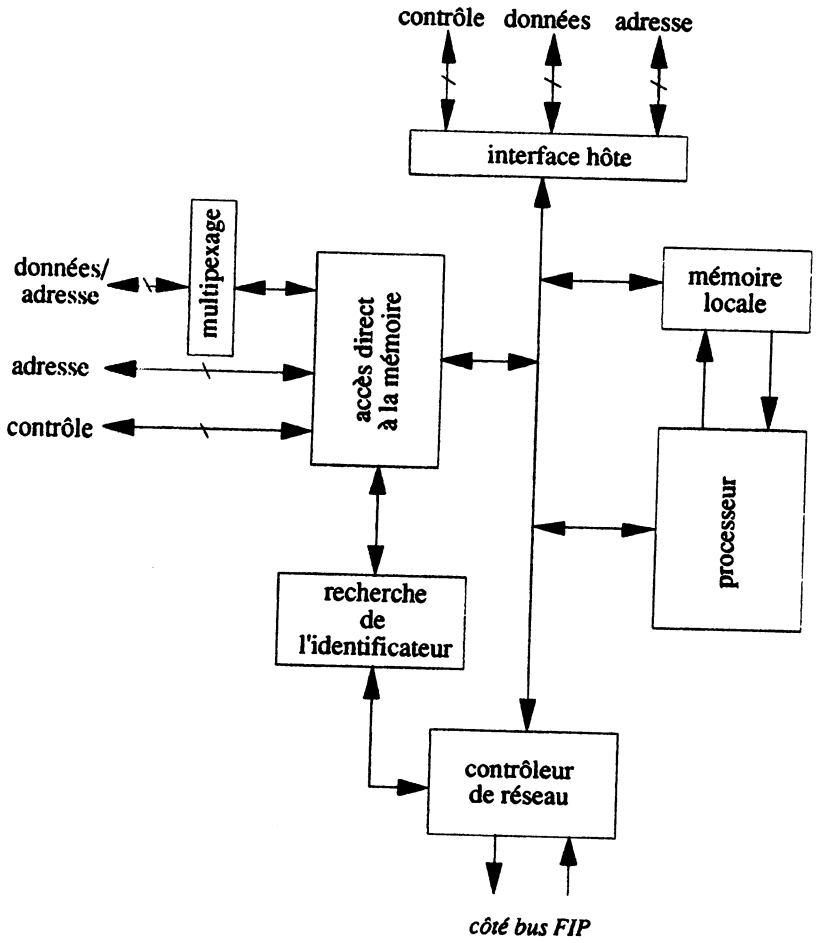


figure 5: schéma bloc du circuit Full-FIP

1.1.3 Le Circuit INTEL 82526 (CAN)

Le circuit 82526 est un contrôleur de communication VLSI qui permet de gérer toutes les fonctionnalités définies par le protocole intra-véhicule CAN [INT91]. Il a été développé pour implanter les fonctions de niveau physique et de niveau liaison de données du protocole. Le schéma-bloc global du circuit est présenté dans la figure 6.

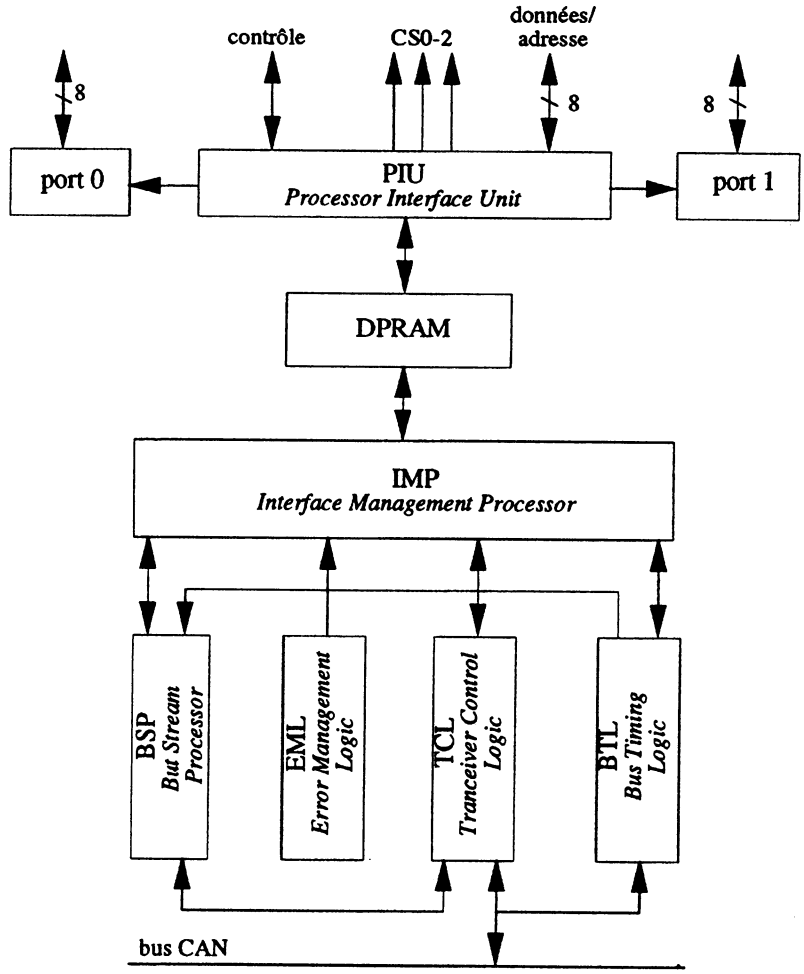


figure 6: schéma général du circuit INTEL 82526

Le circuit s'interface avec un microprocesseur ou un microcontrôleur (de la famille INTEL 8051 [INT90]) par l'intermédiaire d'une mémoire à double-port (DPRAM) dans laquelle peuvent être mémorisées les données et les commandes.

Deux processeurs permettent de gérer chacun une partie du circuit. Le premier est le BSP (*Bit Stream Processor*) qui analyse les messages reçus et forme les messages à émettre. Il permet aussi de commander deux blocs : le TCL (*Transceiver Control Logic*) qui s'occupe du contrôle de l'émetteur/récepteur de ligne (en particulier, l'arbitrage bit-à-bit, le calcul et la détection du CRC et la réalisation de l''insertion/extraction des bits), et le EML (Error Managment Logic) qui s'occupe de la gestion d'erreurs. Le second processeur appelé IMP (*Interface Management Processor*) exécute les différentes commandes envoyées par le hôte. Il consiste à gérer les registres d'état et de commande, et de contrôler les échanges d'informations avec le bus série. La communication entre l'IMP et le processeur hôte est réalisée à travers une mémoire à double-port.

L'accès à la mémoire par le processeur hôte se fait par l'intermédiaire d'une interface appelée PIU (*Processor Interface Unit*). Elle permet de gérer le bus multiplexé données/adresse de 8bits, et les lignes de contrôle (lecture/écriture, interruption et sélection du circuit). Le PIU commande aussi deux ports d'E/S de 8 bits chacun.

1.1.4 Circuits pour les Réseaux Locaux de Type Ethernet

Ce sont des circuits usuels qu'on retrouve dans le commerce et qui implémentent le protocole CSMA/CD (*Collision-Sens, Multiple-Access with Collision Detection*) des réseaux Ethernet. Le schéma-bloc et les caractéristiques correspondantes peuvent être trouvés dans [Bye87].
Parmis ces circuits on a:

. le Coprocesseur 82586 de Intel
. le Contrôleur Am7990 de AMD
. le Contrôleur 8003 de S T (Seeq Technology
. le Contrôleur 82588 de Intel
. le Contrôleur 82C551 de Chips and Technologies
. le Contrôleur COM 9026 de SMC (Standard Microsystems Corp.)
. le Processeur de Communication TMS38010

Les différentes interfaces physiques associées à chaque type de circuit sont aussi présentées. Une étude de ces différents circuits nous a permi de tirer un certain nombre de caractéristiques communes qu'on retrouve souvent dans l'implémentation d'un circuit de communication.

1.2 Caractéristiques Communes des Circuits de Communication

On s'intéresse ici aux différentes caractéristiques architecturales communes des circuits de communication (appelés souvent contrôleurs de communication). L'analyse des différentes architectures des circuits en général (microprocesseur, coprocesseurs, ...), des circuits spécifiques (interfaces spécialisées, processeurs de traitement d'image, de traitement du signal, ...) ou des circuits de communication en particulier, nous permet de distinguer dans les architectures deux parties : une partie traitement interne et une partie interfaçage avec l'environnement externe.

Dans la partie traitement interne, certains circuits sont très orientés flot de données tels que les composants programmables et par contre d'autres non. Les circuits de communication qui sont destinés à implanter des procédures

distribuées, sont plutôt à contrôle dominant. L'architecture générale se compose de plusieurs modules asynchrones dont la partie opérative n'est pas importante. Il n'y a pas d'opérations arithméthiques complexes nécéssaires tels que les opérateurs multiplieurs. Des registres, des comparateurs, des compteurs et des multiplexeurs sont généralement les opérateurs les plus souvent utilisés.

Dans la partie interface avec l'extérieur, les circuits de communication sont composés de deux interfaces indépendantes : interface avec le processeur hôte et interface avec le bus du réseau.

Un découpage architectural d'un circuit de communication dédié à traiter les protocoles de niveau bas, plus particulièrement les protocoles de contrôle-commande est présenté dans la figure 7.

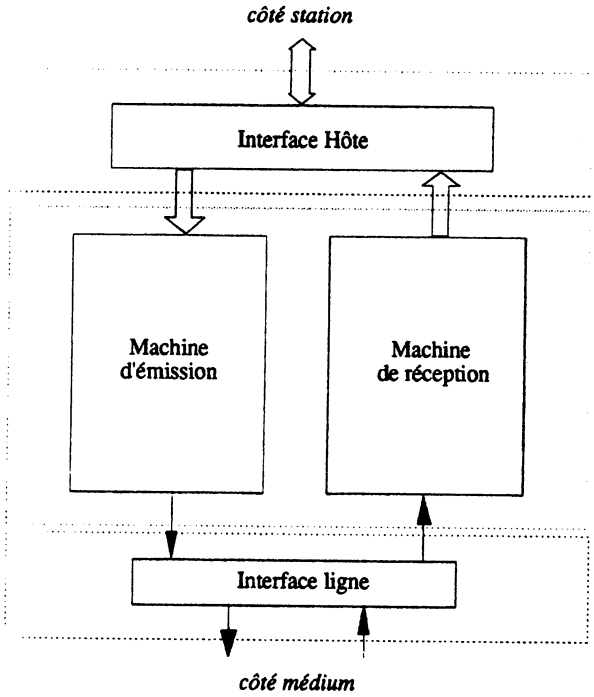


figure 7: découpage architectural d'un circuit de communication

2 Architecture Fonctionnelle du Circuit

2.1 Vue Fonctionnelle d'un Circuit de Communication

Fonctionnellement, un circuit de communication est une implémentation matérielle permettant d'effectuer des transferts d'informations de données et de contrôle entre le réseau (médium) et

l'application (station). Le transfert d'information est réalisé dans les deux sens : de la station vers le médium ou vice-versa.

Cependant, deux vues différentes du circuit de communication peuvent être observées selon qu'on le regarde du point de vue traitement (vue verticale) ou flux d'information (vue horizontale).

2.2 Vue Horizontale du Circuit

Deux flux d'informations sont traités durant une communication : un flux d'émission et un flux de réception. Le flux d'émission représente l'envoi des données reçues du côté de la station (hôte) vers le côté réseau (médium), par contre le flux de réception représente la réception des donnée du côté réseau et leur envoi vers la station.

Vus de haut, ces deux flux ne sont pas tout à fait symétriques car le moment d'envoi de données vers le médium est une décision qui appartient au circuit après celle de la station, par contre l'arrivée des données du côté du médium est complètement imprévisible. Pour cela, le flot de réception est plus prioritaire sur celui de l'émission [Dan88].

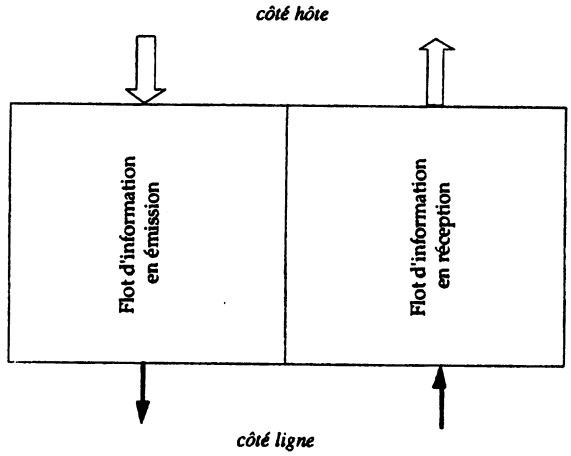


figure 8: flux de données dans un circuit de communication

2.3 Vue Verticale du Circuit

Selon les différents traitements internes dans un circuit de communication, on peut distinguer trois parties différentes (figure 9):

a) interface côté station :

Elle permet au circuit de dialoguer avec la station hôte via un bus externe. Le type d'échange avec le bus est un échange d'information de données, de contrôle et d'états.

b) interface Côté ligne :

Cette partie est chargée de dialoguer avec la ligne de transmission. Elle permet la connexion du circuit au médium de communication par l'intermédiaire d'un émetteur/récepteur de ligne.

c) cœur du circuit :

C'est la partie implémentant le traitement effectif du protocole MAC. Elle se décompose en deux machines de traitement: la machine d'émission et la machine de réception.

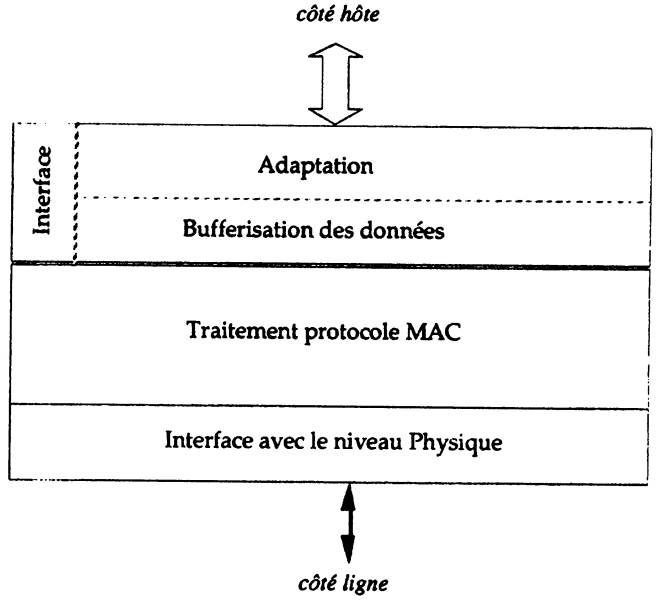


figure 9 : découpage fonctionnel d'un circuit de communication

3 Modèles d'Interfaces Côté Station

La partie interface avec la station regroupe une partie mémorisation de données et une partie adaptation des signaux. La station hôte peut posséder plusieurs degrés d'"intelligence" selon qu'elle soit une station câblée, un capteur (ou un actionneur) de deuxième génération ou un microprocesseur.

La complexité de cette interface station varie selon le degré d'intelligence de la station. Une nouvelle interface station doit être conçue à chaque fois qu'on change de type de station. Elle permet par conséquent d'adapter les signaux externes de la partie traitement protocole MAC avec l'extérieur sans avoir à les modifier.

Plusieurs architectures d'implémentation de l'interface avec la station hôte peuvent être considérées [Sab92] [Dan88] :

par l'intermédiaire

- d'un ensemble de registres (registres de contrôle, registres d'état, registres de données, registres généraux),
- d'une File d'Attente de type FIFO,
- d'une Mémoire partagée (à base de structure de liste plus ou moins simplifiées), simple ou à double accès,
- d'une interface "hybride" qui mélange les différents types d'architectures cités ci-dessus.

Ainsi, le choix d'une interface de transfert d'information côté station dans le circuit de communication dépend essentiellement de la complexité de la station hôte et des caractéristiques du réseau.

4 Modèles d'Architectures

Afin d'implémenter le protocole de communication, c'est-à-dire les machines d'émission et de réception, plusieurs modèles d'architectures peuvent être utilisés. On distingue essentiellement deux types d'architectures : architecture microprogrammée, architecture à chemin de données.

4.1 Modèle d'Architecture Microprogrammée

Une architecture à base d'un microprocesseur est une solution permettant d'implémenter les protocoles de communication. L'architecture globale est composée d'un ensemble d'éléments de mémorisation (RAM et ROM, EPROM, ...) pour stocker les données et les programmes, des éléments d'interface permettant au microprocesseur de dialoguer d'une part avec

l'extérieur du côté station et d'autre part avec le réseau et bien sûr d'un microprocesseur.

Ainsi, on a alors un bus interne dans l'architecture, qui est le bus transfert du microprocesseur, permettant de relier les interfaces et les mémoires au microprocesseur. L'accès au bus transfert de l'extérieur par le hôte doit être résolu de telle façon à ne pas créer des conflits d'accès aux informations de données, de contrôle et d'état. Une mémoire à double ports est une solution qui est généralement bien utilisée pour réaliser des lectures/écritures simultanées. Le schéma bloc d'une telle architecture est représenté par la figure 10.

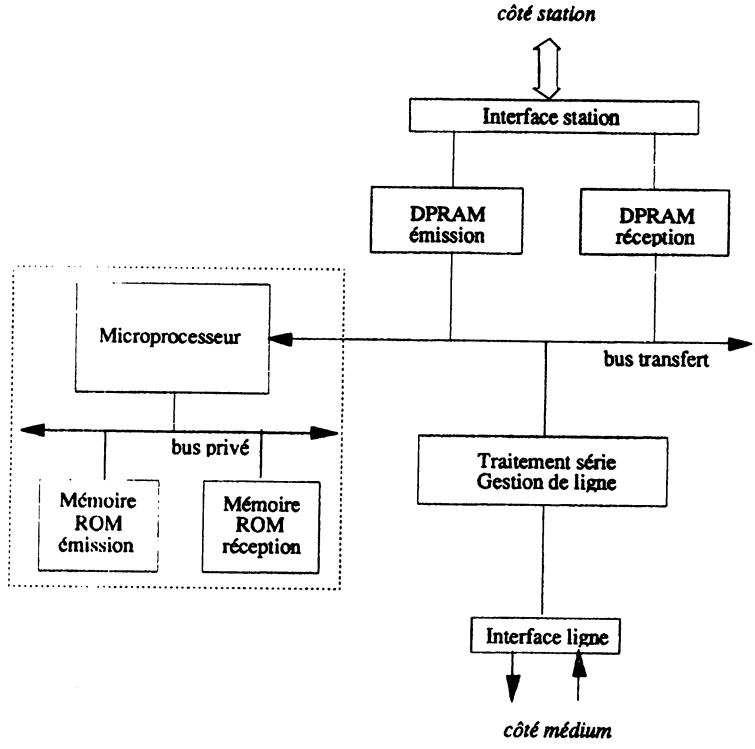


figure 10: modèle d'architecture microprogrammée

Dans ce type de modèle d'architecture, si l'on change de protocole, deux parties changent : la partie interface qui regroupe le bloc gestion de ligne avec l'interface ligne, et la partie programme des procédures d'émission et de réception qui résident dans les mémoires ROM.

L'avantage d'une telle architecture réside dans la connaissance du microprocesseur et de son environnement d'implémentation qui rend la conception et le temps de développement très rapide. Cependant, un paramètre non négligeable, pour ce qui est du moins des protocoles de communication, est le temps d'exécution des procédures de traitement

d'émission et de réception. Ce temps est très lié avec le temps de cycle du microprocesseur. Des instructions de chargement, de test et de rangement sont souvent employées et sont donc étroitement liées avec le temps d'accès à la mémoire.

Un processeur dédié à la communication avec un jeu d'instructions réduit est considéré comme une solution à ce problème. Cependant, le choix du jeu d'instruction reste dépendant de l'application ou d'une classe d'applications dans le domaine de la communication.

4.2 Modèle d'Architecture à Chemin de Données

Un modèle d'architecture à chemin de données est composé d'une partie contrôle et d'une partie opérative. Ce type d'architecture est largement étudié surtout en ce qui concerne la synthèse d'architecture pour des applications spécifiques. Un graphe de flux de données du problème doit être fourni afin de réaliser les différentes étapes de conception de la partie contrôle et de la partie opérative de l'architecture. Des optimisations architecturales et logiques sont appliquées sur ce graphe afin de répondre aux contraintes des spécifications du cahier des charges.

L'architecture globale du circuit de communication est composée donc d'un module de contrôle central et d'un ensemble d'opérateurs de chemin de données. Parmi cet ensemble d'opérateurs, des modules tels que la mémoire de données, un ensemble de registres et de logiques aléatoires sont utilisés. Les interfaces sont donc implémentées par des opérateurs et gérées par la partie contrôle.

Les différents outils de synthèse d'architecture sont d'un grand avantage pour ce modèle d'architecture. Cependant, il existe différents points sur lesquels jusqu'à présent il n'y a pas eu vraiment des solutions satisfaisantes, par exemple: tenir compte des contraintes des interfaces, de l'utilisation des bancs de mémoires comme des RAM ou des ROM et utilisation de modules complexes comme opérateurs. Le paramétre temps d'exécution des procédures d'émission et de réception, d'une telle partie contrôle, risque de ne pas répondre aux contraintes imposées par les spécifications dans certaines applications telles que les applications temps réel critiques.

La décomposition de la partie contrôle en deux pour implémenter séparément les procédures d'émission et de réception permet de réduire la taille de chaque partie contrôle et par conséquent le temps de cycle des parties contrôles. Le schéma bloc du circuit est présenté par la figure 11.

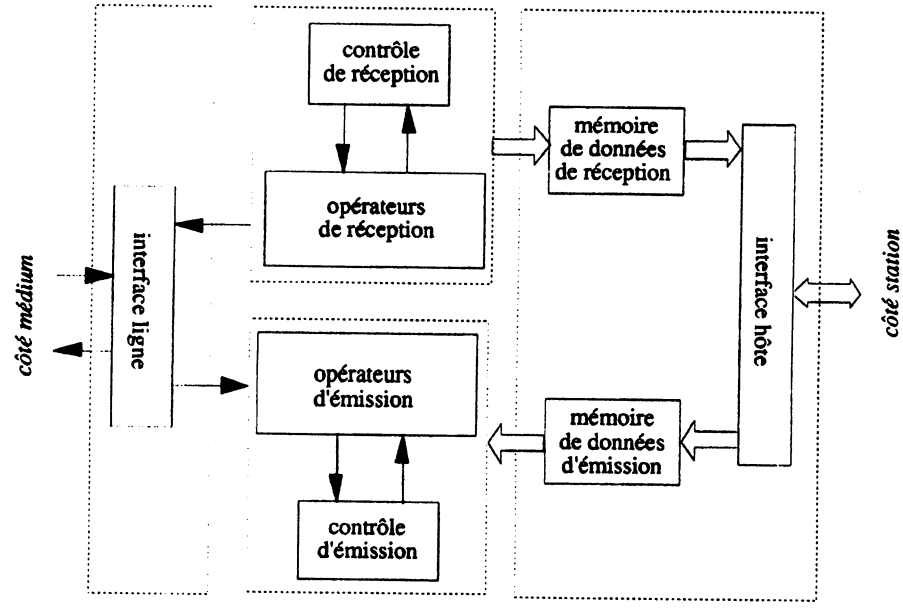


figure 11: modèle d'architecture à chemin de données

Les machines d'émission et de réception peuvent être générées d'une façon automatique à partir de la description de la représentation interne du protocole. Il faut pour cela décrire dans un langage de haut niveau le séquencement des champs comme étant des états de contrôle et les différents traitements par des opérateurs du langage de description du système de synthèse d'architecture. Le passage de la table de description de protocole à sa description dans le langage cible peut être réalisé d'une manière systématique en précisant les opérations qui correspondent aux différentes fonctions de traitement. L'architecture du circuit est obtenue à partir de cette description par le système de synthèse d'architecture.

4.3 Choix d'un Modèle d'Architecture

Avant d'aborder le choix du modèle d'architecture, un élément de réflexion peut être fait sur le protocole de communication et sur son implantation. On constate bien que dans beaucoup de cas, il y a une différence de "frontière" entre la spécification du protocole et son implantation. Ici, la frontière représente les délimitations architecturales de la spécification et de l'implantation. Dans le cas d'une spécification, les traitements sont bien regroupés et très discernables de l'application ou des autres protocoles adjacents. Par contre, dans le cas de l'implantation, les différents traitements d'un protocole donné sont généralement répartis dans l'architecture de façon à satisfaire certaines contraintes d'implantation.

Ainsi, dans le cas des traitements temps réel de niveau *eb*, un certain nombre de traitements peuvent être réalisés lorsqu'il s'agit d'un traitement de durée d'un élément binaire. Dans le cas contraire, une mémorisation serait nécessaire pour réaliser le traitement d'une façon "différée" dans le temps. Une manière d'effectuer le CRC par exemple sur des données d'une trame est d'attendre la fin de la réception de toute la trame pour commencer sa vérification.

Alors, on peut définir trois niveaux d'implémentation dans une application selon la complexité des traitements à réaliser:

· niveau temps réel : dans ce cas, les données de la trame sont traitées au fur et à mesure de leurs arrivées (et de leurs envois);

· niveau temps réel différé : le traitement exige dans ce cas une mémorisation des données de la trame pour qu'il puisse être exécuté par le circuit de communication;

· niveau différé : la donnée peut être, dans ce cas, traitée en différé dans la station hôte.

L'exemple d'un traitement CRC ou de reconnaissance d'adresse nous permet d'éliminer le modèle d'architecture microprogrammée car dans ce cas il faut recevoir toute la trame pour réaliser ces traitements et que dans notre domaine d'application, le temps réel est un facteur prépondérant.

La réponse dans la trame nécessite une réaction critique du circuit de niveau élément binaire à laquelle une architecture à chemin de données ne puisse pas dans certains cas répondre aux exigences en terme de temps de réponse de la partie contrôle du circuit.

Une architecture à chemin de données dédiée aux traitements en temps réel pourra permettre de répondre à ces exigences et d'implanter les différents points critiques du protocole de communication.

Un tel modèle d'architecture (dédié au domaine d'application) permet de fournir les différentes possibilités d'implantation des traitements nécessaires afin de concevoir une architecture qui répondra au mieux aux contraintes exigées. D'autre part, ce modèle servira de cible pour l'implantation automatique de protocoles à partir des spécifications exécutables décrites avec les outils du chapitre 4.

Dans le cas de l'implémentation des protocoles de communication, l'architecture est décomposée en trois parties : partie centrale (ou cœur du circuit) qui s'occupe du traitement proprement dit du protocole, partie interface avec le hôte qui consiste à mémoriser les données échangées entre le coeur du circuit et l'extérieur et d'adapter les signaux internes avec les signaux externes, et enfin une partie interface avec le médium qui sert à gérer la ligne. Le schéma bloc du circuit est présenté dans la figure 12 [Sah92].

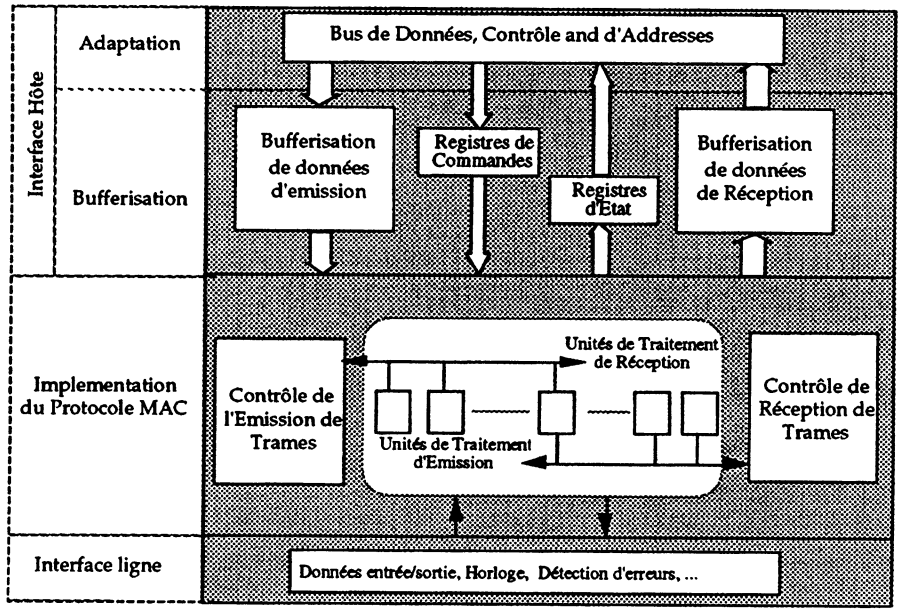


figure 12: modèle d'architecture dédiée

Les procédures d'émission et de réception sont implémentées par des contrôleurs de traitement syntaxique des trames et les différentes fonctions de traitements par des unités de traitement. Dans ce modèle, il s'agit d'implémenter le protocole de communication directement à partir des tableaux de représentation interne des machines d'émission et de réception. Pour cela, un ensemble d'unités de traitement associées aux différentes fonctions de traitement doit être spécifié.

Ce modèle d'architecture est bien un modèle spécifique pour l'implémentation des protocoles de communition car il emploie par sa structure interne les différents éléments du protocole proprement dit.

5 Un Modèle d'Architecture Spécifique

Le modèle d'architecture spécifique au traitement temps réel des protocoles de communication de type contrôle-commande est composé de trois parties : l'interface station, le cœur du circuit et l'interface réseau.

Dans ce qui suit, on présentera les différents éléments nécessaires pour une implémentation structurelle d'une telle architecture. [Sah92_1]

5.1 Interface côté Station

Afin de relier le circuit de communication à une station externe via un bus, une interface appelée "interface station" est utilisée. Elle est composée de deux parties, une partie adaptation et une partie bufferisation.

5.1.1 Partie Adaptation

Cette partie permet d'adapter les signaux internes au circuit de communication avec les signaux du bus externe (bus station). Les signaux échangés sont des signaux de bus d'échange de données et de contrôle. Elle comporte la logique des chemins d'accès aux éléments de mémorisation internes de la partie bufferisation.

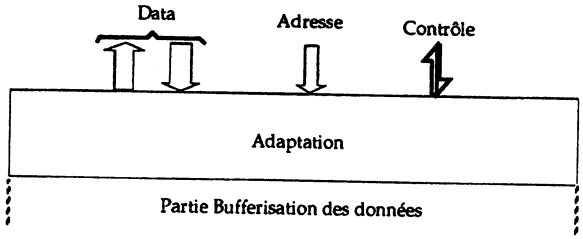


figure 13: Adaptation

Afin de lire ou d'écrire des données à travers l'interface du circuit de communication, le protocole d'échange de données entre le bus externe et la partie mémorisation des données doit être spécifié.

Trois types d'informations sont nécéssaires pour établire ce dialogue:

. Les informations de contrôle
. L'adressage des ressources internes
. Le flux de données en entrée ou en sortie.

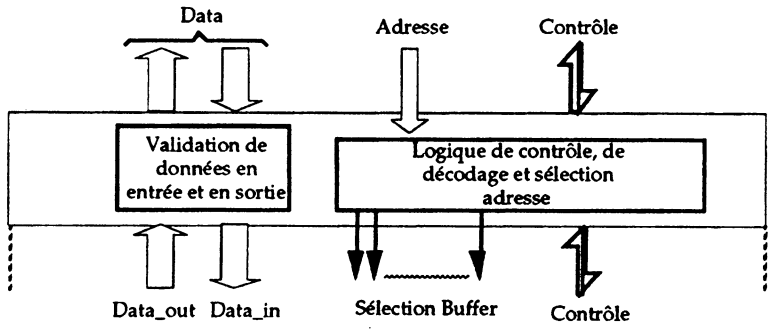


figure 14: éléments structurels de la partie Adaptation

. Les informations de contrôle:

Ce sont les différents signaux de contrôle qui permettent de valider les données et les adresses présentes sur le bus et d'indiquer qu'il s'agit d'une lecture, d'une écriture ou d'une indication d'une interruption.

. Adressage des ressources internes :

Les différentes ressources internes adressables sont les registres et les fifos. Chaque ressource est identifiée par son adresse, le mode d'opération (lecture ou écriture) et la validation de l'opération.

. Le flux de données :

Deux flux de données peuvent traverser le circuit, l'un en émission et l'autre en réception. Les lignes de données correspondant sont spécifiées par leur taille et leur validation en sortie ou en entrée.

5.1.2 Partie Bufferisation

La mémorisation des données est réalisée à l'aide des registres ou des fifos. Ces éléments de mémorisation doivent être spécifiés et identifiés afin de pouvoir les adresser en lecture ou en écriture.

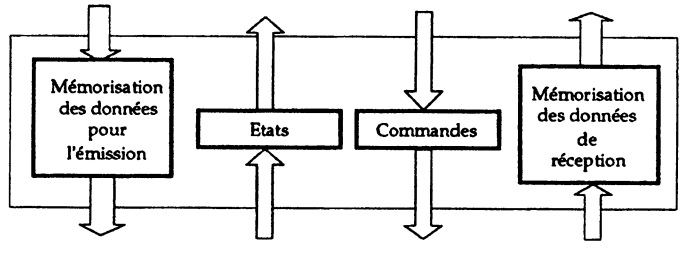


figure 15: bufferisation

. Registre:

Chaque registre est soit en lecture ou en écriture. Les registres à lecture seulement sont appelés les registres d'état et les registres à écriture sont des registres de commandes ou de paramètres. Chaque registre est identifié par une adresse et son mode d'accès en lecture ou en écriture.

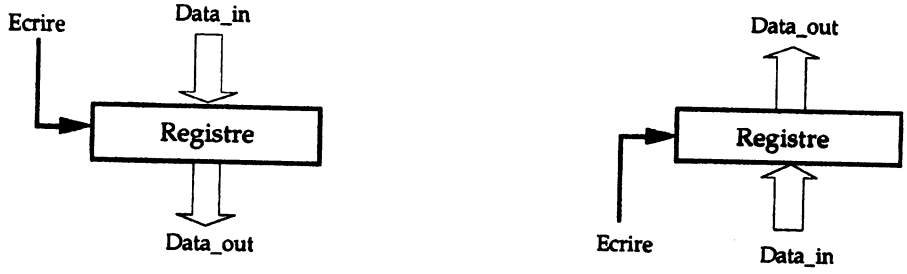


figure 16: registre en réception et en émission

. Fifo:

Les fifos sont aussi en lecture ou en écriture. Un fifo en lecture est vu comme étant une source de données alors qu'un fifo en écriture est un puit de données. Une adresse unique est suffisante pour identifier un fifo en lecture ou en écriture.

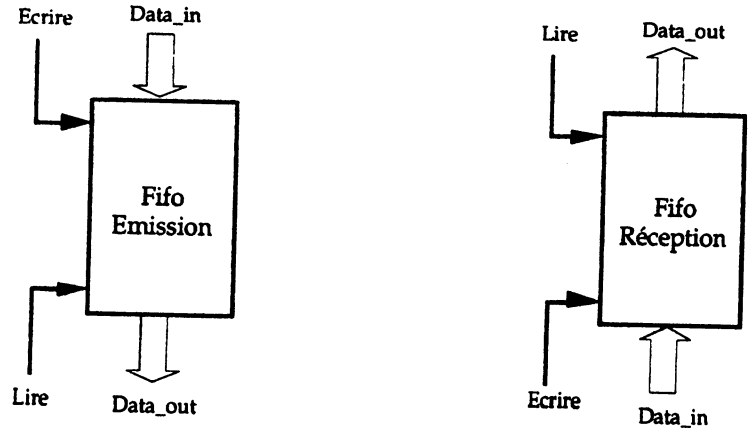


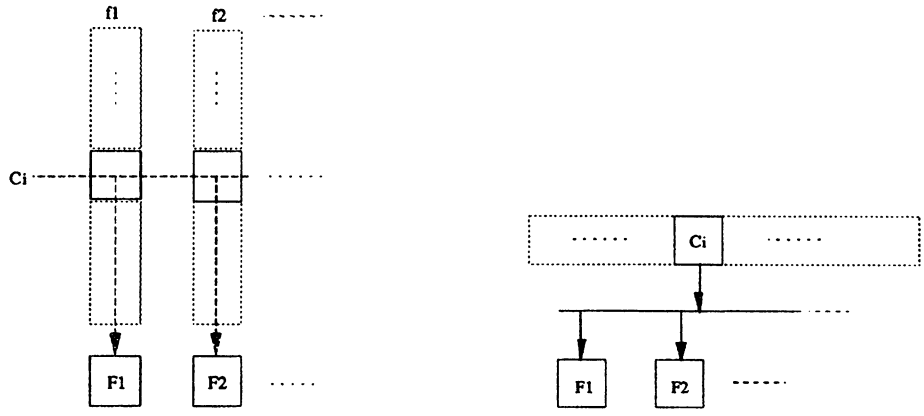
figure 17: fifo en réception et en émission

5.2 Coeur du circuit

Le traitement des fonctionnalités de la sous couche MAC est réalisé par le cœur du circuit. Il consiste principalement en deux machines de traitement interne : la machine de réception et la machine d'émission. L'architecture correspondante à chaque machine est basée sur un module de contrôle de séquencement (ou traitement syntaxique de la trame) et d'un ensemble d'unités de traitement internes du protocole (ou traitement sémantique). Le traitement de séquencement en émission ou en réception utilise le même principe puisqu'il s'agit bien dans les deux cas du traitement de la structure de la trame en émission ou en réception. De ce fait, la présentation du contrôleur de séquencement dans le paragraphe suivant est valable pour la machine d'émission et la machine de réception [Sah92_2].

5.2.1 Contrôle de séquencement : traitement syntaxique de la trame

Le tableau de représentation interne (paragraphe 3.5.2 du chapitre 1) exprime bien le séquencement des différents champs dans une trame. Ce séquencement doit prendre en compte d'une part la fin du champ courant et d'autre part la connaissance du champ suivant. Les différents champs peuvent être disposés verticalement (figure 18.a) ou horizontalement (figure 18.b). Dans le premier cas, lorsqu'on est sur la ligne du champ Ci, toutes les fonctions associées sont positionnées et les autres sont désactivées. Ceci correspond bien au tableau de représentation interne. Dans le second cas, uniquement les fonctions associées au champ Ci sont activées.



(a) (b)

figure 18: activation des fonctions de traitement

(a) verticale

(b) horizontale

L'implémentation du tableau peut être réalisée au moyen de deux techniques : soit par une structure régulière de type matriciel (ex: PLA, ROM) reflétant le contenu du tableau (figure 19.b), ou par une structure irrégulière appelée "câblée" (ex: bascules RS, D, portes logiques, ...) (figure 19.c). Ces deux techniques correspondent bien sûr à l'implémentation des contrôleurs à machine d'état fini. Généralement, l'avantage de la première technique est l'utilisation d'une surface réduite lorsque le tableau est grand, par contre la deuxième technique offre un temps de réponse plus réduit. L'exemple suivant montre l'implémentation dans les deux techniques de séquencement d'une trame simple composées de trois champs :

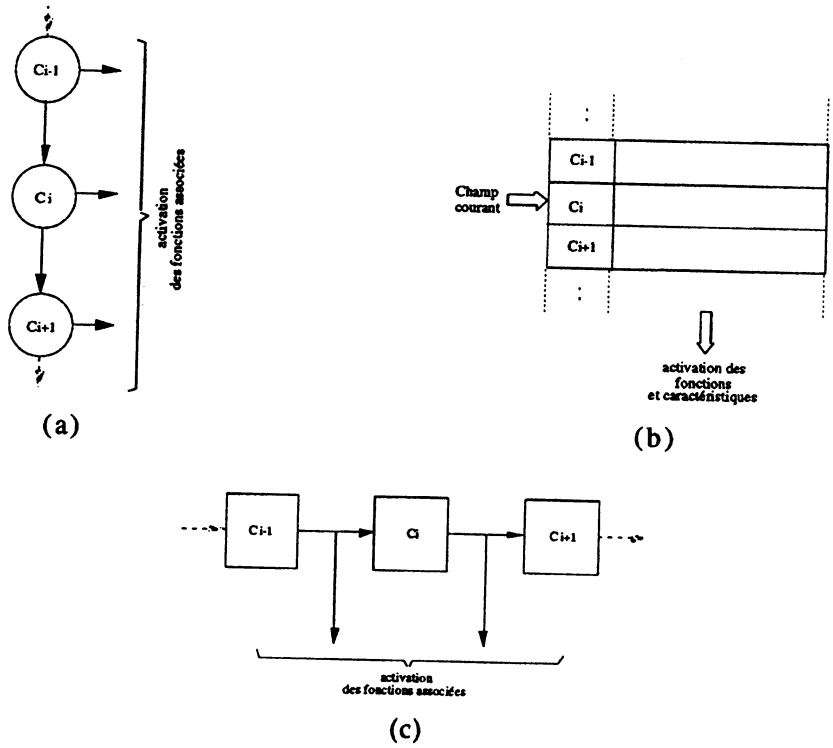


figure 19: implémentation de séquencement de champs cbascules

(a). graphe de représentation

(b). tableau (PLA)

(c). logique câblée.

Dans la suite, les divers exemples d'implémentation des différents types de séquencement seront présentés dans les deux techniques c'est-à-dire matricielle et câblée.

5.2.1.1 Séquencement des éléments binaires dans un champ:

La fin de traitement de chaque champ est connue par la donnée de sa taille ou par un signal indiquant la fin de champ. Le bloc implémentant cette fonction est composé d'un compteur de *eb* et d'un comparateur. Le compteur de bits est actionné par l'horloge de traitement bit et sa valeur est comparée à la taille du champ par l'intermédiaire du comparateur. Un signal externe peut éventuellement être utilisé dans le cas où le champ est de taille variable et la fin de son traitement est signalée par un autre module externe.

La taille du champ peut être fournie par le tableau (figure 20) ou par un multiplexage des différentes valeurs.

