THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

SPÉCIALITÉ INFORMATIQUE

Présentée par Vincent REIBALDI
Pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ PARIS VI

CONCEPTION ET RÉALISATION D'UN ROUTEUR DE PAQUETS À HAUTES PERFORMANCES

Soutenue le 3 Juillet 1997, devant le jury composé de

ALAIN GREINER	Directeur de thèse
DANIEL ÉTIEMBLE	Rapporteur
ALAIN MÉRIGOT	Rapporteur
DANIEL LITAIZE	Examinateur
ROLAND MARBOT	Examinateur

Résumé

Cette thèse présente l'architecture d'un routeur de paquets à hautes performances intégrant 8 liaisons série point-à-point asynchrones full-duplex à 1 Gbit/s développées à BULL et conformes au standard IEEE 1355.

Ce routeur appelé RCUBE est conçu autour d'un crossbar non bloquant 8 x 8 et offre une latence de routage de 150 ns ainsi qu'un débit global de 640 Mo/s à 80 MHz.

Il supporte deux schémas de routage de base qui sont le schéma de routage par intervalles et le schéma de routage explicite. Cette caractéristique permet d'utiliser RCUBE efficacement dans n'importe quelle topologie de réseau comportant un nombre quelconque de nœuds.

La taille des paquets routés par RCUBE est quelconque, ce qui lui permet d'encapsuler des paquets correspondant à n'importe quel protocole de communication.

Afin de réduire les phénomènes de congestion, RCUBE intègre une fonction d'adaptativité dont le coût n'est que d'un cycle. Cette fonction permet à l'utilisateur de spécifier un ordre dans les sorties que peut prendre un paquet.

La présence d'une table de routage par port d'entrée autorise la réalisation de plusieurs réseaux logiques disjoints dans un même routeur.

Développé dans le cadre du projet européen OMI/HIC, RCUBE comporte 465000 transistors et a été fabriqué en technologie CMOS 0.5 μm-3.3 V par SGS-THOMSON. Testé à BULL, il a été mis en carte au laboratoire LIP6 et est opérationnel à 66 MHz.

Mots-clés

réseau haut débit
routage par intervalle
adaptativité
liaison série
gigabit/s

Abstract

This thesis presents the internal architecture of a high performance packet router including 8 IEEE 1355 compliant point-to-point asynchronous full-duplex 1 Gbit/s serial links developped at BULL.

This router called RCUBE, is based on a non blocking 8 x 8 full crossbar and features a 150 ns routing latency as well as a global throughput of 640 Mbytes/s at 80 Mhz.

RCUBE supports two basic routing schemes which are the interval routing scheme and the source routing scheme.

This feature allows RCUBE to be efficiently used in any topology of network including any number of nodes.

RCUBE routes packets of any length that makes possible the encapsulation of packets corresponding to any communication protocol.

In order to reduce "hot spots", RCUBE provides adaptive routing which costs only one cycle. This feature allows users to specify an order in the output ports that can be taken by a packet.

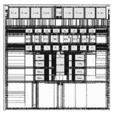
Each input port includes its own routing table which allows independent networks to be implemented with the same RCUBE with a total security.

Developped in the frame of the OMI/HIC european project, RCUBE includes 465000 transistors and has been fabricated in a 0.5 μm-3.3 V CMOS process by SGS-THOMSON.

Tested at BULL, RCUBE has been soldered on an evaluation board at laboratoire LIP6 and it is currently working at 66 MHz.

Keywords

high speed network
interval labeling
adaptivity
serial link
gigabit/s



Remerciements

Tous ceux qui ont connu les affres de la rédaction d'un mémoire de thèse, savent combien ce travail est par essence un effort tout personnel. C'est là, le paradoxe de cet exercice réalisé dans un domaine — la microélectronique, où plus qu'ailleurs, le travail d'équipe est la clé de la réussite.

C'est pourquoi je tiens particulièrement à remercier Belkacem ZERROUK qui fut l'initiateur du projet RCUBE et dont les idées originales ont servi de base à ce travail. Je remercie Frédéric POTTER, qui par la pertinence et l'intelligence de ses remarques a largement contribué à la réussite de ce projet. Je veux exprimer ici toute ma gratitude envers le professeur Alain GREINER qui a dirigé ces recherches, à la fois pour son travail d'encadrement, ses conseils avisés et sa patiente relecture de mon mémoire.

Je remercie Daniel ÉTIEMBLE — professeur au LRI à Orsay et Alain MÉRIGOT — professeur à L'IEF à Orsay qui ont accepté d'être les rapporteurs de mes travaux, ainsi que le professeur Daniel LITAIZE — professeur à l'IRIT à Toulouse, et Roland MARBOT — ingénieur à SGS-THOMSON qui ont bien voulu en être les examinateurs.

Je veux remercier les membres de l'équipe qui pendant 3 ans et demi m'ont donné l'occasion d'évoluer dans un cadre marqué par une excellente ambiance de travail. Je remercie particulièrement les concepteurs d'outils CAO, Pirouz BAZARGAN, Ludovic JACOMME, et Karim DIOURY, pour leur disponibilité et la diligence dont ils ont fait preuve dans leurs interventions, ainsi que les stagiaires de DEA, Jean-Lou DESBARBIEUX, Alain GOEURY, Arnaud TURIER, Dimitri TRANOUDIS qui m'ont apporté une aide précieuse tout au long du développement de ce projet.

Enfin, il me tiens à cœur d'exprimer ma plus profonde reconnaissance envers mes parents, qui m'ont donné la chance de pouvoir poursuivre aussi loin mes études, et ce dans les meilleurs conditions qui soient.

À mes grands-parents, Henri et Léone...

Table des matières

Liste des figures

Liste des tables

CHAPITRE 1

Introduction

Afin d'offrir des performances toujours plus importantes, les systèmes de traitements de données ont évolué au cours du temps, du simple monoprocesseur aux machines massivement parallèles regroupant plusieurs centaines voire, dans un avenir proche, plusieurs milliers de processeurs communiquant ensembles. Le facteur limitant les performances du système tout entier devient dès lors la performance du réseau d'interconnexion permettant aux processeurs d'échanger des données. Notre étude a pour but la conception et la réalisation d'un routeur de paquets à haut débit et faible latence, permettant la construction de réseaux d'interconnexion à hautes performances.

Domaine d'application

Notre domaine d'application est celui des machines massivement parallèles de type MIMD (*Multiple Instruction stream Multiple Data stream*). Au sein de cette grande famille, on distingue plusieurs sous-classes qui se différencient principalement par le degré de couplage qui existe entre les processeurs qui les constituent, et le modèle de programmation qu'elles supportent.

Nous citerons pour exemple, les machines basées sur le modèle de programmation à mémoire partagée, les machines utilisant le modèle de programmation par passage de messages, les machines à flots de données etc... Nous considérons donc des systèmes composés d'éléments (processeurs, bancs mémoire etc...) distants, mais séparés par une distance n'excédant pas plusieurs mètres. Nous excluons ainsi le domaine des télécommunications.

Nous utiliserons dorénavant le terme de **nœud** pour faire référence aux éléments de ces systèmes. Un nœud peut se réduire à un processeur, un banc mémoire, ou comporter plusieurs processeurs avec plusieurs bancs mémoire. Nous supposons uniquement que le nœud dispose d'une interface au réseau d'interconnexion.

1.1 Nécessité d'un routeur

L'architecture idéale d'une machine massivement parallèle suppose l'existence d'une liaison directe entre chacun des nœuds du système, de façon à réaliser une topologie complètement connectée. En effet que l'on se place dans le modèle de programmation à mémoire partagée ou à passage de messages, le but recherché est identique: il s'agit de faciliter au maximum la communication entre les processus s'éxécutant concurremment sur différents processeurs, afin qu'ils puissent s'échanger des données et se synchroniser. Une topologie complètement connectée peut être réalisée grâce à deux types de média de communication : le bus et la liaison point-à-point (fig.1.1).

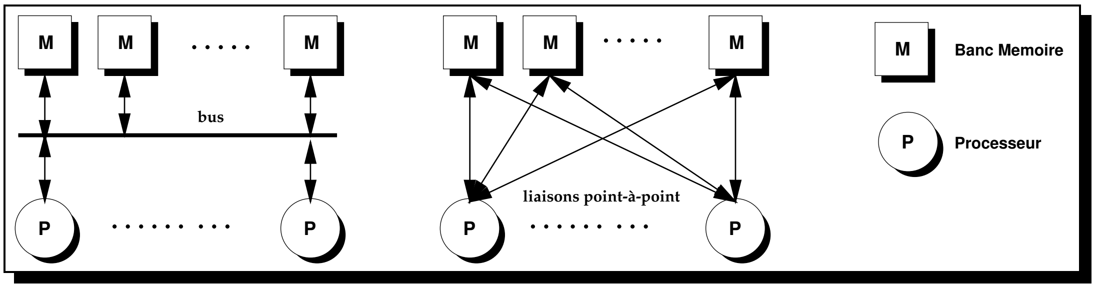


Figure 1.1: Deux exemples de topologies complètement connectées

Le bus multiplexe les échanges entre nœuds pour éviter de réaliser physiquement toutes les liaisons point-à-point possibles. En contrepartie, la congestion de communication augmente avec le nombre de nœuds, dans la mesure où la bande passante du bus reste constante. En outre, pour des raisons électriques, le bus ne peut supporter qu'un nombre limité de nœuds.

La liaison point-à-point relie deux à deux tous les nœuds du système. Cette solution, théoriquement idéale, ne peut être utilisée lorsque l'on vise des systèmes à plusieurs centaines d'éléments car le nombre de connexions devient rédhibitoire. En effet, un système comportant N nœuds nécessite $\frac{N.(N-1)}{2}$ liens soit déjà plus de 8000 pour interconnecter 128 nœuds et plus de 520 000 pour 1024 nœuds.

De ce fait, il est exclu d'envisager des topologies complètement connectées lorsqu'on se propose de réaliser des systèmes massivement parallèles.

Afin de surmonter ce problème, on construit des réseaux d'interconnexion utilisant des liaisons point-à-point, mais non complètement connectés. On distingue deux classes de réseaux d'interconnexion: les réseaux **statiques** et les réseaux **dynamiques**, dont le contrôle peut être soit **centralisé** soit **distribué**.

1.1.1 Réseaux statiques / dynamiques

Dans les réseaux d'interconnexion statiques, les nœuds sont connectés les uns aux autres par des liaisons point-à-point, selon des topologies fixes comme par exemple, les anneaux, les grilles, les arbres ou les hypercubes (Cf. figure 1.2). L'avantage des réseaux statiques réside dans leur régularité, qui permet de mettre en œuvre des algorithmes de routage simples. De nombreuses machines parallèles MIMD utilisant des réseaux d'interconnexion statiques ont été développées. Citons la CM-5 de THINKING MACHINE CORPORATION (Fat-tree), la PARAGON d'INTEL (grille bidimensionnelle), ou encore le T3D de CRAY (tore tridimensionnel). Dans les réseaux statiques, la fonction de routage peut être soit dévolue au processeur du nœud, soit à un composant dédié appelé **routeur** appartenant aussi au nœud. Dans la majorité des cas, le routeur est peu complexe et intègre de façon matérielle un algorithme de routage adapté à la topologie du réseau. Cette adaptation à une topologie fixée rend le routeur incapable de supporter des topologies différentes résultant éventuellement de pannes de certains nœuds ou de rupture de connexions.

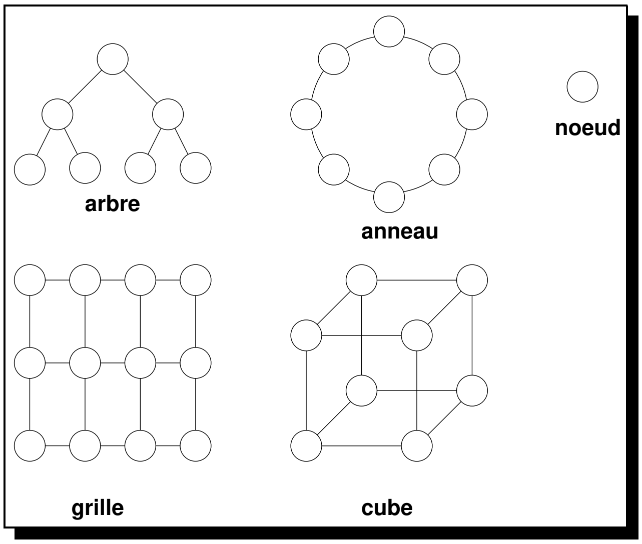


Figure 1.2: Exemples de topologies de réseaux statiques

Dans les réseaux dynamiques, les connexions entre les nœuds varient au cours du temps. Pour ce faire, le réseau d'interconnexion est construit à base de routeurs intégrant un composant de commutation appelé "crossbar". Les crossbars disposent d'un certain nombre d'entrées et de sorties pouvant être reliées temporairement en interne afin de créer une connexion. Ces routeurs ne disposent généralement que d'un petit nombre d'entrées et de sorties, car il est difficile de réaliser un crossbar ayant un grand nombre d'entrées-sorties à un coût matériel raisonnable. En conséquence afin de permettre la connexion entre n'importe quelle paire de nœuds d'un système massivement parallèle, les routeurs sont connectés selon des topologies de type multi-étage comme les réseaux de Clos[1] (utilisé par exemple dans la GF-11 d'IBM [2]). Le problème qui se pose alors est de commander les crossbars de façon optimale.

1.1.2 Contrôle centralisé / distribué

Dans un réseau à contrôle centralisé, l'établissement des connexions entre les entrées et les sorties des crossbars est commandé par un arbitre central qui gère les requêtes de communication émises par les nœuds du système. Le traitement simultané de ces demandes de connexions nécessite que l'arbitre dispose d'une très forte puissance de calcul puisqu'il doit tenir compte à chaque instant de l'état de tous les nœuds et de toutes les liaisons du réseau avant de commander les commutateurs. Ceci impose donc d'amener l'information de l'état du réseau jusqu'à l'arbitre central. Il est inenvisageable d'amener physiquement un signal d'état des nœuds et des liaisons jusqu'à l'arbitre dans le cas d'un système comportant des centaines de nœuds. Il est donc nécessaire d'utiliser le réseau pour acheminer ces informations. En général, un réseau de contrôle existe de façon indépendante du réseau de données. Cependant, l'acheminement des informations d'états n'est pas instantané. Aussi les décisions prises par l'arbitre central peuvent être basées sur des informations périmées. Un autre inconvénient des réseaux à contrôle centralisé est leur grande vulnérabilité dans le cas où le nœud de contrôle tombe en panne. Enfin le nombre de nœuds dans un réseau de ce type est nécessairement borné ne serait-ce que parce que la puissance de traitement de l'arbitre n'est pas infinie.

Par opposition, les réseaux à contrôle distribués mettent en œuvre des routeurs évolués capables de commander leur crossbar interne et donc d'aiguiller les messages selon l'information de destination contenue dans l'en-tête du message. Il n'y a donc plus de contrôle central, la décision de routage est distribuée et s'effectue dans chaque routeur à mesure que les données contenant une information de destination progressent dans le réseau.

1.1.3 Conclusion

Nous avons vu que les réseaux statiques bien que simples à mettre œuvre, ne peuvent supporter des topologies différentes résultant éventuellement de pannes de certains nœuds ou de rupture de connexions. Nous avons montré ensuite que les réseaux à contrôle centralisé n'étaient pas adaptés aux machines massivement parallèles. Les routeurs permettant d'aiguiller les messages selon l'information de destination qu'ils transportent sont donc la solution permettant de réaliser des réseaux d'interconnexion dynamique à contrôle distribué. Le but de cette thèse est d'étudier les problèmes liés à la réalisation d'un tel composant.

1.2 Le projet RCUBE

En 1993 est lancé le projet européen Esprit OMI/HIC regroupant le laboratoire MASI et les industriels BULL (France), THOMSON-SINTRA (France), SGS-THOMSON (France), PARSYTEC (Allemagne) et SINTEF (Norvège). Ce projet a pour but le développement de démonstrateurs des technologies de liaisons série à haut débit pour la construction de systèmes d'interconnexion extensibles à bas coût. Notre tâche au sein de ce projet est la conception et la réalisation d'un routeur de paquets compatible avec la norme IEEE 1355, permettant la construction de réseaux d'interconnexion à hautes performances pour machines massivement parallèles de type MIMD. Ce routeur utilise des liaisons série, point-à-point, asynchrones à 1 Gbit/s dont la technologie a été développée par BULL et le laboratoire MASI. Début 1994, encadrés par B. Zerrouk, F. Potter et moi-même définissons les spécifications du routeur ainsi que son architecture. Mi-94, F. Potter commence le développement d'un modèle du routeur en langage C, tandis que j'écris le modèle en langage VHDL (Very high speed integrated circuit Hardware Design Language). En 1995, la réalisation physique du circuit est menée à son terme. Début 1996, le circuit est envoyé en fabrication. Début 97, le composant est opérationnel à 66 MHz sur une carte d'évaluation.

1.3 Plan du mémoire

Le chapitre 2 met en évidence les problèmes et les difficultés de l'étude entreprise et précise les objectifs. Le chapitre 3 analyse de façon systématique les éléments clés d'un routeur. Nous abordons dans l'ordre les techniques de routage, les schémas de routage, le contrôle de flux, les stratégies de mémorisation et les problèmes d'interblocage. Cette analyse nous permettra d'évaluer pour chacun de ces points, quelle est la complexité matérielle induite au regard des performances obtenues. Le chapitre 4 présente trois exemples industriels de routeurs de paquets. Il s'agit du VULCAN d'IBM, du TIC de SUN MICROSYSTEM et du C104 de SGS-THOMSON. Le chapitre 5 décrit la spécification fonctionnelle du routeur et justifie les choix que nous avons faits. Le chapitre 6 présente l'architecture matérielle du crossbar. Le chapitre 7 fait une synthèse des résultats obtenus en terme de surface et de performances temporelles. Nous explicitons quelle est la chaîne critique du routeur et démontrons la faisabilité d'un routeur de valence supérieure.

Chapitre 2

Problématique

L'objet de ce chapitre est de poser les problèmes que nous nous sommes proposé de résoudre dans cette thèse. Nous présentons tout d'abord deux exemples de machines parallèles et explicitons quels sont leurs besoins en matière de réseau d'interconnexion. Nous précisons ensuite quels sont les objectifs et les contraintes de notre étude, puis nous mettons en évidence les difficultés de réalisation qu'ils soulèvent.

2.1 Exemples d'applications visées

Nous allons présenter deux projets de machines parallèles de type MIMD.

La première machine, appelée MPC (machine MultiPC) est une machine universitaire multiprocesseurs en cours de développement au laboratoire MASI. La seconde, développée par le CERN dans le cadre du projet ATLAS, a pour but le traitement en temps réel de données issues de capteurs situés dans la chambre de collision d'un cyclotron.

Deux autres projets industriels de machines parallèles ont des objectifs très voisins du projet MPC et utilisent également le routeur décrit dans cette thèse. L'une, développée par THOMSON SINTRA-ASM (France) dans le cadre du projet européen EUROPRO, vise des applications de type traitement du signal en temps réel dans des domaines aussi variés que le contrôle de trafic aérien, l'imagerie tridimensionnelle ou la prospection pétrolière. L'autre, développée par PARSYTEC (Allemagne) dans le cadre du projet européen ARCHES, vise des applications de reconnaissance et de classification de formes, de caractères, etc. Du point de vue du réseau d'interconnexion, ces deux machines présentent de nombreuses similitudes avec la machine MPC. C'est pourquoi nous nous contentons de les citer sans les décrire.

2.1.1 La machine MultiPC

La machine MultiPC est une machine parallèle à passage de messages, extensible, à faible coût, en utilisant des cartes mères de PC standards à bus PCI reliées par un réseau de communication performant.

Chacune des cartes mères est connectée au réseau d'interconnexion à travers un contrôleur réseau dédié appelé PCI-DDC (Cf. figure 2.1). Ce composant réalise l'interface entre le bus PCI et notre routeur intégrant huit liaisons série haut débit, tout en implémentant un protocole de communication original [3, 4]). Le principe de ce protocole repose sur le transfert de données directement depuis l'espace mémoire du processus émetteur vers celui du récepteur, sans recopie intermédiaire, contrairement aux protocoles UDP ou TCP. Le but de ce protocole est de réduire au maximum le coût logiciel des communications. En conséquence, le routeur constituant la brique de base du réseau d'interconnexion doit offrir une très faible latence de routage pour ne pas devenir le facteur limitant.

Dans sa configuration 32 bits-33MHz, le bus PCI offre un débit crête de 132 Mo/s. En pratique, les débits atteints sont compris entre 60 et 80 Mo/s, ce qui demeure un débit élevé. Afin d'en profiter, le réseau doit être en mesure d'absorber et de fournir un débit de cet ordre pour chacun des nœuds.

La lecture des données dans la mémoire du processus émetteur est faite par le composant PCI-DDC qui accède à celle-ci en mode DMA. La norme PCI stipule qu'un périphérique ne peut garder le contrôle du bus au delà d'un certain temps et doit donc le relâcher. En conséquence, les données récupérées par le composant PCI-DDC forment des paquets de taille variable. Il est donc nécessaire de pouvoir router des paquets de taille variable afin d'optimiser l'utilisation de la bande passante du réseau.

Les architectes de la machine MPC souhaitent proposer et expérimenter deux modèles de programmation sur la même plate-forme matérielle:

- le modèle à "passage de messages" est généralement utilisé pour des applications distribuées "faiblement couplées". Chaque nœud possède généralement son propre système d'exploitation et son propre espace d'adressage. Le principal objectif est ici de maximiser le débit global du réseau d'interconnexion.
- dans le modèle de programmation READ/WRITE, il n'existe généralement qu'un seul système d'exploitation et un seul espace d'adressage pour tous les nœuds de la machine. Ce modèle est souvent utilisé par des applications parallèles fortement couplées: les quantités d'information échangées sont de petite taille (une ligne de cache, soit typiquement 32 octets). Le principal objectif ici est de minimiser la latence du réseau, tout en conservant un débit global élevé.

La machine MPC a une vocation généraliste. Il est donc important de pouvoir interconnecter n'importe quel nombre de nœuds avec une topologie quelconque selon la puissance désirée et l'application visée. Le routeur doit permettre une extensibilité aisée de l'architecture.

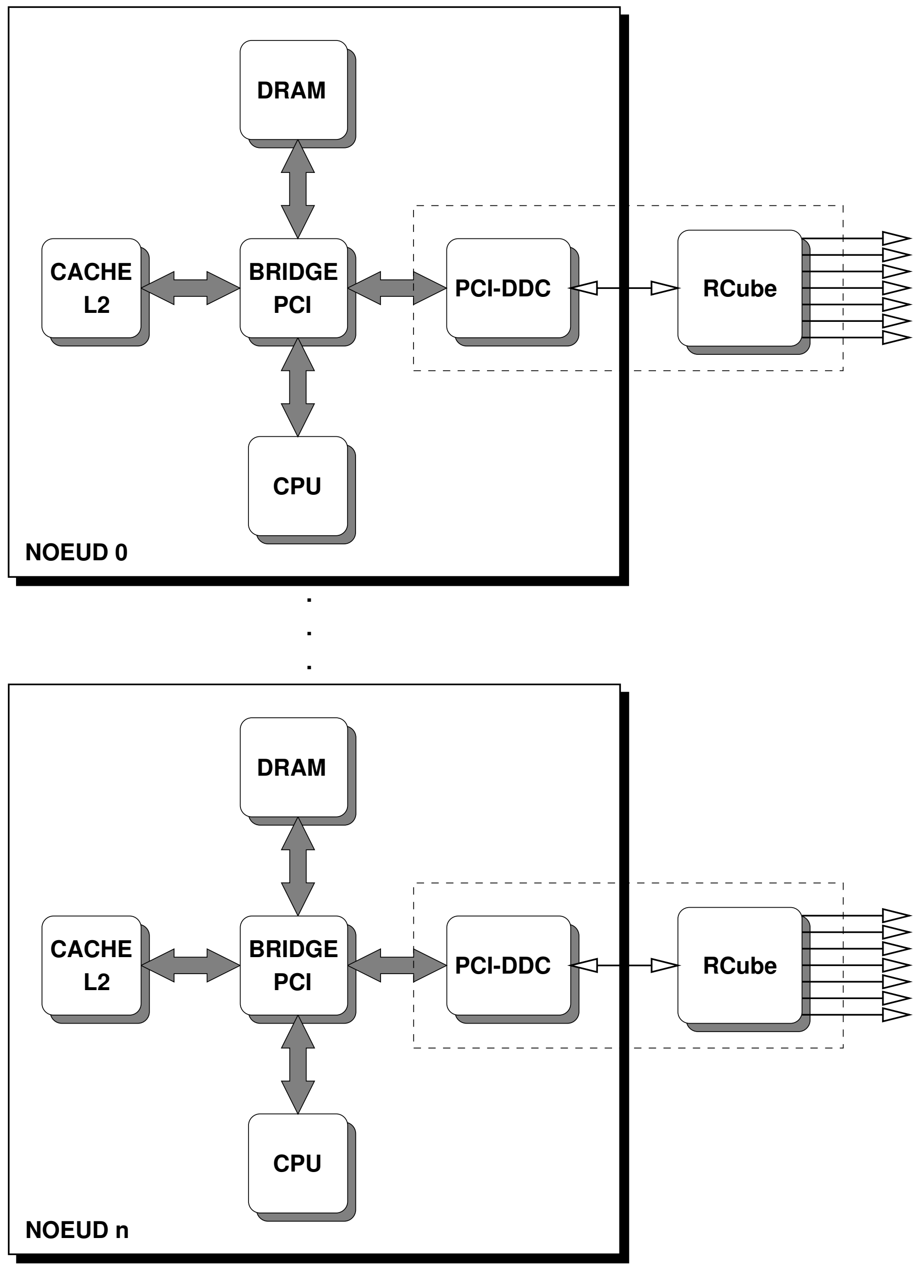


Figure 2.1: Synoptique de la machine MPC

2.1.2 Le système ATLAS

Le CERN développe actuellement le prototype d'un système d'acquisition et de traitement de données (figure 2.2) délivrées par un très grand nombre (plusieurs milliers) de capteurs spatialement distribuées dans la chambre de collision du futur accélérateur de particules LHC (Large Hadron Collider). Le but du système est de reconnaître parmi les données issues des détecteurs, celles qui constituent un "évènement" au sens physique, celui-ci étant défini par des critères auxquelles doivent correspondre les données reçues. Les données brutes sont stockées dans des tampons appelés ROB (*Read Out Buffers*) à un débit de l'ordre de 200 Koctets/s en entrée de chaque ROB. Chacune des cartes *R/O* contiennent 15 ROBs et l'objectif est de supporter jusqu'à 2000 ROBs. Après avoir reçu une adresse de destination, les données issues des ROBs sont acheminées, via un réseau d'interconnexion (L3 *switching matrix*), vers des dizaines de nœuds de traitement (*L3 Processor Farm*), chargés d'effectuer le tri. Les données faisant état de l'occurrence d'un évènement doivent être enregistrées, les autres détruites. La masse de données à traiter est telle qu'il est impossible de les stocker puis de les traiter ultérieurement. Le système doit être en mesure de prendre la décision aussi rapidement que possible pour détruire les données ne correspondant pas à un évènement significatif. La bande passante du réseau d'interconnexion est donc un facteur capital. L'étude menée au CERN par le groupe de travail responsable de la définition du prototype du système d'acquisition a listé les caractéristiques que devrait avoir le réseau d'interconnexion pour répondre à leurs besoins. Celui-ci doit offrir:

- une complète interconnexion entre les cartes R/O et les processeurs de traitement,
- une extensibilité en fonction du nombre de sources et de destinations,
- la possibilité de diviser le système en plusieurs sous-systèmes disjoints,
- la détection des erreurs de transmission,
- une tolérance aux pannes: continuer d'acheminer les données vers leur destination même en cas de rupture de certaines liaisons,
- le redémarrage des liaisons: si un certain nombre de ruptures de liaison peuvent être effectivement supportées, cela entraine néanmoins une augmentation de l'occupation des liaisons fonctionnelles. C'est pourquoi il est important d'avoir des mécanismes d'identification et de redémarrage des liaisons défaillantes qui ne perturbent pas le réseau.

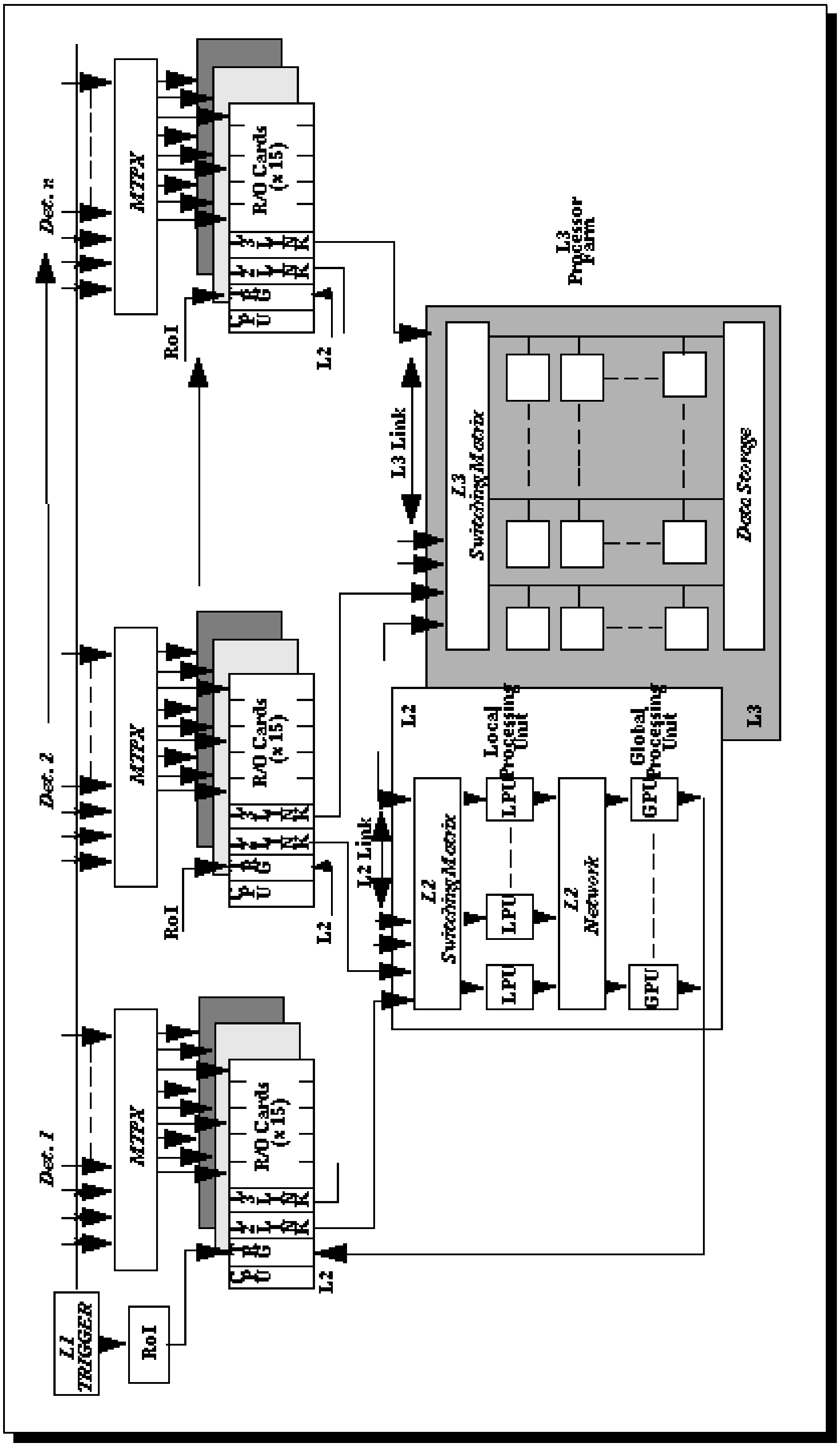


Figure 2.2: Synoptique du système ATLAS

2.2 Caractéristiques souhaitées du routeur

A la lumière des objectifs visés par ces 2 architectures, nous pouvons déduire les fonctionnalités que doit offrir un routeur.

- **Latence:** la latence de communication doit être la plus faible possible. Cela facilite le travail du programmeur d'applications qui n'a pas besoin d'anticiper les temps de transfert des données pour optimiser son programme. En outre, la latence est un facteur déterminant dans les applications temps-réel.
- **Débit:** les processeurs actuels sont capables d'émettre et de recevoir des données à un débit d'environ 80 Mo/s sur leur bus d'entrées/sorties. Il est donc nécessaire que le réseau d'interconnexion puisse offrir un débit de cet ordre, pour chaque processeur, sous peine de sous-utiliser les capacités de calcul des nœuds de traitement.
- **Topologie quelconque:** la possibilité de supporter des topologies de réseaux quelconques permet de s'adapter aux besoins des applications.
- **Tolérances aux pannes:** dans des systèmes comportant plusieurs centaines de nœuds, la probabilité qu'un ou plusieurs nœuds soient en panne n'est plus négligeable. La tolérance aux pannes d'un système massivement parallèle est donc une caractéristique fondamentale. Il faut donc que le routeur implémente des solutions permettant d'utiliser le système, même lorsque certains nœuds ou liaisons sont en panne. Cet objectif renforce la nécessité de supporter des topologies quelconques résultant de la dégradation éventuelle d'une topologie initialement régulière.
- **Réseau sans perte:** Contrairement à un réseau de type ATM, nous désirons assurer une transmission sans perte des données (en l'absence de pannes).
- **Résistance à la congestion:** Il arrive que certaines liaisons soient occupées de façon intensive. Cela provoque des phénomènes de congestion ("points chauds") et donc une augmentation de la latence qui, pour les applications temps-réel, est intolérable. Afin de soulager ces liaisons, il est nécessaire de pouvoir router les données par d'autres chemins (adaptativité).
- **Extensibilité:** L'extensibilité d'un système est un critère économique important. En effet, une solution pour accroître la puissance d'une machine parallèle est d'augmenter le nombre de processeurs qu'elle contient. Le réseau d'interconnexion doit donc être en mesure d'accepter facilement cet ajout sans que soit dégradé le rapport coût/performances du système.
- **Paquets de taille quelconque:** La possibilité de router des paquets de taille quelconque permet d'optimiser l'utilisation de la bande passante, mais aussi d'encapsuler des messages correspondant à n'importe quel type de protocole de communication.

2.3 Objectifs

Après avoir défini les fonctionnalités que le routeur doit offrir, nous allons poser clairement nos objectifs, opérer d'ores et déjà certains choix et montrer les implications et difficultés de réalisation qu'ils entrainent.

2.3.1 Faible latence

La latence de communication entre deux nœuds se définit comme le temps qui sépare le début de l'émission d'un message par le nœud émetteur et sa complète réception par le nœud récepteur. On peut décomposer cette latence de communication comme la somme de trois termes: la latence de traitement du message L1, la latence du réseau L2 et le temps de blocage L3 [5]. La latence de traitement du message L1 est le temps requis par le système pour traiter le message au niveau de l'émetteur et du récepteur. Cette latence est indépendante du routeur. La latence du réseau L2 est le temps qui s'écoule entre le moment où le message pénètre dans le réseau et celui où il en sort en l'absence de contention. Le temps de blocage L3 regroupe tous les délais d'attente que peut subir le message en transitant de routeurs en routeurs, comme par exemple les délais introduits lorsque deux messages entrent en conflit pour obtenir le même port de sortie. Un de nos principaux objectifs est d'offrir une latence de routage, somme des latences L2 et L3, aussi faible que possible. En effet, seules ces latences sont affectées par les choix architecturaux du routeur. Notre but initial était d'atteindre une latence élémentaire sans contention de 100 nanosecondes par routeur. Cette valeur, inférieure d'au moins un ordre de grandeur aux temps de traitement logiciel couramment obtenus, nous assure que le réseau d'interconnexion n'est pas l'élément limitant la latence globale de communication à travers le système.

2.3.2 Haut débit et extensibilité: la liaison série HSL

Une des caractéristique importante d'un système massivement parallèle est son extensibilité. En effet, il est capital de pouvoir augmenter la puissance d'un système en lui ajoutant des nœuds de traitement. L'extension d'un système entraine l'accroîssement du nombre des connexions et de la distribution spatiale des nœuds, ce qui soulève deux difficultés bien connues: la distribution d'horloge dans le système et le routage des connexions sur les cartes.

En effet, le routage de bus parallèles sur de longue distance concourt à l'apparition d'un déphasage entre les bits d'un même mot. Garantir un déphasage faible est extrêmement coûteux. En outre, le nombre et la taille des bus sont limités par le nombre de broches disponibles sur les boîtiers qui encapsulent les circuits. Pour s'affranchir du problème de déphasage entre les bits d'un octet progressant sur un bus et repousser la limite du nombre de ports d'entrée/sortie des routeurs, il est avantageux d'utiliser des

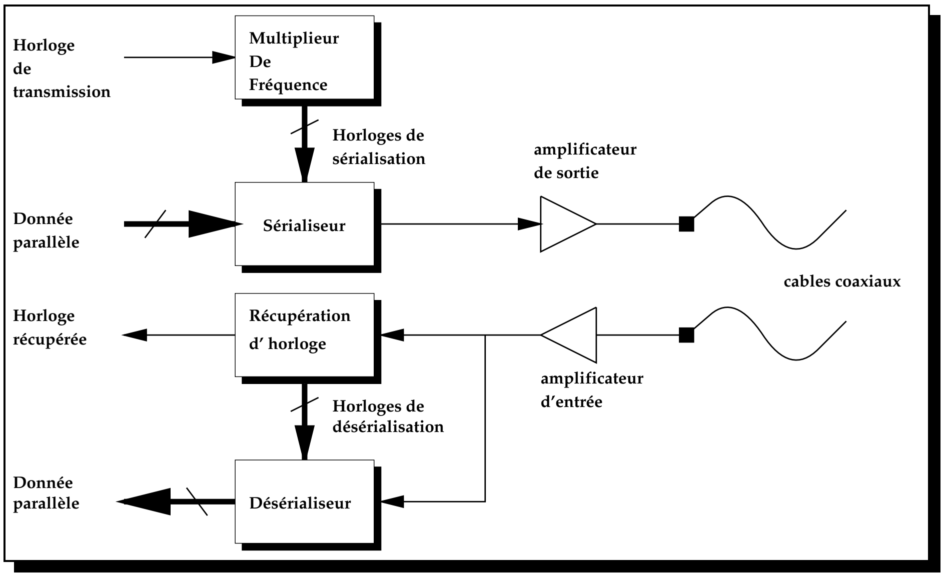


Figure 2.3: Synoptique de la macrocellule série HSL

liaisons série. Toutefois, la conservation d'un débit global identique à celui d'un bus constitue un défi technologique.

La technologie HSL (High Speed Link)[6] de lien série à 1 Gbit/s a été développée conjointement par BULL et le laboratoire MASI. Elle est conforme au standard IEEE 1355. La principale qualité de cette technologie est sa compacité: une macrocellule (figure 2.3) réalisant la sérialisation (en émission) et la désérialisation ainsi que la récupération d'horloge (en réception) occupe une surface de 1,5 mm^2 et consomme 0,3 W en technologie CMOS 0,5 μm-3,3V. La liaison série HSL transmet les données via deux câbles coaxiaux (un dans chaque direction) sur une distance pouvant atteindre jusqu'à 5 mètres avec une fiabilité qui atteint celle d'un bus "fond de panier". Un réseau constitué de routeurs communiquant grâce à ces simples câbles coaxiaux peut aisément être étendu, sous réserve, comme nous le verrons par la suite, que le schéma de routage mis en œuvre par le routeur le permette. Cette technologie offre donc l'avantage de permettre un débit élevé et une extensibilité facile du réseau, mais son utilisation entraîne des contraintes et soulève les difficultés suivantes.

1) La liaison série HSL transporte des caractères codés sur 12 bits. Les caractères reçus doivent être décodés selon un algorithme 9B/12B défini par le standard IEEE 1355. De la même manière, tout octet doit être codé sur 12 bits avant émission. La liaison série HSL induit donc un coût incompressible en terme de latence (temps de désérialisation et de sérialisation des données) auquel il faut ajouter le temps nécessaire au codage et au décodage 9B/12B.
2) Atteindre un débit de 1 Gbit/s par port requiert de la part du circuit d'être en mesure de recevoir de la macrocellule et d'envoyer vers celle-ci un mot de 12 bits toutes les 12 nanosecondes. Ceci pose la question de dimensionnement des

chemins de données internes. Si l'on est incapable de tenir un temps de cycle de 12 ns, il faut envisager de doubler la largeur des chemins de données. Cela autorise une fréquence interne de fonctionnement inférieure mais conduit à un accroîssement de la surface du circuit. Or, nous souhaitons utiliser le plus possible les outils de synthèse logique pour éviter une méthode de dessin "sur mesure" extrêmement coûteuse en temps de conception.

3) La technologie HSL est asynchrone et impose des mécanismes de resynchronisation dont il faut minimiser l'impact sur la latence.

2.3.3 Réseau sans perte en l'absence de pannes

Pour les applications que nous visons, il est essentiel de transmettre les paquets sans en perdre. Il est donc nécessaire de mettre en œuvre dans le routeur des mécanismes de contrôle de flux assurant qu'un caractère envoyé pourra être stocké s'il ne peut être routé immédiatement. Ce mécanisme de contrôle de flux bas-niveau est défini par le standard IEEE 1355. Pour un débit de 1 Gbit/s, ce contrôle ne peut se faire au niveau du caractère, ni au niveau du paquet puisque les paquets sont de taille quelconque.

2.3.4 Tolérance aux pannes

Dans un système comportant des centaines de nœuds de traitement, la probabilité d'une panne ne peut plus être négligée. Nos objectifs sont multiples:

- assurer l'acheminement des paquets malgré la rupture de certaines liaisons,
- pouvoir utiliser le système même lorsque certains de ses composants (routeurs, processeurs) sont en panne,
- détecter les ruptures de liaisons (panne franche ou panne transitoire),
- redémarrer les liaisons sans perturber le réseau en cas de panne transitoire.

2.3.5 Détection et traitement des erreurs de transmission

Une erreur de transmission peut avoir des conséquences diverses selon le caractère du paquet qui a été corrompu. Par exemple, si le ou les caractères codant la destination d'un paquet sont modifiés, le paquet est routé vers une destination erronée, ce qui peut conduire à des phénomènes de blocage du réseau. Deux attitudes peuvent être choisies: se contenter de détecter les erreurs de transmission et reporter leur gestion au niveau logiciel, ou mettre en œuvre des mécanismes matériels de correction ou de reprise sur erreur qui induisent un temps et une complexité de conception accrus. Du fait de la grande fiabilité des liaisons HSL, nous avons adopté le principe suivant: le matériel détecte, signale et minimise les risques de blocage dûs aux erreurs de transmission tandis que le logiciel se charge de leur traitement.

2.3.6 Évolutivité

Nous visions initialement la réalisation d'un routeur de valence 8, mais un des objectifs de cette étude est d'évaluer la faisabilité de routeurs de valence supérieure. Notre but a donc été de concevoir une architecture très modulaire permettant d'extrapoler le coût en surface et en latence de routeurs de valences supérieures.

2.3.7 Testabilité

Le test d'un routeur disposant de liaison série à 1 Gbit/s présente des difficultés particulières. En effet, les testeurs sont incapables de gérer des broches dont la fréquence de commutation atteint le gigahertz. Dès lors, notre accès au routeur ne peut se faire par ces liaisons. Il nous faut donc mettre en œuvre des solutions permettant malgré tout d'effectuer le test sur tranche, c'est à dire avant encapsulation.

2.4 Conclusion

Après avoir défini les domaines d'applications que nous visons, (les systèmes massivement parallèles de type MIMD), nous avons démontré la nécessité de construire des réseaux d'interconnexion dynamiques à base de routeurs capables d'aiguiller les messages selon l'information de destination qu'ils transportent dans leur en-tête.

Nous avons ensuite présenté deux machines représentatives des besoins actuels en matière de réseau d'interconnexion. Elles nous ont permis de déterminer quelles sont les caractéristiques fonctionnelles d'un routeur répondant à leurs besoins. Celles-ci sont résumées ci-dessous:

- faible latence (de l'ordre de 100 ns),
- haut débit (80 Mo/s pour chaque port),
- résistance à la congestion,
- détection des ruptures de liaisons et tolérance aux pannes,
- possibilité de router des paquets de taille quelconque,
- possibilité de construire des réseaux de topologie quelconque,
- détection des erreurs de transmissions,
- extensibilité.

Nous avons ensuite défini quels étaient nos objectifs et opéré certains choix. Notre but est de réaliser un routeur de paquets adaptatif à faible latence (100 ns) disposant de liaisons série à 1 Gbit/s. Cela a permis de mettre en lumière des difficultés de réalisation nous amenant à nous poser les questions suivantes.

- comment atteindre une latence de 100 nanosecondes?
- comment régler les problèmes d'asynchronisme sans dégrader la latence?

- comment implémenter l'adaptativité en minimisant son impact sur la latence?
- où se situe la chaîne critique qui limite les performances en fréquence du routeur. Sa position et sa valeur dépendent-elles de la valence du routeur?

CHAPITRE 3

Notions fondamentales

Le but de ce chapitre est de présenter les notions théoriques qui entrent en ligne de compte dans la définition de l'architecture d'un routeur, à savoir:

- les techniques de routage,
- les schémas de routage,
- le contrôle de flux,
- les stratégies de mémorisation,
- les problèmes d'interblocage.

3.1 Architecture générale

L'architecture d'un routeur s'articule autour d'un crossbar, d'une zone mémoire, de ports d'entrée contenant principalement un calculateur de route et de ports de sortie (Cf. figure 3.1).

- Le crossbar est constitué de deux parties principales, à savoir une matrice de commutation permettant de réaliser physiquement la connexion d'un port d'entrée vers un port de sortie, et un allocateur de sortie. Ce dernier a plusieurs fonctions:
  - il réalise l'arbitrage entre les requêtes simultanées faites par plusieurs ports d'entrée désirant accéder au même port de sortie,
  - il effectue et mémorise l'allocation d'un port de sortie à un port d'entrée,
  - il commande la matrice de commutation de façon à réaliser les connexions demandées.
- Le calculateur de route détermine vers quelle sortie orienter le paquet, selon l'information de destination qu'il transporte. Il émet vers l'allocateur de sortie la requête de connexion idoine.

- La zone mémoire a pour rôle de stocker les données reçues lorsque des contentions se produisent. Citons comme exemple de contention le cas où:
  - deux paquets issus de deux ports d'entrée différents sont en compétition pour accéder au même port de sortie qui ne peut en accepter qu'un;
  - le port de sortie que doit emprunter un paquet n'a pas l'autorisation d'émettre, à cause d'une contention en aval.

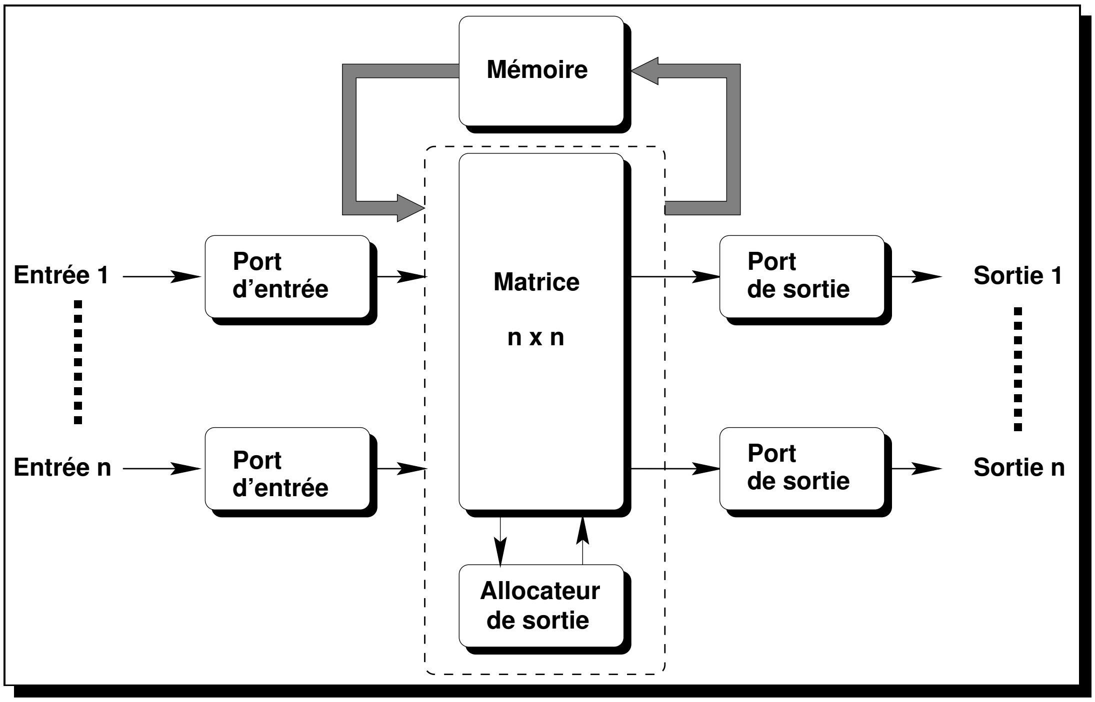


Figure 3.1: Architecture générale d'un routeur

3.2 Techniques de routage

La technique de routage détermine la façon dont les paquets sont transmis de l'émetteur vers le récepteur. L'approche intuitive consiste à établir un chemin entre l'émetteur et le récepteur puis d'envoyer les données dès que la connexion est établie. Cette technique est appelée **commutation de circuit**. L'avantage de cette technique est qu'après établissement de la connexion, aucune ressource de mémorisation n'est nécessaire tout au long du chemin emprunté par les données. Par contre, cette technique n'est réellement efficace que lorsque les échanges de données sont relativement long et continus entre deux nœuds (ex: en téléphonie). Dans les machines parallèles, les communications entre nœuds ont plutôt tendance à se faire sous forme de blocs de données de taille réduite (des lignes de cache par exemple) et requièrent des latences très faibles. De ce fait, le surcoût dû à la phase d'établissement de la connexion devient intolérable. Un autre inconvénient de la commutation de circuit est que la connexion demeure jusqu'à ce que la dernière donnée soit arrivée à destination. Les liaisons des routeurs qui participent à la connexion entre l'émetteur et le récepteur sont inutilisables par d'autres données durant ce laps de temps. Cela se traduit par une perte en bande passante et en latence.

La technique de routage **store-and-forward** consiste à stocker entièrement le message dans chacun des routeurs intermédiaires avant de le réémettre. Il est alors possible d'effectuer un contrôle d'intégrité des messages dans chaque routeur. Par contre la latence est directement proportionnelle au nombre de routeurs traversés.

Afin de diminuer la latence lorsqu'il n'y a pas de contention, Kermani et Kleinrock [7] ont introduit la technique **virtual-cut-through**. Dans cette technique, l'émission d'un message n'est pas subordonnée à sa complète réception si la sortie qu'il doit emprunter est libre. Les liaisons qu'utilisait le message sont alors disponibles pour router d'autres messages. Par contre il faut pouvoir stocker entièrement le message dans le routeur s'il ne peut être routé immédiatement. Cela impose que les messages ait une taille limitée; s'ils sont de taille variable, il est possible de les découper, préalablement à leur envoi, en paquets de taille fixe. L'espace de stockage requis peut donc être réduit à la taille maximale d'un paquet. Une utilisation plus optimale de la bande passante est possible puisque les ressources de mémorisation sont utilisées pleinement (exception faite du dernier paquet qui peut être de taille très faible). La latence totale de routage d'un message est réduite grâce à l'effet de pipe-line. En effet, plusieurs paquets d'un même message peuvent être en cours de transmission simultanément sur différents liens entre la source et la destination du message. Enfin, en cas de corruption des données, il suffit de retransmettre uniquement le ou les paquets affectés et non pas tout le message. En contrepartie, on introduit un surcoût en ajoutant à chaque paquet l'information de destination. Un autre inconvénient de la division du message en paquets est l'obligation de reconstruire le message en réception. Lorsque le routeur est adaptatif, la détection de la fin de reconstruction du message est plus difficile car, dans ce cas, le réseau ne garantit pas que les paquets arriveront dans l'ordre d'émission.

Afin de réduire la capacité mémoire minimale nécessaire dans chaque routeur tout en gardant une latence faible, deux chercheurs du Caltech: Dally et Seitz [8] ont proposé la technique de routage **wormhole** dans laquelle le message est divisé en unités insécables appelées *flit* pour *Flow Control Unit* composées de un ou plusieurs caractères. Le *flit* de tête contient la destination du message et progresse de routeur en routeur en entrainant à sa suite les *flits* constituant le reste du message. Le chemin créé par le *flit* de tête se referme au fur et à mesure de la progression de la queue du message, permettant ainsi aux canaux libérés d'être utilisés par d'autres messages. La décision de routage est prise dès que le *flit* de tête arrive sur un port d'entrée du routeur. Un message peut donc être réparti sur plusieurs routeurs à un moment donné (figure 3.2) et la tête d'un message peut arriver à destination avant que la fin n'ait été émise par la source. La latence est donc diminuée et il suffit que chaque routeur ait la place suffisante pour stocker au moins un *flit*. En contrepartie, le contrôle d'intégrité des messages ne peut plus se faire dans chaque routeur. Enfin, en cas de contention, le message reste bloqué interdisant à d'autres messages d'utiliser les canaux qu'il occupe.

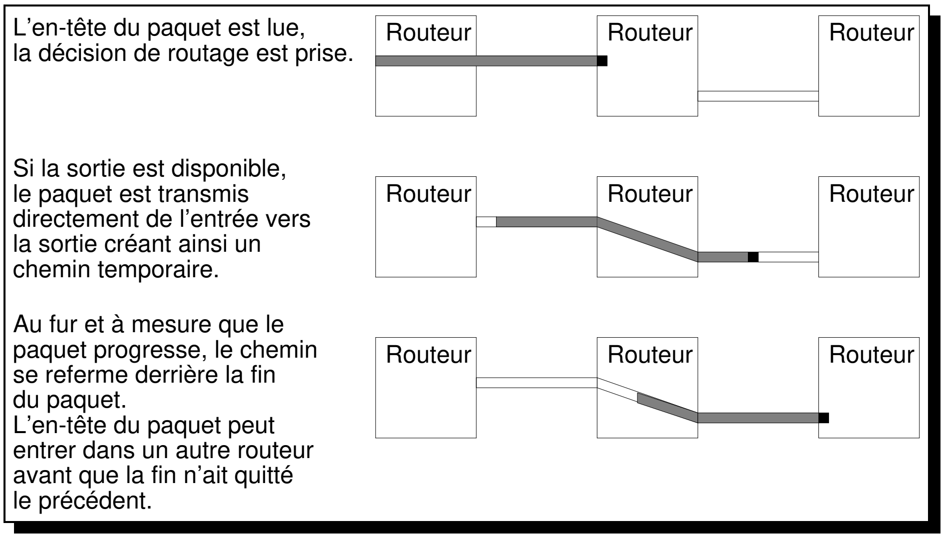


Figure 3.2: Routage *wormhole*

Les trois techniques *wormhole*, *virtual-cut-through* et *store-and-forward* ont en commun que les données de tête ouvrent le chemin au fur et à mesure de leur progression. Elles sont donc regroupées sous le terme générique de **commutation de messages** pour les différencier de la commutation de circuit.

Le tableau 3.1 résume les différentes caractéristiques des techniques de routage de type commutation de messages en terme de latence, mémorisation et contrôle d'intégrité.

Les paramètres utilisés sont:

- $\mathbf{L_h}$: longueur de l'en-tête,
- **L**: longueur du message (ou du paquet),
- **B**: bande passante de la liaison
- **D**: nombre de routeurs traversés.

Technique de routage	**Latence sans contention**	**Mémorisation minimale**	**Contrôle d'intégrité**
store-and-forward	$\frac{L+L_h}{B} \times D$	1 message (ou 1 paquet)	niveau routeur
virtual-cut-through	$\frac{L_h}{B} \times D + \frac{L}{B}$	1 message (ou 1 paquet)	niveau récepteur
wormhole	$\frac{L_h}{B} \times D + \frac{L}{B}$	1 flit	niveau récepteur

Table 3.1: Techniques de routage de type commutation de messages

3.3 Schémas de routage

Définition: on appelle **schéma de routage**, la structure de données sur laquelle s'appuie le calculateur de route pour choisir la sortie que doit emprunter un message arrivé sur un de ses ports d'entrée.

L'importance du choix du schéma de routage réside dans le fait qu'il détermine le type d'algorithmes de routage que le routeur peut implémenter. Les algorithmes de routage sont mis en œuvre par le calculateur de route et permettent de choisir la sortie que doit prendre le message pour atteindre sa destination. Par la suite, nous utiliserons indifféremment le terme algorithme ou fonction de routage.

Les schémas de routage peuvent être **déterministes** ou **adaptatifs**. Dans le premier cas, il existe un chemin unique entre l'émetteur d'un message et son récepteur. Dans le second cas, si le chemin "normal" est déjà occupé, le routeur peut choisir de router le message par un chemin auxiliaire. L'adaptativité permet de mieux répartir le trafic dans le réseau et ainsi de réduire les phénomènes de congestion. Elle permet aussi d'offrir un certain degré de tolérance aux fautes. Cependant l'adaptativité implique de faire un choix parmi plusieurs solutions de routage, ce qui influe sur la latence. Cet impact doit être minimisé. D'autre part, un schéma de routage adaptatif ne garantit pas que les différents paquets associés à un même message arriveront dans l'ordre d'émission. Cela rend la reconstruction du message plus difficile. Mais, qu'ils soient déterministes ou adaptatifs, les schémas de routage doivent permettre la définition de fonctions de routage non bloquantes (absence de *deadlock*) et telles que les données arrivent à destination en un temps fini (absence de *live-lock*). Enfin, un dernier critère de choix d'un schéma de routage peut être le compromis qu'il offre entre la taille de la partie du message codant sa destination et la surface de silicium dédiée dans le routeur au choix de la sortie.

On peut distinguer deux types extrêmes de schéma de routage: le schéma de routage par table complète centralisée encore appelé schéma de routage explicite ou à la source, et le schéma de routage par table distribuée.

3.3.1 Schéma de routage explicite

Dans le schéma de routage explicite, le nœud émetteur dispose d'une table où sont stockés un ou plusieurs chemins permettant de rallier n'importe quel autre nœud du réseau. Il inclut dans l'en-tête du message la description de ce chemin. En général ces informations correspondent au numéro de sortie à prendre dans le routeur de niveau $i_{1..n}$, en considérant qu'il y a n routeurs entre l'émetteur et la destination du message. Dans ce type de schéma de routage, la taille de l'en-tête du message, proportionnelle à la taille du réseau, conduit à une perte en bande passante dans un système comportant plusieurs milliers de nœuds. D'autre part, en routage explicite, l'adaptativité ne peut être mise en œuvre qu'à la source. Ce schéma de routage ne peut tenir compte des éventuelles ruptures ou occupations de liaisons qui surviennent en cours de transfert d'un message et router le message par un chemin auxiliaire. La figure 3.3 illustre un exemple de réseau (un cube) sur lequel on utilise un schéma de routage explicite.

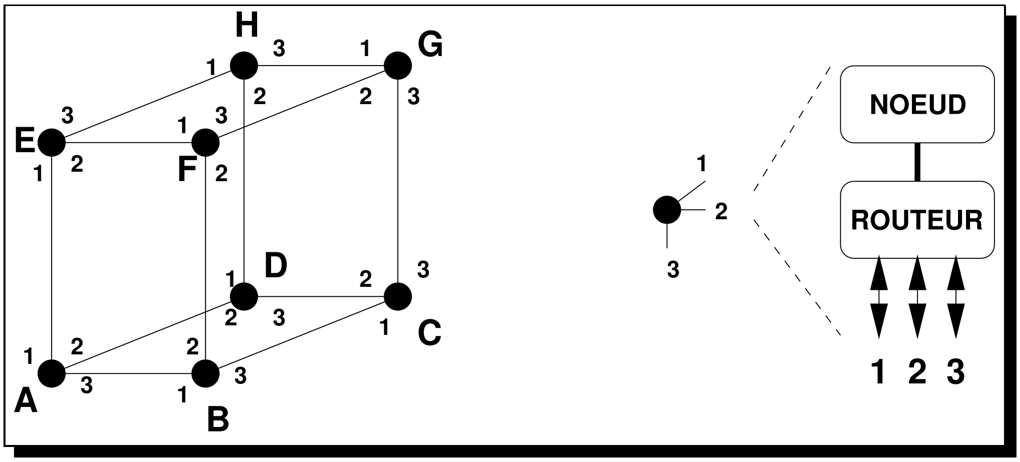


Figure 3.3: Schéma de routage explicite sur un cube

La table 3.2 présente un extrait du contenu d'une table de routage dans laquelle sont décrits un ou plusieurs chemins pour aller du nœud A vers les autres nœuds du réseau.

Nœud émetteur	Nœud récepteur	Chemins possibles
A	B	3
	C	(3,3) ; (2,3)
	D	2 ; (1,3,2) ; (3,3,2)
	E	1
	F	(3,2) ; (2,2)
	G	(3,3,3) ; (1,2,3)
	H	(2,1)

Table 3.2: Extrait d'une table de routage en routage explicite

3.3.2 Schémas de routage par tables distribuées

Des tables situées dans chaque routeur, contiennent les informations permettant de choisir une sortie selon l'information de destination que transporte le message dans son en-tête. Cette information de destination est en général le numéro du nœud destinataire. Lorsque le message arrive sur l'un des ports d'entrée d'un routeur, le calculateur de route associé à ce port regarde dans la table quelle est le port de sortie que doit emprunter le message.

Chaque port d'entrée peut avoir une table propre ou bien partager une table unique avec tous les autres. Le fait de disposer d'une table par port d'entrée offre plusieurs avantages. Premièrement, il n'est pas nécessaire de gérer les éventuels accès simultanés des ports d'entrée à la table ni de réaliser, pour remédier à cet inconvénient, des tables à plusieurs ports de lecture. Deuxièmement, il est possible de réaliser plusieurs réseaux logiques disjoints sur un même réseau physique (partitionnement). Ceci est une façon d'émuler un degré de priorité des messages alors que le protocole de communication n'en prévoit pas. Ces avantages s'obtiennent évidemment au détriment d'une surface de silicium occupée plus importante.

Considérant qu'un réseau peut contenir plusieurs milliers de nœuds et donc de destinations, l'utilisation de tables de routage complètes (une entrée par nœud dans le réseau), requiert une taille mémoire embarquée importante, et ne permet pas d'étendre la taille du réseau puisque la taille mémoire est fixée par construction. Cette contrainte est aggravée lorsqu'on désire mettre en œuvre des algorithmes de routage adaptatifs, où l'on peut utiliser plusieurs sorties pour atteindre une même destination. Il est donc intéressant de pouvoir utiliser des tables de routage compactes. Cette compacité est obtenue en effectuant un codage de l'information de routage.

Schéma de routage "câblé"

Si la topologie du réseau est fixe et présente des caractéristiques de régularité, il est possible de définir une fonction de routage *R* qui permette à n'importe quel nœud du réseau d'aiguiller un message vers sa destination en calculant *R(h)* dans laquelle h est l'en-tête du message. Le câblage matériel de la fonction de routage *R* est une façon de réaliser la compaction de l'information de routage.

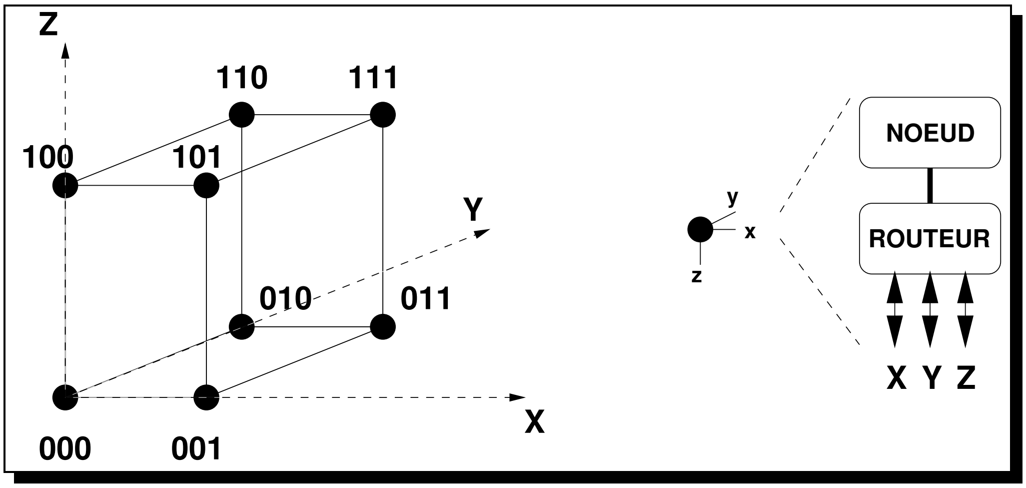


Figure 3.4: Schéma de routage "câblé" dans un cube

Par exemple, le routeur du T3D de CRAY utilise un algorithme de routage sur tore tridimensionnel par dimension ordonnée. L'émetteur est défini par ses coordonnées *(xe, ye, ze)*, le récepteur par ses coordonnées *(xd, yd,zd)*. Une donnée est routée d'abord selon l'axe des X de façon que x=xd, puis ensuite selon l'axe des y afin que y=yd et enfin selon l'axe des z jusqu'à ce que z=zd. Chaque routeur doit simplement connaître sa propre position dans le réseau. Un exemple de cet algorithme de routage appliqué au réseau de la figure 3.4 peut être défini matériellement de la façon suivante:

- La sortie X du routeur est prise si: *xe* XOR *yd* = 1
- La sortie Y du routeur est prise si: (*ye* XOR *yd*) AND NOT X = 1
- La sortie Z du routeur est prise si: (*ze* XOR *zd*) AND NOT (X OR Y) = 1

Schéma de routage par intervalle

Une autre technique de compaction des tables de routage est le schéma de routage par intervalle introduit par Santoro et Khatib[9] et développé par Van Leeuwen et Tan[10]. Son principe est le suivant: chaque nœud du réseau est identifié par un numéro unique, ce numéro étant utilisé comme adresse de destination dans l'en-tête du paquet. L'espace des adresses possibles est divisé en intervalles. Chaque routeur dispose d'une ou plusieurs tables contenant les séparateurs d'intervalles.

A chaque intervalle est associé un port de sortie; seuls les messages dont l'en-tête appartient à cet intervalle sont routés vers ce port de sortie. Lorsque l'en-tête d'un message arrive sur un port d'entrée, il est comparé simultanément aux séparateurs d'intervalles afin de déterminer à quel intervalle il appartient. Cette méthode impose que les intervalles soient consécutifs et non recouvrants, afin de s'assurer que l'en-tête n'appartienne qu'à un unique intervalle. Le port de sortie associé à l'intervalle élu est choisi pour router le message, si toutefois il est disponible.

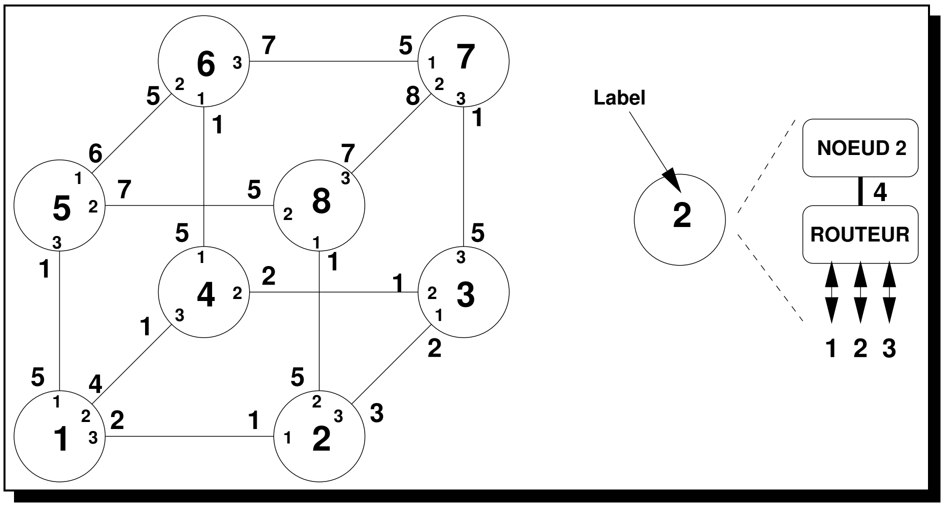


Figure 3.5: Schéma de routage par intervalle dans un cube

La figure 3.5 présente un exemple d'étiquetage d'un cube. Les tables de routage contenues dans les routeurs sont décrites ci-après. La méthode utilisée pour déterminer le contenu des tables de routage est tirée de [11].

Borne d'intervalle	Intervalle défini	Sortie associée	Borne d'intervalle	Intervalle défini	Sortie associée
routeur 1			**routeur 3**		
borne0 = 1	[1,2[	4	borne0 = 1	[1,2[	2
borne1 = 2	[2,4[	3	borne1 = 2	[2,3[	1
borne2 = 4	[4,5[	2	borne2 = 3	[3,5[	4
borne3 = 5	[5,8]	1	borne3 = 5	[5,8]	3
routeur 2			**routeur 4**		
borne0 = 1	[1,2[	1	borne0 = 1	[1,2[	3
borne1 = 2	[2,3[	4	borne1 = 2	[2,4[	2
borne2 = 3	[3,5[	3	borne2 = 4	[4,5[	4
borne3 = 5	[5,8]	2	borne3 = 5	[5,8]	1

Table 3.3: Exemple de contenu des tables en routage par intervalle

Borne d'intervalle	Intervalle défini	Sortie associée	Borne d'intervalle	Intervalle défini	Sortie associée
routeur 5			**routeur 6**		
borne0 = 1	[1,2[	3	borne0 = 1	[1,5[	1
borne1 = 5	[2,6[	4	borne1 = 5	[5,6[	2
borne2 = 6	[6,7[	1	borne2 = 6	[6,7[	4
borne3 = 7	[7,8]	2	borne3 = 7	[7,8]	3
routeur 7			**routeur 8**		
borne0 = 1	[1,2[	3	borne0 = 1	[1,5[	1
borne1 = 5	[2,7[	1	borne1 = 5	[5,7[	2
borne2 = 7	[7,8[	4	borne2 = 7	[7,8[	3
borne3 = 8	[8]	2	borne3 = 8	[8]	4

Table 3.4: Exemple de contenu des tables en routage par intervalle (suite)

3.3.3 En résumé

Nous avons vu que le schéma de routage conditionne le type d'algorithmes de routage que le routeur peut réaliser (déterministe ou adaptatif). Les deux types extrêmes de schéma de routage sont le schéma de routage à la source et le schéma de routage par tables distribuées. Des algorithmes de routage plus efficaces et une émulation de plusieurs réseaux logiques disjoints sur un même réseau physique peuvent être mis en œuvre s'il existe une table de routage par port d'entrée. Cependant, cela conduit à une augmentation de la surface de silicium utilisée, surtout si ces tables sont complètes et que le réseau comporte un grand nombre de nœuds. L'autre inconvénient des tables complètes est que le réseau ne peut être étendu. Pour minimiser cette surface, il est possible de procéder à un codage de l'information de routage mais en perdant de la souplesse au niveau de l'algorithme de routage.

3.4 Contrôle de flux

Un réseau à commutation de messages consiste en un ensemble de ressources (liaisons, tampons mémoire) qui doit être partagé dynamiquement entre plusieurs nœuds désirant communiquer les uns avec les autres. Si l'attribution dynamique des ressources offre un plus grande flexibilité et de meilleures performances que l'approche commutation de circuit, cela s'obtient au prix d'un coût qui est le contrôle de l'allocation de ces ressources. En effet, sans contrôle, la demande de ressources peut très vite saturer le réseau et conduire à des phénomènes de congestion qui font chuter dramatiquement les performances (débit, latence). Cette dégradation peut même aller jusqu'au blocage complet du réseau (*deadlock*). On peut arguer du fait que la congestion du réseau peut être réduite en procédant à des reroutages des paquets vers des chemins sous-utilisés afin de soulager les chemins surchargés. Cependant il faut être conscient du fait que si ce reroutage peut réduire la congestion ou la retarder, il ne peut en aucun l'empêcher. C'est pourquoi il est capital de mettre en œuvre des techniques de contrôle de flux dont le rôle est:

- éviter les pertes d'informations,
- prévenir la dégradation du débit global et de l'efficacité du réseau dues à la surcharge,
- prévenir les interblocages. Ce point fera l'objet de la section 3.6,
- allouer de façon équitable les ressources aux différents utilisateurs du réseau (prévention des famines),
- ajuster les vitesses de fonctionnement entre le réseau et les nœuds qui y sont attachés.

Afin de réaliser ces fonctions, les procédures de contrôle de flux incluent un ensemble de contraintes sur, par exemple, l'allocation des liaisons, des tampons mémoire, sur le taux d'émission de paquets par les nœuds etc... Ces contraintes ont pour but de limiter l'accès au réseau ou à certaines parties du réseau pour prévenir la surcharge, de conditionner l'allocation de ressources pour éviter les interblocages ou restreindre la quantité de ressources que peut acquérir un utilisateur afin de prévenir les famines. Ces contraintes peuvent être fixes ou ajustées dynamiquement selon le trafic.

Le contrôle de flux réclame un échange d'information entre les acteurs du réseau. À partir des informations obtenues sur l'état des tampons des routeurs et l'état des liaisons (occupation, fonctionnalité), il est possible de réagir en conséquence. Cependant, cette connaissance de l'état du réseau n'est pas instantanée et consomme de la bande passante puisqu'il faut véhiculer ces informations. Malgré ceci on peut voir sur la figure suivante tirée de [12] que le contrôle de flux est indispensable au bon fonctionnement du réseau.

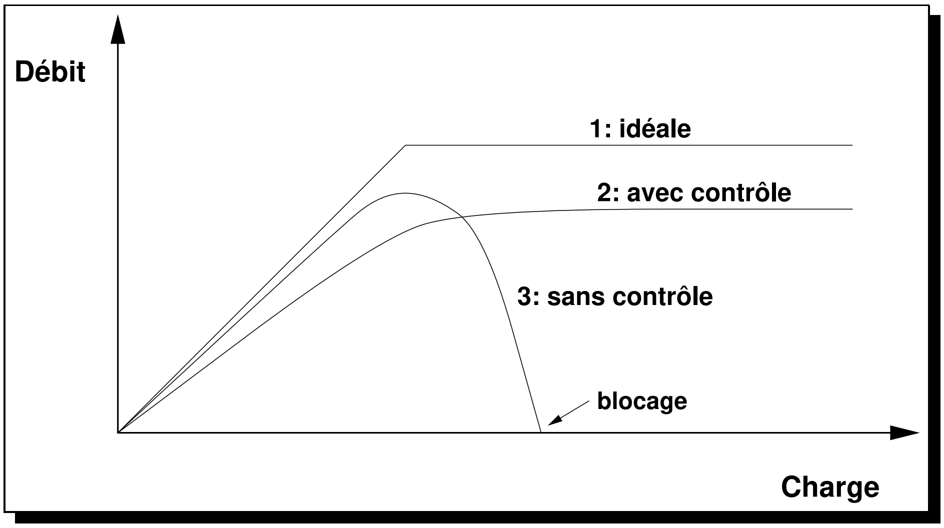


Figure 3.6: Effet du contrôle de flux

Sur cette figure, les 3 courbes expriment le débit en fonction de la charge. La courbe 1 que l'on appellera idéale correspond à un contrôle parfait effectué à partir d'informations complètes et instantanées sur l'état du réseau. À partir d'une certaine charge, le débit atteint un niveau de saturation qui est le débit maximal théorique du réseau. La courbe 3 correspond à un réseau dans lequel aucun contrôle de flux n'est effectué. Pour des faibles charges on remarque que celle-ci est très proche de la courbe idéale, mais dès que la charge atteint des valeurs moyennes, les performances chutent et la situation peut empirer jusqu'à atteindre le blocage du réseau. La courbe 2 illustre l'évolution du débit du réseau en fonction de la charge dans un réseau où existe des moyens de contrôle de flux. Le débit réel ne peut atteindre le maximum théorique du fait d'un contrôle imparfait et du pourcentage de la bande passante utilisé pour transporter les informations de contrôle de flux. Cependant les phénomènes de blocage sont évités.

Le contrôle de flux peut s'exercer à plusieurs niveaux. Ainsi on peut distinguer:

- le niveau liaison: il assure que toute donnée émise pourra être stockée par le récepteur si elle ne peut être routée immédiatement. Ce contrôle s'effectue sur la base d'unité de contrôle de flux qui peut être l'octet, le *flit* en routage *wormhole* ou le paquet en routage *store-and-forward*. Cette unité de contrôle de flux est la plus petite unité indivisible qui peut être acceptée par le récepteur.
- le niveau paquet: il définit le format et le protocole d'échange des paquets entre nœuds. Il vérifie que les paquets ou les messages reçus sont complets, non corrompus et demande le cas échéant la retransmission du message ou du paquet.
- le niveau accès au réseau: il régule le taux d'émission des nœuds attachés au réseau afin de ne pas le surcharger. Ceci est effectué sur la base d'information

envoyées explicitement par les routeurs (par ex: taux d'occupation des tampons, des liaisons) ou par effet rétroactif du contrôle de flux au niveau liaison.

Cette décomposition en 3 niveaux peut se réduire à deux selon la technique de routage choisie. En effet, si l'on considère la technique *store-and-forward*, l'unité de contrôle de flux est le paquet. En conséquence, il n'y a pas de différence entre les niveaux liaison et paquet.

3.5 Stratégies de mémorisation

Un routeur doit accepter l'arrivée simultanée de messages sur ses N ports d'entrée tandis qu'il transmet des données à travers chacun de ses N ports de sortie.
Si l'on essaye de définir un routeur idéal, on peut citer deux caractéristiques principales:

- Il dispose d'une capacité de stockage infinie: un message est stocké si la sortie qu'il doit emprunter est indisponible.
- Un message stocké disposant d'une sortie libre n'est pas bloqué par un message stocké précédemment et en attente d'une autre sortie.

Hélas, dans la réalité, les tampons mémoire ont une taille et une bande passante limitées. Il s'ensuit que des conflits peuvent apparaitre lors d'accès simultanés à des tampons partagés, par exemple. L'organisation des tampons (indépendants ou partagés) et leur position (en entrée, centrale, en sortie) affectent les performances du routeur. Nous allons faire l'inventaire des trois principales solutions existantes en spécifiant pour chacune d'elles leurs contraintes et avantages.

3.5.1 Mémorisation centralisée partagée

L'utilisation d'un tampon centralisé partagé signifie que les messages arrivant de n'importe quel port d'entrée sont dirigés vers un unique tampon dont l'espace disponible est alloué dynamiquement. Un tel tampon est appelé CBDA (*centrally-buffered, dynamically-allocated*)[13]. Cette approche dont l'architecture générale est présentée sur la figure 3.7 permet une utilisation plus efficace de l'espace disponible qu'une approche où l'espace de mémorisation serait a priori divisé de façon statique entre les ports d'entrée. Par contre, une telle approche nécessite, dans le pire cas, un tampon dont la bande passante doit être égale à la somme des bandes passantes de tous les ports. Ainsi, dans un routeur $N \times N$, le tampon doit avoir $2 \times N$ ports afin de permettre N accès simultanés en lecture et N accès simultanés en écriture. Si l'on désire fabriquer un routeur disposant d'un plus grand nombre de ports d'entrées/sorties, il faut donc développer un nouveau CBDA. En outre, des études [14, 15, 16] ont montré qu'avec un tampon centralisé partagé, un seul port de sortie congestionné peut conduire au remplissage d'une grande partie de l'espace disponible et affecter les autres

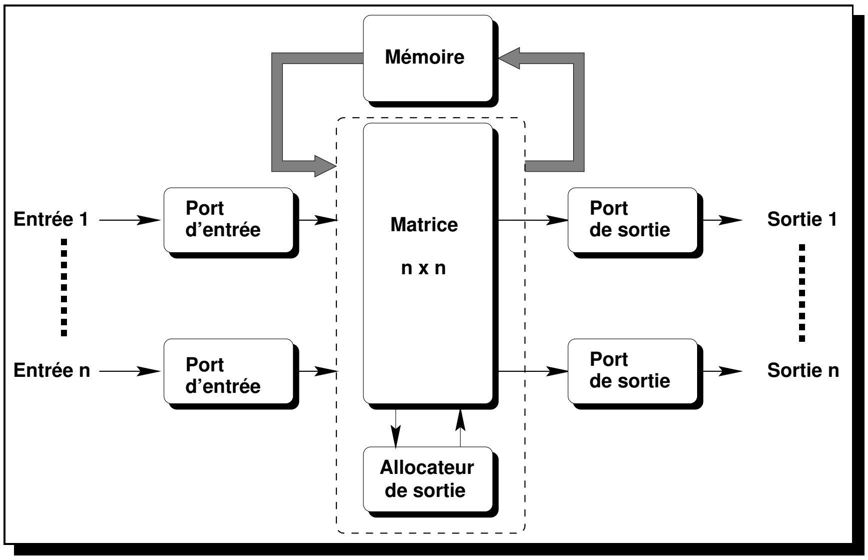


Figure 3.7: Architecture à mémoire centralisée

communications à travers le routeur. Ceci a pour conséquence d'empêcher les routeurs voisins de transmettre leurs données au routeur dont le tampon est plein, et de propager la congestion, au départ localisée dans un seul routeur, à tout le système. Pour éviter ce phénomène, il est possible de limiter l'espace alloué à un seul port mais cela accroit la complexité de la réalisation[15, 16].

3.5.2 Mémorisation en entrée

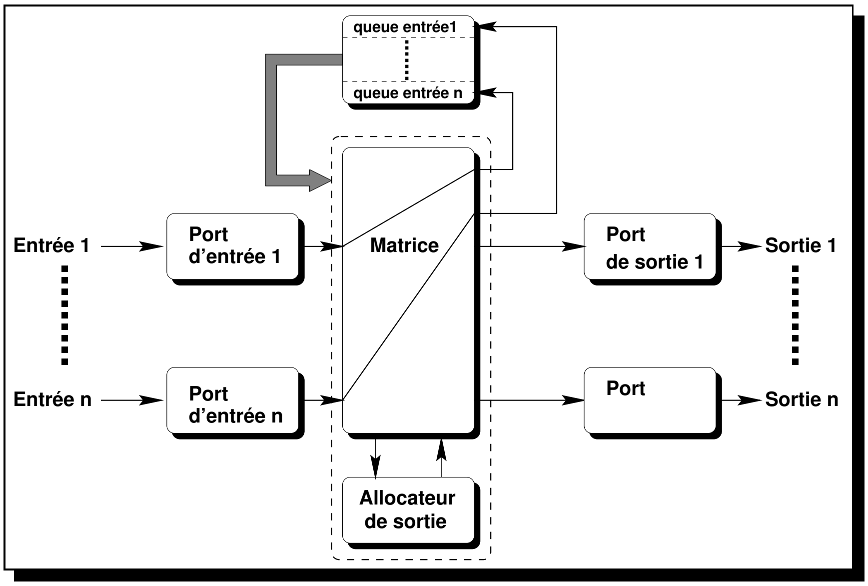


Figure 3.8: Architecture de routeur à mémorisation en entrée

L'architecture générale d'un routeur à mémorisation en entrée est illustrée sur la figure 3.8. Dans cette approche, l'espace mémoire est divisé en autant de partitions qu'il y a de ports d'entrée. Ces partitions forment des tampons indépendants. L'utilisation de tampons indépendants en entrée a pour avantage de ne nécessiter que des tampons disposant d'un seul port d'écriture puisqu' à un moment donné au plus un message arrive d'un port d'entrée. Plusieurs implantations de cette architecture peuvent être mises en œuvre: certaines de ces architectures reposent sur une division statique de l'espace global, d'autres sur une allocation dynamique de l'espace mémoire. Tamir présente dans [13] une étude comparée de plusieurs exemples d'architecture de tampons placés en entrée d'un routeur. Trois approches sont passées en revue: l'approche FIFO, l'approche SAFC/SAMQ et l'approche DAMQ.

Tampon FIFO

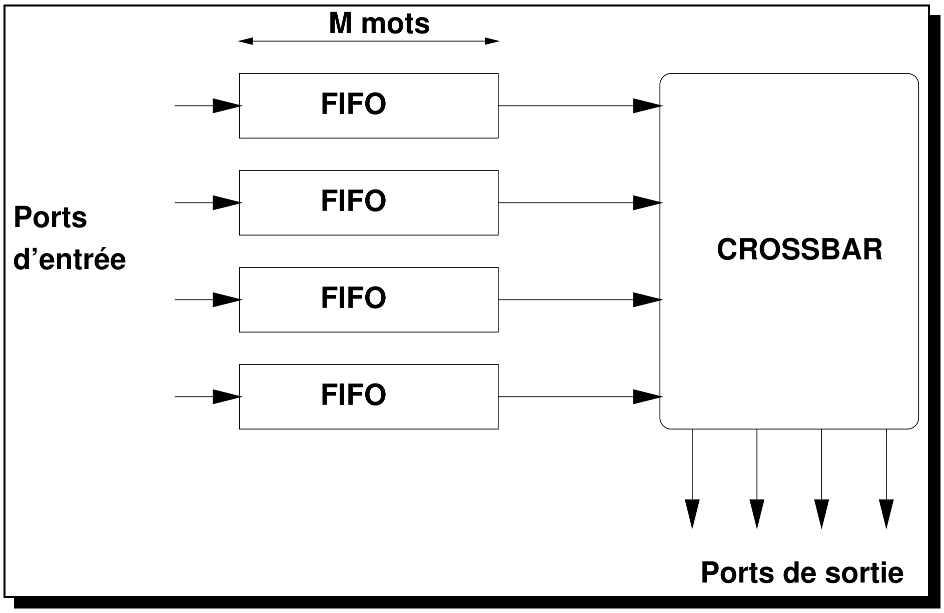


Figure 3.9: Exemple d'architecture à tampons FIFO (N=4)

Une FIFO est un type de mémoire organisée de façon à restituer les données dans l'ordre de leur écriture. Les réalisations les plus plus courantes disposent d'un seul port d'écriture et de lecture. En utilisant une FIFO comme tampon (Cf.figure 3.9), il est très simple de composer avec des messages de taille variable puisqu'il n'est plus nécessaire de gérer l'allocation de l'espace mémoire accordé à chaque port comme dans les tampons CBDA. Cette approche est très modulaire et permet facilement de réaliser des routeurs ayant un nombre N d'entrées quelconque. Enfin, la connexion des entrées et des sorties peut être assurée par un simple d'un crossbar N $\times$ N.

L'inconvénient des tampons FIFO placés en entrée réside dans le fait que des messages peuvent être bloqués, même si le port de sortie qu'ils devraient emprunter est disponible, du simple fait qu'ils sont bloqués derrière un message à destination d'un port de sortie indisponible. Cela entraine deux conséquences: premièrement les ports de sorties peuvent être sous-utilisés, deuxièmement, et c'est là le point le plus négatif, la charge maximale théorique acceptée avant saturation est inférieure à 60%[17].

Tampons SAFC/SAMQ

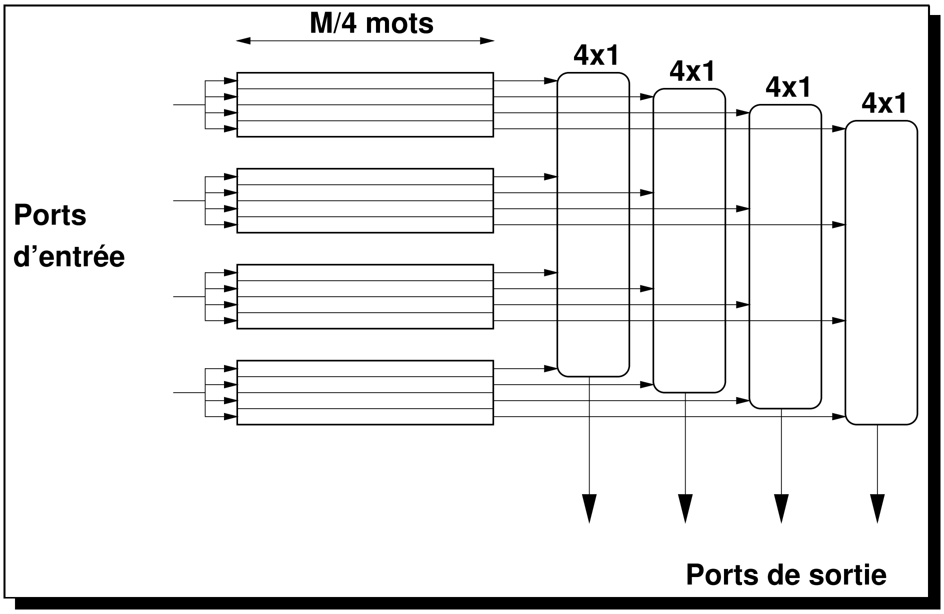


Figure 3.10: Exemple d'architecture à tampons SAFC/SAMQ (N=4)

Afin de remédier à ce problème et d'améliorer le débit, Tamir présente une alternative dans laquelle les messages peuvent être triés selon le port de sortie qu'ils prendront. Cette solution consiste à placer à chaque entrée N FIFO (N étant le nombre de ports de sortie du routeur) dont on multiplexe les ports d'écriture[18, 19] (Cf. figure 3.10). La FIFO résultante dispose de N ports de lecture que l'on peut multiplexer pour simplifier la gestion du crossbar interne.

Cette architecture soulève plusieurs problèmes. En premier lieu le surcoût matériel: il faut gérer à chaque entrée N tampons séparés. En outre, le fait de séparer de façon fixe la FIFO d'entrée en N tampons, fait que seule une fraction de l'espace mémoire total est disponible pour un message. Dans le cas de trafic non uniforme où des sorties sont plus sollicitées que d'autres, les tampons associés à ces sorties sont souvent pleins, interdisant la réception de messages, bien qu'il existe potentiellement de la place dans les tampons adjacents. Ceci est à comparer avec l'approche précédente où tout l'espace disponible est accessible aux messages quelle que soit leur destination.

L'inconvénient majeur de cette approche concerne le contrôle de flux, dans le cas où un port de sortie n'est autorisé à émettre que si le port d'entrée du routeur destinataire dispose d'un espace mémoire suffisant (pas de perte de paquets). Lorsque N tampons séparés existent en entrée d'un port, c'est N fois plus d'informations sur l'état des tampons par rapport à un tampon FIFO qui doit être transmis à l'émetteur. Avant d'émettre, un port de sortie doit donc effectuer un pré-routage du message pour pouvoir déterminer quelle capacité de stockage est disponible en réception.

Tampon DAMQ

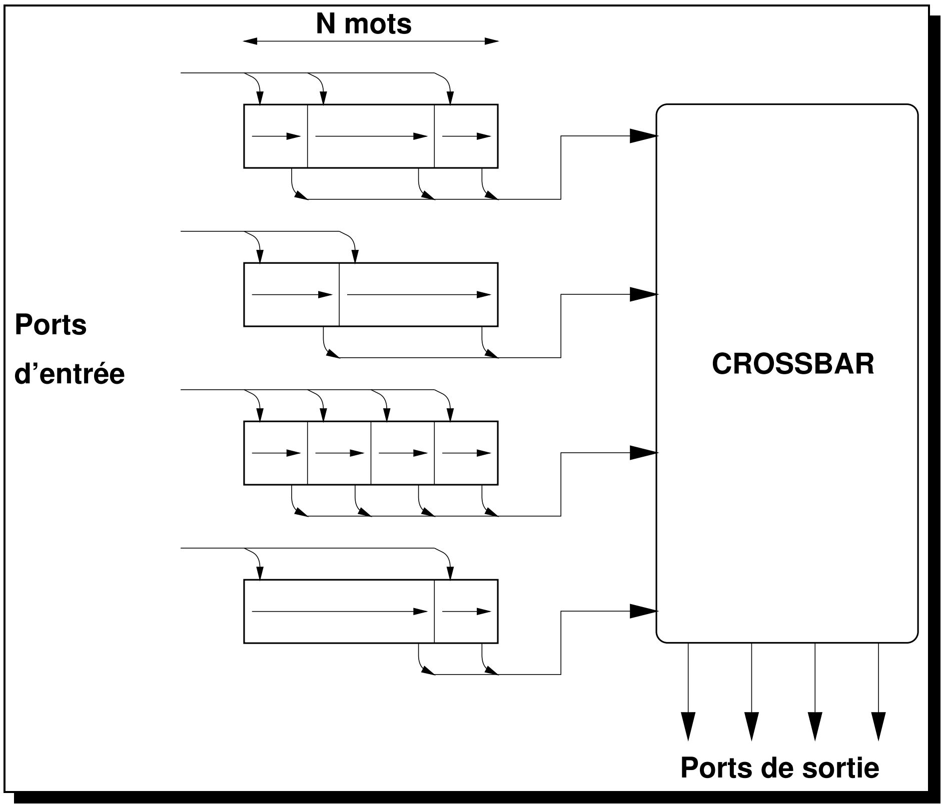


Figure 3.11: Exemple d'architecture à tampon DAMQ

Afin de remédier au problème intrinsèque du tampon FIFO et s'affranchir du pré-routage nécessité par l'approche SAFC/SAMQ, Tamir propose une architecture de tampon DAMQ (*dynamically-allocated, multi-queue*). Son principe est le suivant: l'espace mémoire associé à un port d'entrée est divisé en N queues de type FIFO associées aux N sorties, dont la taille varie dynamiquement selon les messages reçus (Cf. figure 3.11). Ainsi un message ne peut être bloqué parce qu'un autre est en attente d'une sortie disponible. L'autre avantage concerne le contrôle de flux car, dans cette approche, seule l'information de remplissage du tampon global doit être transmise au routeur émetteur (le pré-routage n'est plus obligatoire). Par contre la gestion du tampon DAMQ, basée sur des listes chaînées fait que sa réalisation matérielle est lourde.

3.5.3 Mémorisation en sortie

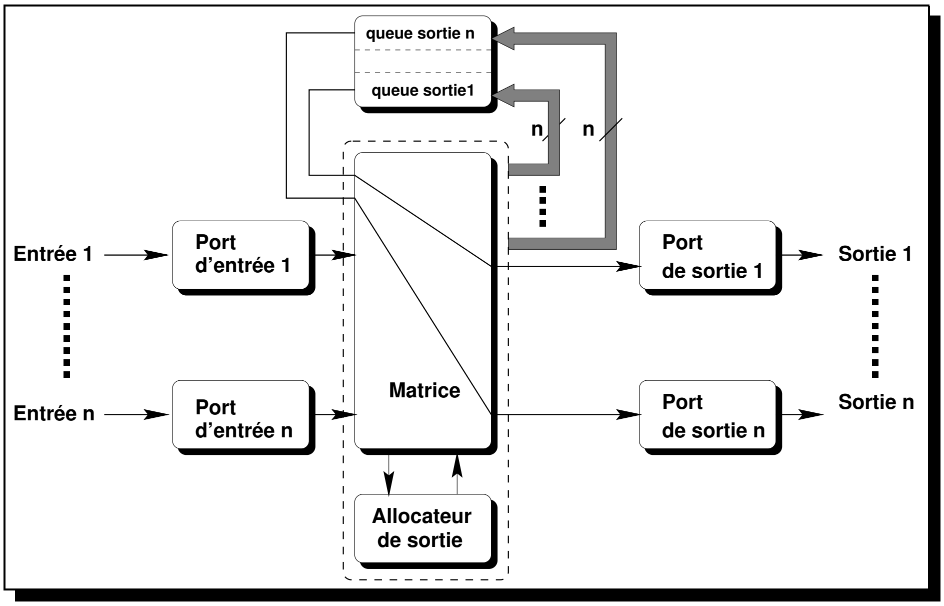


Figure 3.12: Architecture de routeur à mémorisation en sortie

L'architecture générale d'un routeur à mémorisation en sortie est présentée sur la figure 3.12. Dans cette architecture, l'espace mémoire est divisé en autant de partitions qu'il existe de ports de sortie. Chacun des ports d'entrée peut écrire, après analyse de l'en-tête, dans la partition correspondant à la sortie à prendre, et plusieurs entrées peuvent accéder simultanément à n'importe quelle partition. Karol[17] a montré que les architectures à mémorisation en sortie sont sensiblement plus performantes en terme de charge supportée avant saturation que les architectures à mémorisation en entrée de type FIFO.

Une implantation possible utilise des tampons FIFO. Le problème qui se pose est de disposer de FIFO qui soient capables de recevoir N fois le débit d'un port d'entrée, puisque N ports d'entrée peuvent émettre des messages à destination de la même sortie. Plusieurs solutions sont envisageables. Réaliser une FIFO qui dispose de N ports d'écriture ou d'une FIFO n'ayant qu'un port d'écriture multiplexé mais fonctionnant N fois plus vite que les ports d'entrée. La première solution conduit à une réalisation matérielle sur-mesure, coûteuse en temps et en surface, qui dépend du nombre d'entrée du routeur. La seconde solution n'est raisonnablement envisageable que si le produit $N \times F_{port}$ (où F_{port} est la fréquence à laquelle le port d'entrée émet une donnée vers la sortie) est inférieur à quelques dizaines de MHz en utilisant les technologies actuelles (CMOS 0.8 ou 0.5 μm).

Ajoutons que la stratégie de mémorisation en sortie nécessite la présence d'un contrôle de flux interne entre les ports d'entrée et les ports de sortie du routeur. En effet, dans le cas de l'approche FIFO, si la sortie ne peut émettre à cause d'une contention en aval, la FIFO se remplit. Il faut informer les entrées qui transfèrent de données vers cette FIFO de stopper leur émission avant que celle-ci ne soit pleine, sous peine d'écraser des données.

3.5.4 En résumé

De la stratégie de mémorisation dépendent les performances du routeur en terme de charge maximale supportée. Les études effectuées montrent que les meilleurs résultats en terme de seuil de saturation sont obtenus avec les approches mémorisation en sortie ou les approches à tampon centralisé. Cependant, ces performances sont obtenues au prix d'une grande complexité matérielle. Comme très souvent, dans de telles situations, le concepteur doit trouver un compromis entre les objectifs visés et les moyens mis en œuvre pour les atteindre. C'est pourquoi des outils de simulation d'architectures de routeurs dans diverses topologies de réseaux et sous différents modèles de trafic sont une aide précieuse pour le concepteur. C'est dans ce but que MILE (*Methodology for Interconnection networks Labeling and Evaluation*) a été développé au laboratoire MASI par Abdelhafid Bouaraoua.

3.6 Problèmes d'interblocage

En général, un phénomène d'interblocage a pour origine un mauvais ordonnancement dans l'allocation de ressources, ce qui induit une dépendance cyclique de leur attribution. Ce phénomène peut être expérimenté très souvent lorsque l'on se risque en voiture à Paris. Un exemple d'interblocage est décrit sur la figure 3.13. Dans cet exemple, les véhicules de tête (en grisé) désirent tourner à gauche mais ne peuvent avancer. Une mauvaise gestion des feux tricolores a conduit à une situation inextricable.

Dans un réseau de communications, l'interblocage se manifeste par l'impossibilité qu'ont un certain nombre de messages ou paquets de progresser dans le réseau. Les causes de ce phénomène sont dépendantes de nombreux paramètres. On peut citer de façon non exhaustive: l'algorithme de routage, la topologie du réseau, le temps mis par les récepteurs pour consommer les messages qui leur parviennent, le protocole de communication, le mode de génération des messages etc...

Plusieurs approches préventives ou préemptives ont été étudiées pour résoudre les problèmes d'interblocage. Les méthodes préemptives consistent à casser l'interblocage lorsqu'il survient: des paquets sont donc détruits et doivent être reroutés. Les méthodes préventives sont de loin les plus utilisées. Nous présentons donc quelques unes de ces méthodes utilisées en routage *store-and-forward*, avant de citer quelques méthodes utilisées en routage *wormhole*.

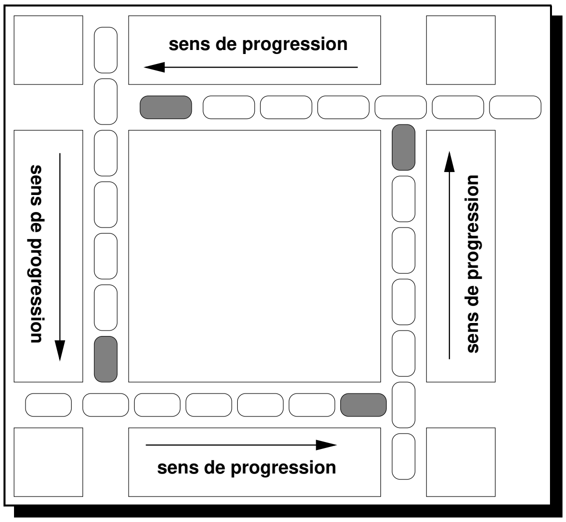


Figure 3.13: Exemple d'interblocage en ville

3.6.1 Interblocage en routage *store-and-forward*

L'interblocage en routage *store-and-forward* intervient lorsque par exemple, le tampon d'un routeur *I* est rempli de paquets à destination d'un routeur *J*, tandis que le tampon du routeur *J* est rempli de paquets à destination du routeur *I*. On peut étendre ce cas de figure à un cycle portant sur un nombre quelconque de routeurs. La figure 3.14 illustre une telle situation.

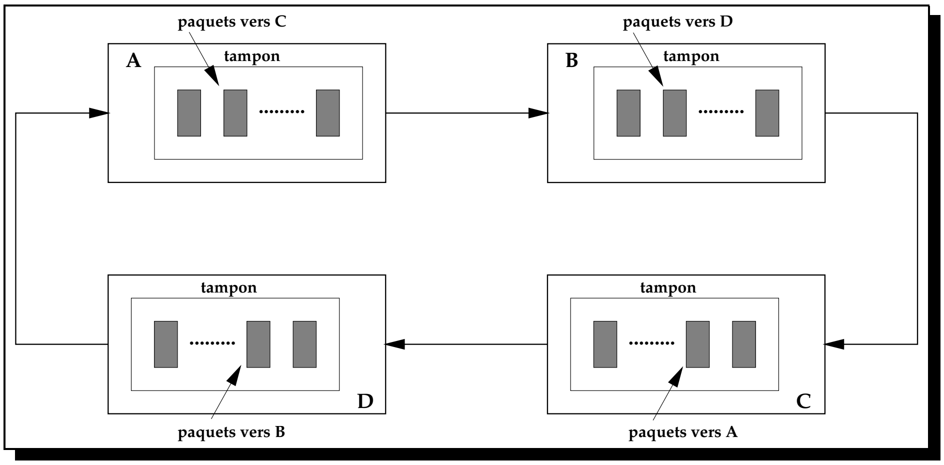


Figure 3.14: Exemple d'interblocage en routage *store-and-forward*

Les problèmes d'interblocage en routage *store-and-forward* ont été largement étudiés dans le cadre des réseaux statiques de communication utilisant la commutation de paquets [20, 21, 22, 23, 12, 24, 25, 26, 27]. Ces études montre qu'il n'existe pas de solution globale adaptée à n'importe quelle topologie de réseau et à n'importe quel algorithme de routage. Toutes ces études traitent donc d'algorithmes de routage utilisés dans des réseaux de topologies bien précises (grilles, anneaux, hypercubes etc...). Cependant un résultat important a été mis en évidence par Gunther[25], qui a montré pour le routage *store-and-forward*, que si le graphe de dépendance des tampons correspondant à l'algorithme de routage sur un réseau statique de topologie donnée, ne contient pas de cycles, alors l'algorithme de routage n'est pas bloquant sur cette topologie.

Le graphe de dépendances des tampons est un graphe orienté dont les sommets représentent le ou les tampons de chaque nœud du réseau. Deux sommets i et j appartenant à deux nœuds différents sont reliés par un arc orienté si un message peut être routé d'un tampon représenté par un sommet i vers un sommet représenté par un sommet j suivant l'algorithme de routage utilisé.

Des solutions au cas par cas ont été proposées pour résoudre les problèmes d'interblocages. Leur principe repose sur le concept de tampons structurés divisés en classes distinctes. Une solution proposée par Raubold et Haenle[20], discutée par Giessler[21], puis par Gerla et Kleinrock[12] s'applique aux topologies régulières pour des algorithmes de routage non adaptatifs optimaux. Elle consiste à partitionner les tampons du routeur en *K+1* classes où *K* est la longueur du plus long chemin dans le réseau, encore appelé diamètre du réseau si le routage se fait toujours par un plus court chemin (optimalité). L'allocation d'un tampon à un message se fait ensuite selon le nombre de routeurs traversés [22, 25, 26]. Cet algorithme de routage est connu sous le nom de *hop-based buffer reservation*. Un message arrivant dans un routeur après en avoir traversé H est stocké dans le tampon de classe H. Afin de diminuer le nombre de classes nécessaires, Gopal[27] a introduit un nouvel algorithme basé sur le partitionnement des nœuds du réseau en sous-ensembles de telle sorte que deux nœuds adjacents appartiennent à des sous-ensembles distincts. Gopal a montré que le nombre de classes peut se réduire à: $\frac{(M-1)}{M} \times H + 1$ où M est le nombre de labels différents utilisés pour étiqueter chaque sous-ensemble de nœuds et H le diamètre du réseau. Si M est petit, une économie conséquente peut être faite. Des réductions supplémentaires sont possibles au prix d'une perte de liberté sur les chemins que peuvent suivre les messages. Par exemple, sur une grille bidimensionnelle où l'on applique un algorithme de routage ligne-colonne, les interblocages peuvent être évités en utilisant 4 classes distinctes de tampons (une pour chaque direction) et ce quelle que soit la taille du réseau.

3.6.2 Interblocage en routage *wormhole*

Par sa nature même, la technique de routage *wormhole* est particulièrement sensible au phénomène d'interblocage. En effet, un message peut occuper un ensemble de canaux (le terme canal désignant un lien et son tampon de *flits* associé). Un exemple d'interblocage décrit sur la figure 3.15 met en jeu quatre paquets voulant accéder à un tampon détenu par un autre. Aucun paquet ne peut avancer et tous sont bloqués.