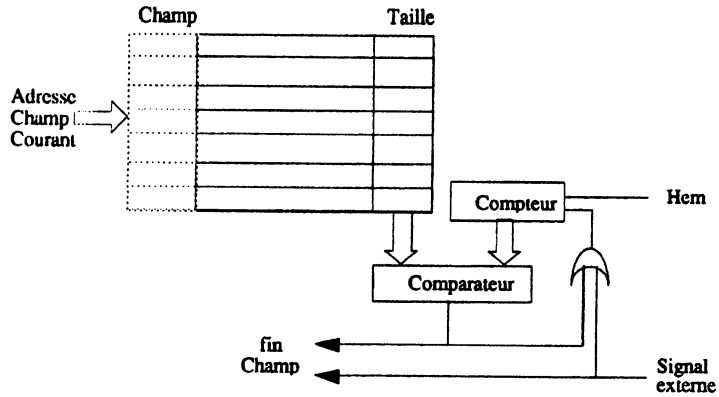


figure 20: calcul du séquencement des bits dans un champ

5.2.1.2 Séquencement des champs dans une trame:

Le séquencement des champs dans une trame est spécifié par le protocole considéré. Le champ suivant peut être soit:

a. *le même champ:*

Dans ce cas, le champ suivant est le même. C'est le cas où le champ courant est en cours de traitement. La même adresse du champ en question est regénérée en entrée du tableau (figure 21).

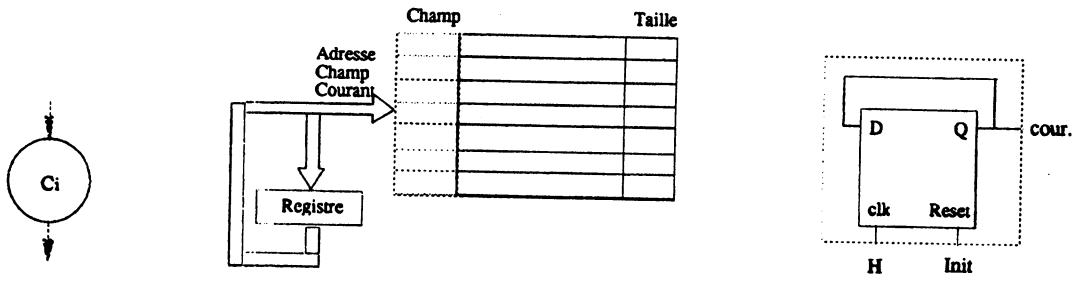


figure 21: traitement du même champ

Cela se traduit dans le cas de l'implémentation à base de bascules, par la même sortie de la bascule qui est rebouclée sur son entrée.

b. *le champ suivant sans condition:*

Dans le cas où le traitement du champ courant est terminé, le champ suivant est celui qui se trouve juste après, et dont l'adresse est l'incrémentation du champ courant (figure 22).

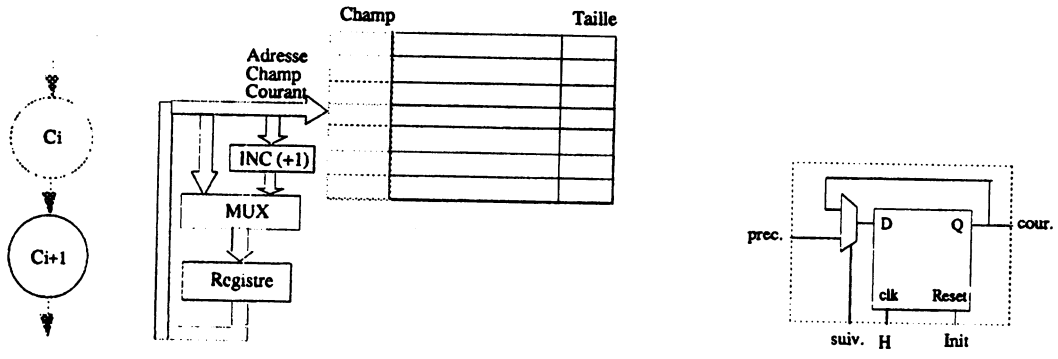


figure 22: traitement du champ suivant sans condition

Dans le cas de l'implémentation à base de bascules, le traitement du champ suivant est indiqué par la fin de traitement du champ précédent (suiv.). Ce signal est positionné soit par la partie traitant le séquencement des bits ou par un signal provenant d'une unité de traitement.

c. *le champ suivant avec condition:*

Le champ suivant est un branchement vers un autre champ lorsqu'une condition est réalisée. La condition est un signal externe générée par une unité

de traitement. Dans ce cas, l'adresse du champ suivant est une adresse de branchement (figure 23).

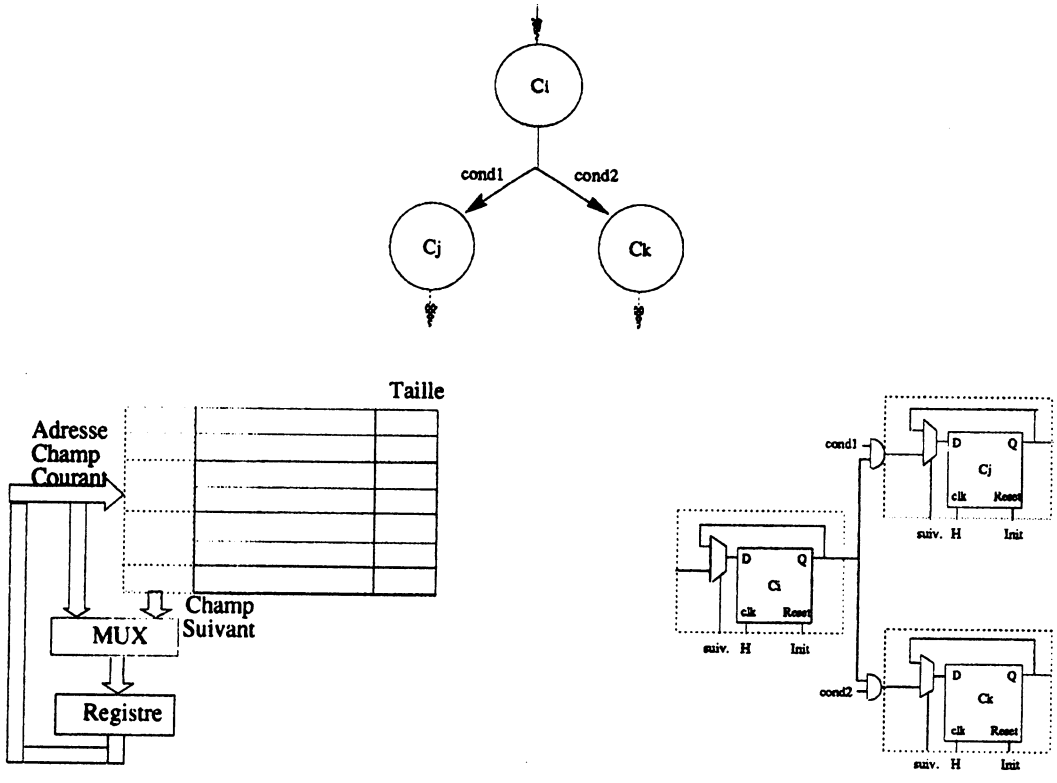
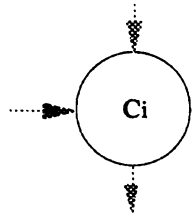


figure 23: traitement d'un branchement à un champ

Deux portes logiques ET sont rajoutées à l'entrée de chaque bascule de destination afin de tester si la condition a été réalisée dans l'implémentation à base de bascules.

d. *le champ suivant spécifié par l'extérieur:*

Afin de donner la possibilité de réaliser des branchements à un champ donné spécifiés par l'extérieur (ex: réponse dans la trame), une adresse de branchement externe est utilisée avec un signal de validation (figure 24).



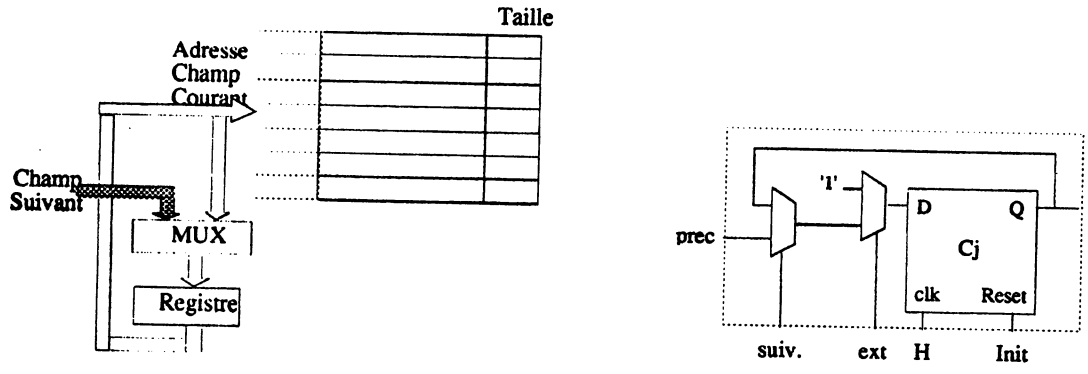


figure 24: traitement d'un branchement externe

Le branchement externe est prioritaire, ainsi le signal externe indiquant l'activation (ext.) est positionné à l'entrée de la bascule.

5.2.1.3 Modèle général du séquencement des champs:

Le modèle complet de séquencement des champs est représenté par la figure suivante:

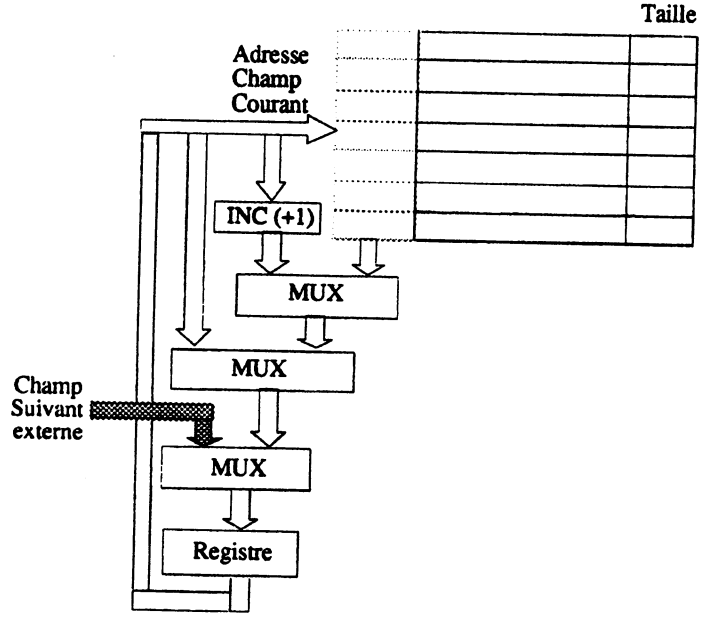


figure 25: modèle général de séquencement des champs

Selon les besoins du protocole en question, le modèle de séquencement peut être plus ou moins complet. Certains cas peuvent ne pas figurer dans un protocole de communication. Le cas d'un branchement externe vers un champ est typiquement utilisé lorsqu'il y a une demande d'une réponse dans la trame. Dans ce cas, l'adresse de branchement est fournie par la machine de réception.

5.2.2 Traitement : Unités de Traitement internes (Temps-Réel)

Une unité de traitement permet de réaliser un traitement sur les *eb* d'un champ. Ce traitement est valide pendant toute la durée du champ et dépend de sa taille.

L'unité de traitement est spécifiée par son interface avec l'extérieur et son corps qui définit le comportement interne. L'interface est décrite en spécifiant les différents signaux de données *Data_in* et de contrôle *Ctrl* nécessaires pour reconnaître quel type d'unité de traitement (synchrone ou asynchrone). Le module de contrôle se contente uniquement de positionner les signaux de contrôle (*Ctrl*) et de choisir les paramètres externes (*Data_in*) correspondant au traitement de chaque champ.

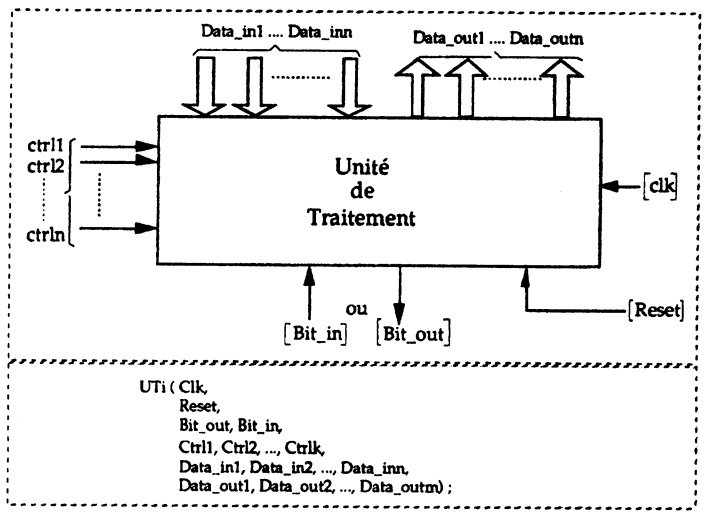


figure 26: Unité de traitement

Le corps qui définit le comportement de l'unité de traitement sur une durée d'un *eb* peut être de type synchrone ou asynchrone. Dans le cas d'une unité synchrone, le traitement est réalisé sur les différents *eb* du champ au rythme de l'horloge. Le type d'unité de traitement est reconnu à la spécification de l'interface par la présence d'un signal horloge en entrée.

Un exemple de traitement est le traitement codage CRC des champs en émission.

5.2.3 Communication inter-modules

a) Partie Contrôle — Unité de traitement

La communication entre le gestionnaire syntaxique (Partie Contrôle) et une unité de traitement consiste en un signal d'activation et des signaux de sélection de paramètre de fonctionnement (figure 27).

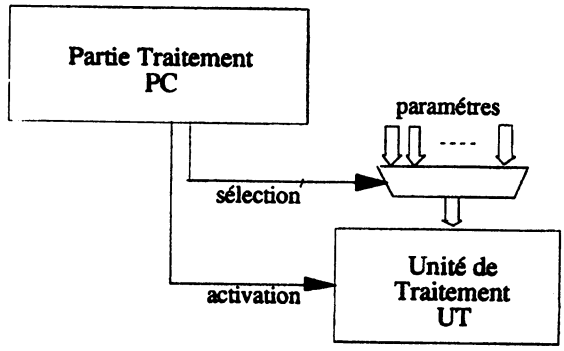


figure 27: communication partie contrôle—unité de traitement

Le signal d'activation est positionné au début de traitement du champ associé et désactivé à sa terminaison. La durée de cette activation est égale à la taille du champ. Les paramètres qui doivent être positionnés durant cette activation sont sélectionnés s'il y en a plusieurs avant l'activation.

b) Unité de traitement — Partie Contrôle

L'unité de traitement rend compte alors de son état à la partie contrôle en positionnant un certain nombre de signaux d'état (figure 28). Ces signaux sont sélectionnés pour réaliser des tests de contrôle de séquencement ou pour réaliser d'autres fonctions telles que l'initialisation ou la signalisation.

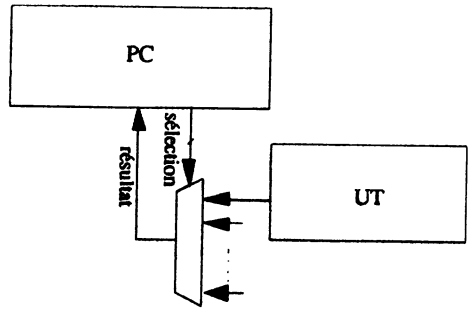


figure 28: communication
unité de traitement—partie contrôle

c) Unité de traitement — Unité de traitement

La communication entre les unités de traitement est réduite à un simple positionnement de paramètres (figure 29). Une unité de traitement donnée peut utiliser éventuellement la sortie d'une autre. La sélection du paramètre dans le cas de plusieurs est effectuée par la partie contrôle.

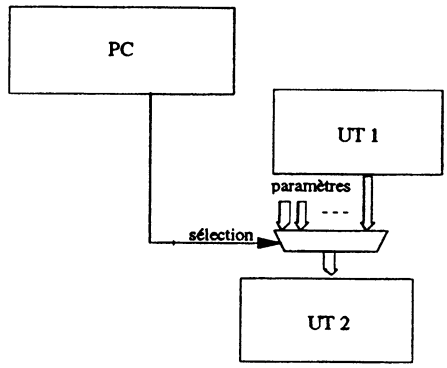


figure 29: communication unité de traitement—unité de traitement

d) Contrôle d'Emission — Contrôle de Réception

L'activation d'une procédure d'émission (ou de réception) n'est pas déclenchée uniquement par l'interface station (hôte), mais elle peut être réalisée par la réception (ou l'émission) (figure 30).

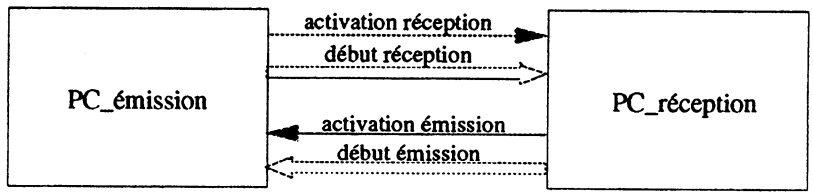


figure 30: communication
partie contrôle émission—partie contrôle réception

Dans le cas d'une réponse dans la trame, la station émettrice passe automatiquement en réception en activant la procédure de réception après le dernier *eb* émis. Ceci est réalisé en envoyant l'adresse début de la trame à recevoir et du signal d'activation de début de réception. La station qui était réceptrice passe en émission en positionnant l'adresse de la trame à émettre et le signal d'activation de l'émission.

Ce fonctionnement est une réaction très critique car la décision doit être prise dans la durée d'un élément binaire (même au niveau du dernier slot) lorsqu'une condition est réalisée sur le dernier bit du champ d'une trame.

5.3 Interface Côté Réseau

La partie physique permet au circuit de communication de se connecter au médium de communication. Parmis les différentes fonctionnalités de cette partie, il y a plusieurs opérations dont elle est chargée d'exécuter:

- En réception:

  - détection de début de réception
    (présence d'une trame sur la ligne: préambule).
  - extraction de l'horloge de réception;
  - décodage des informations (ex: Manchester, NRZ ...);
  - synchronisation avec l'horloge de réception;
  - détection d'une collision ou d'un viol de code sur le médium.

- En émission:

  - détection d'état de repos sur le médium;
  - extraction de l'horloge de réception;
  - encodage des informations à émettre;
    (ex: Manchester, NRZ ...);
  - synchronisation avec l'horloge d'émission;
  - détection d'une collision sur le médium.

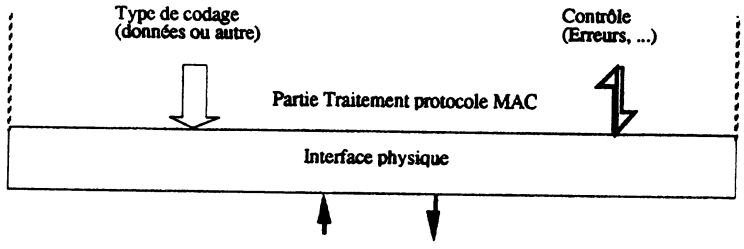


figure 31: interface physique

5.3.1 Les Signaux de l'Interface Ligne

Le cœur du circuit de communication dialogue avec la partie physique par le biais d'un ensemble de signaux qui sont classés en deux groupes:

signaux de l'interface émission et signaux de l'interface réception. Parmis ces signaux, il y a :

. Horloge de réception;
. Horloge d'émission;
. Ligne de données en réception;
. Ligne de données en émission;
. Signal d'indication de repos sur la ligne;
. Signal d'indication de présence d'une trame;
. Signal d'indication de collision et de viol de code ;
. Signaux indiquant le type de codage des informations émettre.

5.3.2 Les Modules Internes

Les différents modules composant la partie physique représentent les fonctionnalités internes qui sont:

. module de codage des bits d'information en émission
. module de décodage des bits d'information en réception
. module de gestion d'horloge: extraction d'horloge d'émission et de réception
. module de détection de l'état repos de la ligne de communication;
. module de détection de collision et de viol de code ...

En ce qui nous concerne, le travail de conception du circuit de communication s'arrête au niveau de l'interface physique en spécifiant les différents signaux échangés avec la partie physique.

5.4 Architecture Générale du Circuit de Communication

L'architecture générale du circuit de communication est composée de trois parties : une partie de traitement d'émission (machine d'émission), une partie de traitement de réception (machine de réception) et une partie interface avec l'extérieur.

La machine d'émission (ou de réception) est constituée d'une partie contrôle qui fournit tous les signaux de contrôle nécessaires et d'une partie traitement qui englobe toutes les unités de traitements internes. Il y a deux types d'unités de traitements : les unités de traitement internes qui réalisent des traitements sur les champs ou autres, et des unités de traitement

d'interface qui sont utilisées spécialement pour gérer les échanges de données avec la partie interface.

La partie interface est composée d'une partie adaptation et d'une partie bufferisation. La partie adaptation permet de réaliser une interface de signaux entre le bus externe et le bus interne du circuit. Elle est constituée essentiellement de portes logiques et de multiplexeurs. Par contre la partie bufferisation regroupe tous les éléments de mémorisation qui sont les registres et les FIFOs. L'accès à ces différents éléments est réalisé à travers les signaux de données, de contrôle et d'adresse de l'interface adaptation.

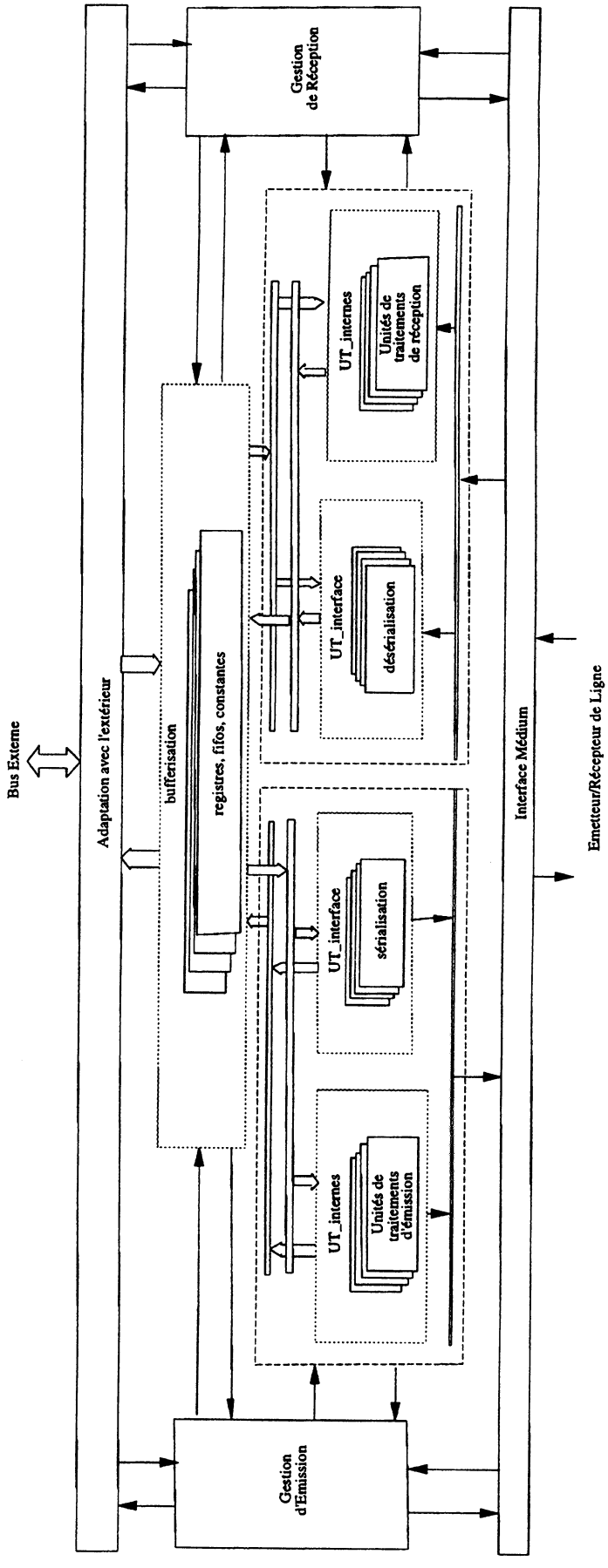


figure 32: modèle d'architecture générale

Conclusion

Un modèle d'architecture de circuit de communication a été proposé pour l'implémentation des protocoles de communication de contrôle-commande.

L'adaptabilité de cette architecture aux différents protocoles tient à deux facteurs:

(1) Flexibilité : Autrement dit, l'architecture du circuit de communication peut être construite en assemblant les différents modules paramétrables nécéssaires à l'exécution du protocole à implanter.

(2) Généricité : Le modèle est basé sur la séparation entre l'analyse de la syntaxe des trames et le traitement des données contenues dans ses champs.

Notons enfin que ce modèle d'architecture n'est pas bien adapté aux protocoles offrant des services orientés connexion. Une architecture de type microprogrammé peut prendre en compte de telles fonctionnalités qui nécessitent de suivre la liaison entre plusieurs trames successives.

A partir des spécifications du protocole d'une part et du modèle d'architecture d'autre part, une démarche d'implémentation automatique (appelée *synthèse*) peut être adoptée pour réaliser des circuits de communication.

Le processus de synthèse se décompose alors en plusieurs étapes dont la première est la spécification du protocole de communication et la dernière consiste à générer l'architecture du circuit de communication correspondant.

Chapitre 3

Synthese d'Architecture de Circuits de Communication

Chapitre 3

— SYNTHESE D'ARCHITECTURE DE CIRCUITS DE COMMUNICATION —

Introduction

Dans ce chapitre, on présentera d'abord la démarche de conception des circuits de communication en partant des spécifications pour aboutir à l'architecture. Un langage de description de protocoles de communication (de niveau bas) appelé MACS est ensuite introduit. Les différentes unités de description du langage seront par conséquent détaillées. Et enfin, les différentes étapes de conception d'architecture seront décrites à travers des transformations de la représentation interne de la spécification pour aboutir à une implémentation structurelle de l'architecture.

1 Méthode de Conception

Plusieurs travaux ont été réalisés dans le domaine de la synthèse d'architecture pour l'implémentation de circuits que ce soit dans un domaine général tel que les circuits programmables (microprocesseurs) ou dans un domaine spécifique tel que le traitement du signal (voir Annexe A). Les méthodes utilisées dans un domaine ne présentent généralement pas les mêmes performances dans d'autres domaines si elles sont appliquées telles quelles. Bien qu'un corps de connaissances existe sur les architectures, aucune théorie unificatrice n'a été élaborée pour qu'elle puisse être appliquée dans les divers domaines d'application.

Une approche de conception pour produire des outils automatiques, qui est généralement adoptée, est d'utiliser toutes les connaissances du domaine d'application afin de répondre aux contraintes des cahiers des charges imposés. Pour cela, notre domaine d'application qui est celui des circuits de communication, et particulièrement ceux qui implantent les protocoles de niveau bas c'est à dire avec des fonctionnalités temps-réel, présente quelques caractéristiques permettant la mise en œuvre d'outils de synthèse spécifiques.

Une démarche de conception appropriée a été élaborée en partant d'une description du protocole dans un langage spécifique pour aboutir à l'architecture du circuit de communication correspondant [Sah93].

1.1 Spécificité du Domaine d'Application

Le modèle du chapitre 2 est bien adapté pour implémenter les protocoles de niveau bas qu'on appellera "protocoles temps-réel au niveau élément binaire". Tous les traitements internes doivent respecter la contrainte de fonctionnement sur la durée d'un ou de plusieurs *eb*. Les traitements différés sont supposés être réalisés au niveau des couches supérieures.

Par conséquent, la nature du problème fait que l'ordonnancement des opérations à réaliser est déjà figé au départ car chaque traitement doit être invoqué dès que les données sont arrivées. L'arrivée des données est connue au départ par la connaissance syntaxique des différentes trames du protocole (voir chapitre 1). Ces données sont représentées par les différents champs de chaque trame. Ainsi, un champ ne peut pas être traité alors qu'il n'est pas encore arrivé, et son traitement ne doit pas être différé.

1.2 Schéma de Conception Adopté

Le schéma de conception adopté consiste à suivre les trois étapes principales qui sont généralement adoptées par les systèmes de synthèse d'architectures :

(1) A partir d'une spécification sous forme d'un cahier des charges, l'utilisateur doit la transcrire dans un langage spécifique à l'outil de synthèse (figure 1). La description produite est considérée dans les langages de programmation comme étant un programme. Cette étape constitue la première phase dans une démarche de synthèse de haut niveau.

(2) La deuxième étape consiste à traduire cette description sous une forme intermédiaire (ou une représentation interne) afin de pouvoir réaliser des transformations sur ses objets. Cette étape utilise un compilateur du langage spécifique (appelé MACS) permettant d'effectuer une analyse syntaxique et sémantique de la description et de générer la représentation interne.

(3) A partir de la représentation interne, plusieurs transformations sont appliquées permettant d'aboutir à une architecture en respectant le modèle d'architecture déjà spécifié. L'allocation des ressources est réalisée à travers une bibliothèque de composants qui est utilisée pour implémenter les différents traitements et objets internes. Celle-ci est spécifiée par l'utilisateur afin d'être indépendante d'une technologie donnée.

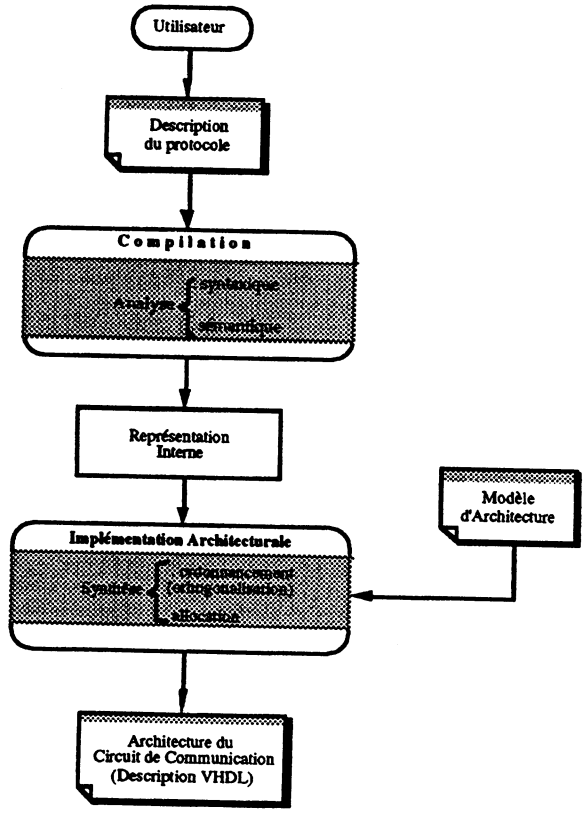
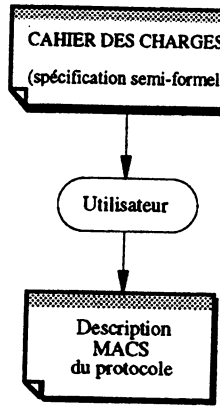


figure 1: schéma de conception générale

2 Structure Générale d'une Description

A partir du cahier des charges représentant la description semi-formelle du protocole de communication et de l'environnement d'implémentation, l'utilisateur produit une description MACS de l'application composée de sept unités de description.



Les différentes unités de description correspondent bien à certaines parties de l'architecture cible déjà présentée dans le chapitre 2. Ainsi la description globale

comporte une partie description du protocole (processus émission, processus réception, déclaration des champs) et une partie concernant les interfaces externes (adaptation, bufferisation et interface basse).

Le schéma global de la description selon l'architecture du circuit est représenté dans la figure suivante:

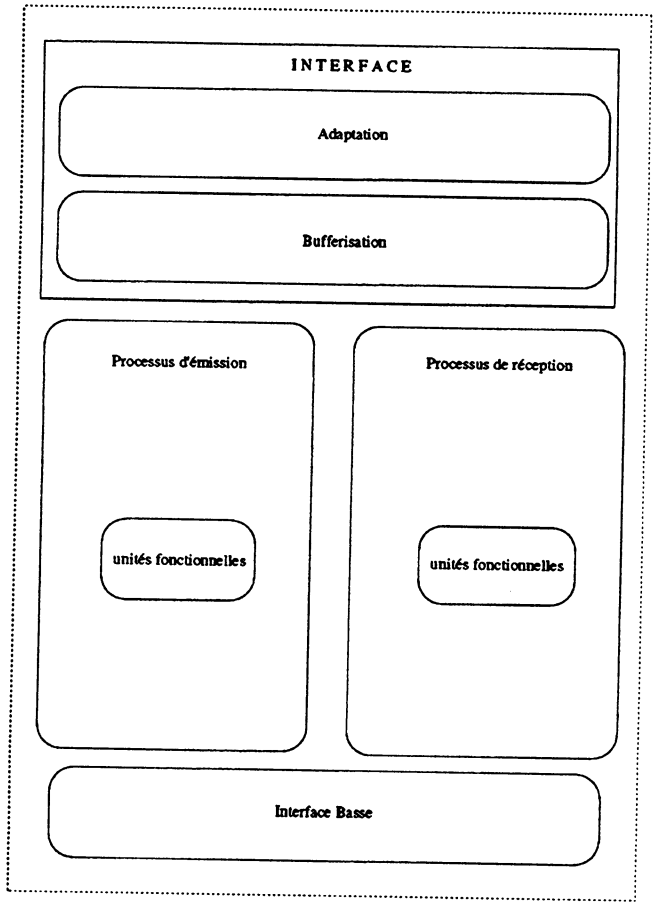


figure 2 : schéma global de la description MACS

Les différentes unités de description comportementale sont:

(1) Unité d'interface

La description de l'environnement côté station est réalisée par la spécification de la partie adaptation et de bufferisation.

(1.1) Unité d'adaptation

Elle comporte la description des différents signaux d'interface qui se composent de lignes de contrôle et de bus de données.

(1.2) Unité de bufferisation

Elle permet de déclarer les différents éléments de mémorisation utilisés pour stocker des informations destinées à l'interface haute telles que les données utilisateurs et les comptes rendus de contrôle.

(2) Unité d'interface physique

L'interface avec la partie qui s'occupe du traitement physique est décrite en spécifiant des signaux de contrôle échangés tels que les signaux indiquant l'état de la ligne du réseau.

(3) Unité de fonctions de traitement

Les différentes fonctions de traitement internes sont utilisées dans les unités d'émission et de réception en tant que procédures externes et sont déclarées au niveau de l'unité de description des unités de traitement.

(4) Unité de champs

La spécification de la structure des trames d'un protocole est réalisée par la déclaration des différents champs constituant ces trames. Les caractéristiques de chaque champ sont alors décrites.

(5) Unités de Description du Corps du Protocole:

Le protocole de communication est spécifié par deux processus: Processus d'émission et Processus de réception.

(5.1) Unité d'émission

Le processus d'émission permet de décrire le séquencement des différents champs et les différents traitements réalisés sur chaque champ.

(5.2) Unité de réception

Comme dans le cas de l'émission, le processus de réception permet aussi de spécifier le séquencement des différents champs et des traitements associés.

3 Compilation de la Description

3.1 Eléments du Langage

La description comportementale dans les systèmes de synthèse s'exprime sous diverses formes de description : graphique (IMHOTEP) [Dron88], algorithmique (Pascal) ou des langages de description fonctionnelle de circuits (ISPS).

Dans le domaine de la communication, un langage de description dérivé des langages de spécification de protocoles tel que ESTELLE a été utilisé pour décrire les protocoles en vue de leurs implantations materielles [Kris92]. Ce langage qui a été déjà évoqué dans le chapitre 1, est appelé APSL. Il est basé sur un modèle de machine d'état fini étendu dont les communications sont synchrones et il ne possède pas de notion de hiérarchie. Le système de synthèse introduit des états intermédiaires entre les états dits globaux du système à implémenter. Ce langage s'apprête bien à la description des protocoles des réseaux locaux et inter-urbains.

Notre approche consiste à proposer un langage de description pour décrire le système en deux parties : Partie interface et partie traitement protocole. Ce langage est appelé MACS.

La première partie spécifie les interfaces en exprimant d'une part les différents signaux et les bus externes, et d'autre part les variables internes qui sont les différents éléments de mémorisation internes associés aux différents chemins d'accès à partir de l'extérieur.

La seconde partie permet de décrire le protocole de communication sous forme de deux processus : processus d'émission et processus de réception. Chaque processus est représenté par une machine d'état fini où chaque état correspond bien à un objet du protocole qui est le champ. Les transitions entre les différents états correspondent alors au séquencement des champs dans le protocole.

L'avantage de cette forme de description est qu'elle permet à l'utilisateur d'être très proche de la spécification informelle de départ du protocole de communication à implémenter. Elle permet aussi de fournir des éléments de choix d'interface avec l'environnement externe et par conséquent une flexibilité

dans l'implémentation. Par la suite, afin de garder le même schéma de présentation d'un protocole de communication que celui du chapitre 2, on commencera par décrire la partie qui s'occupe de la gestion syntaxique, suivie par celle de la gestion sémantique et finir par la description de l'environnement d'implémentation.

3.1.1 Description des Trames d'un Protocole

Cette description de la gestion syntaxique permet de déclarer les différentes trames d'un protocole de communication, les différents champs de chaque trame et les caractéristiques de chaque champ.

Il s'agit alors d'interpréter le graphe de représentation d'un protocole de communication par la déclaration de l'ensemble des champs de toutes les trames.

3.1.1.1 Declaration d'un Champ

Un ensemble de caractéristiques peuvent être associées à la déclaration d'un champ: son identificateur, son type, sa taille et sa valeur.

3.1.1.1.1 Identification d'un Champ

Un champ est déclaré par un identificateur qui est un ensemble de lettres alphanumériques. La première lettre doit être alphabétique.

3.1.1.1.2 Type d'un Champ

La déclaration du type permet au niveau MAC d'indiquer au niveau physique le type de champ à traiter. Pour cela, deux types sont associés à un champ. Il peut être de type "data" ou de type "synchro". Le type "data" indique que le champ est constitué d'un ensemble de *eb* de niveau MAC ou du niveau supérieur, alors que le type "synchro" indique que le champ est de niveau bas (physique).

3.1.1.1.3 Taille d'un Champ

Un champ peut avoir une taille constante ou variable. La taille d'un champ `synchro` est un multiple d'unités *eb* . La taille indique alors le nombre d'unités *eb* équivalentes.

3.1.1.1.4 Valeur d'un Champ

Dans le cas d'un champ constant, une valeur lui est associée. Cette valeur est exprimée sous forme d'un vecteur de *eb* . La taille du vecteur de *eb* est égale à celle du champ.

3.1.1.2 Exemple d'une Déclaration d'un Champ

L'exemple suivant donne la description générale de l'unité de déclaration des champs des trames.

```
field
        data_field
            [déclaration des champs de données]
            IDENT [12] ;        # champ de taille 12 bits ¥
            ...
        enddata
        synchro
            [déclaration des champs de synchronisation]
            SOF [5] ;       # champ de taille 5 bits ¥
            ...
        endsynchro
endfield
```

3.1.1.3 Interprétation Matérielle

L'implémentation de la gestion syntaxique est réalisée dans la partie contrôle des processus d'émission et de réception. Elle consiste à implémenter les différentes connaissances sur les caractéristiques des champs.

- La taille d'un champ est une donnée constante permettant d'indiquer la fin de traitement du champ.

- La valeur d'un champ est une donnée constante utilisée en réception pour la comparer avec la valeur du champ reçue.

- Le type d'un champ permet de générer un signal vers l'entité physique pour indiquer la nature du codage (émission) ou de décodage (réception) qu'il faut utiliser.

Toutes ces valeurs sont des constantes et sont caractérisées au niveau matériel dans le cas :

- d'une implémentation matricielle, par une colonne de valeurs constantes correspondant à une caractéristique donnée.
- d'une implémentation câblée, par un multiplexage des différentes constantes.

3.1.2 Description des Traitements Associés

Cette description permet de décrire les traitements associés aux champs des trames d'un protocole de communication. Parmi ces traitements, on a le séquencement des champs et les différentes fonctions de traitement internes.

Il s'agit alors d'interpréter les actions associées à chaque noeud du graphe de représentation d'un protocole de communication en un ensemble d'instructions de traitements.

3.1.2.1 Séquencement des Champs

Chaque noeud du graphe de représentation d'un protocole de communication est indiqué par un identificateur. On appelera un noeud du graphe par un état du graphe. Alors, dans chaque état, un champ lui est associé.

Une transition entre deux états est indiquée par une instruction de branchement (`nextstate`). Un paramètre indiquant l'identificateur de l'état de branchement est spécifié.

```
nom_etat    :   nom_champ
                begin
                        nextstate (nom_etat_i) ;
                    [ instructions_traitement]
                end
```

Dans le cas où l'instruction `nextstate` n'a pas été employée, l'état suivant dans la description sera pris par défaut.

3.1.2.2 Traitement Interne d'un Champ

Le traitement interne d'un champ consiste à appliquer des traitements spécifiques qui sont définis par l'utilisateur. Chaque fonction de traitement est déclarée auparavant dans la partie déclaration des unités de traitement.

L'utilisation d'une unité de traitement est réalisée à travers un appel de la fonction en positionnant les différents paramètres associés.

```
nom_etat    :   nom_champ
                begin
                        ...
                        nom_fonct (param_1, param_2, ..., param_n) ;
                        ...
                end
```

3.1.2.3 Traitement d'Interface

Que ce soit en émission ou en réception, les différentes informations de l'interface haute transitent par la partie bufferisation permet de stocker les informations à transférer.

(a) Cas de la Réception:

Les différents champs qui sont destinés à l'interface haute sont mémorisés soit dans des registres ou des FIFOs. L'indication de la partie bufferisation qui permet de stocker le champ en question est traduite par l'instruction *receivef*. L'identificateur de l'unité de mémorisation dans la partie bufferisation est spécifiée comme valeur du paramètre de l'instruction.

```
nom_etat    :   nom_champ
                begin
                        ...
                        receivef(reg_nom) ; # ou receivef(fifo_nom) ; #
                        ...
                end
```

(b) Cas de l'Emission:

Les différents champs envoyés par l'interface haute sont mémorisés soit dans des registres ou des FIFOs. L'indication de la partie bufferisation qui permet de fournir le champ en question est traduite par l'instruction *sendf*. Comme dans le cas de la réception, une valeur de paramètre doit être spécifiée et qui est représentée par l'identificateur de la partie bufferisation concernée.

3.1.2.4 Traitement des Initialisations

La fonction d'initialisation du circuit peut être activée généralement par :

. la fin d'une émission d'une trame,

. la fin d'une réception d'une trame,

. ou une erreur d'émission ou de réception.