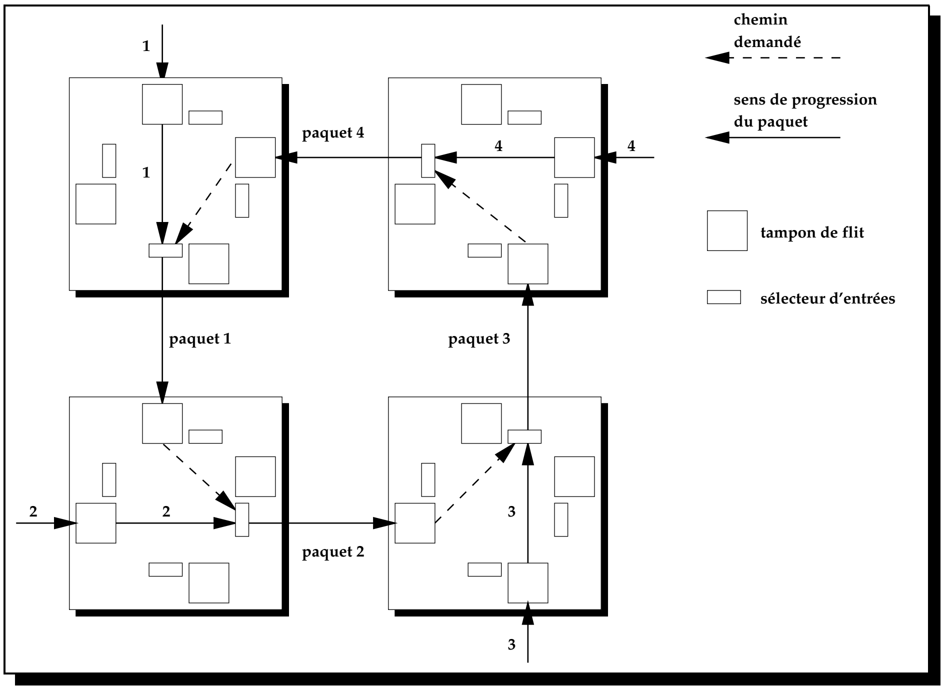


Figure 3.15: Exemple d'interblocage en routage *wormhole*

En routage *store-and-forward*, un paquet est entièrement stocké dans un tampon et n'a pas la mémoire du port d'entrée par lequel il est arrivé dans le routeur. En définissant N comme l'ensemble des nœuds du réseau et C l'ensemble des liens de communications, on peut dès lors mettre en œuvre des fonctions de routage du type $N \times N \rightarrow C$ qui ne tiennent compte que du routeur émetteur et du routeur récepteur et non du canal d'arrivée dans le routeur récepteur. En routage *wormhole*, le problème est différent. Un paquet bloqué occupe un ensemble de canaux. Pour tenir compte de ce fait, Dally et Seitz[28] définissent leur fonction de routage comme une fonction de $N \times C \rightarrow C$. La fonction de routage dépend du routeur émetteur et du canal qu'occupe le paquet. Cela leur permet d'introduire la notion de graphe de dépendance des canaux et non plus des tampons. Ce graphe de dépendances des canaux pour une topologie donnée et une fonction de routage R, est un graphe orienté dont les sommets sont les canaux et dont les arcs orientés connectent un paire de canaux pouvant être reliés par la fonction R. Ils ont ensuite énoncé une condition suffisante pour assurer qu'un algorithme de routage soit non bloquant. Cette condition est que le graphe de dépendance des canaux soit acyclique.

En introduisant le concept de canaux virtuels, ils ont proposé une nouvelle méthode permettant de construire des algorithmes non bloquants pour n'importe quel type de topologie de réseau. En effet, s'il n'est pas possible de faire en sorte que le graphe de dépendances des canaux soit acyclique sans détruire une connexion entre deux nœuds, il suffit de multiplexer plusieurs canaux virtuels ayant chacun leur queue propre sur un seul canal physique. Ceci a pour effet de multiplier le nombre de connexions du graphe de dépendances des canaux entre deux nœuds. De cette façon, il est possible de définir des contraintes sur l'allocation des canaux virtuels de façon à rendre le graphe de dépendances acyclique. L'exemple de la figure 3.16 illustre cette méthode.

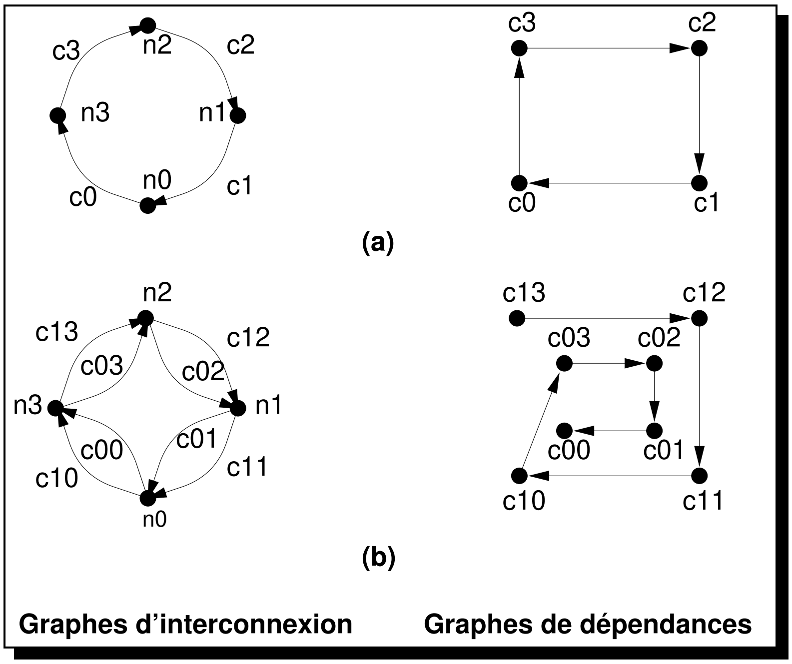


Figure 3.16: Graphe de dépendances des canaux

Le réseau considéré est ici un anneau unidirectionnel composé de 4 nœuds $\mathrm{Ni}_{i=0..3}$ et 4 canaux $\mathrm{Ci}_{i=0..3}$. Une telle topologie est susceptible de conduire à un interblocage si les nœuds ne contiennent qu'un tampon de *flit*. Le graphe de dépendance des canaux correspondant à cette structure est présenté sur la figure 3.16(a). En divisant chaque canal de communication en deux canaux virtuels, chacun disposant de son propre tampon et en adoptant la fonction de routage R décrite ci-après, le graphe de dépendance des canaux est sans cycle (figure 3.16(b)).

Fonction de routage R: Les messages issus d'un nœud Ni et à destination d'un nœud Nj tel que i<j sont routés sur les canaux $C1, k_{0..3}$, tandis que les messages issus d'un nœud Ni et à destination d'un nœud Nj tel que i>j sont routés sur les canaux $C0, k_{0..3}$. Le canal C00 et C13 sont inutilisés.

Grâce à cette méthode, Dally et Seitz ont proposé des algorithmes de routage sans risque d'interblocages dans les topologies de type *k-ary n-cubes* et *cube-connected cycles*.

3.6.3 Interblocage dans le routage par intervalle

Les méthodes proposées ci-dessus basées sur la division des tampons en classes ou sur le multiplexage d'un canal physique en plusieurs canaux virtuels ont l'inconvénient de nécessiter des ressources matérielles qui dépendent, entre autres, des topologies et des protocoles de communication. Le schéma de routage par intervalle permet de s'affranchir de ces contraintes matérielles et d'implanter des algorithmes de routage optimaux et non bloquants pour de nombreuses topologies régulières, telles que les grilles, les hypercubes, les arbres et différents réseaux multi-étages. On entend par algorithme optimal, un algorithme qui permet de choisir le chemin le plus court entre deux nœuds du réseau. Zerrouk[11] a montré que ce schéma de routage permettait de trouver des algorithmes non bloquants mais pas forcément optimaux pour n'importe quelle topologie de réseaux.

3.6.4 En résumé

Le risque d'interblocage existe, quelle que soit la technique de routage utilisée, si le graphe de dépendance de l'algorithme de routage contient un cycle. L'effet le plus dramatique est que l'interblocage entre paquets peut s'étendre et conduire au gel du trafic tout entier dans le réseau. Gunther, Dally et Seitz ont montré que l'absence de cycle dans le graphe de dépendance de l'algorithme de routage est une condition suffisante pour affirmer que l'algorithme de routage appliqué à la topologie du réseau est non bloquant. À partir de cette condition, des algorithmes de routage non bloquants ont été développés pour certaines topologies de réseaux au prix d'un surcoût matériel. Plus récemment, des algorithmes non bloquants ne nécessitant pas d'ajout matériel basés sur le schéma de routage par intervalle ont été découverts.

3.7 Conclusion

Dans ce chapitre, nous avons présenté les trois composants de base de l'architecture d'un routeur (mémoire, crossbar et calculateur de route) et le rôle qu'ils jouent. Nous avons ensuite rappelé les cinq notions fondamentales que sont:

- les techniques de routage,
- les schémas de routage,
- le contrôle de flux,
- les stratégies de mémorisation,
- les problèmes d'interblocages.

Pour chacune d'elles, nous avons expliqué en quoi elles déterminaient les caractéristiques d'un routeur et présenté plusieurs exemples de mise en œuvre, en insistant sur leur complexité matérielle au regard des possibilités et des performances qu'elles offrent. Dans le chapitre suivant, nous ferons une étude critique des solutions adoptées par trois réalisations industrielles.

CHAPITRE 4

Analyse de trois routeurs industriels

4.1 Les architectures sélectionnées

Nous avons sélectionné trois architectures qui présentent des caractéristiques proches de celles que nous visons. Ces architectures sont donc monolithiques (intégrées sur une seule puce), extensibles, à faible latence, et adaptatives (il existe plusieurs chemins permettant de relier deux nœuds du réseau). Elles supportent, enfin, des topologies quelconques et offrent un certain degré de tolérance aux fautes.

Ces routeurs sont le VULCAN [29, 30, 31] d'IBM, le TIC [32, 33] de SUN MICROSYSTEM et le C104 [34] de SGS-THOMSON.

4.2 VULCAN

4.2.1 Introduction

Le routeur VULCAN constitue la brique de base du réseau d'interconnexion des systèmes SP2 de la société IBM connectant de 4 à 512 processeurs Risc System/6000. Le but de l'architecture SP2 est de réaliser une machine massivement parallèle d'une puissance de calcul d'un tera-flop et disposant d'un réseau d'interconnexion capable de supporter un trafic d'un tera-octet par seconde. Le routeur VULCAN utilise un schéma de routage explicite et n'offre pas de fonction d'adaptativité dynamique. L'émetteur peut cependant choisir parmi 4 chemins possibles pour rallier un destinataire mais le paquet, un fois émis, ne peut changer de route afin de tenir compte d'un éventuel engorgement ou d'une panne. La technique de routage utilisée est le routage *wormhole*. Le routeur VULCAN fonctionne à une fréquence de 40 MHz et offre une latence sans contention de 5 cycles (125 ns). Le réseau fonctionne de façon synchrone.

4.2.2 Format des paquets

Les messages échangés par les processeurs d'un système SP2 sont divisés en paquets de taille n'excédant pas 255 octets (figure 4.1). Le premier octet du paquet indique sa longueur, les N suivants constituent le champ "route" et contiennent les informations de routage, viennent enfin les octets de données.

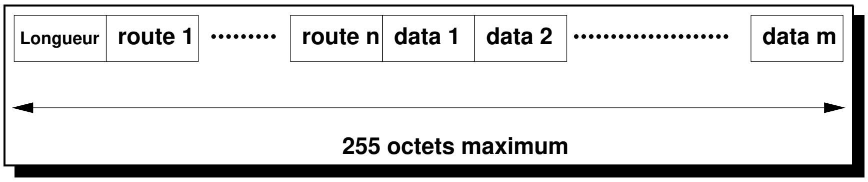


Figure 4.1: Format des paquets routés par le routeur VULCAN

4.2.3 Synoptique du routeur VULCAN

Le routeur VULCAN dispose de 8 ports parallèles bidirectionnels (Cf. figure 4.2). Chaque port se compose d'un bus d'entrée et d'un bus de sortie, tous deux d'une largeur de 10 bits. Un crossbar complet 8 x 8 dont la largeur des chemins de données est 8 bits, permet de connecter n'importe quelle entrée à n'importe quelle sortie. Une mémoire centrale d'une taille de 1 Koctet permet de stocker tout ou partie des paquets arrivant sur un port d'entrée et n'obtenant pas de sortie.

4.2.4 Description d'un port parallèle

Chaque port parallèle se compose d'une partie émission et d'une partie réception. La connexion entre deux routeurs VULCAN est décrite sur la figure 4.3. L'échange des données se fait sur 8 bits de large dans chaque direction. Un signal appelé TOKEN est utilisé pour gérer le contrôle de flux entre routeurs. Un dernier signal appelé TAG permet de différencier les octets de données des deux octets de contrôle utilisés (NULL et FAULT).

Récepteur

Le récepteur assure le contrôle de flux en envoyant un signal TOKEN à l'émetteur auquel il est connecté, chaque fois qu'il est en mesure de recevoir un octet. Le récepteur dispose en entrée d'une file d'attente de 31 octets. A l'initialisation, le récepteur envoie 31 TOKEN à l'émetteur pour l'informer qu'il peut recevoir 31 octets. Cette file d'attente permet de maintenir un flux constant entre deux routeurs, tant que la distance qui les

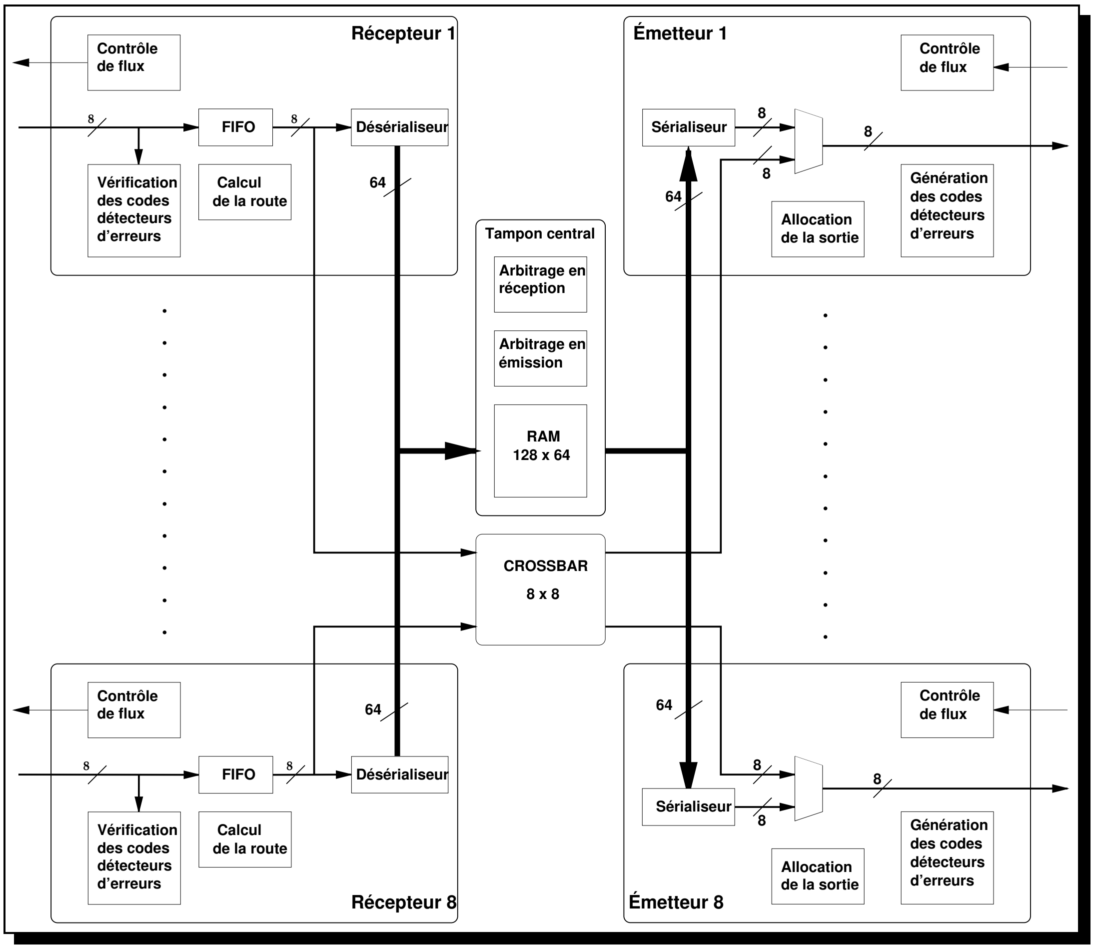


Figure 4.2: Synoptique du routeur VULCAN

sépare est inférieure à la distance parcourue sur le lien par un octet pendant 15 cycles d'horloge.

Le récepteur détermine la sortie que doit prendre le paquet, effectue la vérification des codes détecteurs d'erreur et construit des mots de 8 octets lorsque le paquet doit être stocké dans le tampon central.

Émetteur

L'émetteur transmet prioritairement les octets qui sont stockés dans le tampon central. Les mots lus dans le tampon central font 64 bits et nécessitent une sérialisation en 8 octets. Cette opération est effectuée par l'émetteur qui stocke les octets dans une FIFO de 7 mots de profondeur avant de les émettre, selon le crédit d'émission dont il dispose. Si aucun octet présent dans le tampon central ne lui est destiné, il route les octets issus d'un port d'entrée qui en a fait la demande.

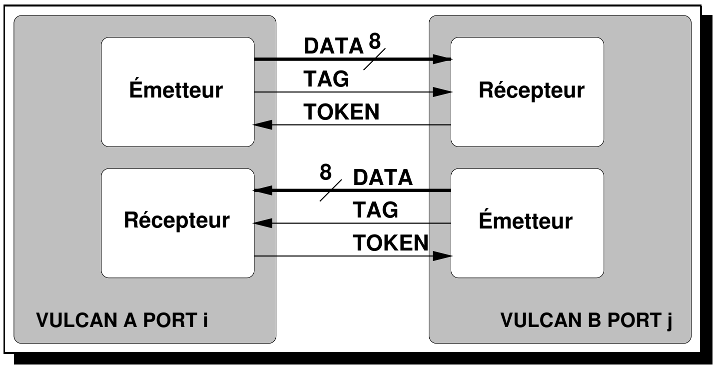


Figure 4.3: Connexion entre deux routeurs VULCAN

4.2.5 Technique de routage

La technique de routage utilisé par le routeur VULCAN s'apparente au routage *wormhole*. Elle se rapproche de la technique *virtual-cut-through* dans la mesure où un paquet bloqué en entrée d'un port (parce que la sortie qu'il demande est indisponible) est dirigé vers le tampon central. Ceci permet dans la plupart des cas de libérer le port d'entrée et de permettre à un autre paquet d'être routé. Cependant, il n'est pas nécessaire d'avoir la place suffisante pour stocker entièrement un paquet pour autoriser son aiguillage vers le tampon. En conséquence, si le tampon ne dispose pas de suffisamment de place pour stocker entièrement le paquet, une partie de celui-ci demeure dans la FIFO d'entrée et bloque le port d'entrée tout comme en *wormhole*.

4.2.6 Choix de la sortie

Le choix de la sortie se fait grâce au second octet du paquet. Si le bit 7 est à 0, les bits 6,5,4 indiquent le numéro de la sortie à prendre. Si le bit 7 est à 1, ce sont les bits 2,1,0 qui codent le numéro de la sortie. Si le paquet arrive sur un des ports du premier routeur rencontré après son émission par un processeur, le bit 7 est à 0, les bits 6,5,4 sont utilisés. Avant d'envoyer le paquet vers la sortie ou vers le tampon, le récepteur modifie le bit 7 et le force à 1. Lorsque ce paquet arrive sur le port d'entrée d'un second routeur, le récepteur utilise les bit 2,1,0 pour faire sa requête, détruit l'octet et met à jour la valeur de l'octet indiquant la taille du paquet. La route est donc indiquée par l'octet suivant. Lorsque le paquet arrive à destination, le paquet ne contient plus que des données et leur nombre d'octets.

4.2.7 Le tampon central

Le tampon central est une mémoire RAM de 128 mots de 64 bits autorisant une écriture et une lecture par cycle. Chaque port de sortie a un minimum de place réservée (1 mot de 64 bits), mais le reste de l'espace mémoire est alloué de façon dynamique selon les besoins des ports d'entrée. Lorsque deux ports d'entrée demandent l'accès au tampon central simultanément, c'est celui qui a été le moins récemment servi qui est choisi. Ce tampon est similaire au tampon CBDA dont nous avons parlé au chapitre précédent.

Afin de faire en sorte que le débit entrant dans chaque port soit maintenu à 1 octet par cycle, même dans le pire des cas où les huit ports d'entrée demandent l'accès au tampon central, le bus d'écriture du tampon est d'une largeur de huit octets. Ainsi le port qui obtient l'accès écrit un mot de 8 octets et commence à construire un nouveau mot avec les octets qu'il reçoit. La construction du mot prend 8 cycles ce qui permet aux 7 autres ports d'écrire séquentiellement dans le tampon avec un débit entrant d'un octet par cycle par port.

En lecture le principe est le même. Chaque port de sortie peut lire 8 octets tous les 8 cycles, et maintient donc un débit sortant d'un octet par cycle par port.

4.2.8 Allocation des sorties

Le routeur VULCAN dispose d'un crossbar complet 8 x 8 qui permet l'envoi d'un octet de n'importe quelle entrée vers n'importe quelle sortie. Si un port d'entrée obtient la sortie demandée avant que les huit premiers octets du paquet n'aient été reçus, le paquet est routé à travers le crossbar directement vers la sortie. Si la requête n'est pas satisfaite avant que les huit octets aient été reçus, le paquet est dirigé vers le tampon central sous forme de mots de huit octets. Si plusieurs entrées demandent simultanément la même sortie, cette dernière est allouée à l'entrée qui a été la moins récemment servie. Le tampon central est prioritaire sur les entrées.

4.2.9 Configuration du réseau

Plutôt que de disposer d'un réseau de contrôle physiquement séparé et fonctionnant en parallèle avec le réseau de transfert des données, le routeur VULCAN fonctionne selon trois modes principaux : un mode de transfert normal des données (*Run Mode*), un mode de service (*Service Mode*) et un mode de gestion des erreurs (*Fault Mode*). Le passage du mode de service au mode normal se fait par l'intermédiaire de deux séquences de transition *Run-down* et *Run-up*. La séquence *Run-down* dure 16 cycles et a pour but de vider les liaisons des octets en cours de transmission et le mode *Run-up* réinjecte ces octets sauvegardés pour retrouver l'état du réseau précédant le passage au mode *Run-down*. Ceci oblige tout comme la profondeur de la FIFO d'entrée, à utiliser des liens dont la longueur n'entraîne pas un délai de transmission excédant 15 cycles.

Afin de configurer la table de routage contenue dans la mémoire de chaque nœud, le réseau démarre en mode de service dans lequel le logiciel de gestion du réseau (WORM) essaye d'en retrouver la topologie en s'appuyant sur un fichier décrivant celle qui est attendue. Ce logiciel de gestion est implanté indifféremment sur n'importe quel nœud du réseau qui est alors désigné comme le nœud de service. Si des différences existent entre la topologie réelle et celle attendue, celles-ci sont enregistrées, et le générateur de table de routage en tient compte lors de la détermination de leur contenu. Le générateur détermine pour chaque paire de nœuds, 4 chemins permettant de les relier. Cette phase d'initialisation intervient après chaque détection de panne de nœuds ou de liaisons.

4.2.10 Gestion des erreurs et tolérances aux pannes

Un réseau de routeurs VULCAN fonctionne de façon synchrone. Cela permet d'effectuer la détection d'erreurs de transmission à des intervalles de temps fixes. Le temps est ainsi divisé en périodes de 64 cycles et, durant les 2 derniers cycles de chaque période, des codes détecteurs d'erreur (calculés par les émetteurs sur les octets transmis), sont envoyés aux récepteurs auxquels ils sont connectés. Les récepteurs ayant recalculé à la volée les codes détecteurs d'erreur, ils vérifient que les octets reçus sont corrects. L'utilisateur est libre d'insérer des codes détecteurs d'erreur dans les paquets pour effectuer un contrôle d'intégrité de bout en bout au niveau paquet.

Lorsqu'une erreur est détectée, le routeur se met dans le mode spécial *Fault mode* dans lequel il informe ses voisins qu'il a détecté un problème, en leur transmettant un mot de contrôle spécifique (FAULT). Dès la réception de ce code, les autres routeurs propagent l'information jusqu'à ce que tous les composants du réseau soient avertis. Le logiciel de contrôle du réseau collecte les informations contenues dans les routeurs pour connaître le type et la localisation de l'erreur. Une fois le traitement de l'erreur effectuée, le réseau retourne dans son mode de fonctionnement normal.

Le fonctionnement synchrone offre la possibilité pour chaque routeur de disposer d'une doublure qui fonctionne en mode esclave. Les broches de sorties identiques des deux circuits sont connectées ensembles. Le maitre impose la valeur tandis que l'esclave la vérifie.

4.2.11 Synchronisation des liens parallèles

Afin de pouvoir fonctionner de façon synchrone et résoudre le problème de déphasage des bits transitant sur un port parallèle, le routeur VULCAN met en œuvre un mécanisme de synchronisation des liens appelé *tuning*. Cette synchronisation est assurée par le logiciel WORM lorsque le routeur VULCAN est en mode Service. Le principe de synchronisation est le suivant : chaque port de sortie dispose d'éléments de retard programmables; le WORM calcule le nombre de cycles pris pour qu'une impulsion fasse un aller-retour sur le lien; plusieurs comptages sont effectués pour diverses valeurs de retard croissantes. Pour une certaine valeur du retard, le nombre de cycle requis augmente d'une unité. La valeur du retard est telle que l'on est dans la zone de *set-up time / hold time* de la bascule de réception. En conséquence, la valeur correcte du retard se situe à un demi-cycle de cette valeur.

4.2.12 En résumé

Le routeur VULCAN offre une faible latence (125 ns) et un débit global de 320 Mo/s. Il n'offre pas de fonction d'adaptativité dynamique mais offre, comme solution à la congestion, la possibilité de stocker des paquets bloqués dans un tampon central d'une taille de 1 Koctet partagé par tous les ports d'entrée. La taille maximale des paquets limitée à 255 octets fait que, la plupart du temps, un paquet entier peut être stocké, libérant ainsi le canal qu'il occupe et offrant la possibilité à un autre paquet d'être routé. Cette méthode permet de retarder le seuil de saturation du réseau comme le montre les simulations effectuées dans [29].

Cependant plusieurs critiques peuvent être faites, notamment l'utilisation de ports parallèles qui restreint le nombre de liens du fait du nombre limité des broches d'un boîtier. D'autre part, la méthode de synchronisation des liaisons est complexe et coûteuse. Elle atteint ses limites à 50 MHz; d'autres méthodes doivent être mises en œuvre pour opérer à des fréquences de fonctionnement supérieures.

4.3 TIC

4.3.1 Introduction

S-CONNECT est un système d'interconnexion extensible à haute vitesse, développé par SUN MICROSYSTEMS COMPUTER CORPORATION, afin de supporter un réseau de stations de travail et de partager efficacement les ressources de calcul disponibles. Pour cela, un composant de commutation à hautes performances a été conçu : le routeur TIC. Ce prototype de routeur dispose de 4 liens série bidirectionnels full duplex en fibre optique à 1.3 Gbit/s et 2 ports parallèles de 32 bits de large (16 bits dans chaque direction) fonctionnant à 66 Mhz. Le routeur TIC utilise un schéma de routage adaptatif et supporte des topologies de réseaux quelconques. La technique de routage est le *virtual-cut-through*. Un réseau S-CONNECT nécessite un fonctionnement synchrone de tous les éléments du système. La latence de routage sans contention est varie de 100 à 200 ns par routeur.

4.3.2 Synoptique du routeur TIC

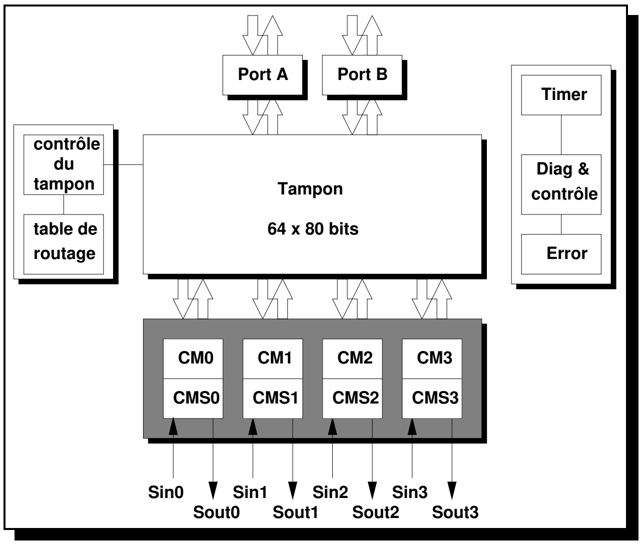


Figure 4.4: Synoptique du routeur TIC

4.3.3 Description d'un port parallèle

Les ports parallèles A et B consistent en deux bus de 16 bits de large (un dans chaque direction) et deux broches de contrôle de flux (*valid* : un paquet est prêt et *accept* : le paquet a été lu). Ces ports, cadencés à 66 MHz, ne peuvent connecter que des composants proches du fait de leur fréquence de fonctionnement. Ils sont donc principalement utilisés pour relier un routeur TIC à un nœud ou à un autre routeur TIC de façon à réaliser un routeur de valence supérieure. Plusieurs configurations possibles sont visibles sur la figure 4.5.

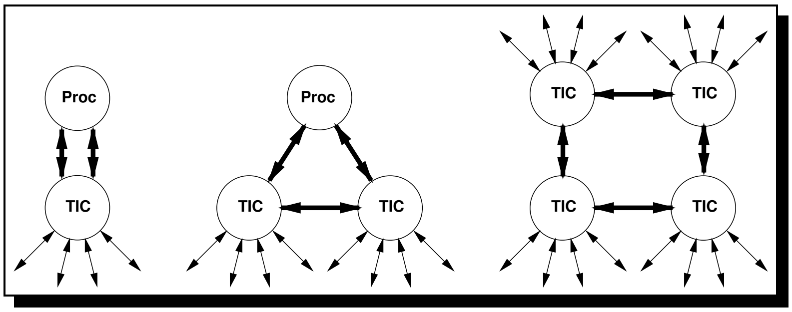


Figure 4.5: Utilisation des ports parallèles

4.3.4 Description d'un port série

Un port série est constitué de deux composants : le *channel module* CM et le système de maintenance du canal CMS (*channel maintenance system*). La partie réception du CM accepte les paquets, vérifie leur intégrité et les écrit dans le tampon central. La partie émission du CM gère l'extraction des paquets du tampon et leur émission. Le CMS contient un sérialiseur qui convertit des mots de 16 bits en un flot série de bits après avoir effectué un codage 8B/10B. Il contient également le désérialiseur qui effectue l'opération inverse.

4.3.5 Format des paquets

Le routeur TIC véhicule des informations de type contrôle et de type donnée (des lignes de cache de 32 octets). Les paquets routés par le routeur TIC ont une taille fixe de 80 bits. Les paquets simples possèdent un en-tête sur 32 bits qui se décompose en :

- 12 bits codant la destination du paquet,
- 2 bits codant le type du paquet,
- 2 bits codant la priorité du paquet,
- 12 bits codant la source du paquet,
- 4 bits codant l'âge du paquet.

Pour transmettre une ligne de cache de 32 octets, le routeur TIC utilise 4 paquets constituant un *quad-packet*. Chaque octet est couvert par un bit de parité. La partie données s'étend donc sur 288 bits (32 $\times$ 9 bits). Seul le premier paquet contient les 32

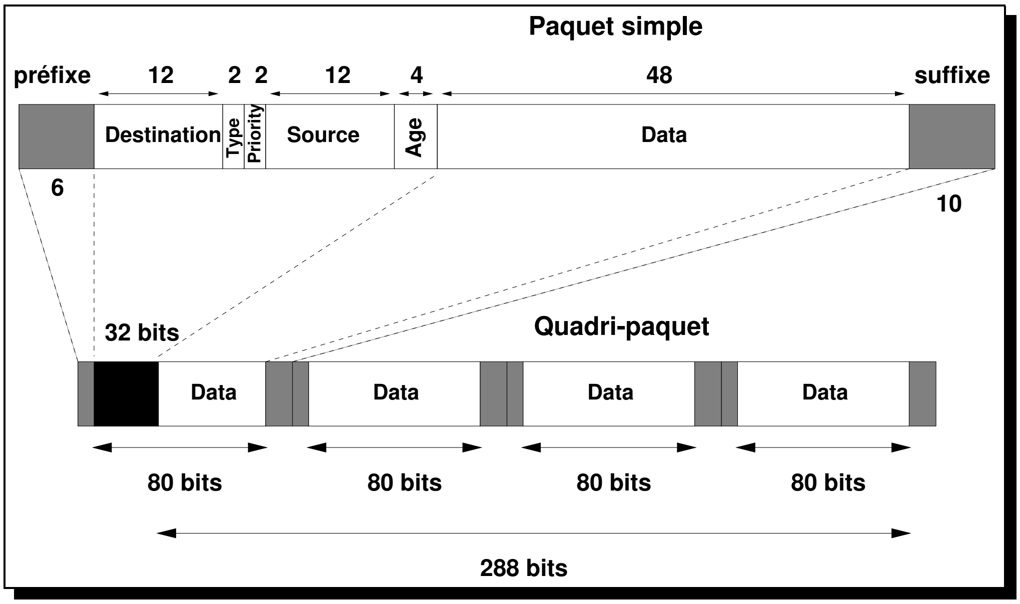


Figure 4.6: Format des paquets routés par le routeur TIC

bits d'en-tête, suivis de 48 bits de données. Les 3 derniers paquets ne contiennent que des données soit 240 bits (3×80).

Le routeur TIC utilise un schéma de routage adaptatif. Il ne préserve donc pas l'ordre des paquets. Cependant, il reconnaît les *quad-packets* grâce au type et les traite comme une seule entité indivisible.

Un préfixe et un suffixe forment un mot de 16 bits qui est ajouté aux paquets par les CMs à l'émission et détruit à la réception. Ils n'apparaissent pas sur les ports parallèles. Le suffixe correspond au paquet qui le précède et contient le CRC plus un bit indiquant la validité du paquet. Le préfixe contient le numéro du canal virtuel d'où est issu le paquet entrant et l'accusé de réception correspondant au dernier paquet ayant transité dans le sens opposé.

4.3.6 Choix de la sortie

Chaque routeur TIC dispose d'une table complète qui définit pour chaque nœud du système, une ou plusieurs sorties possibles permettant de l'atteindre. Il supporte ainsi des topologies quelconques mais de taille limitée. Le nombre de nœuds supportés, ainsi que le nombre de sorties différentes que peut emprunter un paquet pour rallier sa destination, ne sont pas indiqués dans les documents en notre possession. Jusqu'à présent, la taille maximale des systèmes réalisés est de 16 nœuds.

4.3.7 Structure logique du TIC

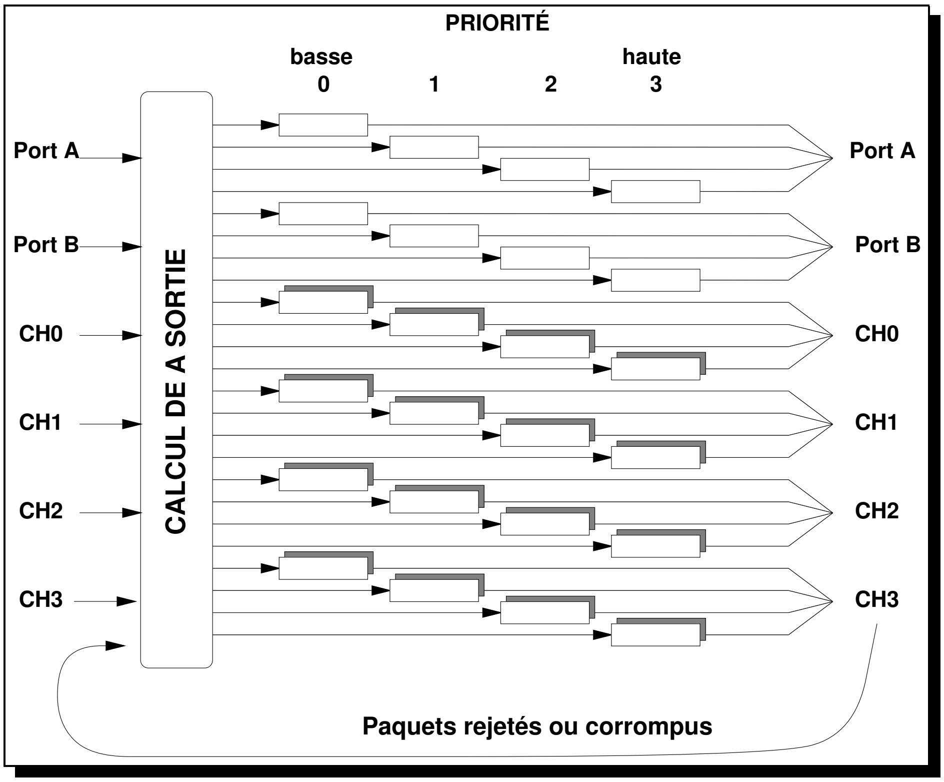


Figure 4.7: Structure logique de TIC

La structure logique du routeur TIC (Cf. figure 4.7) s'articule autour d'un tampon central partagé par les 6 ports. Il est constitué de 64 tampons de 80 bits, gérés par le contrôleur du tampon qui maintient un système de listes chaînées. À chaque port série en sortie sont associés 2 canaux virtuels pour chaque degré de priorité. Les ports parallèles en sortie n'ont pas de canaux virtuels. Ceci représente donc un total de 40 canaux. Afin d'assurer que les paquets de plus haute priorité ne seront pas bloqués par des paquets de priorité inférieure, un tampon est dédié à chaque canal virtuel pour chaque niveau de priorité. Les 24 tampons restants sont alloués selon les besoins.

4.3.8 Le tampon central

Le tampon est structuré en 5 bancs mémoire (M0 à M4) d'une largeur de 16 bits (soit 5 $\times$ 16 = 80 bits : longueur d'un paquet) disposant d'un port d'écriture et deux ports de lecture R0 et R1. Le port de lecture R0 de chaque banc est partagé par les ports B, CH0 et CH1, le port R1 est partagé par les ports A, CH2 et CH3. À chaque cycle, un paquet entier peut être écrit dans le tampon. Ceci conduit à un débit entrant de 528 Mo/s.

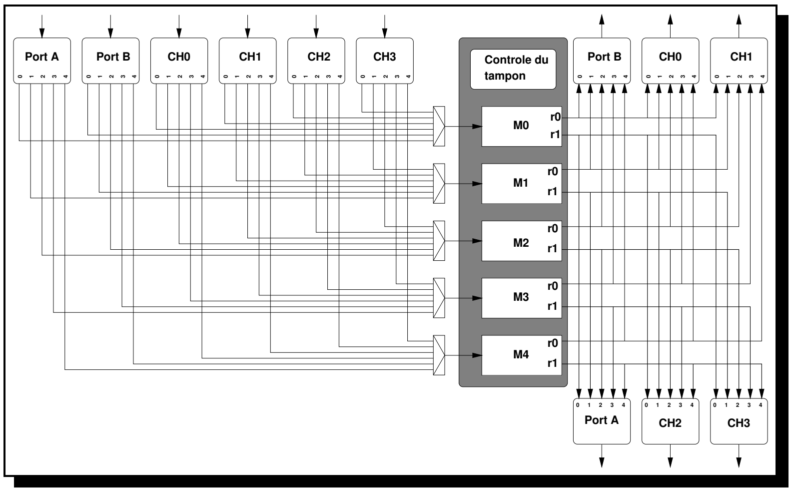


Figure 4.8: Structure du tampon central

4.3.9 Routage d'un paquet

Le routeur TIC dispose d'une seule table de routage et d'une unique logique de routage pipe-linée en réception. À chaque cycle, un paquet peut arriver sur n'importe quel port d'entrée. Le routeur TIC scrute séquentiellement les CMs pour savoir si un paquet est en cours de réception sur l'un d'eux. Si c'est le cas, le premier mot de 16 bits contenant la destination et la priorité du paquet est émis par le CM et écrit dans le banc M0. Cela marque le début d'un cycle de routage qui durera 6 cycles. Les mots suivants du paquet seront écrits par le CM dans les bancs M1 à M4. Ceci permet de commencer un cycle de routage à chaque cycle pour chacun des ports d'entrée. La phase de routage se déroule comme suit :

1. Concurremment avec son écriture dans le banc M0, le premier mot est envoyé vers la logique de routage.
2. La table de routage est interrogée pour connaître la ou les sorties possibles. Le contrôle du tampon accepte ce paquet s'il y a de la place disponible dans le tampon pour la priorité et le canal virtuel spécifié.
3. La sortie est déterminée d'après la réponse de la table de routage et le nombre de paquets en attente à chacune des sorties possibles.
4. Le nombre des paquets en attente de la sortie choisie est incrémenté. Les pointeurs de tête et de queue sont lus.
5. le pointeur de queue est positionné sur le nouveau paquet. Dans le cas où le paquet est à destination d'une queue vide, les pointeurs de tête et de queue sont modifiés. Dans le cas d'un paquet rejeté faute de place dans le routeur récepteur par exemple, le pointeur de tête est positionné sur l'adresse de ce paquet afin de le réémettre prioritairement.

De même il n'existe qu'une seule logique de transmission elle-aussi pipe-linée. Contrairement à la logique de réception qui ne traite qu'un paquet à la fois, la logique d'émission lit deux paquets simultanément grâce à la présence des deux ports de lecture R0 et R1. Ceci permet de diminuer la latence de routage.

Chaque émetteur sert ses files d'attente dans l'ordre strict de leur priorité. Tant que la file d'attente de haute priorité contient des données les autres files ne seront pas servies. Les canaux virtuels sont servis alternativement et équitablement. Évidemment si un canal virtuel est vide, la capacité total du lien physique est dédiée au second. Cette phase se déroule selon les étapes suivantes :

1. le port de sortie détermine le paquet en attente de transmission qui a la priorité la plus haute et l'âge le plus avancé. Parmi les paquets ex-aequo, un dernier algorithme de choix est effectué mais sa nature n'est pas indiquée dans les documents à notre disposition.
2. les premiers 16 bits du paquet sont lus simultanément avec l'adresse du paquet suivant dans la queue.
3. Le pointeur de tête est positionné à la valeur de l'adresse du paquet suivant.

Dans l'hypothèse où les sorties spécifiées dans la table de routage sont occupées et que le tampon est plein, le paquet est routé aléatoirement vers une sortie libre.

4.3.10 Contrôle de flux et gestion des erreurs

Lorsqu'un paquet est transmis, il est conservé dans le tampon jusqu'au retour de l'accusé de réception indiquant si le paquet a été correctement reçu ou non. Si le paquet a été transmis sans erreur le tampon est libéré et disponible pour un autre paquet. Dans le cas où le paquet a subi une altération ou a été rejeté faute de place dans le routeur récepteur, le paquet est envoyé dans une file d'attente spéciale, son âge est incrémenté et il subit à nouveau un cycle de routage. À l'issu de ce cycle, il est placé en tête de la file d'attente correspondant à la sortie choisie. Dans le cas où l'accusé de réception est corrompu, le paquet est retransmis. Dans ce cas le paquet n'est pas dupliqué et est détruit à la réception si la transmission s'est bien déroulée.

Le routeur TIC utilise un routage *virtual-cut-through*. Un paquet peut être routé avant sa complète réception. Le contrôle d'intégrité du paquet est effectué par le CM récepteur. En conséquence, la validité du paquet n'est pas connue avant le routage. S'il apparait que le paquet a subi une altération un bit *Abort* est positionné dans le suffixe du paquet sortant permettant éventuellement au routeur récepteur de détruire ce paquet.

4.3.11 Mécanisme de synchronisation entre nœuds

S-CONNECT fonctionne de façon synchrone, permettant ainsi aux pipe-lines de tous les routeurs TIC de fonctionner en phase. La synchronisation des routeurs repose sur un mécanisme de boucle à verrouillage de phase. La transmission série des données transporte aussi l'horloge de l'émetteur. Le routeur TIC effectue une moyenne des fréquences reçues sur ses ports d'entrée et ajuste son oscillateur local en conséquence. L'horloge locale est ensuite transmise aux autres routeurs lors de l'envoi des données. Ce mécanisme permet au système d'atteindre un état de synchronisation globale. Un autre mécanisme est implanté dans les CMSs pour ajuster la phase des données émises avec celle du récepteur.

4.3.12 Configuration

Un nœud dédié a pour rôle de découvrir la topologie du réseau et de calculer le contenu des tables de routage de chaque routeur, qui sera chargé à l'initialisation du système. Le routeur TIC peut recevoir sur n'importe lequel de ses canaux des messages de contrôle destinés à le configurer. Ces fonctions de contrôle incluent :

- la lecture et l'écriture des tables de routage.
- le routage explicite de paquets vers un canal spécifique. Ceci permet en particulier de véhiculer des messages destinés à reconnaître la topologie du réseau alors que les tables de routage ne sont pas encore initialisées.
- la lecture des registres contenant les codes d'erreur qui ont été détectées ainsi que l'accès aux registres d'état. Des canaux peuvent ainsi être explicitement activés ou désactivés.
- la mesure du taux d'occupation d'un canal.

4.3.13 En résumé

Le routeur TIC offre une faible latence de routage de l'ordre de 170 ns soit 11 cycles pour une fréquence de 66 MHz (dont 5 dûs à la sérialisation / désérialisation). Afin de réduire les phénomènes de congestion, le routeur TIC offre la fonction d'adaptativité et utilise en dernier recours un schéma de routage universel où le paquet est routé aléatoirement vers une sortie libre. Le routeur TIC est de loin celui qui offre le débit le plus important mais son architecture, très fortement influencée par le choix de ne router que des paquets de 80 bits a pour conséquence de réduire fortement son champ d'utilisation.

4.4 C104

4.4.1 Introduction

Développé par SGS-THOMSON, le C104 est un routeur de paquets disposant de 32 ports série bidirectionnels asynchrones à débit programmable (10 à 100Mbit/s par port et par direction), interconnectés par un crossbar non bloquant 32 x 32. Les ports transmettent les données de façon concurrentes et le transfert d'un paquet entre un port d'entrée et un port de sortie n'affecte pas le transfert d'un paquet entre un autre couple de ports d'entrée et de sortie. La technique de routage utilisée est le routage *wormhole*. La latence de routage sans contention est inférieure à une microseconde pour une fréquence de 50 MHz.

4.4.2 Data/Strobe link

Le protocole de communication des *DS-links* (Data/Strobe link) utilise deux câbles dans chaque direction, l'un transportant les données, l'autre un signal strobe. Chaque *DS-link* véhicule des caractères de données et de contrôle. La figure 4.9 décrit le format de ces caractères. Les caractères de données sont codés sur 10 bits et consistent en 1 bit de parité, 1 bit de type (0 : donnée, 1 : contrôle) et 8 bits de données. Les caractères de contrôle sont codés sur 4 bits et consistent en 1 bit de parité, 1 bit de type et 2 bits indiquant le type du caractère de contrôle.

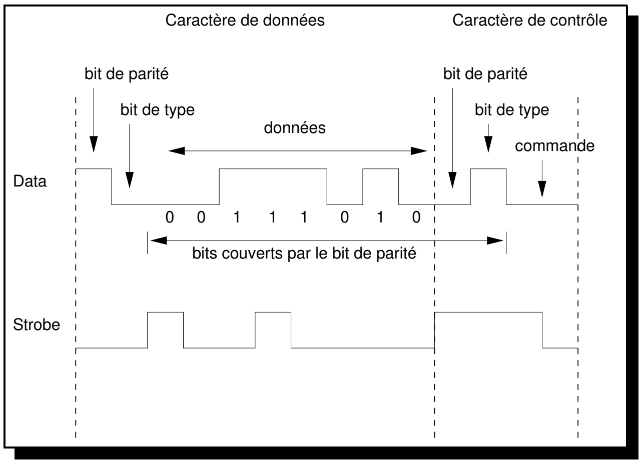


Figure 4.9: Format des caractères de données et de contrôle des *DS-links*

Le protocole *DS-link* assure qu'un seul des deux signaux de la paire Data/Strobe a un front pour chaque bit. Le niveau sur le fil Data donne la valeur du bit transmis. Le signal Strobe varie à chaque fois que le signal Data reste stable. De cette façon, l'horloge d'émission est transmise au récepteur. Les transmissions peuvent donc être asynchrones et l'horloge est reconstruite par le récepteur en effectuant un OU exclusif entre le signal Strobe et le signal Data.

Les caractères de données et de contrôle sont de longueurs différentes. Pour cette raison le bit de parité, dans chaque caractère, couvre la parité des bits de données (ou de contrôle) du caractère précédent ainsi que le bit de type du caractère courant. Cela permet de détecter une erreur sur le bit de type. Le bit de parité est positionné de façon à assurer que les bits couverts par ce contrôle, y compris le bit de parité, ont un nombre impair de '1'.

4.4.3 Synoptique du routeur C104

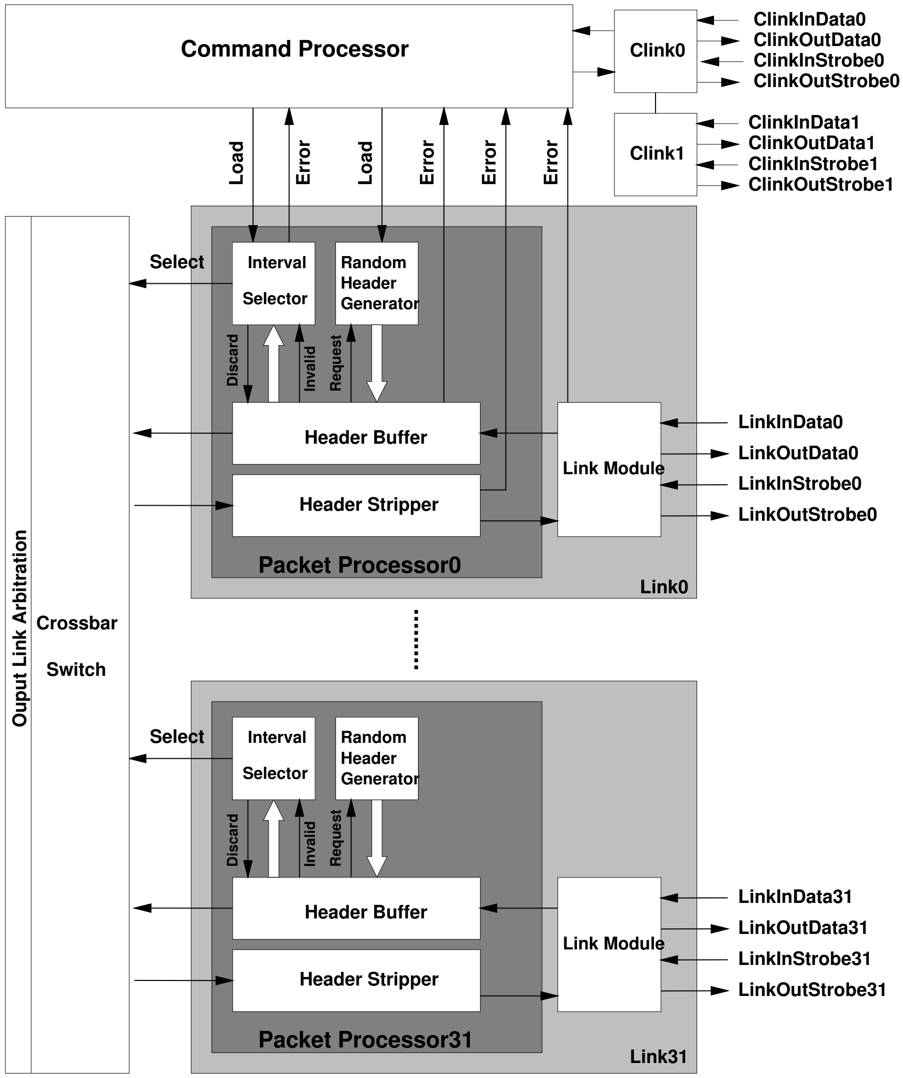


Figure 4.10: Synoptique du routeur C104

4.4.4 Format des paquets

Les messages transmis dans un système à base de routeurs C104 sont divisés en paquets (figure 4.11) contenant dans leur en-tête l'information nécessaire à leur routage. L'en-tête se compose d'un ou deux octets; chaque port d'entrée peut être programmé pour traiter un en-tête sur un ou deux octets. Le routeur utilise l'en-tête du paquet pour déterminer la sortie à emprunter. Les caractères suivant l'en-tête forment le corps du paquet. Celui-ci se termine par un caractère Fin de Paquet (EOP) ou Fin de Message (EOM). Le routeur C104 route ainsi des paquets de taille quelconque.

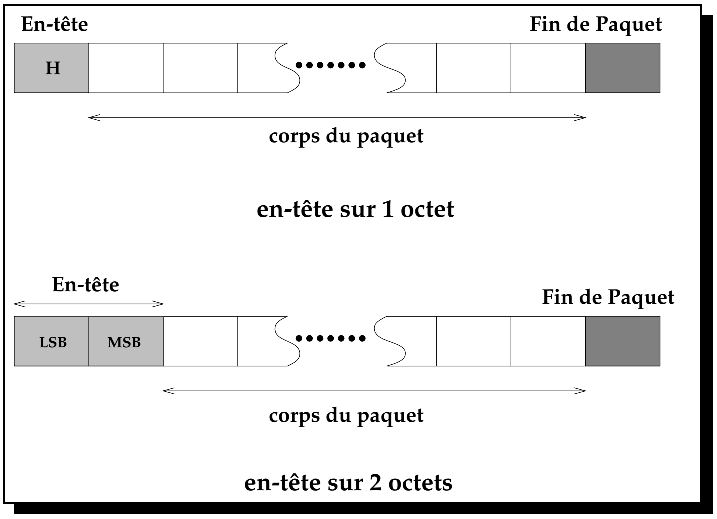


Figure 4.11: Format des paquets routés par le routeur C104

4.4.5 Schéma de routage

Le routeur C104 utilise le schéma de routage par intervalle et dispose d'une table de routage indépendante pour chaque port d'entrée.

Partitionnement

Le routeur C104 peut être programmé de sorte que les sorties choisies ne dépendent pas des ports d'entrée mais uniquement des en-têtes. Dans ce cas, toutes les tables de routage ont le même contenu. Cependant, puisque l'on dispose d'une table indépendante par entrée, on peut réaliser plusieurs réseaux logiques indépendants sur le même réseau physique, dans la mesure où un paquet circulant sur un réseau ne peut jamais se retrouver sur un autre. Cette méthode permet entre autre de router des messages de priorité différente, bien que les paquets ne contiennent aucune information de priorité.

Implémentation

Chaque port dispose d'une unité de sélection d'intervalle qui détermine la sortie à emprunter selon l'intervalle auquel appartient l'en-tête du message entrant. 36 intervalles sont définis par 36 séparateurs d'intervalles contenant des valeurs codées sur 16 bits et strictement croissantes De ce fait, les intervalles définis sont consécutifs et non recouvrants A chaque intervalle est associée une sortie pouvant être choisie parmi les 32 possibles.

N.B : le nombre d'intervalles peut être supérieur au nombre de ports : on se rapproche alors d'une table de routage complète.

4.4.6 Latence de routage sans contention

La latence de routage sans contention dans le routeur C104 dépend de deux horloges : l'horloge système à 50 MHz qui cadence le cœur du routeur et l'horloge programmable des *DS-links*.

Étape	cycle système	cycle DS link
sérialisation	1	22
routage	8	
désérialisation	4	17
TOTAL	13	39

Si les liens sont cadencés à 100 Mbit/s, la latence obtenue en pratique est proche d'une microseconde.

4.4.7 Résistance à la congestion et tolérance aux pannes

routage universel

Afin de réduire les phénomènes de "points chauds" qui peuvent survenir en cas de fort trafic, le routeur C104 permet de mettre en œuvre un algorithme de routage en deux phases appelé **routage universel**, dans lequel le paquet est d'abord envoyé vers une destination aléatoire avant d'être ensuite routé vers sa destination initiale. Cette méthode a montré en pratique une amélioration de la répartition du trafic et une réduction des phénomènes de points chauds [35].

Adaptativité

Un autre moyen utilisé pour diminuer les phénomènes de congestion et offrir un certain de degré de tolérance aux fautes est appelé *grouping*. Cette méthode consiste à grouper des ports de sortie de numéros consécutifs : si le premier port du groupe est occupé ou en panne, le paquet est routé par le premier port suivant disponible et fonctionnel.

4.4.8 Routage hiérarchique

Chaque port de sortie du routeur C104 peut être programmé pour détruire l'en-tête du paquet avant son émission. Le caractère suivant devient donc le nouvel en-tête. Cette technique appelée *Header Deletion* permet de router des paquets dans des hiérarchies de réseaux (figure 4.12) et de s'affranchir du nombre limité de destinations possibles (65536), dû à la taille fixe de l'en-tête. En outre cette méthode permet de faire du routage explicite, dans le cas où l'émetteur du paquet connaît a priori la route pour atteindre la destination.

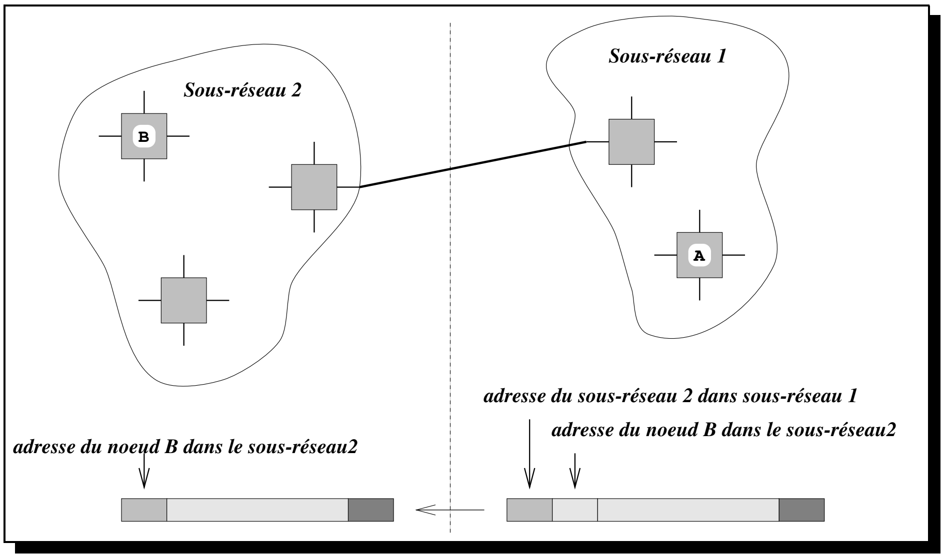


Figure 4.12: Routage d'un paquet dans une hiérarchie de réseaux

4.4.9 Contrôle de flux

Le contrôle de flux inter-routeur repose sur la technique de crédit d'émission : il est assuré par un caractère de contrôle dédié qui est inséré par le récepteur du message dans le flot des données qui transitent dans le sens inverse sur le lien. Ce caractère est reconnu par l'émetteur qui est ainsi autorisé à envoyer un nouveau *flit* de 8 caractères.

4.4.10 Description fonctionnelle

Chaque *DS-link* est connecté à un processeur de paquets qui contient un sélecteur d'intervalle, un générateur aléatoire d'en-tête et un destructeur d'en-tête (Cf. figure 4.10). Le port de sortie est déterminé d'après la valeur de l'en-tête. Si ce port est occupé, les données entrantes sont stockées dans une FIFO jusqu'à ce que faute de place le flot entrant soit bloqué par le contrôle de flux.

Trois traitements sont possibles :

- l'en-tête est comparé aux bornes d'intervalle et la sortie correspondante choisie. Le paquet est transmis sans modification;
- le port de sortie est en mode *Header Deletion*. L'en-tête du paquet est détruit et la paquet transmis;
- le port d'entrée est en mode routage universel. Le nouvel en-tête fourni par le générateur aléatoire est ajouté en début du paquet et ce dernier est transmis vers la sortie choisie d'après ce nouvel en-tête.

4.4.11 Traitement des erreurs

Le routeur C104 détecte plusieurs types d'erreur.

- les erreurs sur les *DS-links*, à savoir les erreurs de parité et de déconnexion de *DS-link*. Lorsqu'un DS-link détecte une erreur de type parité sur son entrée, il coupe l'émission sur sa sortie. Cette coupure est détectée comme une erreur de déconnexion à l'autre bout du lien. Ceci a pour effet de couper l'émission de ce lien. La détection d'une erreur par un lien provoque le lancement d'une procédure de réinitialisation des *DS-link*.
- les erreurs sur les paquets : il s'agit des paquets nuls (seul reste un caractère EOP ou EOM après *Header Deletion*) ou des paquets courts (le second octet de l'en-tête est absent et remplacé par un caractère EOP ou EOM).

La détection d'une erreur sur un paquet entraine sa destruction et l'envoi d'un message indiquant le type de l'erreur détectée au nœud de contrôle via le port **Clink0**.

4.4.12 Configuration du réseau

Le routeur C104 dispose de deux liens *DS-link* dédiés à son contrôle et à sa configuration. Ces deux liens sont **Clink0** et **Clink1**. L'accès au routeur se fait par **Clink0**, **Clink1** servant à chaîner les routeurs C104 entre eux afin de réaliser un chemin de configuration. Des messages de commandes sont envoyés aux routeurs C104 sur leur port **Clink0** afin, par exemple, de paramétrer la vitesse des *DS-links* ou de charger le contenu des tables de routage.

4.4.13 En résumé

Le routeur C104 offre un débit global est 320 Mo/s. Le schéma de routage par intervalle permet la construction de réseaux de topologies quelconques et l'implémentation d'algorithmes non bloquants pour une large classe de topologies. La technique de destruction de l'en-tête d'un message à l'émission permet de router facilement des messages dans des hiérarchies de réseaux et de faire du routage explicite.

Afin d'offrir un certaine résistance à la congestion du réseau, le routeur C104 offre la possibilité d'implémenter un algorithme de routage universel. Dans le but d'offrir un certain degré de tolérance aux pannes et de réduire les phénomènes de congestion, le routeur C104 implante une fonction d'adaptativité appelée *grouping*. Cependant cette dernière technique manque de souplesse.

D'autre part, la latence de l'ordre d'une microseconde lorsqu'il n'y a pas de contention est 10 fois plus élevée que celle que nous visons. Enfin les *DS-links* offrent un débit 10 fois inférieur à celui que nous voulons atteindre.

4.5 Récapitulatif

	C104	**VULCAN**	**TIC**
routage	wormhole	wormhole	virtual-cut-through
topologie	quelconque	quelconque	quelconque
mémorisation	en entrée	centrale	centrale
débit	320 Mo/s	320 Mo/s	528 Mo/s
crossbar	32 x 32	8 x 8	6 x 6
latence	< 1μs	125 ns	170 ns
adaptativité	oui	à la source	oui
schéma de routage	par intervalle	explicite	table complète
fréquence	50 MHz	40 MHz	66 MHz

De l'analyse de ces trois exemples de routeurs industriels, on peut tirer les remarques suivantes.

Le routeur VULCAN n'est pas capable de modifier la route d'un paquet afin de réagir à des congestions dans le réseau ou à des pannes de liaisons qui peuvent survenir pendant le trajet d'un nœud à un autre. La méthode de synchronisation entre routeurs atteint ses limites vers 50 MHz et interdit l'amélioration des performances par une augmentation de la fréquence de fonctionnement au-delà de cette valeur. Enfin l'utilisation de ports parallèles interdit d'envisager des routeurs de valence élevée. Le routeur VULCAN utilise déjà un boitier PGA à 223 broches pour une valence de 8.

Le routeur TIC offre un débit important mais sa capacité à ne router que des paquets de 80 bits le cantonne dans des applications très spécifiques. En outre, l'utilisation d'une unique table complète lui interdit de réaliser des réseaux disjoints et limite le nombre de nœuds pouvant appartenir au réseau.

Le routeur C104 est certainement celui qui présente le plus d'analogie avec le routeur que nous nous proposons de réaliser. Cependant, ses performances en latence et en débit (par port) sont 10 fois inférieures à celles que nous visons.

CHAPITRE **5**

Description Fonctionnelle

L'objet de ce chapitre est de présenter l'architecture interne du routeur RCUBE, en insistant sur l'aspect modulaire de notre approche. Nous décrirons d'abord:

- le format et le type des caractères reconnus par la macrocellule HSL,
- le codage 9B/12B,
- le format des paquets,
- le mécanisme d'initialisation du routeur,
- le mécanisme de calibration automatique des macrocellules HSL.

Suite à cette présentation, nous décrirons plus finement l'architecture du routeur en présentant successivement:

- la mise en œuvre du contrôle de flux,
- les étapes du routage d'un paquet,
- la gestion des erreurs,
- la stratégie de test.

5.1 Choix architecturaux

5.1.1 Mémorisation en entrée

Bien que la stratégie de mémorisation en entrée ne permette de supporter que 60 % de la charge maximale théorique du réseau, elle présente l'avantage d'être simple à mettre en œuvre comparée aux stratégies de mémorisation centrale ou en sortie. D'autre part, nous disposons d'un générateur de FIFO asynchrones qui réalisent la double fonction de stockage et de resynchronisation des données envoyées par les désérialiseurs avec l'horloge interne du circuit.

5.1.2 Routage wormhole

Le choix de la technique de routage *wormhole* a deux motivations principales. Premièrement, nous souhaitons router des paquets de taille quelconque. Cela nous interdit d'utiliser la technique *store-and-forward* ou *virtual-cut-through*, puisque nous ne pouvons présumer de la taille maximale d'un paquet. Deuxièmement, nous voulons minimiser la latence. La fiabilité des liaisons série nous a permis de choisir une technique de routage qui offre une faible latence, puisqu'il n'y a pas de contrôle d'intégrité au niveau paquet dans chaque routeur.

5.1.3 Schéma de routage adaptatif par intervalle

Comme nous l'avons vu dans le chapitre 3, le schéma de routage par intervalle permet de mettre en œuvre des algorithmes de routage non bloquants pour des topologies quelconques ayant un nombre de nœuds quelconques. Si l'on s'en tient au schéma originel du routage par intervalle, on voit immédiatement que le paquet restera en attente si la sortie associée à l'intervalle est en panne ou déjà allouée à un autre port d'entrée. Pour remédier à cet état de fait, nous utilisons une extension du routage par intervalle en associant non pas une, mais plusieurs sorties à chaque intervalle. Dès lors, si parmi les sorties possibles certaines sont en panne ou occupées, il est possible d'en choisir une autre. Nous avons vu dans le chapitre 4 que le routeur C104 implémente un mécanisme permettant d'associer, à un intervalle, une suite de sorties consécutives. Afin d'offrir une souplesse accrue et permettre la mise en œuvre d'algorithmes de routage plus performants, nous avons voulu pouvoir associer à un intervalle un ensemble quelconque de sorties et définir un ordre de priorité au sein de cet ensemble. Le problème est de trouver un mécanisme simple et rapide permettant de sélectionner une sortie parmi l'ensemble des sortie disponibles. On entend par sorties disponibles toutes les sorties qui ne sont ni déjà allouées, ni en panne.

Ensuite, afin de permettre la réalisation de plusieurs réseaux logiques sur un même réseau physique (partitionnement), nous avons choisi d'implanter une table de routage par port d'entrée.

5.2 Synoptique du routeur RCUBE

Le routeur RCUBE est composé de quatre blocs principaux (fig.5.1):

- le routeur à ports parallèles R3P.
- la macrocellule série HSL: elle assure la sérialisation et la désérialisation des données.
- le bloc SMI (*Serial Macrocell Interface*): il réalise le codage/décodage 9B/12B défini par le standard IEEE 1355 et contient l'automate d'autocalibration de la liaison

série. La spécification fonctionnelle de SMI ainsi que la description de ses deux interfaces vers HSL et FCB sont décrites en annexe D.

- le bloc FCB (*Flow Control Block*): il gère le contrôle de flux défini par le standard IEEE 1355. Ce bloc contient en particulier une file d'attente (FIFO) asynchrone de 64 mots de 10 bits. La spécification fonctionnelle de FCB ainsi que la description de ses deux interfaces vers SMI et R3P sont décrites en annexe C.

Cette décomposition offre plusieurs avantages:

1. **réutilisation**: les blocs HSL, SMI et FCB sont indépendants de la fonction réalisée par le circuit. Ces trois macro-cellules réalisent les trois fonctions définies par le standard IEEE 1355: sérialisation/désérialisation; autocalibration, encodage/décodage 9B/12B; contrôle de flux. En conséquence, ces modules peuvent être utilisés comme des cellules de bibliothèque pour d'autres circuits comportant des liens HSL.
2. **indépendance vis à vis des liaisons série**: pour adapter le routeur R3P à une nouvelle liaison série, il suffit de modifier les blocs HSL, SMI et/ou FCB.

Le routeur parallèle R3P (figure 5.2) est constitué d'un crossbar complet 8×8 et de 8 instances du même modèle de port. Le port (figure 5.3) est quant à lui divisé en 4 blocs, à savoir:

- un calculateur de route,
- un contrôleur du port,
- une file d'attente de sortie de 5 mots de 9 bits,
- une table de routage.

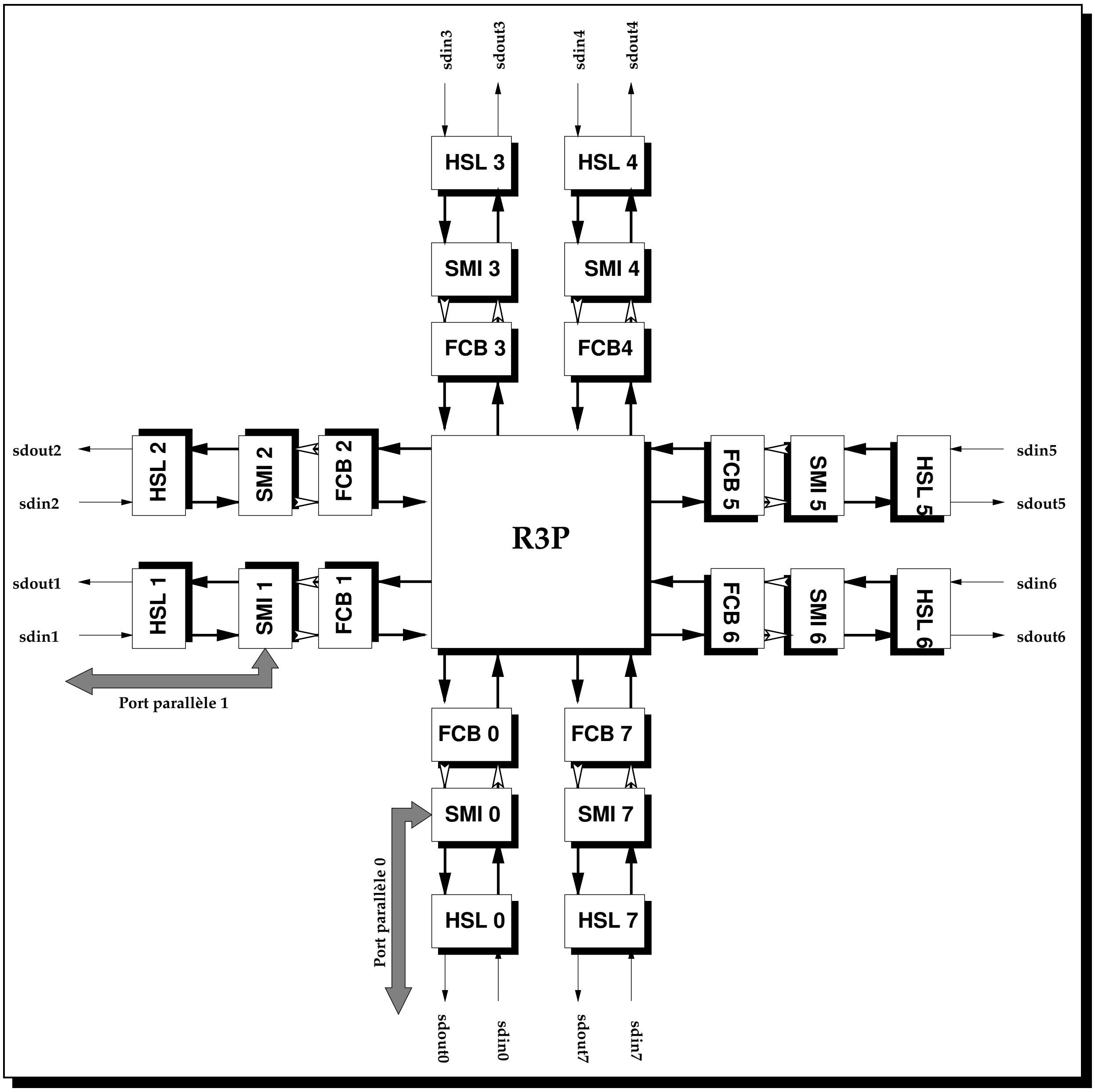


Figure 5.1: Structure logique du routeur RCUBE

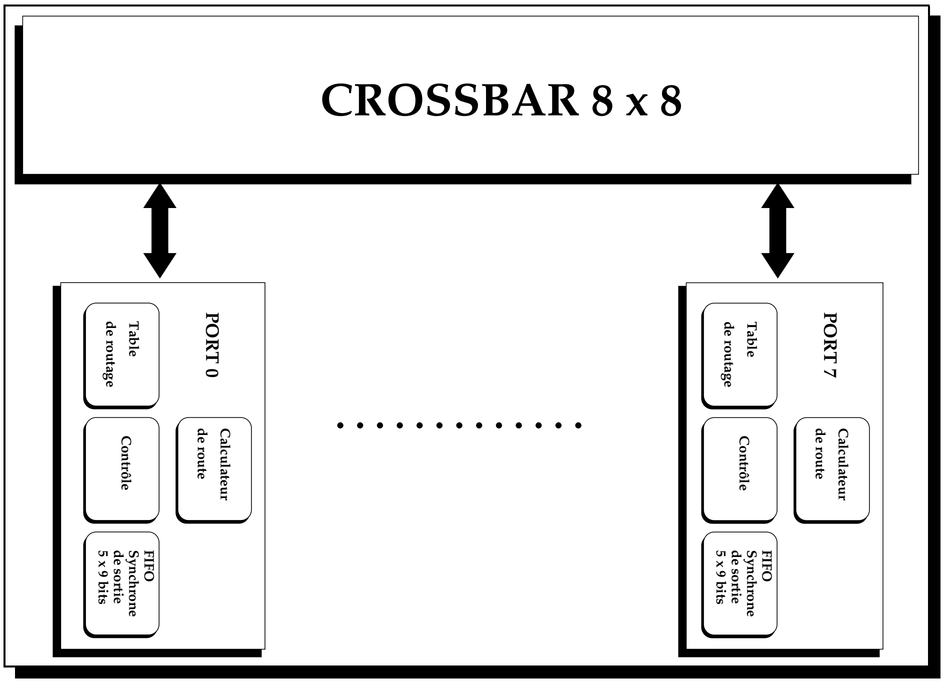


Figure 5.2: Synoptique du routeur parallèle R3P

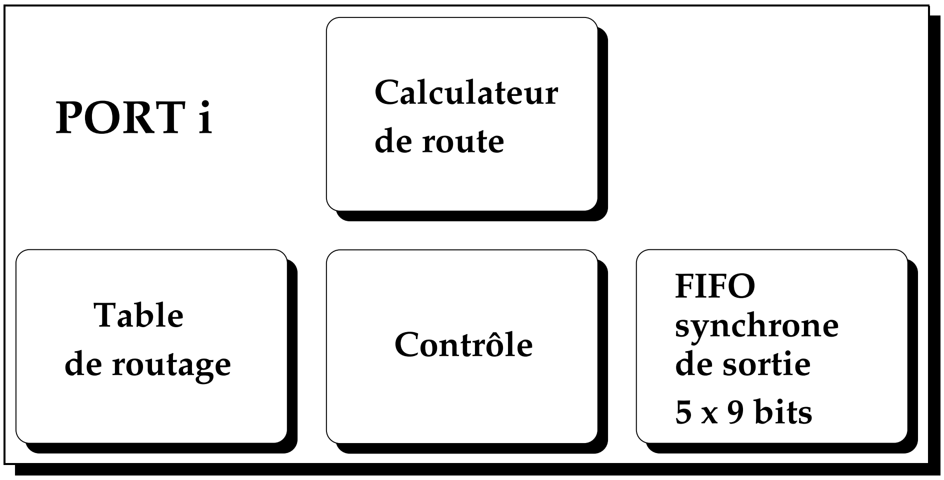


Figure 5.3: Synoptique d'un port du routeur

5.3 Codage 9B/12B

Comme nous l'avons vu dans le chapitre 2, la liaison série HSL impose un codage et un décodage des caractères. Ce codage 9B/12B a pour but:

- d'assurer un front montant précédent chaque caractère afin que la liaison HSL puisse reconstruire un signal d'horloge en réception,
- de maintenir un nombre moyen de '1' et de '0' envoyés identiques afin de conserver une composante électrique continue nulle,
- d'ajouter un bit de parité impaire afin de détecter les erreurs simples de transmission.

5.3.1 Format et type d'un caractère

format d'un caractère

La liaison série HSL manipule des caractères codés sur 12 bits dont le format est le suivant:

MSB											LSB
START	AP	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	INVERT	STOP
1											0

Table 5.1: Codage des caractères sur 12 bits

Les bits START et STOP sont toujours respectivement à '1' et '0' afin d'assurer la présence d'un front montant de synchronisation en début de chaque caractère.

D[0:7] est l'octet que l'on veut transmettre.

Le bit INVERT est à 1 si AP & D[0:7] a été inversé pour des raisons électriques que nous aborderons ci-après, sinon il est à '0'.

Le bit AP est à 1 si D[0:7] & INVERT a un nombre pair de '1', sinon il est à '0'.

type d'un caractère

La norme IEEE 1355 définit deux types de caractères codés sur 9 bits qui sont les caractères de contrôle et les caractères de données. La différence entre les deux types se fait par la valeur du neuvième bit (bit de poids fort) qui est à 1 pour les caractères de contrôle. Ce bit est obtenu en effectuant le décodage du mot de 12 bits issu du désérialiseur de la macrocellule HSL suivant l'algorithme décrit plus loin. Dans l'éventail des codes possibles permettant de définir un caractère de contrôle, certains sont fixés par la norme (par exemple pour le protocole de calibration des liaisons HSL), d'autres non encore utilisés sont laissés libres. Nous utilisons neuf caractères de contrôle.

Dénomination	**Valeur D[0:8]**	**Rôle**
IDLE	1111 0000 1	maintient de la calibration lorsqu'il n'y a pas de données transmises
START_REQ	1111 1000 1	protocole de calibration des liaisons HSL
START_ACK	1110 1000 1	protocole de calibration des liaisons HSL
STOP_REQ	1101 1000 1	protocole de calibration des liaisons HSL
STOP_ACK	1011 1000 1	protocole de calibration des liaisons HSL
STOP_NACK	0111 1000 1	protocole de calibration des liaisons HSL
FCC	0001 1111 1	contrôle de flux entre routeurs
EP	0010 1011 1	marque la fin d'un paquet
EEP	0010 1101 1	marque la fin d'un paquet tronqué suite à la détection d'une erreur

Table 5.2: Table des mots de contrôle utilisés par RCUBE

5.3.2 Code DC-balanced

Afin de maintenir une composante électrique continue nulle sur le média de communication (fibre optique ou coaxial en cuivre), on peut inverser des caractères avant de les transmettre.

Définition 1: La disparité **D** d'un caractère est la différence entre le nombre de '1' et de '0' qui le compose. Les bits à '1' ont le poids 1 tandis que les bits à '0' ont le poids -1. La disparité d'un mot de 8 bits peut donc varier de -8 à +8.

Définition 2: La disparité courante **RD** est la somme des disparités des caractères déjà émis.

Algorithme d'inversion:

- Calculer la parité du mot D[0:7] avec INVERT=0.
- Calculer la disparité D du mot AP & D[0:7] & INVERT=0.
- SI sa disparité D fait décroitre la disparité courante RD, ALORS transmettre le mot en l'état avec le bit INVERT à 0, SINON transmettre le mot inversé avec INVERT à 1. De par sa définition, AP est aussi inversé.
- Les caractères de contrôle ont une disparité D nulle. Afin de les différencier de caractères de données qui ont aussi une disparité nulle, ces derniers sont transmis

inversés contrairement aux caractères de contrôle qui ne sont pas inversés. Cela permet de fixer la valeur du neuvième bit de type.

Le codage et le décodage des mots sont effectués à chaque émission et à chaque réception des caractères. L'algorithme de codage / décodage doit être réalisé matériellement de façon efficace, sous peine d'être la chaîne critique du circuit. Cette fonction est réalisée par le bloc SMI.

5.4 Format des paquets

Compte-tenu du schéma de routage choisi, le format de paquet est peu contraignant. Ce format est décrit sur la figure 5.4.

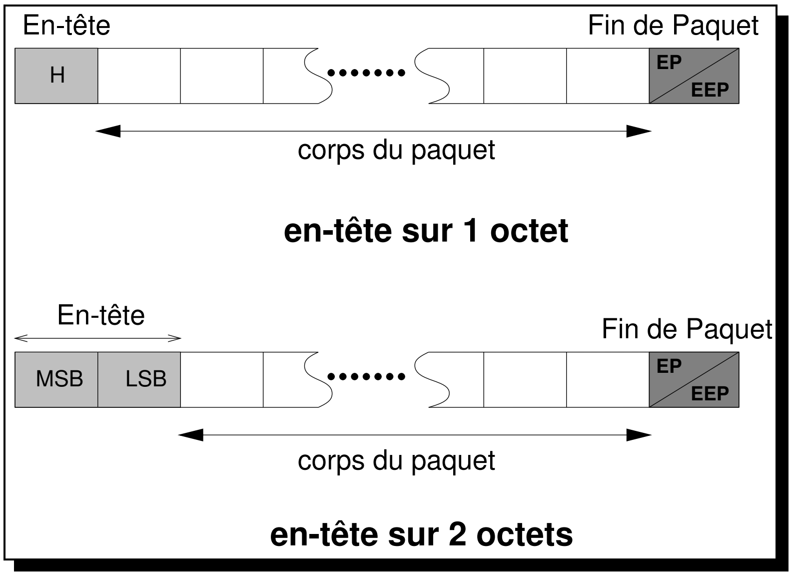


Figure 5.4: Format des paquets routés par RCUBE

EN-TÊTE Les paquets débutent par un en-tête sur un ou deux octets. Cet en-tête contient l'adresse de destination du paquet. On peut donc router un paquet dans un réseau ayant jusqu'à 65536 nœuds.

CORPS Le corps du paquet est de taille quelconque. Il peut contenir des caractères de donnée ou des caractères de contrôle n'appartenant pas aux neuf cités dans la table 5.2. Une liberté est donc donnée aux utilisateurs de définir un protocole de transaction utilisant des mots de contrôle non utilisés et ce, de façon transparente pour le routeur. De la même façon, l'utilisateur peut ajouter dans le corps du paquet des codes de détection d'erreur (codes CRC) qui sont exploités par les protocoles de plus haut niveau.

FIN Puisque le paquet est de taille quelconque, sa fin est marquée par un terminateur de paquet. Ce terminateur de paquet peut être l'un des deux caractères de contrôle suivant: EP ou EEP. EP est le terminateur normal tandis que EEP indique que le paquet a subit une altération et a été tronqué durant son trajet (Cf. section 5.9 et 5.10).

5.5 Configuration du routeur RCUBE

Le routeur RCUBE doit permettre la construction de réseaux de topologies quelconques. Comme nous l'avons vu précédemment, nous avons choisi le schéma de routage par intervalle. Ce schéma de routage utilise des tables (une par port d'entrée) (figure 5.5) que l'utilisateur doit initialiser. Cette phase de configuration consiste à charger dans différents types de registres les valeurs adéquates. Nous allons décrire la structure et la fonction de ces registres.

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
BORNE 0	BORNE 0	LISTE SORTIES 0	PRIORITE 0
BORNE 1	BORNE 1	LISTE SORTIES 1	PRIORITE 1
BORNE 2	BORNE 2	LISTE SORTIES 2	PRIORITE 2
BORNE 3	BORNE 3	LISTE SORTIES 3	PRIORITE 3
BORNE 4	BORNE 4	LISTE SORTIES 4	PRIORITE 4
BORNE 5	BORNE 5	LISTE SORTIES 5	PRIORITE 5
BORNE 6	BORNE 6	LISTE SORTIES 6	PRIORITE 6
BORNE 7	BORNE 7	LISTE SORTIES 7	PRIORITE 7
MSB	LSB		
Limites d'intervalle		Listes des sorties	Mots de priorite

Figure 5.5: Structure d'une table de routage

Définition des intervalles

Le nombre d'intervalles que l'on a défini est huit. C'est le nombre minimum d'intervalles qui permet de profiter pleinement des huit ports et d'émuler la fonction de routage explicite. Cette émulation est effectuée en associant le port de sortie i à l'intervalle i. En utilisant le mode *Header Deletion*, l'utilisateur peut construire l'en-tête de des paquets en spécifiant la suite de sorties à emprunter pour rallier la destination.

Afin de permettre la réalisation de réseaux comportant plus de 256 nœuds, nous avons choisi de supporter des en-têtes de paquets codés sur 1 ou 2 octets. Les intervalles sont donc définis par deux valeurs codées sur deux octets. Ces deux valeurs sont appelées borne inférieure et borne supérieure de l'intervalle. Les valeurs des bornes sont ordonnées de façon strictement croissante afin que les intervalles définis soient non recouvrants. La définition des intervalles est illustrée sur la figure 5.6.

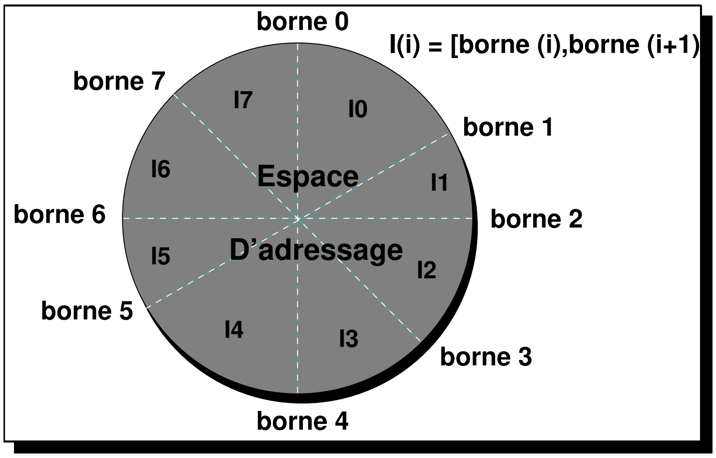


Figure 5.6: Division de l'espace d'adressage en intervalles

Nous disposons donc de huit intervalles définis comme suit:

1. [borne0, borne1[
2. [borne1, borne2[
3. [borne2, borne3[
4. [borne3, borne4[
5. [borne4, borne5[
6. [borne5, borne6[
7. [borne6, borne7[
8. [borne7, 65535] $\cup$ [0, borne0[

Pour chacun de ces intervalles, on associe:

- le registre des sorties (8 bits): en positionnant le bit correspondant à 1, l'utilisateur sélectionne de une à huit sorties pouvant être prises par un paquet dont l'en-tête appartient à l'intervalle;
- le registre de priorité (8 bits): afin de définir un ordre de priorité parmi les sorties possibles associées à un intervalle, l'utilisateur doit spécifier un mot de priorité. Seuls les 7 bits de poids faible sont effectivement utilisés.

Deux autres registres de configuration existent. Ce sont:

- le registre HD (*Header Deletion*) (8 bits): ce registre permet à l'utilisateur de spécifier quels sont les ports de sortie qui sont en mode *Header Deletion* en positionnant à 1 le bit correspondant.
- le registre HL (*Header Length*) (8 bits): sur le même principe, ce registre permet à l'utilisateur de spécifier quelle est la taille des en-têtes traités par chaque port d'entrée.

Cette phase de configuration doit être effectuée à chaque mise sous tension du routeur. Tous ces registres sont chaînés de façon à former un registre à décalage de largeur 8 bits et d'une longueur de 258 octets. Le chargement de ce registre à décalage est cadencé par une horloge dédiée, différente de l'horloge de fonctionnement du routeur.

5.6 Autocalibration des macrocellules HSL

5.6.1 Problème du transfert asynchrone

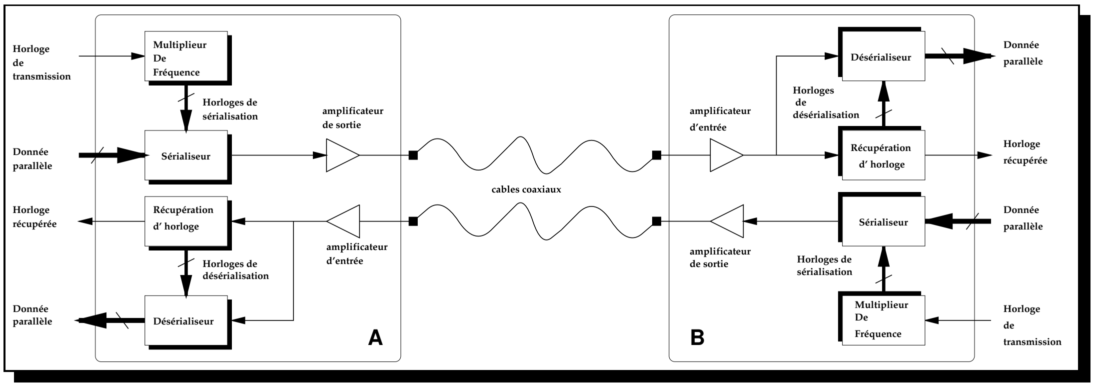


Figure 5.7: Connexion de deux macrocellules HSL

Dans le chapitre 2, nous avons expliqué en quoi le caractère asynchrone des échanges entre macrocellules HSL apportait une solution aux problèmes de distribution d'horloge au niveau système. En contrepartie, une difficulté apparaît: faire en sorte que deux macrocellules HSL (fig. 5.7), connectées via deux câbles coaxiaux et appartenant à deux systèmes différents cadencés par deux horloges différentes, soient néanmoins en mesure de s'échanger des données. En effet, une macrocellule HSL émettrice envoie sur le câble un flot série de bits qui doivent être échantillonnés par la macrocellule HSL réceptrice. L'horloge de transmission qui a servi à sérialiser les mots en émission est transmise sur le câble sous la forme d'une transition montante tous les 12 bits (bits START et STOP). Le macrocellule réceptrice doit se synchroniser sur cette horloge transmise, pour être en mesure d'échantillonner correctement les bits de données. Cette phase de synchronisation des macrocellules est appelée **calibration** et est automatique.

5.6.2 Protocole de calibration

Le protocole de calibration d'une macrocellule HSL est implanté dans le bloc SMI sous la forme d'un automate d'états finis, cadencé par l'horloge interne du routeur. Sa description est faite en annexe. Nous ne donnons ici que son principe.

Après une phase d'initialisation des sérialiseurs contenus dans chaque macrocellule, la phase de synchronisation effective peut débuter. Les sérialiseurs transmettent des caractères de contrôle spécifiques (IDLE), dont le rôle est de permettre aux désérialiseurs de se synchroniser avec l'horloge transmise par les sérialiseurs. Lorsque le désérialiseur est synchronisé avec l'horloge récupérée, un signal d'état appelé CAL, synchrone avec l'horloge récupérée, passe à '1'. Ce signal doit lui-même être resynchronisé avec l'horloge interne du routeur afin que l'automate de gestion du protocole de calibration puisse en tenir compte. Le signal CAL resynchronisé permet à l'automate de conclure la phase de calibration en démarrant un protocole d'échange de requêtes et d'accusés de réception visant à assurer que les deux macrocellules A et B peuvent communiquer l'une avec l'autre. À la fin de cette phase, un signal LINKSTATUS (un par liaison série) passe à '1', indiquant au contrôleur du port que le lien série est fonctionnel.

5.6.3 Autocalibration

La déconnexion physique d'une liaison, ou la perte de calibration ont pour effet de faire passer à '0' les signaux CAL et LINKSTATUS indiquant au routeur que la liaison ne doit plus être utilisée. Cet état est détecté par l'automate de calibration qui relance le protocole de synchronisation, au cas où la connexion physique serait rétablie. De cette façon, l'autocalibration est réalisée. Il est possible de forcer l'initialisation sélective d'un lien grâce à un signal NRESET_LINK (un par lien) présent dans l'interface du circuit et ce, sans remettre en cause le fonctionnement des autres liaisons.

5.7 Contrôle de flux

5.7.1 Contrôle de flux externe

Afin d'assurer un transport sans perte (en l'absence de panne), la norme IEEE 1355 définit un mécanisme de contrôle de flux. Ce contrôle de flux permet à une routeur A d'indiquer à son voisin B qu'il dispose de suffisamment de place dans sa mémoire pour recevoir des données.

L'unité indivisible sur lequel se base le contrôle de flux est le *flit*, composé de 32 caractères. Ces caractères peuvent être des caractères de données ou des caractères de contrôle à l'exception des 7 premiers caractères de contrôle cités dans la table 5.2. Ce sont en fait tous les caractères qui sont transmis d'une entrée vers une sortie du routeur. La technique employée utilise un caractère de contrôle dédié appelé FCC qui est inséré dans le flux des données progressant en sens inverse. Pour un *flit* de 32 caractères, cela conduit à une perte en bande passante de 3 % environ. La figure 5.8 illustre ce mécanisme.

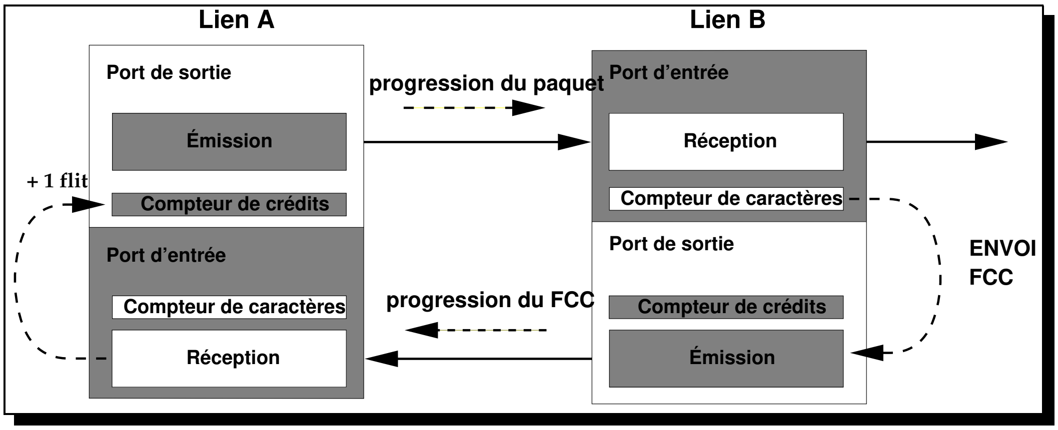


Figure 5.8: Contrôle de flux externe

Deux routeurs A et B sont reliés par un lien série bidirectionnel. A chaque fois que B consomme, dans sa FIFO d'entrée, un caractère de données envoyé par A, un compteur de caractères s'incrémente. Lorsque le compteur atteint 32, la partie réception du lien B commande à sa partie émission d'insérer dans le flux de données qui transitent de B vers A, un caractère FCC afin d'informer la partie émission du lien A qu'il dispose d'un crédit d'émission de 32 caractères.

Dans le sens inverse, le port de sortie de A dispose d'un crédit d'émission qui peut être égal à un ou deux *flit*. À chaque envoi de donnée, le crédit d'émission est décrémenté de 1. Lorsque ce crédit est nul, A stoppe son émission. Ce crédit est augmenté d'un *flit* à chaque réception d'un FCC mais il est limité à deux *flits* (taille de la FIFO d'entrée).

5.7.2 Contrôle de flux interne

Le mécanisme de contrôle de flux a pour effet de rétro-propager la contention vers l'amont. Il y a donc aussi un contrôle de flux interne au routeur lui-même, entre un port d'entrée et le port de sortie qui lui a été alloué. Conséquemment au fait de router des paquets de taille quelconque, un *flit* peut contenir des caractères appartenant à deux paquets différents ayant des destinations différentes. De ce fait, le contrôle de flux entre l'entrée et la sortie ne peut se faire sur la base d'un *flit* mais doit se faire au niveau caractère. Chaque port de sortie contient une petite FIFO disposant d'un indicateur de remplissage dont le but est de stocker les quelques caractères que l'entrée émet avant d'être informée que le crédit d'émission en sortie est nul.

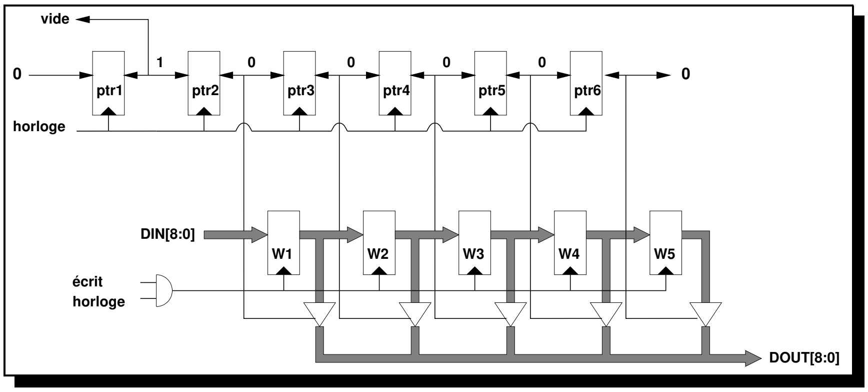


Figure 5.9: Principe du contrôle de flux interne

Cette FIFO de sortie est totalement synchrone. Elle se compose d'une partie mémoire de 5 mots de 9 bits. Le fonctionnement de cette FIFO ainsi que l'explication de sa profondeur, sont décrites en annexe A.

5.8 Les différentes phases du routage d'un paquet

Outre les phases de désérialisation et de sérialisation des caractères, les différentes phases du routage d'un paquet peuvent être décomposées comme suit:

1. décodage des caractères reçus par le désérialiseur,
2. mémorisation et resynchronisation des caractères dans la FIFO d'entrée,
3. sélection de la sortie à emprunter lorsqu'il s'agit de l'en-tête d'un paquet,
4. transfert des caractères de la FIFO d'entrée vers la FIFO de sortie,
5. codage et envoi des caractères vers le sérialiseur.

Il est clair que chacune de ces opérations peut prendre un ou plusieurs cycles selon sa complexité. Toute la difficulté réside dans le fait d'équilibrer chacune de ces opérations, afin d'obtenir un compromis entre le nombre de cycles nécessaire à router l'en-tête et la fréquence de fonctionnement visée de 80 MHz. Nous avons donc choisi de réaliser le séquenceur chargé de gérer ces différentes phases sous la forme d'un automate de Mealey plus compact permettant de diminuer le nombre de cycles de routage. Par contre, des problèmes de chaînes longues sont susceptibles d'apparaître. Dans la suite du chapitre, nous nous contenterons de décrire le séquencement des phases de routage sans aborder les problèmes temporels rencontrés. Ces problèmes feront l'objet de chapitre 7.

5.8.1 Cycle 1: décodage d'un caractère

Après qu'un caractère a été désérialisé, il doit être décodé et resynchronisé avec l'horloge interne du routeur. En effet, le caractère reconstruit par le désérialiseur est synchrone avec l'horloge récupérée par ce même désérialiseur. Cette horloge récupérée est différente de l'horloge interne du routeur. La phase de décodage a pour but d'effectuer la conversion 12B/9B et de contrôler la parité des caractères reçus.

Réception d'un caractère

Parallèlement au décodage, nous filtrons les caractères de contrôle du protocole d'initialisation des liens série, les caractères de contrôle de flux et les caractères IDLE, afin de ne transmettre que les caractères "utiles" à l'utilisateur.

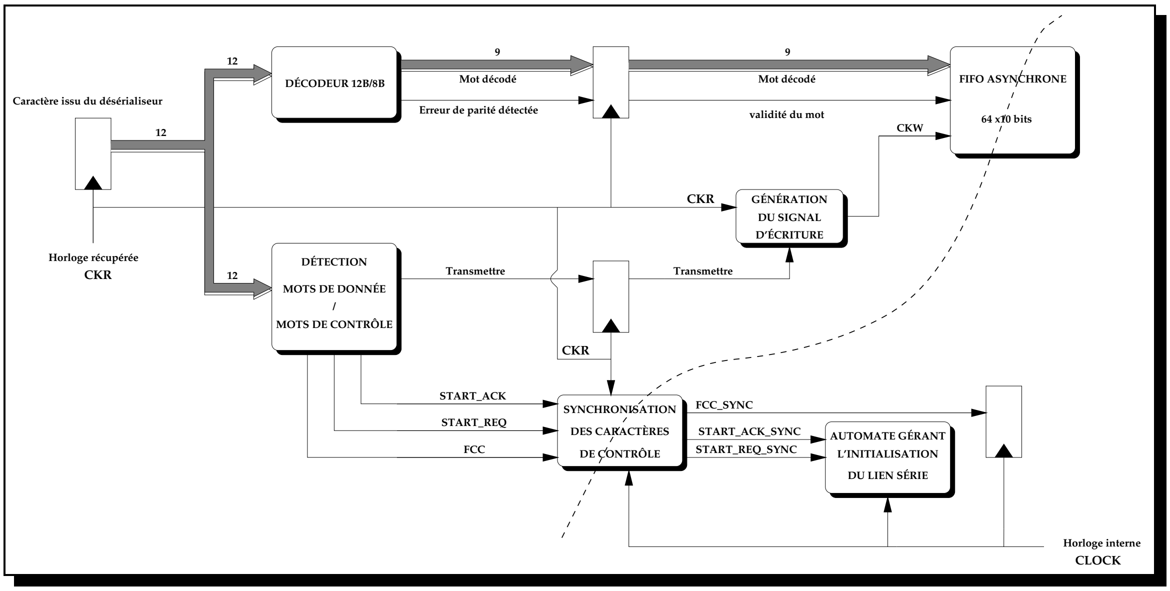


Figure 5.10: Réception d'un caractère

Le décodeur génère une donnée sur 10 bits (8 bits de données plus 1 bit de type et un bit indiquant la validité de la parité du caractère reçu). Le module de filtrage génère un signal *Transmettre* indiquant si le caractère reçu doit être écrit dans la FIFO d'entrée.

5.8.2 Cycle 2: resynchronisation des caractères reçus

La FIFO asynchrone

Le rôle de cette FIFO asynchrone est de resynchroniser les caractères de données reçus: on écrit dans la FIFO en utilisant l'horloge récupérée par la macrocellule HSL, appelée CKR et on lit dans la FIFO à la fréquence interne de fonctionnement du routeur contrôlé par l'horloge CK. L'écriture dans la FIFO est assurée par un bloc dédié qui, à partir de l'horloge CKR et du signal *Transmettre*, génère une impulsion d'une durée égale à un cycle de l'horloge récupérée. Notons que même si une erreur de parité a été détectée sur un mot de donnée, celui-ci est écrit dans la FIFO. Le traitement de cette erreur est effectuée par le contrôleur du port après resynchronisation.

La taille de la FIFO est limitée à 64 caractères soit deux *flits*, à cause de problèmes électriques dégradant les performances de la FIFO pour des capacités supérieures à deux *flits*.

Resynchronisation des caractères de contrôle de flux

Les caractères transmis d'un port d'entrée vers un port de sortie sont resynchronisés grâce à la FIFO asynchrone. Par contre, il n'est pas possible d'utiliser la FIFO asynchrone pour resynchroniser les caractères de contrôle de flux FCC. En effet, considérons deux routeurs A et B qui communiquent. Lorsque A envoie un caractère FCC à B, A indique ainsi à B qu'il est en mesure de recevoir un *flit* de 32 caractères. B procède de même en sens inverse. Si le caractère FCC est autorisé à entrer dans la FIFO, A et B ne peuvent le détecter que lorsqu'ils ont lu tous les caractères précédant le caractère FCC dans la FIFO. Or la lecture dans la FIFO dépend du fait que la sortie à laquelle est connectée l'entrée du routeur, est autorisée à émettre. Si ce n'est pas le cas, le caractère FCC est bloqué dans la FIFO d'entrée du routeur et celui-ci n'émet plus de données. Le caractère FCC envoyé par A vers B et celui envoyé par B vers A sont tous les deux dans la FIFO et dépendent l'un de l'autre pour être lu. Une dépendance cyclique est créée et une situation d'interblocage apparaît. Il faut donc un mécanisme de resynchronisation différent pour les caractères FCC. Une contrainte forte de cet opérateur de resynchronisation est qu'il ne doit pas générer d'impulsion parasite qui pourrait induire la logique d'émission en erreur en lui indiquant à tort qu'elle dispose de plus de crédits qu'il n'en existe en réalité. Cet opérateur est décrit en annexe B.

5.8.3 Cycle 3: lecture dans la FIFO

La lecture des caractères dans la FIFO d'entrée est commandée par le contrôleur du port qui fonctionne à la fréquence de l'horloge interne CK. La FIFO asynchrone dispose d'un signal NEMPTY qui indique si le mot présent dans le registre de sortie de la FIFO est valide ou non. Tant que la donnée présente est invalide, une lecture est effectuée. Cela a pour effet de charger un nouveau mot dans le registre de sortie de la FIFO. Lorsque le signal NEMPTY indique que la donnée est valide, le contrôleur du port continue ou suspend la lecture suivant le cas. La lecture est interrompue pendant la durée nécessaire au traitement de l'en-tête d'un paquet, ou lorsque la FIFO de sortie est pleine.

5.8.4 Cycles 3 et 4: comparaison de l'en-tête

Le choix d'une sortie dépend de la valeur de l'en-tête du paquet, du contenu de la table de routage du port d'entrée et de l'état des ports de sortie du routeur.

L'en-tête du paquet est lu dans la FIFO. Si cet en-tête est sur deux octets, c'est d'abord l'octet de poids fort présent dans le registre de sortie de la FIFO d'entrée qui est comparé simultanément aux huit bornes d'intervalle contenues dans la table de routage (figure 5.11). Le résultat de cette comparaison est sauvegardé, ainsi que l'octet de poids fort, dans deux registres temporaires. Si l'en-tête est sur deux octets, au cycle suivant, l'octet de poids faible présent dans le registre de sortie de la FIFO est à son tour comparé aux bornes d'intervalle (figure 5.12). En tenant compte de la valeur de la comparaison de l'octet de poids fort, le résultat final de la comparaison est donc mémorisé en fin de cycle 4.

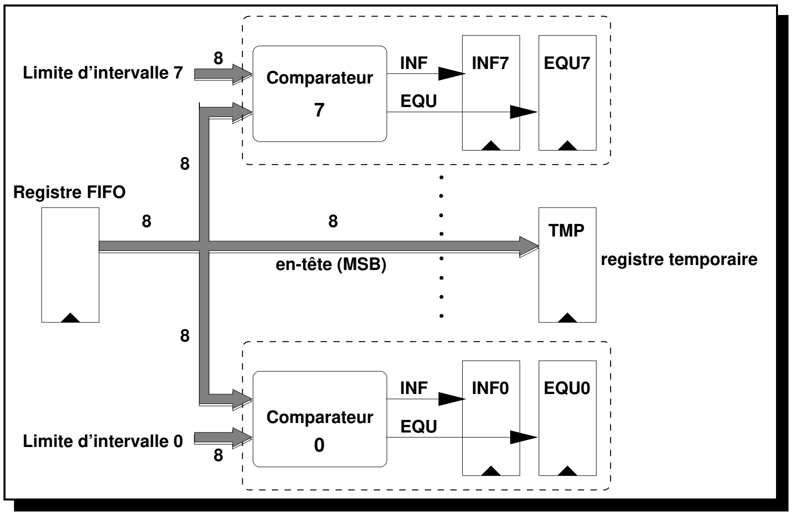


Figure 5.11: Comparaison de l'octet de poids fort de l'en-tête

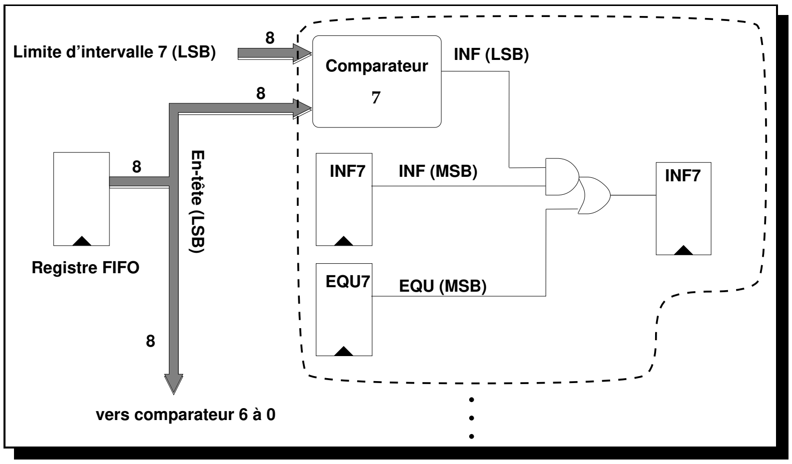


Figure 5.12: Comparaison de l'octet de poids faible de l'en-tête

5.8.5 Cycle 5: génération d'une requête de sortie

À partir des résultats de comparaison de l'en-tête avec les bornes d'intervalle, le sélecteur d'intervalle (figure 5.14) détermine à quel intervalle appartient l'en-tête. Cela permet d'accéder au registre décrivant l'ensemble des sorties permises, ainsi qu'au registre de priorité associés (figure 5.13). L'ensemble des sorties possibles associées à l'intervalle est filtré pour éliminer celles qui sont indisponibles (en panne ou déjà allouées). Ensuite il s'agit de sélectionner parmi les restantes celle qui a la plus grande priorité. Si le registre des sorties ne contient que des zéros (aucune sortie permise), alors le paquet est détruit afin qu'il ne bloque pas les paquets suivants stockés dans la FIFO.