Une instruction d'initialisation est nécessaire dans le cas d'une émission ou d'une réception. Cependant, deux instructions sont utilisées pour réaliser la fonction d'initialisation : initialisation locale et initialisation globale.

(a) Initialisation Locale :

Quand on veut faire une initialisation du processus en cours (émission ou réception) sans affecter le déroulement de l'autre, une instruction d'initialisation locale est utilisée (`init_loc`).

(b) Initialisation Globale:

Dans le cas où une initialisation générale du circuit est nécessaire, une instruction (`init_all`) est utilisée pour faire une initialisation de toutes les ressources internes sans affecter la partie qui s'occupe de l'interface.

3.1.2.5 Activation de la réception durant l'émission et vice-versa

La réponse dans la trame est une solution souvent utilisée dans les protocoles de communication de contrôle-commande. Généralement, elle est employée lorsqu'il y a un échange de trames de question et de réponse. Dans ce cas, au lieu d'envoyer la réponse dans une autre trame, elle est insérée systématiquement dans la trame courante. Alors la station qui était en émission passe en réception et celle qui était en réception passe en émission.

Le passage d'un mode à l'autre est réalisé en activant le processus de réception dans le premier cas et le processus de réception dans le second.

(a) Activation de la Réception:

Le passage de l'émission en réception est réalisé en activant le processus de réception et en arrêtant éventuellement celui de l'émission.

Dans ce cas, une instruction d'activation du processus (*recept*) est spécifiée dans la description en précisant l'état dans lesquel le processus devrait commencer le traitement.

(b) Activation de l'Emission:

Le passage de la réception en émission est réalisé en activant le processus d'émission et en arrêtant éventuellement celui de la réception.

Comme dans le cas précédent, une instruction d'activation du processus (*emit*) est spécifiée dans la description en précisant l'état dans lesquel le processus devrait commencer le traitement.

```
nom_etat    :   nom_champ
                begin
                        ...
                        emit (nom_etat_i) ;
                        ...
                end
```

3.1.2.6 Instruction Conditionnelle

L'instruction conditionnelle permet d'exécuter ou non un certain nombre d'instructions qui doivent vérifier une certaine condition. L'exemple type qu'on peut citer est celui du branchement conditionnel. Dans ce cas, l'instruction de branchement (*nextstate*) est utilisée à l'intérieur d'une instruction conditionnelle.

Une condition est réalisée sur le résultat d'une expression booléenne dont les termes sont composés de signaux logiques.

```
nom_etat    :   nom_champ
                begin
                        ...
                        if ( CONDITION )
                        then
                                [ instructions_traitement_sous_condition]
                        endif
                        ...
                end
```

(a) Types d'Expressions Logique dans une Condition

a.1 Constante

Le test de la valeur reçue qui est représentée par une constante de vecteur de *eb*s peut être réalisé en comparant les deux vecteurs *eb* à *eb*.

a.2 Signaux d'interface

Il s'agit de tester les signaux d'interface en prenant en compte l'état des signaux logiques.

a.3 Signaux des Unités de Traitement

On peut également réaliser des tests sur les signaux appartenant à des unités de traitement. Ces signaux indiquent généralement l'état de l'unité de traitement associée.

(b) Instructions sous condition

Toutes les différentes instructions qui ont été présentées précédemment peuvent être utilisées dans une instruction conditionnelle. Une exception est faite sur l'utilisation d'imbrication d'instructions conditionnelles. Celle-ci n'est pas nécessaire car les différentes instructions sont exécutées en parallèle.

3.1.3 Description de l'Environnement d'Implémentation

La description de l'environnement d'implémentation consiste à décrire les deux parties constituant l'interface externe côté station du circuit : la partie bufferisation et la partie adaptation.

3.1.3.1 Partie Bufferisation

La partie bufferisation est spécifiée en déclarant les différents éléments de mémorisation utilisés pour stocker les données d'information et de contrôle. Ces éléments sont classés en deux parties : les registres et les FIFOs.

(a) Registres : Un registre est identifié par un nom et sa taille.

(b) FIFOs : Un FIFO est identifié par un nom, la taille d'un mot et sa profondeur.

```
BUFFERISATION
        ...
        REG_A [16] : reg ;
        ...
END_BUFFER
```

```
BUFFERISATION
        ...
        FIFO_A [16 , 8] : fifo ;
        ...
END_BUFFER
```

3.1.3.2 Partie Adaptation

Elle est composée de deux parties : une partie déclaration des différents signaux externes et une partie indiquant les différents chemins d'accès aux différents éléments de la partie bufferisation.

A. Déclaration des Signaux d'Interface

L'interface externe avec la station permet au circuit de communication et à la station de dialoguer à travers un bus qui est généralement un bus parallèle. Ce bus est composé d'un ensemble de signaux qui sont représentés par des bus de données et des lignes de contrôle.

(a) Les Signaux de Contrôle

Dans le cas des protocoles de type "poignée de main" (hand-shake) ou autres permettant des échanges de données entre entités, un ou plusieurs signaux de contrôle sont généralement utilisés pour exécuter le protocole en question selon sa complexité.

Chaque signal de contrôle est défini par son identificateur et son sens (entrée ou sortie). Un bus de contrôle peut être également défini en indiquant sa taille et son sens.

(b) Les Bus de Données

Un bus de données est également défini par son identificateur, sa taille et son sens.

L'exemple suivant présente quelques déclarations de signaux de contrôle et de bus de données:

```
ADAPTATION
        # Declaration des différents signaux     #
        #        externes de l'interface         #
        ...
        BUS1 [8] : in  data ;
        BUS2 [8] : out data ;
        SIG  [1] : in  ctrl ;
        ...
begin
        [ corps de la partie Adaptation ]
end_ADAPT
```

B. Chemins d'Accès

Le corps de la partie adaptation est constitué d'un ensemble de fonctions d'accès en lecture et en écriture des différents éléments de la partie bufferisation. Trois fonctions sont définies : *INPUT, OUTPUT et RESET.*

(a) INPUT : elle permet d'indiquer comment réaliser la fonction d'écriture dans un élément donné de la partie bufferisation. Pour cela, trois paramètres sont utilisés : l'identificateur de l'élément concerné, le nom du bus externe et la logique qui positionne l'activation de l'opération d'écriture.

(b) OUTPUT : elle permet d'indiquer comment réaliser la fonction de lecture dans un élément donné de la partie bufferisation. Comme dans le cas précédent, trois paramètres sont utilisés : l'identificateur de l'élément concerné, le nom du bus externe et la logique qui positionne l'activation de l'opération de lecture.

(c) RESET : elle permet d'indiquer comment réaliser la fonction d'initialisation externe d'un élément donné de la partie bufferisation. Pour cela, deux paramètres sont utilisés : l'identificateur de l'élément concerné et la logique qui positionne l'activation de l'opération d'initialisation.

L'exemple suivant présente l'utilisation des trois fonctions :

```
ADAPTATION
        # Declaration des différents signaux externes de l'interface #
        ...
begin
```

```
        ...
        input(FIFO_A, BUS_A) = CS . /RS0 . /H ;
        output(FIFO_A, BUS_A) = CS . /RS0 . /H ;
        reset(REG_A) = CS . /RS0 . /H . /RESET ;
        ...
end_ADAPT
```

3.1.4 Déclaration des Unités de Traitement

Un appel à une fonction de traitement peut être utilisé lors d'un traitement d'un ou de plusieurs champs. Chaque fonction de traitement doit être déclarée auparavant par une unité de traitement en précisant les différents signaux correspondant aux ports d'entrée/sortie de l'unité. L'utilisation d'une unité donnée correspond à une instanciation dans le corps de l'architecture en positionnant les différents signaux des ports par les valeurs associées définies dans le corps de la description du protocole.

La déclaration générale d'une unité de traitement se présente comme suit:

```
func_unit
    unit_name
    {
        [inst_funct_port]
    }
    ...
endfunc_unit
```

3.1.5 Déclaration des Signaux d'Interface Basse

L'interfaçage avec la partie physique qui s'occupe des fonctions de codage/décodage bit et de synchronisation, est défini par un ensemble de signaux de contrôle. Ces différents signaux de l'interface bas permettent d'indiquer le type de codage et de décodage bit, le contrôle d'erreurs et la synchronisation du circuit (début et fin de traitement, horloge bit, ...).

```
interf_bas
    ...
        SOF     : in  ctrl;
        EOD     : out ctrl;
```

```
        ERROR  : in  ctrl ;
    ...
endinterf_bas
```

A partir de la description MACS de l'application, une analyse syntaxique et sémantique de cette description est réalisée afin de produire une représentation interne sous forme d'une structure de données spécifique. Pour cela, des outils de compilation (compilateur de compilateur) ont été utilisés : le Lex et le Yacc.

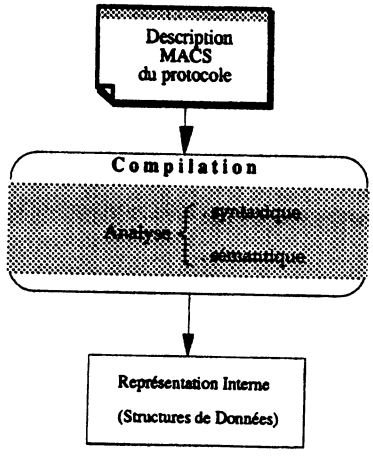


figure 3: Compilation de la description

L'outil appelé Lex permet de faire une analyse lexicale du texte écrit en MACS. Il permet ainsi de reconnaître et de réccupérer les entités syntaxiques et de les présenter à l'analyseur syntaxique qui est le Yacc.

Yacc permet de réaliser une analyse syntaxique du programme afin de vérifier la syntaxe du texte en déroulant les différentes règles de la grammaire du langage.

L'analyse sémantique de la description permet de vérifier l'interprétation des différentes instructions et de réaliser les actions sémantiques associées. Elle permet de produire comme résultat une structure de données de représentation intermédiaire de la description.

3.2 Structure de données de la représentation intermédiaire

Une représentation intermédiaire est produite après les différentes étapes de la compilation. Elle se présente sous forme d'une structure de donnée spécifique pour caractériser les processus d'émission et de réception. Chaque processus est

représenté par un ensemble d'états et dans chaque état, un ensemble d'instructions lui est associé.

La représentation intermédiaire des processus d'émission et de réception est constituée de deux graphes dont chacun représente les différents états de chaque processus. Un ensemble d'instructions lui est associée dans chaque état. Ainsi dans la figure 4, la première colonne indique les différents états et les lignes représentent les différentes instructions dans chaque état.

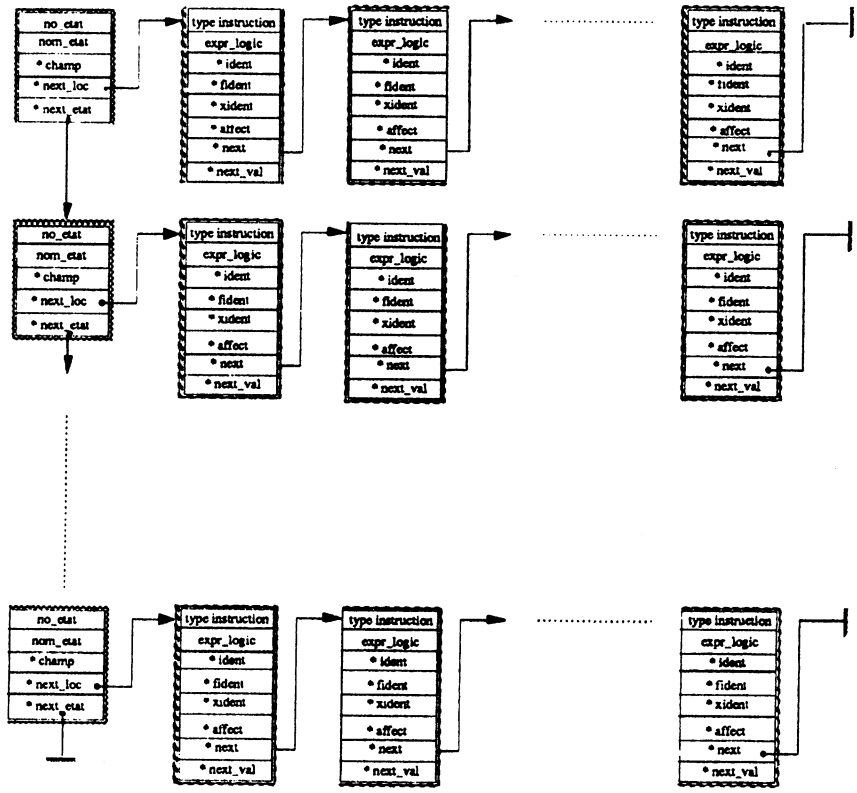


figure 4 : représentation interne d'un processus d'émission ou de réception

(a) La Représentation d'un Etat

Les informations nécessaires pour décrire un état (figure 5) dans la représentation intermédiaire sont :

. nom de l'état : c'est l'identificateur de l'état spécifié dans la description, (associé à un numéro d'état)
. nom du champ : identificateur du champ qui est associé à cet état,
. pointeur vers l'ensemble des instructions dans cet état,
. pointeur vers l'état suivant.

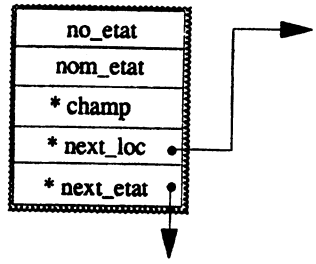


figure 5 : représentation d'un état

(b) La Représentation d'une Instruction

Dans une instruction (figure 6), la représentation interne contient les informations suivantes:

. le type de l'instruction (*Sendf, NextState, ...*) ,
. pointeurs vers les différents paramètres de l'instruction (selon le type) :
  . *fident*, quand il s'agit d'un seul paramètre;
  . *xident*, quand il s'agit de plusieurs paramètres;
  . *affect*, valeurs d'affectation.

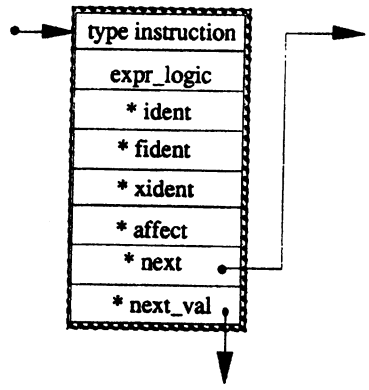
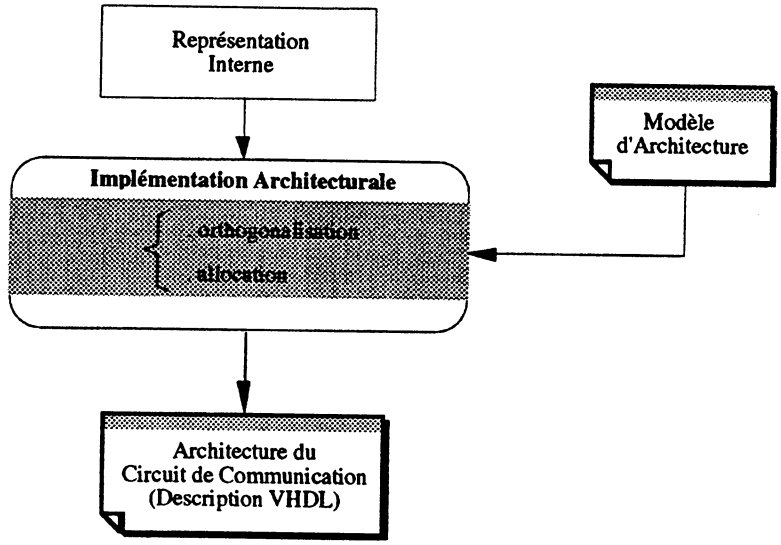


figure 6: représentation d'une instruction dans un état

4 Transformations

A partir de la représentation intermédiaire, plusieurs étapes de transformation (appelée ci-après orthogonalisation) sont réalisées pour aboutir à une représentation susceptibles d'être prise en compte pour une génération automatique en une implémentation structurelle.



L'architecture cible présentée dans le chapitre 2 fournit un modèle d'implémentation où le séquencement des opérations est réalisé à travers l'implémentation de la partie contrôle. Les différents signaux de commande sont générés selon l'état dans lequel se trouve le processus de traitement du protocole (émission ou réception).

Comme il a été déjà mentionné dans le paragraphe 3.1 de ce chapitre, il n'y a pas d'étape d'ordonnancement spécifique dans la mesure où l'application est déjà naturellement ordonnée.

Ainsi, plusieurs étapes de transformation de la description initiale sont appliquées pour aboutir à une description structurelle implémentable suivant le modèle d'architecture déjà établi.

4.1 Traitement des instructions

Les différents signaux de contrôle qui doivent être générés à partir de la partie contrôle dépendent étroitement des différentes instructions s'exécutant dans chaque état. Dans la représentation précédente, ces instructions sont représentées horizontalement selon l'état associé. Alors, plusieurs instructions de même nature dans les différents états peuvent être utilisées.

Afin de réaliser le regroupement des différentes instructions pour générer les différents signaux correspondant à chacune d'elles, toute instruction de même type est reportée verticalement en précisant l'état dans laquelle elle est activée et les caractéristiques de son activation (conditions et paramètres).

Ainsi, on obtient une nouvelle représentation qui est formée de plusieurs colonnes dont chacune correspond à une instruction et dont les éléments représentent les différents états dans lesquels elle est déclenchée (figure 7).

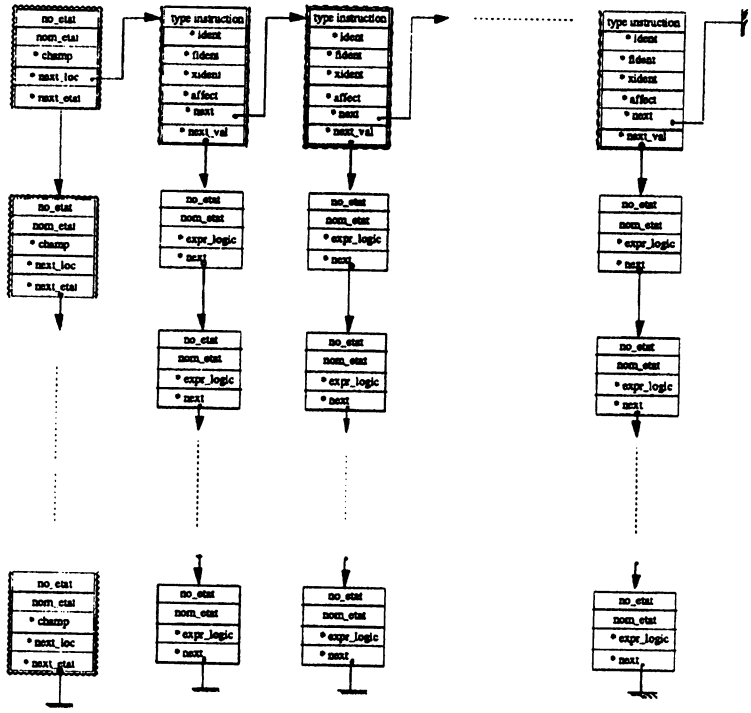


figure 7 : schéma de la représentation verticale

Ainsi, les éléments indiquant la présence d'une instruction dans un état donné sont représentés par la figure 8.

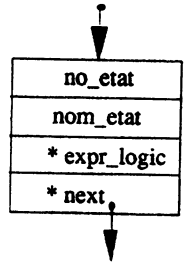


figure 8 : représentation de la présence d'une instruction dans un état

Dans ce cas, chaque élément de la colonne contient le nom de l'état et la condition de son activation.

Dans cette première transformation, les différentes instructions redondantes dans un état sont éliminées et des simplifications sont réalisées sur les conditions d'activation.

Selon les différents types d'instructions, on peut distinguer trois autres traitements:

. traitement des conditions (ou des expressions logiques),
. traitement des valeurs de chaque paramètre,
. traitement des paramètres de l'instruction (pour éviter les confusions du langage, on appelera ces paramètres sous le vocable "*port*").

Dans le cas d'une instruction *init_all* par exemple, seulement le traitement des conditions est effectué car cette instruction ne possède pas de ports et par conséquent pas de paramètres. Alors que dans le cas de l'instruction *Sendf*, deux traitements sont nécessaires: traitement des conditions et le traitement des valeurs du paramètre.

4.2 Traitement des expressions logiques

L'activation de chaque instruction dans la deuxième représentation dépend de la condition associée (expression logique). De la même manière que pour le traitement des instructions, la sélection d'une expression logique donnée dépend de l'état dans lequel la condition est validée. Ainsi, toutes les expressions logiques de chaque instruction et qui sont de même type sont regroupées sur une même colonne. Chaque élément de la colonne indique alors le nom de l'état de validation correspondante.

Dans la figure 9, les différentes conditions sont sélectionnées en utilisant un multiplexage des différentes expressions logiques.

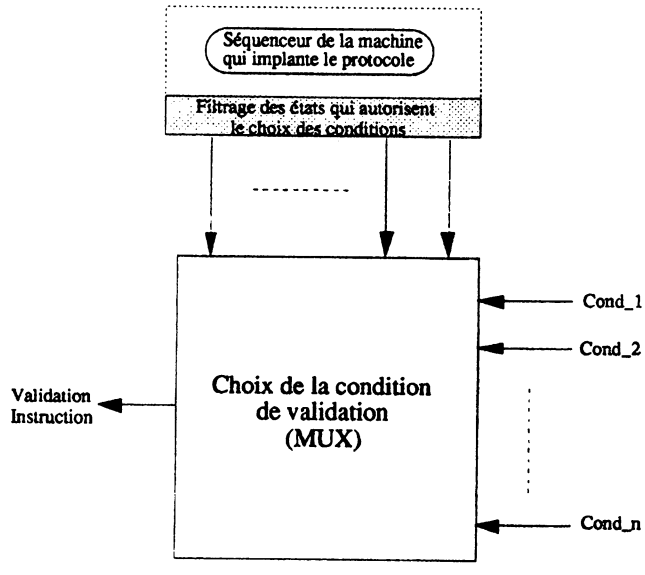


figure 9 : sélection des expressions logiques

La structure de données qui résulte de la deuxième transformation est présentée dans la figure 10 pour le cas d'une seule instruction.

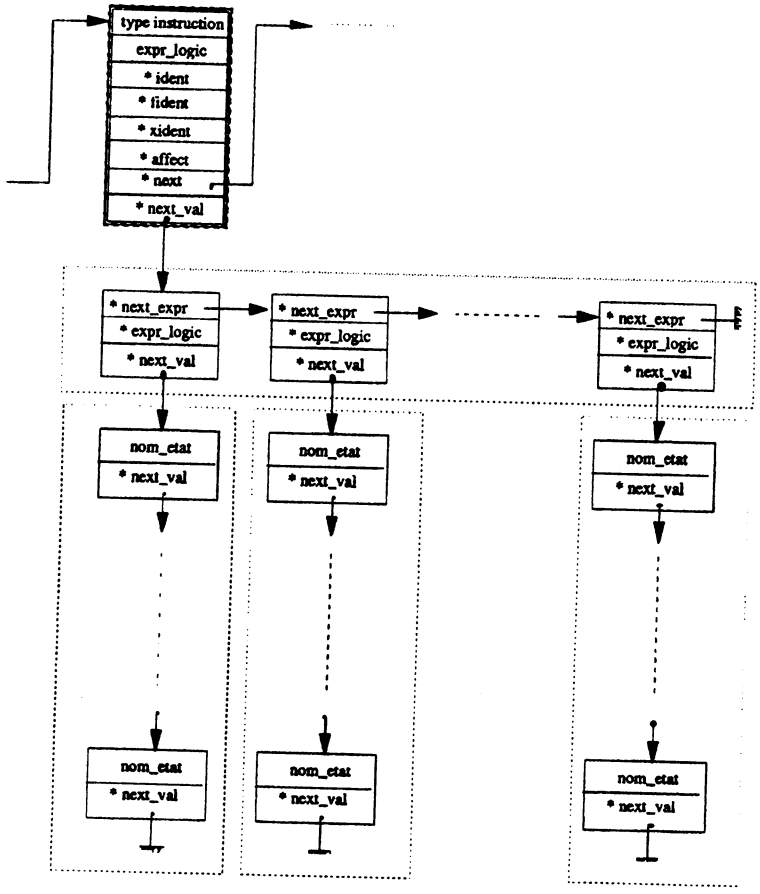


figure 10 : structure de données pour la sélection des conditions

4.3 Traitement des paramètres

Dans le cas d'une instruction qui admet un ou plusieurs ports, le choix de la valeur d'un paramètre pour un port donné dépend du résultat de l'activation associée.

Dans la figure 11, les différentes valeurs des paramètres sont sélectionnées en utilisant un multiplexage selon les résultats des expressions logiques correspondantes. Le multiplexage des expressions logiques précédent (4.2) est utilisé pour la sélection de la condition associée.

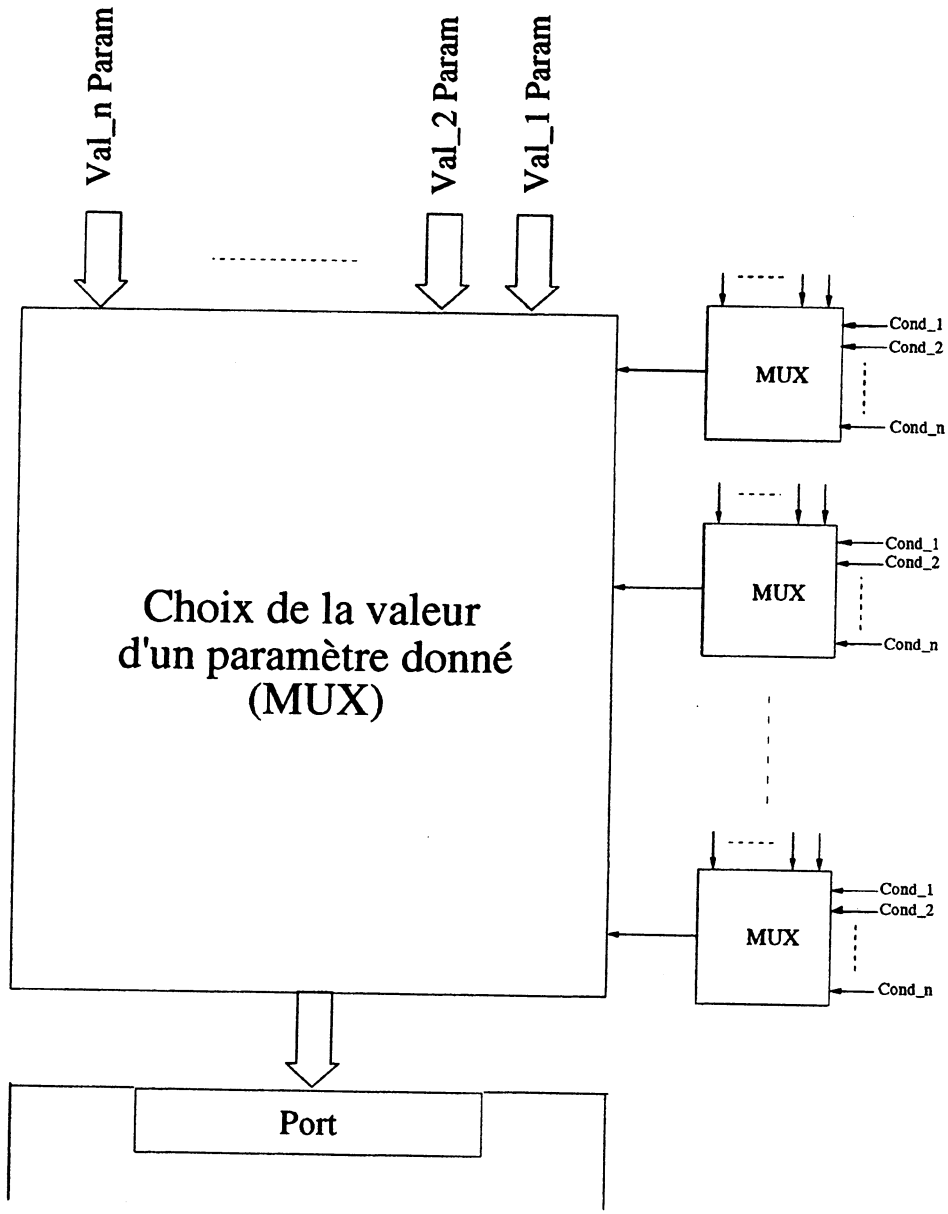


figure 11 : sélection des paramètres

La structure de données correspondante à une instruction possédant un port est représentée dans la figure suivante:

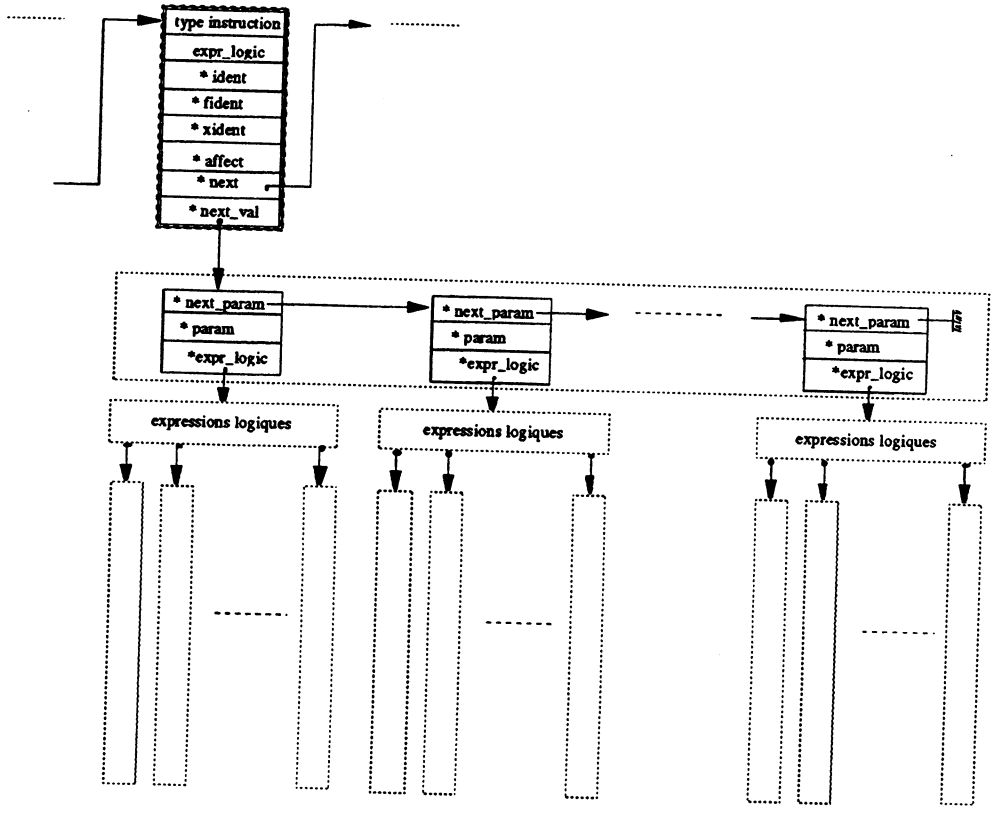


figure 12 : structure de données de sélection de paramètres

4.4 Traitement des ports

Dans le cas d'une instruction à plusieurs ports, le même traitement que précédemment est utilisé pour résoudre le problème de la sélection des différents paramètres pour chaque port de l'instruction.

On obtient ainsi le schéma global d'implémentation présenté dans la figure suivante:

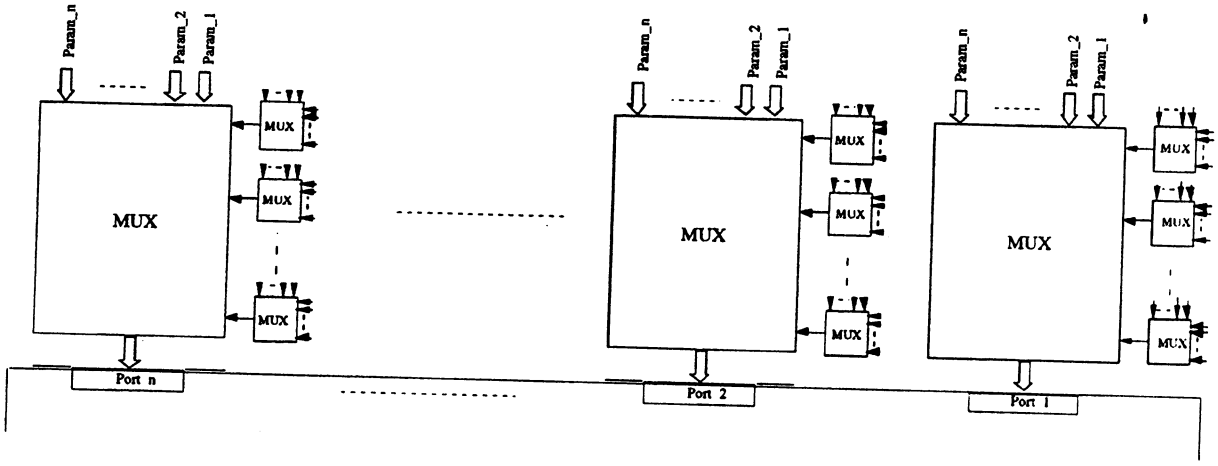


figure 13 : instruction à plusieurs ports

La représentation interne d'une instruction à plusieurs ports est illustrée par la structure de données suivante:

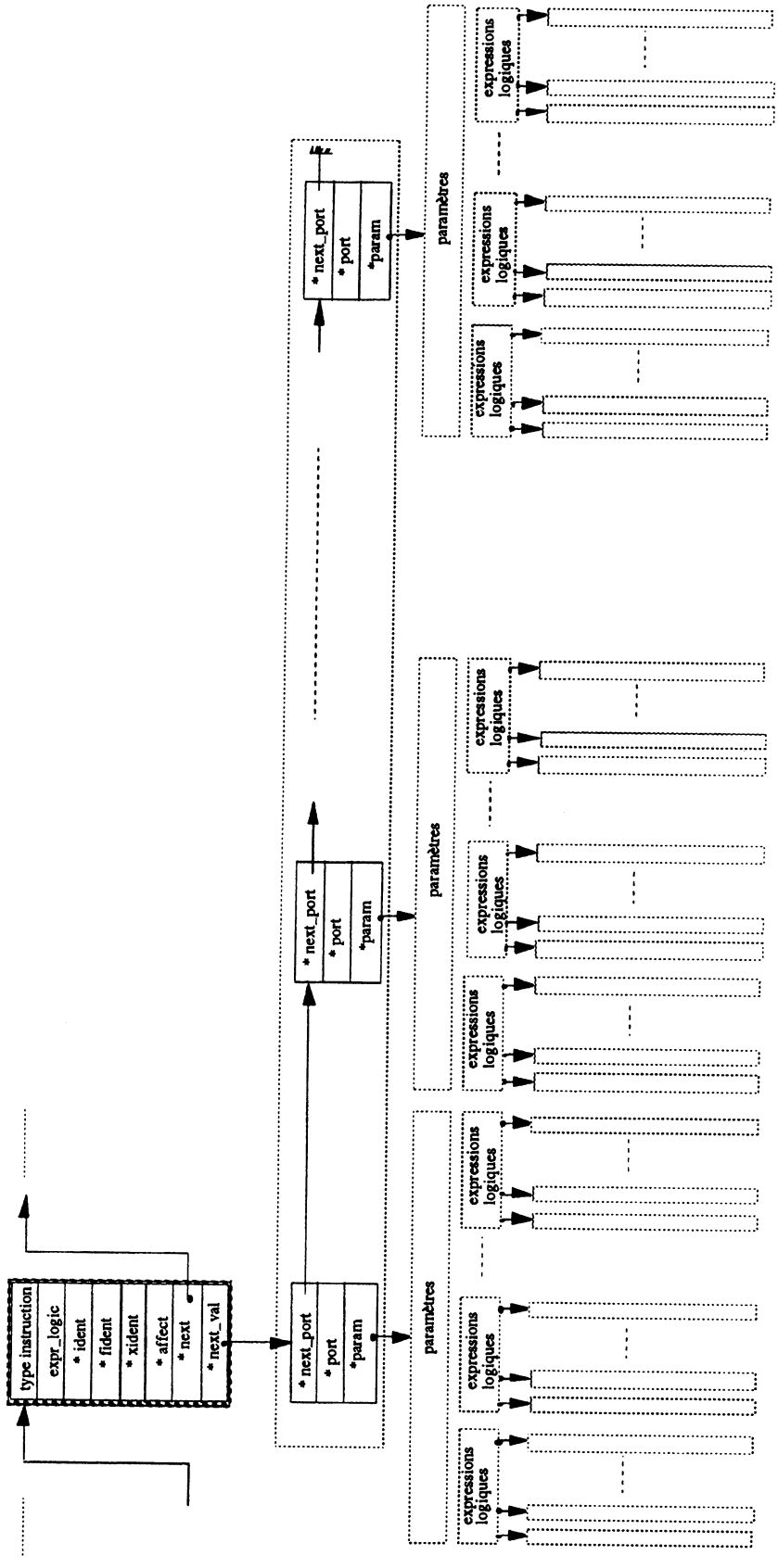


figure 14 : structure de données
pour les instructions à plusieurs ports

4.5 Traitements Spécifiques des Etats Intermédiaires

4.5.1 Largeur d'un Champ

Si pour un état donné, la largeur d'un champ dépasse la largeur mot de l'unité de mémorisation de la partie bufferisation, que ce soit en réception ou en émission, des états intermédiaires sont automatiquement rajoutés pour prendre en compte le début et la fin du champ.

Les mêmes instructions doivent être appliquées dans les états intermédiaires qui sont également ajoutés. La fin de traitement du champ en question dépend de sa largeur et de la largeur du mot du buffer.

Un état intermédiaire est alors rajouté pour traiter le reste des *eb* lorsque la largeur du champ n'est pas un multiple de la largeur du buffer.

4.5.2 Instruction *NextState*

Dans le cas où le branchement vers un état donné où le champ associé est de taille de un *eb* ou de type différent, des informations concernant l'état de destination doivent être prises en compte pour anticiper les traitements critiques, ceci suivant le modèle d'architecture choisi.

4.5.3 Activation de Processus par les instructions *Emit et Recept*

L'activation des processus d'émission ou de réception par les instructions *Emit* et *Recept* nécessite de prendre en compte le positionnement au préalable des paramètres d'initialisation en rajoutant systématiquement un état intermédiaire.

4.6 Schéma Global de la connectique de sélection

Selon le type d'instruction, différentes représentations internes peuvent être construites comme on a pu observer dans les sous-chapitres 4.1, 4.2, 4.3 et 4.4.

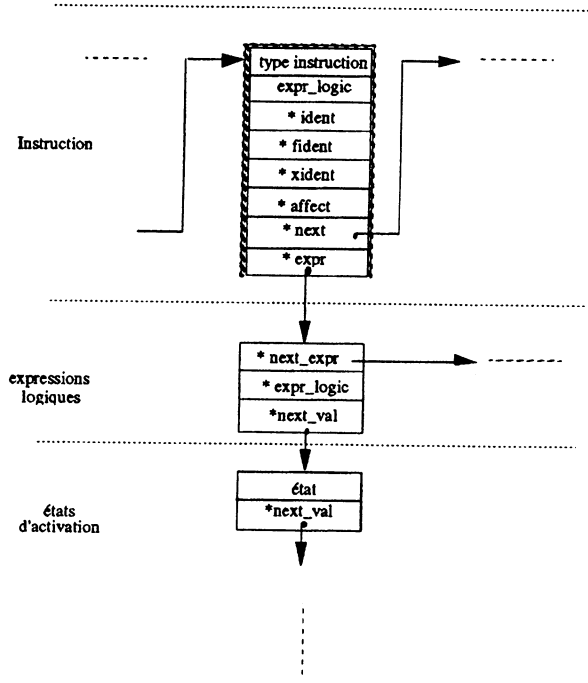
Ainsi on peut classer les différentes instructions en trois groupes de représentation suivant la présence ou l'absence des ports de paramètres:

(1) Instructions Init_loc, Init_all

Du fait que ce groupe d'instructions ne possède pas de ports, la représentation associée consiste seulement à définir le type de l'instruction et les conditions (expressions logiques) de validation.

La structure de donnée suivante présente alors trois niveaux de représentation:

(a) le type d'instruction
(b) les conditions de validation
(c) les états de validation



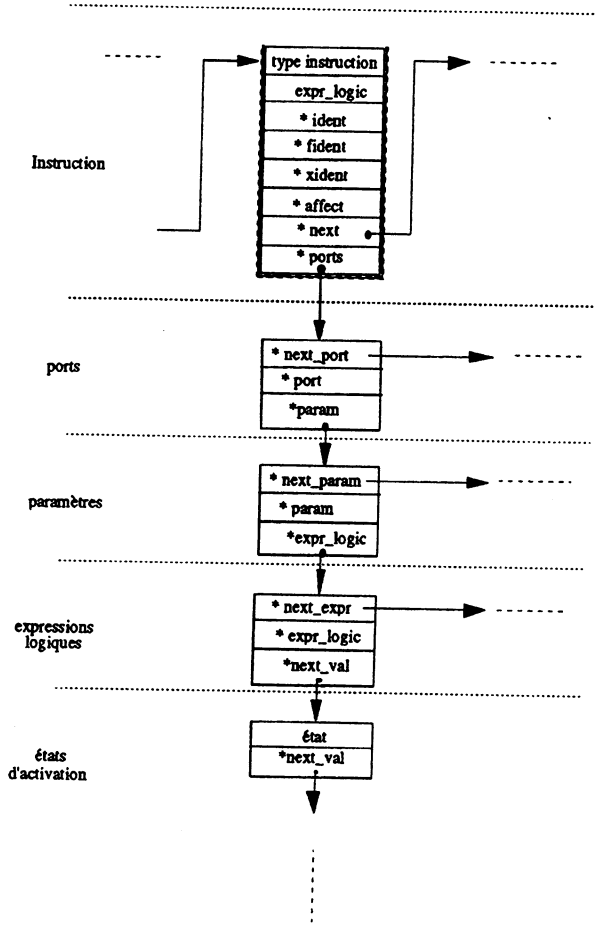
(2) Instructions *Sendf, Receivef, Emit, Recept, NextState*

Dans ce groupe, les instructions ne possèdent qu'un seul port de paramètres, la représentation associée consiste à définir le type d'instruction, les différentes valeurs du paramètre et les conditions (expressions logiques) de validation.