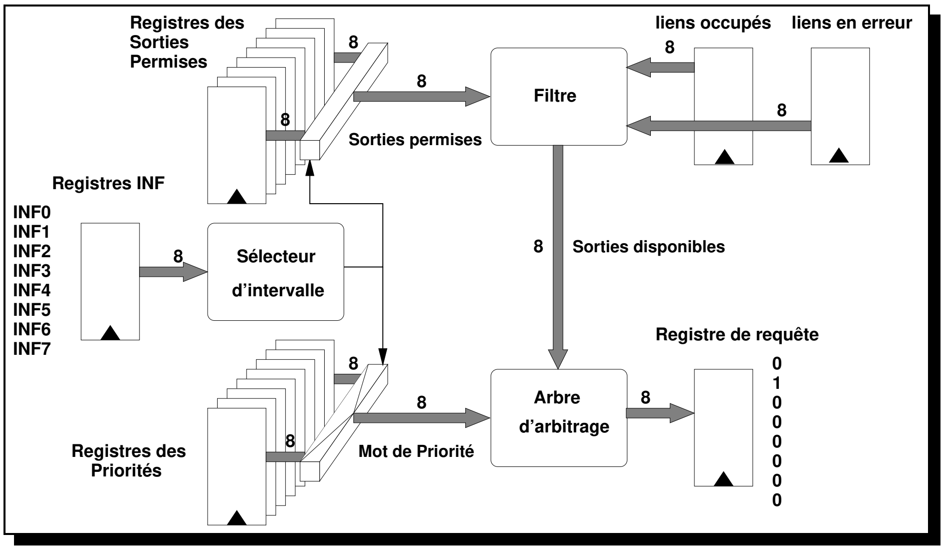


Figure 5.13: Génération d'une requête de sortie

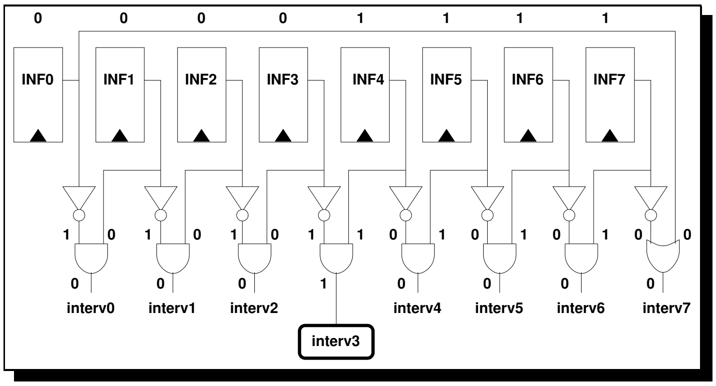


Figure 5.14: Sélecteur d'intervalle

Les registres de priorité

L'ordre de priorité entre les sorties associées à un intervalle est défini par l'utilisateur qui charge, lors de la phase de configuration du routeur, les valeurs adéquates dans les registres de priorité.

L'arbre d'arbitrage

Le choix parmi plusieurs ports de sortie est réalisé à l'aide d'un arbre d'arbitrage dont les entrées sont les sorties disponibles d'une part, et un mot de priorité d'autre part. Sur la figure 5.15, les nœuds de l'arbre représentent les bits du mot de priorité pouvant prendre la valeur '0' ou '1'. Les feuilles terminales de l'arbre représentent les sorties éventuellement présentes en entrée de l'arbitre. Le principe de la sélection est le suivant. Si la valeur d'un nœud est égale à '1' alors les feuilles terminales appartenant au sous-arbre gauche issu du nœud sont prioritaires sur les feuilles terminales appartenant au sous-arbre droit. En appliquant récursivement ce principe à tous les nœuds, on peut définir un ordre de priorité des sorties. Finalement une seule et unique requête est émise en direction de l'allocateur associé à la sortie demandée. L'écriture dans le registre des requêtes n'est autorisée que lorsque l'on traite un en-tête.

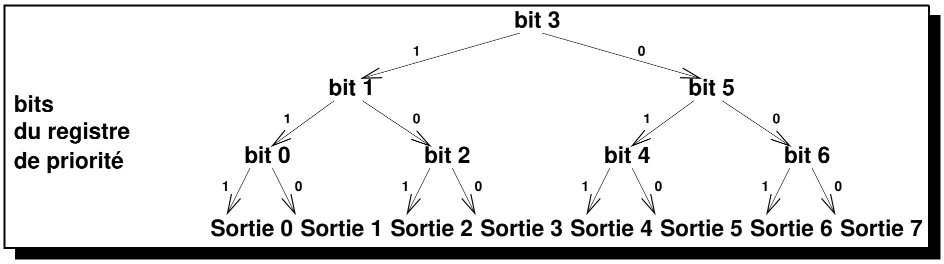


Figure 5.15: Principe de l'arbre d'arbitrage

Cette structure a l'avantage d'être rapide et de croître en nombre de couches logiques en logarithme du nombre de sorties.

5.8.6 Cycle 6: allocation d'une sortie

Il existe un allocateur de sortie par port. Son rôle est d'arbitrer entre les requêtes issues de différents ports d'entrée demandant la même sortie (figure 5.16). Deux principaux problèmes doivent être résolus par l'allocateur: assurer d'une part, l'allocation de la sortie à un seul port d'entrée et assurer d'autre part, qu'un port d'entrée verra sa requête satisfaite au bout d'un temps fini.

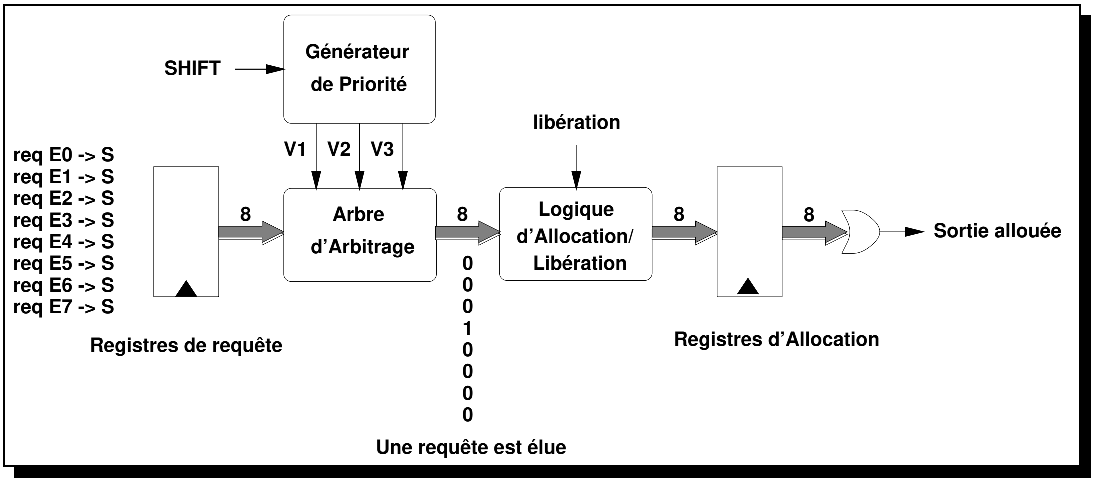


Figure 5.16: Synoptique de l'allocateur de sortie

Logique d'arbitrage

À ce stade, une ou plusieurs entrées Ei émettent une requête vers l'allocateur de la sortie S. Lorsque plusieurs requêtes simultanées sont reçues par l'allocateur, l'arbitrage est effectué d'une manière similaire à celle employée pour choisir une sortie parmi l'ensemble des sorties permises. Nous utilisons donc la même structure d'arbre d'arbitrage, dont les entrées sont cette fois les requêtes simultanées émanant des entrées et un mot de priorité issu d'un générateur de priorité tournante qui est un simple compteur modulo 8. La valeur générée par le compteur change à chaque allocation de la sortie. On évite ainsi les problèmes de famine en donnant successivement à chaque entrée, la priorité la plus élevée.

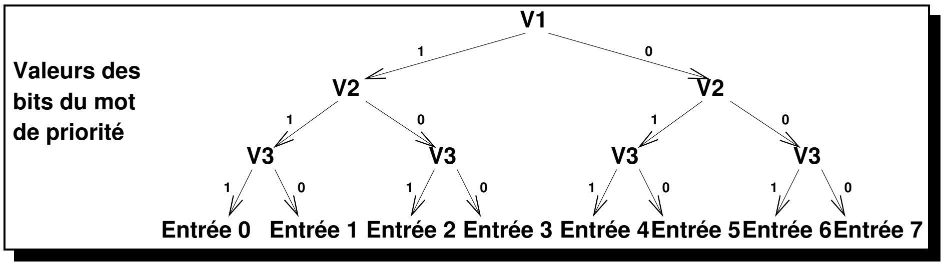


Figure 5.17: Arbre d'arbitrage de l'allocateur de sortie

V1	V2	V3	entrée prioritaire	V1	V2	V3	entrée prioritaire
0	0	0	7	1	0	0	3
0	0	1	6	1	0	1	2
0	1	0	5	1	1	0	1
0	1	1	4	1	1	1	0

Les entrées qui n'ont pu être servies réémettent une nouvelle requête pour une autre sortie au cycle suivant, si elles en ont la possibilité.

5.8.7 Cycle 7: transfert des données de l'entrée vers la sortie

Le signal d'allocation effectuée ouvre le chemin à travers le crossbar entre le port d'entrée et le port de sortie alloué. L'en-tête qui est dans le registre temporaire est transmis et écrit dans la FIFO de sortie. Les mots suivants sont extraits de la FIFO d'entrée et écrits successivement dans la FIFO de sortie jusqu'à ce que le caractère lu soit identifié par le contrôleur du port comme un terminateur de paquet EP ou EEP. Dans ce cas, le terminateur est écrit dans le registre de sortie et la connexion courante entre l'entrée et la sortie rompue. La sortie est donc libérée et disponible pour une autre entrée.

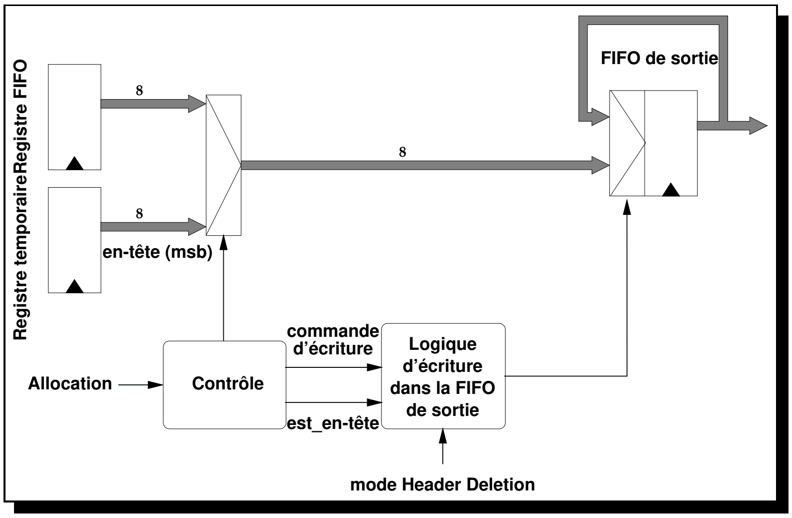


Figure 5.18: Transfert des données d'un port d'entrée vers un port de sortie

Simultanément avec la commande d'écriture dans la FIFO de sortie, le contrôleur du port d'entrée transmet un signal indiquant si le caractère transféré est l'en-tête du message. Grâce à cette information, et au contenu des registre HD (*Header Deletion*) et HL (*Header Length*), la logique d'écriture peut invalider l'ordre d'écriture envoyé par le contrôleur du port. L'en-tête n'est donc pas transmis et la fonction de destruction de l'en-tête *Header Deletion* est réalisée.

5.8.8 Cycle 8: codage des données

La dernière opération consiste à effectuer le codage 9B/12B avant de sérialiser les caractères à émettre. Cette opération spécifique au protocole IEEE 1355 est réalisée par le bloc SMI.

5.8.9 Conclusion

Nous allons maintenant résumer les étapes du routage d'un paquet et discuter le nombre de cycles nécessaire au routage d'un en-tête et d'une donnée.

routage d'un en-tête sur deux octets

Le routage de l'en-tête d'un paquet prend 13 cycles (8 + 5). L'utilisation des liaisons série HSL coûte à elle seule 7 cycles soit plus de la moitié de la latence totale. C'est le prix à payer pour disposer d'un débit élevé et d'une extensibilité aisée. La prise de décision dès lors que le premier octet d'un en-tête est écrit dans la FIFO d'entrée n'est que de 6 cycles. Pour une fréquence de fonctionnement de 80 MHz, cela conduit à une latence de routage de 72 ns (environ 150 ns avec la sérialisation et la désérialisation).

Cycle 1:	Décodage d'un caractère.
Cycle 2:	Resynchronisation du caractère reçu.
Cycle 3:	Lecture dans la FIFO d'entrée et comparaison de l'octet de poids fort de l'en-tête.
Cycle 4:	Lecture dans la FIFO d'entrée et comparaison de l'octet de poids faible de l'en-tête.
Cycle 5:	Génération d'une requête de sortie avec mise en œuvre de l'adaptativité.
Cycle 6:	Allocation d'une sortie avec gestion des conflits entre requêtes.
Cycle 7:	Transfert du caractère du port d'entrée élu vers le port de sortie demandé.
Cycle 8:	Codage du caractère.

Table 5.3: Cycles de routage de l'en-tête d'un paquet

routage d'un caractère de donnée

La latence de routage d'un caractère du corps du paquet est au total de 9 cycles (4 + 5). Cette latence réduite est observable dans le cas où il existe des "bulles" dans un paquet dues par exemple, à l'insertion de caractères de contrôle de flux. Les "bulles" sont donc automatiquement résorbées.

Cycle 1:	Décodage d'un caractère.
Cycle 2:	Resynchronisation du caractère reçu.
Cycle 3:	Lecture dans la FIFO et transfert du caractère du port d'entrée vers le port de sortie.
Cycle 4:	Codage du caractère.

Table 5.4: Cycles de routage d'un caractère de donnée d'un paquet

5.9 Gestion des pertes de calibration

Les pertes de calibration entre deux macrocellules HSL se produisent en cas d'incident matériel sur un lien. Elles sont gérées en deux phases: une phase qui traite le paquet en cours de transmission, une phase de réinitialisation des macrocellules.

Du côté du récepteur du paquet, ce dernier est tronqué et un caractère EEP est ajouté en fin de paquet. La FIFO asynchrone d'entrée est vidée, les caractères qu'elle contenait sont perdus.

Du côté de l'émetteur, le port de sortie est libéré et le lien d'entrée auquel il était connecté via le crossbar, lit et détruit les caractères contenus dans sa FIFO asynchrone d'entrée jusqu'à trouver un caractère Fin de paquet (EP ou EEP).

Dans les deux cas, la procédure de réinitialisation des macrocellules décrite en 5.6 est entamée automatiquement.

5.10 Gestion des erreurs de transmission

Malgré le très faible taux d'erreur de transmission des liaisons HSL relevé en pratique, le cas ne peut être négligé dans la mesure où les conséquences d'une telle erreur peuvent être dramatiques pour le réseau. Voici répertoriées les erreurs possibles et leur degré de gravité.

1. altération d'un caractère de donnée: ce type d'erreur ne compromet pas le fonctionnement global du réseau;

2. altération de l'en-tête d'un paquet : si l'en-tête est incorrect, le paquet est routé vers une destination arbitraire, qui peut entrainer un phénomène d'interblocage;

3. transformation d'un caractère en caractère Fin de Paquet : ce cas rejoint le précédent dans la mesure où le caractère suivant le faux caractère Fin de Paquet est considéré comme l'en-tête d'un nouveau paquet qui sera routé vers une destination quelconque, avec les mêmes risques d'interblocage;

4. perte ou apparition d'un caractère de contrôle de flux FCC : si un caractère de contrôle de flux est perdu, l'émetteur peut cesser de transmettre, faute de crédits d'émission. Les paquets sont donc bloqués et la contention peut se retropropager jusqu'à bloquer le réseau. Si par contre, un caractère de contrôle de flux supplémenaitre apparait alors que la FIFO d'entrée du port récepteur est pleine, cela conduit à un écrasement des données déjà stockées.

Si les erreurs du type 1 peuvent être détectées au niveau du récepteur final, les erreurs du type 2,3 et 4 doivent être détectées le plus rapidement possible. Chaque port d'entrée opère donc diverses vérifications visant à détecter et à réagir aux erreurs de transmission afin de minimiser leur impact sur le fonctionnement global du réseau.

5.10.1 vérification de la parité du caractère entrant

La parité d'un caractère est calculée lors de la conversion 9B/12B. La parité d'un caractère est contrôlée lors de sa réception. Si la parité est correcte, le bit "parité correcte" est mis à 1 sinon il est mis à 0. Quelle que soit la valeur de ce bit, le caractère est écrit dans la FIFO asynchrone d'entrée. Lorsque le caractère est lu, le contrôleur du port s'assure que la parité du caractère est valide. Dans le cas contraire, ce caractère est détruit et remplacé par un caractère de contrôle appelé EEP (Exceptional End of Packet) qui clôt le paquet tout en le marquant incomplet. Le port de sortie est libéré dès l'envoi du caractère EEP permettant ainsi le routage d'une autre paquet issu d'un autre port d'entrée. Le contrôleur du port d'entrée lit et détruit tous les caractères suivant le caractère altéré jusqu'à trouver un caractère Fin de Paquet (EP ou EEP).

5.10.2 vérification du type du caractère entrant

L'erreur de type intervient lorsqu'un caractère de contrôle autre que FCC ou IDLE est reçu entre deux paquets, c'est à dire après un caractère EP ou EEP et avant l'en-tête d'un nouveau paquet qui doit être un caractère de données. La procédure suivie consiste à détruire le caractère altéré ainsi que tous ceux qui le suivent jusqu'à trouver un caractère Fin de Paquet (EP ou EEP).

5.10.3 Apparition ou perte de FCC

Le problème posé par la possible apparition ou perte d'un FCC est ardu, au point même de soulever l'éventualité de la remise en cause de ce protocole de contrôle de flux dans la norme IEEE 1355. En effet, si l'on peut mettre en place des moyens de contourner la perte d'un FCC (grâce à des décompteurs de temps par exemple), par contre, l'apparition d'un FCC parasite est imparable. C'est pourquoi nous avons choisi de ne pas gérer du tout ce problème au niveau matériel.

5.11 Port parallèle et stratégie de test

Afin de faciliter l'accès au routeur par des composants standards, nous avons rendu directement accessibles deux ports du routeur parallèle R3P. L'utilisateur peut au choix utiliser les ports 0 et 1 en port parallèle ou en port série. La présence de ces ports parallèles se justifie aussi par la nécessité de pouvoir accéder au routeur pour le test sur tranche. En effet, le test des puces nues ne peut se faire à la fréquence minimale de fonctionnement requise par les macrocellules HSL (50 MHz) et il est inenvisageable de piloter les ports d'entrée/sortie à 1 gigabit/s. Il faut néanmoins tester les puces nues pour éliminer au maximum celles qui sont non fonctionnelles et éviter une encapsulation coûteuse et inutile.

Ayant un nombre de broches limité, nous ne pouvons nous permettre d'ajouter autant d'accès parallèles qu'il y a de liaisons série. Il nous faut donc prévoir des mécanismes internes permettant de tester les communications entre tous les ports du routeur et ce, à partir de deux accès directs seulement. C'est dans ce but que nous avons intégré un rebouclage parallèle entre l'entrée et la sortie des ports parallèles de R3P (Cf. figure 5.19). En configurant correctement les tables de routage, il est ainsi possible de tester fonctionnellement sur tranche une grande partie du circuit, à l'exception des liens séries, des encodeurs/décodeurs 9B/12B et du mécanisme de resynchronisation dédié aux caractères de contrôle de flux.

Après encapsulation, le circuit est à nouveau testé et peut recevoir une horloge de fréquence élvée. Un mécanisme de rebouclage interne série intégré dans les macro-cellules HSL permet de tester l'autocalibration et d'effectuer, tout comme en mode de rebouclage parallèle, des tests fonctionnels qui valident cette fois, les encodeurs 9B/12B, les décodeurs 12B/9B et la resynchronisation des caractères FCC.

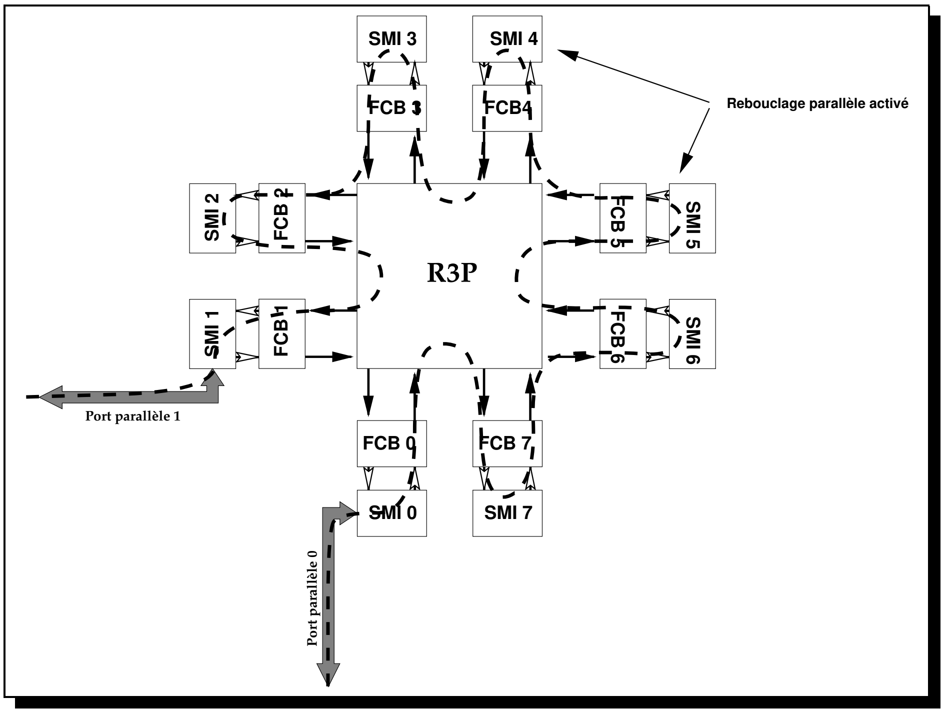


Figure 5.19: Utilisation des ports parallèles en vue du test sur tranche

5.12 Conclusion

Dans ce chapitre, nous avons présenté les choix architecturaux et les différentes phases du routage d'un paquet.

Nous avons montré comment résoudre les problèmes d'asynchronisme en utilisant une FIFO asynchrone qui permet de resynchroniser les caractères reçus à la fréquence récupérée avec l'horloge interne du routeur. Nous avons justifié le besoin de développer des opérateurs dédiés à la resynchronisation des caractères de contrôle FCC et du signal indiquant la calibration des liens HSL.

Nous avons montré comment implanter un mécanisme d'adaptativité rapide et peu coûteux en surface, nous permettant de soulager des ports de sortie fortement chargés. Sa complexité évoluant en logarithme du nombre de sorties dépend peu de la valence du routeur. Un des avantages de ce mécanisme par rapport au routeur C104 est qu'il permet à l'utilisateur de définir une suite non consécutive de sorties, offrant par là même, une plus grande lattitude dans le développement d'algorithmes de routage adaptatifs.

La fonction d'adaptativité permet en outre d'offrir un degré de tolérance aux pannes de type rupture de liaison. La mise en œuvre d'un mécanisme d'autocalibration des liaisons HSL complète l'adaptativité en permettant une remise en fonction des liaisons reconnectées.

L'approche modulaire que nous avons choisie nous permet de nous adapter facilement à de nouveaux standards de liaisons série en ne modifiant que les macro-cellules SMI et/ou FCB (si le contrôle de flux est différent de celui préconisé par le standard IEEE 1355). La concentration dans le bloc SMI des fonctions de calibration, de codage/décodage 9B/12B fait de ce bloc une macrocellule qui peut être réutilisée dans un autre circuit utilisant les liaisons série HSL.

Cette modularité nous permet enfin d'envisager la réalisation de routeurs de valence supérieure. Le principal problème est de disposer d'un crossbar de taille supérieure. Le chapitre suivant aborde donc le problème de la réalisation du crossbar.

CHAPITRE 6

Réalisation matérielle du crossbar

La connexion effective d'un port d'entrée vers un port de sortie est toujours subordonnée au traitement par un opérateur dédié d'une requête émanant du port d'entrée pour obtenir un port de sortie. Le double rôle de cet opérateur est de résoudre les conflits d'allocation lorsque plusieurs ports d'entrée demandent le même port de sortie et d'assurer la connexion physique du port d'entrée vers le port de sortie. Les contraintes sur cet opérateurs sont multiples. Il doit tout d'abord éviter les phénomènes de famine en assurant que chaque port d'entrée verra sa requête satisfaite au bout d'un temps fini. Il doit être non bloquant: la connexion entre un port d'entrée et un port de sortie ne doit pas influencer la connexion d'un autre couple de ports d'entrée/sortie; tous les couples de ports d'entrée/sortie doivent être connectables. En tant qu'élément central du routeur, il doit être compact car il a un impact direct sur la valence maximal du routeur et la surface de la puce. Enfin, il doit être rapide tant pour la résolution des conflits que pour le temps de traversée des données.

Le chapitre 4 nous a permis de présenter les deux principales façons de réaliser cet opérateur.

La première solution retenue par SUN pour le TIC, consiste à utiliser un tampon de type CBDA dont la gestion à base de listes chaînées est assurée par la logique de routage. Cette approche unifie la fonction de mémorisation et de commutation mais elle devient extrêmement couteuse en nombre de pointeurs à gérer lorsque la valence du routeur augmente. En effet, chaque sortie série dispose d'une file d'attente par canal virtuel et pour chaque niveau de priorité. Cela nous donne un total de 4 liens séries x 2 canaux virtuels x 4 niveaux de priorité x 2 pointeurs soit 64 pointeurs. Un routeur TIC de valence 32 nécessiterait de gérer 512 pointeurs. Mais le reproche le plus sévère est le manque totale de modularité de cette approche. L'architecture du tampon et la présence d'un pipe-line à 5 étages dans la logique de routage afin de permettre de scruter séquentiellement les ports d'entrées, sont certes viables pour une routeur à 6 ports mais ne sont plus concevables lorsqu'il s'agit de réaliser un routeur de valence 32.

La seconde méthode, choisie par IBM et SGS-THOMSON est de séparer la fonction de mémorisation et de commutation en utilisant pour cette dernière un crossbar. Nous n'avons malheureusement pas d'informations sur la structure du crossbar du VULCAN, par contre celui du C104 est le résultat d'une conception "sur-mesure".

Nous avons écarté la solution mise en œuvre dans le TIC pour son manque de modularité. Nous avons opté pour l'utilisation d'un crossbar. Cependant, contrairement au C104, nous avons voulu avoir une approche aussi souple que possible pour permettre de réaliser facilement des crossbars de taille quelconques.

C'est dans ce but que nous avons conçu un générateur de matrices de commutation ayant pour paramètres, le nombre de ports n et la largeur des bus m. Les réalisations les plus communes trouvées dans la littérature utilisent une structure multi-étage basée sur des multiplexeurs 2 vers 1 ou 4 vers 1. En utilisant des bus trois-états, nous avons cherché à obtenir une surface aussi faible que possible et un temps de traversée amélioré par rapport aux structures multi-étages. Pour la partie contrôle, là encore, notre effort a porté sur sa modularité et sa rapidité.

6.1 Modèle du crossbar

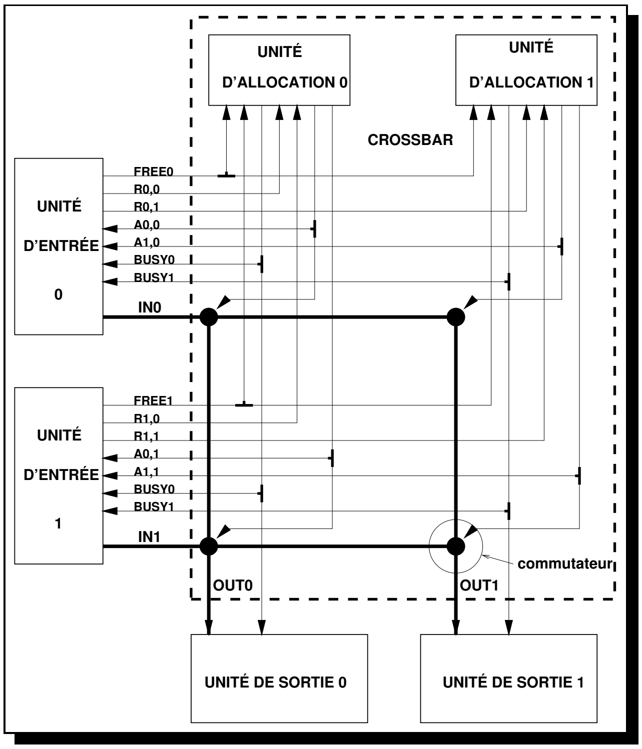


Figure 6.1: Synoptique du crossbar

Notre modèle se compose de n unités d'entrée identiques et indépendantes, et n

unités de sortie identiques et indépendantes. Pour le générateur, n est une puissance de 2 supérieure à 4. Les unités sont numérotées de 0 à $n-1$.

Chaque unité de sortie j est directement connectée à un unique bus de sortie. Chaque unité d'entrée i accède à toutes les unités de sortie. La connexion d'un bus d'entrée à un bus de sortie est effectué grâce à un commutateur réalisé au moyen de portes trois-états.

Les bus véhiculent soit des données soit des signaux de contrôle nécessaires au protocole de transfert mis en œuvre entre une unité d'entrée et une unité de sortie.

Chaque bus de sortie j est commandé par sa propre unité d'allocation j. La figure 6.1 montre l'exemple d'un crossbar 2×2 qui connecte deux unités d'entrée à deux unités de sortie. Les éléments compris dans le rectangle en pointillé constituent le crossbar.

Chaque unité d'entrée i peut envoyer, à tout moment, un signal de requête $R(i, j)$ à chacune des unité d'allocation j afin d'obtenir le bus de sortie correspondant. Plusieurs unités d'entrée peuvent demander simultanément le même bus de sortie. Les unités d'allocation assurent que leur bus de sortie associé n'est alloué qu'à une seule unité d'entrée à la fois. Dans le cas où plusieurs unités d'entrée font la requête du même bus de sortie, une seule d'entre elles est satisfaite grâce à un arbre d'arbitrage dont le principe a été présenté au chapitre précédent. Rappelons que cette logique d'arbitrage utilise un système de priorités tournantes afin d'assurer que toute requête persistante sera satisfaite au bout d'un temps fini. Cette approche est cohérente dans la mesure où le transfert d'un message d'un port d'entrée vers un port de sortie prend un temps limité.

Lorsqu'une unité d'allocation j a élu la requête $R(i, j)$ émanant de l'unité d'entrée i, elle émet en retour, en direction de cette dernière, un signal "allocation effectuée" $A(j, i)$, et simultanément connecte le bus d'entrée i au bus de sortie j en activant le commutateur (i, j). Elle distribue un signal indiquant l'attribution du port de sortie (*busy*) à toutes les unités d'entrée afin que celles-ci n'essayent pas de l'obtenir. Lorsque le transfert est terminé, l'unité d'entrée i envoie à toutes les unités de sortie un signal de libération *free*, mais seule l'unité d'allocation j relache le bus de sortie j en ouvrant le commutateur (i, j). Dès lors, le port de sortie peut être à nouveau alloué. La liste des signaux composant l'interface du crossbar est donnée dans la table 6.1.

6.2 Allocateur de sorties

L'allocateur de sortie contrôle la matrice de commutation. Il est composé de plusieurs unités d'allocation contrôlant chacune les connexions vers leur bus de sortie. Ces unités sont constituées de trois parties (Cf. Figure 6.2):

- un circuit d'arbitrage,
- un générateur de configuration de priorité,
- un circuit d'allocation.

Nom	**Fonction**
$R(i,j)$	Requête du port d'entrée i demandant le port de sortie j
$A(j,i))$	Allocation du port de sortie j au port d'entrée i
	Contrôle du commutateur (i,j)
$busy_j$	État du port j: occupé ou libre
in_i	Bus d'entrée i
out_j	Bus de sortie j
$free_i$	Signal de libération du port de sortie issu du port d'entrée i
$reset$	Signal Reset
$clock$	Horloge

Table 6.1: Interface du crossbar

Le générateur de configuration de priorité est un compteur qui est incrémenté si et seulement si le signal $next$ est actif. La logique d'arbitrage permet de choisir une unique requête parmi toutes celles qui demandent le bus de sortie. Cette logique reçoit en entrée les sorties du générateur de configuration de priorité ainsi que les signaux requêtes $R(i,j), i = 0, \cdots, n-1$. Lorsqu'une requête $R(i,j)$ est élue, le signal s_i correspondant est mis à 1 sinon il demeure à 0.

Le circuit d'allocation contient n registres d'allocation $A(j,i)$. La table 6.2 donne la liste des signaux importants de l'unité d'allocation j.

Signal	**Fonction**
s_i	Sélection de la requête faite par le port i
$A(j,i)$	Signal d'allocation du port de sortie j au port d'entrée i
$free_i$	Signal de libération émanant du port d'entrée i
	Force le bit i de a_j à 0
$busy_j$	OU logique des bits de $A(j,i)$
$clock$	Horloge
$reset$	Reset
$next$	Signal de changement de la configuration des priorités

Table 6.2: Signaux de l'unité d'allocation j

Après initialisation (reset), le circuit d'allocation est dans un état tel que tous les bits des regitres d'allocation A(j,i) sont à '0'. Le signal $busy_j$ est alors à '0' et valide les entrées des registres d'allocation.

Après la phase d'arbitrage des requêtes, un seul bit du registre d'allocation est positionné à '1'. Le signal $busy_j$ passe à '1' et verrouille l'accès au registre d'allocation. Cela assure la mutuelle exclusion des allocations.

Une transition de 0 à 1 du signal $busy_j$ génère une impulsion positive d'une durée d'un cycle d'horloge sur le signal $next$ et le compteur du générateur de configuration de priorités est incrémenté.

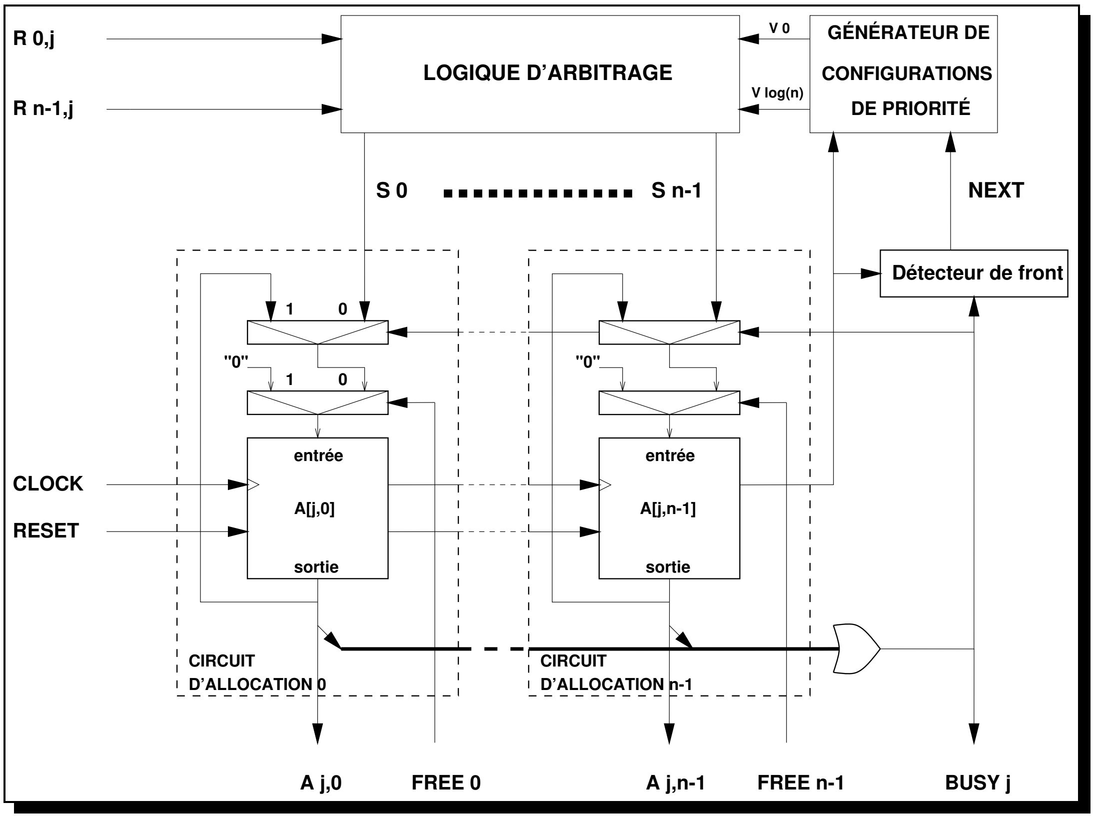


Figure 6.2: Unité d'allocation j

6.3 Réalisation physique

La connexion entre un bus d'entrée et un bus de sortie est assurée par des amplificateurs trois-états (ATE). Le nombre de ces portes à chaque point de jonction dépend de la largeur du bus. La réalisation physique de la matrice de points de commutation repose sur une approche chemin de données basées sur les principes suivants:

- utilisation d'une technologie CMOS à deux niveaux de métal,
- utilisation d'une cellule trois-états de type chemin de données de hauteur T_h et de largeur T_w, disposant de connecteur virtuels et d'un nombre N_t de transparences horizontales[36],
- utilisation d'une cellule de chemin de données vide mais de hauteur identique à l'ATE et ne contenant que des transparences ainsi que les rails d'alimentations.

Le placement relatif des différentes unités pour un crossbar 8×8 ainsi que la disposition des bus de données est décrite figure 6.4. Les bus d'entrée sont routés horizontalement, les bus de sortie sont routés dans les deux directions. La matrice de commutateurs peut être divisée en $\frac{n}{2} \times \frac{n}{2}$ macrocellules régulières $C_{i,j}$ contenant 4

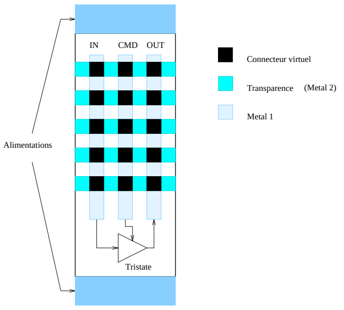


Figure 6.3: Amplificateur trois états

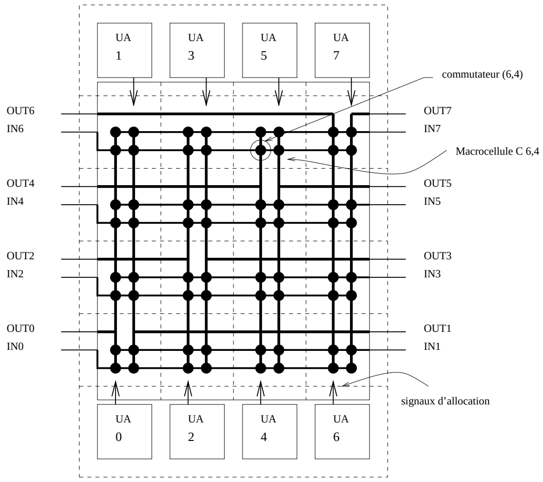


Figure 6.4: Plan de masse du crossbar et stratégie de placement des bus

points de jonction , où i est l'index de l'unité d'entrée/sortie située à gauche et j l'index de l'unité d'allocation située au-dessous du crossbar (i et j sont des entiers pairs). Une macrocellule $C_{i,j}$ est constituée:

- d'ATE qui assurent la connexion du bus d'entrée i aux bus de sortie j et $j+1$,
- d'ATE qui assurent la connexion du bus d'entrée $i+1$ aux bus de sortie j et $j+1$.

Une ligne de macrocellules $C_{i,*}$ forme un chemin de données. On peut distinguer deux types de macrocellules:

- des macrocellules diagonales ($C_{i,j}$ avec $i = j$): elles contiennent une paire de bus de sortie horizontaux et verticaux, ainsi qu'une paire de bus d'entrée horizontaux. Les bus de sortie horizontaux vont de droite à gauche pour une unité de sortie située à gauche, et de gauche à droite pour un unité de sortie située à droite,
- des macrocellules non-diagonales: elles contiennent une paire de bus d'entrée horizontaux et une paire de bus de sortie verticaux ainsi que des transparences horizontales dédiées à une autre paire de bus de sortie.

Les points de connexion internes à une macrocellule sont arrangés de façon que les ATE appartenant à un même point de connexion soient empilés verticalement. Les structures résultantes sont alignées horizontalement. La largeur du chemin de données est donc $H_d \times T_h$ où H_d inclut la largeur du bus b_w et les cellules vides ajoutées nécessaires au routage et à l'aboutement des rails verticaux d'alimentation. L'ordre des points de connexion dans une macrocellule $C_{i,j}$ est de gauche à droite (i, j), $(i+1, j)$, $(i, j+1)$ puis $(i+1, j+1)$. Les canaux verticaux contenant les bus de sortie sont placés respectivement entre les points de connexion (i, j) et $(i+1, j)$ pour le bus de sortie j

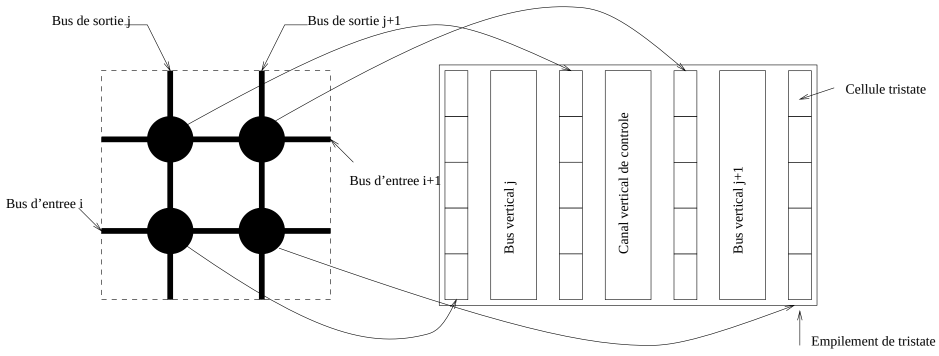


Figure 6.5: Placement des macrocellules de commutation

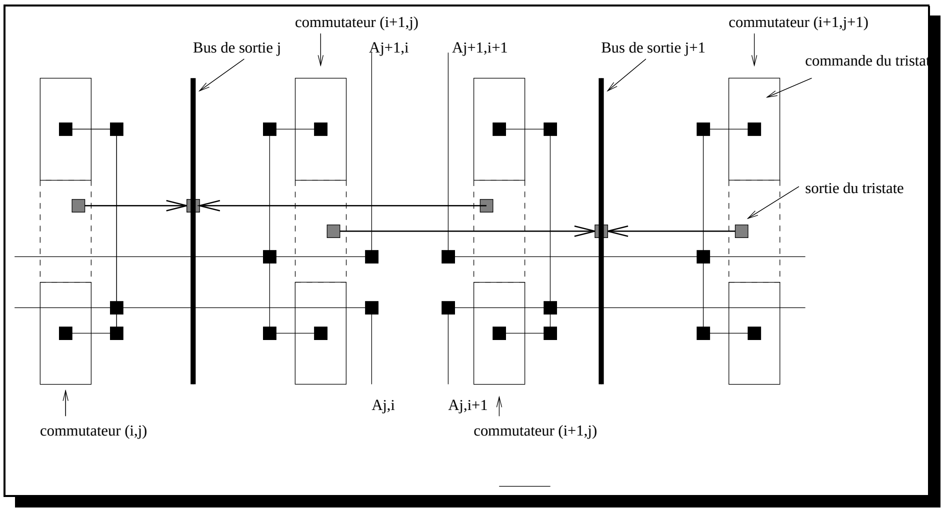


Figure 6.6: Connexions des bus de sorties et des signaux d'allocation dans une macrocellule $C_{i,j}$

et entre les points de connexion $(i, j + 1)$ et $(i + 1, j + 1)$ pour le bus de sortie $(j + 1)$ (Cf. figure 6.5). Un canal situé entre les points de connexions $(i + 1, j)$ et $(i, j + 1)$ est réservé pour le routage de signaux de contrôle.

La figure 6.6 illustre la façon dont les ATE sont connectés aux bus. Le nombre total des pistes d'un bus horizontal est $5 \times b_w$ et le nombre de pistes disponible est $N_t \times H_d$. Le routage des signaux de contrôle restants dépend du type du signal qui peut être:

- un vers tous (*broadcast*): c'est le cas des signaux *busy* et *free*.
- un vers un (unicast): c'est le cas des signaux R(i,j) et A(j,i).

Les signaux de contrôle sont routés le long des canaux verticaux et sur des pistes horizontales utilisant les transparences des ATE.

La figure 6.7 illustre le routage des signaux d'allocation effectuée. Dans chaque unité d'allocation, ces signaux sont numérotés selon l'unité d'entrée de destination.

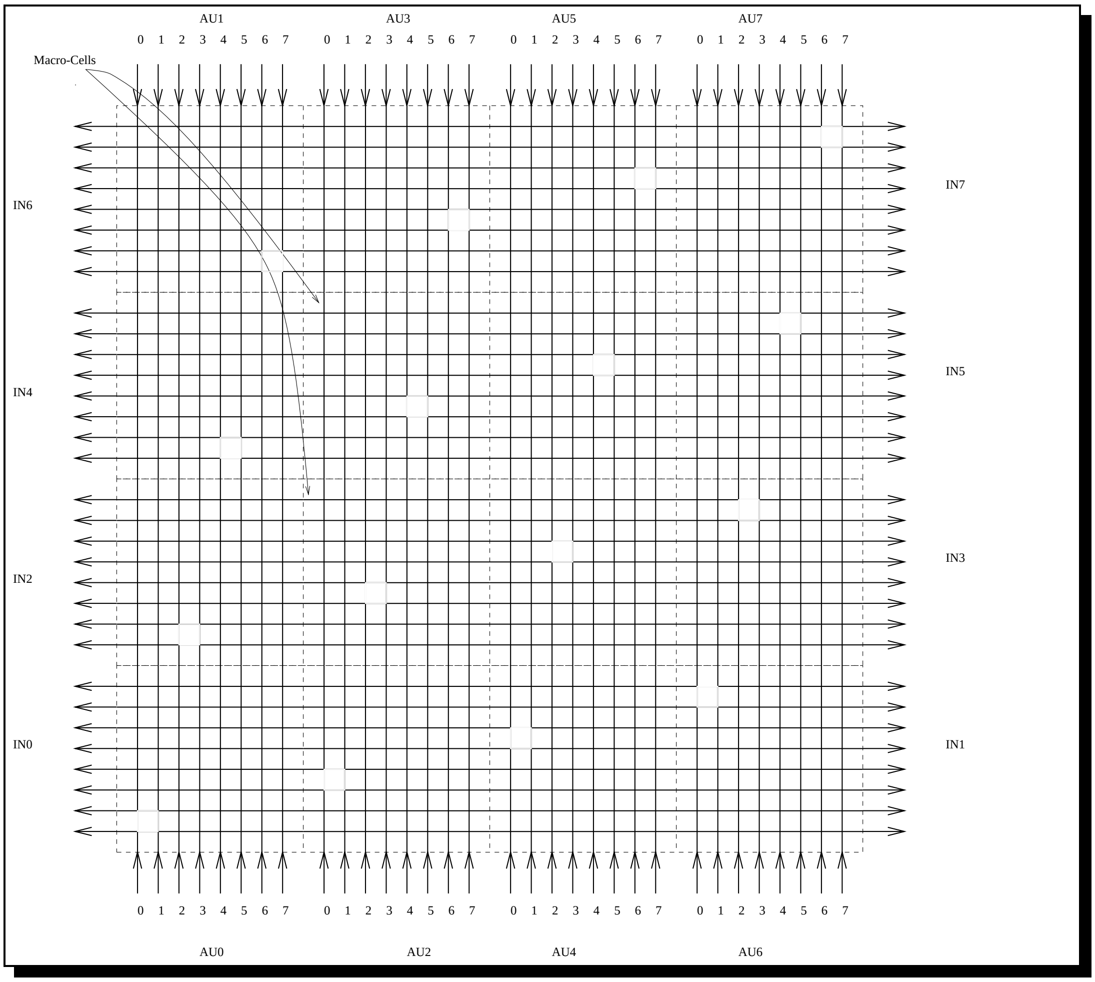


Figure 6.7: Motif de routage des signaux d'allocation

n pistes verticales dans chaque canal de contrôle et n pistes dans chaque chemin de données sont nécessaires pour router ces signaux d'allocation. La connexion d'un signal d'allocation à la commande de la pile correspondante d'ATE est illustrée sur la figure 6.6. Le coût de ce routage est de 4 pistes verticales par macrocellules et b_w pistes horzontales par chemin de données. De manière similaire, les signaux R(i,j) occupent n pistes verticales par canal de contrôle et n pistes horizontales dans chaque chemin de données. Enfin, 2 pistes horizontales par chemin de données et n pistes verticales dans les canaux de contrôle sont utilisées pour router les signaux *free*. Un coût identique est payé pour router les signaux *busy* dans la direction opposée.

Les tables 6.3 and 6.4 résument le nombre total de pistes nécessaires au routage horizontal et vertical des signaux.

Bus d'entrée/sortie	A(j,i)	R(i,j)	Free	Busy	Total
$5 \times b_w$	$n + b_w$	n	2	n	$3 \times n + 6 \times b_w + 2$

Table 6.3: Nombre de pistes horizontales par *slice*

Canaux des bus de sortie	Canal de contrôle	Total
$2 \times b_w$	$3 \times n + 6$	$2 \times (n + b_w) + 6$

Table 6.4: Nombre de pistes verticales par macrocellule $C_{i,j}$

En conséquence la condition nécessaire pour que le routage horizontal soit possible est:

$$\boxed{n \leq (H_d \times N_t - 6 \times b_w - 2)/3}$$

Rappelons que $b_w \leq H_d$.

6.4 Surface et évaluation de performances

Nous avons utilisé une approche mixte utilisant à la fois des cellules précaractérisées pour les modules d'allocation, et des cellules de chemins de données pour générer la matrice de commutateurs. Nous avons développé un générateur spécifique qui à partir de la spécification d'une porte trois-états, de la largeur du bus de données et du nombre d'unités d'entrée/sortie produit la vue physique de la matrice de commutateurs.

L'assemblage final est effectué par routage. L'utilisation de cellules précaractérisées nous permet de réunir les unités d'allocations situées d'un même côté du crossbar en un seul bloc physique dont la longueur correspond à peu de chose près à la largeur de la matrice de commutateurs. Les dimensions S_w (largeur) et S_h (hauteur) de la matrice de commutateurs sont données par les formules suivantes:

$$\boxed{S_h = n/2 \times H_d \times T_h \text{ et } S_w = n/2 \times (4 \times T_w + c_{dr} \times (3 \times n + 2 \times b_w + 6))}$$

où c_{dr} est le *pitch* métal vertical. Cette valeur inclut la largeur de la piste et l'espacement minimal en considérant les vias. Les dimensions de la matrice de commutation pour différentes valeurs de la valence et de la largeur des bus sont données dans la table 6.5. Puisque nous utilisons une technique de dessin symbolique[37], elles sont exprimées en $lambda$ (λ). Dans notre cas, nous utilisons une technologie CMOS 0.5μm à deux niveaux de métal où le λ vaut 0.5μm. La figure 6.8 montre l'exemple du crossbar 8 x 8 utilisé dans RCUBE. Les dimensions de la matrice sont de 1×2mm^2.