La structure de donnée suivante présente alors quatre niveaux de représentation:

(a) le type d'instruction
(b) les différentes valeurs du paramètre

(c) les conditions de validation
(d) les états de validation



(3) Instruction *Fonct_Unit*

Dans ce groupe, les instructions peuvent avoir plusieurs ports, alors la représentation associée consiste à définir le type de l'instruction, les différents ports, les différentes valeurs des paramètres et les conditions (expressions logiques) de validation.

La structure de donnée suivante présente alors cinq niveaux de représentation:

(a) le type d'instruction
(b) les différents ports
(c) les différentes valeurs des paramètres
(d) les conditions de validation
(e) les états de validation

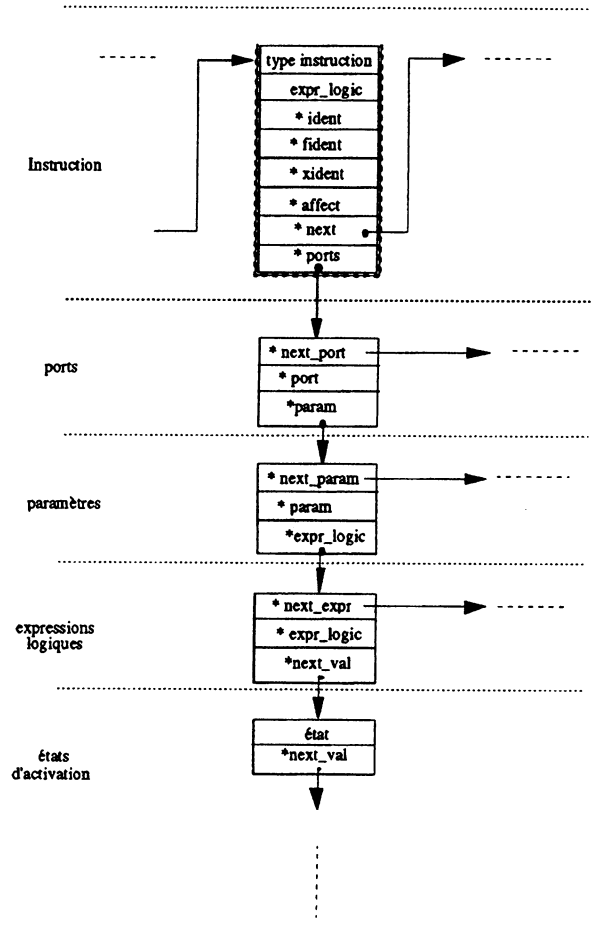


figure 15 : structure de donnée globale

Après les diverses étapes précédentes de transformation, on obtient alors une représentation d'implémentation correspondant au modèle architectural déjà présenté. La génération structurelle de l'architecture consiste en une allocation des différents éléments structurels pour implémenter les différentes parties de l'architecture.

Ces parties consistent alors en la génération des parties contrôles des machines d'émission et de réception, des différents multiplexeurs et des fonctions logiques.

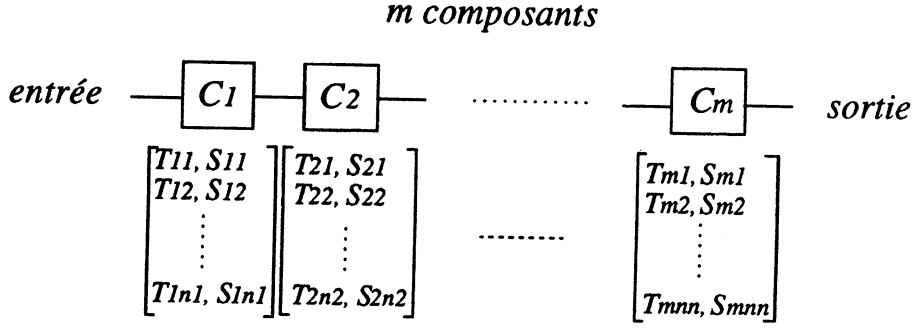
5 Allocation Automatique

Le processus d'allocation effectue les divers affectations des éléments structurels d'une bibliothèque aux éléments architecturaux du circuit. Il consiste alors à choisir et mettre à la place d'une fonction architecturale l'élément structurel correspondant.

L'optimisation du choix des éléments structurels dépend de deux paramètres : la surface totale et le temps de réponse du composant choisi.

Il s'agit alors de minimiser la surface globale sur un ensemble de composants constituant un chemin tout en respectant une contrainte de temps donnée.

Soient m composants dans un chemin, et chaque composant peut être choisi parmi n_i dans une bibliothèque d'éléments structurels :



avec :

Tij : le temps de réponse du composant Ci
Si : la surface du composant Ci

On a alors :

$$\begin{cases} \sum_{j=1}^{ni} Vij = 1 \\ Vij = \{0, 1\} \end{cases}$$

Minimiser :

$$\sum_{i=1}^{m} \sum_{j=1}^{ni} Vij \,.\, Sij$$

avec la Contrainte :

$$\sum_{i=1}^{m} \sum_{j=1}^{ni} Vij \,.\, Tij \leq T \text{ donné}$$

La résolution de ce problème peut être réalisée en utilisant le logiciel du commerce CPLEX [CPL89] qui permet de résoudre les différents problèmes linéaires.

Les différents éléments structurels (opératifs) d'un circuit de communication sont relativement traditionnels et que la hiérarchie de ces divers éléments est connue et considérée par hypothèse dans ce travail comme existante. Elle est initialement déduite d'une phase d'analyse descendante du protocole de communication et d'une construction ascendante de l'architecture [Dan 86] [Dan 87].

Le sous-chapitre suivant décrit cette hiérarchie. Chaque niveau est spécifié en présentant d'une part ses éléments structurels et d'autre part, les outils de conception associés.

6 Schéma Général du Système de Conception

On présente dans cette partie, d'une part l'organisation générale de tous les éléments structurels d'une architecture sous forme de plusieurs niveaux d'une hiérarchie de composants, et d'autre part la démarche de conception pour aboutir à une implémentation physique ("layout") du circuit en utilisant les outils de CAO déjà existant.

6.1 Hiérarchie des Différents Eléments Structurels

Les différents composants constituant l'architecture peuvent être répartis sur trois classes :

- classe de composants de base,
- classe de composants à usage général
- et classe de composants spécifiques.

Ils constituent ainsi une bibliothèque hiérarchisée en trois niveaux dont on peut rajouter un quatrième qui constituera alors l'architecture du circuit.

(1) Premier niveau

Le premier niveau est considéré comme étant la couche de base de tout système de CAO. Il est composé d'une bibliothèque de cellules de base qui sont de trois types:

- les cellules de type porte logique telles que les portes NOT, NOR, NAND, OR, AND, XOR, ... qui peuvent être à une ou plusieurs entrées;

. les cellules de type bascule telles que les bascules RS, D, JK, ... dont l'activation peut être sur front ou sur niveau (haut ou bas);

. les cellules spécifiques utilisées par des générateurs pour la construction des modules de structure régulière telle que les RAM, ROM, PLA, UAL, ... Ces cellules ne sont pas utilisées directement par l'utilisateur, il doit pour cela faire appel aux générateurs correspondant en précisant les paramètres indiquant par exemple la taille et le temps de réponse voulus.

(2) Deuxième niveau

Les composants de ce niveau admettent généralement une structure régulière. Leur utilisation peut être immédiate en choisissant le composant directement de la bibliothèque ou à travers l'utilisation de programmes appelés générateurs. Ces composants peuvent être de deux types:

. Les structures régulières matricielles telles que la RAM, la ROM et le PLA. Leur utilisation se réalise par un appel au générateur associé en précisant quelques paramètres de caractérisation de forme et de fonctionnement.

. Les structures régulières sont des opérateurs tels que les additionneurs/ soustracteurs, les compteurs et les comparateurs. Ces composants peuvent être construits à partir des éléments du premier niveau. Ce sont généralement des composants fournis en terme de tranches de bits monolithiques dont l'utilisation est laissée à la disposition du concepteur pour le choix du nombre de boîtiers (tranches) et de leur connexion.

(3) Troisième niveau

Ce niveau est considéré comme étant le niveau dédié au domaine d'application qui est dans notre cas celui des circuits de communication. Les composants de ce niveau sont les différents modules spécifiques qu'on rencontre souvent dans un circuit de communication tels que le FIFO, le sérialiseur/désérialiseur, le module de traitement de CRC, le module de reconnaissance d'adresse, ...

Ces composants spécifiques sont construits en utilisant généralement les éléments du premier et du deuxième niveau [Sab 92].

(4) Dernier niveau

Ce niveau est en réalité le résultat de l'assemblage des différents éléments des niveaux précédents pour bâtir l'architecture finale d'un circuit de communication.

La figure 16 présente le schéma global illustrant la hiérarchie des différents éléments structurels que peut contenir une architecture de circuit de communication.

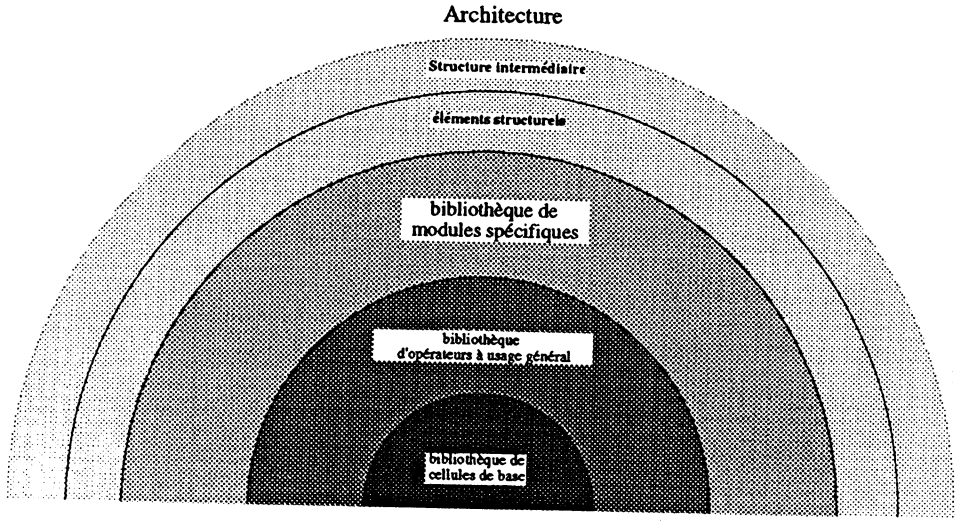


figure 16: Hiérarchie de composants

6.2 De la Description Structurelle au Layout

L'instanciation des différents éléments structurels à partir de la bibilothèque de composants est réalisée selon la bibliothèque cible et le système de CAO choisi. L'architecture obtenue est constituée d'un ensemble de composants interconnectés et appartenant aux différents niveaux présentés précédemment.

Selon l'outil d'implémentation choisi, on peut générer en définitive l'architecture en utilisant les primitives d'instanciation et d'interconnexion de l'outil à travers soit un langage de description de matériel (HDL) tel que VHDL ou à travers l'utilisation des commandes interactives de la saisie schématique tel que Solo 2030.

Les différents outils correspondant aux différents niveaux de la hiérarchie des composants sont:

. Des outils de placement, d'interconnection et de simulation sont fournis par le système de base correspondant au premier niveau.

. Des générateurs de structures régulières à usage général sont aussi fournis par le système de base. Ces générateurs peuvent être programmés moyennant l'existence d'un langage de programmation fourni par le système.

. Des générateurs de modules spécifiques sont construits au préalable pour le système de CAO dédié au domaine des circuits de communication.

. Des outils de description d'un protocole et de synthèse d'architecture dédiée constituent un outil de conception complet pour faciliter au concepteur la spécification et l'implémentation des protocoles de communication.

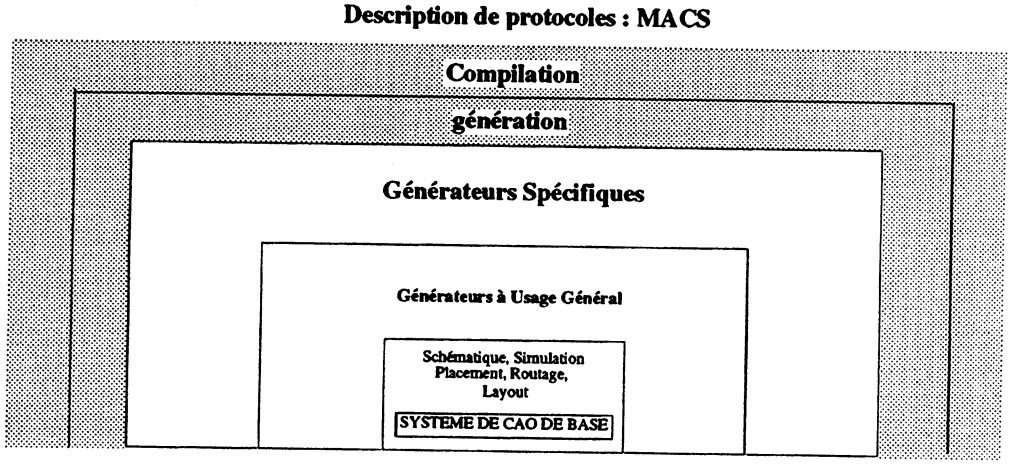


figure 17 : hiérarchie d'outils de CAO

Dans notre approche, le résultat de la synthèse est une description structurelle de l'architecture qui peut être sous forme de deux langages de description:

(a) Description en langage VHDL [Air90]

Ceci permet de prototyper rapidement le circuit afin de l'intégrer dans un système plus complet et de réaliser par exemple des simulations comportementales et des évaluations de performance de tout l'environnement du circuit.

D'autre part, une description structurelle en VHDL est très compréhensible dans le monde de la conception de circuit et implémentable dans beaucoup de systèmes de conception.

(b) Description en SKILL

L'architecture est décrite dans ce cas directement dans le langage de description spécifique au système de CAO Solo 2030 appelé SKILL [CAD89]. Ceci permet d'effectuer des simulations en utilisant une bibliothèque de composants d'une technologie donnée, de faire un placement routage pour produire un layout et de réaliser le circuit.

6.3 Schéma Général du Système de Conception

Le schéma général de conception peut être défini en présentant les deux produits qui sont souvent manipulés : les bibliothèques de composants et les outils de conception.

La figure 18 présente la hiérarchie des outils de conception suivant la hiérarchie des bibliothèques de composants.

A partir de la spécification du protocole de communication par une description spécifique MACS, une architecture est synthétisée selon le modèle d'architecture déjà présenté. Elle est composée d'un ensemble de modules spécifiques interconnectés qui sont construits par des générateurs spécifiques aux circuits de communication. Ces derniers utilisent des cellules et des composants de base existant dans le système de conception cible.

Ainsi, une "netlist" de l'architecture est enfin produite. Des outils de placement et routage sont ensuite utilisés pour obtenir finalement le layout du circuit de communication.

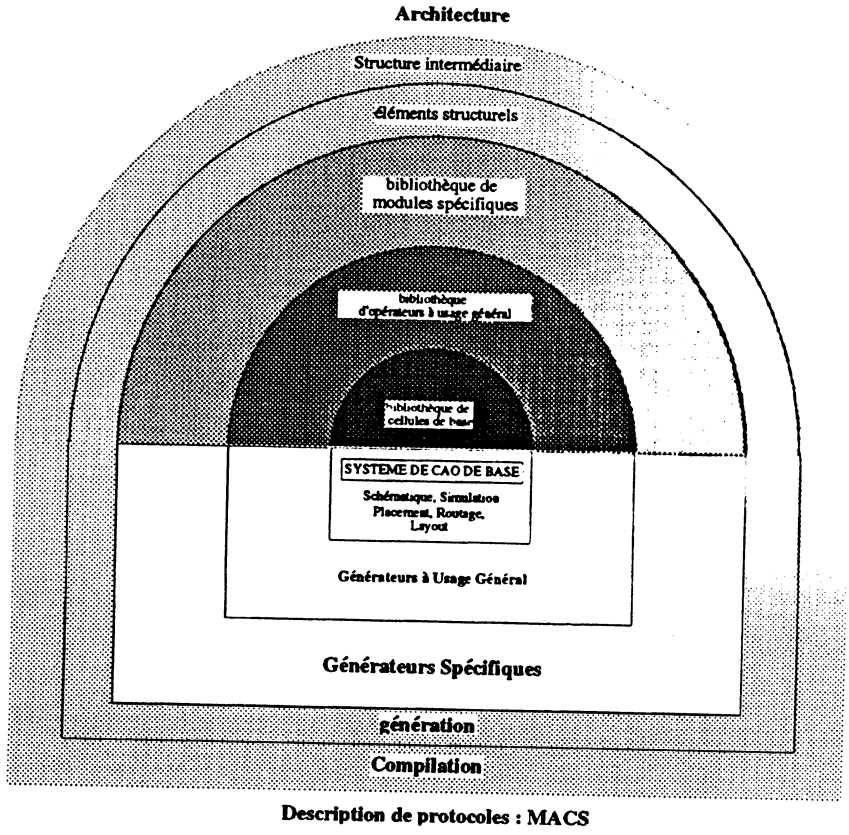


figure 18: organisation générale du système de conception

Conclusion

L'utilisation d'un langage de description de protocole de communication en vue d'implémentation et d'une architecture dédiée à un domaine d'application (contrôle-commande) permet le prototypage rapide d'un circuit de communication. Une démarche de synthèse a été présentée afin d'illustrer les différentes étapes de traitement systématique d'implémentation.

Le langage MACS composé d'un ensemble d'instructions permet de décrire les différentes composantes d'une application. Parmi ces composantes, nous avons la description des interfaces, les déclarations des objets de communication du protocole et la description des processus d'émission et de réception.

Le schéma global suivi est représenté d'une manière hiérarchique en partant des éléments de base d'une bibliothèque pour aboutir à l'architecture compète du circuit de communication. Ainsi, l'implémentation du circuit dans un environnement de communication permet d'effectuer des simulations comportementales de tout le système.

Chapitre 4

APPLICATIONS

Chapitre 4

— Applications —

Introduction

Dans ce chapitre, nous présentons les premières applications permettant de valider les différentes possibilités de notre approche de conception d'un circuit de communication. Il ne s'agit pas ici de faire une présentation exhaustive des applications mais surtout de donner un aperçu de la démarche de conception en partant d'une spécification donnée sous forme d'un cahier des charges pour arriver à une implémentation conforme.

Nous commençons d'abord par présenter une partie application générale en spécifiant la description de l'environnement d'implémentation d'une part et d'autre part les différents cas de formats de trames qui peuvent être généralement rencontrés dans les protocoles de communication existants.

La deuxième partie consiste à présenter des applications spécifiques en décrivant quelques protocoles de communication industriels tels que les protocoles utilisés dans le domaine des réseaux intra-véhicule.

1 Application Générale avec des exemples génériques

1.1 Environnement d'Implémentation

Il s'agit de la spécification de l'interface avec la station afin de permettre à l'application de dialoguer avec le circuit. Comme indiqué dans le chapitre 3, l'interface est composée d'une partie adaptation et d'une partie bufferisation.

1.1.1 Adaptation des Signaux

Les signaux de l'interface externe sont souvent regroupés en un seul bus de communication qui est associé à un bus de données. Ils permettent de véhiculer des informations de contrôle, d'état et de données. Ainsi, on peut spécifier alors plusieurs bus de communication possédant chacun ses propres signaux d'échange et de contrôle.

(a) Les Signaux dans un Bus de Communication

L'échange de données entre l'application et le circuit de communication est généralement réalisé à travers un bus de signaux de données et de signaux de contrôle. Ces derniers permettent d'exécuter des échanges de type poignées de main (*HandShake*).

La description MACS d'un bus de données d'émission et un bus de données de réception avec des signaux de contrôle pour chacun se présente d'abord par la description des différents signaux composant cet interface.

```
Adaptation
begin
    # bus d'entée #
    bus_entree [8]      : data in ;
    Rdy_e               : in ctrl;
    Ack_e               : out ctrl;
    # bus de sortie #
```

```
        bus_sortie [8]      : out data;
        Rdy_s               : in ctrl;
        Ack_s               : out ctrl;
end
```

(b) Cas de Plusieurs Bus Externes

Lorsque plusieurs bus de communication indépendant sont spécifiés, le même schéma que précédemment se répète pour la description et l'implémentation au niveau circuit de la partie adaptation de l'interface.

1.1.2 Bufferisation des données

Cette partie contient uniquement la spécification des différents éléments de mémorisation servant à stocker les informations qui doivent être échangées entre le circuit et l'application. Ces informations sont regroupées en trois types : les données, les commandes et les comptes rendus.

Comme nous l'avons déjà indiqué dans le chapitre 3, les unités de mémorisation sont de deux types: Registre ou Fifo. Ces éléments sont souvent classés en deux catégories: Emission et Réception.

- En Emission

Dans ce cas, ces unités sont considérées comme étant des sources d'informations pour le circuit de communication. On peut avoir une ou plusieurs sources de données.

(a) Une seule Source de données

Dans ce cas , une seule unité est utilisée pour stocker les données à émettre sur le réseau. L'application adresse alors cette unité à chaque fois qu'elle envoie des données dans le réseau.

(b) Plusieurs Sources de données

Si plusieurs unités sont utilisées pour l'émission des données, l'application doit adresser l'unité correspondante en utilisant dans ce cas un ensemble de signaux (bus d'adresse) permettant de sélectionner chacune d'elles.

- En Réception

Contrairement au cas de l'émission, les unités en réception sont considérées comme des puits d'informations pour le circuit de communication. On peut avoir un ou plusieurs puits de données.

On utilise également un bus d'adresse pour désigner une unité donnée dans le cas où l'on a plusieurs puits de données.

1.1.3 Accès aux Unités de Bufferisation

Les fonctions de lecture et d'écriture dans les éléments de mémorisation sont réalisées en utilisant des primitives "INPUT" et "OUTPUT" en indiquant l'unité concernée et la logique de validation.

Ainsi, à l'aide de ces deux primitives, on peut effectuer des lectures et des écriture sur un même bus de donnée externe en spécifiant à chaque fois le même bus de communication.

1.1.4 Exemple d'Interface Pratique

Pour illustrer avec un cas pratique la spécification d'un interface, deux exemples d'interface microprocesseur sont choisis : Motorola et Intel.

A Interface Motorola

A.1 Spécification du Bus Externe

L'interface Motorola est constitué d'un bus d'adresse, d'un bus de données et d'un ensemble de lignes de contrôle. La taille du bus d'adresse dépend du nombre d'unités de mémorisation (registres et Fifos) internes.

Le bus de contrôle est composé :

. d'une ligne R/W indiquant qu'il est en mode de lecture ou d'écriture,
. d'une ou de plusieurs lignes CSs permettant la sélection du circuit,
. d'une ligne d'initialisation du circuit Reset,
. et d'une ligne E pour la validation du cycle d'opération de lecture et d'écriture.

Le schéma suivant montre les différents signaux d'un exemple de bus de communication Motorola:

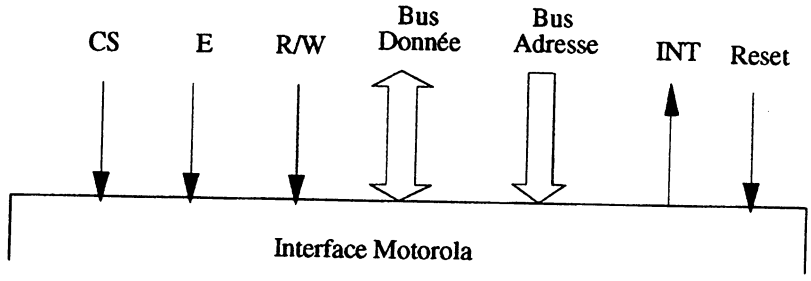


figure 1: Bus de communication Motorola

A.2 Spécification des Unités de Mémorisation

Prenons l'exemple simple d'un interface composé :

. d'une FIFO d'émission de données application,
. d'une FIFO de réception de données application,
. d'un registre de commande,
. de deux registres contenant l'adresse de destination,
. . et de deux registres d'état.

Alors, le schéma global de l'interface est le suivant :

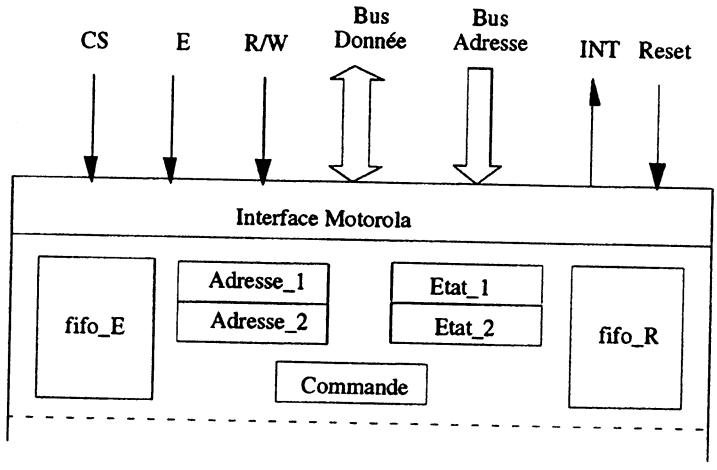


figure 2: exemple d'interface

A.3 Description MACS de l'Interface

L'adressage et la logique de sélection des différents registres et fifos est présentée par le tableau suivant:

	RS0	RS1	Lecture	Ecriture
S0	0	0	fifo_R	fifo_E
S1	0	1	Etat_1	Commande
S2	1	0	Etat_2	Adresse_1
S3	1	1		Adresse_2

Les signaux S0, S1, S2 et S3 sont les signaux de sélection des différents registres et fifos. RS0 et RS1 représentent dans cet exemple, le bus d'adresse externe.

La description MACS de l'interface consiste à décrire les différents bus externes, les différents éléments de mémorisation et les chemins d'accès correspondants c'est-à-dire la logique de sélection des registres et des fifos.

A.4 Schéma Bloc de l'Implémentation

L'architecture qui découle de la spécification après le processus de synthèse est présentée dans la figure 3. Elle est composée d'un ensemble de multiplexeurs pour le choix entre les différentes sorties des éléments de mémorisation et d'un décodeur permettant de sélectionner un de ces éléments.

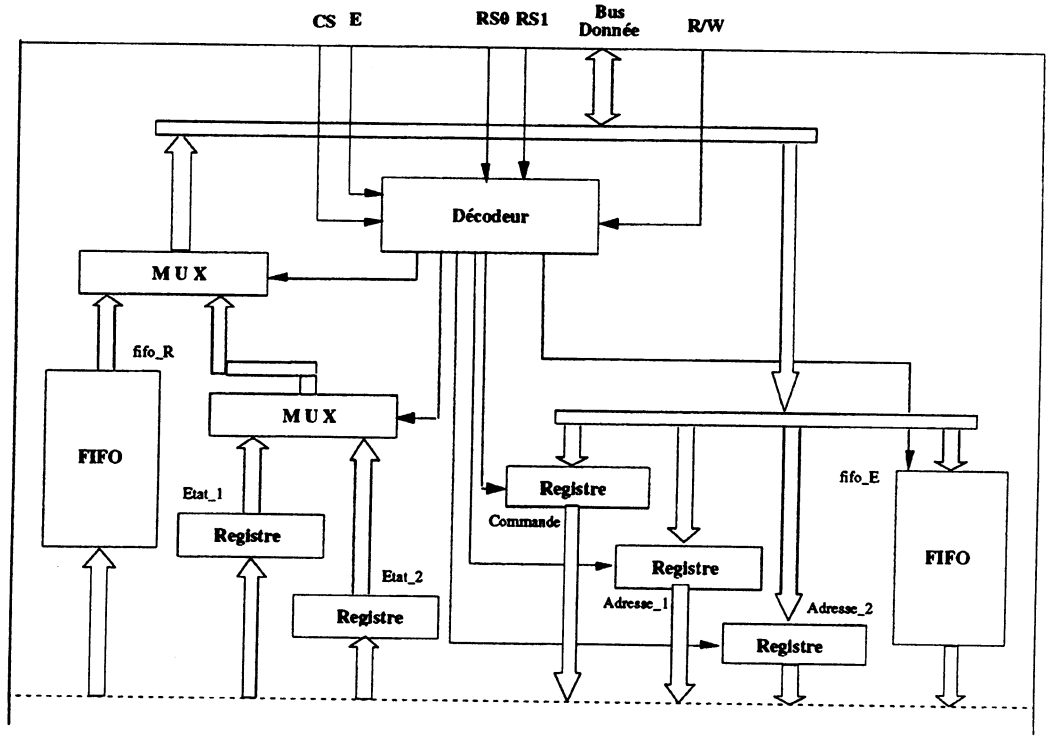


figure 3: Architecture de l'interface

B Interface Intel

B.1 Spécification du Bus

Le bus d'adresse et le bus de données dans une interface Intel sont multiplexés.

Le bus de contrôle est composé :

- d'une ligne de validation d'adresse ALE,
- d'une ligne RD indiquant la lecture de données,
- d'une ligne WR indiquant l'écriture de données,
- d'une ou de plusieurs lignes CSs permettant la selection du circuit,
- d'une ligne d'initialisation du circuit Reset,
- et d'une ligne indiquant la présence d'une interruption INT.

Le schéma suivant montre les différents signaux du bus de communication Intel:

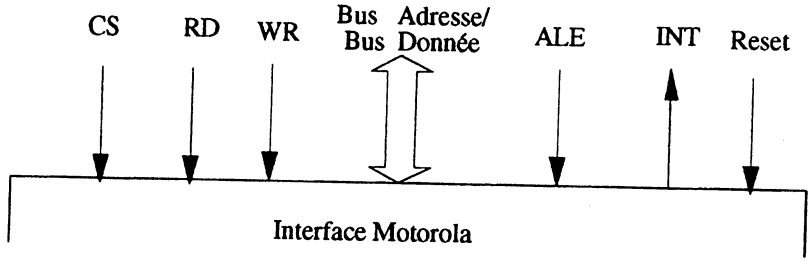


figure 4: Bus de communication Intel

B.2 Spécification des Unités de Mémorisation

Soit un exemple d'interface composé de:

- deux FIFOs de réception,
- une fifo d'émission
- deux registres d'état,
- deux registres d'adresse,
- et un registre de commande.

Alors, le schéma global de l'interface est le suivant :

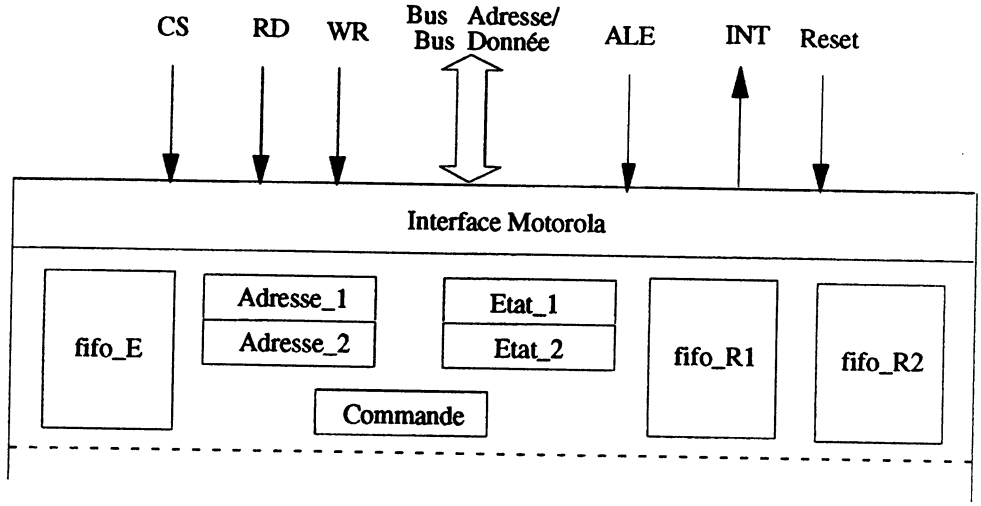


figure 5: exemple d'interface Intel

B.3 Description MACS de l'Interface

L'adressage et la logique de sélection des différents registres et fifos est présentée par le tableau suivant:

	A0	A1	Lecture (RD)	Ecriture (WR)
S0	0	0	fifo_R1	fifo_E
S1	0	1	fifo_R2	Commande
S2	1	0	Etat_1	Adresse_1
S3	1	1	Etat_2	Adresse_2

Les signaux S0, S1, S2 et S3 sont les signaux de sélection des différents registres et fifos. RS0 et RS1 représentent dans cet exemple, le bus d'adresse externe. La description de cette partie est similaire à celle qui a été décrite dans le cas précédent.

B.4 Implémentation

L'architecture qui en découle de la spécification est présentée dans la figure 6.

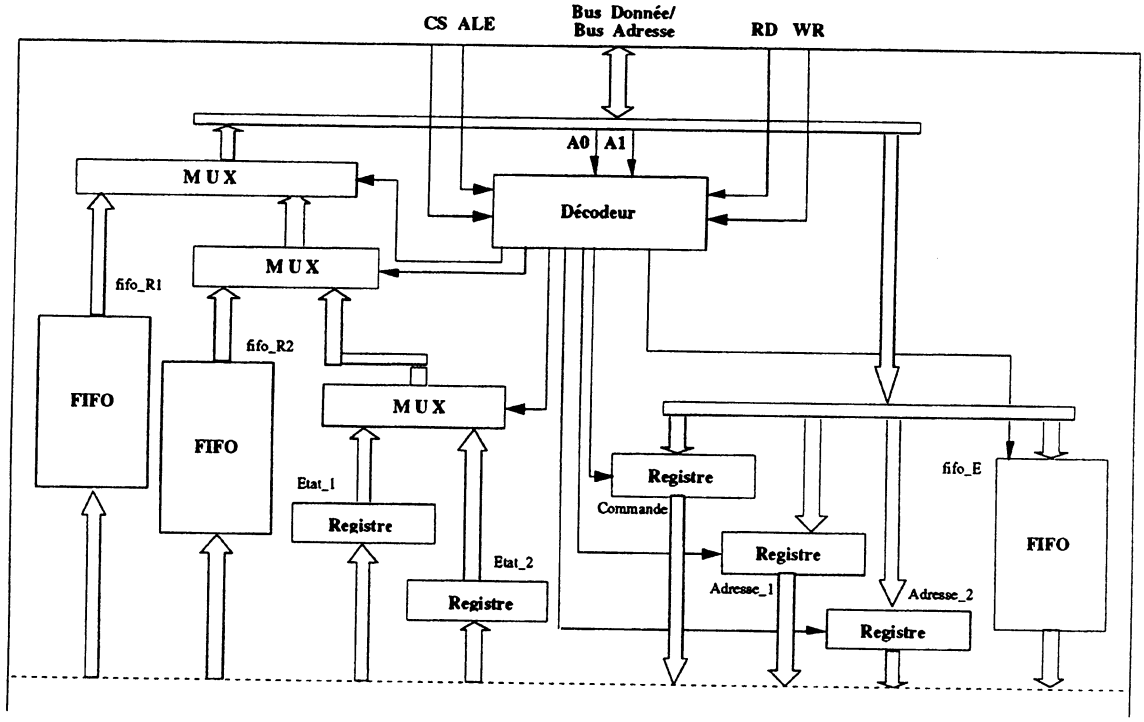


figure 6: Architecture de l'interface Intel

1.2 Trame à Structure Statique

Dans le cas général des protocoles de communication existants, la structure commune des trames peut être représentée par le format suivant:

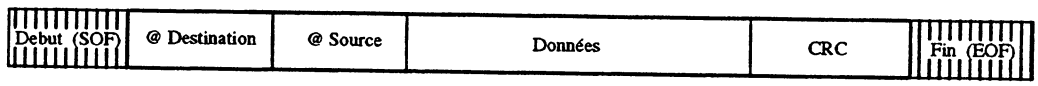


figure 7 : Format Général d'une Trame

La trame est constituée d'un champ appelé SOF (*Start Of Frame*) indiquant le début de la trame et d'un champ EOF (*End Of Frame*) pour indiquer la fin de la trame. Un champ de données appartenant à l'application est inséré avec le champ d'adresse de destination, le champ d'adresse source et un champ de contrôle d'erreur. On désigne souvent ce dernier par le champ CRC qui permet de faire un contrôle d'erreur sur les champs d'adresses et de données.

1.2.1 Types de Champs

1.2.1.1 Un Champ Statique

Dans le cas d'un champ statique c'est à dire dont la taille est constante, un champ est défini durant la description du protocole de communication. L'implémentation du traitement de ce type de champ consiste à déterminer si la taille de ce champ est plus grande que celle du buffer d'émission ou de réception.

Dans le cas où la taille du champ est plus petite ou égale à la taille du buffer, un contrôle de séquencement d'éléments binaires du champ suffit pour reconnaitre la fin de traitement de ce champ.

L'exemple des champs d'adresse source et d'adresse de destination sont des champs statiques.

Dans le cas d'un champ dont la taille est supérieure à la taille mot de son buffer, on prend en compte le nombre de mots que le champ contient et éventuellement le résidu qui est inférieur à la taille du buffer. L'exemple d'un champ de données peut avoir une taille qui peut être un multiple de la taille mot de son buffer (qui est généralement le fifo).

1.2.1.2 Un Champ Variable

Un champ de taille variable est délimité par une fonction de reconnaissance de la fin du champ spécifiée par le concepteur du protocole de communication. Le champ de donnée peut également être dans certain cas spécifié comme étant un champ variable. Un exemple de ce type de fonction est le cas d'un CRC sur l'ensemble de la trame. Il permet d'indiquer en cas de détection d'une égalité avec le polynome caractéristique (le reste), la fin de la trame.

1.2.3 Description MACS

La trame définie dans le paragraphe 1.2 est entiérement spécifiée dans le langage de description MACS. Le type et la taille de chaque champ sont spécifiés dans la partie déclaration "field". Le champ de données est choisi dans ce cas comme étant un champ de taille égale à 32 *eb*.

1.2.4 Implémentation

L'architecture du circuit de communication correspondant est composé alors de deux parties : la partie interface avec l'application et le coeur du circuit.

1.2.4.1 Interface avec l'application

On définit dans ce cas, une interface semblable à celle qui a été déjà définie dans le paragraphe 1.1. Elle sera composée alors d'une partie adaptation et d'une partie bufferisation.

1.2.4.2 Coeur du Circuit

Le coeur du circuit est composé d'une machine traitant le processus d'émission et d'une machine traitant le processus de réception. Chacune d'elles est constituée d'une partie contrôle et d'un ensemble de composants tels que :

- bloc de début d'émission et de début de réception,
- bloc d'initialisation,
- bloc de traitement CRC,
- sérialiseur/désérialiseur,
- comparateurs et portes logiques,
- multiplexage/demultiplexage des champs,
- ...

1.3 Trame à Structure Dynamique

On parlera d'une trame à structure dynamique lorsque à partir d'un champ donné de la trame, le champ suivant n'est pas unique. La figure suivante présente un exemple d'une trame dynamique:

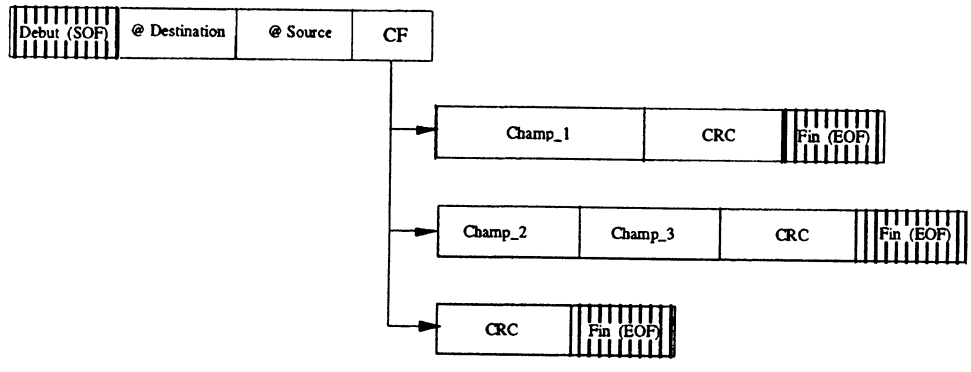


figure 8: trame à structure dynamique

Dans ce cas, le champ suivant de CF peut être Champ_1 ou Champ_2 ou le CRC selon l'état du champ CF. Ce dernier est souvent appelé champ code fonction qui représente généralement un champ de contrôle indiquant le type de

trame. Cependant, deux types de trame sont principalement utilisés dans beaucoup de protocoles de communication : Trame Question et Trame Réponse.

1.3.1 Contrôle de Séquencement des Champs

Le passage d'un champ à un autre est interprété par un branchement d'un état à un état dans la partie traitant le séquencement des champs. Ce branchement est valide si les conditions associées sont satisfaites.

1.3.2 Description MACS

L'instruction "*NextState*" permet d'exprimer le branchement sous condition pour décrire une trame à structure dynamique. Dans un état sans instruction "*NextState*", l'état suivant est par défaut l'état qui le suit dans la description.

1.3.3 Implémentation

C'est dans la partie contrôle que le contrôle de séquencement des champs est implémenté. Le cas de branchement sous condition ou inconditionnel est traité en sélectionnant les différents états suivants comme paramètres et les différentes expressions logiques comme fonction de validation permettant d'activer le branchement correspondant.

1.4 Emission avec Réponse dans la Trame

Dans certains protocoles industriels, la réponse dans la trame est employée pour répondre à un besoin de temps réel dans les échanges de données de contrôle ou d'état dans le système de communication.

La figure suivante présente l'exemple d'une émission avec une réponse dans la même trame :

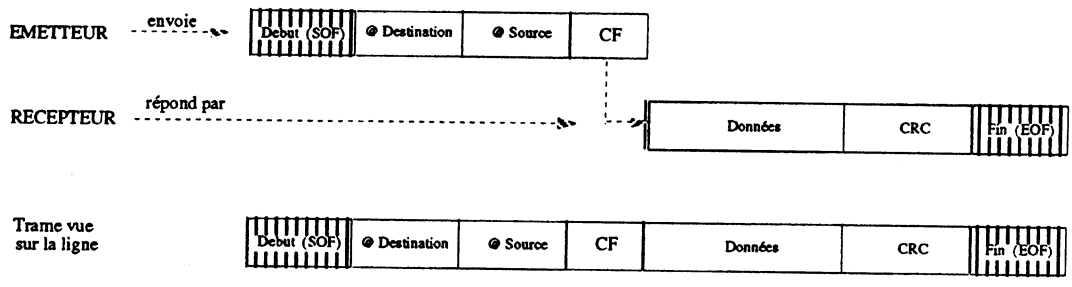


figure 9 : emission avec réponse dans la trame

Dans ce cas, le champ suivant de CF permet d'indiquer au récepteur d'envoyer les champs restants dans la même trame.