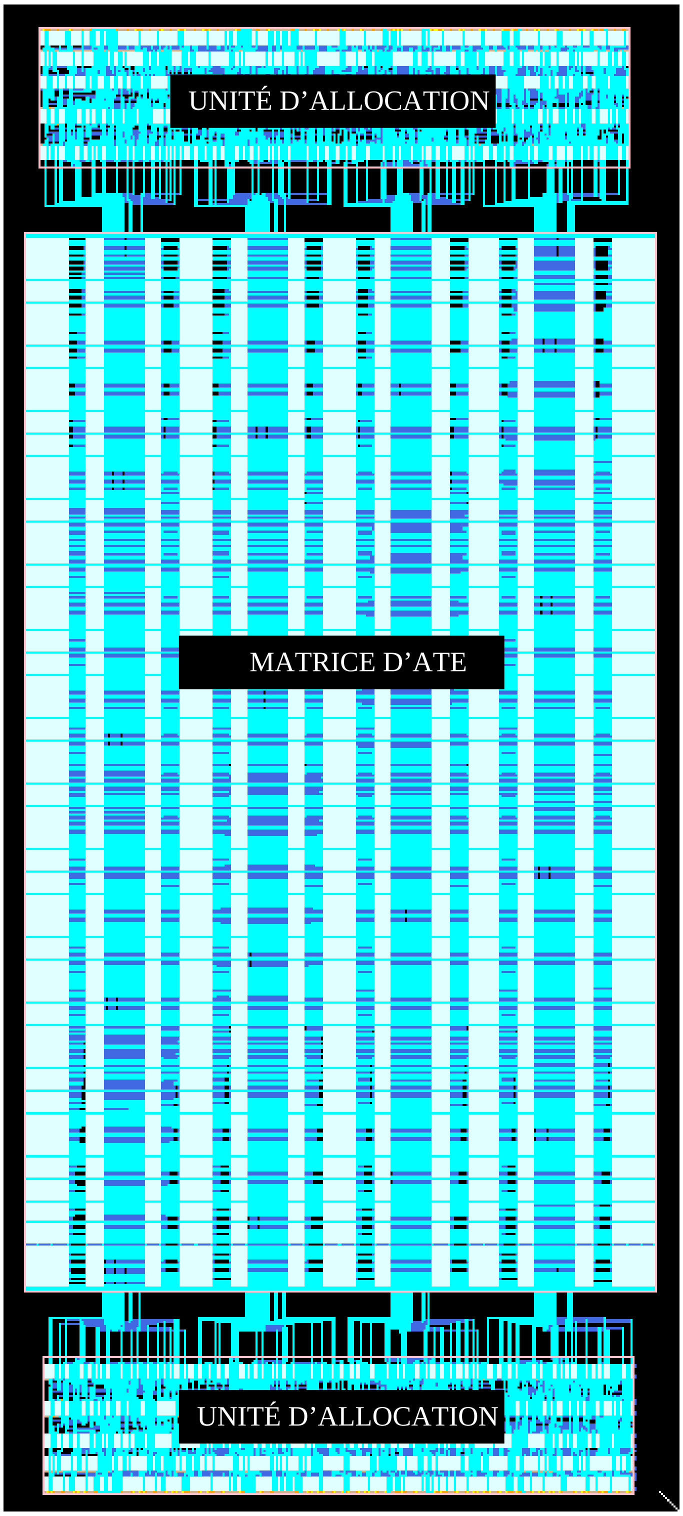


Figure 6.8: Vue physique du crossbar 8 x 8

Le tableau suivant résume les tailles des matrices obtenues pour différentes valences et largeurs de bus. Les performances temporelles sont aussi indiquées.

n	b_w	$H_d(\lambda)$	$S_h(\lambda)$	$S_w(\lambda)$	C_h	délai (ns)
4	1	2	240	520	1138	4
	2	4	480	540	1344	
	4	4	480	580	1285	
	8	8	960	660	1667	
	16	16	1920	820	2489	
	32	32	3840	1140	4249	
8	1	4	960	1280	2630	5
	2	4	960	1320	2580	
	4	6	1440	1400	2967	
	8	8	1920	1560	3290	
	16	16	3840	1880	4977	
	32	32	7680	2520	8528	
16	1	6	2880	3520	5030	6
	2	8	3840	3600	5943	
	4	8	3840	3760	5853	
	8	10	4800	4080	6655	
	16	16	7680	4720	9284	
	32	32	15360	6000	16621	
32	1	12	11520	10880	14678	8
	2	12	11520	11040	14632	
	4	14	13440	11360	16464	
	8	16	15360	12000	18223	
	16	20	19200	13280	21787	
	32	32	30720	15840	32889	

Table 6.5: Dimensions et performances temporelles du crossbar pour les valeurs typiques: $T_h = 60\lambda$, $T_w = 40\lambda$, $c_{dr} = 5\lambda$

6.5 Conclusion

La stratégie que nous avons employée permet de réaliser facilement des matrices de commutation à faible latence pour un choix varié de valences et de largeurs de bus. Grâce à une complexité en délai qui croît de façon logarithmique avec le nombre de ports, l'arbre d'arbitrage permet à la partie contrôle de conserver de bonnes performances temporelles tandis que le nombre de ports augmente. Pour une largeur de bus identique égale à 8 bits, une matrice 16 x 16 occupe une surface de 2.4 mm x 2 mm et présente un délai de traversée de 6 ns. Ceci nous permet d'envisager sans aucun problème la réalisation d'un routeur 16 x 16. Ces résultats montrent que le crossbar n'est pas le facteur limitant l'augmentation de la valence. L'objectif de proposer une architecture aisément extensible en nombre de ports d'entrée/sortie est donc atteint.

CHAPITRE 7

Résultats et performances

Après avoir exposé dans les chapitres précédents, les choix architecturaux, nous présentons ici, les résultats de notre étude en terme de surface et de performances temporelles. Cela nous permettra de tirer des conclusions sur la faisabilité de routeurs de valences supérieures à 8.

7.1 Surface du circuit

Nous avons réutilisé pour le routeur RCUBE, un boîtier et une couronne de plots définis par BULL pour un autre circuit comportant également 8 liaisons série: le circuit ISL [38]. La surface totale de la puce est de 14.6 x 14.6 mm^2. La position des liaisons série nous a été imposée par BULL qui nous a livré un bloc "dur" instanciant les huit liaisons HSL et le bloc d'adaptation d'impédance ZC. Nous nous baserons sur la surface définie par la couronne de plots pour évaluer la faisabilité d'un routeur de valence 16. Nous allons résumer les caractéristiques physiques de chacun des blocs du circuit et leur éventuelle dépendance vis à vis de la valence du routeur.

Il est à noter que le bloc FCB, présent dans le synoptique du routeur RCUBE, n'apparait pas dans le plan de masse. Le synoptique présente un découpage logique du circuit qui est différent du découpage physique. Les fonctions de FCB sont donc réalisées par la FIFO d'entrée qui effectue la resynchronisation et le stockage des caractères, et par le contrôleur du port LC qui effectue le contrôle de flux.

Les différents blocs que nous allons présenter sont:

- SMI: interface avec une macrocellule HSL.
- FIFO: FIFO asynchrone d'entrée.
- LC: contrôleur d'un port du routeur.
- TBREG: table de routage (bornes d'intervalles, registres de configuration des sorties et des priorités), comparateurs.
- ATLOG: générateur d'une requête de sortie.

- OAM: allocateur de sorties.
- CROSSBAR: matrice de commutateur.
- HSL: macrocellule série.
- ZC: adaptateur d'impédance.

SMI : 0.997 mm x 0.587 mm = 0.585 mm^2**, 5066 transistors**

Le bloc SMI assure l'interface avec une macrocellule HSl. Il intègre les fonction de codage/décodage 9B/12B, l'autocalibration des liens HSL, et la resynchronisation des caractères de contrôle de flux. Ces fonctions sont indépendantes de la valence du routeur.

FIFO : 1.057 mm x 0.694 mm = 0.734 mm^2**, 9996 transistors**

La FIFO asynchrone d'entrée a pour rôle de resynchroniser et de stocker les caractères transmis d'un port d'entrée vers un port de sortie. Sa profondeur et la largeur des mots stockés sont indépendantes de la valence du routeur.

LC : 0.906 mm x 0.725 mm = 0.657 mm^2**, 5898 transistors**

Ce bloc assure la fonction de contrôleur d'un port parallèle du routeur R3P. Cette fonction ne dépend pas de la valence du routeur. Les principaux constituants de ce bloc sont la fifo de sortie, le compteur de credit d'émission et l'automate d'états finis qui gère la lecture dans la FIFO d'entrée, les erreurs de parité, de type et de calibration, la commande d'écriture en FIFO de sortie etc...

TBREG : 1.235 mm x 0.802 mm = 0.990 mm^2**, 8834 transistors**

Ce bloc contient une table de routage constituée de 32 registres huit bits (8 bornes d'intervalle sur deux octets, 8 configurations de sortie, 8 configuration de priorité) et les huit comparateurs d'en-tête.

Le passage à une valence de 16 affecte le nombre des registres d'intervalles et la largeur des registres de configuration (sorties et priorité).

Les registres de configuration des sorties indiquent pour chaque intervalle, quelles sont les sorties possibles. En conséquence, pour supporter une valence de 16, la largeur de ces registres doit doubler.

Les registres de configuration de priorité voient aussi leur largeur doubler pour permettre de spécifier un ordre de priorité parmi les 16 sorties permises au maximum.

Pour ce bloc, le coût du passage à une valence 16 est donc de 16 registres 8 bits soit une augmentation de 0.8 mm^2 si l'on conserve par ailleurs le même placement de registres.

ATLOG : 0.429 mm x 0.793 mm = 0.340 mm^2, 2072 transistors

Ce bloc est chargé de l'émission de la requête d'allocation d'un port de sortie. Il reçoit l'état des liaisons (allouée, en panne), les sorties permises, la configuration de priorité. Il contient notamment le sélecteur d'intervalle qui détermine à quel intervalle appartient l'en-tête d'après le résultat des comparaisons simultanées avec les bornes d'intervalles. Il contient aussi un arbre d'arbitrage qui permettra de faire la requête de la sortie disponible de plus haute priorité. Ce bloc est fortement dépendant de la valence du routeur. Cependant, ce bloc est obtenu par synthèse et ne présente pas de difficulté de réalisation particulière. En considérant que pour une valence de 16, le nombre de comparateurs est doublé, mais que par contre, l'arbre d'arbitrage ne comporte qu'un étage supplémentaire, on peut faire l'estimation pessimiste que la surface du bloc ATLOG pour une valence de 16 est multipliée par 2. La surface serait donc de 0.6 mm^2.

OAM : 1.167 mm x 0.441 mm = 0.515 mm^2, 4200 transistors

Ce bloc regroupe 4 allocateurs de sorties. Il contient donc 4 arbres d'arbitrage et les registres d'allocation correspondant aux 4 sorties auxquelles il est associé. Chaque allocateur de sortie doit donc être modifié pour tenir compte de l'augmentation de la valence du routeur. Mais tout comme le bloc précédent, il est obtenu par synthèse. Le passage à une valence de 16 ne présente pas de difficulté. Tout comme pour le bloc ATLOG, on peut faire l'hypothèse que le passage à un valence de 16 entrainerait le doublement de la surface de ce bloc soit 1 mm^2.

CROSSBAR 8 x 8 : 1.1 mm x 1.9 mm = 2.1 mm^2, 11520 transistors

La matrice de commutateurs est obtenue grâce à un générateur. Nous avons vu dans le chapitre précédent, que nous pouvions obtenir une matrice 16 x 16 dont les dimensions seraient de 2 mm x 2.4 mm.

HSL,ZC : 14.2 x 2 = 28.4 mm^2, 190 000 transistors

La macrocellule série HSL assure la sérialisation et la désérialisation des caractères. Ces macrocellules nous ont été livrées en un bloc "dur" instanciant huit d'entre-elles plus le bloc d'adaptation d'impédance ZC. Ce bloc occupe entièrement une face du circuit (la face Nord dans le routeur RCUBE). Si l'on conserve la topologie de ce bloc, un routeur de valence 16 peut être facilement obtenu en le plaçant ces blocs sur deux faces opposées.

Résumé et conclusion

Le tableau suivant résume la surface, le nombre de transistors et la dépendance vis à vis de la valence des différents blocs du circuit .

Blocs	**nombre**	**surface**(mm^2)	**transistors**	**dép. valence**
SMI	8	0.997 x 0.587 = 0.585	5066	non
FIFO	8	1.057 x 0.694 = 0.734	9996	non
LC	8	0.906 x 0.725 = 0.657	5898	non
TBREG	8	1.235 x 0.802 = 0.990	8834	oui
ATLOG	8	0.429 x 0.793 = 0.340	2072	oui
OAM	2	1.167 x 0.441 = 0.515	4200	oui
CROSSBAR	1	1.152 x 1.932 = 2.226	11520	oui
	43	29.6	275000	
8 HSL, ZC	1	28.4	190000	
TOTAL (avec routage)	52	94	465000	

La surface disponible est déterminée par la couronne de plots: 14.6 x 14.6 = 213 mm^2. La surface occupée par les différents blocs dans un routeur de valence 8 est 94 mm^2 (routage inclu) soit 45 % de la surface disponible. En se basant sur le fait que seuls les blocs TBREG, ATLOG, OAM et CROSSBAR voient leur taille augmenter avec la valence, l'évaluation de surface pour un routeur de valence 16 est la suivante:

Blocs	**suface ajoutée** (mm^2)	**nombre**	**total** (mm^2)
TBREG	0.8	16	12.8
ATLOG	100 %	16	5.5
OAM	100 %	2	1.0
CROSSBAR	2.6	1	2.6
TOTAL	22 mm^2		

L'estimation de la surface occupée par un routeur de valence 16 est donc de:

$$\boxed{94 \times 2 + 22 = 210 \text{ mm}^2}$$

La surface occupée par un routeur à 16 liens est parfaitement compatible avec l'actuelle surface disponible.

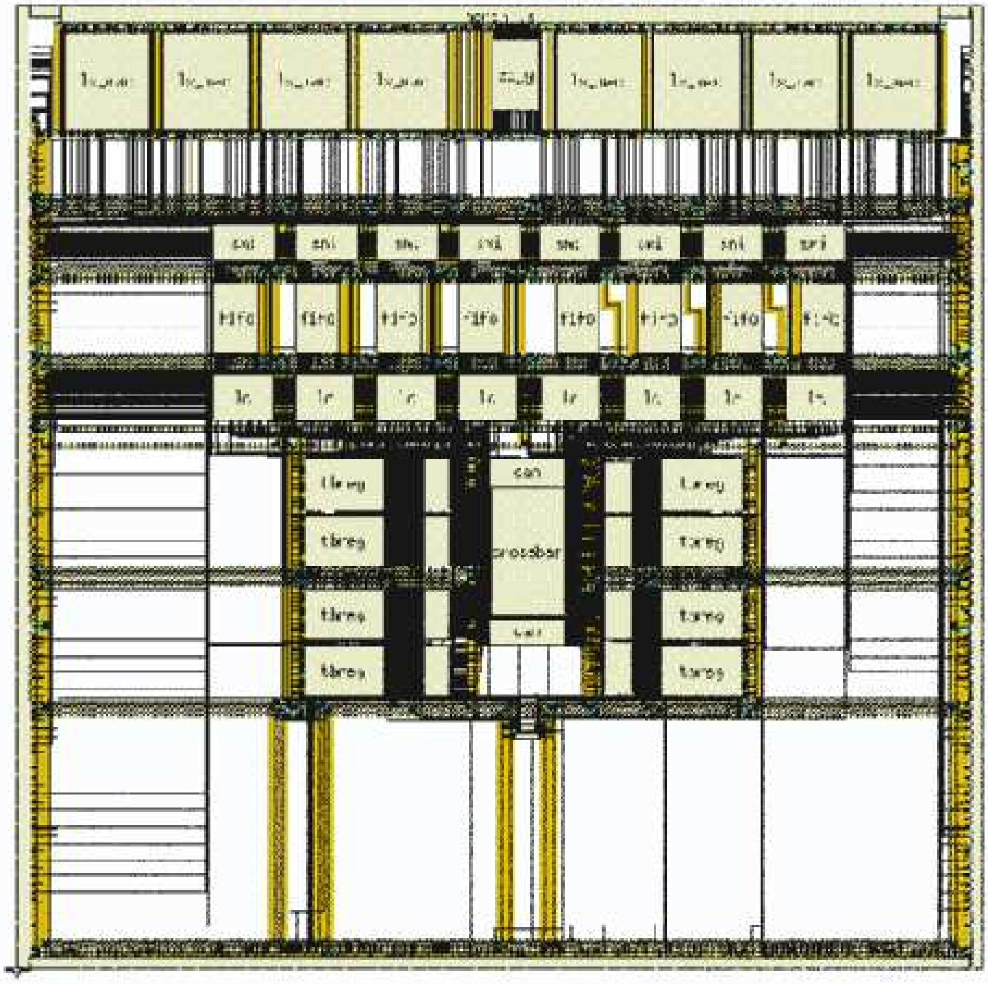


Figure 7.1: Plan de masse du routeur RCUBE

7.2 Analyse temporelle

Le but de l'analyse temporelle est de déterminer où se situent la chaîne critique et d'évaluer la fréquence de fonctionnement maximale que peut atteindre le routeur RCUBE. Nous allons donc évaluer tout d'abord le déphasage du signal d'horloge sur son réseau de distribution, puis nous analyserons les différentes phases de routage décrites au chapitre précédent. Pour chacune, nous expliciterons quelle est la chaîne critique et nous donnerons une estimation de sa valeur.

Cette analyse a aussi pour but de déterminer si la chaîne critique est fortement dépendante de la valence du routeur. Pour chacun des cycles de routage, nous indiquerons la dépendance éventuelle de la chaîne critique vis à vis de la valence du routeur.

7.2.1 La distribution d'horloge

La distribution de l'horloge dans un circuit rapide est toujours un point critique qu'il faut aborder avec soin. En effet, il est important de minimiser le déphasage du signal d'horloge sur le réseau de distribution car cela limite d'une part la fréquence maximale de fonctionnement et d'autre part en cas de déphasage sévère, cela peut conduire à un dysfonctionnement total du circuit.

Distribution interne à un bloc de cellules précaractérisées

Dans la mesure où nous ne disposions pas d'outil de synthèse d'arbre d'horloge permettant de générer au sein d'un bloc de cellules précaractérisées une distribution d'horloge équilibrée (*Clock Tree Synthesis*), nous avons intégré dans chaque bloc de cellules précaractérisées une macrocellule d'amplification d'horloge.

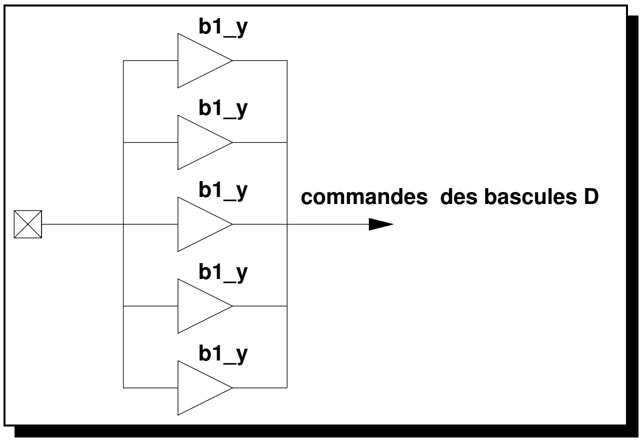


Figure 7.2: Distribution d'horloge reconvergente

Afin de minimiser le déphasage du signal d'horloge entre les blocs de cellules précaractérisées, nous avons choisi de mettre autant d'amplificateurs en parallèle que nécessaire (Cf. figure7.2) pour obtenir un délai identique (ou du moins aussi proche que possible) dans tous les blocs. Nous avons donc effectué le placement/routage de chaque bloc sans amplificateur d'horloge puis nous avons extrait la capacité totale de l'équipotentielle d'horloge (routage + capacité des commandes des bascules). Préalablement nous avions évalué grâce à des simulations SPICE, quelle était la capacité que pouvait charger en 1 ns, un amplificateur b1_y de notre bibliothèque de cellules SCLIB (temps équivalent au *skew* mesuré sur la commande d'horloge de lecture des registres de sortie de la FIFO asynchrone). Le nombre d'amplificateurs obtenu était adapté pour tenir compte de l'ajout de la capacité de drain et de source des amplifcateurs connectés en parallèle.

Distribution d'horloge au niveau plan de masse

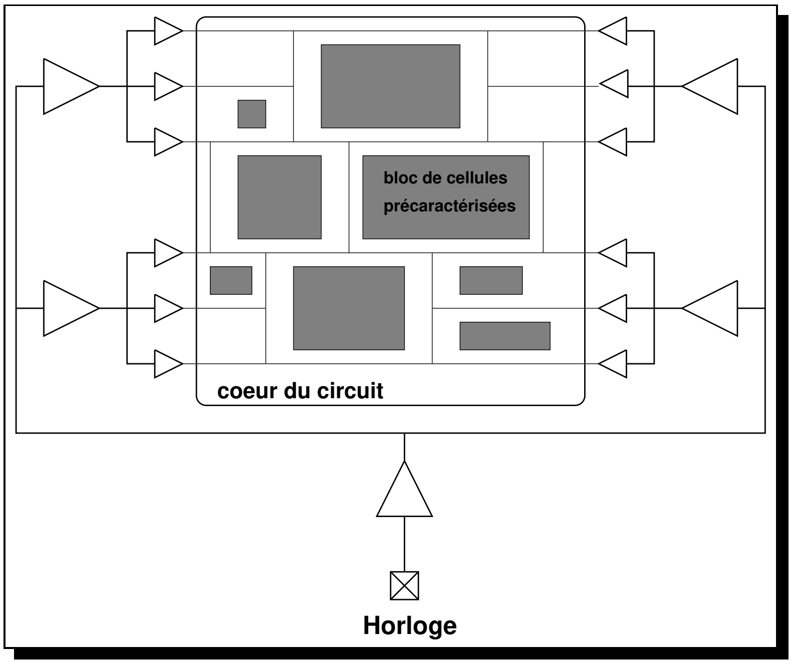


Figure 7.3: Principe de la distribution d'horloge au niveau plan de masse

Afin d'homogénéiser la distribution d'horloge au niveau plan de masse, nous avons adopté la solution du maillage. L'horloge fonctionnelle est d'abord amplifiée par le plot d'entrée puis distribuée à un second étage d'amplification placé dans la couronne de plots qui lui même alimente un dernier étage d'amplification dont les éléments sont disposés dans les plots d'alimentation de façon régulière sur chacune des faces Est et Ouest du circuit (la face Nord contenant les liaisons série est traitée de façon différente). Chaque face Est et Ouest dispose donc de 16 amplificateurs capables ensembles de charger une capacité totale de 90 pF en 1 ns. La capacité de notre réseau de distribution est de 70pF.

Évaluation du déphasage du signal d'horloge sur le réseau maillé

L'évaluation du déphasage du signal d'horloge sur le réseau maillé de distribution (Cf.figure 7.4) a été faite en extrayant une description RC au format SPICE du maillage. À partir de cette vue extraite nous avons construit une *netlist* SPICE dans laquelle étaient instanciés le maillage d'horloge ainsi que les amplificateurs d'horloge. Ceci permettait de tenir compte de leur capacité drain/source en sortie. Nous avons appliqué une impulsion de durée 5 ns et de temps de montée et de descente de 1 ns. Les résultats de la simulation sont visibles sur les figures 7.5 et 7.6.

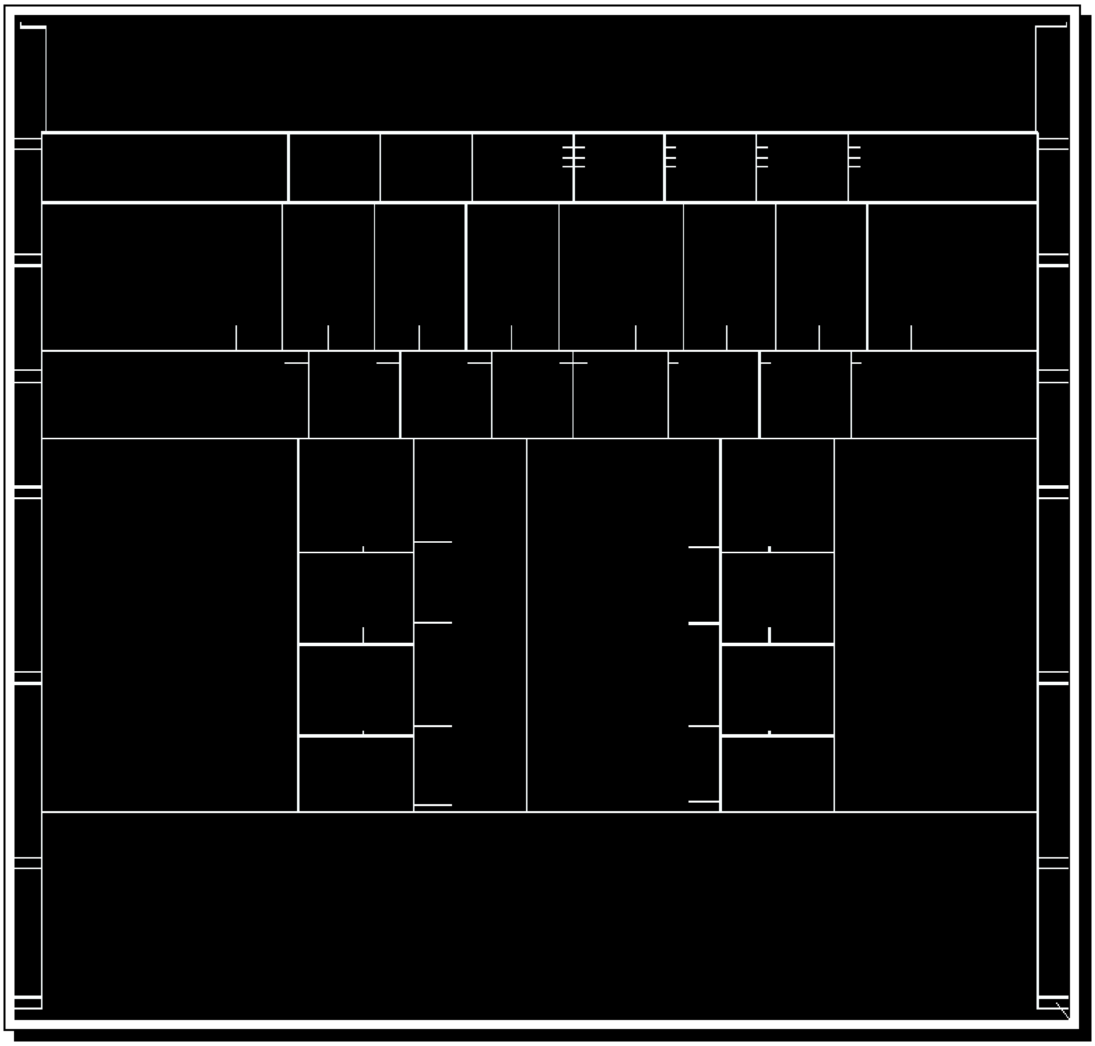


Figure 7.4: Maillage d'horloge au niveau plan de masse

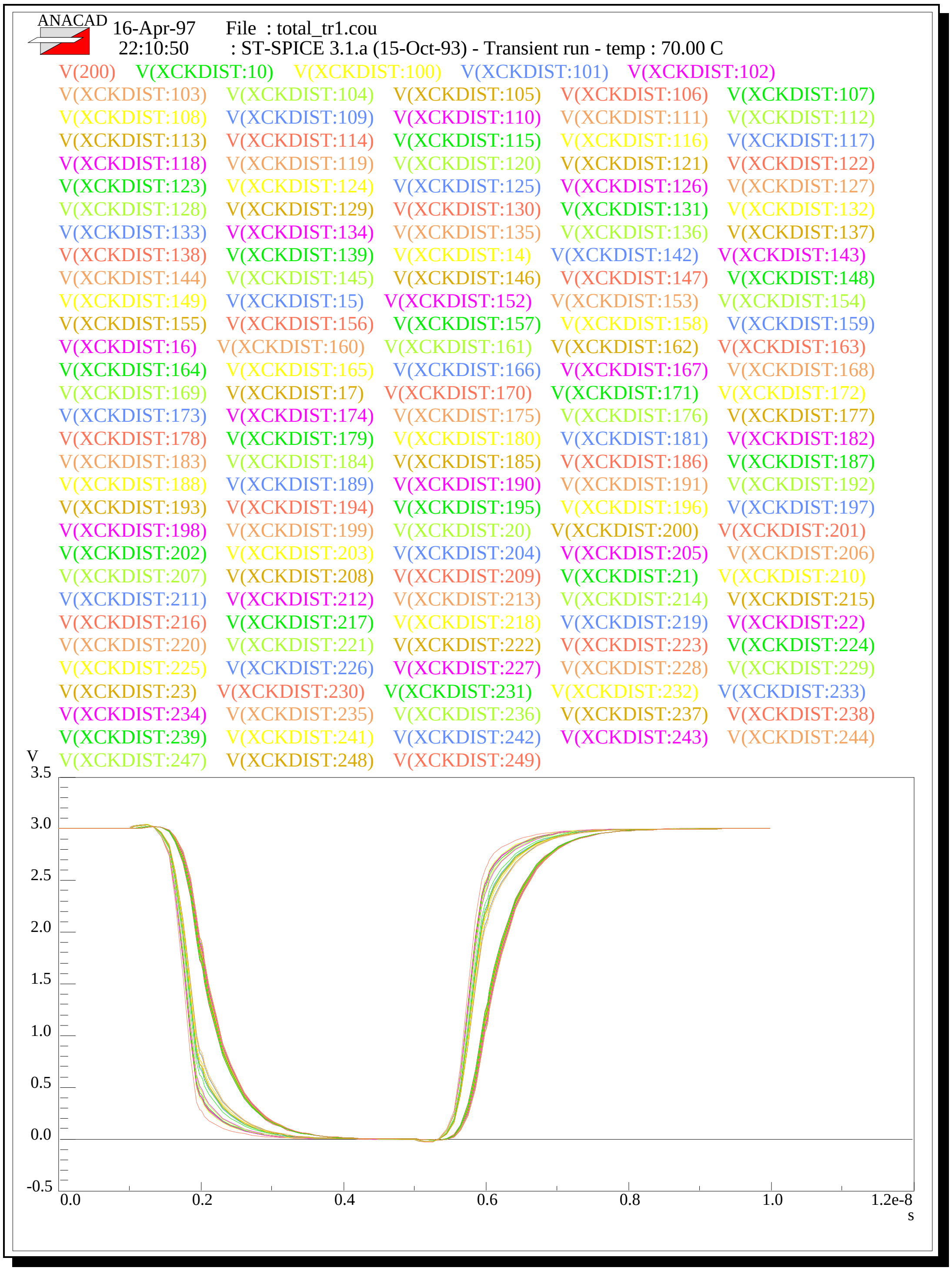


Figure 7.5: Répartition du déphasage du signal d'horloge sur le réseau maillé

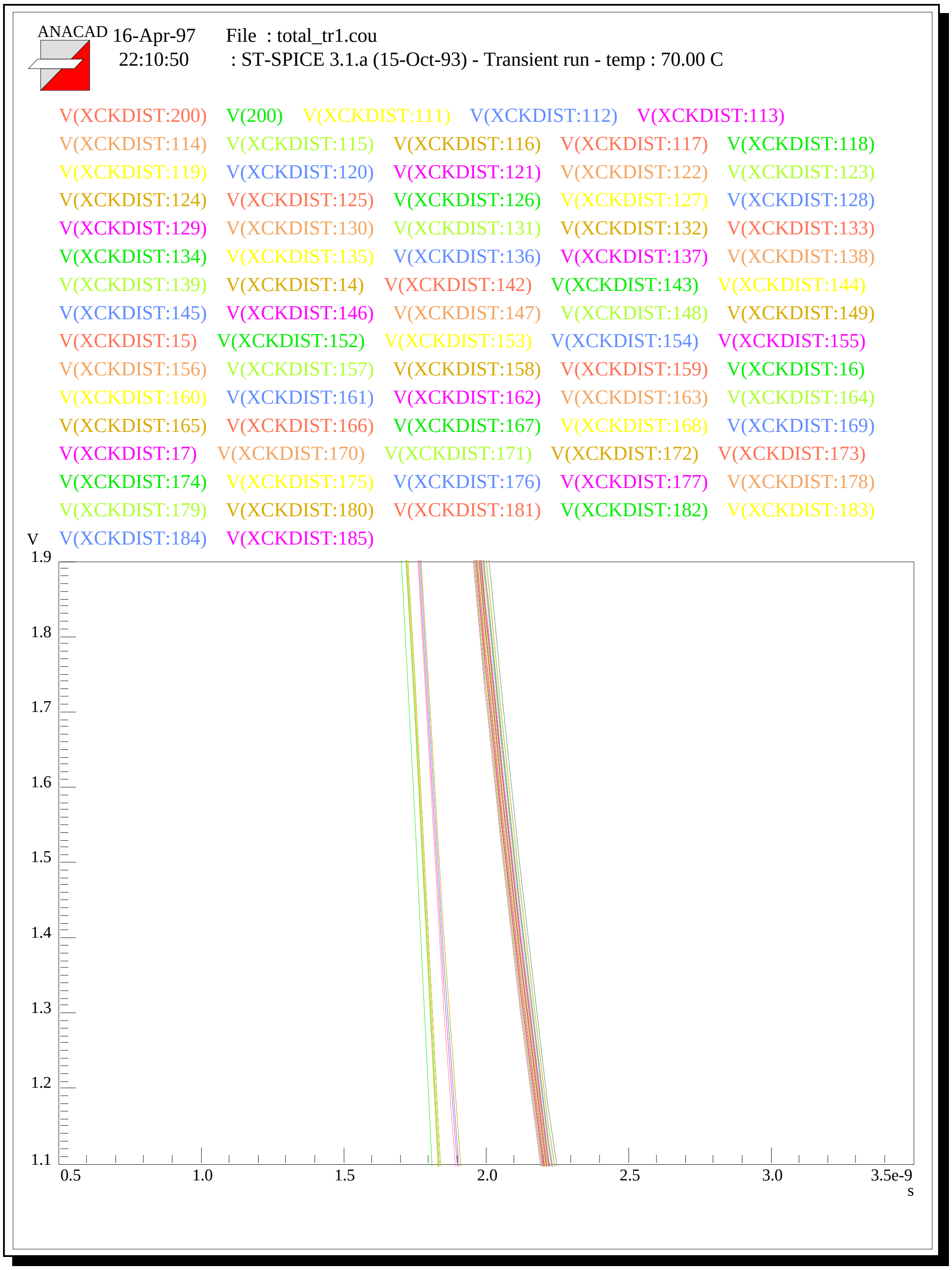


Figure 7.6: Agrandissement de la figure précédente

La valeur obtenue est d'environ 0,5 ns sur l'ensemble du réseau maillé de distribution d'horloge.

7.2.2 Détermination de la chaîne critique

Nous avons effectué une analyse temporelle statique en utilisant l'outil TAS développé au laboratoire MASI. Cet outil se base sur une vue en transistors du circuit obtenue après extraction de la vue physique du circuit placé-routé, afin de prendre en compte les informations de capacités d'interconnexion. Nous allons reprendre chacun des cycles du routage d'un paquet et expliciter pour chacun le coût temporel qu'il induit. À partir de ces informations, nous pourrons discuter les points critiques et éventuellement indiquer comment les améliorer. Cette analyse a été faite en considérant une technologie de fabrication CMOS 0.5 μm, une qualité de process de fabrication pire cas, une température de 70 degrés Celsius et une tension d'alimentation de 3 Volts.

7.2.3 Cycle 1: décodage d'un caractère

Les données issues du désérialiseur sont décodées et filtrées. Les deux opérateurs fonctionnent en parallèle et ont à peu de choses près le même temps de traversée, soit 9 ns. Ce cycle est complètement indépendant de la valence du routeur.

7.2.4 Cycle 2: resynchronisation des caractères reçus

Des simulations électriques ont montré que la FIFO pouvait atteindre une fréquence de fonctionnement de 80 MHz en technologie $0.5\mu m$. La resynchronisation impose donc dans le pire cas un temps de cycle de 12 ns.

7.2.5 Cycles 3 et 4: lecture dans la FIFO d'entrée et comparaison de l'en-tête aux bornes d'intervalle

La FIFO d'entrée impose d'effectuer une opération de lecture pour savoir si le mot récupéré est valide. Ce signal de validité (NEMPTY) est utilisé par l'automate de gestion du port pour commander le multiplexeur permettant de sélectionner l'octet de poids fort de la borne d'intervalle. Suite à cette sélection, l'octet de poids fort de l'en-tête est comparé simultanément aux octets de poids fort des bornes d'intervalles. Au cycle suivant, c'est au tour de l'octet de poids faible d'être comparé. Dans les deux cas, la chaîne critique est identique et a une valeur de 14 ns.

Cette chaîne se décompose comme suit:

- FIFO d'entrée: signal NEMPTY,
- automate de gestion du port: signal MSBSELECT contrôlant le multiplexeur entre l'octet de poids fort et l'octet de poids faible d'une borne d'intervalle,
- comparateur d'en-tête,
- bascule stockant le résultat de la comparaison.

L'opération de comparaison est effectuée dans le bloc TBREG. Constitué de cellules précaractérisées, il contient les tables de routage et les comparateurs. Le placement des registres des tables de routage a été imposé de façon à obtenir une structure régulière aux côtés de laquelle se situent les comparateurs synthétisés. Une approche chemin de données avait d'abord été envisagée mais les résultats en surface et en temps ont montré l'avantage d'une approche en cellules précaractérisées.

7.2.6 Cycle 5: génération d'une requête de sortie

La génération d'une requête de sortie dépend du résultat de la comparaison de l'en-tête et de l'état d'occupation des ports de sorties. Le but est ici d'effectuer le choix d'une sortie le plus rapidement possible d'après les informations données par l'utilisateur (sorties permises, priorité) et l'état de disponibilité des sorties.

La chaîne longue dans ce cycle est située entre les registres d'allocation qui indiquent si une sortie est disponible ou non (signal BUSY) et le registre de requête. Cette chaîne est de l'ordre de 14 ns. Elle se décompose en:

- registres d'allocation vers le signal BUSY: 3 ns,
- traversée du crossbar: 5 ns,
- traversée du filtre et de l'arbre d'arbitrage: 6 ns.

Cette chaîne de 14 ns dépend de la valence du routeur principalement à travers l'arbre d'arbitrage qui comporte un étage de plus dans un routeur de valence 16.

La chaîne longue suivante, située entre les registres de comparaison et le registre des requêtes est de 12 ns.

Grâce à l'arbre d'arbitrage que nous avons mis en œuvre fait que l'adaptativité ne pénalise pas la fréquence de fonctionnement par rapport aux autres cycles de la phase de routage d'un paquet. Cependant, pour des valences supérieures, l'ajout d'un ou plusieurs étages à l'arbre d'arbitrage placera la chaîne critique du routeur dans ce cycle.

7.2.7 Cycle 6: allocation d'une sortie

La requête émise par un port d'entrée vers l'allocateur de la sortie demandée traverse le crossbar (à travers des transparences dédiées) puis un arbre d'arbitrage qui permet de régler les conflits entre plusieurs demandes simultanées émanant de plusieurs ports d'entrée. Cette opération est effectuée en 9 ns. Nous voyons ici que la résolution des conflits est extrêmement rapide.

Si l'on envisage de réaliser un routeur de valence 16, cette chaîne sera allongée puisque l'arbre d'arbitrage sera constitué d'un étage supplémentaire. Le coût supplémentaire introduit ne sera cependant, que de l'ordre de 2-3 ns au maximum. Ce cycle n'est donc pas critique.

7.2.8 Cycle 7: transfert des données de l'entrée vers la sortie

Le transfert des données de la FIFO d'entrée vers la FIFO de sortie intervient après que l'allocation du port de sortie est effectuée. La commande d'écriture en FIFO de sortie est générée par l'automate de gestion du port qui reçoit l'information d'allocation effectuée issue de l'allocateur de sortie allouée. Comme nous le mentionnions dans le chapitre 5, la principale difficulté dans la conception de cet automate a été de tenir la fréquence de 80 MHz tout en conservant un nombre de cycle de routage aussi faible que possible. Un automate de type Mealy plus compact permet sur le papier de réduire le nombre de cycle de routage. Par contre, ce choix a entrainé l'apparition de chaînes longues notamment dans ce cycle où l'allocation d'une sortie commande simultanément l'établissement de la connexion à travers le crossbar et la génération de la commande d'écriture dans la fifo de sortie. Les performances temporelles de notre crossbar ont permis de maintenir la durée de ce cycle à une valeur compatible avec nos objectifs en fréquence.

Dans ce cycle, la chaîne longue de 13 ns est donc la suivante:

- registres d'allocation du port de sortie alloué qui commandent l'établissement d'une connexion à travers le crossbar entre la FIFO d'entrée et la FIFO de sortie,
- génération de la commande d'écriture par l'automate de gestion du port d'entrée (automate de Mealy),
- pointeurs de la FIFO de sortie.

7.2.9 Cycle 8: codage des données

Dans le paragraphe 5.3.2 relatif au code DC-balanced, nous avons évoqué le possible impact du codage dans la position de la chaîne critique. Ceci se confirme. Le temps de transfert entre la FIFO de sortie et le registre situé après l'encodeur 9B/12B est 16 ns. Ce temps est dû à la complexité de l'encodeur lui-même qui de surcroit, a été réalisé en cellules précaractérisées. Il serait intéressant d'évaluer le gain d'une approche "sur-mesure" en réalisant cet encodeur sous la forme d'une macrocellulle.

L'autre raison qui conduit à cette valeur de 16 ns est la structure de la FIFO de sortie et notamment à la manière dont est sélectionné le mot à émettre grâce aux registres pointeurs. Six couches logiques séparent la sortie des registres contenant le mot à émettre et l'entrée de l'encodeur.

7.2.10 Résumé et conclusion

Cycle	Opération	durée	dép. valence
1	Décodage d'un caractère	9 ns	non
2	Resynchronisation du caractère reçu	12 ns	non
3	Lecture dans la FIFO et comparaison de l'octet de poids fort de l'en-tête	14 ns	non
4	Comparaison de l'octet de poids faible, détermination de l'intervalle auquel appartient l'en-tête	14 ns	non
5	Génération d'une requête de sortie avec mise en œuvre de l'adaptativité	14 ns	oui
6	Allocation d'une sortie avec gestion des conflits entre requêtes	9 ns	oui
7	Transfert du caractère de la FIFO d'entrée vers la FIFO de sortie allouée	13 ns	non
8	Codage du caractère	16 ns	non

Table 7.1: Durée estimée et dépendance vis à vis de la valence du routeur, des cycles de routage de l'en-tête d'un paquet

L'analyse temporelle montre que la chaîne critique du routeur RCUBE d'une valeur de 16 ns se situe dans le cycle d'encodage 9B/12B (cycle 8), ce qui conduit à un fréquence de fonctionnement maximale d'environ 60 MHz. Cette valeur devrait pouvoir être améliorée en revoyant la structure de la FIFO de sortie et en adoptant une approche de conception sur-mesure pour l'encodeur 9B/12B. Par contre, cette chaîne est indépendante de la valence du routeur. Au contraire, la chaîne suivante située dans le cycle de génération d'une requête de sortie avec mise en œuvre de l'adaptativité (cycle 5) est dépendante de la valeur du routeur. Elle constitue la chaîne critique de l'architecture du routeur RCUBE.

7.3 Test et mise en carte du routeur RCUBE

7.3.1 Test fonctionnel sur testeur

Afin de tester fonctionnellement les puces sur tranches de retour de fabrication, nous avons générer un ensemble de fichiers de test destinés à vérifier:

- le chargement des registres de configuration,
- l'accès en lecture et en écriture des ports parallèles,
- le contrôle de flux sur le port parallèle,
- la logique de routage en utilisant le mode rebouclage parallèle.

• les valeurs initiales des registres par décalage du chemin de test.

Cette première phase ne testait en fait que R3P mais avait pour but d'éliminer le maximum de puces défectueuses avant l'encapsulation. Le fonctionnement des macrocellules HSL et des blocs SMI ne pouvaient être vérifié durant cette phase, du fait de l'impossibilité d'amener un signal d'horloge de fréquence suffisante sur la tranche.

Une seconde série de test a donc été menée sur des puces mises en boîtier. Cette fois, il était possible de fournir au circuit une horloge de fréquence suffisante (50 MHz) pour pouvoir réaliser une calibration des liaisons HSL placées en mode rebouclage série interne. Ce mode de rebouclage, contrôlable par des broches présentes sur l'interface du circuit, permet de connecter de façon interne le signal série émis par le sérialiseur d'une macrocellule HSL sur son désérialiseur. Des scénarii des tests identiques à ceux effectués en mode rebouclage parallèle pouvaient donc être mis en œuvre. Ces tests avaient pour but de valider l'automate d'initialisation des liaisons HSL, les encodeurs 9B/12B et les décodeurs 12B/9B, la resynchronisation des caractères de contrôle de flux ainsi que la logique d'écriture dans la FIFO d'entrée.

À l'issue de ces deux séries de test, nous disposions de routeurs RCUBE qui pouvaient être mis en carte pour subir une évaluation dans un contexte d'utilisation réelle.

7.3.2 Mise en carte

Afin de pousser plus avant la validation fonctionnelle du routeur RCUBE et d'évaluer ses performances, J.J. Lecler a développé une carte d'évaluation permettant de connecter deux PCs disposant de cartes mères à bus PCI via des routeurs RCUBE (Cf. figure 7.7).

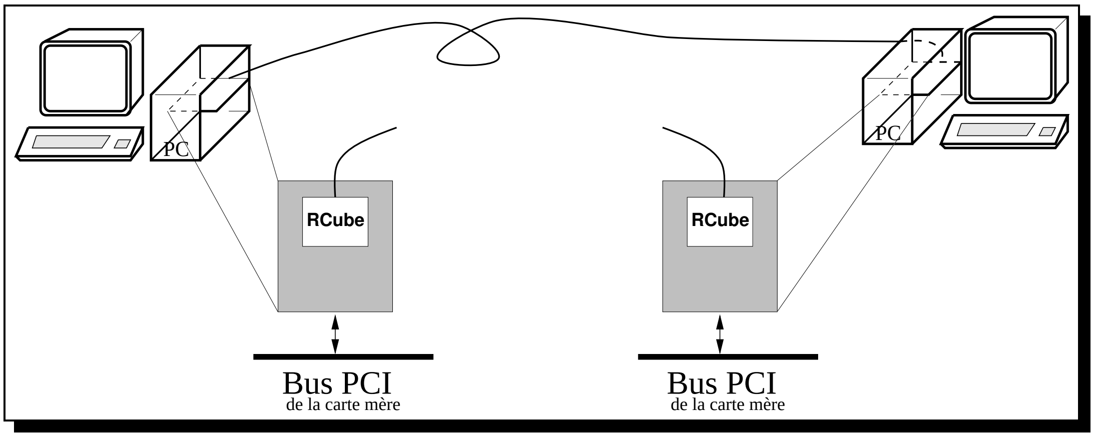


Figure 7.7: Connexion de deux PCs intégrant des cartes à base de routeur RCUBE

De notre point de vue, la carte réalisée (Cf. figure 7.8) possède 2 fonctions:

- valider le fonctionnement du routeur RCUBE,
- expérimenter la mise en carte d'un circuit disposant de broches émettant des signaux à une fréquence de 1 GHz,

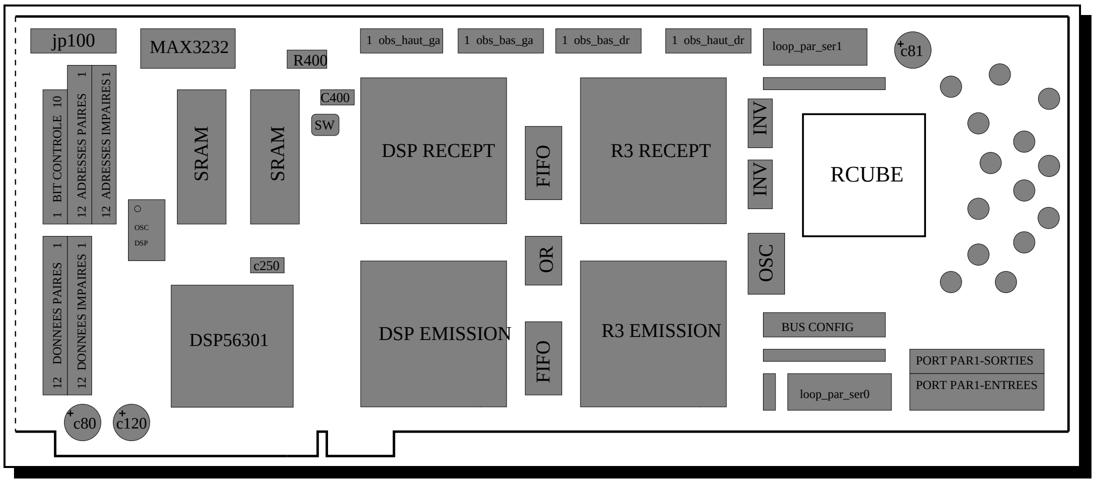


Figure 7.8: Carte d'évaluation du routeur RCUBE

Cette carte contient principalement le routeur RCUBE, un processeur de traitement de signal MOTOROLA DSP56301 disposant d'une interface PCI, et des composants programmables ALTERA effectuant l'interface entre le DSP56301 et le routeur RCUBE.

Nos tests ont consisté à effectuer des transferts de données d'un PC à l'autre sous la forme d'une recopie d'écran en mode graphique.

Ces tests ont montré que le routeur RCUBE était opérationnel à 66 MHZ. Nous avons vérifié l'autocalibration des liens HSL en déconnectant en cours de transfert le câble coaxial puis en le rebranchant. Après la phase d'autocalibration attendue, la transmission a repris.

Afin de mesurer le taux d'erreur, nous avons laissé le système en fonctionnement pendant une semaine. Nous n'avons pas constaté d'erreur.

Ces premiers résultats sont donc prometteurs. Cependant, des tests plus poussés restent à effectuer afin de déterminer sa fréquence maximale de fonctionnement. Faute de moyens matériels suffisants, nous n'avons pu les faire. C'est pourquoi nous attendons beaucoup des différents partenaires industriels qui eux aussi ont développé des cartes d'évaluation destinées à recevoir le routeur RCUBE.

7.4 Conclusion

Nous avons présenté les résultats de notre étude en terme de surface occupée par les blocs et d'estimation des performances temporelles.

Le routeur RCUBE occupe une surface utile de 94 mm^2. La surface totale de la puce est de 213 mm^2, car la couronne de plots nous a été imposée pour permettre la réutilisation d'un bloc dur contenant les huit macrocellulles HSL et le bloc d'adaptation d'impédance ZC. Une analyse de la dépendance de la surface des blocs du circuit vis à vis de la valence du routeur a montré qu'un routeur de valence 16 peut être intégré sur une puce de même taille et que les modifications requises sont mineures.

La chaîne critique du routeur RCUBE d'une valeur de 16 ns se situe dans le cycle d'encodage 9B/12B, ce qui conduit à un fréquence de fonctionnement de 60 MHz. Cependant, cette valeur estimée en considérant un process de fabrication pire cas permet d'espérer que des fréquences de fonctionnement supérieures peuvent être atteintes. Cette chaîne est indépendante de la valence du routeur. Par contre, la chaîne suivante située dans le cycle de génération d'une requête de sortie avec mise en œuvre de l'adaptativité est dépendante de la valeur du routeur. C'est elle qui constitue la chaîne critique de l'architecture du routeur RCUBE.

Le routeur RCUBE a été fabriqué en technologie CMOS 0.5 μm-3.3 V et encapsulé dans un boitier QFP à 208 broches. Une carte d'évaluation à interface PCI a été réalisée afin de le tester dans un contexte d'utilisation réelle. Les tests menés montrent que le routeur RCUBE est opérationnel à 66 Mhz.

CHAPITRE 8

Conclusion

Le but de notre étude était de concevoir et réaliser un routeur de paquets pouvant servir de brique de base à la construction de réseaux d'interconnexion extensibles et à hautes performances pour des systèmes massivement parallèles. À partir de l'analyse de plusieurs systèmes développés par nos partenaires industriels et par nous-même, nous avons arrêté les caractéristiques que devaient posséder le routeur répondant aux besoins en matière de réseaux d'interconnexion. Ces caractéristiques sont rappelées ci-après :

- faible latence,
- haut débit,
- topologie quelconque,
- résistance à la congestion,
- tolérance aux pannes,
- pas de pertes de données,
- extensibilité aisée.

L'intégration des liaisons série HSL nous a permis d'offrir un haut débit de 1 Gbit/s par lien et par direction et d'atteindre ainsi un débit global de 640 Moctet/s à une fréquence de 80 MHz. L'utilisation des liaisons HSL, associée au choix du routage par intervalle nous permet d'offrir une extensibilité aisée et de supporter des topologies de réseaux quelconques. Un mécanisme de destruction de l'en-tête à l'émission autorise la construction de réseaux de taille virtuellement illimitée.

Afin d'offrir une résistance à la congestion et un certain degré de tolérance aux pannes, nous avons mis en œuvre un mécanisme de routage adaptatif qui permet à l'utilisateur de spécifier un ordre dans le choix des sorties à emprunter. Ce mécanisme dont le principe repose sur un arbre d'arbitrage est simple et ne coûte qu'un cycle. La mise en œuvre d'un double contrôle de flux externe et interne assure une transmission des données sans perte en l'absence de pannes. Le contrôle de flux basé sur le concept de crédit d'émission ne consomme que 3% de la bande passante du réseau. La latence de routage adaptatif d'un paquet dont l'en-tête est codée sur deux octets est de 13 cycles soit 150 ns à 80 MHz, dont 7 dûs à l'utilisation des liaisons série.

L'accès au routeur est facilité par la présence de deux ports parallèles qui permettent d'une part de le connecter facilement à un processeur et de rendre possible le test de fabrication.

Pour évaluer la faisabilité de routeurs de valence supérieure à 8, nous avons mis en œuvre une approche modulaire. Nous avons donc centralisé dans un même bloc SMI, les fonctions relatives aux liaisons série HSL, à savoir l'autocalibration, l'encodage 9B/12B et la resynchronisation des caractères de contrôle de flux. Nous avons développé un générateur de crossbar qui nous permet de générer des crossbars de valence quelconque. Les blocs dont la taille dépend de la valence ont été obtenus par synthèse. L'analyse de la surface des blocs vis à vis de la valence montre qu'on peut réaliser un routeur de valence 16 en conservant la même taille de puce. L'analyse temporelle a montré que la chaîne critique du routeur RCUBE se situait dans le cycle d'encodage 9B/12B des données et conduisait à une fréquence maximale de fonctionnement de 60 MHz. Cependant, cette chaîne longue n'est pas dépendante de la valence du routeur. La chaîne critique de l'architecture de RCUBE se situe dans le cycle d'émission d'une requête de sortie avec mise en œuvre de l'adaptativité. Le passage à une valence de 16 n'ajoute qu'un étage à l'arbre d'arbitrage. Ceci nous permet d'être optimiste quant aux performances temporelles d'un routeur de valence 16.