1.4.1 Contrôle de Séquencement des Champs

Le passage d'un mode d'émission en un mode de réception est réalisé en spécifiant à la machine de réception l'état de branchement et la validation de son activation. Cette dernière peut être valide si les conditions associées sont satisfaites.

1.4.2 Description MACS

Pour décrire une réponse dans la trame, les instructions "*Recept*" et "*Emit*" permettent d'exprimer le passage de l'émission vers la réception et vice-versa.

1.4.3 Implémentation

C'est dans la partie contrôle que le contrôle du passage de l'émission en réception (et réception en émission) est implémenté. Alors, un ensemble de paramètres représentant les différents états de branchement peuvent être sélectionnés. Ainsi, une ligne d'activation et un bus d'état sont les paramètres d'entrée de chaque machine.

1.5 Traitements de Champs Imbriqués

Pour montrer la possibilité de spécifier des trames avec des champs à traitement imbriqués, la figure suivante donne un exemple de deux traitements CRC, l'un portant sur toute la trame et l'autre sur une partie de la trame.

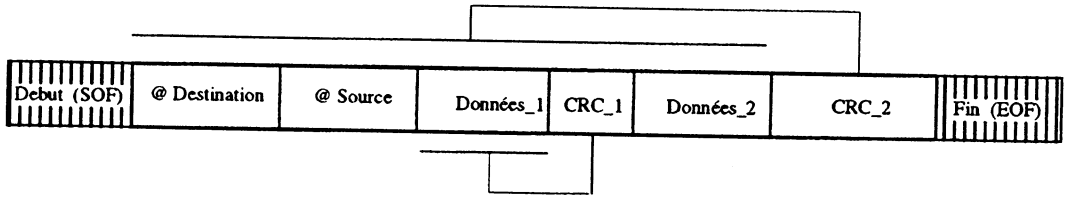


figure 10: trame à champs imbriqués

Le champ CRC1 s'applique sur le champ de données 1, par contre le champ CRC2 s'applique sur l'ensemble de la trame y compris le champ CRC1 et le champ de données 1.

1.5.1 Description MACS

Dans ce cas, tous les champs de la trame font appel au traitement CRC2. Le champs données 1 fait appel en plus du CRC2 au traitement CRC1.

1.5.2 Implémentation

Dans ce cas, deux unités de traitement CRC (CRC_1, CRC_2) sont nécessaires pour générer les champs correspondants durant l'émission et de vérifier les données durant la réception de la trame.

2 Application Spécifique — Le Protocole VAN —

Dans la deuxième partie de ce chapitre, nous présentons une application sur des protocoles de communication existants dans les réseaux locaux industriels. Nous prenons comme exemple l'implémentation du protocole de communication VAN appliqué dans les réseaux intra-véhicules car d'une part, il a été défini dans le cadre du projet PROMETHEUS et d'autre part, il présente des caractéristiques assez intéressantes du point de vue complexité fonctionnelle.

- **Le Protocole VAN**

Le VAN (Vehicul Area Network) est un protocole développé dans le cadre du projet PROMETHEUS pour les réseaux intra-véhicules par ses partenaires français (en particulier les constructeurs automobiles PSA et RNUR) [ISO90] [PSA90].
Il permet d'interconnecter des équipements dont la complexité est variée en partant des dispositifs câblés aux stations à microprocesseur. Ainsi, deux types de stations sont distingués:

- les stations autonomes : ce sont des stations maîtres capables d'effectuer un transfert d'informations sur le bus;

- les stations esclaves ne peuvent que recevoir une trame de données envoyée par une station autonome ou réaliser une réponse dans la trame.

Le VAN a un modèle de référence qui est conforme avec le modèle OSI [VAN90]. les couches physiques, MAC et LLC sont bien spécifiées, par contre les couches supérieures font partie de l'application. La figure 11 présente ce modèle.

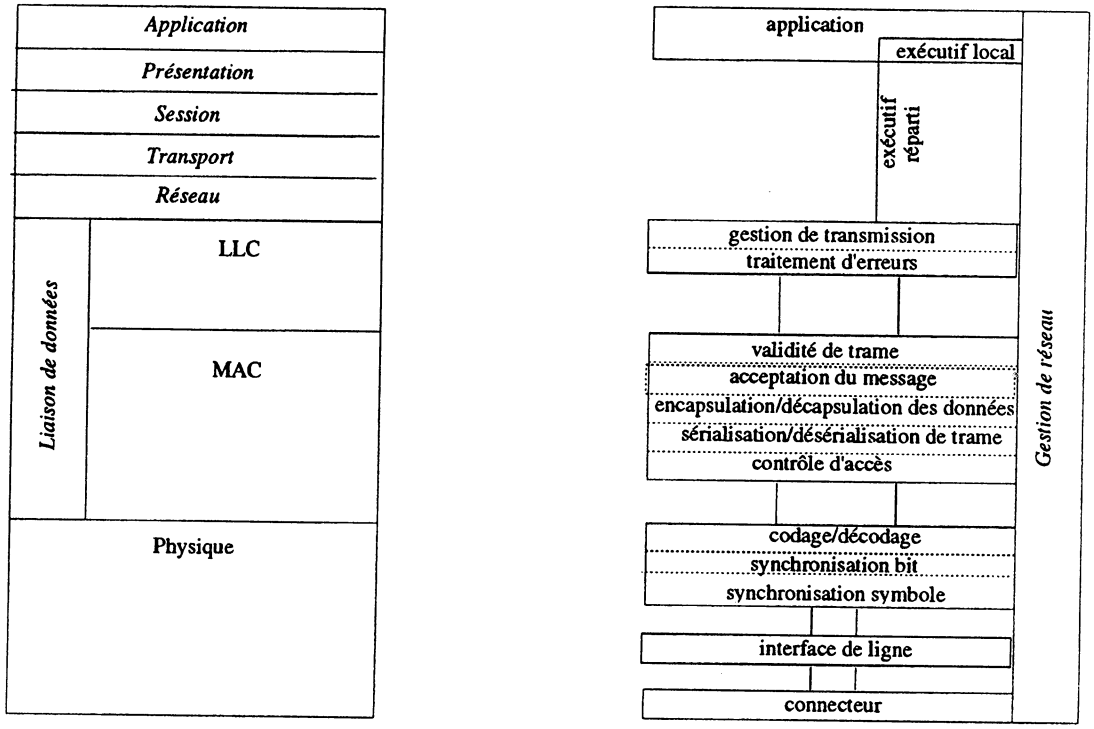


figure 11 : modèle de référence du protocole VAN

2.1 Description du Protocole VAN

Nous présentons ici, quelques caractéristiques générales des deux premières couches du modèle de référence du protocole VAN. Il ne s'agit pas de présenter ici la norme VAN, mais de donner un aperçu sur les couches basses et surtout sur la couche MAC (qui nous intéresse dans le cadre de l'étude).

2.1.1 La couche Physique

Le médium de communication a des caractéristiques électriques (inductance, capacitance, isolement) qui sont entiérement spécifiées dans la norme.

Il peut être une paire torsadée classique, une nappe de deux fils de transmission ou un câble coaxial (voire fibre optique: extension possible).

On distingue deux états sur le bus :

- un état récessif (R) qui correspond à un niveau 1 TTL,
- un état dominant (D) qui correspond à un niveau 0 TTL.

Alors, lorsque deux états de la même nature sont émis simultanément, l'état du médium prend la valeur de ceux-ci. Par contre, si un état dominant et un état récessif sont envoyés au même instant, le dominant l'emporte. Ainsi, le médium de communication se comporte comme un ET câblé lors d'une collision et permet de résoudre le problème d'accès au médium. Cette méthode d'accès est appelée la méthode d'accès à compétition non-destructive.

Le même type de codage doit être employé sur tout le réseau. La technique du codage/décodage adoptée dans le protocole utilise le deux types de code: le code Manchester normal ou Manchester_1 et le code Manchester comprimé.

Un codage spécifique est utilisé pour différencier le début et la fin de la trame du codage de données employé.

2.1.2 La couche MAC

Plusieurs types de trames du niveau MAC dans le protocole VAN, utilisent le format générique suivant:

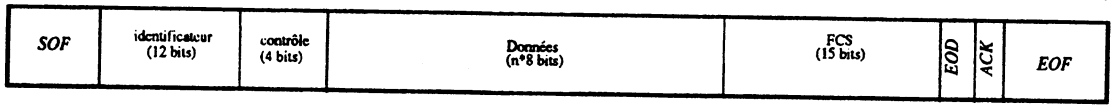


figure 12: format générique de la trame VAN

La trame est délimitée par un champ début appelé *SOF* (*Stat Of Frame*) et un champ fin de trame EOF (*End Of Frame*). Le *SOF* est composé d'un préambule et d'un *start bit* et doit être précédé par un état de repos.

Les différents types de trames sont:

. *Trame de données sans acquittement* : Il s'agit d'une trame de transmission de données ne nécessitant pas un accusé de réception (*ACK*).

. *Trame de données avec acquittement* : Elle est utilisée dans le cas d'un échange point-à-point. C'est une trame de transmission de données qui nécessite une réponse (accusé de réception *ACK*) de la part du récepteur.

. *Trame de question/réponse* : C'est une trame sans données émise par un module autonome pour demander à un autre module un transfert d'informations. Selon la station, on peut distinguer deux cas:

- Dans le cas d'une station esclave, elle doit compléter immédiatement la trame en insérant ses données.
- si elle est de type autonome, elle a dans ce cas la possibilité de répondre soit dans la trame, comme dans le cas précédent, ou dans une trame différente (ceci dans le cas où les données ne sont pas encore disponibles).

C'est le champ de contrôle appelé (aussi commande *COM*) qui permet de différencier entre les différents types de trames. Ce champ est composé de 4 bits dont la signification est la suivante:

- 1 er bit, *EXT* : son usage est laissé pour une future extension;

- 2 ème bit, *RAK* (*Requet AcKnowledge*), : il permet d'indiquer lorsqu'il est validé, une demande d'acquittement de la part du récepteur;

- 3 ème bit, *R/W* : Il indique si la trame est une trame de lecture (demande de données) ou d'écriture (émission de données);

- 4 ème bit, *RTR* (*Remote Transmission Request*) : Si ce bit est positionné, il permet de faire une demande de transmission distante (le récepteur insère éventuelement des données dans la même trame après ce bit.

2.1.2.1 Le Champ Identificateur :

Le champ identificateur (*IDENT*) permet d'identifier le(s) récepteur(s) de la trame. Il peut s'agir soit :

- d'une adresse physique qui désigne un destinataire unique,
- ou bien d'une adresse fonctionnelle de groupe qui décrit plutôt la signification des informations contenues dans la trame que de désigner ses destinataires.

La norme n'impose pas un mode d'adressage particulier, c'est au concepteur du réseau de choisir la nomenclature la plus adaptée à son application.

2.1.2.2 Le Champ *FCS*

Les données des champs de la trame sont protégées par un CRC 15 bits correspondant au champ *FCS* (*Frame Control Sequence*).

2.1.2.3 Acquittement d'une Trame

Lorsque le bit de demande d'acquittement (*RAK*) est positionné, un acquittement est demandé par la station source, alors le champ accusé de réception *ACK* (*ACKnowledge*) sera positionné par le destinataire.

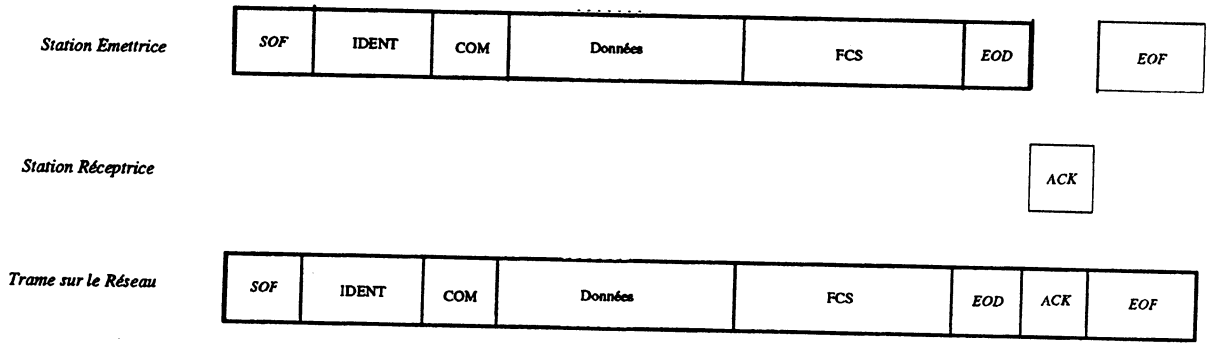


figure 13: Trame de données avec un acquittement (VAN)

On remarque que tous les champs de la trame ne sont pas émis par la même station. L'émetteur envoie les champs SOF, IDENT, COM, Données, FCS et EOD, le récepteur insère le champ ACK pour que l'émetteur finisse par envoyer le champ EOF.

2.1.2.4 Réponse dans la Trame

Lorsqu'une station maître envoie une trame de question vers une station esclave, celle-ci peut répondre immédiatement dans la même trame. Les différents champs envoyés par le maître sont : SOF, IDENT et COM. La demande de données est exprimée dans le champ COM par:

- R/W : positionné en mode lecture;
- RTR : positionné en une demande de transmission distante par une valeur récessive.

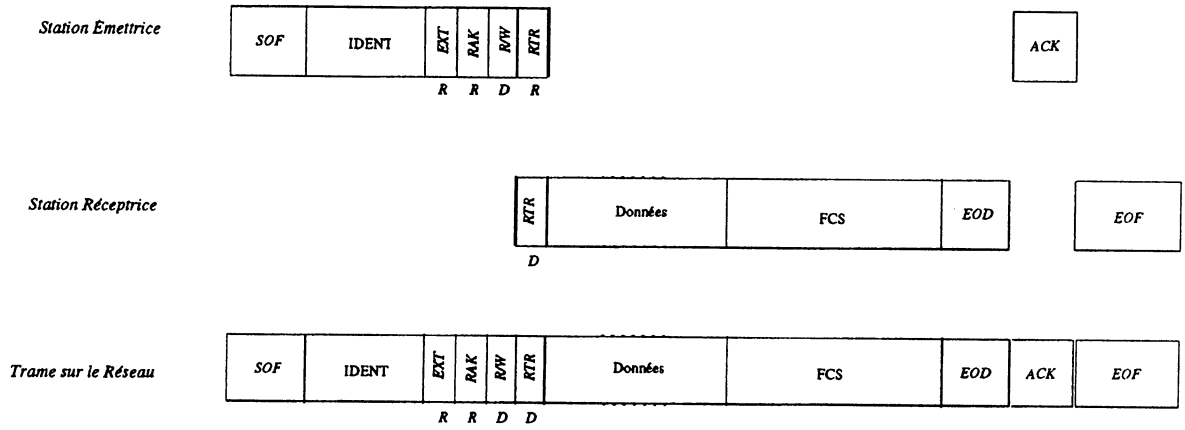


figure 14 : réponse dans la trame (VAN)

Dans le cas où la station interrogée peut répondre dans la trame, alors dès qu'elle reçoit le R/W, elle envoie un bit dominant RTR qui va écraser le RTR récessif de l'émetteur et insère ensuite le champ de données, le champ FCS et le champ EOD aussitôt après. La station acquittera alors les données en positionnant le champ ACK et la station interrogée finit la trame en envoyant le champ EOF.

2.1.2.5 La Méthode d'Accès

Des conflits d'accès au médium de communication peuvent arriver car le VAN est un protocole multi-maître multi-esclave. Après avoir détecté que le bus est disponible, plusieurs modules maîtres peuvent commencer à transmettre simultanément et par conséquent il y a l'apparition d'une contention.

L'arbitrage bit à bit est utilisé dans la méthode d'accès pour garantir l'envoi d'une trame et une seule pour qu'elle soit non-destructive. Chaque station écoute en permanence sur la ligne et doit immédiatement cesser d'émettre dès qu'elle détecte une interférence.

A l'aide de cette technique, l'identificateur du message donne des niveaux de priorité entre les différents messages émis au même temps.

2.2 Spécification des Interfaces

Afin de réaliser le circuit de communication VAN, nous spécifions dans cette partie les deux interfaces:

- l'interface avec le côté application
- et l'interface avec le côté réseau.

2.2.1 Côté Station

Nous prenons l'exemple d'interface Motorola déjà défini dans le paragraphe 1.1.4.A.

2.2.1.1 Adaptation : Bus Motorola

Nous définissons alors l'interface Motorola avec un bus de données de 8 bits (*D0* à *D7*) et un bus d'adresse de 2 bits (*RS0* et *RS1*). Le bus de contrôle est composé alors des signaux suivants:

- une ligne *R/W* indiquant qu'il est en mode de lecture ou d'écriture;
- d'une ligne de validation du circuit *CS* ;
- d'une ligne *Reset* d'initialisation du circuit;
- d'une ligne INT qui est une sortie indiquant la présence d'une interruption;
- et d'une ligne *E* pour la validation du cycle de lecture ou d'écriture.

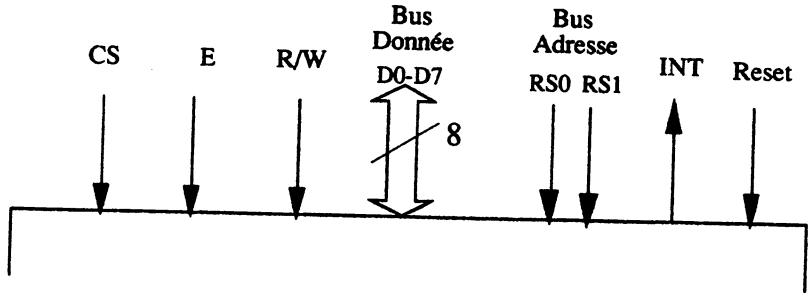


figure 15: interface du circuit VAN

2.2.1.2 Bufferisation des Informations

Dans la partie mémorisation de données, nous utilisons deux fifos de données pour l'émission et la réception de 32 mots de 8 bits chacune.

Nous définissons alors:

- deux registres d'état de 8 bits chacun pour signaler les comptes rendus;
- un registre de commande de 8 bits;
- deux registres d'Adresse.

Le schéma de la partie bufferisation est similaire à celui qui a été proposé au pragraphe 1.1.4.A.

2.2.2 Côté Réseau

L'interface côté réseau est constitué essentiellement d'un ensemble de signaux de synchronisation pour dialoguer avec la couche physique. Trois classes de signaux sont à définir: les signaux de synchronisation, les signaux d'erreurs et les sigaux de données.

2.2.2.1 Signaux de Synchronisation

Parmis les signaux de synchronisation implicites, on a essentiellement les signaux suivants:

. horloge d'émission et de réception; (entrée)
. signaux indiquant les différents types de champ à traiter; (sortie)
. signal indiquant le début de réception; (entrée)
. signal indiquant que le bus est libre; (entrée)
. signal indiquant le début d'émission; (sortie)

2.2.2.2 Signaux d'Erreurs

Ce sont les différents signaux indiquant la présence d'une erreur sur la ligne. Ces erreurs peuvent être de différentes natures telles que : les erreurs de format de la trame, les erreurs de synchronisation, collision (perte du bus), ...

Dans le cas de notre exemple, on considère qu'il y a au niveau de l'interface une ligne indiquant une erreur regroupant toutes les erreurs d'interface basse.

2.2.2.3 Signaux de données

Il y a deux lignes de données représentant :
. la ligne des bits de réception : en entrée;
. la ligne des bits d'émission : en sortie.

2.3 Description MACS de l'Application

Une description MACS de l'application a été effectuée pour l'implémentation du protocole VAN pour la synthèse du circuit de communication correspondant.

Ce programme comporte un ensemble d'unités de description permettant de définir :

- l'interface Station : le choix a été porté sur l'interface Motorola.
- l'interface Réseau : déclaration des signaux de contrôle et de synchronisation.
- les différents Champs des différentes trames : tous les champs qui ont été décrits dans la trame VAN sont déclarés.
- les unités de traitements impliquées : l'unité de traitement qui a été essentiellement utilisée est le traitement CRC.
- le Protocole : description des processus d'émission et de réception correspondant à une station autonome.

2.4 Implémentation Architecturale

La synthèse du circuit avec l'outil MACS permet de générer une architecture suivant le modèle d'architecture déjà défini dans le chapitre 3.

2.4.1 Architecture du Circuit

L'architecture est décomposée principalement de deux parties: la partie interface avec le côté station et la partie intégrant le protocole de communication (Coeur du circuit).

Le schéma bloc général du circuit est présenté dans la figure suivante:

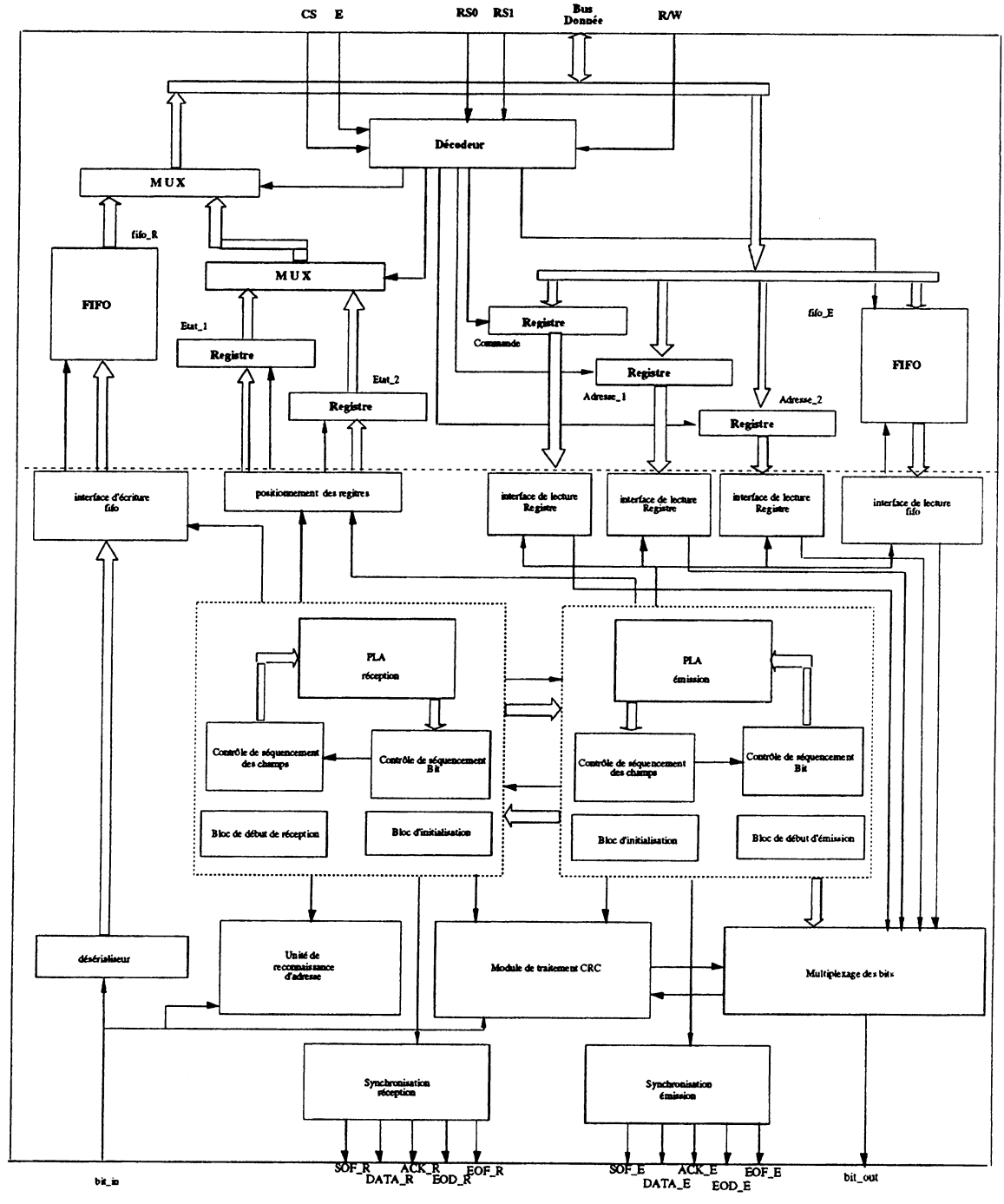


figure 16: Architecture du circuit VAN

2.4.1.1 La Partie Interface

La partie interface comporte d'une part une partie adaptation qui est constituée d'un ensemble de portes logiques pour former un décodeur de sélection des unités de bufferisation et de validation des fonctions de lecture et d'écriture des données d'application, et d'autre part, la partie bufferisation des données qui est constituée d'un ensemble de registres, de fifos et de multiplexeurs.

2.4.1.1 Le Coeur du Circuit

Le coeur du circuit est composé essentiellement de deux machines : la machine d'émission et la machine de réception. Chaque machine est composée d'une partie contrôle et d'un ensemble de modules de traitement.

A Partie Contrôle :

Les séquencements des bits et des champs sont traités dans cette partie contrôle. Parmi les traitements particuliers de séquencement, il y a le branchement vers d'autres états et le passage de l'émission vers la réception et vice-versa dans le cas d'une réponse dans la trame. La partie contrôle génère les différents signaux de synchronisation et d'activation des modules de traitement.

B Modules Internes:

Les différents modules de traitement sont :

- Les interfaces de lecture: Ces interfaces permettent d'effectuer des lectures à partir des éléments de mémorisation qui sont les registres et le fifo. Du côté de l'émission, 4 interfaces sont nécessaires pour lire les registres d'adresse, de commande et le fifo de données d'émission.

- Les interfaces d'écriture : Deux interfaces d'écriture sont nécessaires pour mémoriser des données dans le fifo de réception et les comptes rendus dans les registres d'état.

- L'unité de traitement CRC : Le traitement CRC est réalisé par un module CRC généré avec la spécification du polynome caractéristique pour la couverture des erreurs de données de la trame VAN. Ce polynome est spécifié dans la norme.

- Multiplexage des données d'émission : L'envoi des différents champs est réalisé à travers un module multiplexeur de bits des différentes lignes séries des interfaces de lecture.

- L'unité de traitement d'Adresse : La fonction de reconnaissance d'adresse est réalisée par un comparateur sériel présenté dans l'annexe B. On peut distinguer plusieurs types d'adresses :

. adresse physique : celle qui est implémenté dans cet exemple où l'adresse de la station est spécifiée dans le circuit par le concepteur (adresse figée),

. adresse logique : dans ce cas , un registre dans la partie bufferisation est utilisé pour stocker le paramètre adresse spécifié par l'exterieur (la station).

. adresse de groupe : Le concepteur peut dans ce cas désigner une adresse de groupe implémentée dans le circuit permettant de reconnaitre une diffusion d'une trame.

- L'unité de reconnaissance de ACK : Dans le cas où le champ d'acquittement est présent dans la trame, cette unité permet de signaler à la partie contrôle la réception d'un acquittement.

- L'unité de reconnaissance de RTR: La réponse dans la trame est reconnue par le positionnement du bit RTR. Cette unité permet donc de signaler à la partie contrôle le passage de l'émission en réception.

- Module de Synchronisation : Ce module permet la génération des différents signaux de synchronisation vers la couche physique.

- Désérialiseur : à la réception, les données reçues sont désérialisées par ce module et envoyées vers l'interface d'écriture dans le fifo de réception.

2.4.2 Description VHDL du Circuit

La description VHDL du circuit est générée par MACS sous forme de plusieurs fichiers contenant chacun une description d'un niveau donné. Cette description décrit la structure du circuit sous forme d'un ensemble de composants de base interconnectés. La hiérarchie de l'architecture a été respectée pour permettre une structuration modulaire de la description.

Les différents fichiers qui sont générés en respectant la hiérarchie sont les suivants:

- fichier de description du PLA :
  . *PLA_EMIT* et *PLA_RECEPT*

- fichier de description du Contrôleur :
  . *CTRL_EMIT* et *CTRL_RECEPT*

- fichier de description du coeur du circuit :
  . *MAC_"PROTOCOL_NAME" : MAC_VAN*

- fichier de description de l'interface station :
  . *INTERFACE_"PROTOCOLE_NAME" : INTERFACE_VAN*

- fichier de description du circuit :
  . *CIRCUIT_"PROTOCOLE_NAME" : CIRCUIT_VAN*

2.5 Modélisation d'un Système de Communication VAN

2.5.1 Spécification du Réseau VAN

Afin de modéliser complètement le système de communication, il faut commencer par décrire les applications qui doivent être intégrées dans le réseau, de configurer les protocoles de contrôle-commande correspondant et d'établir la liaison entre les applications par l'intermédiaire du réseau de communication constitué d'un circuit de communication pour chaque application et d'un médium de liaison.

On présente dans cette partie une modèlisation du réseau VAN dont la description a été réalisée à la main. Le protocole de contrôle-commande utilisé est trés simplifié. Il consiste à initialiser le circuit et à le mettre en état d'activation pour l'émission et la réception de données de l'application.

A. Modélisation du système:

Tout le système de communication devient complétement modélisé lorsque tous les éléments de la hiérarchie sont reliés : Médium de communication, circuit de communication, partie protocole contrôle-commande et l'application

(figure 17). Le but final est de réaliser des simulations comportementales de tout le système pour avoir des évaluations précises du comportement global.

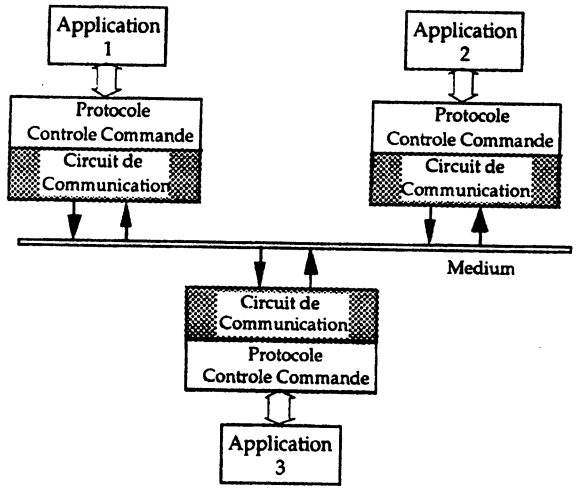


figure 17: Modélisation du système de communication

B. Génération des stimulis de simulation:

Elle consiste à générer les différents vecteurs de test à différents niveaux de la hiérarchie. La méthode de validation du système consiste à vérifier le bon fonctionnement du système à partir du médium de communication jusqu'à l'application.

C. Observation et analyse des résultats:

Le résultat de la simulation s'effectue à différents niveaux:

. *niveau médium de communication:*
.observation des trames sur le médium;

. *niveau circuit de communication:*
. observation de l'état de la communcation;

. *niveau de la partie intégrant le protocole contrôle-commande:*
. observation des informations échangées et des problèmes dûs à la charge du réseau.

2.5.2 Simulations et Observations du Réseau

Un exemple de simulation du protocole VAN au niveau MAC est illustré après son implémentation architectural en VHDL. Le réseau est configuré en quatre stations, deux maîtres et deux esclaves. Chaque station est configurée par son adresse réseau permettant d'établir une priorité entre les différentes stations. La simulation de trois types de trames est présentée dans ce qui suit:

A. Trame de données avec demande d'acquittement:

La station maitre MASTER I envoie des données à la station maitre MASTER II avec une demande d'acquittement.

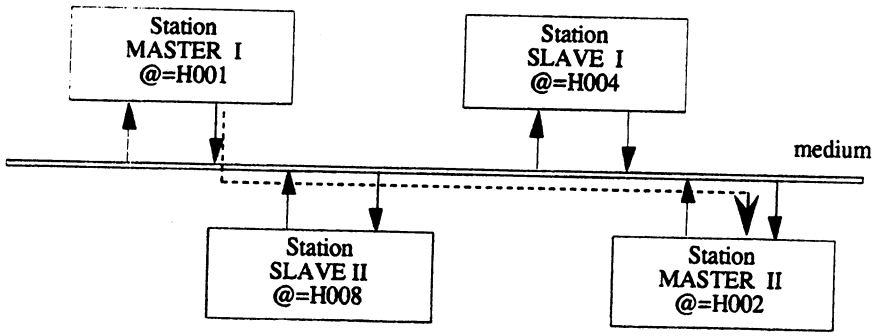


figure 18 : Emission d'une trame de données(VAN)

B. Trame de demande de données avec réponse différée:

La station maitre MASTER II demande des données à la station maitre MASTER I et la réponse de données est différée.

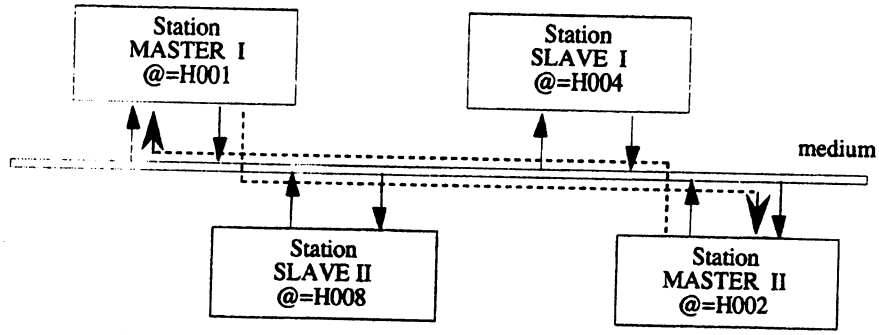


figure 19 : Emission d'une trame de demande de données
avec réponse dans une trame différée (VAN)

C. *Trame de demande de donnée avec une réponse dans la trame:*

La station maitre MASTER I envoie une trame de demande de données à la station esclave SLAVE I et en même temps, la station maître MASTER II envoie une trame de demande de données à la station esclave SLAVE II. Une résolution de conflit d'accès au médium est réalisée et une réponse dans la trame est envoyée.

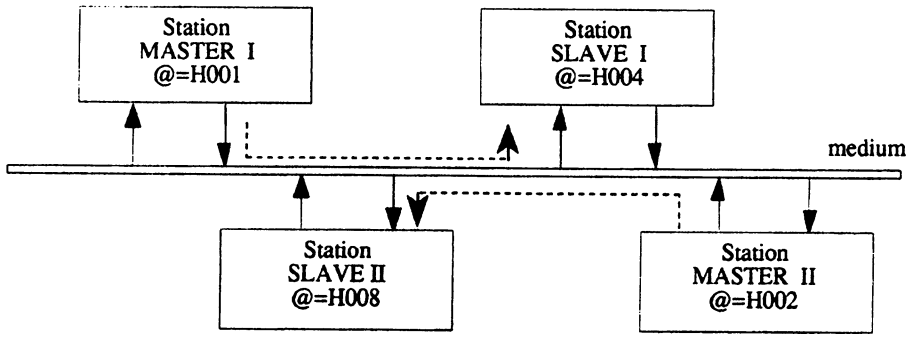


figure 20 : Emission d'une trame de demande de données
avec réponse immédiate dans la même (VAN)

Résultat de simulation:

L'annexe E présente les chronogrammes des différentes simulations VHDL de chaque cas en décrivant les différents signaux de début d'émission et de réception et les états de la ligne de communication.

Conclusion

Un environnement de spécification, d'implantation et d'évaluation constitue un système de développement permettant de faire des évaluations comportementales de l'ensemble du réseau en intégrant les différentes ressources de communication de différents niveaux avec les contraintes du réseau.

Un système de synthèse d'architecture pour l'implémentation des protocoles de communication de niveau MAC permet à un utilisateur de spécifier un protocole MAC donné, de l'implémenter et de faire sa simulation en l'intégrant dans un réseau.

Plusieurs applications ont été présentées dans ce chapitre afin de mettre en évidence la démarche de conception que l'utilisateur est amenée à suivre en partant de la description du protocole jusqu'à son implantation.

CONCLUSION

— CONCLUSION —

LA CONCEPTION des circuits de traitement de l'information en général est un domaine dans lequel divers réflexions et travaux de recherches sont menés. Le circuit de communication en particulier constitue un vaste domaine si l'on tient compte des applications sur les réseaux temps-réel, les réseaux de contrôle-commande, les réseaux RNIS, ...

L'implantation des protocoles de communication de niveau bas (et intermédiaire) sur silicium a été choisie comme domaine de réflexion pour la conception des circuits de communication. L'importance de la couche MAC dans la classe des réseaux de contrôle-commande nous a conduit à faire une analyse de plusieurs réseaux et des circuits du marché pour dire que: "Les ressemblances significatives dans la spécification du protocole MAC nous a permis de définir d'une part, un modèle d'architecture cible générique pour l'implantation de circuits de communication et d'autre part, une facilité pour la description de leurs fonctionnalités en vue de l'implémentation desdits protocoles de communication".

La spécification et plus généralement le développement d'un système complexe pose des problèmes importants. La prise en compte des diverses contraintes doit être abordée dès le stade de la spécification en vue de l'obtention d'un optimum global.

L'objectif recherché est d'implanter systématiquement les protocoles des méthodes d'accès au médium (niveau MAC) à partir d'une description comportementale du protocole. Ceci permet de prototyper rapidement l'implantation des protocoles de communication de contrôle-commande pour étudier leur comportement par la simulation complète de tout le système.

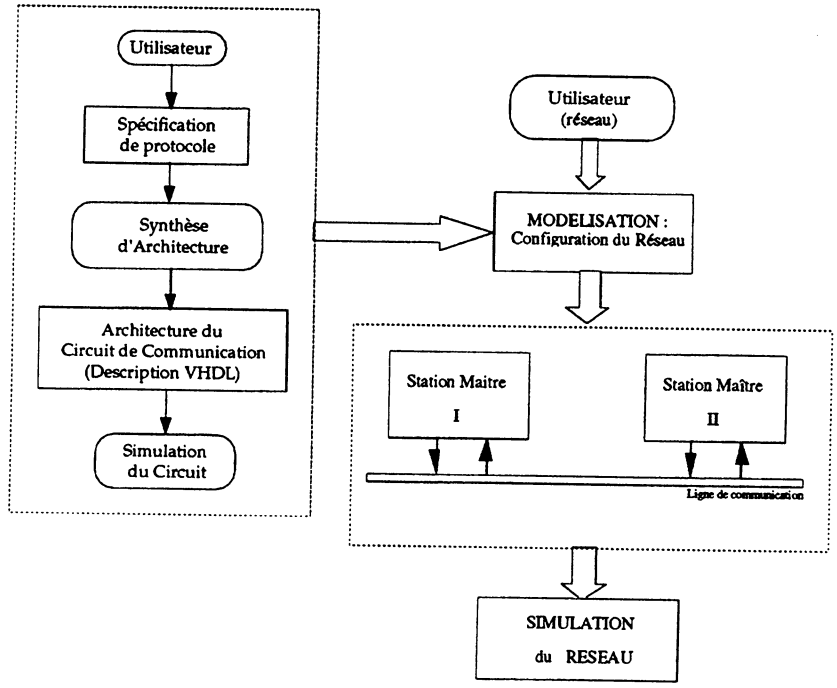
La flexibilité et la généricité du modèle d'architecture proposé ont conduit à l'utilisation d'une méthode de conception automatique qui est la synthèse.

L'architecture cible est à contrôle dominant et constituée d'un ensemble de modules asynchrones. Elle est basée sur la séparation entre l'analyse de la syntaxe de la trame et le traitement des données contenues dans ses champs. Ceci a permis de définir un langage, appelé MACS, plus compact que des langages généraux tels que VHDL comportant naturellement des traits particulièrement spécifiques des protocoles de niveau MAC.

Un système de synthèse d'architecture pour l'implémentation des protocoles de communication de niveau MAC permet à un utilisateur de spécifier un protocole donné, de l'implémenter et de faire sa simulation en l'intégrant dans un réseau de communication.

Ainsi, un environnement de spécification, d'implantation et d'évaluation constitue un système de développement permettant de faire des évaluations comportementales de l'ensemble du réseau en intégrant les différentes ressources de communication de différents niveaux avec les contraintes du réseau.

Le schéma global de cet environnement est le suivant:



Environnement de spécification, de modélisation et de simulation d'un système de communication

La première perspective, est l'extension de la démarche de conception pour les protocoles de niveaux supérieurs. Cette extension se traduit généralement dans les réseaux de contrôle-commande par la migration des fonctionnalités de l'intelligence de la station vers le circuit de communication. Dans ce cas, le circuit va pouvoir étendre la prise en compte des aspects temps-réel vers les protocoles de niveau haut. Par conséquent, d'autres instructions deveront être ajoutées pour la spécification de ces dits protocoles. Le travail de synthèse sera alors, de différencier l'ensemble des instructions à implémenter au niveau bas de ceux du niveau supérieur.

La deuxième perspective découle de la première en généralisant la démarche pour des applications générales afin de spécifier les systèmes de communication. Ceci aboutit à ce qu'on appelle déjà la conception mixte matérielle-logicielle (*"Hardware-Software Co-design "*).

BIBLIOGRAPHIE

Bibliographie

[Abri84] J.R. Abrial, "Spécifier ou comment matérialiser l'abstrait"
TSI Technique et Science Informatique, Mars 1984.

[Air90] R. Airiau, J.-M. Bergé, V. Olive et J. Rouillard, "VHDL : Du Langage à la Modélisation"
Collection Technique et Scientifique des Télécommunications.
Presses Polytechniques Romandes. 1990.

[AuFC87] E. Audureau, L. Farinas Del Cerro, P. Enjalbert, "Theorie de la programmation en logique temporelle"
TSI Technique et Science Informatique, Décembre 1987.

[BaGW82] Balzer R., Goldman N., Wile D., "Operational specification as the basis for rapid prototyping" ACM SIGSOFT Software Engeneering Notes, Vol 7 No 5, Dec 1982.

[Barb81] M.R. Barbacci, "Instruction Set Processor Specification (ISPS) : The notations and its applications"
IEEE Transactions on Computers, vol C-30 No. 1 January 1981, pp24—40.

[BeCo85] G. Berry, L. Cosserat, "The synchronous programming language ESTEREL and its mathematical semantics" LNCS 197, Springer-Verglag 1985.

[BeCG87] G. Berry, P. Couronne, G. Gonthier, "Programmation synchrone des systèmes réactifs: le langage ESTEREL"
TSI Technique et Science Informatique, Aout 1987.

[BiGa85] Bidoit M., Gaudel M.-C.,"PLUSS : Proposition pour un langage de spécification structurée"
Proc. Journées AFCET, Paris, Oct 1985, BIGRE-GLOBULE n° 45, pp 28-35

[Brac88] G. Bracon, "La Spécifiocation de logiciel dans l'industrie"
Bigre n° 58, Janvier 1988

[BuGo86] R.M. Burstall, J.A. Goguen, "The semantics of CLEAR, A Specification Language" LNCS 86, Spingler-Verla, 1980.

[Bye87] TJ Byers, "Microprocessor Communications Support Chips" Elsevier Science Publishing Co., Inc. 1989.

[CAD89] "SKILL Language Manual" CADENCE Data System. 1989.

[CEG89] "Full-FIP Component Specifications", CEGELEC RED. 1989.

[ChGa88] C. Choppy et M. Gaudel, "Impact des specifications formelles sur le développement des logiciels" Bigre n° 58, Janvier 1988

[CoSB82] Cohen M., Swartout W., Balzer R., "using symbolic execution to characterize behaviour" ACM SIGSOFT Software Engineering Notes, Vol. 7 No. 5, Dec 1982, pp25-32.

[Cour89] J.P. Courtial, P. Dembinski, R. Groz, C. Jard,"ESTELLE : un langage ISO pour les algorithmes distribués et les protocoles" TSI Technique et Science Informatique, Avril 1987.

[CPL89] "CPLEX", CPLEX Optimization, Inc. Manuel d'utilisation. 1989.

[Csb82] Cohen M., Swartout W., Balzer R., "Using symbolic execution to characterize behaviour", ACM SIGSOFT Software Engineering Notes, Vol. 7, No. 5, Dec 1982.

[Dan86] M. Dang, M. Diaz-Nava and G. Michel, "Design of a VLSI Communicating Circuit for an Industrial Local Network in Control Process and Automates Production", Proceedings of the EUROMICRO'86. Venise — 09/1986.

[Dan87] M. Dang,"Les Circuits VLSI de Communication de Données : Architecture et Méthodologie", 9ème Journée Francophone sur l'informatique. Liège — 01/1987.

[Dan88] Ng. X. Dang, Ch. Diot, I. Sabouni and L. Sponga, "Specific Data Structure intended for the Implementation of High level ISO Standards — Associated and Dedicated Hardware".
Proceedings of the EUROMICRO'88. Zurich — August 1988.

[Dan88] M. Ng. X. Dang,"Les Circuits Intégrés de Communication de Données — Architecture et Méthodologie de Conception",
Thèse de Doctorat ès-Sciences. INP de Grenoble. 1988.

[Dia87] M. Diaz, "ESTELLE : une Technique de Description Formelle des Protocoles"
9ème Journée Francophone sur l'Informatique. Liège — Janvier 1987.

[DLC92] "VAN Data Link Controller"
Preliminary Data Sheet.
MHS.RE/PSA/RNUR. 1992.

[Dron88] P. J. Drongows, J.R Bammi, R. Ramaswamy, S. Iyengar and T.H Wang,
" A Graphical Hardware Design Language"
25th Design Automation Conference, 1988.

[EhMa85] H.D. Ehrich, B. Mahr, "Fundamentals of Algebraic Specification"
Springer-Verlag 1985.

[Futa85] Futatsugi, "Principles of OBJ2"
in Proc. ACM Principles of programming languages 1985.

[FVC92] "Full-VAN Microcontroller Specification",
Preliminary Data Sheet.
TEXAS INSTRUMENTS 1992.

[Gaud83] Gaudel M.-C., "Proposition pour un langage d'utilisation de spécification structurées: PLUSS",
Rapport de recherche C.G.E 1983.

[GuHo83] Guttag J. V., Horning J.J. ,"An introduction to the Larch Shared Language",
Proc. IFIF Conference, 1983, North-Holland, pp 809-814.

[Hare88] D. Harel, "On visual formalisms"
Comm. of ACM, Vol. 31, No 5, may 1988, pp. 514-530.

[Horn85] Horning J.J., "Combining algebraic and predicative specifications in Larch",
Proc. International Joint Conference on Theory and Practice of Software Development (TAPSOFT), Berlin, Mars 1985, L.N.C.S. 186, Springer Verlag.

[HPSS87] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman, "On the formal semantics of StateCharts"
in Proc. of the 2nd IEEE Symposium on logic in Computer Science, 1987.

[IEEE83] "IEEE Project 802, Local Area Network Standard"
Draft IEEE standard 802.2 Logical Link Control. ISO/TC97/SC6.03/1983.

[IEEE83] "IEEE Project 802, Local and Metropolitan Area Network Standard"
Draft IEEE standard 802.1. Overview and Architecture. ISO/TC97/SC6.06/1983.

[IEEE85] "IEEE Project 802, Local and Metropolitan Area Network Standard"
Draft International Standard, ANSI/IEEE Standard 802.3 ISO/DIS 8802/3. 1985.

[INT91] "Automotive Handbook".
INTEL 1991.

[INT90] "Microcontroller Handbook".
INTEL 1990.

[ISO90] "Vehicule Area Network Standard",
ISO/TC22/SC3/WG1.04/1990.

[Jone86] Jones C., "Systematic software development using VDM",
Prentice-Hall Int., Series in Computer Science, 1986.

[Kris92] A.S. Krishnakumar,"A Synthesis System for Communication Protocols"
Proceeding of the 5th Annual IEEE International ASIC Conference and Exhibit.
Rochester , September 1992.

[KrKS87] A.S. Krishnakumar, B. Krishnamurthy and K. Sabnani,"Translation of Formal Protocol Specifications into VLSI Design", Protocol Specification, Testing and Verification, VII

H. Rudin and C. West (Eds.), May 1987, Elseivier Science Publishers B.V.(North-Holland).

[KrKS89] A.S. Krishnakumar and Sabnani,"VLSI Implementations of Communication Protocols — A Survey", IEEE Journal on Selected Areas and in Communications. vol 7, n° 7 09/1989.

[Liss] M. Lissandre, "La méthode SADT"
Génie Logiciel n° 4

[Loto87] ISO-DIS-8807,"LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour"
Jully 1987.

[Luck85] D. Luckham, F.W. Von Henke, " An overview of Anna, a specification Language for ADA"
IEEE Software, Vol. 2, No 2, pp. 9-23, March 1985.

[Macc87] C. Macchi et J.-F. Guilbert. "Téléinformatique"
Dunod Informatique.Nouvelle Edition. 1987.

[Marc79] M. De Marco, "Structured analysis and system specification"
Yourdon Press 1979

[Meye85] B. Meyer,"On formalism in specification"
IEEE Software, January 1985

[Miln80] Milner, "A Calculus of Communicating Systems",
Lecture Notes in Computer Science Vol. 92, Springer-Verlag, 1980.

[NHWG89] M. Nielsen, K. Haveland, K.R. Wagner, C. George, "The RAISE language, Method and Tools"
Formal aspects of Computing, Vol. 1, 1989.

[PSA90] "VAN Remote Control Switch",
Draft Proposal for Recommanded Practice. PSA DT-DAT-TEA. 1990.

[RCP91] "Spécification du Circuit Contrôleur de Protocole VAN"
RNUR-EC/PHILIPS. 1991.

[Sab92] I. Sabouni, "Génération de Modules Dédiés aux Circuits de Communicaytion de Contrôle-Commande"
Thèse INPG, 1992.

[Sab92_1] I. Sabouni, H. Saheb and M. Dang, "An ASIC Architecture for Implementing MAC-Level Communication Circuits"
Proceeding of the 5th Annual IEEE International ASIC Conference and Exhibit.
Rochester , September 1992.

[Sah91] H. Saheb, J. Guedes-Silveira, I. Sabouni and M. Dang, "A Special-Purpose Flexible FIFO Design intended for Communication Circuits"
Proceedings of the 3rd International Conference on Microelectronicx. Cairo — 12 1991.

[Sah92] H. Saheb, "Etude de l'Implantation d'un Module Analyseur de Trame"
Rapport de DEA Microelectronique. Université Joseph Fourier 1989.

[Sah92_1] H. Saheb, I. Sabouni and M. Dang "An Architectural Model for Low-Level Communication Circuits"
Proceeding of the 4th International Conference on Microelectronics. Monastir, 1992.

[Sah92_2] H. Saheb, I. Sabouni and M. Dang "An Efficient Approach for Implementing MAC-Level Communication Circuits"
Proceeding of the 35th Midwest Symposium on Circuits and Systems.
Washington , August 1992.

[Sah93] H. Saheb and M. Dang, "An Architecture Synthesis of low level Communication Circuits"
Proceeding of the 6th Annual IEEE International ASIC Conference and Exhibit.
Rochester , September 1993.

[Smol82] Smoliar S., "Approaches to executable specifications",
ACM SIGSOFT Software Engineering Notes, Vol 7, No5, Dec 1982.

[Tard] H. Tardieu, "La méthode MERISE"
Génie Logiciel n° 4

[VAN92] "Etat d'avancement du Développement VAN"

Rapport PSA Etudes et Recherches. Septembre 1992.

[Wing90] J. M. Wing,"A specifier's Introduction to Formal Methods"
IEEE Computer, September1990

Bibliographie de Annexe A

[AbBr85] M. Abadir and M.A. Breuer, "A knowledge-based system for designing testable VLSI chips"
IEEE Design and Test of Computers, August 1985, pp56—68.

[Barb81] M.R. Barbacci, "Instruction Set Processor Specification (ISPS) : The notations and its applications"
IEEE Transactions on Computers, vol C-30 No. 1 January 1981, pp24—40.

[BCMO88] R.K. Brayton, R. Composano, G. De Micheli, R.H.J.M. Otten and J. Van Eijndhoven, " The Yorktown Silicon Compiler System (YSC)"
Silicon Compiler edited by Daniel D. Gajski, Addison-Wesley, 1988, pp204—310

[Bers89] V. Berstis, "The V Compiler: Automating Hardware Design"
IEEE Design and Test of Computers, April 1989, pp8—17.

[BoDe88] G. Boriollo and E. Detjens, "High-level Synthesis : Current Status and Future Directions"
25th ACM/IEEE Design Automation Conference, 1988, pp477—482.

[BoKa87] G. Borriello and R. H. Katz, "SYNTHESIS and OPTIMIZATION of INTERFACE TRANSDUCER LOGIC"
IEEE International Conference on CAD, November 1987.

[BoKa87] G. Borriello and R. H. Katz, "Synthesizing Transducers from Interface Specifications"
IEEE International Conference on VLSI, August 1987.

[Borr88] G. Borriello, "Combining Event and Data-Flow Graphs in Behavioral Synthesis"
IEEE International Conference on CAD, November 1988.

[Bran87] D. Brand, "Logic Synthesis"
Design systems for VLSI circuits, logic synthesisand silicon compilation,
Martinus Nijhoff, 1987, pp301—326.

[ChBr85] T. Chen and M.A. Breuer, "Automatic design for testability via testability mesures"
IEEE Transactions on CAD, vol CAD-4 No. 1, January 1985, pp3—11.

[CITh90] R.J. Cloutier and D.E. Thomas, "The combination of scheduling, allocation and mapping in a single algorithm"
27th ACM/IEEE Design Automation Conference, 1990, pp71—76.

[DeNe89] S. Devadas and R. Newton, "Algorithms for hardware allocation in data path synthesis"
IEEE Transactions on CAD, vol 8 No. 7, July 1988, pp768—781.

[DGMC90] F. Depuydt, G. Goosens, J. Van Meerbergen, F. Cothoor and H. De Man,
"Scheduling of large data flow graphs based on metric graph clustering"
Proceedings of Euro-ASICs, Paris 1990, pp168—177.

[Ewer90] C. Ewering, "Automatic high-level synthesis of partitionned buses"
International Conference on CAD, 1990, pp304—307.

[FaKo90] M.C. Mc Farland and T.J. Kowalski, "Incorporating Bottom-Up Design in hardware synthesis (BUD)"
IEEE Transactions on CAD, Vol 9 No. 9,September 1990, pp939—950.

[Farl86] M.C. Mc Farland, "Using Bottom-UpDesign techniques in the synthesis of digital hardware from abstract behavioural descriptions"
23rd ACM/IEEE Design Automation Conference, 1986, pp474—480.

[Farl87] M.C. Mc Farland, "Reevaluating the design space fot register-transfer hardware synthesis"
24rd ACM/IEEE Design Automation Conference, 1987, pp262—265.

[FPCo88] M.C. Mc Farland, Alice C. Parker and Raul Composano, "Tutorial on
High-level Synthesis"

25rd ACM/IEEE Design Automation Conference, 1988, pp330—336.

[Fren82] S. French, Sequencing and Scheduling, "An introduction to the mathematics of the job-shop", Ellis Horwood Lted, 1982.

[FuHi86] H.S Fung and S. Hirschhorn, "An automatic DFT system for the Silc silicon compiler" IEEE Design and Test of Computers, February 1986, pp45—57.

[GAS86] GASP, Logiciel de Synthèse de Systèmes Logiques, Combinatoires et Séquentiels. Manuel Utilisateur. APTOR S.A. 1986.

[GaJo79] M.R. Garey and D.S. Johnson, "Computers and Intractability: A guide to the theory of NP-Completeness"
W.H. Freeman and Co., 1979.

[GaJs88] D.D. Gajski, "Silicon Compilation"
Addison-Wesley, 1988.

[GBKn85] E.F. Girczyc, R.A. Buhr and J.P. Knight, "Applicability of a subset of ADA as an algorithmic hardware description language for graph-based hardware compilation" IEEE Transactions on CAD, vol CAD-4 No. 2, April 1985, pp134—142.

[GDPa86] D.D. Gajski, N.D. Dutt and B.M. Pangrle, "Slicon Compilation (Tutorial)" IEEE 1986 Custom Integrated Circuits Conference, pp102—110.

[GeEI88] C.H. Gebotys and M.I. Elmasry, "Integrated Design and Test synthesis" IEEE International Conference on CAD, 1988, pp398—401.

[GeEI89] C.H. Gebotys and M.I. Elmasry, "Integration of algorithmic VLSI synthesis with testability incorporation"
IEEE Journal of Solid-State-Circuits, vol 24 No. 2, April 1989, pp409—415.

[GeEI90] C.H. Gebotys and M.I. Elmasry, "A global optimization approach for architectural synthesis"
IEEE International Conference on CAD, 1990, pp258—261.

[GRVM87] G. Goosens, J. Rabaey, J. Vanderwalle and H. De Man, "An efficient macrocode-compiler for custom DSP processor"

24th ACM/IEEE Design Automation Conference, 1987, pp24—27.

[GRVM87] G. Goosens, J. Rabaey, J. Vanderwalle and H. De Man, "An efficient macrocode-compiler for application specific DSP processor"
IEEE Transactions on CAD, vol 9 No. 9, September 1990, pp925—937.

[HaPa82] L.J. Hafer and A.C. Parker, "Automated synthesis of digital hardware"
IEEE Transactions on Computers, vol C-31 No. 2, February 1982, pp93—109.

[HCLH90] C. Huang, Y. Lin and Y. Hsu, "Data path allocation based on bipartite weighted matching"
27th ACM/IEEE Design Automation Conference, 1990, pp499—504.

[HHLi90] C. Huang, Y. Hsu and Y. Lin, "Optimum and heuristic data path scheduling under resource constraints"
27th ACM/IEEE Design Automation Conference, 1990, pp65—70.

[HiTh83] C.Y. Hitchcock III and D.E. Thomas, "A method of automatic data path synthesis"
20th ACM/IEEE Design Automation Conference, 1983, pp484—489.

[HPKi87] Y. Hong, K. Ho Park and M. Kim, "Automatic synthesis of datapaths based on the path-search algorithm"
24th ACM/IEEE Design Automation Conference, 1987, pp270—273.

[KGVe83] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing"
Science, May 13th, 1983, vol 220, No. 4598, pp671—680.

[Knap90] D.W. Knapp,"Feedback-driven data path optimization in FASOLT"
IEEE International Conference on CAD, 1990, pp300—303.

[KuMi90a] D. Ku and G. De Micheli,"High-level synthesis and optimization strategies in Hercules Hebe"
Proceedings of Euro-ASICs, Paris 1990, pp9—14.

[KuMi90a] D. Ku and G. De Micheli,"Relative scheduling under timing constraints"
27th ACM/IEEE Design Automation Conference, 1990, pp59—64.

[KuPa87] F.J. Kurdahi and A. Parker, "REAL: a program for register allocation"

24th ACM/IEEE Design Automation Conference, 1987, pp210—215.

[KuPa90] K. Kuçukçakar and A. Parker,"Data path trade-offs using MABAL"
27th ACM/IEEE DesignAutomation Conference, 1990, pp511—516.

[LDSh80] D. Landskov, S. Davidson and B. Shriver,"Local microcode compaction techniques"
Computing Surveys, vol. 12 No. 13, September 1980, pp261—294.

[LEGi90] T. A. Ly, W.L Elwood and E.F. Girczyc, "A generalized interconnect model for data path synthesis"
27th ACM/IEEE DesignAutomation Conference, 1990, pp168—173.

[LHLi89] J. Lee, Y. Long, "A new integer linear programming formulation for the scheduling problem in data path synthesis"
IEEE International Conference on CAD, 1989, pp20—23.

[MiKu88] G. De Micheli and D. Ku,"Hercules: a system for high-level synthesis"
25th ACM/IEEE DesignAutomation Conference, 1988, pp483—487.

[NeKr90] J. A. Nestor and G. Krishnamoorthy,"SALSA: A new approach to scheduling with timing constraints"
IEEE International Conference on CAD, 1990, pp262—265.

[NeTh86] J.A. Nestor and D. E. Thomas, "Behavioral Synthesis with Interfaces"
IEEE International Conference on CAD, November 1986.

[OlR83] V. Olivier and D. Rouquier, "A Systematic Method for the Synthesis of Control Parts defined by GRAFCET"
Proceedings of the Design Automation Conference. June 1988.

[OuKa91] H. Oudghiri et B. Kaminska, "Synthèse de haut niveau: ordonnancement et allocation"
EPM/RT-91-2, Ecole Polytechnique de Montréal, Mai 1991.

[PaGa87a] B.M. Pangele and D.D Gajski,"SLICER: a state synthetizer for intelligent silicon compilation"
IEEE International Conference on Computer Design, October 1987, pp42—45.

[PaGa87b] B.M. Pangele and D.D Gajski,"Design tools for intelligent silicon

compilation"
IEEE Transactions on CAD, vol CAD-6 No. 6, November 1987, pp1098—1112.

[PaKn87] P. G Paulin and J.P Knight, "Forced-directed scheduling in automatic data path synthesis"
24th ACM/IEEE Design Automation Conference, 1987, pp195—202.

[PaKn89a] P. G Paulin and J.P Knight, "Forced-directed scheduling for the behavioral synthesis of ASICs"
IEEE Transactions on CAD, vol CAD-8 No. 6, June 1989, pp661—679.

[PaKn89a] P. G Paulin and J.P Knight, "Algorithms for High-level synthesis"
IEEE Design and Test of Computers, December 1989, pp18—31.

[PaKo90] C.A Papachristou and H. Konuk, "A linear program driven scheduling and allocation method followed by an interconnect optimization algorithm"
27th ACM/IEEE Design Automation Conference, 1990, pp77—83.

[Pang88] B.M Pangrle, "SPLICER : a heuristic approach to connectivity binding"
25th ACM/IEEE Design Automation Conference, 1988, pp536—541.

[PaPa86] N. Park and A.C Parker, "SEHWA: A program for synthesis of pipelines"
23rd Design Automation Conference, 1986, pp454—460.

[Park84] A.C Parker, "Automated synthesis of digital systems"
IEEE Design and Test of Computers, November 1984, pp75—81.

[RaCG92] R. K. Gupta, C. N. Coelho and G. De Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components"
29th ACM/IEEE Design Automation Conference, pp225—230.

[RaGi92] R. K. Gupta and G. De Micheli, " System-level Synthesis using Re-programmable Components"
Proceedings of the European Design Automation Conference, Mars 1992.

[Rou89] D. Rouquier. "L'Echo des Recherches",
N° 135/1989.

[TsHs90] F. Tsai and Y. Hsu,"Data path construction and refinement"
IEEE International Conference on CAD, 1990, pp308—310.

[TsSi83] C. Tseng and D.P Siewioreck, "FACET: a procedure for the automated synthesis of digital systems"
20th ACM/IEEE Design Automation Conference, 1983, pp490—496.

[TsSi83] C. Tseng and D.P Siewioreck, "Automated synthesis of data paths in digital systems"
IEEE Transactions on CAD, vol CAD-5 No. 3, July 1986, pp376—395.

[WGHe90] N. Wehn, M. Glesner and M. Held, "A novel scheduling/allocation approach for data path synthesis based on genetic paradigms"
Proceedings of Euro-ASICs, Paris 1990, pp186-193.

[Woo90] N. Woo, "A global, dynamic register allocation and binding for data path synthesis system"
27th ACM/IEEE Design Automation Conference, 1990, pp505—510.

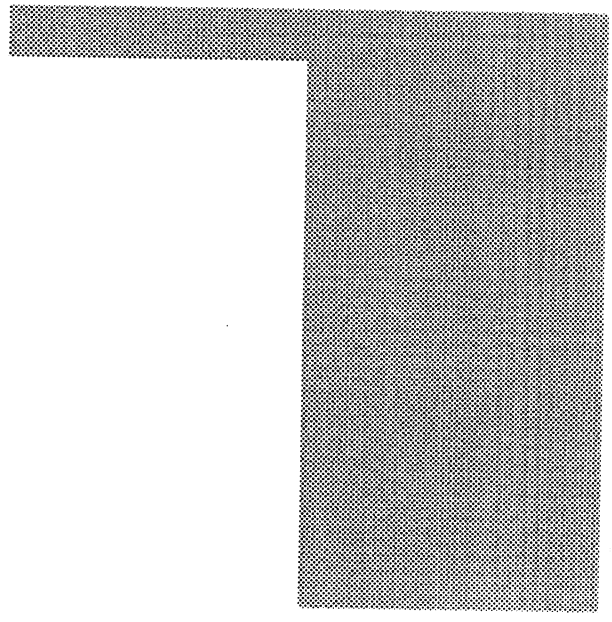
Annexe B

[RaG88] T.V. Ramabadran and S. S. Gaitonde,"A Tutorial on CRC Computation",
IEEE Micro. August 1988.

[Sah91] H. Saheb, J. Guedes-Silveira, I. Sabouni and M. Dang, "A Special-Purpose Flexible FIFO Design intended for Communication Circuits"
Proceedings of the 3rd International Conference on Microelectronic. Cairo — 12 1991.

Annexe A

Panorama des Démarches en Synthèse d'Architecture



Annexe A

Introduction

1 Synthèse de Haut Niveau : Synthèse d'Architecture
- 1.1 Processus de Conception d'un Circuit Intégré
- 1.2 Synthèse d'Architecture

2 Description Comportementale

3 La Représentation Interne et son Optimisation
- 3.1 La Représentation Interne
- 3.2 Optimisation de la Représentation Interne

4 Ordonnancement et Allocation
- 4.1 Ordonnancement
- 4.2 Allocation
- 4.3 Interaction entre l'Allocation et l'Ordonnancement
  - 4.3.1 Ordonnancer d'Abord
  - 4.3.2 Allouer d'Abord
  - 4.3.3 Ordonnancer et Allouer au même Temps
- 4.4 Techniques d'Ordonnancement
  - 4.4.1 Algorithmes transformationnels
  - 4.4.2 Algorithmes itératifs/constructifs
  - 4.4.3 Programmation mathématique
- 4.5 Techniques d'Allocation
  - 4.5.1 Allocation itérative/constructive
  - 4.5.2 Allocation globale
- 4.6 Fonction de Coût
- 4.7 Techniques de Raffinement d'un Design

Conclusion

Bibliographie

PANORAMA DES DEMARCHES EN SYNTHESE D'ARCHITECTURE

La Synthèse de Haut Niveau

Introduction

Durant les années quatre vingts, le développement accru des technologies VLSI qui a permis de réaliser des circuits de plus en plus complexes a provoqué une crise qui a ralenti la conception des circuits intégrés. En raison du nombre et de la complexité des éléments à considérer dans ces circuits, le temps nécessaire pour réaliser ces puces équivalait à la durée de vie du circuit lui-même, ce qui a provoqué un engorgement dans le cycle de développement des produits.

Un début de solution au problème était l'apparition d'un certain nombre d'outils automatiques d'aide à la conception tels que les simulateurs logiques, les vérificateurs de règles , les compacteurs ..., qui sont intégrés dans un environnement CAO. Malgré ces outils qui ont permis d'automatiser quelques étapes du processus de conception, les décisions à prendre durant la conception revenaient au concepteur. L'idée d'une automatisation de tout le processus de conception a germé dès lors.

La conception peut être orientée sur deux axes différents: la conception dédiée ou la conception générale. Dans le premier cas, le produit réalisé est dédié à une application bien spécifique alors que dans le deuxième cas le produit est appelé à être utilisé dans diverses applications. Le choix d'implémentation dépend du coût de réalisation, des caractéristiques fonctionnelles et surtout des contraintes temporelles exigées.

Dans les deux cas, le même cheminement d'implémentation a été empreinté pour concevoir des circuits. Le premier pas est celui de la conception à la main où le concepteur réalise toutes les étapes de la conception en partant des spécifications jusqu'au dessin du "*layout* ". Et puis, est apparue l'utilisation de bibliothèques de cellules précaractérisées, des outils de placement et routage automatiques, des outils de synthèse logique, des générateurs de modules , ...

Aujourd'hui, presque dans tous les systèmes de CAO VLSI, un environnement de conception "complet" est fourni au concepteur pour implémenter son circuit. Cependant, partir de la spécification d'un cahier des charges jusqu'au circuit est un problème qui préoccupe beaucoup de concepteurs. Cette partie est bien réalisée dans certains domaines d'applications et reste à désirer dans d'autres.

Le terme "compilation de silicium" a été utilisé pour la première fois par Dave Johansen pour décrire l'assemblage de cellules *"layout"* paramétrisées. Le terme est devenu maintenant très populaire dans le domaine de la CAO des circuits intégrés. En général, la compilation de silicium est définie comme étant le processus de transformation d'une description de haut niveau en un *"layout"*. Une description de haut niveau permet de cacher un certain niveau de détail à l'utilisateur. Cette description de haut niveau peut être, par ordre d'éloignement du dessin, un *"layout"* symbolique, une schématique, un ensemble d'expressions booléennes, un schéma logique, une description comportementale d'une architecture, un jeu d'instructions ou un algorithme de traitement de signal [Rou89].

Un compilateur de silicium permet d'étendre le champ des ASICs (*Application Specific Integrated Circuit*) puisqu'il permet de travailler à des niveaux d'abstraction très élevés sans se préoccuper des détails de bas niveau (*"layout"* et technologie). Il permet aussi d'améliorer la qualité de la conception, puisque les circuits conçus sont corrects par construction. Les erreurs commises à un haut niveau sont plus faciles à détecter et à corriger. En plus, le compilateur de silicium permet d'augmenter la productivité en diminuant le temps de conception car on ne se soucie plus des détails du "*design* ".

Cependant, la souplesse et la standardisation des compilateurs de silicium, qui ont tendance à générer des circuits pour toute application, se traduisent finalement

par une perte au niveau de l'optimisation des systèmes conçus, comparativement aux résultats obtenus par un concepteur humain qui s'intéresse à une application spécifique.

Néanmoins, la compilation de silicium demeure une discipline intéressante principalement en ce qui concerne la force motrice introduite pour l'intégration de différents champs de l'informatique tels que les langages de description, les algorithmes de conception et d'optimisation, les bases de données, l'environnement CAO et les bases de connaissance dans une nouvelle discipline qu'on pourra appeler la science de la conception.

Le processus de conception dans un compilateur de silicium peut comporter plusieurs étapes de transformation dont chacune est un compilateur pour l'étape suivante. On parle de synthèse logique lorsque le résultat de la transformation est un ensemble de portes logiques et de bascules. Par contre la synthèse d'architecture démarre avec un jeu d'instruction pour arriver à un ensemble de registres, de bus et d'unités de traitement. [OuKa91]

On s'intéresse dans ce qui suit à la synthèse de haut niveau qui reste toujours un domaine d'exploration. Par contre tout ce qui concerne la synthèse logique ou l'implémentation à partir d'une description structurelle d'une architecture, beaucoup de travaux ont été réalisés et plusieurs outils sont déjà opérationnels[GAS86] [OlR83].

1 Synthèse de Haut Niveau : La Synthèse d'Architecture

1.1 Processus de Conception d'un Circuit Intégré

Un circuit intégré peut être décrit dans trois domaines de description: domaine comportemental, domaine structurel ou domaine géométrique. L'implémentation d'un circuit est le passage d'un domaine de description à un autre.

- Domaine comportemental: à ce niveau, le système est vu par ce qu'il fait et non pas par comment il est fait. C'est son comportement qui nous intéresse ici, et qui peut être décrit par: une table de vérité, une équation booléenne, un langage de haut niveau, ...

- Domaine structurel: dans ce domaine, le circuit est vu comme un ensemble de composants interconnectés. Ces composants peuvent être une structure logique, électrique ou un transfert de registres.

• Domaine géométrique: à ce niveau, la description du circuit se présente sous forme d'un ensemble de rectangles superposés représentant les différents matériaux du processus de fabrication. Dans ce cas, le circuit est décrit tel qu'il sera implanté sur le silicium.

Le processus de conception est essentiellement un passage itératif d'un domaine à un autre. On commence par décrire le circuit dans une représentation comportementale pour arriver à un résultat dans une représentation géométrique. Des compromis sont considérés durant le processus d'itération pour satisfaire les contraintes imposées (temps, surface, ...). Le diagramme en Y de D. Gajski (Y-chart) est une représentation tripartite d'un système de conception VLSI (figure 1). Les trois axes représentent les différents domaines de représentation: comportementale, structurelle et géométrique. Chaque axe est composé de plusieurs niveaux d'abstraction, et plus on s'éloigne du centre de Y plus le niveau d'abstraction est élevé. Le déplacement sur le même axe ou d'un axe à un autre, représente les outils utilisés lors du processus de conception [GaPa86] [Gajs88] [OuKa91].

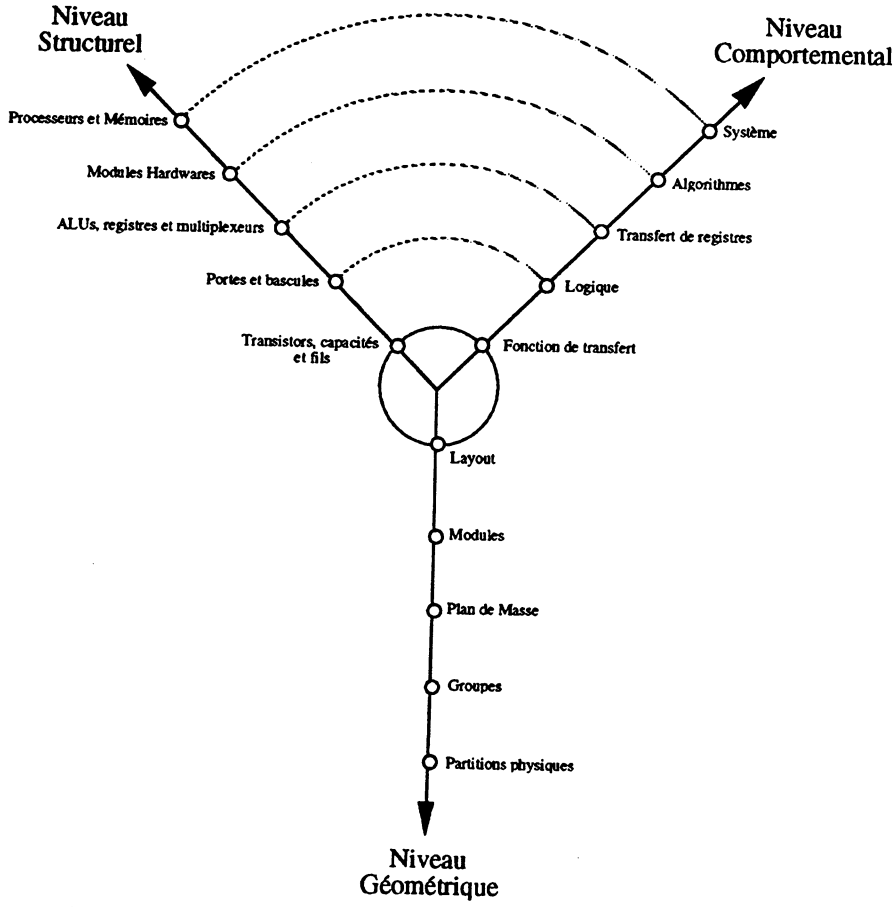


figure 1: Processus de conception

1.2 Synthèse d'Architecture

La synthèse d'architectures consiste à générer une description de circuit au niveau structurel (RTL) à partir d'une description comportementale. Cette dernière prend généralement la forme d'un algorithme. L'espace de recherche des solutions possède deux dimensions:

- d'une part, la durée ou le temps nécessaire à l'exécution de l'algorithme sur l'architecture générée. Cette durée est habituellement mesurée en étapes ("*control steps* ");
- d'autre part, la surface de silicium nécessaire pour implanter l'architecture.

La durée et la surface sont deux grandeurs qui évoluent en sens inverse. En effet, le nombre d'opérations élémentaire qui peuvent être simultanément réalisées dépend directement des ressources matérielles disponibles (opérateurs, mémoire, interconnexions, etc...). En général, l'espace des solutions est exploré en bornant une des dimensions et en cherchant une solution minimisant la deuxième dimension.

La synthèse d'architecture est un processus qui se décompose en plusieurs étapes (figure.2.).

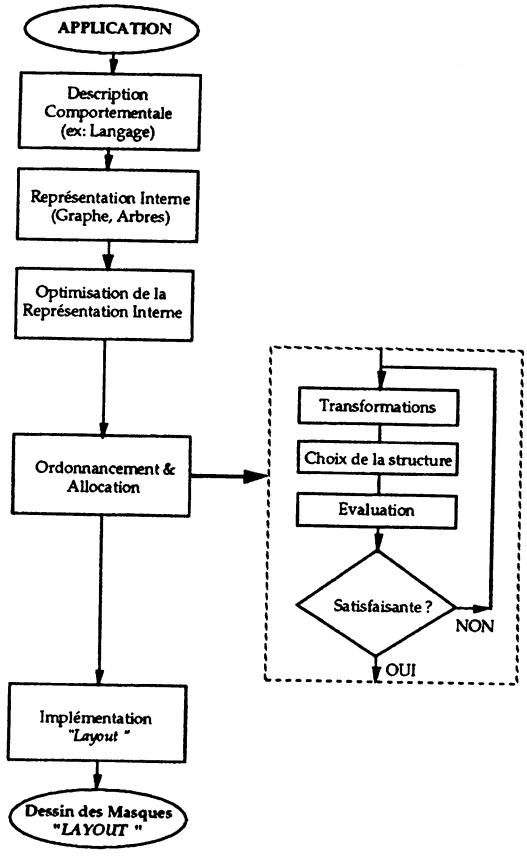


figure.2: Etapes de la Synthèse de Haut Niveau:

2 Description comportementale

On commence dans la première étape par décrire le comportement du système dans un langage de haut niveau qui peut être:

. un langage de programmation (Pascal, ADA, ...)
. un langage de description hardware (ISPS, VHDL, ...)
. un langage de description hardware non procédural (PROLOG, like-LISP, ...)
. ou autres ...

Un langage de description de niveau comportemental doit être très souple, facile à apprendre et comporter des caractéristiques facilitant la description des circuits. Parmi ces caractéristiques, les plus importantes sont:

• Le typage des données: pour faciliter la détection d'erreurs comme la combinaison de chemins de données de largeurs différentes, la largeur des données doit être spécifiée.

• Les appels de routines: pour offrir la possibilité d'appeler des parties de la description qui se répètent.

• La simulation fonctionnelle: pour vérifier la fonctionnalité du système au plus haut niveau.

Généralement, les différents langages utilisés pour la description comportementale sont des langages procéduraux. Ils permettent de décrire la manipulation des données en instructions élémentaires d'assignation, constituant à leur tour des blocs avec des structures standards de contrôle (conditionnelles et itérations). Les langages les plus utilisés sont des langages de description hardware comme ISPS (Instruction Set Processor Specification) [Barb81], VHDL [PLNG90] et VT [Bers89]. Des pseudo-langages de programmation sont également utilisés tels que ADA [GBKn85] et C [KuMi90].

3 La représentation interne et son Optimisation

3.1 La représentation interne

Un système de synthèse ne peut travailler directement sur une description textuelle. Les systèmes de synthèse passent par une représentation interne où les relations de dépendance et d'exclusion entre les différentes opérations et les données sont dégagées. Les graphes sont les plus fréquement utilisés vu la diversité et le nombre de techniques développées pour leur manipulation (parcours des graphes, recherche de cycles, partitionnement, etc ...).

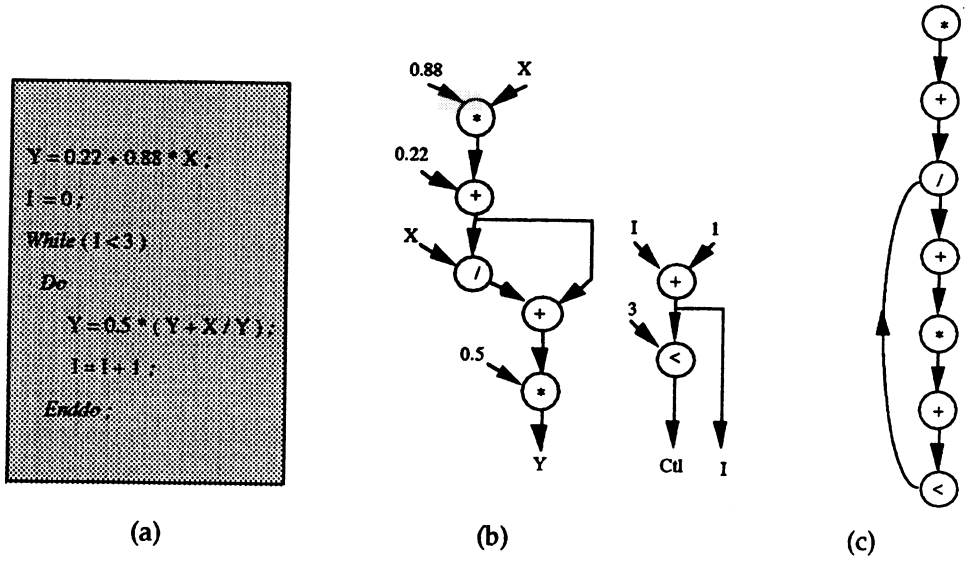
Un comportement de circuit peut être décrit par deux graphes:

- Un graphe de flux de données ("*DATA FLOW* ") qui représente les différentes opérations et les dépendances de données entre elles.

- Un graphe de flux de contrôle ("*CONTROL FLOW* ") qui décrit le séquencement des opérations tel que spécifié par la description comportementale.

La figure 3 présente une partie d'un programme qui calcule la racine carrée de x en utilisant la méthode de Newton ainsi que sa description en graphe [FDCo88]. Le graphe de flux de contrôle est extrait directement de l'ordre explicite donné par le programme et du choix du compilateur dans le calcul des expressions arithmétiques (figure.3.b). Quant au graphe de flux de contrôle, il indique l'ordre des opérations imposé par les dépendances de données. Par exemple, la première addition dans le graphe de la figure.3.c dépend du résultat obtenu par la multiplication, puisque ce résultat constitue un opérande de l'addition. Cela signifie que la première multiplication doit toujours être réalisée avant l'addition quelque soit l'ordre choisi pour l'ensemble des opérations. Par contre, aucune dépendance n'a lieu entre l'incrémentation de i et n'importe quelle opération participant dans le calcul de y. Donc l'opération i+1 peut être exécutée en parallèle avec tout autre opération du graphe.

Chaque groupe de recherche développe son propre compilateur pour son propre pseudo-langage. Il n'y a pratiquement pas de compilateurs standards. Un exemple de tels compilateurs spécifiques est le V-compiler pour le langage VT décrit dans [Bers89].



(a) (b) (c)

figure 3 : (a) description comportementale
(b) Graphes de flux de données
(c) Graphe de flux de contrôle

3.2 Optimisation de la représentation interne

Cette étape consiste à réaliser quelques transformations préliminaires sur la représentation interne. Ces transformations ont pour but de réaliser une première optimisation sur la description fournie par le concepteur. Des exemples de transformations sont l'élimination de sous-expression commune, l'élimination du code mort (les opérations jamais atteignables), l'expansion des procédures et le déroulement des boucles. L'outil d'optimisation de la représentation interne est identique aux outils d'optimisation du code utilisé dans les compilateurs des langages de programmation. Des exemples d'optimisation applicables à l'exemple de la figure.3. sont: remplacer la multiplication par 0.5 par un décalage à droite, l'addition d'un 1 par une incrémentation et dérouler la boucle puisque le nombre d'itération est réduit.

Le système Hercules [MiKu88] est un exemple de système où différentes techniques d'optimisation de la représentation interne sont utilisées. Il permet de traduire une description en langage C modifié dans un graphe de flux de données.

4 Ordonnancement et Allocation

La principale tâche de la synthèse est de trouver la structure la plus adaptée au comportement, compte tenu de certaines contraintes (surface, temps d'exécution, etc ...). Deux problèmes sont à résoudre : l'ordonnancement et l'allocation.

4.1 Ordonnancement

L'ordonnancement permet de trouver le meilleur ordre d'exécution des opérations, c'est-à-dire assigner les opérations à des étapes de contrôle (cycles de la machine) pour exécuter cette séquence en un temps minimal, sans violer les dépendances en données des opérations. Le degré de parallélisme de l'exécution de ces opérations est alors défini par l'ordonnancement trouvé.

La figure.4. présente deux possibités ordonnancement pour le graphe flux de données donné dans la figure.4.a. Le premier ordre, figure.4.b, s'exécute en trois étapes et requiert deux multiplieurs et un additionneur comme ressources matérielles de traitement alors que l'ordre proposé à la figure.4.c s'exécute en quatre étapes mais n'a besoin que d'un multiplieur et un additionneur. Cet exemple montre bien que l'ordonnancement nous confronte à un problème de compromis entre la vitesse et le matériel.