Finalement, cette étude a abouti à la fabrication du routeur RCUBE en technologie CMOS $0.5\mu\text{m} - 3.3\text{V}$. Encapsulé dans un boitier QFP à 208 broches , il contient 465000 transistors, et est opérationnel à 66 MHz sur une carte d'évaluation.

Nos objectifs sont donc atteints.

Critiques et perspectives

Dans un futur proche, une carte issu du prototype d'évaluation sera disponible. À court terme, il est prévu de réaliser un nouveau circuit intégrant le routeur RCUBE et un contrôleur de bus PCI. Cela sera l'occasion d'améliorer certains points de la réalisation du routeur, et de faire évoluer son architecture et ses fonctionnalités. En effet, on peut faire plusieurs critiques. Premièrement, les tables de routage ne sont pas initialisées au démarrage du circuit. Dans un système intégrant plusieurs de ces routeurs, il est nécessaire de concevoir un réseau de contrôle séparé pour configurer les routeurs car ceux-ci sont incapables d'échanger des informations tant que leurs tables ne sont pas chargées. Deuxièmement, il n'y a pas de support matériel pour la diffusion de messages. Troisièmement, la stratégie de mémorisation en entrée n'est pas optimale du point de vue du seuil de saturation maximal supporté. L'ajout d'un tampon central comme dans le routeur VULCAN apporterait une solution dont le coût serait raisonnable. Enfin, du fait de la fiabilité des liaisons série, RCUBE ne comporte aucun mécanisme de reprise en cas d'erreur.

Tous ces points sont donc sujets à des études plus approfondies. Néanmoins, si nous sommes à même de faire ces critiques, c'est que que avons su mener la conception du routeur RCUBE jusqu'à sa réalisation et sa mise en œuvre sur carte. Cet objectif premier nous a obligé à faire des compromis entre les performances théoriques des solutions choisies, les fonctionnalités que l'on aurait aimer intégrer et la complexité matérielle qu'elles introduisaient au regard des moyens disponibles.

Bibliographie

[1] C. CLOS. A study of non-blocking switching networks. In *Bell System Technical Journal*, volume 32, pages 406–424, Mars 1953.

[2] J. BEETEM, M. DENNEAU, and D. WEINGARTEN. The gf11 supercomputer. In *Proceedings of the 12th Annual International Symposium on Computer Architecture*, pages 108–115, 1985.

[3] F. POTTER. Conception et réalisation d'un réseau d'interconnexion à faible latence et haut débit pour machines multiprocesseurs. In *Thèse de Doctorat de l'Université de Paris 6*, 1996.

[4] F. WAJSBÜRT. Spécifications du composant PCIDDC. 1997.

[5] L. M. NI and P. K. McKINLEY. A survey of wormhole routing techniques in direct networks. In *COMPUTER*, pages 62–76, Février 1993.

[6] A. COFFLER, M. COMBES, J.C. LEBIHAN, R. MARBOT, and R. NESAMZADEH. Integration of multiple bidirectional point-to-point serial links in the gigabits per second range. In *Hot Interconnects Symposium*, 1993.

[7] P. KERMANI and L. KLEINROCK. Virtual-cut-through: A new computer communication switching technique. In *Computer Networks*, volume 3, pages 267–286, Septembre 1979.

[8] W. J. DALLY and C. L. SEITZ. The torus routing chip. In *Journal of Distributed Computing*, volume 1, pages 187–196, Avril 1986.

[9] N. SANTORO and R. KHATIB. Labeling and implicit routing in networks. In *The Computer Journal*, volume 28, pages 5–8, 1985.

[10] J. VAN LEEUWEN and R. B. TAN. Interval routing. In *The Computer Journal*, volume 30, pages 298–307, 1987.

[11] B. ZERROUK, J. M. BLIN, and A. GREINER. Encapsulating networks and routing. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 546–553, Avril 1994.

[12] M. GERLA and L. KLEINROCK. Flow control: A comparative survey. In *IEEE Trans. on Communications*, volume COM-28, page 553, Avril 1980.

[13] Y. TAMIR and G. L. FRAZIER. Dynamically-allocated multi-queue buffers for vlsi communications switches. In *IEEE Trans. on Computers*, volume 41, pages 725–737, Juin 1992.

[14] M. I. IRLAND. Buffer management in a packet switch. In *IEEE transactions on Communications*, volume COM-26, pages 328–337, Mars 1978.

[15] R. M. FUJIMOTO. Vlsi communication components for multicomputer networks. In *CS Division Report UCB/CSD 83/136, Univ. of California*, 1983.

[16] D. A. REED and R. M. FUJIMOTO. Multicomputer networks: Message-based parallel processing. In *Cambridge, MA: MIT Press*, 1987.

[17] M. J. KAROL, M. G. HLUCHYJ, and S. P. MORGAN. Input vs. output queueing on a space-division packet switch. In *Proceedings of the IEEE Global Telecommunications Conference, Houston,TX*, pages 659–665, Décembre 1986.

[18] M. KUMAR and J. R. Jump. Performance enhancement in buffered delta networks using crossbar switches and multiple links. In *Journal of Parallel Distributed Computing*, volume 1, No. 1, pages 81–103, 1984.

[19] R. J. McMILLEN and J. SIEGEL. The hybrid cube network. In *Proceedings of Distributed Data Acquisition*, pages 11–22, Décembre 1980.

[20] E. RAUBOLD and J. HAENLE. A method of deadlock-free resource allocation and flow control in packets networks. In *ICCC'76 Toronto*, page 483, Août 1976.

[21] A. GIESSLER, J. HAENLE, A. KOENIG, and E. PADE. Free buffer allocation - an investigation by simulation. In *Computer Networks*, volume 2, page 191, 1978.

[22] S. TOUEG and J. ULLMAN. Deadlock-free packet-switched networks. In *Proceedings of the 11th ACM Symposium on Theory of Computing*, pages 89–98, 1979.

[23] S. TOUEG. Deadlock- and livelock-free packet-switching networks. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 94–99, 1980.

[24] D. GELERTNER. A dag-based algorithm for prevention of store-and-forward deadlocks in packets networks. In *IEEE transactions on Computing*, volume C-30, pages 709–715, Octobre 1981.

[25] K. D. GUNTHER. Prevention of deadlocks in packet-switched data transport systems. In *IEEE Transactions on Communications*, volume COM-29,No. 4, pages 512–524, Avril 1981.

[26] P. M. MERLIN and P. J. SCHWEITZER. Deadlock avoidance in store-and-forward networks. In *IEEE Transactions on Communications*, volume COM-28,No. 3, pages 345–354, Mars 1981.

[27] I. S. GOPAL. Prevention of store-and-forward deadlock in computer networks. In *IEEE Transactions on Communications*, volume COM-33,No. 12, pages 1258–1264, Décembre 1985.

[28] W. J. DALLY and C. L. SEITZ. Deadlock-free message routing in multiprocessor interconnection networks. In *IEEE Transactions on Computers*, volume C-36, No.5, pages 547–553, Mai 1987.

[29] Craig B. Stunkel, Monty M. Denneau, Ben J. Nathanson, Dennis G. Shea, Peter H. Hochschild, Michael Tsao, B lent Abali, Douglas J. Joseph, and Philip R. Varker. Architecture and implementation of vulcan. In *International Parallel Processing Symposium,Cancun,Mexico*, Avril 1994.

[30] Craig B. Stunkel, Dennis G. Shea, Don G. Grice, Peter H. Hochschild, and Michael Tsao. The sp1 high-performance switch. In *Proceedings of the Scalable High Performance Computing Conference, Knoxville, TN*, pages 150–157, Mai 1994.

[31] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. M. Bender, D. G. Grice, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker. The sp2 high-performance switch. In *IBM SYSTEMS JOURNAL*, volume 34, No 2, pages 185–204, 1995.

[32] A. G. Nowatzyk and M. Parkin. The s3.mp interconnect system and tic chip. 1995.

[33] A. G. Nowatzyk, M. C. Browne, E. J. Kelly, and M. Parkin. S-connect: From networks of workstation to supercomputer performance. In *Proceedings of the 22nd annual International Symposium on Computer Architecture*, Juin 1995.

[34] SGS-THOMSON microelectronics. Stc104, asynchronous packet switch. In *Datasheet*, Avril 1995.

[35] A scheme for fast parallel communication. In *SIAM Journal of Computing*, volume No 11, pages 350–361, 1982.

[36] Lotfi BenAMMAR. Compilation de chemins de données optimisés pour circuits vlsi. In *Thèse de doctorat de l'Université Paris VI*, 1993.

[37] Frédéric PETROT. Outils d'aide au développement de bibliothèques vlsi portables. In *Thèse de doctorat de l'Université Paris VI*, 1994.

[38] M. HALILA. ISL: Étude et réalisation d'un réseau de communication à hautes performances pour machines multiprocesseurs à mémoire distribuée. In *Thèse de doctorat de l'Université Paris VI*, 1996.

[39] J. U. Horstmann, H. W. Eichel, and R. L. Coates. Metastability behavior of cmos asic flip-flops in theory and test. In *IEEE Journal of solid-state circuits*, volume 24, No.1, pages 146–157, Février 1989.

[40] L. Kim and R. W. Dutton. Metastability of cmos latch/flip-flop. In *IEEE Journal of solid-state circuits*, volume 25, No.4, pages 942–951, Août 1990.

Annexes

Ces annexes présentent le fonctionnement de la fifo de sortie, les éléments de resynchronisation dédiés aux caractères de contrôle de flux et au signal de calibration issu des désérialiseurs, ainsi que les data-sheets des composants SMI et FCB.

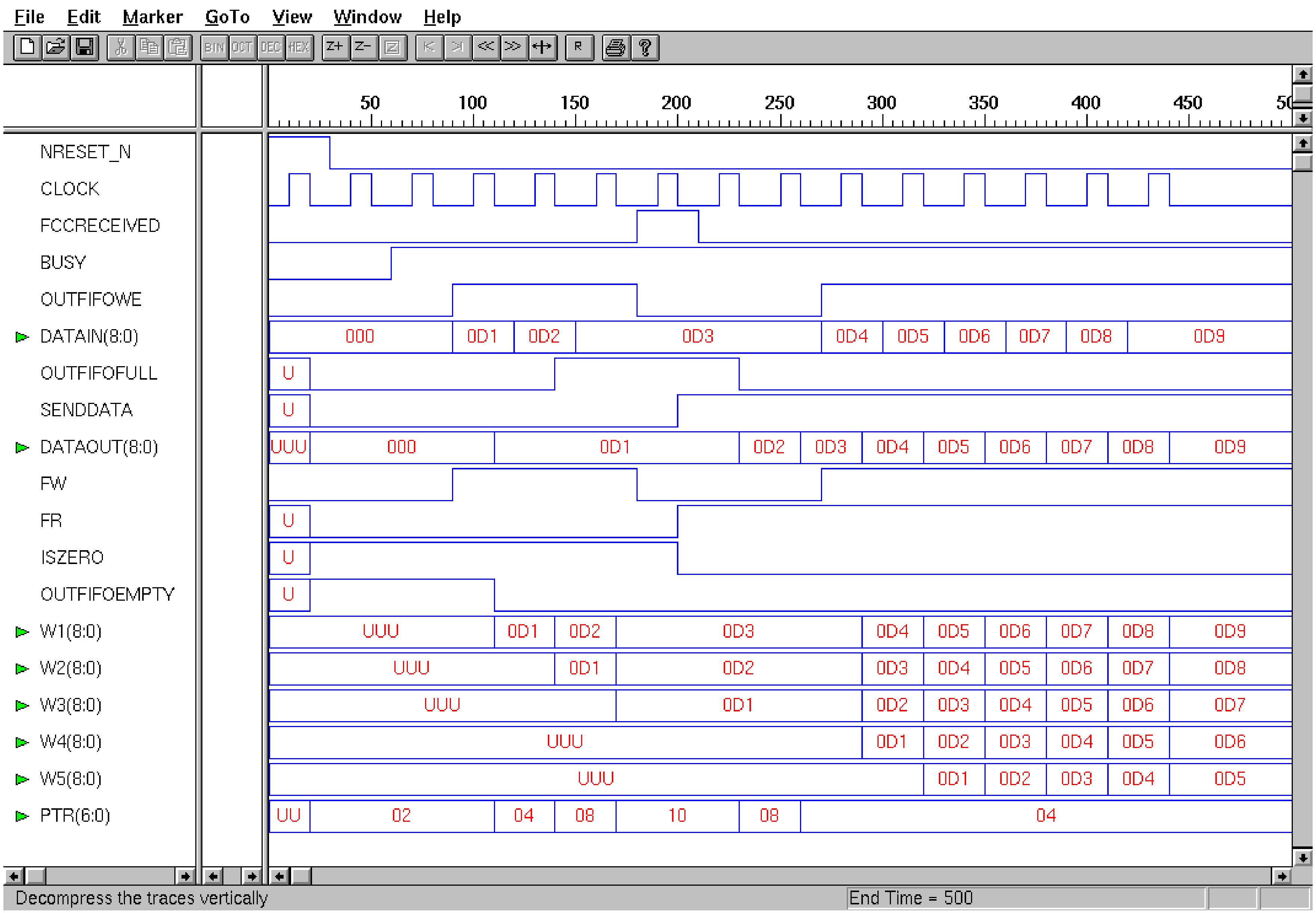
ANNEXE A

La FIFO de sortie

Les chronogrammes de la page suivante illustrent le comportement de cette fifo de sortie. Tout d'abord, cette fifo n'est sollicitée que lorsque le port de sortie auquel elle appartient est alloué à une entrée d'où l'état '1' du signal BUSY. La fifo de sortie étant vide au démarrage (OUTFIFOEMPTY est à'1'), le contrôleur du port peut commander l'écriture d'un mot dans cette fifo. La donnée DATAIN(8:0) est positionnée (valeur D1) et la commande d'écriture OUTFIFOWE activée (état '1'). Le signal OUTFIFOFULL étant toujours à '0', l'écriture peut continuer. Une nouvelle donnée D2 est écrite. Le port de sortie n'ayant pas de crédit d'émission (ISZERO à '1') les données demeurent dans la fifo. D1 est décalé dans le registre W2 tandis que D2 est écrit dans le regitre W1. Le pointeur de lecture PTR(6:0) pointe successivement sur le registre contenant D1 (0000010, 0000100, 0001000, 0010000). L'écriture de D2 a eu pour effet de lever le signal OUTFIFOFULL. Le contrôleur du port réagit à cette information avec un cycle de retard. En conséquence, une donnée D3 est écrite avant que le signal OUTFIFOWE ne soit mis à '0'.

Dès qu'un crédit d'émission est accordé au port de sortie (FCCRECEIVED à '1'), la valeur D1 présente sur le bus de sortie DATAOUT(8:0) est validée par la montée à '1' du signal SENDDATA. Les données écrites dans la fifo sont ensuite envoyées sur le bus de sortie une à une.

Nous avons étendu la profondeur de la fifo à 5 mots (alors que 3 aurait été suffisant) pour pouvoir ne pas tenir compte du signal OUTFIFOFULL lorsque l'on traite les en-têtes des paquets. Cela a permis de simplifier le contrôleur du port. Par contre, il faut être en mesure de supporter le pire cas de figure qui est: la fifo est vide, on reçoit la séquence de paquets suivante: (H01, H02, EP ou EEP), (H11, H12, D1, ...Dn., EP) où Hij est l'octet j de l'en-tête du paquet i et Di les données constituant le corps du paquet. En effet, le signal OUTFIFOFULL ne peut être pris en compte par l'automate de contrôle qu'après avoir commandé l'envoi de EP ou EEP. Mais, dans la mesure où il ne tient pas compte de ce signal lorsqu'il traite H11 et H12, il faut malgré tout pouvoir les stocker dans la fifo de sortie. C'est pourquoi la profondeur de 3 mots initialement prévue a été étendue à 5. L'envoi du EP ou du EEP est quant à lui stoppé. Ce cas de figure est le seul qui puisse conduire à un remplissage total de la fifo. Dans tous les autres cas, 3 mots au plus sont stockés permettant toujours à un en-tête sur deux octets d'être stocké.


File Edit Marker GoTo View Window Help
NRESET_N
CLOCK
FCCRECEIVED
BUSY
OUTFIFOWE
DATAIN(8:0)
OUTFIFOFULL
SENDDATA
DATAOUT(8:0)
FW
FR
ISZERO
OUTFIFOEMPTY
W1(8:0)
W2(8:0)
W3(8:0)
W4(8:0)
W5(8:0)
PTR(6:0)
50 100 150 200 250 300 350 400 450
000 0D1 0D2 0D3 0D4 0D5 0D6 0D7 0D8 0D9
UUU 000 0D1 0D2 0D3 0D4 0D5 0D6 0D7 0D8 0D9
UU 02 04 08 10 08 04
Decompress the traces vertically
End Time = 500

ANNEXE **B**

Les éléments de resynchronisation

Dans le chapitre 5, nous avons justifié la nécessité de disposer d'un opérateur de resynchronisation dédié au caractère de contrôle FCC. Nous avons insisté sur l'importance capitale de la reconnaissance à coup sûr de la réception de ce caractère. Le problème principal auquel sont sensibles les dispositifs de resynchronisation est la métastabilité qui maintient l'élément mémorisant (latch, flip-flop) dans un état indéfini pendant un laps de temps indeterminé. Si cet état dure plus d'un cycle, l'information est perdue. D'autre par, on ne peut présumer de l'état stable que prendra l'élément mémorisant après avoir quitté son état métastable. Il faut donc s'assurer que le dispositif que l'on met en œuvre est extrèmement robuste face à ce risque. Ceci nous a amené à étudier avec attention quels étaient les facteurs déterminant la robustesse d'un élément mémorisant au phénomène de métastabilité.

B.1 Introduction au problème de métastabilité

Pour comprendre au moins intuitivement le problème de la métastabilité, il faut revenir à la structure de base d'un point mémorisant élémentaire , à savoir deux inverseurs rebouclés l'un sur l'autre. Ce système, illustré sur la figure B.1, dispose de trois états d'équilibre dont deux sont stables: Vout=0, Vout=1 et un métastable: Vout=Vin. Du fait du bruit toujours présent, cet état métastable ne peut durer indéfiniment. En effet une très faible variation autour du point d'équilibre métastable fait basculer le système dans l'un de ses états stables. Par contre la durée de l'état métatstable ainsi que l'état stable dans lequel il va tomber sont imprévisibles.

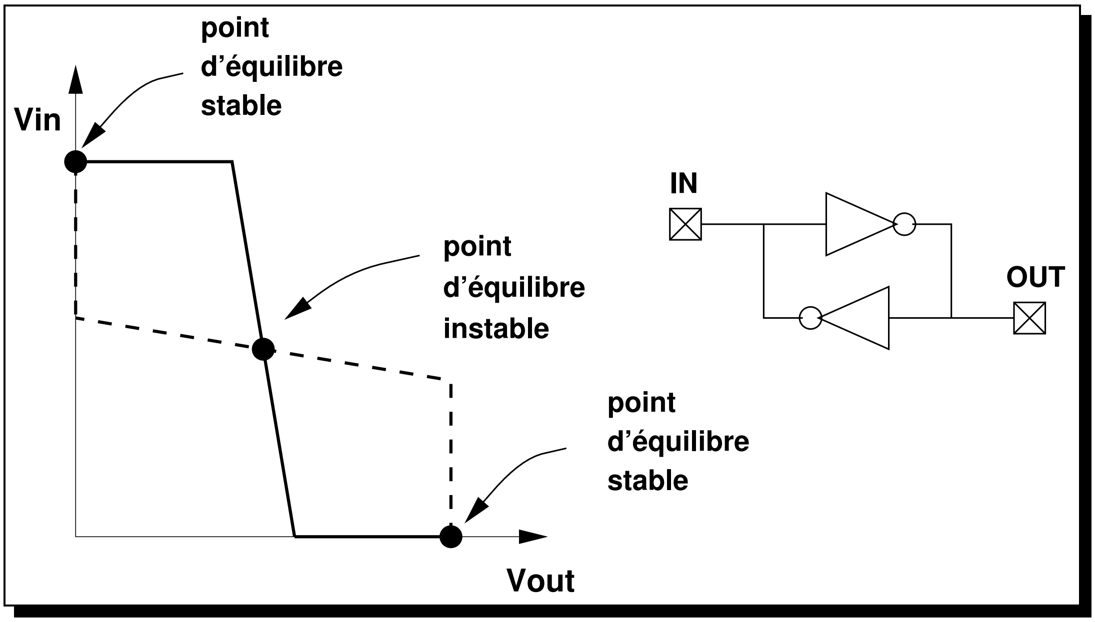


Figure B.1: Structure de principe d'un point mémorisant élémentaire

La capacité d'un élément mémorisant à s'extraire de la zone métastable dépend de la vitesse à laquelle évolue la valeur de la tension du point mémorisant. Cette tension est imposée par l'inverseur de rebouclage et par le dispositif d'entrée qui permet d'imposer une valeur au point mémorisant. L'action du dispositif d'entrée n'étant effective que sur l'état actif de l'horloge, celui-ci a peu d'influence sur la robustesse de l'élément mémorisant. Le facteur dominant est le gain en boucle ouverte des deux inverseurs constituant le point mémorisant[39, 40].

B.2 Étude comparée des latches l1_y, ls_y et lsr_y

Nous avions à notre disposition le latch l1_y de notre bibliothèque de cellules précaractérisées SCLIB. L'étude de ce latch a montré un comportement désastreux vis à vis des risques de métastabilité. En outre, son dispositif d'entrée constitué d'un simple transistor de passage est fortement déconseillé en technologie $0.5\mu\mathrm{m} - 3.3\mathrm{V}$. Ceci nous a conduit à utiliser une nouvelle structure de latch déclinée en deux versions ls_y et lsr_y, ce dernier disposant d'une entrée reset asynchrone. Le schéma de ces latches est décrit sur les figures B.2, B.3 et B.4.

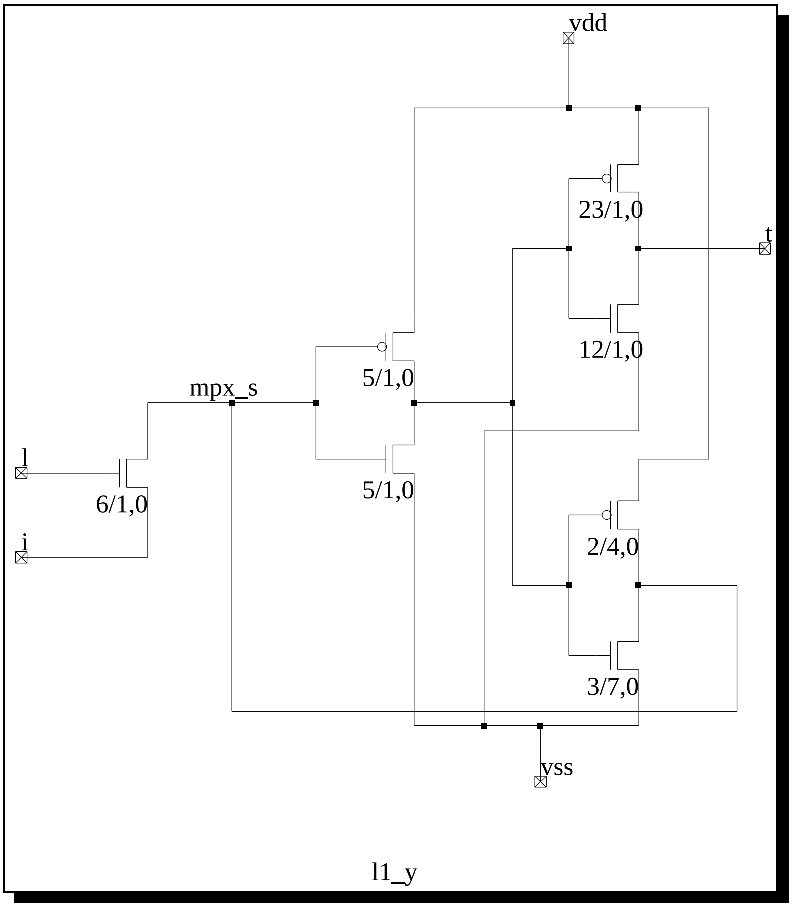


Figure B.2: Schéma du latch l1_y

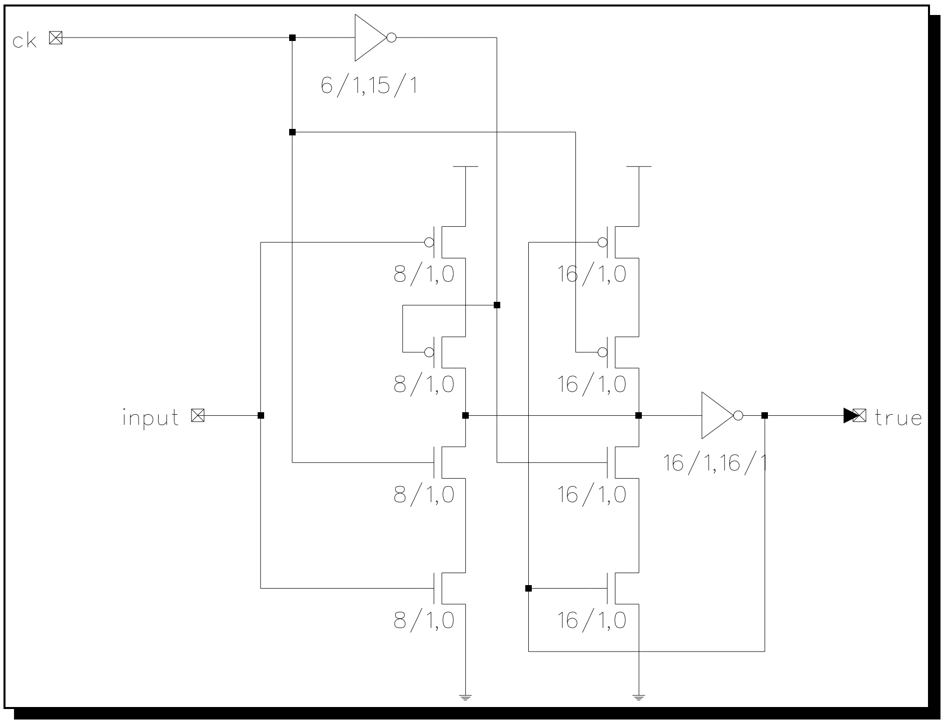


Figure B.3: Schéma du latch ls_y

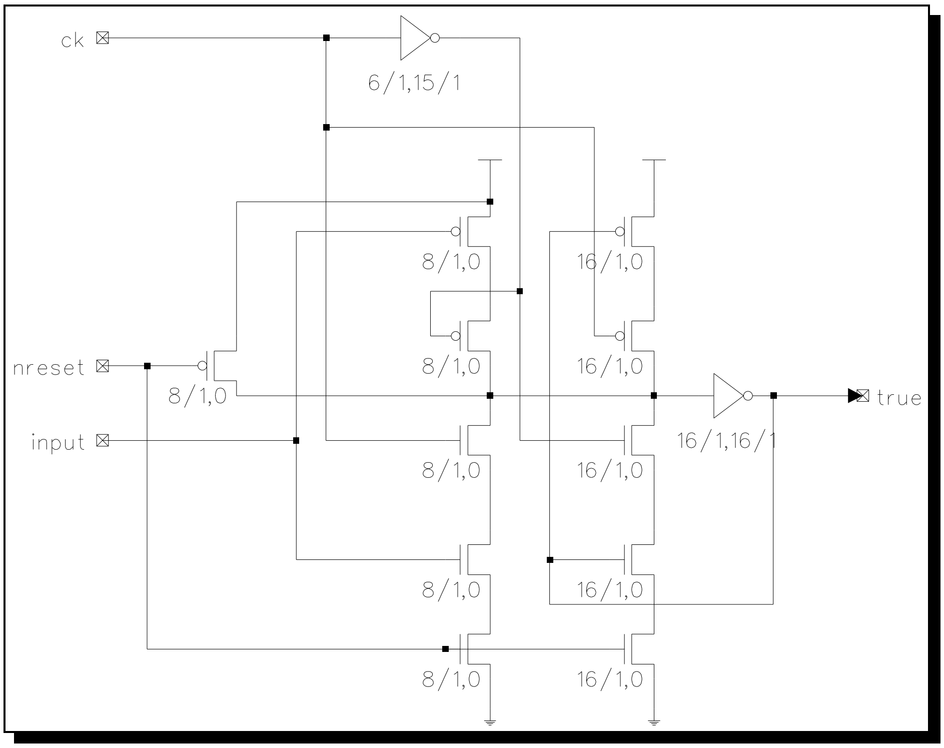


Figure B.4: Schéma du latch lsr_y

B.2.1 But de l'étude

Le but de l'étude entreprise est d'évaluer le comportement des latches l1_y, ls_y et lsr_y confrontés au phénomène de métastabilité. Pour ce faire, nous allons déterminer pour chacun d'entre eux le gain en boucle ouverte, ainsi que la constante de temps qui régit l'évolution exponentielle de la valeur de tension du point mémorisant lorsqu'il sort de l'état métastable pour rejoindre l'un des deux états stables possibles.

B.2.2 Principe de l'étude

L'étude menée se déroule en deux phases. La première phase consiste à calculer le gain statique du latch en boucle ouverte en faisant varier la tension d'entrée du point mémorisant et en observant la tension de rebouclage. Ensuite, il s'agira de faire varier la tension du point mémorisant autour de la valeur de métastabilité et déterminer la constante de temps avec laquelle il s'extrait de cet état. Le problème est de déterminer la valeur de la tension de métastabilité. C'est donc par dichotomie que l'on va s'approcher de cette valeur en positionnant à une tension V le point mémorisant et en observant vers quel état stable il converge. Pour une valeur V+ϵ, le point mémorisant convergera vers l'autre état stable. La tension de métastabilité se situe donc entre V et V+ϵ. L'évolution de la valeur de la tension du point mémorisant hors de l'état métastable suit une progression exponentielle du type:

$$\boxed{V = k \exp \frac{t}{\tau}}$$

Afin de déterminer la constante de temps τ du latch et s'affranchir du besoin de connaitre avec précision la valeur de tension du métaéquilibre, il suffit de faire la différence à temps identique des valeurs de tension obtenues dans chacun des deux cas. La courbe obtenue est aussi une exponentielle dont on peut déduire la constante de temps du latch:

$$\boxed{\tau = \frac{\Delta t}{\Delta log(V)}}$$

B.2.3 Caractéristique statique du latch l1_y

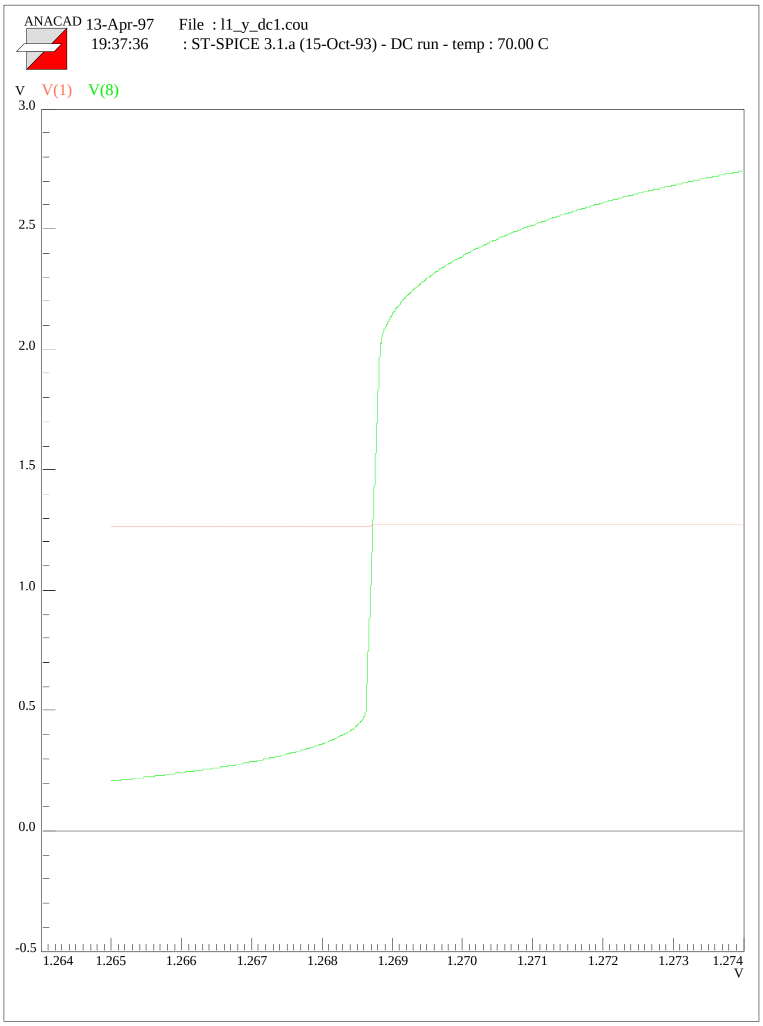


Figure B.5: Caractéristique statique du latch l1_y

Le gain obtenu pour ce latch est:

$$\boxed{G = \frac{\Delta V_{loop}}{\Delta V_{in}} = 7000}$$

Ce latch est un latch à conflit dans la mesure où l'inverseur de rebouclage est en compétition avec la tension appliquée en entrée. Cet inverseur de rebouclage est faible afin de faciliter l'écriture dans le latch. En conséquence, le gain en boucle ouverte est faible.

B.2.4 Caractéristique statique du latch ls_y

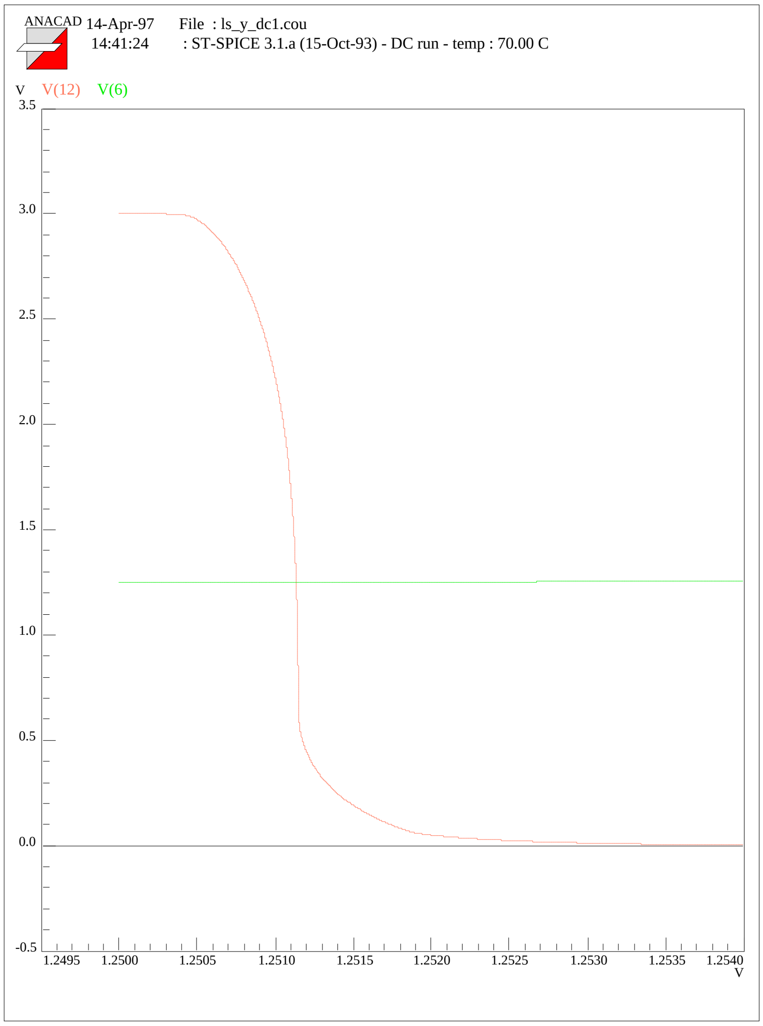


Figure B.6: Caractéristique statique du latch ls_y

Le gain obtenu pour ce latch est:

$$\boxed{G = \frac{\Delta V_{loop}}{\Delta V_{in}} = 20000}$$

B.2.5 Caractéristique statique du latch lsr_y

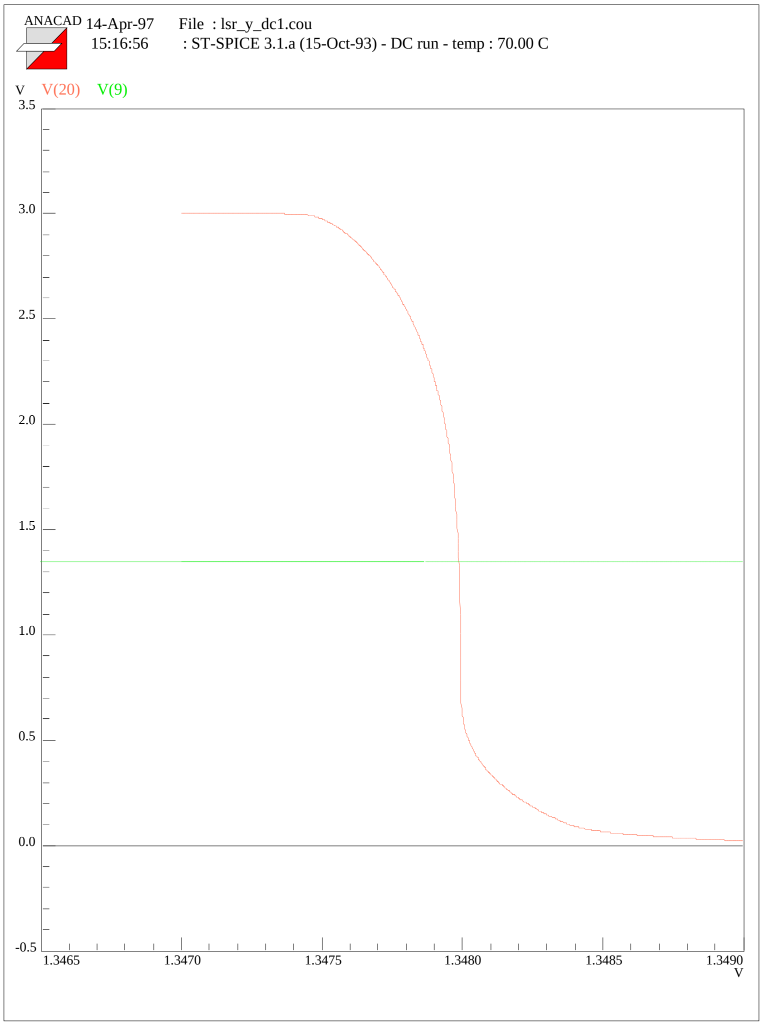


Figure B.7: Caractéristique statique du latch lsr_y

Le gain obtenu pour ce latch est:

$$\boxed{G = \frac{\Delta V_{loop}}{\Delta V_{in}} = 32000}$$

Les valeurs de gain obtenues sont à considérer d'un point de vue relatif. La conclusion que l'on peut tirer de cette étude est que les latches ls_y et lsr_y ont un gain 3 à 5 fois plus important que celui de l1_y, ce qui concourt à améliorer leur robustesse vis à vis de la métastabilité. Nous allons maintenant évaluer la probabilité d'apparition d'un état métastable en déterminant la constante de temps des trois latches étudiés. Pour déterminer cette constante de temps nous allons procéder à l'étude des caractéristiques dynamiques de ces latches.

B.2.6 Caractéristique dynamique du latch l1_y

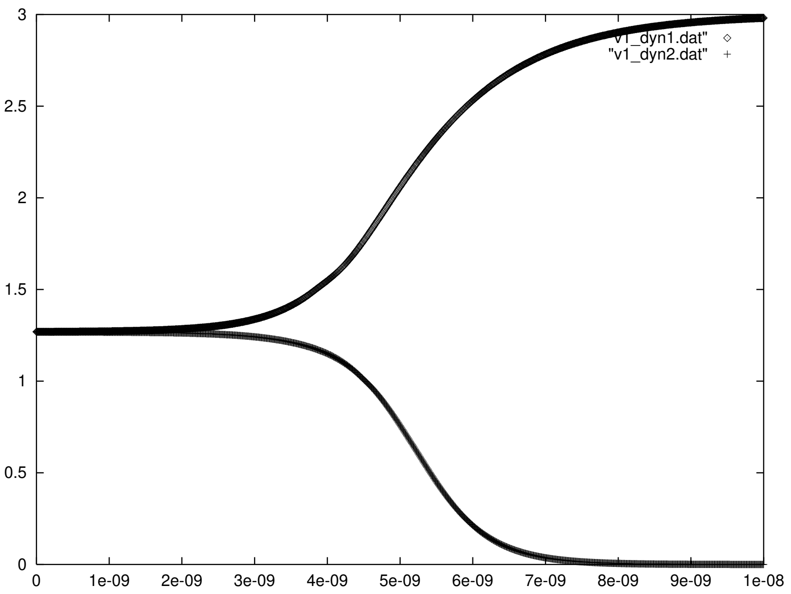


Figure B.8: Évolution de la valeur du point mémorisant du latch l1_y initialement placé à 1.2688 V et 1.2687 V

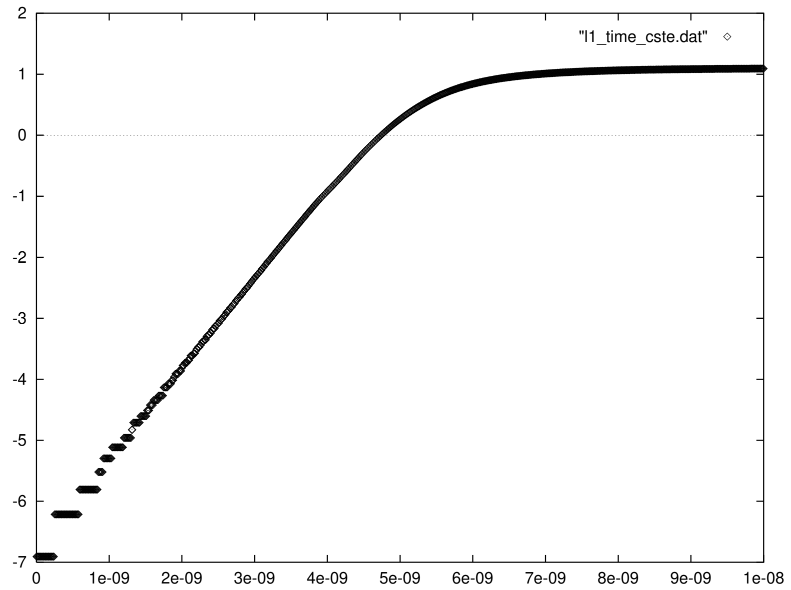


Figure B.9: Différence entre les courbes de la figure pour le point mémorisant (échelle logarithmique)

De la courbe B.9, on déduit la constante de temps du latch l1_y: 0,68 ns.

B.2.7 Caractéristique dynamique du latch ls_y

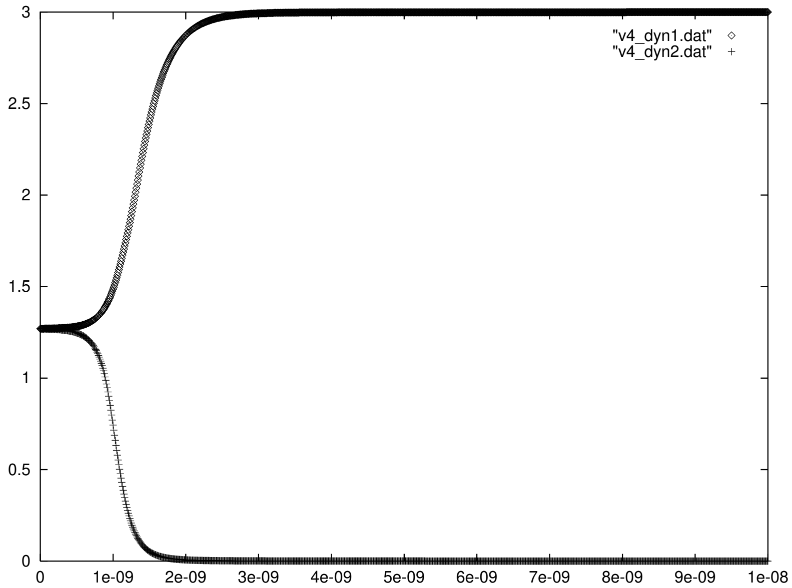


Figure B.10: Évolution de la valeur du point mémorisant du latch ls_y initialement placé à 1.2698 V et 1.2699 V

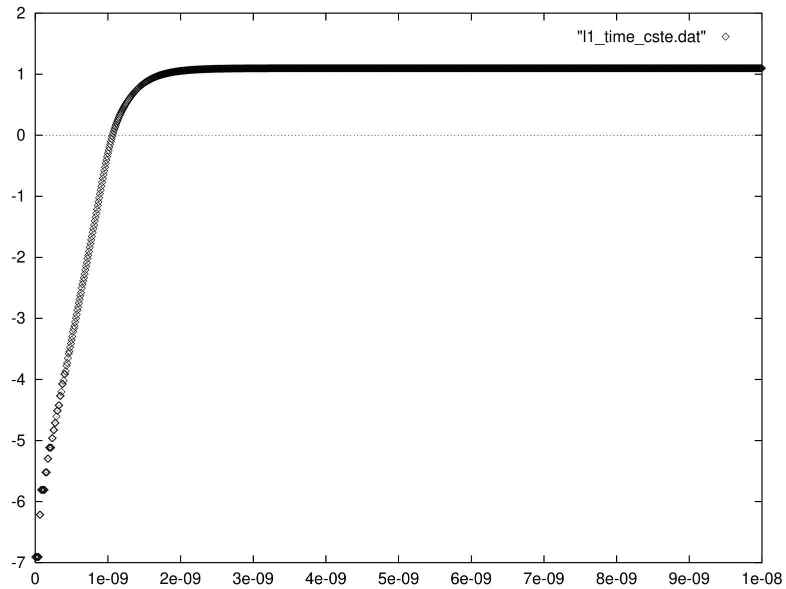


Figure B.11: Différence entre les courbes de la figure pour le point mémorisant (échelle logarithmique)

De la courbe B.11, on déduit la constante de temps du latch ls_y: 0,17 ns.

B.2.8 Caractéristique dynamique du latch lsr_y

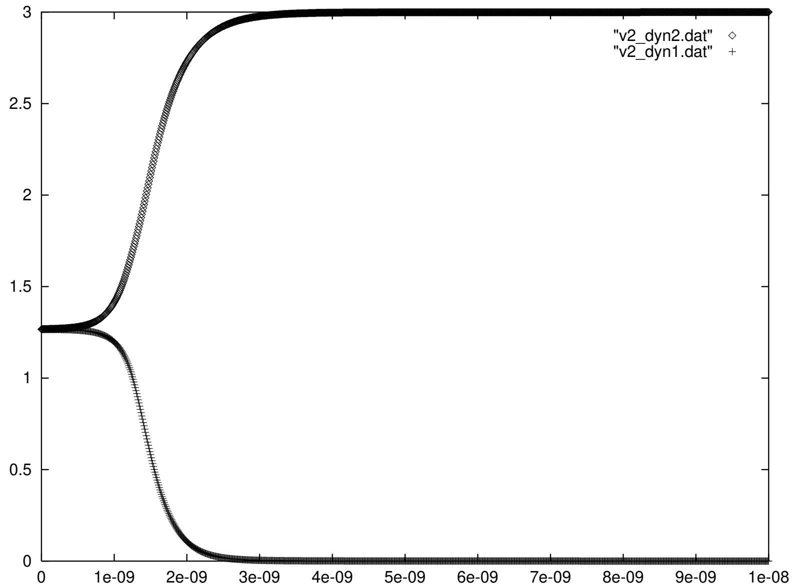


Figure B.12: Évolution de la valeur du point mémorisant du latch lsr_y initialement placé à 1.2671 V et 1.2672 V

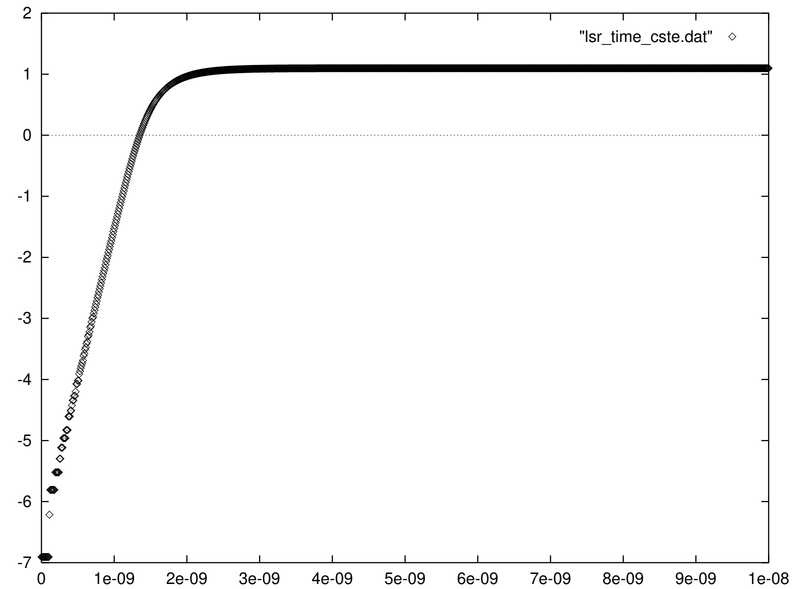


Figure B.13: Différence entre les courbes de la figure pour le point mémorisant (échelle logarithmique)

De la courbe B.13, on déduit la constante de temps du latch lsr_y: 0,20 ns.

B.2.9 Caractéristiques dynamiques comparées des latches l1_y, ls_y et lsr_y

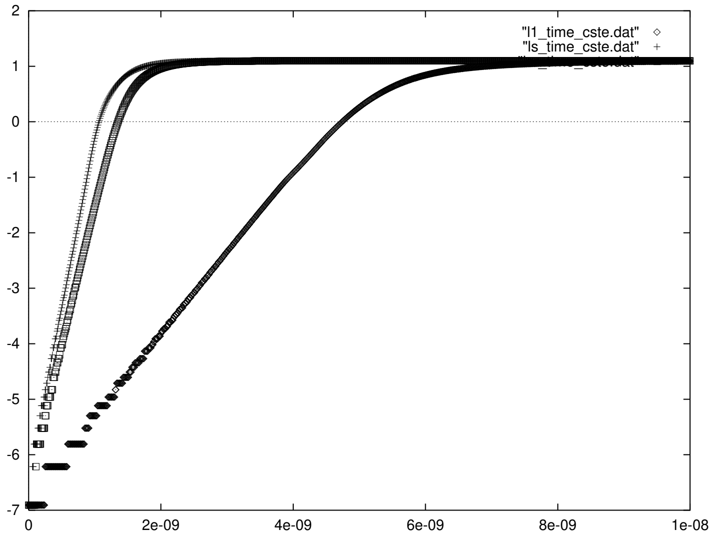


Figure B.14: Caractéristiques dynamiques comparées

B.3 Calcul de la probabilité d'apparition d'un état métastable

Le risque de métastabilité peut être évalué en exprimant le MTBF (*Mean Time Before Failure*) du latch. D'après [39], sa valeur est donnée par:

$$\boxed{\text{MTBF}=\frac{\exp(\frac{t_u}{\tau})}{2F_cF_dT_p}}$$

t_u	: temps laissé avant d'utiliser la valeur de la donnée resynchronisée.
τ	: constante de temps du latch.
F_c	: fréquence de l'horloge de resynchronisation.
F_d	: fréquence du flot de données à échantillonner.
T_p	: fenêtre temporelle de coïncidence (durée pendant laquelle le risque de métastabilité est présent).

Le facteur prépondérant est la constante de temps τ. La valeur exacte de T_p n'a que peu d'influence sur le résultat. Dans notre cas, l'ordre de grandeur de T_p est la nanoseconde. Les fréquences F_c et F_d sont de l'ordre de 80 MHz. On déduit de ces valeurs, le temps utile $t_u = T/2 - t_{set-up} - t_{traversee}$. Le tableau suivant résume les valeurs du MTBF de chaque latch pour différentes valeurs de t_u et F_c,F_d.