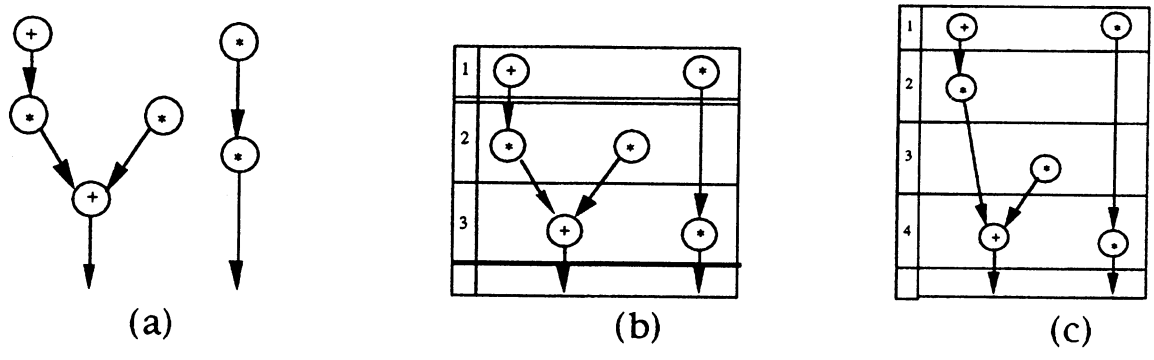


figure 4 : Exemple d'ordonnancement

4.2 Allocation

Trois assignations doivent être réalisée pour résoudre le problème de l'allocation:

a) L'assignation des opérations aux unités fonctionnelles. Elle doit prendre en compte le compromis entre la quantité de matériel et la durée d'exécution.

Pour minimiser le coût, les opérations qui ne sont pas exécutées simultanément sont allouées à la même unité fonctionnelle.
On a besoin de moins de matériels nécéssaires lorsque plusieurs opérations sont assignées à la même unité fonctionnelle, mais le degré de parallélisme diminue. Par contre, lorsque les opérations sont distribuées sur plusieurs unités fonctionnelles, le degré de parallélisme augmente mais plus de matériels sont utilisés.

b) L'assignation des variables aux éléments de mémoire ou registres. La minimisation du nombre de registres alloués se fait par l'assignation d'un même registre aux variables dont les intervalles d'utilisation sont disjoints.
Pour ce faire, on définit pour chaque variable du code d'entrée sa durée de vie, qui n'est autre que l'intervalle entre la première et la dernière référence à cette variable. Le nombre de registres requis est alors égal au recouvrement maximal de toutes les durées de vie de toutes les variables (densité maximale ou nombre maximal des variables utilisées en même temps).

c) L'assignation des chemins de données à des interconnexions physiques (multiplexeurs ou bus). La minimisation du nombre d'interconnexions se fait en assignant les chemins de données qui ne sont pas utilisés en même temps aux mêmes interconnexions.

Un contrôleur doit être réalisé après les étapes d'ordonnancement et d'allocation dont le rôle est de générer les différentes commandes nécéssaires au contrôle de la partie opérative allouée. En général, la partie contrôle peut être câblée ou micro-programmée. Dans le premier cas, la partie contrôle est synthétisée par différentes techniques telles que l'encodage des états ou l'optimisation des réseaux logiques combinatoires. Dans le second cas, le micro-programme est optimisé au moyen des techniques d'encodage.

4.3 Interaction entre l'allocation et l'ordonnancement

L'ordonnancement et l'allocation sont étroitement liés. Chaque technique a besoin de détail sur l'autre pour que son traitement soit efficace. En général, pour trouver l'ordonnancement le plus efficace pour le "hardware" réel, on doit connaître les délais des différentes opérations. Cependant, on ne peut connaître ces délais qu'après avoir défini les unités fonctionnelles et leurs interconnexions. Inversement, pour trouver le nombre optimal des unités fonctionnelles requis, on doit savoir si les opérations seront exécutées en parallèle ou en série, une information qui ne peut être fournie que par l'ordonnancement. On est donc confronté à un problème de

dépendance, tel qu'il est illustré à la figure.5. Selon l'ordre d'exécution de ces deux sous-tâches, on distingue trois approches : ordonner d'abord, allouer d'abord, ou ordonner et allouer au même temps.

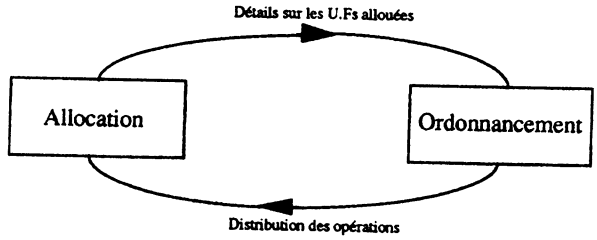


figure.5: Intéraction entre l'allocation et l'ordonnancement

On peut dire qu'on est en présence d'un problème où le point de départ est défini différemment selon les systèmes de synthèse.

4.3.1 Ordonnancer d'abord

Cette technique consiste à définir (ou à ne pas définir) une limite sur le nombre d'unités fonctionnelles disponibles et ensuite réaliser l'ordonnancement. De nombreux systèmes utilisent ce principe, non pas pour une seule limite, mais plutôt en répétant ce processus pour plusieurs limites. A chaque itération, le système est synthétisé et évalué pour chaque limite définie. Les itérations sont arrêtées dès l'obtention d'une solution satisfaisant les contraintes imposées. Parmi les systèmes utilisant cette approche, il y a DAA [KGWF85], Emerald [TsSi86] et [PaGa87b].

4.3.2 Allouer d'abord

Dans ce cas, le système de synthèse partitionne les opérations en groupes selon une métrique. Cette métrique prend en compte le potentiel de partage, d'interconnexion et de parallèlisme entre opérations. Une unité fonctionelle est allouée à chaque groupe d'opérations et l'ordonnancement est effectué en fonction de cette allocation. BUD est un exemple de système utilisant cette approche [Far186] et [KuMi90b].

Une variante consiste à construire une partie opérative avec un parallélisme maximal. Une optimisation est ensuite réalisée, guidée par les contraintes temporelles et de surface. Les opérations sont ordonnées selon les contraintes imposées par la partie opérative allouée [SGLM90].

4.3.3 Ordonnancer et Allouer au même temps

Dans cette approche, le système alloue des unités au fur et à mesure qu'il ordonne les opérations. De nouvelles unités sont allouées lorsqu'il n'est plus possible de partager celles déjà allouées. Cette approche est utilisée dans MAHA [PPM186], HAL [PaKn87] [CITh90] [HPKi87] et [PaKo90].

Une autre façon de procéder suivant le même principe d'interaction consiste à démarrer avec toutes les opérations dans la même étape de contrôle et une unité fonctionnelle pour chaque opération [BCMO88]. Des étapes de contrôle sont ajoutées au fur et à mesure afin d'éviter les conflits (dans l'accès aux variables). Le matériel est alors optimisé de façon à exploiter au mieux le parallélisme au maximum.

4.4 Techniques d'ordonnancement

Trois grandes classes d'algorithmes d'ordonnancement sont distinguées : les algorithmes transformationnels, les algorithmes itératifs et la programmation mathématique en nombres entiers.

4.4.1 Algorithmes transformationnels

Ce type d'algorithme d'ordonnancement commence par un ordre par défaut (série ou parallèle). L'ordonnancement final est obtenu après une série de transformations (série au parallèle ou vice versa). Ce qui différencie les algorithmes transformationnels les uns des autres est le type de transformation utilisée.

L'un des premiers algorithmes utilisait la recherche exhaustive (EXPL de Barbacci en 1973). Il générait toutes les transformations possibles parmi lesquelles il choisissait la meilleure. Cette méthode a l'avantage d'être efficace car elle est exhaustive, mais elle comporte un risque d'explosion combinatoire lorsque la taille du problème augmente. C'est ce qui fait qu'elle n'est pas pratique, car son temps de calcul est toujours exponentiel.

En pratique, tous les systèmes utilisent des heuristiques. Ces heuristiques servent à choisir seulement les transformations qui rapprochent le plus du but [BCMO88], [PoRa89] et [PLNG90]. Ces heuristiques permettent de réduire la complexité du problème sans toutefois garantir l'obtention d'une solution optimale. Ce premier type d'algorithme a l'avantage de la globalité mais sa grande complexité de calcul explique l'apparition des méthodes dites itératives.

4.4.2 Algorithmes itératifs/constructifs

Le principe de ces algorithmes est de construire l'ordre des opérations au fur et à mesure. L'algorithme sélectionne une opération à chaque itération jusqu'à ce que toutes les opérations soient assignées. La nécéssité de sélectionner une opération à chaque étape explique l'apparition de nombreuses techniques de sélection et l'importance de l'inventaire d'algorithmes itératifs existants. On peut cependant décrire tous ces algorithmes par deux facteurs déterminants: comment sélectionner une opération à l'itération courante ? et comment l'assigner à une étape de contrôle?. Il y a quatre classes d'algorithmes constructifs/itératifs: ASAP (ALAP), l'ordonnancement par liste (List-scheduling), l'ordonnancement basé sur la liberté (freedom-based scheduling) et l'ordonnancement orienté selon la force (force-directed scheduling).

4.4.2.1 Ordonancement ASAP (As Soon As Possible) [ALAP: As Late As Possible]

ASAP et ALAP sont les types d'algorithmes d'ordonnancement les plus simples. Ce type d'ordonnancement est local tant à l'égard de la sélection d'une opération qu'à l'égard de l'endroit où l'opération est assignée. Les opérations sont ordonnées dans l'ordre topologique (selon l'ordre du graphe de flux de contrôle). Les opérations sont considérées selon cet ordre et chacune des opérations est assignée à la première (la dernière pour ALAP) étape imposée par les contraintes de ce type d'ordonnancement est de n'accorder aucune priorité aux opérations sur le chemin critique. L'assignation des opérations moins critiques en premier lieu avec en plus une limite sur les ressources peut bloquer les opérations critiques, ce qui a pour effet d'augmenter le nombre d'étapes de contrôle nécéssaires.

4.4.2.2 Ordonnancement par liste (" Liste Scheduling ")

Pour compenser l'inconvénient des ordonnancements ASAP et ALAP, l'ordonnancement par liste a été proposé. L'ordonnancement par liste utilise un critère de sélection plus élaboré que l'ordre topologique. On construit la liste des opérations qui peuvent être assignées à chaque étape de contrôle. La liste est ordonnée selon une fonction de priorité donnée. On considère les opérations de la liste une à une. Pour chaque opération, lorsque les ressources permettent son exécution, elle est assignée à l'étape courante, sinon son exécution est différée [PaGa87a] et [LDSSh80].

4.4.2.3 Ordonnancement basé sur la liberté ("Freedom-based Scheduling")

Lorsque l'ordonnancement est basé sur la liberté, la priorité est donnée aux opérations sur le chemin critique. Toutes ces opérations sont traitées en premier lieu. Les autres opérations sont ensuite traitées une à la fois [PPM186]. A chaque itération, l'opération la moins libre est choisie (celle qui comporte le moins d'étapes de contrôle là où elle peut être placée).

4.4.2.4 Ordonnancement orienté selon la force ("Forced-directed Scheduling")

Dans cet ordonnancement, une fonction de force est associée à chaque opération. La différence entre une fonction de priorité et une fonction de force est que la fonction de force est plus globale. Un exemple simple de fonction de force: le nombre d'étapes de contrôle possibles pour une opération [PaKn87], [PaKn89] et [CITh90]. Ces valeurs serviront à construire un graphe de distribution indiquant la charge de chaque étape de contrôle (nombre maximum d'opérations dans l'étape). Les opérations sont ensuite assignées selon leur force de façon à équilibrer la distribution sur le graphe. Cette approche permet d'aborder le problème d'une façon beaucoup plus globale, mais le résultat reste dépendant de la fonction de force considérée.

Les algorithmes itératifs/constructifs ont l'avantage d'être plus simples, mais leur simplicité se paie souvent par un manque de globalité comparativement aux algorithmes transformationnels et à la programmation mathématique.

4.4.3 Programmation mathématique

De nombreuses études proposent comme solution au problème d'ordonnancement de le formuler comme un programme mathématique en nombres entiers. Un programme mathématique est simplement une forme générale d'un problème d'optimisation avec contraintes. Il existe des programmes mathématiques capables de trouver des solutions optimales et il serait tentant de les utiliser pour l'ordonnancement. Cependant, en pratique, ces programmes ne sont applicables qu'aux problèmes de taille raisonnable [Fren82].

La théorie et les programmes désignés sous le nom de la programmation mathématique sont destinés aux problèmes formulés comme suit:

Minimiser la fonction F(x1, x2, x3, ... , xn) en considérant les contraintes suivantes sur les variables x1, x2, x3, ... , xn :

$$g1(x1, x2, \ldots, xn) \leq b1$$
$$g2(x1, x2, \ldots, xn) \leq b2$$

$$\ldots\ldots$$
$$gk(x1, x2, \ldots, xn) \leq bk$$

Où x1, x2, ..., xn sont des variables indépendantes et F, g1, g2, ... , gn sont des fonctions linéaires.

Dans la programmation en nombres entiers, un ensemble de variables indépendantes sont contraintes à être des entiers. En général, ces valeurs entières sont restreintes aux valeurs 0 ou 1 pour signifier l'absence ou la présence d'une certaine propriété. Comment cette technique est-elle appliquée au problème de l'ordonnancement dans la synthèse de haut niveau?
Une formalisation tirée de [LHLi89], [PaKo90] et [HHLi90] est présentée ci-dessous.

Etant donné la liste des opérations Oi du graphe de flux contrôle, le coût d'une unité fonctionnelle de type Ti est Cti et disposant de m types d'unités fonctionnelles, les variables utilisées sont:

Mti : Variable entière indiquant le nombre d'unités fonctionnelles de type ti requises.

xij : Variable entière (0,1) associée à l'opération Oi telle que:

$Xij = 1$ Si i est assignée à j
$Xij = 0$ Sinon

Le problème est alors formulé comme suit:

Il faut minimiser:

$$(1) \quad \sum_{i=1}^{m} Cti \times Mti$$

en respectant les contraintes suivantes:

$$(2) \quad \sum_{i=1}^{n} xij - Mtk \leq 0 \quad , \quad 1 \leq j \leq s \text{ et } 1 \leq k \leq m$$

$$(3)\quad \sum_{j=Si}^{Li} xij = 1\ , \qquad 1 \le i \le n$$

$$(4)\quad \sum_{j=Si}^{Li} J \times xij - \sum_{j=Sk}^{Li} J \times xkj \le -1$$

Si Oi est le prédécesseur immédiat de Ok.

où:

(1) signifie, minimiser la fonction du coût total.

(2) : on ne doit pas utiliser plus de Mtk unités fonctionnelles de type tk dans n'importe quelle étape de contrôle.

(3) : une opération ne peut être assignée qu'à une étape de contrôle entre Si et Li (partitions ASAP et ALAP).

(4) : On considère la préservation des relations de précédence dans le graphe.

Après la formulation du problème d'ordonnancement dans la programmation mathématique, il s'agit de minimiser la fonction de coût. Une des méthodes de minimisation de la fonction de coût telle que formulée dans le paragraphe précédent est la méthode du gradient utilisée dans [ShWo89], pour le problème de l'ordonnancement.

4.5 Techniques d'allocation

Une technique d'allocation des ressources matérielles consiste à trouver une assignation des opérations, des variables et des chemins de données aux unités de traitement, aux registres et aux interconnexions respectivement. Cette assignation est généralement réalisée avec une optimisation de délai ou de matériel afin de satisfaire les contraintes imposées par le concepteur.

De même que pour l'ordonnancement, de nombreuses techniques ont été développées pour l'allocation. Ces techniques sont classées en 2 classes principales: l'allocation constructive et l'allocation globale.

4.5.1 Allocation itérative/constructive

Cette technique d'allocation permet de sélectionner un élément à allouer à chaque itération du processus d'allocation, jusqu'à ce que tous les éléments virtuels du graphe soient assignés à des ressources matérielles. On distingue trois types d'éléments virtuels: Les opérations, les variables et les transferts de données.

Les méthodes itératives ont toujours besoin d'une technique de sélection. Plusieurs heuristiques de sélection ont été utilisées qui varient des sélections les plus locales jusqu'aux plus globales.

La sélection locale considère les éléments à allouer selon leur ordre d'apparition dans le graphe tandis que les sélections globales utilisent toujours une métrique de sélection plus globale. Le coût additionnel causé par l'allocation d'un élément à une unité de traitement donnée est un exemple de métrique.

La plupart des systèmes de synthèse utilisent cette approche comme CMUDA, [HaPa82], EMUCS [HiTh83], DAA [KGWF85], REAL [KuPa87], SPLICER [Pang] et CHARM [Woo90].

La propriété de construction et d'itération rend ce type d'allocation plus locale, même si la sélection est globale, parce que l'interdépendance des éléments n'est pas considérée.

4.5.2 Allocation globale

Afin de prendre en compte l'interdépendance des éléments à allouer, des méthodes globales ont été proposées. L'allocation globale considère plusieurs possibilités d'allocation simultanément. La recherche exhaustive peut être vue comme un cas extrême de l'allocation globale.

Ce type d'allocation est basé sur la théorie des graphes. Le principe consiste à formuler le problème de l'allocation en un graphe. Un certain traitement est alors appliqué à ce graphe pour trouver les ressources requises. Un graphe est construit pour chacun des types d'éléments à allouer, graphe des opérations, graphe des variables et graphe des interconnexions. Les noeuds du graphe sont les éléments à allouer et les arêtes sont créées de façon à relier deux éléments qui peuvent être assignés à la même ressource physique (unité de traitement, registre, bus ou multiplexeur).

L'allocation est réalisée en recherchant le nombre minimal de sous-graphes complets dans le graphe construit. Les noeuds du même sous-graphe sont assignés à la même ressource physique. Cependant, il est prouvé que le problème de la recherche de sous-graphes complets dans un graphe est un problème NP-Complet [GaJo79]. Des heuristiques sont toujours utilisées comme dans le système Emerald [TsSi86]. Le champ d'exploration reste ouvert pour trouver des heuristiques de plus en plus simples et efficaces.

Ces deux approches permettent d'assigner les éléments virtuels du graphe de flux de données ("*data flow* ") à des éléments physiques. Ces éléments physiques constitueront le chemin de données ("*data path* ") qui réalise le comportement désiré. Un "data path" est constitué principalement de trois composantes : les unités de traitement, les registres pour stocker les données et les interconnexions pour connecter ces éléments les uns aux autres. Les interconnexions peuvent être réalisées avec deux types d'éléments physiques, les bus et les multiplexeurs.

Lorsque l'architecture est implémentée à base de multiplexeurs, on associe à chaque unité de traitement trois multiplexeurs, un multiplexeur à sa sortie pour transférer son résultat aux différents destinataires et deux multiplexeurs à ses entrées pour ramener les données des différentes sources. On associe aussi un démultiplexeur à chaque registre lorsqu'il est mis à jour par plusieurs sources.

Lorsque le nombre des multiplexeurs devient considérable, une fusion de plusieurs multiplexeurs en un bus est réalisée puisqu'un bus peut jouer le rôle de multiplexeur et de démultiplexeur au même temps.

Selon le modèle d'interconnexion choisi, on distingue une variété de systèmes de synthèse. Parmi eux, SPLICER [Pang88], qui génère des architectures à base de multiplexeurs et [DeNe89] qui génère des parties opératives à base de bus. Des modèles d'interconnexion plus complexes ont été introduits comme l'allocation, dans la même architecture, de bus ou de multiplexeurs selon le coût induit comme dans MABAl, [KuPa90], ou l'allocation de bus partitionnés pour simplifier le shéma des interconnexions comme dans [Ewer90].

En plus du modèle d'interconnexion, il existe des systèmes qui génèrent des architectures pouvant s'exécuter en "pipeline" pour minimiser le matériel requis comme celui de SEHWA [PaPa86] et HAL89 [PaKn89].

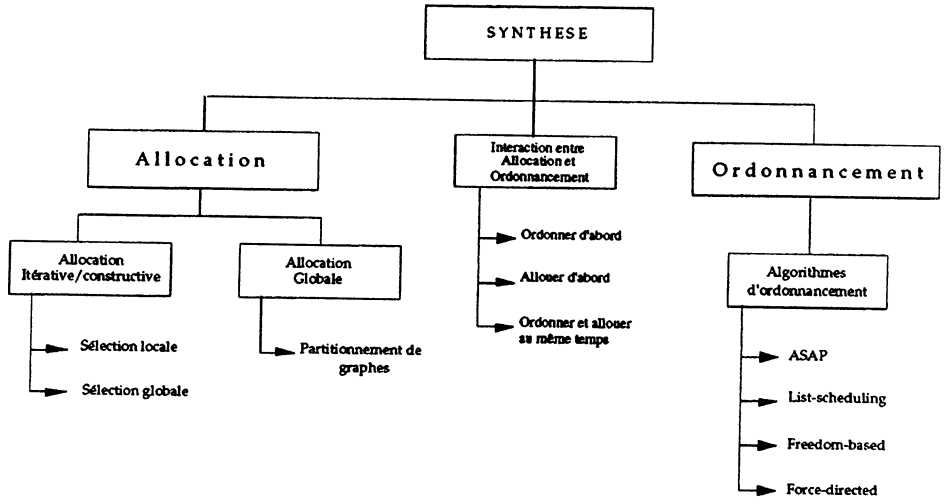


figure 6: techniques de synthèse

4.6 Fonction de coût

Afin de considérer les N compromis lors du choix d'une implémentation dans un processus de synthèse pour respecter au mieux les contraintes imposées (temps d'exécution, surface, ...), il faut pouvoir les considérer tous au même temps. Ceci est pratiquement impossible vu la complexité et la lenteur introduites dans un système de synthèse.

Cependant, pour limiter l'espace de recherche à explorer, tous les systèmes de synthèse, même les plus récents, ne s'intéressent généralement qu'à un nombre limité de paramètres. Dans la plupart des cas, deux paramètres sont considérés: le temps d'exécution et la surface.

On pourrait penser que le fait de ne considérer que deux paramètres simplifie considérablement le problème, mais il n'en est rien. En effet, calculer avec précision le délai ou la surface d'un circuit intégré peut s'avérer une tâche très complexe et très longue à accomplir.

Ainsi, en plus des techniques d'ordonnancement et d'allocation, les techniques d'évaluation de la surface et du temps d'exécution constituent un autre facteur de différenciation entre les systèmes de synthèse. L'évaluation du coût d'une architecture est très importante, puisqu'elle déterminera les décisions ultérieures.

On présentera dans ce qui suit, un ensemble de techniques les plus connues utilisées pour évaluer le coût d'un système.

4.6.1 Evaluation de la surface

Différentes techniques ont été utilisées pour évaluer la surface d'une architecture. La technique la plus primitive consiste à évaluer le nombre d'unités de traitement et à les multiplier par le coût unitaire de l'unité de traitement la plus

chère. L'objectif à atteindre lors de la synthèse est de ne pas dépasser une limite maximale pour la surface ou ce qui est souvent formulé par :

Nombre d'unités de traitement * Coût unitaire ≤ Smax

Cette première technique sous-estime considérablement la surface, puisque seules les unités de traitement sont prises en considération.

Une autre approche utilise le nombre de connexions (bus et multiplexeurs) comme mesure de la surface étant donné que les interconnexions constituent une grande partie de la surface d'une puce. Cette méthode a été utilisée dans les systèmes présentés dans [PaGa87a] et [PaGa87b]. Cependant, la surface reste sous-estimée car si les interconnexions constituent une proportion importante de la surface, elles ne sont pas la totalité de la surface.

Une considération plus détaillée de la surface a été faite dans [DeNe89], où l'on tient compte du nombre d'unités de traitement, de registres et de bus. Chacun des facteurs a été multiplié par le coût unitaire du type de l'élément. La surface s'exprime alors comme suit:

W1 X #UAL + W X #REG + W3 X #BUS

où #UAL, #REG et #BUS sont respectivement le nombre d'unités de traitement, de registres et de bus et W1, W2 et W3, les coûts unitaires d'une UAL, d'un registre et d'un bus.

On remarque que l'évaluation de la surface est ici beaucoup plus détaillée que dans les cas précédents, mais elle reste abstraite. Il faut être capable d'associer à W1, W2 et W3 des coûts qui correspondent le plus possible à la réalité. L'une des approches les plus détaillées a été présentée dans [FaKo90]. Le système déduit un plan de masse en réalisant des partitionnements physiques des différents éléments. La surface est alors évaluée selon ce plan de masse. Cette évaluation est la technique la plus proche de la réalité. En effet, seul un plan de masse peut nous donner une idée de la position des éléments et de la longueur des interconnexions.

4.6.2 Evaluation du temps d'exécution

Deux techniques principales sont utilisées pour évaluer le temps d'exécution d'une architecture. La première est basée sur l'idée que le temps d'exécution correspond au produit arithmétique du nombre d'étapes de contrôle par le temps du cycle machine. Le cycle machine peut être déduit par le système comme étant le temps d'exécution de l'opération la plus lente comme dans [DeNe89]. Cette évaluation est grossière et ne reflète pas la réalité. Une autre manière de définir le

cycle est de laisser le concepteur le fixer; le système s'arrangera alors pour réduire au minimum le nombre des étapes de contrôle. Cette minimisation est réalisée à l'aide des techniques de multicyclage et de chaînage. La technique de chaînage est utilisée pour les cycles longs car elle permet d'exécuter plusieurs opérations rapides durant le même cycle. Mais lorsque le cycle est court, la technique de multicyclage est utilisée; elle permet d'exécuter une opération lente sur plusieurs cycles [PaGa87a].
Une autre technique d'évaluation du temps d'exécution consiste à faire une estimation du délai en considérant un plan de masse. Malgré sa complexité, elle est la technique la plus efficace. Il est prouvé que le délai à travers les interconnexions est souvent plus critique que celui des blocs. On ne peut donc faire un calcul de délai en négligeant les fils; cela serait trop abstrait.

4.6.3 Evaluation de la testabilité

Le circuit conçu manuellement et dont on n'a pas vérifié sa testabilité peut s'avérer non testable après sa fabrication. Il devient donc très important pour les concepteurs que les circuits soient testables dès le départ. Lors d'une conception manuelle, beaucoup d'outils sont mis à la disposition du concepteur afin qu'il puisse vérifier la testabilité d'un circuit avant de le fabriquer. L'intégration de la testabilité dans les systèmes générateurs de layout a été prise en compte il y a déjà quelques années dans quelques systèmes comme [ChBr85], [AbBr85] et [FuHi86]. Il est donc important qu'un outil de synthèse se préoccupe de générer des circuits qui soient testables à sa sortie. Cependant, ce facteur a été négligé dans tous les systèmes de synthèse jusqu'en 1985 lorsque Gebotys et Elmasry ont commencé à intégrer cet aspect dans leurs recherches.

Cependant, comment allons-nous mesurer la testabilité d'un circuit à un haut niveau comme l'architecture, sachant que tous les outils développés pour la testabilité traitent un circuit au niveau transistors ou portes logiques, mais pour l'instant sans aller plus haut. L'évaluation à un niveau haut de la testabilité d'un circuit est un domaine de recherche qui reste à explorer. La seule technique disponible à l'heure actuelle est celle utilisée dans [GeEl88] et [GeEl89], où l'on détermine la testabilité après que le système de synthèse ait généré le circuit au niveau portes logiques. La testabilité est alors mesurée comme la couverture d'erreurs et le nombre de vecteurs de test requis. Un autre paramètre pourrait être intégré à cette mesure : le coût additionnel en surface et le délai, pour rendre le circuit testable. Ce coût additionnel est dû au fait que pour rendre un circuit testable, on doit toujours augmenter la circuiterie, ce qui entraine une augmentation de la surface et du délai.

4.6.4 Interfaçage avec l'extérieur

Le circuit produit par un système de synthèse doit être facilement utilisable dans son environnement d'implémentation. Généralement, les systèmes de synthèse n'offrent pas la possibilité de spécifier le comportement exact du circuit au niveau des interactions entre les signaux échangés avec l'extérieur. Lors de l'implémentation du circuit généré, il est souvent complété par une logique externe d'adaptation afin de satisfaire les contraintes d'interface.

Quelques travaux ont été entamés dans [NeTh86] [BoKa87] [BoKa87] [Borr88] pour spécifier à l'aide d'un langage les contraintes temporelles et électriques des signaux d'interface.

4.7 Techniques de Raffinement d'une Conception

Lorsqu'une architecture est générée par un système de synthèse en respectant le comportement demandé, elle peut dans certains cas ne pas satisfaire les contraintes imposées au départ par le concepteur; dans ce cas, le système doit reboucler jusqu'à la satisfaction de ces contraintes. Il existe trois principales techniques pour raffiner une conception : un retour guidé par le concepteur, un retour par l'intelligence artificielle ou une boucle aléatoire.

4.7.1 Boucle concepteur

Dans ce cas, c'est le concepteur lui-même qui fera les modifications adéquates sur le circuit dès que le système de synthèse le visualise. Ces modifications peuvent se faire à différents niveaux selon la gravité du dépassement par rapport aux contraintes. Le concepteur peut demander, à un bas niveau, une autre génération de layout, par exemple, l'utilisation d'un autre type (un additionneur à anticipation de retenue au lieu d'un additionneur "carry-save"), ou bien, changer la description de départ pour rajouter ou enlever des contraintes entre opérations, à un niveau plus haut. Cette technique exige une bonne interface graphique au niveau de la réalisation de l'interface usager. Elle a été appliquée dans le système EMUCS [HiTh83].

4.7.2 Boucle par l'intelligence artificielle

Une technique plus souple qui n'exige pas l'intervention de l'usager, consiste à intégrer au système de synthèse une base de connaissance munie d'un moteur d'inférence qui prendra les décisions nécessaires selon les cas d'insatisfaction. Cette technique exige le rassemblement de toutes les techniques possibles pour réaliser la modification. La base de connaissance s'appuie donc sur les connaissances des concepteurs et la technique mise beaucoup sur leur collaboration. L'avantage de cette

technique réside dans sa souplesse: la base de connaissances peut être mise à jour et enrichie continuellement. Le système DAA [KGWF85] utilise cette technique et il a été enrichi pendant des années à partir des descriptions en entrée et des propositions des experts.

4.7.3 Boucle non-déterministe

Une autre façon de réaliser tout le processus de synthèse est de formuler le coût d'une architecture par une somme pondérée comme dans [DeNe89], puis de minimiser cette fonction à l'aide d'un algorithme de recuit simulé. L'algorithme de recuit simulé consiste à générer une configuration ou une architecture, à évaluer son coût et à considérer ensuite cette solution lors de la prochaine itération si le coût est amélioré par rapport au coût courant, sinon à accepter la nouvelle solution avec une probabilité donnée [KGVe83]. L'algorithme est répété jusqu'à satisfaction des contraintes ou jusqu'à ce qu'on ne puisse améliorer davantage le design. Cette technique a été utilisée par Devadas et Newton dans [DeNe89].

Le principe de la sélection naturelle a aussi été utilisé pour développer un algorithme, basé sur la mutation et la sélection, pour générer des solutions aléatoirement. Cet algorithme a été utilisé, pour le problème d'ordonnancement et d'allocation dans [WGHe90].

Conclusion

La dernière étape de la synthèse consiste à choisir la structure interne de chaque module alloué, pour son implantation sur silicium. C'est le "Module Binding" ou assignation de modules. Cette étape est suivie par la génération de la description physique des différents éléments alloués. Cette tâche peut être réalisée par des outils de bas niveau connus et efficaces, tels que les outils de synthèse logique et les générateurs de layout.

La synthèse de haut niveau offre alors des possibilités d'exploration des solutions très souples. Ce point ne constitue pas le seul avantage de la synthèse de haut niveau car le temps de conception est considérablement réduit; le concepteur n'a plus ou presque plus à intervenir pour réaliser un traitement manuel. De plus, le circuit est correct par construction et les erreurs de conception sont détectées au plus haut niveau, ce qui facilite leur correction. Un dernier avantage de la synthèse de haut niveau est de rendre la technologie disponible à un plus grand nombre de personnes. La description en entrée est complètement indépendante de la technologie et c'est au système d'implanter le circuit dans la technologie désirée.

Plusieurs méthodes de synthèse ont été jusqu'à maintenant proposées, et sont adaptées à des applications spécifiques ou générales. Dans le premier cas, la synthèse devient orientée application (spécifique) alors que dans le deuxième cas, elle est générale.

On s'est intéressé dans notre étude à une application spécifique qui est celle de la conception des circuits de communication. Contrairement aux autres, le domaine particulier de l'implémentation des protocoles de communication a des caractéristiques spécifiques de traitement du flux de données et du flux de contrôle.

Ainsi, une approche spécifique a été adoptée pour proposer une démarche de synthèse d'architecture dédiées aux circuits de communication.

Récemment apparue dans le domaine de la synthèse des systèmes [RaGi92] ***"Hardware-Software Co-Design "***, l'automatisation de la conception des systèmes ne se limitant pas uniquement à la synthèse des architectures matérielles mais elle s'intéresse surtout à la possibilité d'implémentation logicielle des applications. Elle consiste alors, à regrouper les diverses fonctionnalités de l'application en deux classes :

- la classe des fonctions "*hardware* "
- et la classe des fonctions "*software* "

Chaque classe est alors synthétisée sous forme d'architecture, qui est matérielle pour la première et logicielle pour la seconde. Ainsi, une implémentation logicielle-matériel devient possible pour les systèmes complexes en partant d'une description de niveau système.

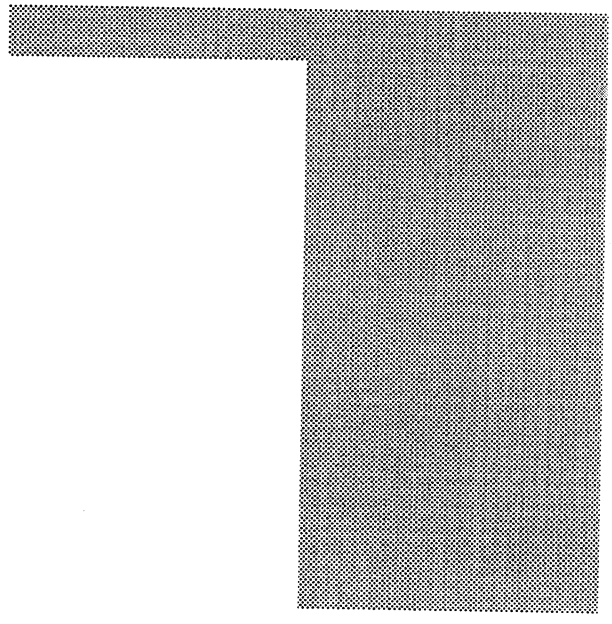
Le grand problème de la synthèse de système est comment reconnaitre et séparer les fonctionnalités dites "*Hardware*" des fonctionnalités dites "*Software*" ?

Actuellement, il existe quelques approches utilisant également des heuristiques permettant de proposer des solutions rapides mais pas efficaces [RaCG92]. Des travaux sont en cours de réflexion dans ce vaste domaine ...

Notre équipe s'oriente actuellement en partie vers ce domaine avec le démarrage des travaux sur la fiablité des systèmes mixtes "Hardware-Software".

ANNEXE B

Exemples d'Unités de Traitement



— EXEMPLES D'UNITES DE TRAITEMENT —

Nous présentons dans cette annexe quelques exemples d'unités de traitement généralement utilisés dans les circuits de communication.

Parmi ces unités, il y a:

- unité de reconnaissance de valeurs d'un champ;
- unité de protection contre les erreurs;
- unité de mémorisation de données.

Unité de Reconnaissance de Valeurs d'un champ

L'unité de reconnaissance permet d'identifier la valeur d'un champ sur une ou plusieurs valeurs attendues. L'exemple type d'utilisation est celui de la reconnaissance d'adresse où l'on peut avoir à identifier une adresse unique, une adresse de sous-groupe ou une adresse de groupe. Il s'agit alors de faire une comparaison entre le champ reçu avec les différentes valeurs.

Cette comparaison peut être effectuée de deux manières: parallèle ou série. La comparaison parallèle consiste à utiliser des comparateurs parallèles en attendant la fin de réception du champ à identifier. Ce champ est donc mémorisé et après comparé aux différentes valeurs. Par contre, la comparaison série compare au fur et à mesure l'arrivée des différents *eb* du champ à ceux des valeurs. On présentera dans ce qui suit l'architecture de la comparaison série pour les différents cas suivants: reconnaissance d'une valeur, reconnaissance de plusieurs valeurs et utilisation d'un masque de valeurs.

a) Reconnaissance d'une Valeur

La reconnaissance d'une valeur est basée sur l'utilisation d'une cellule comparateur de bit en série(figure 1). Elle est constituée d'un ou exclusif et d'une bascule D. La porte logique ET permet de prendre en compte le résultat de comparaison des bits précédents.

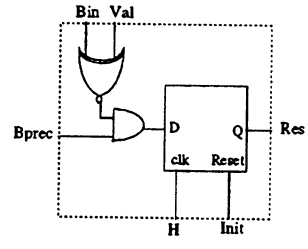


figure 1 : cellule comparaison bit

Dans le cas d'une valeur de plusieurs bits, l'entrée résultat précédent de la première cellule est positionné à "1" (figure 2).

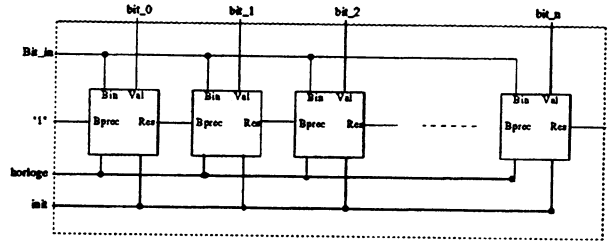


figure 2: comparateur sériel d'une valeur

b) Reconnaissance de Plusieurs Valeurs

Dans le cas où l'on cherche à reconnaître plusieurs valeurs, alors plusieurs comparateurs séries sont utilisés et chacun correspondant à une valeur. La même entrée série du champ à comparer est utilisée pour tous les comparateurs (figure 3).

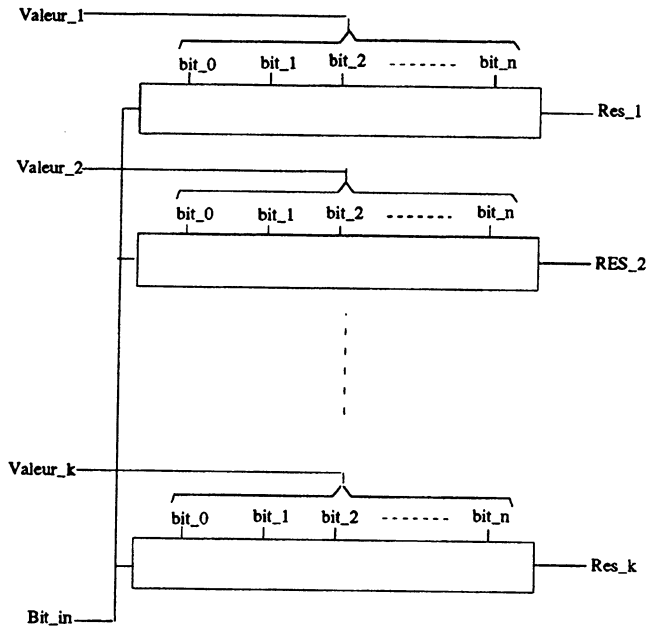


figure 3: comparateur sériel de plusieurs valeurs

Un ou plusieurs résultats sont pris en compte par la partie contrôle. Un OU logique entre les différents résultats permet d'indiquer si une valeur est identifiée ou pas.

c) Utilisation d'un Masque d'Eléments Binaires

Un masque de certains *eb* des différentes valeurs à identifier peut être utilisé. Dans la figure 4, on présente un exemple de masquage de *eb* qui est figé

pour une seule valeur du masqua "0010". La présence d'un "0" indique que le *eb* doit être identifié et dans le cas contraire sa valeur est ignorée. Ainsi la troisième cellule est réduite à une simple bascule de temporisation.

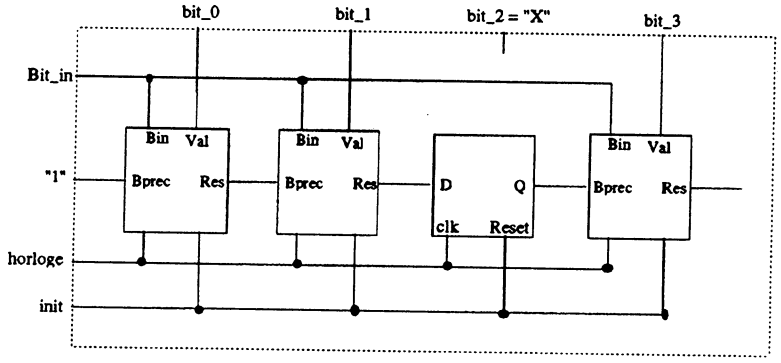


figure 4: masque de *eb*

Ce masque peut être programmable par la station hôte en utilisant un registre spécifique accédé en écriture à partir de l'extérieur. Dans ce cas, la cellule de base doit tenir compte de la valeur du masque en rajoutant un OU logique au résultat de la comparaison. Alors, si la valeur du masque est à '0', le résultat doit être tenu en compte et dans le cas contraire ('1'), le résultat est toujours à '1'.

Les différentes valeurs qui doivent être reconnues peuvent aussi être programmables par l'utilisateur. Pour cela un ensemble de registres est utilisé dans la partie bufferisation de l'interface spécialement pour y stocker ces différentes valeurs. Ces registres sont mis à jour à chaque fois qu'on change de valeurs et sont accessibles en écriture à partir de l'extérieur.

Unité de Protection Contre les Erreurs

Plusieurs techniques de protections contre les erreurs existent et visent essentiellement à détecter ces erreurs et éventuellement à les corriger. La technique de Contrôle de Redondance Cyclique (CRC pour Cyclic Redundancy Check) utilise les codes polynomiaux cycliques. Elle est souvent utilisée dans le protocoles de niveau bas des réseaux locaux. Le principe de fonctionnement est bien présenté dans [RaG88]. On décrit dans la suite les architecture associées aux circuits codeur et décodeur CRC.

Ce sont généralement des unités de traitement série sur les bits d'un ou de plusieurs champs. Les unités de traitements parallèles ont une architecture plus complexes et ne sont donc pas utilisées fréquement dans durant l'implémentation.

(a) Unité de Codage CRC

Le circuit codeur CRC est utilisé dans la machine d'émission. Il est construit selon le polynôme générateur p(x) associé. Le choix de ce polynôme dépend de la couverture d'erreurs voulu et du taux d'erreurs sur la ligne.

L'exemple de la figure 5, représente l'implémentation du polynôme p(x) tel que :

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1$$

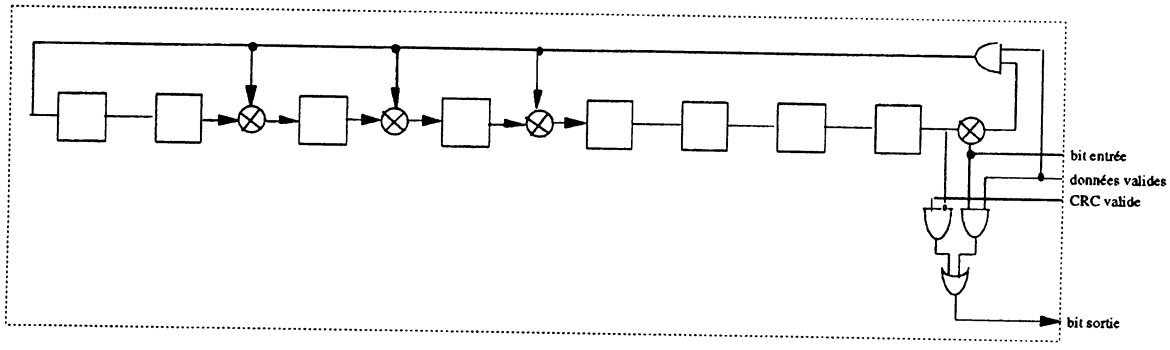


figure 5 : codeur CRC

Le circuit est constitué d'un ensemble de bascules correspondant au degré du polynôme générateur. Ces bascules sont reliées en registre à décalage qui sont éventuellement séparées par des portes OU exclusives. Le placement de ces

portes dépend du degré du monôme associé et leur nombre est égal au nombre de termes du polynôme moins un.

(b) Unité de Décodage CRC

L'unité de traitement du décodage CRC est utilisé dans la machine de réception et associé au même polynôme générateur de celui utilisé dans la machine d'émission.

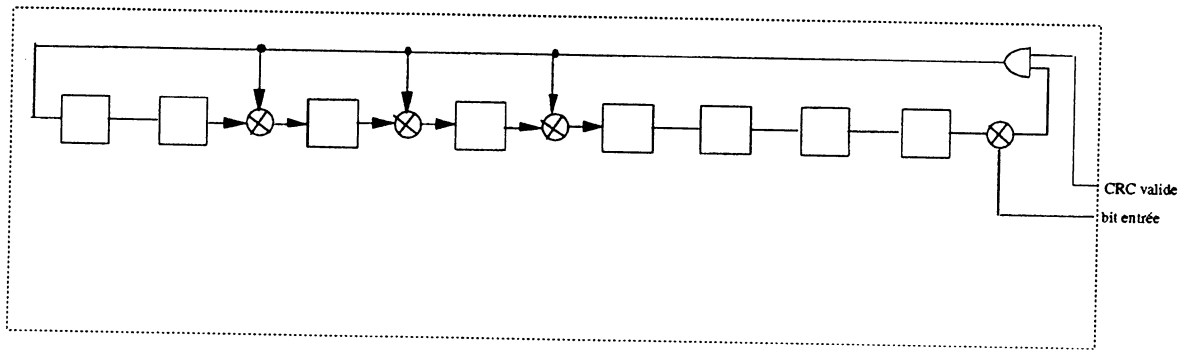


figure 6: décodeur CRC

Le fonctionnement global consiste à initialiser, avant l'émission, le registre à décalage à '1' et à envoyer le message suivi de l'inverse (complément à '1') du reste de la division. Le reste de la division en réception sera toujours égal à un polynôme unique (appelé polynôme reste) qui dépend du choix du polynôme générateur.

Unité de Mémorisation de Données — FIFO

Dans la partie bufferisation des données, des FIFOs peuvent être utilisées pour le stockage des données que ce soit en émission ou en réception [Sah91]. Le schéma général du module FIFO est représenté dans la figure 7.

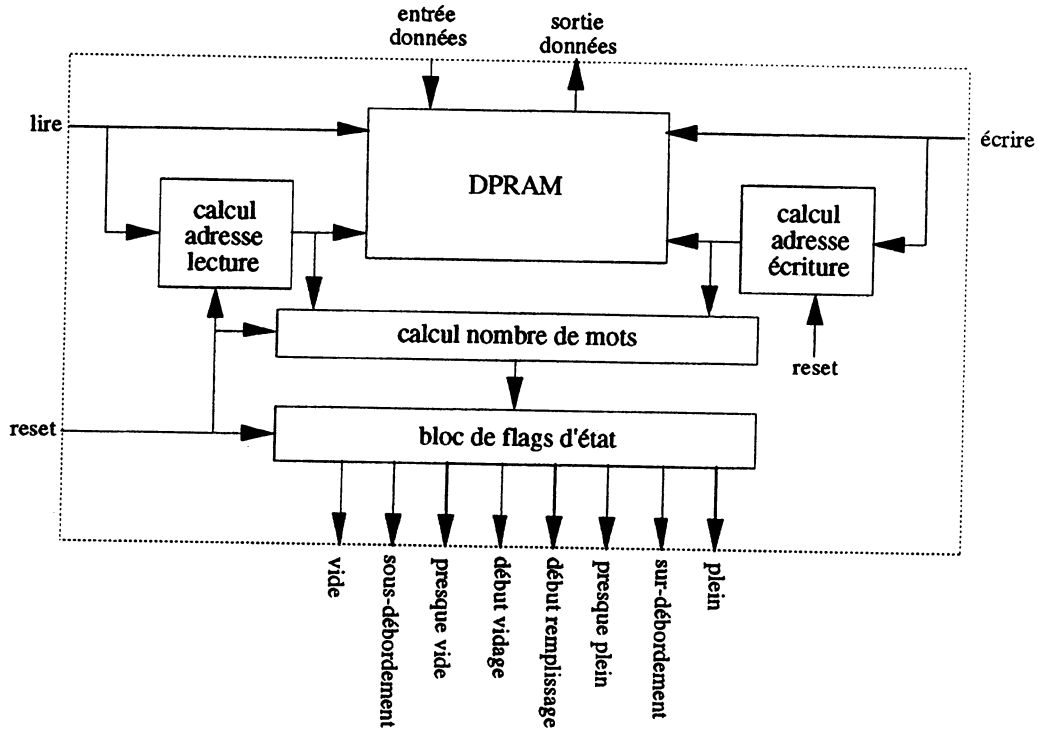


figure 7 : architecture du module FIFO

Ce module est constitué essentiellement de cinq blocs :

• Une mémoire de données (DPRAM) :

Elle constitue la zone mémoire dans laquelle seront stockées les données. C'est une mémoire à double ports qui permet d'effectuer des accès simultanés (en lecture et en écriture). La première donnée écrite dans la mémoire est la première lue.

• Un bloc de calcul d'adresse d'écriture :

Ce bloc permet de fournir l'adresse du prochain mot à écrire dans cette mémoire. L'adresse est calculée en rajoutant un UN à l'adresse courante avec un modulo la taille de la mémoire. Ainsi, on a une mémoire "circulaire" où l'adresse "0" suit systématiquement l'adresse " taille mémoire - 1". Pour cela, un compteur d'adresse est utilisé et initialisé à zéro au début de l'utilisation du FIFO. Il est incrémenté à chaque demande d'écriture et passe à zéro dès qu'il atteint la taille de la mémoire.

• Un bloc de calcul d'adresse de lecture :

Ce bloc réalise la même fonction que celle du précédent. Il est aussi composé d'un compteur qui est initialisé à zéro au début de l'utilisation du FIFO.

• Un bloc de calcul du nombre de mots :

Ce bloc permet de calculer régulièrement le nombre de mots que la mémoire contient en réalisant la différence entre les deux pointeurs d'adresse (de lecture et d'écriture).

• Un bloc de positionnement des flags d'état :

Ce bloc fournit les indications nécessaires pour connaître l'état de la FIFO. Ceci est réalisé en positionnant des flags correspondant à chaque état. On peut recenser huit types de flags sans pour autant les utiliser tous.

Ils peuvent être classés selon le processus de lecture ou d'écriture:

(a) les flags de lecture :

. vide : indique que le FIFO est vide. Ceci est connu dès que le nombre de mots calculé est égal à zéro. Dans ce cas le processus de lecture doit s'arrêter.

. presque vide : indique qu'on a atteint un seuil où le FIFO est presque vide. Il permet d'activer le processus d'écriture afin que le FIFO ne soit pas vide.

. début vidage : indique que le FIFO contient quelques données qui peuvent être lues dès maintenant pour équilibrer la différence de débits entre le processus d'écriture et le processus de lecture.

. sous-débordement : Ce flag est positionné lorsqu'on fait une lecture alors que le FIFO est vide.

(b) les flags d'écriture :

. plein : indique que le FIFO est plein. Ceci est connu dès que le nombre de mots calculé est égal au maximum. Dans ce cas le processus d'écriture doit s'arrêter.

. presque plein : indique qu'on a atteint un seuil où le FIFO est presque plein. Il permet d'activer le processus de lecture afin que le FIFO ne soit pas plein.

. début remplissage : indique que le FIFO contient quelques espaces vides qui peuvent être remplis dès maintenant pour équilibrer la différence de débits entre le processus de lecture et le processus d'écriture.

. sur-débordement : Ce flag est positionné lorsqu'on fait une écriture alors que le FIFO est plein.

Un ensemble de paramètres est nécessaire pour construire une FIFO. Parmi ces paramètres, on a la largeur du FIFO (ou le nombre de bits par mots), la profondeur du FIFO (ou le nombre de mots du FIFO) et les différentes valeurs des différents seuils.

Unité d'Interface avec la Partie Bufferisation

Afin de lire ou écrire dans un registres ou dans un FIFO de la partie bufferisation, une unité d'interface est utilisée pour gérer le transfert des données de lecture et d'écriture.

Pendant l'émission, l'unité est chargée d'activer la lecture des données à envoyer. Dans le cas d'une FIFO, celle-ci consiste en un vidage des mots et de les sérialiser. Par contre pour un registre, la donnée est déjà disponible à sa sortie et il ne reste plus qu'à la sérialiser. On présentera dans ce qui suit le cas d'une unité d'interface FIFO (figure 38).

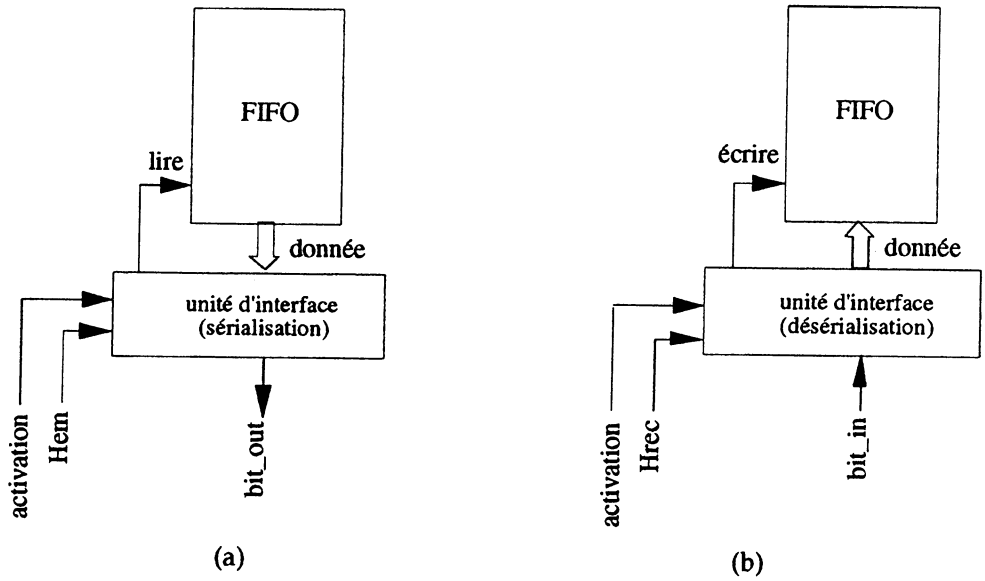


figure 8: unité d'interface avec le module FIFO

(a) en émission

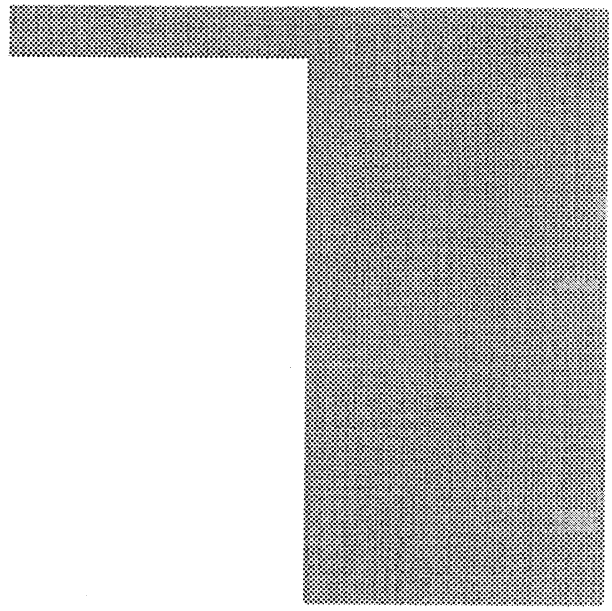
(b) en réception

Soit en émission ou en réception, selon la taille des données à transférer et la largeur de la FIFO, plusieurs lectures (cas de l'émission) ou écritures (cas de la réception) doivent êtres activées pour terminer le transfert de toutes les données. Généralement, la taille des données est un multiple de la largeur de la FIFO.

Les signaux d'activation sont générés par la partie contrôle d'émission dans le cas d'une lecture ou de réception dans le cas d'une écriture. On peut avoir plusieurs éléments de mémorisation (FIFOs ou registres) qui peuvent accéder à la même unité de traitement. Ceci est réalisé en utilisant un multiplexage des lignes de sortie des données des différents éléments. Le détail d'implémentation est présenté dans le chapitre suivant.

ANNEXE C

Grammaire de MACS



— GRAMMAIRE DE MACS —

Nous présentons dans cette annexe la grammaire de description de MACS. Elle a été utilisée par YACC pour la génération de l'analyseur syntaxique et sémantique.

Un tableau récapitulatif des instructions de base pour la description des processus d'émission et de réception est ensuite présenté.

```
"p : /* empty */",
"p : PROTOCOL IDENT body ENDPROTOCOL",
"body : buff adapt func_unit field interf_bas emit recept caract",
"buff : BUFFERISATION inst_buff ENDBUFF",
"inst_buff : /* empty */",
"inst_buff : IDENT '[' NUM ']' ':' sens REG ';' inst_buff",
"inst_buff : IDENT '[' NUM ',' NUM ']' ':' sens FIFO ';' inst_buff",
"adapt : ADAPTATION PORT '(' dec_port ')' BEGIN inst_adapt ENDADAPT",
"dec_port : /* empty */",
"dec_port : dec_port_inst dec_port",
"dec_port_inst : IDENT '[' NUM ']' ':' sens DATA ';'",
"dec_port_inst : IDENT ':' sens CONTROL ';'",
"inst_adapt : /* empty */",
"inst_adapt : sens '(' IDENT ',' IDENT ')' '=' inst_logic ';'
              inst_adapt",
"inst_adapt : RESET '(' x_ident ')' '=' inst_logic ';' inst_adapt",
"inst_adapt : sens '(' IDENT ',' IDENT ')' '=' const ';' inst_adapt",
"inst_adapt : RESET '(' x_ident ')' '=' const ';' inst_adapt",
"const : '"' binaire '"'",
"binaire : /* empty */",
"binaire : NUM binaire",
"inst_logic : IDENT",
"inst_logic : IDENT '(' IDENT ')'",
"inst_logic : IDENT '[' NUM ']'",
"inst_logic : inst_logic '+' inst_logic",
"inst_logic : inst_logic '.' inst_logic",
"inst_logic : '(' inst_logic ')'",
"inst_logic : '/' inst_logic",
"interf_bas : INTERF_BAS EMISSION inst_int_bas ENDEMISSION RECEPTION
              inst_int_bas ENDRECEPTION ENDINTERF_BAS",
"inst_int_bas : /* empty */",
"inst_int_bas : IDENT ':' IDENT ';' inst_int_bas",
"inst_int_bas : IDENT ':' DATA ';' inst_int_bas",
"inst_int_bas : IDENT ':' sens CONTROL ';' inst_int_bas",
"sens : IN",
"sens : OUT",
"func_unit : FUNC_UNIT inst_unit ENDFUNC_UNIT",
"inst_unit : /* empty */",
"inst_unit : IDENT '(' inst_unit_funct ')' inst_unit",
```

```
"inst_unit_funct : /* empty */",
"inst_unit_funct : IDENT '[' NUM ']' ':' sens DATA ';'
                   inst_unit_funct",
"inst_unit_funct : IDENT ':' unit_type ';' inst_unit_funct",
"unit_type : BIT_IN",
"unit_type : BIT_OUT",
"unit_type : CLK",
"unit_type : RESET",
"field : FIELD DATA_FIELD inst_field1 ENDDATA SYNCHRO inst_field2
                ENDSYNCHRO ENDFIELD",
"inst_field1 : /* empty */",
"inst_field1 : IDENT '[' NUM ']' ';' inst_field1",
"inst_field1 : IDENT '[' NUM ',' NUM ']' ';' inst_field1",
"inst_field2 : /* empty */",
"inst_field2 : IDENT '[' NUM ']' ';' inst_field2",
"inst_field2 : IDENT '[' NUM ',' NUM ']' ';' inst_field2",
"emit : TRANSMIT GENERAL inst_gen ENDGEN BEGIN inst_emit ENDTRANSMIT",
"inst_gen : /* empty */",
"inst_gen : STATES '(' x_ident ')' ';' inst_gen",
"inst_gen : INIT_LOC '(' inst_logic ',' affect ')' ';' inst_gen",
"inst_gen : INIT_LOC '(' inst_logic ')' ';' inst_gen",
"inst_gen : INIT_ALL '(' inst_logic ',' affect ')' ';' inst_gen",
"inst_gen : INIT_ALL '(' inst_logic ')' ';' inst_gen",
"inst_gen : START '(' inst_logic ')' ';' inst_gen",
"affect : /* empty */",
"affect : IDENT no_bit '=' '"' NUM '"'",
"inst_emit : /* empty */",
"inst_emit : IDENT ':' IDENT no_bit BEGIN inst_emit_inst END
                inst_emit",
"no_bit : /* empty */",
"no_bit : '[' NUM ']'",
"inst_emit_inst : /* empty */",
"inst_emit_inst : inst_emission_all inst_emit_inst",
"inst_emission_all : inst_emission",
"inst_emission_all : IF '(' inst_logic ')' THEN inst_emission_sans
                     END_IF",
"inst_emission_all : IF '(' const ')' THEN inst_emission_sans END_IF",
"inst_emission_sans : /* empty */",
"inst_emission_sans : inst_emission inst_emission_sans",
"inst_emission : SENDF '(' f_ident ')' ';'",
```

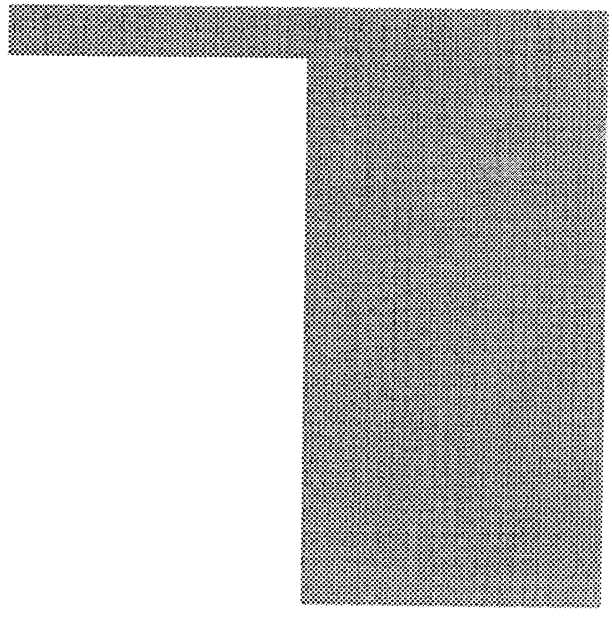
```
"inst_emission : IDENT '(' u_ident ')' ';'",
"inst_emission : INIT_LOC ';'",
"inst_emission : INIT_ALL ';'",
"inst_emission : NEXTSTATE '(' IDENT ')' ';'",
"inst_emission : RECEPT '(' IDENT ')' ';'",
"inst_emission : affect ';'",
"f_ident : const",
"f_ident : IDENT",
"f_ident : IDENT '(' IDENT ')'",
"f_ident : IDENT '[' NUM ']'",
"f_ident : IDENT '(' IDENT '[' NUM ']' ')'",
"recept : RECEPT GENERAL inst_gen ENDGEN BEGIN inst_recept ENDRECEPT",
"inst_recept : /* empty */",
"inst_recept : IDENT ':' IDENT no_bit BEGIN inst_recept_inst END
                inst_recept",
"inst_recept_inst : /* empty */",
"inst_recept_inst : inst_reception_all inst_recept_inst",
"inst_reception_all : inst_reception",
"inst_reception_all : IF '(' inst_logic ')' THEN inst_reception_sans
                      END_IF",
"inst_reception_all : IF '(' const ')' THEN inst_reception_sans
                    END_IF",
"inst_reception_sans : /* empty */",
"inst_reception_sans : inst_reception inst_reception_sans",
"inst_reception : RECEIVEF '(' f_ident ')' ';'",
"inst_reception : IDENT '(' x_ident ')' ';'",
"inst_reception : INIT_LOC ';'",
"inst_reception : INIT_ALL ';'",
"inst_reception : NEXTSTATE '(' IDENT ')' ';'",
"inst_reception : EMIT '(' IDENT ')' ';'",
"inst_reception : affect ';'",
"x_ident : /* empty */",
"x_ident : IDENT y_ident",
"y_ident : /* empty */",
"y_ident : ',' IDENT y_ident",
"u_ident : /* empty */",
"u_ident : f_ident h_ident",
"h_ident : /* empty */",
"h_ident : ',' f_ident h_ident",
"caract : /* empty */",
```

Tableau de description des instructions de base

Instruction	Description	paramétres
Sendf	indique l'unité source d'un champ	nom de l'unité
Fonct_Unit(fident)	appel d'une fonction de traitement	paramétres de la fonction
Init_loc	initialisation Locale	
Init_All	initialisation Globale	
Goto	Branchement	nom_état
Recept	passage en réception	nom_état
Affectation	positionnement des bits d'un registre	valeur binaire
Receivef	indique l'unité de de destination d'un champ	nom de l'unité
Emit	passage en émission	nom_état

ANNEXE D

Description MACS de l'Application VAN



— DESCRIPTION DE L'APPLICATION VAN —

Dans cette annexe, nous présentons la description MACS de l'application VAN. Elle permet de montrer les différents unités de description syntaxique d'un protocole de communication et de son environnement d'implémentation.

```
#-------------------------------------------------------#
#     description d'un protocole de communication       #
#                  Le protocole VAN                     #
#-------------------------------------------------------#

protocol VAN_PROTOCOL

#-------------------#
#  bufferisation :  #
#-------------------#

     bufferisation

# registre commande: #

          reg_C [8]      : in reg ;

# registre adresse: #

          reg_A0 [8]     : in reg ;
          reg_A1 [8]     : in reg ;

# fifo d'emission: #

          fifo_E [8,32] : in fifo ;

# registre d'etat: #

          reg_S0 [8]     : out reg ;
          reg_S1 [8]     : out reg ;

# fifo de reception: #

          fifo_R [8,32] : out fifo ;

# interruption: #

          INT [1]        : out reg ;

     endbuff

#----------------#
#  ADAPTATION    #
#----------------#

     adaptation

# declaration des ports : #

     port (
          DATA_in  [8] : in  data;
          DATA_out [8] : out data;
          RS0          : in  control;
          RS1          : in  control;
          RW           : in  control;
          CS           : in  control;
          E            : in  control;
          Reset1       : in  control;
          IT           : out control;
          )

     begin

          out(fifo_R, data_out) = /RS0./RS1.RW.E./CS ;
          out(reg_S0, data_out) =  RS0./RS1.RW.E./CS ;
          out(reg_S1, data_out) = /RS0. RS1.RW.E./CS ;

          in(fifo_E, data_in) = /RS0./RS1./RW.E./CS ;
          in(reg_C , data_in) =  RS0./RS1./RW.E./CS ;
          in(reg_A0, data_in) = /RS0. RS1./RW.E./CS ;
          in(reg_A1, data_in) =  RS0. RS1./RW.E./CS ;
#
```

```
          reset(INT, reg_S0, reg_S1, fifo_R) = RW.E./CS.
                    (/data_in[1]).RS0.(/RS1)+/reset1;
#

     endadapt

#------------------#
#  INTERFACE BAS : #
#------------------#

     interf_bas
          emission

               sof_E : SOF ;
               eod_E : EOD ;
               eof_E : EOF ;
               ack_E : ACK ;
               data_E :DATA  ;
               collision : in control ;
               Error_E : in control ;
               Start_E : out control ;

          endemission

          reception

               sof_R : SOF ;
               eod_R : EOD ;
               eof_R : EOF ;
               ack_R : ACK ;
               data_R :DATA  ;
               Error_R : in control ;
               Start_R : in control ;

          endreception
     endinterf_bas

#-------------------------------------------------#
# declaration des differentes unites de traitement #
#-------------------------------------------------#

     func_unit

          CRC
            (
               bit_in    : in data [1] ;
               bit_out   : out data[1] ;
               H         : clk;
               sel       : ctrl;
               reset0    : reset;
               Error     : state;
            )

          adr_rec
            (
               bit_in    : in data[1] ;
               init      : reset;
               result    : state;
            )
     endfunc_unit

#--------------------------#
# Declaration des champs   #
#--------------------------#

     field
          data_field
               IDENT[12] ;
               COM  [4]  ;
               DATA [32] ;
               CRCC [15] ;
          enddata

          synchro
               SOF [5] ;
               EOD [1] ;
```

```
                EOF [1] ;
                ACK [1] ;
                NUL [1] ;
           endsynchro
     endfield

#--------------------------------------#
# Description du processus d'emission  #
#--------------------------------------#

     transmit
          general

               # declaration des etats #

               states(etat0, etat1,etat2, etat3, etat4, etat5,
               etat6, etat7, etat8, etat9, etat10, etat11, etat12,
               etat13, etat14, etat15, etat16);

               init_loc (Sig1 , regz [5] = "0" );

               init_all (reset1 , reg_etat[2] = "1");

               start ( / reset0 );

          endgen
    begin
          etat0 : SOF
               begin
               end

          etat1 : IDENT  # identificateur #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
                    adr_rec(bit_in);
                    if (/adr_rec(res))
                    then
                         init_all;
                    end;
               end

          etat2 : COM [1]  # EXT  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
               end

          etat3 : COM [2]  # RAK  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
                    if ("1") then
                         goto(etat10);
                    endif
               end

          etat4 : COM [3]  # RW  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
                    if ("0") then
                         init_loc;
                         reg_S0[2] = "1";
                    endif
               end

          etat5 : COM [4]  # RTR  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
                    if ("1") then
                         init_loc;
                         reg_S0[3] = "1";
                    endif
               end
```

```
          etat6 : DATA  # DONNEES  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
               end

          etat7 : CRC  # le CRC  #
               begin
                    sendf(CRC(bit_out));
               end

          etat8 : EOD
               begin
               end

          etat9 : EOF
               begin
               end

#-------------------------#

          etat10 : COM [3]  # RW  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
                    goto(etat6);
                    if ("0") then
                         init_loc;
                         reg_S0[2] = "1";
                    endif
               end

          etat11 : COM [4]  # RTR  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
               end

          etat12 : DATA  # DONNEES  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
               end

          etat13 : CRC  # le CRC  #
               begin
                    sendf(CRC(bit_out));
               end

          etat14 : EOD
               begin
               end

          etat15 : NUL
               begin
                    recept(etat18);
               end

          etat16 : EOF
               begin
                    reg_S1[8] = "1";
                    init_all;
               end

#-------------------------#

          etat17 : COM [4]  # RTR1  #
               begin
                    sendf(fifo_E);
                    crc(bit_in);
                    goto(etat12);
                    if ("1") then
                         init_loc;
                         reg_S0[4] = "1";
                    endif
```

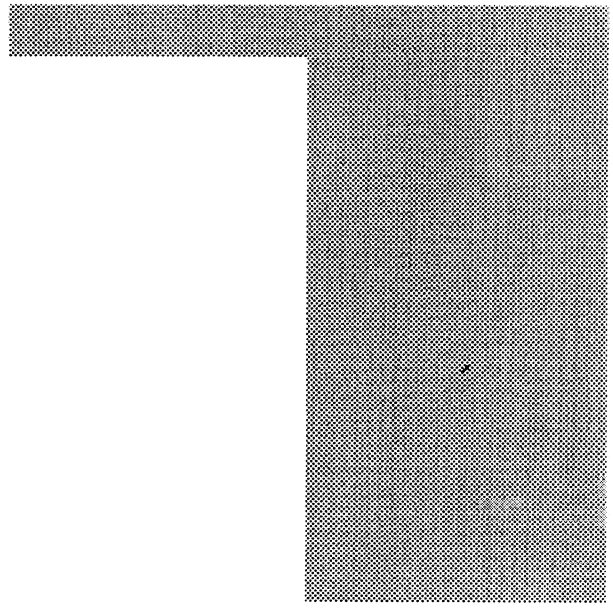
```
                end

#-------------------------#

          etat18 : COM [4]  # RTR2  #
                begin
                     sendf(fifo_E);
                     crc(bit_in);
                     goto(etat12);
                     if ( "1" ) then
                          init_loc;
                     endif
                end

#-------------------------#

          etat19 : ACK
                begin
                     crc(bit_in);
                     init_loc;
                end

     endtransmit

#----------------------------------------#
# Description du processus de Reception : #
#----------------------------------------#


     recept

          general

               # declaration des etats #

               states(etat0, etat1,etat2, etat3, etat4, etat5,
               etat6, etat7, etat8, etat9, etat10, etat11, etat12,
               etat13, etat14, etat15, etat16);

               #
               init_loc (Sig1 , regz [5] = 0 );

               init_all (reset1 , reg_etat[2] =1);
               #

               start ( reg_C[1] );

          endgen
     begin
          etat0 : SOF
               begin
               end

          etat1 : IDENT  # identificateur #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
               end

          etat2 : COM [1] # EXT #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
                    if ("1") then
                         goto(etat10);
                    endif
               end

          etat3 : COM [2]  # RAK  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
                    if ("1") then
                         goto(etat10);
                    endif
               end
```

```
          etat4 : COM [3]  # RW  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
                    if ("0") then
                         init_loc;
                         reg_S0[5] = "1";
                    endif
               end

          etat5 : COM [4]  # RTR  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
                    if ("1") then
                         init_loc;
                         reg_S0[6] = "1";
                    endif
               end

          etat6 : DATA  # DONNEES  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
               end

          etat7 : CRC  # le CRC  #
               begin
                    receivef(CRC(bit_in));
                    if (CRC(error)) then
                         init_loc;
                         reg_S0[7] = "1";
                    endif
               end

          etat8 : EOD
               begin
               end

          etat9 : EOF
               begin
               end

#-------------------------#

          etat10 : COM [3]  # RW  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
                    goto(etat6);
                    if ("1") then
                         init_loc;
                         reg_S1[7] = "1";
                    endif
               end

          etat11 : COM [4]  # RTR  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
               end

          etat12 : DATA  # DONNEES  #
               begin
                    receivef(fifo_R);
                    crc(bit_in);
               end

          etat13 : CRC  # le CRC  #
               begin
                    receivef(CRC(bit_in));
                    if (CRC(error)) then
                         init_loc;
                         reg_S0[7] = "1";
                    endif
```

```
                end

           etat14 : EOD
                begin
                end

           etat1 : NUL
                begin
                     emit(etat18);
                end

           etat16 : EOF
                begin
                     reg_S0[8] = "1";
                     init_all;
                end

#-------------------------#

           etat17 : COM [4]  # RTR1  #
                begin
                     receivef(fifo_R);
                     crc(bit_in);
                     goto(etat12);
                     if ("1") then
                          init_loc;
                          reg_S0[4] = "1";
                     endif
                end

#-------------------------#

           etat18 : COM [4]  # RTR2  #
                begin
                     receivef(fifo_R);
                     crc(bit_in);
                     goto(etat12);
                     if ("1") then
                          init_loc;
                     endif
                end

#-------------------------#

           etat19 : ACK
                begin
                     init_loc;
                end
     endrecept


endprotocol
```

ANNEXE E

Simulation du Réseau VAN



— SIMULATION DU RESEAU VAN —

Dans cette annexe, nous présentons quelques résultats sous forme de chronogrammes de la simulation du réseau VAN. Elle permet de montrer d'une part les différents cas du VAN (fonctionnement du circuit) et d'autre part l'observation à différents niveaux du système : application, circuit de communication, médium de communication.

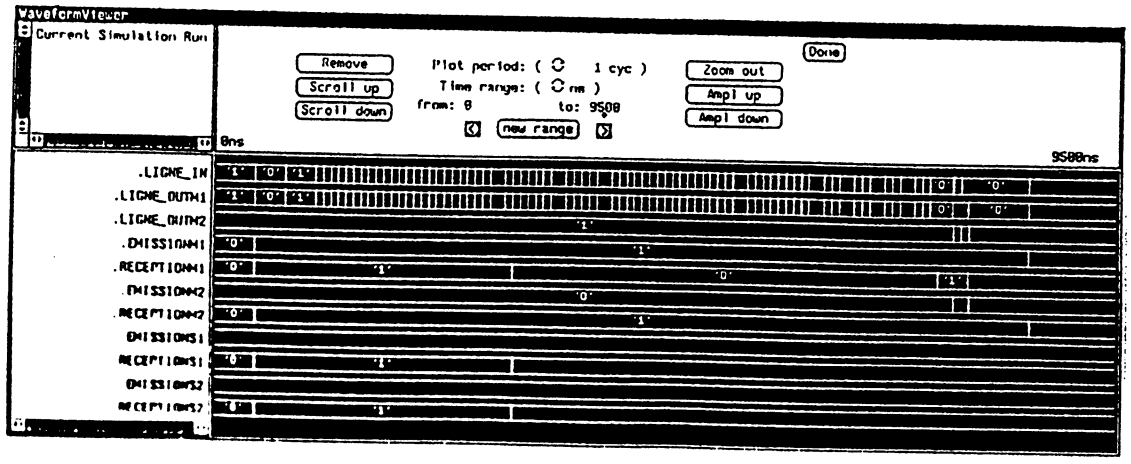
Premier Cas:

Trame de Données avec demande d'acquittement

La Station maître MASTER I envoie des données à la station maître MASTER II avec une demande d'acquittement. Dans ce cas le champ ACK est positionné par le maître MASTER II.

Ce chronogramme montre les différents signaux indiquant:

. l'état de la ligne de communication,
. l'état de chaque station (émission ou réception)
. les données de sortie de chaque station
. les données d'entrée de chaque station.



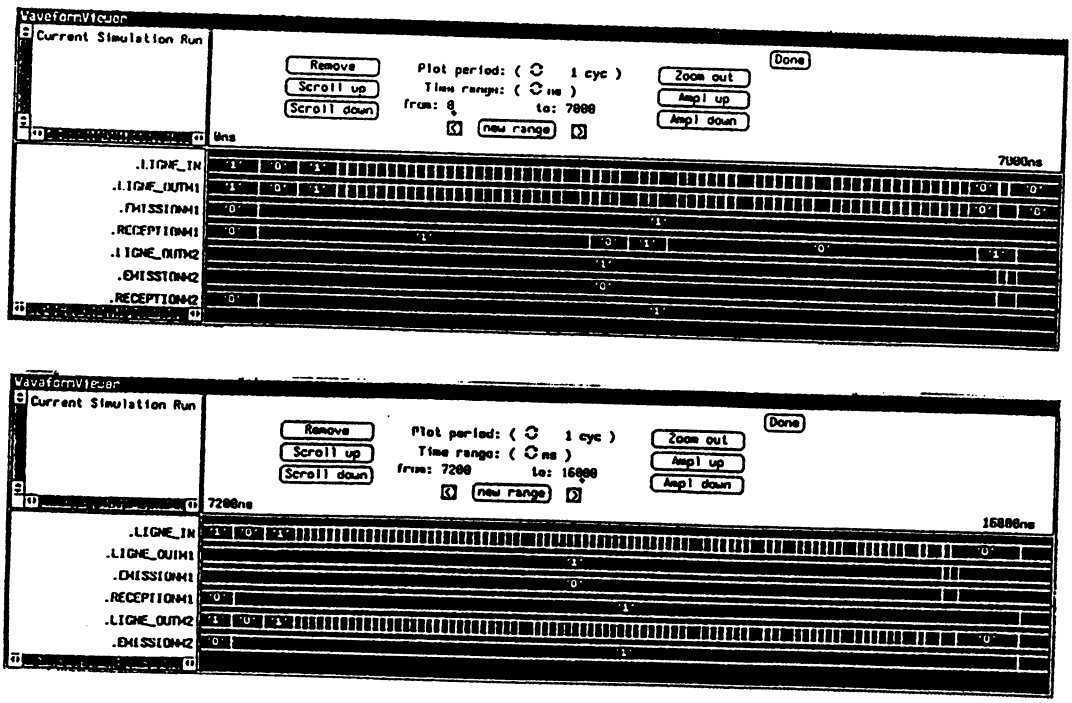
Deuxième Cas :

Trame de Demande de Données avec Réponse Différée

La station maître MASTER II envoie une trame de demande de données à la station maître MASTER I. Celle-ci répond après réception de toute la trame, par une autre trame de réponse.

Les deux chronogrammes montrent les différents signaux indiquant pour chaque cas:

. l'état de la ligne de communication,
. l'état de chaque station (émission ou réception)
. les données de sortie de chaque station
. les données d'entrée de chaque station.



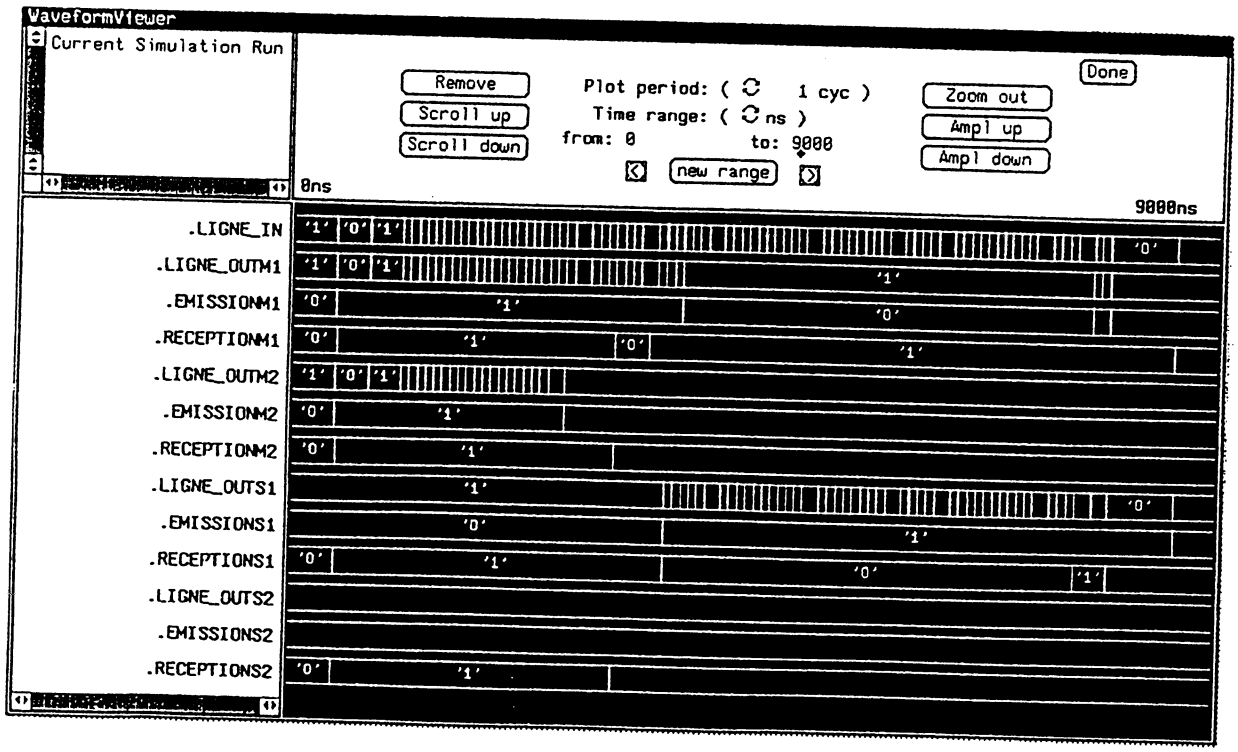
Troisième Cas:

Trame de Demande de Données avec Réponse dans la trame

Dans ce cas, les deux stations maîtres MASTER I et MASTER II envoient au même instant des trames de demande de données respectivement aux stations esclaves SLAVE I et SLAVE II.

La trame de la station maître MASTER I l'emporte sur celle de la station MASTER II car durant le conflit d'accès, l'adresse de la station SLAVE I est prioritaire sur celle de SLAVE II.

Le chronogramme suivant montre l'état des différentes stations et la réponse dans la trame de SLAVE I.



AUTORISATION DE SOUTENANCE

Vu les dispositions de l'arrêté du 30 Mars 1992 relatifs aux Etudes Doctorales

Vu les Rapports de présentations de :

Monsieur ROUQUIER

Monsieur MAURIN

Monsieur Hakim SAHEB

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de **Docteur de l'Institut National Polytechnique de Grenoble,** spécialité "Microélectronique".

Fait à Grenoble, le 17 Novembre 1993

Pour le Président de l'INPG
et par délégation
le Directeur de l'Ecole Doctorale
J.L. LACOUME

INPG. 46, Avenue Felix Viallet. 38031 GRENOBLE CEDEX. FRANCE. Tel. (33) 76 57 45 00 Télex 320 205 F Telecopie (33) 76 57 45 01