Fréquence		66 MHz	80 MHz
Latch	τ	T_u=6 ns	T_u=5 ns
l1_y	0,68 ns	800 μs	120 μs
lsr_y	0,20 ns	14 jours	1,6 heures
ls_y	0,17 ns	7,7 ans	5,4 jours

Table B.1: Valeur du MTBF pour différentes valeurs de fréquence et de temps utile

B.4 Structures de resynchronisation mises en œuvre

B.4.1 Resynchronisation du signal de calibration du lien HSL

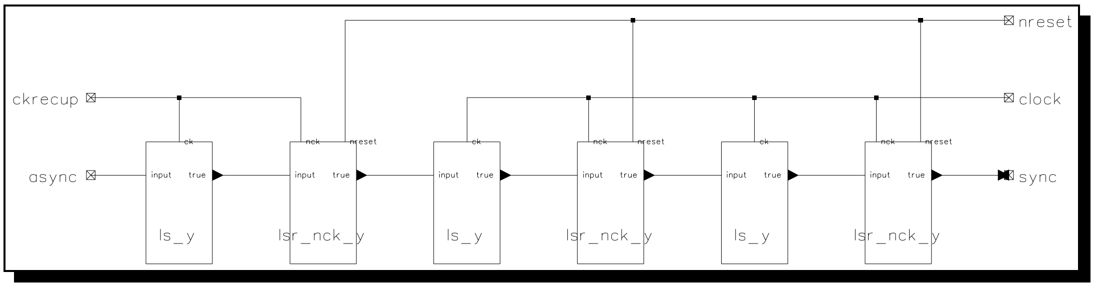


Figure B.15: Structure de resynchronisation du signal CAL

B.4.2 Resynchronisation du signal de réception d'un FCC

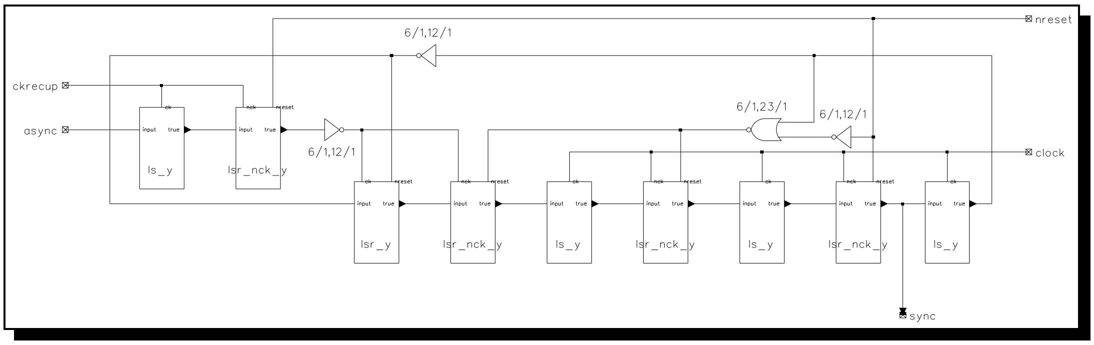


Figure B.16: Structure de resynchronisation du signal de réception d'un FCC

Les figure B.15 et B.16 présentent respectivement le schéma du bloc de resynchronisation du signal de calibration d'un lien HSL et le schéma du bloc de resynchronisation dédié aux caractères FCC. Ces deux schémas présentent une partie commune constituée de 6 latches de type ls_y ou lsr_y formant l'équivalent de trois flip-flops dans laquelle est réalisée l'opération de resynchronisation. Le MTBF de ces deux blocs dépend du MTBF de la bascule constituée par un couple de latches ls_y et lsr_y. Ce MTBF s'exprime par la formule:

$$\text{MTBF}=\frac{exp(\frac{t_{ls}}{\tau_{ls}}+\frac{t_{lsr}}{\tau_{lsr}})}{2F_cF_dT_p}$$

t_{ls} et t_{lsr} sont égaux à T/2 - t_{set-up} - $t_{traversee}$ soit 5ns pour une fréquence de 80 MHz. Avec ces valeurs le MTBF obtenu pour ces blocs de resynchronisation est de l'ordre de: 10^9 ans. Ce même calcul effectué pour le latch l1_y donne un MTBF de: 190 ms !

B.5 Conclusion

Il faut certainement relativiser la valeur du MTBF obtenue. D'autre part, dans cette étude réalisée certes, en mode pire cas, nous n'avons pas pris en compte les paramètres de température, de bruit sur l'alimentation etc... qui selon [39, 40] ont une influence. Cependant, il est clair que l'utilisation des nouveaux latches ls_y et lsr_y accroit le MTBF de très nombreux ordres de grandeur.

ANNEXE C

FCB block data-sheet

C.1 Introduction

The FCB (Flow control Block) includes all the logic and bufferisation to implement a RCUBE compliant flow control mechanism. The FCB is interfaced with another macroblock called SMI which insures the HSL boot protocol, the 8b/12b coding and decoding as well as some signals resynchronization. The FCB consists of a 64 10-bit words asynchronous fifo, an emission credit counter, a word read counter and a control finite state machine.

C.2 Interface description

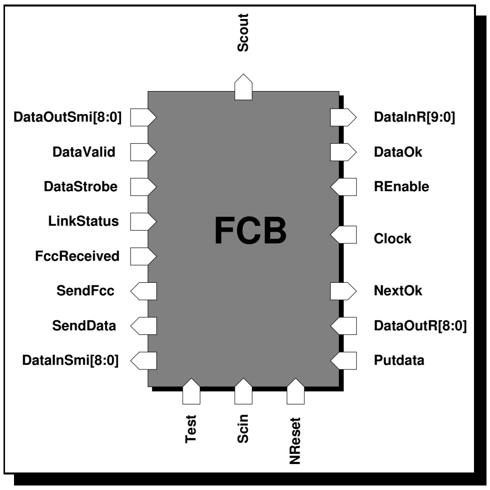


Figure C.1: FCB Interface description

PIN NAME	I/O	ACTIVE LEVEL	BRIEF DESCRIPTION
Clock	I	-	Functional clock.
Inputs from SMI			
DataOutSmi[8:0]	I	-	Parallel decoded data.
DataValid	I	high	Valid odd parity flag for **DataOutSmi[8:0]**.
DataStrobe	I	-	**DataOutSmi[8:0]** write strobe.
FccReceived	I	high	FCC received flag.
LinkStatus	I	high	Link calibration status.
Outputs to SMI			
DataInSmi[8:0]	O	-	Parallel data to be encoded.
SendFcc	O	high	FCC sending command.
SendData	O	high	Parallel data valid flag.
Inputs from router			
DataOutR[8:0]	I	-	Parallel data from the router.
PutData	I	high	Send command.
REnable	I	high	Fifo content read enable.
Outputs to router			
DataInR[9:0]	O	-	Resynchronized Parallel data.
DataOk	O	high	Valid **DataInR[9:0]**.
NextOk	O	high	Allow Send command.

TOTAL: 53 pins

The informations given hereafter specify for each signal name, its direction IN (for input) or OUT (for output) and its functionality.

C.2.1 Functional signals

Clock .... **IN**

Functional system clock.

C.2.2 Signals coming from SMI

DataOutSmi(8:0) .... **IN**

This data is the decoded parallel data that will be written on the falling edge of **DataStrobe** into the FCB asynchronous fifo.

DataValid .... **IN**

When high, this flag means that the **DataOutSmi(8:0)** had a good odd parity. This bit will be written as the 10th bit into the input asynchronous fifo.

DataStrobe .... **IN**

This strobe is used to write **DataOutSmi(8:0)** into the FCB asynchronous fifo.

FccReceived .... **IN**

Every time this signal is high for one **Clock** cycle, the router emission credit is increased by one flit (32 characters).

LinkStatus .... **IN**

When active, this signal means that the HSL is calibrated. This signal must be synchronous with **Clock**.

C.2.3 Signals going to SMI

SendFcc .... **OUT**

This signal synchronous with **Clock** commands whenever it's high for one **Clock** cycle the sending of a FCC character on the serial output.

SendData .... **OUT**

This signal synchronous with **Clock** is high whenever a valid **DataInSmi(8:0)** must be encoded and sent to the HSL to be serialized.

DataInSmi(8:0) . . . . **OUT**

This data synchronous with **Clock** will be encoded to be sent to the HSL only if **SendData** is active.

C.2.4 Signals coming from the router

REnable . . . . **IN**

When active, a read operation is performed on the asynchronous fifo at each **Clock** falling edge.

PutData . . . . **IN**

When active, this signal means that the **DataOutR[9:0]** is valid.

DataOutR(9:0) . . . . **IN**

Parallel data to be sent to SMI. If **PutData** is active, a new data can be sent every **Clock** cycle.

C.2.5 Signals going to the router

DataInR(9:0) . . . . **OUT**

Parallel data read from the asynchronous input fifo.

DataOk . . . . **OUT**

When active, this signal means that the data **DataInR(9:0)** is a valid one.

NextOk . . . . **OUT**

When this signal is unactive at zero for on **Clock** cycle, no data **DataOutR(9:0)** can be sent and thus, **PutData** must be in its inactive state during the next cycle.

C.3 Functional Description

C.3.1 SMI-Router transfer

SMI writes **DataOutSmi[8:0]** and **DataValid** in the asynchronous Fifo on the falling edge of **DataStrobe**. The router performs a read action on each **Clock** cycle if the signal **REnable** is active. Each time the signal **DataOk** is high, the available data **DataInR[9:0]** is valid and can be processed. If after a read operation, the signal **DataOk** is low, that means that the available data **DataInR[9:0]** is not valid and must not be taken in account.

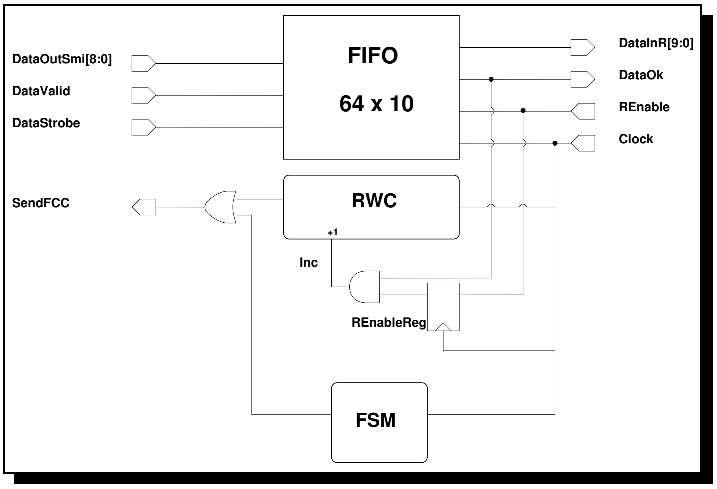


Figure C.2: Reception Block Diagram

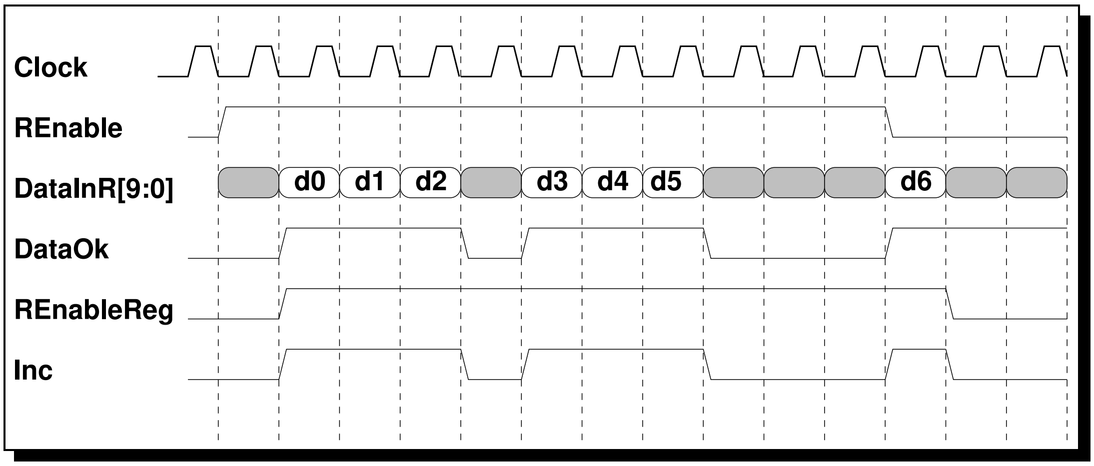


Figure C.3: SMI to Router transfer timing description

C.3.2 Router-SMI transfer

The router writes **DataOutR[8:0]** into FCB on the **Clock** falling edge. In order to validate the transfer , the signal **PutData** must be high. The actual data which is transmitted to SMI is **DataInSmi[8:0]**. This data is valid if and only if **SendData** is high. Due to FCC sending, (which forbids any data sending), a one level pipeline is used to memorize the possibly simultaneous transfer command coming from the router. In case of simultaneous FCC sending and transfer command, the signal **NextOk** goes low for one **Clock** cycle. The signal **PutData** must remain at low level during the next cycle. Otherwise, the data which would have been sent if no FCC sending had occured is lost.

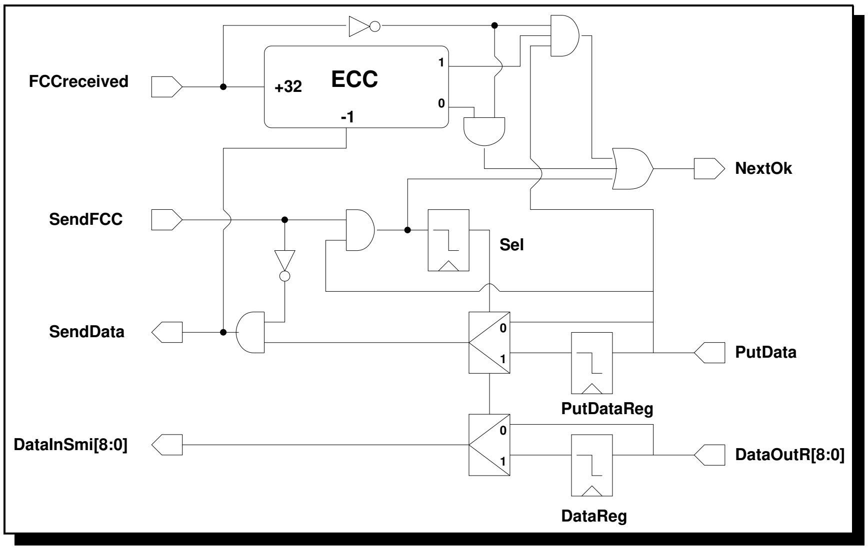


Figure C.4: Emission Block Diagram

C.3.3 Flow control policy

The flow conrol policy is flit based. A flit is a group of 32 characters. Characters are 9-bit words which are transitting from the router through FCB and SMI to be serialized by HSL. To get more information, please refer to the RCube specification report, V1.6.

Start-up

Once **LinkStatus** is high, the FSM pulses the signal **SendFCC** twice (with an interval of 8 **Clock** cycles) to inform the device situated at the other end of the cable, that the Fifo can receive 2 flits.

The Read Word Counter (RWC)

Each time the router performs a successfull read operation from the asynchronous fifo, a read word counter is decremented into FCB. When 32 words have been read, the signal **SendFcc** raises for one **Clock** cycle. This commands SMI to send a FCC character on the cable to inform the device on the other side that it can send another flit.

The Emission Credit Counter (ECC)

FCB includes a counter which is decremented each time a data is sent to SMI. When the content reaches zero, no data should be sent. The ECC is increased by 32 when a **clock** cycle wide pulse on **FccReceived** is detected. The maximum credit is 64. If more than 2 **FccReceived** are detected, the credit remains at 64.

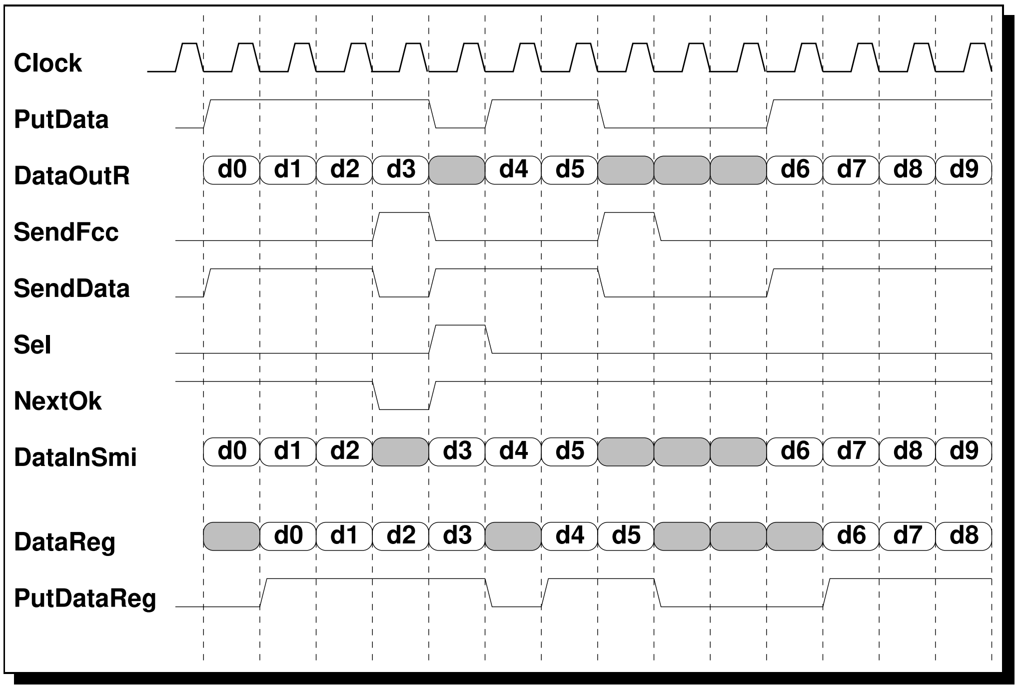


Figure C.5: Conflict between Data and Fcc sending.

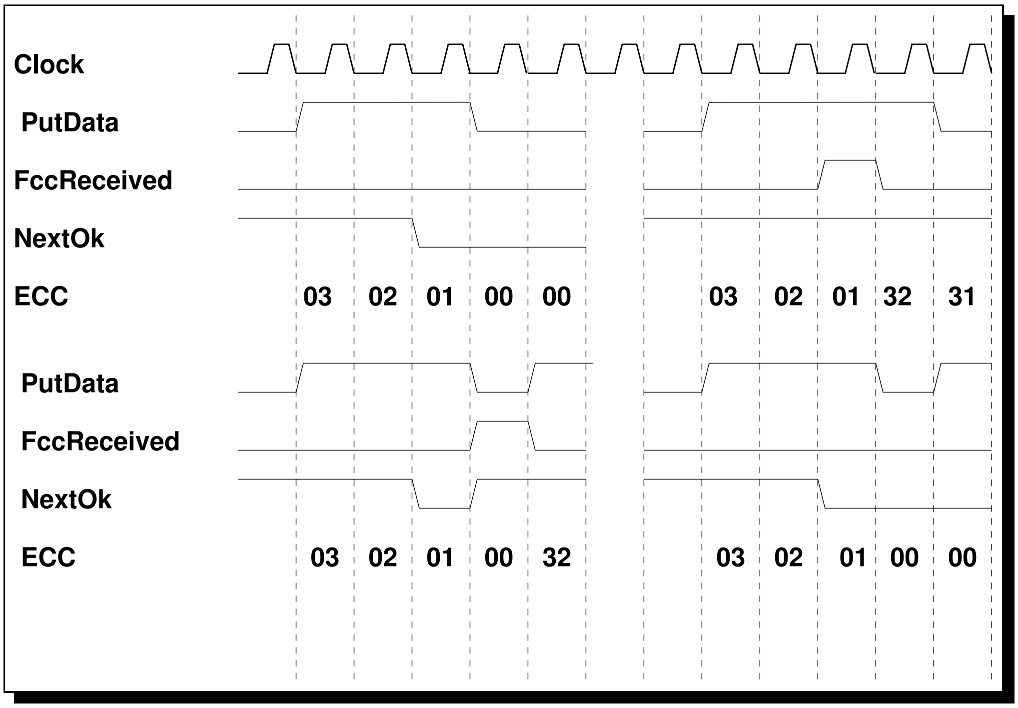


Figure C.6: Emission flow control

C.4 Timing Constraints

symbol	parameters	min	max
t_{Clock}	Clock period	12 ns	18 ns
twh_{Clock}	Clock high width	5 ns	8 ns
twl_{Clock}	Clock low width	5 ns	8 ns
tr_{Clock}	Clock edge rise time	1 ns	1 ns
tf_{Clock}	Clock edge fall time	1 ns	1 ns
t_{DIR}	Clock falling edge to DataInR[9:0]	ns	3 ns
t_{DOk}	Clock falling edge to DataOk	ns	ns
t_{NOk}	Clock falling edge to NextOk	ns	ns
t_{su}DOR[8:0]	Set-up time for DataOutR[8:0]	ns	14 ns
t_hDOR[8:0]	Hold time for DataOutR[8:0]	ns	ns
t_{su}PD	Set-up time for PutData	ns	8 ns
t_hPD	Hold time for PutData	ns	ns
t_{su}REn	Set-up time for REnable	ns	12 - $\frac{t_{Clock}}{2}$ ns
t_hREn	Hold time for REnable	ns	3 ns

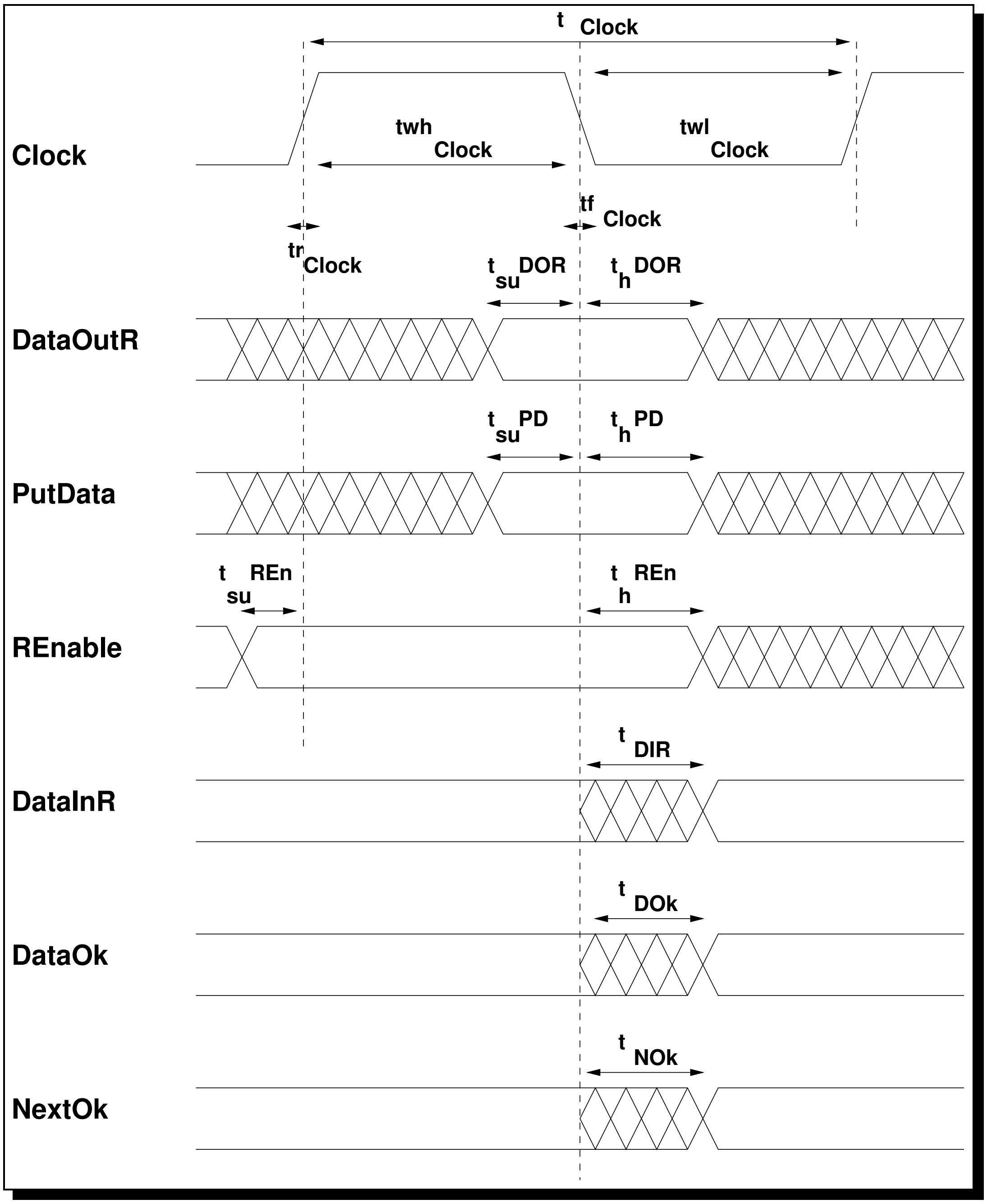


Figure C.7: FCB interface timing constraints

ANNEXE **D**

SMI block data-sheet

D.1 Introduction

The Serial Macroblock Interface (SMI) is an interface block that has been designed in the frame of the OMI-HIC HS router RCube project[1]. Its purpose was to enable the connection between an internal parallel port of the RCube router and a BULL high speed serial link (HSL). This document gives functional and timing informations on the SMI block in order to make it reusable in an other router project using HSLs. SMI has been developped using a 0.5 μm-3.3V CMOS technology. Every timing value is given assuming a worst case 0.5 μm-3.3V CMOS process.

SMI major features are the following:

- It implements the IEEE 1355[2] boot protocol for the BULL HSL.
- It performs the HSL 12b/8b decoding and 8b/12b encoding functions.
- It resynchronizes a number of signals clocked by the recovered clock coming from the HSL deserializer, with the functional clock used in SMI.
- It implements a parallel loop-back mode for test purpose.
- It provides a direct access mode that enable externally provided parallel data to be written into the router input port.

[1] The V1.6 RCube specifications report is available by anonymous ftp at the following address: ftp.humanus.fr/pub/RCube/spec_R3.ps

[2] Standard for Heterogeneous InterConnect (HIC)

D.2 Interface description

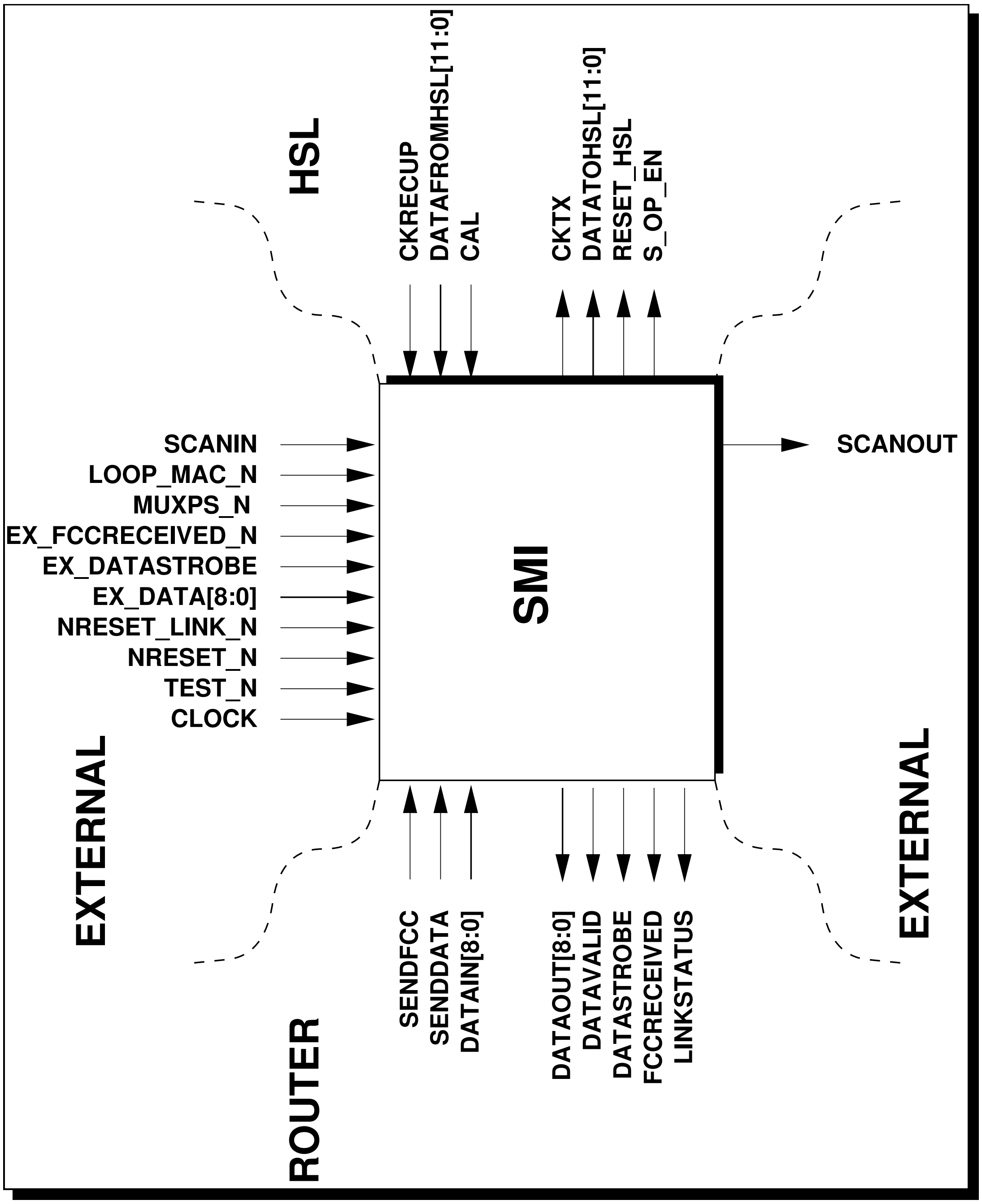


Figure D.1: SMI Interface description

PIN NAME	I/O	ACTIVE LEVEL	BRIEF DESCRIPTION
clock	I	-	Functional clock.
nreset_n	I	high	SMI and HSL reset.
nreset_link_n	I	high	HSL reset.
test_n	I	low	Test mode.
scanin	I	-	Scanpath input.
loop_mac_n	I	low	Enable the parallel loop-back mode.
muxps_n	I	low	Switch between externally provided data or data coming from the HSL.
ex_datastrobe_n	I	-	Externally provided write data strobe into the router input port (INVERTED).
ex_fccreceived_n	I	-	Externally provided FCC received signal (INVERTED).
ex_data[8:0]	I	-	Externally provided data to the router (INVERTED).
sendfcc	I	high	FCC sending command coming from the router.
senddata	I	high	Parallel data valid flag coming from the router.
datafromcore[8:0]	I	-	Parallel data to be encoded coming from the router.
ckrecup	I	-	Recovered clock coming from the HSL deserializer.
datafromhsl[11:0]	I	-	Deserialized parallel data coming from the HSL deserializer.
cal	I	high	Calibration status coming from the HSL deserializer.
cktx	O	-	Transmission clock synchronous with **clock**.
datatohsl[11:0]	O	-	Parallel encoded data to be serialized.
datatocore[8:0]	O	-	Parallel decoded data going to the router.
datavalid	O	high	Valid odd parity flag for **datatocore[8:0]**.
datastrobe	O	-	**Datatocore[8:0]** write strobe.
fccreceived	O	high	FCC received flag.
linkstatus	O	high	Calibration status (resynchronized with **clock**).
reset_hsl	O	high	HSL reset command.
s_op_en	O	high	HSL serial output enable.
scanout	O	-	Scanpath output.

TOTAL: 72 pins

The informations given hereafter specify for each signal name, its direction IN (for input) or OUT (for output) and its functionality.

D.2.1 Externally provided signals

clock .... **IN**

Functional clock.

nreset_n .... **IN**

When high, this signal asynchronously resets the HSL and synchronously resets on the falling edge of **clock**, all the SMI registers clocked by **clock**. The signal **nreset_n** must be active at least for 100 ns to insure the minimal reset period of the HSL. The HSL is set into an uncalibrated state and the serial output is forced at 0. When the signal falls, the HSL enters a boot phase (unless **nreset_link_n** is active).

nreset_link_n .... **IN**

When high, this signal asynchronously resets the HSL and synchronously resets on the falling edge of **clock**, the boot FSM. It must remains at high level for at least 100 ns to insure the minimal reset period of the HSL. The HSL is set into an uncalibrated state and the serial output is forced at 0. When the signal falls, the HSL enters a boot phase (unless **nreset_n** is active).

loop_mac_n .... **IN**

When low, this signal enables the parallel loop-back mode.

test_n .... **IN**

When this signal is low, all the flip-flops clocked by **clock** are chained in a scanpath.

WARNING: **loop_mac_n** must be active to shift the scanpath content.

scanin .... **IN**

Scanpath input.

muxps_n .... **IN**

When low, this signal enables the parallel direct access of SMI.

ex_datastrobe_n .... **IN**

When the loop-back mode is on, this externally provided strobe is connected to the output **datastrobe**.

WARNING: ex_datastrobe_n is inverted inside SMI.

ex_fccreceived_n . . . . **IN**

When the loop-back mode is on, this externally provided flow control signal is connnected to the output **fccreceived**.

WARNING: ex_fccreceived_n is inverted inside SMI.

ex_data(8:0) . . . . **IN**

When the loop-back mode is on, the externally provided parallel data is connected to the output **datatocore(8:0)**.

WARNING: ex_data(8:0) is inverted inside SMI.

D.2.2 Signals coming from the router

sendfcc . . . . **IN**

This signal that must be synchronous with **clock** commands whenever it's high for one **clock** cycle the sending of a FCC character on the serial output. **sendfcc** and **senddata** must be mutually exclusive.

senddata . . . . **IN**

This signal that must be synchronous with **clock** is high whenever a valid **datafromcore(8:0)** must be encoded and sent to the HSL to be serialized. **sendfcc** and **senddata** must be mutually exclusive.

datafromcore(8:0) . . . . **IN**

This data that must be synchronous with **clock** will be encoded to be sent to the HSL only if **senddata** is active.

D.2.3 Signals coming from the HSL

ckrecup . . . . **IN**

This clock is the recovered clock given by the HSL deserializer.

datafromhsl(11:0) . . . . **IN**

This data comes from the HSL. This data changes on the rising edge of the clock **ckrecup**.

cal . . . . **IN**

When active, this signal means that the HSL is calibrated. This signal is clocked by **ckrecup**.

D.2.4 Signals going to the HSL

cktx . . . . **OUT**

This signal is the transmission clock used by the serializer. This clock is the signal **clock** buffered.

datatohsl(11:0) . . . . **OUT**

This is the encoded data synchronous with **cktx** which is sent to the HSL.

reset_hsl . . . . **OUT**

When high, this signal resets the HSL. This signal comes from a flip-flop clocked by **clock** and depends upon **nreset_n** and **nreset_link_n**. It must last at least 100 ns to match the HSL minimal reset time.

s_op_en . . . . **OUT**

When high, this signal validates the serial output. Otherwise, the serial output is forced at 0.

D.2.5 Signals going to the router

datatocore(8:0) . . . . **OUT**

This data is the decoded parallel data that will be written on the falling edge of **datastrobe** into the router input port.

datavalid . . . . **OUT**

When high, this flag means that the **datatocore(8:0)** had a good odd parity.

datastrobe . . . . **OUT**

This strobe is used to write **datatocore(8:0)** into the router input port.

fccreceived . . . . **OUT**

Every time this signal is high for one **clock** cycle, the router increases its emission credit by one flit (32 characters in RCube).

linkstatus .... **OUT**

When active, this signal means that the HSL is calibrated. This signal is synchronous with **clock**. This signal is the **cal** signal that has been resynchronized.

D.2.6 Signal externally sent

scanout .... **OUT**

Scanpath output.

D.3 Timing

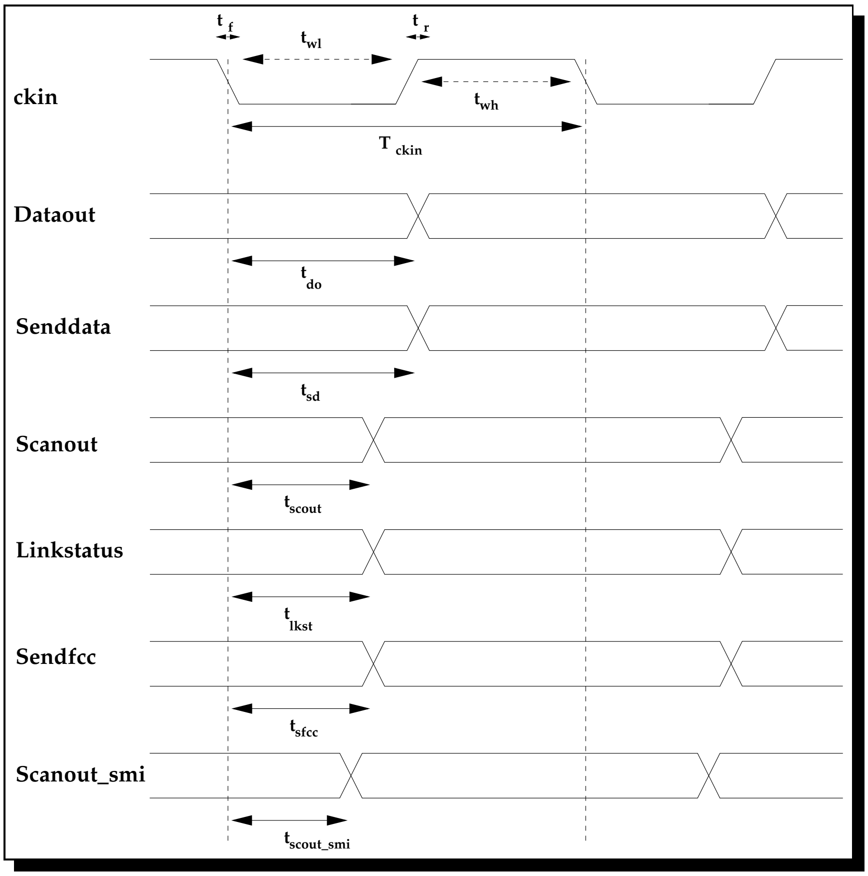


Figure D.2: Outputs timings

SYMBOL	PARAMETERS	MIN	MAX
Tck	clock period	12 ns	18 ns
tckwh	clock high width	5 ns	7 ns
tck	clock low width	5 ns	7 ns
tckre	clock edge rise time	1 ns	2 ns
tckfe	clock edge fall time	1 ns	2 ns
tck-lkst	clock falling edge to **linkstatus**		2 ns
tck-fccrcvd	clock falling edge to **fccreceived**		2.5 ns
tck-rshsl	clock falling edge to **reset_hsl**		2 ns
tck-sopen	clock falling edge to **s_op_en**		5.5 ns
tck-dtc	clock falling edge to **datatocore(8:0)**		2.5 ns
tck-dvld	clock falling edge to **datavalid**		2 ns
tck-dthsl	clock falling edge to **datatohsl(11:0)**		1 ns
tck-dstr	clock falling edge to **datastrobe**		1.5 ns
tck-dscout	clock falling edge to **scanout**		2 ns

Table D.1: Outputs timing values

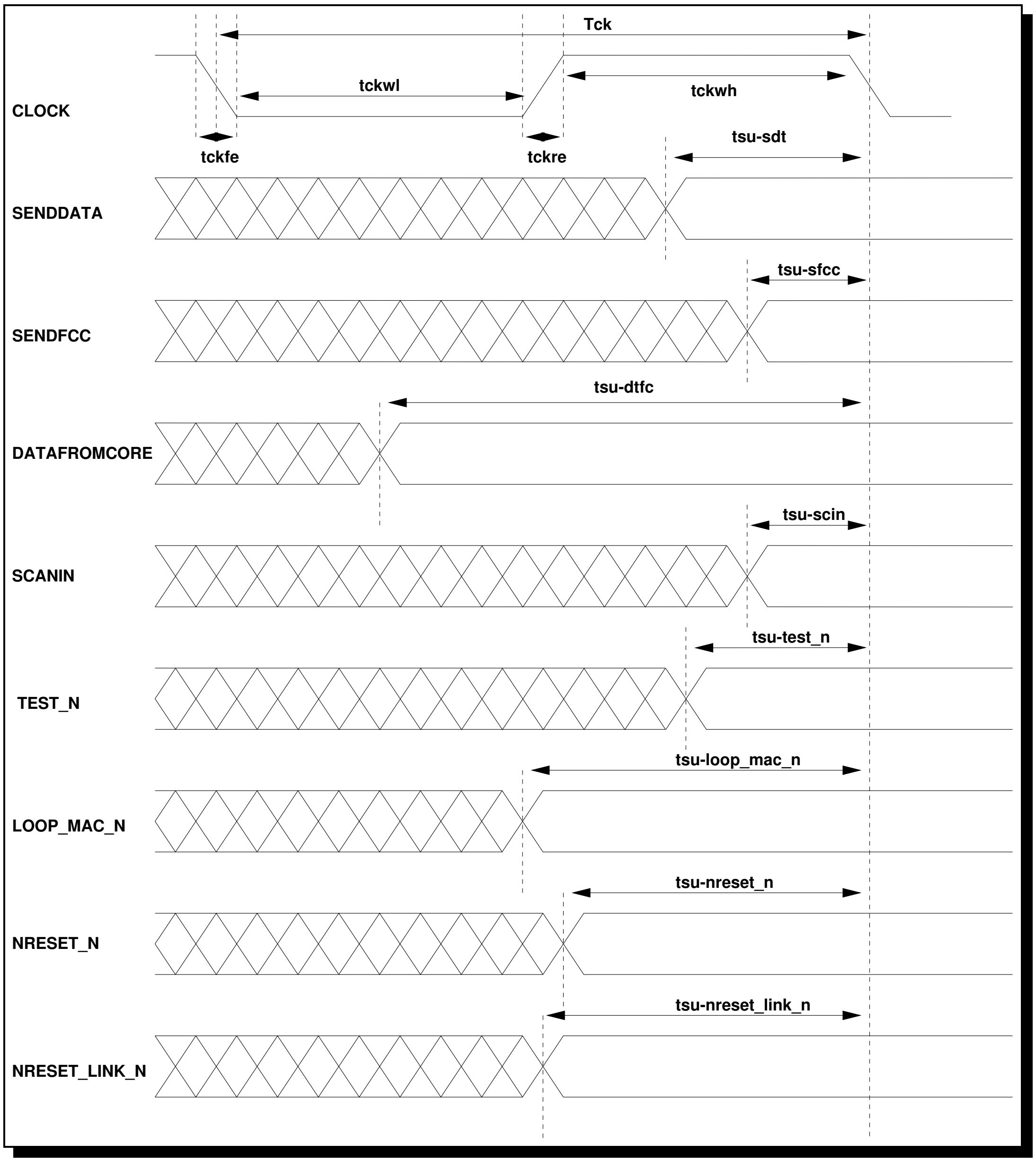


Figure D.3: Inputs timings

SYMBOL	PARAMETERS	MIN	MAX
Tck	clock period	12 ns	18 ns
tckwh	clock high width	5 ns	7 ns
tckwl	clock low width	5 ns	7 ns
tckre	clock edge rise time	1 ns	2 ns
tckfe	clock edge fall time	1 ns	2 ns
tsu_{sdt}	set-up time of **senddata**		5 ns
th_{sdt}	hold time of **senddata**		
tsu_{sfcc}	set-up time of **sendfcc**		3 ns
th_{sfcc}	hold time of **sendfcc**		
$tsu_{dtfc(8:0)}$	set-up time of **datafromcore(8:0)**		12 ns
$th_{dtfc(8:0)}$	hold time of **datafromcore(8:0)**		
tsu_{scin}	set-up time of **scanin**		3 ns
th_{scin}	hold time of **scanin**		
tsu_{nreset_n}	set-up time of **nreset_n**		7.5 ns
th_{nreset_n}	hold time of **nreset_n**		
$tsu_{nreset\_link\_n}$	set-up time of **nreset_link_n**		8 ns
$th_{nreset\_link\_n}$	hold time of **nreset_link_n**		
tsu_{test_n}	set-up time of **test_n**		4.5 ns
th_{test_n}	hold time of **test_n**		
$tsu_{loop\_mac\_n}$	set-up time of **loop_mac_n**		8.5 ns
$th_{loop\_mac\_n}$	hold time of **loop_mac_n**		

Table D.2: Inputs timing values

SYMBOL	PARAMETERS	MIN	MAX
$t_{muxps \rightarrow datastrobe}$	**muxps_n** to **datastrobe**		2 ns
$t_{muxps \rightarrow datavalid}$	**muxps_n** to **datavalid**		3 ns
$t_{muxps \rightarrow fccreceived}$	**muxps_n** to **fccreceived**		3 ns
$t_{muxps \rightarrow linkstatus}$	**muxps_n** to **linkstatus**		2.5 ns
$t_{muxps \rightarrow datatocore(8:0)}$	**muxps_n** to **datatocore(8:0)**		3 ns
$t_{loop\_mac \rightarrow datastrobe}$	**loop_mac_n** to **datastrobe**		4 ns
$t_{nreset\_n \rightarrow reset\_hsl}$	**nreset_n** to **reset_hsl**		1.5 ns
$t_{nreset\_n \rightarrow s\_op\_en}$	**nreset_n** to **s_op_en**		3 ns
$t_{nreset\_link\_n \rightarrow reset\_hsl}$	**nreset_link_n** to **reset_hsl**		2 ns
$t_{nreset\_link\_n \rightarrow s\_op\_en}$	**nreset_link_n** to **s_op_en**		3.5 ns
$t_{ex\_data(8:0) \rightarrow datatocore(8:0)}$	**ex_data(8:0)** to **datatocore(8:0)**		3 ns
$t_{ex\_datastrobe \rightarrow datastrobe}$	**ex_datastrobe** to **datastrobe**		2 ns
$t_{ex\_fccreceived \rightarrow fccreceived}$	**ex_fccreceived_n** to **fccreceived**		3 ns

Table D.3: Inputs to Outputs timing values

D.4 HSL boot protocol

Before any use, two HSLs (each composed of a serializer and a deserializer fig.D.4) connected together must follow a boot phase (called calibration) which insures that they can exchange data. This phase occurs at the circuit powering and each time a calibration loss happens.

D.4.1 HSL interface

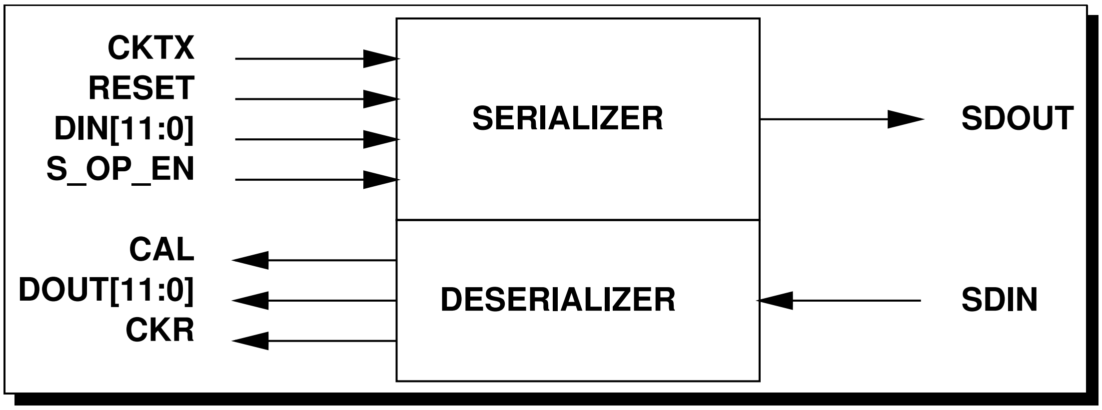


Figure D.4: HSL interface

name	I/O	active level	brief description
CKTX	I	-	transmission clock
DIN[11:0]	I	0 I xxxxxxxx AP 1	parallel data to be serialized
SDOUT	O	-	serial output
RESET	I	1	HSL macrocell reset
S_OP_EN	I	1	SDOUT validated
CKR	O	-	recovered clock
DOUT[11:0]	O	-	deserialized data
SDIN	I	-	serial input
CAL	O	1	deserializer is calibrated

Table D.4: HSL interface description

CKTX . . . . IN

This clock is the transmission clock used to serialize the outgoing parallel data.

DIN(11:0) . . . . IN

This is the encoded parallel data that must be serialized. This data is clocked by the transmission clock CKTX.

SDOUT .... OUT

Serial output.

RESET .... IN

When this signal is active, the HS link is reseted, the serial output is forced at 0.

S_OP_EN .... IN

When active, this signal validates the serial output. When inactive, the serial output is forced at 0.

CKR.................... OUT

This clock is the clock recovered by the deserializer.

DOUT(11:0) .... OUT

This is the incoming encoded parallel data clocked by CKR.

SDIN................... IN

Serial input.

CAL.................... OUT

When active this signal indicates that the HS link is calibrated. This signal is clocked by the recovered clock CKR.

D.4.2 Boot Protocol

The boot protocol of a HSL is implemented in a finite state machine inside SMI. This finite state machine (FSM)which state diagram is described on figure D.6, is clocked by **clock** (different from the recovered clock **ckrecup**). Nonetheless the FSM needs informations given by signals clocked by **ckrecup**. Resynchronization mechanisms have been implemented to make these signal synchronous with **clock**.

The calibration procedure between two HSL connected together through a coax cable (fig.D.5) is describe thereafter.

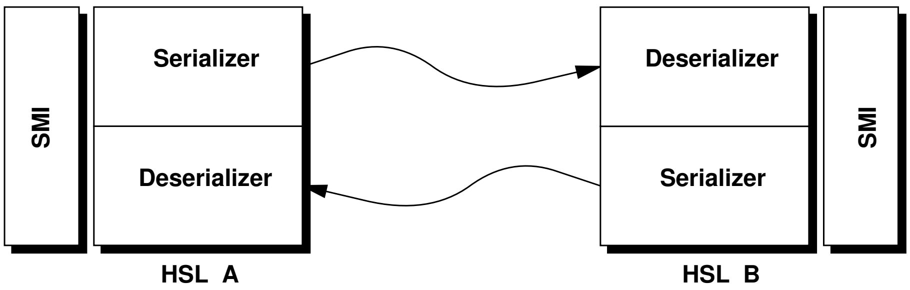


Figure D.5: HSL macrocells connected together

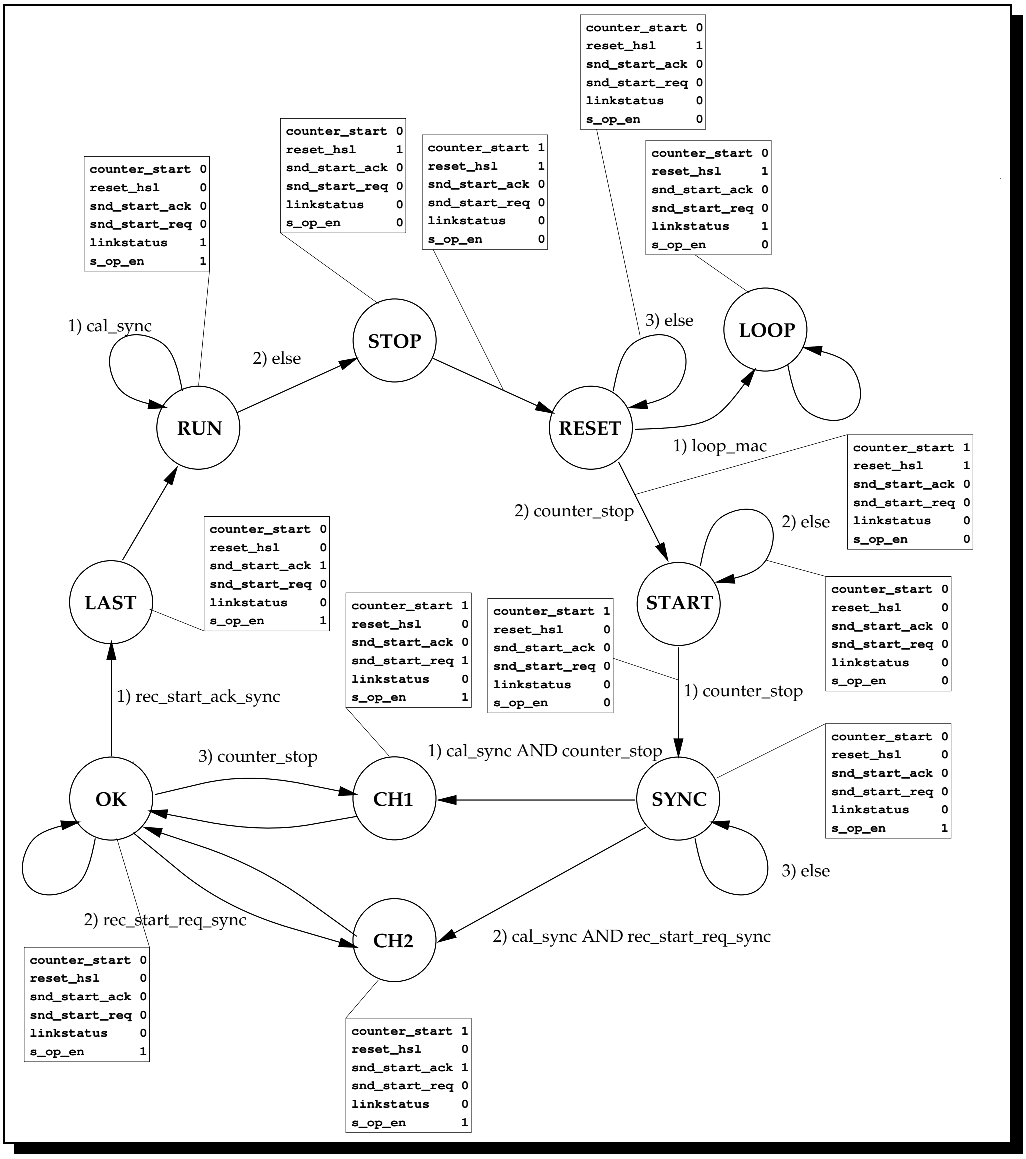


Figure D.6: HSL boot protocol finite state machine

STOP . . . . Whenever the signals **nreset_n** or **nreset_link_n** are active and **clock** is running, the serial link boot protocol is inhibited. The HS link is reseted and its serial output is forced at 0.

RESET . . . . When **nreset_n** and **nreset_link_n** are released, the finite state machine resets the HSL for 4096 **clock** cycles whereas the serial output is forced at 0.

START . . . . The serializer begins calibrating. This phase lasts 4096 **clock** cycles. To prevent the serializer from sending anything on the cable that could make the deserializer it's connected to, unable to calibrate, the serial output is forced to 0.

SYNC . . . . The serializer must be calibrated. The serial output is validated and starts emitting IDLE characters. Meanwhile the deserializer waits for IDLE characters. It must receive IDLE characters for at least 3000 **clock** cycles in order to calibrate. This phase actually lasts 4096 **clock** cycles in SMI.

CH1 . . . . When the deserializer is calibrated, a start_request is sent on the cable and a time-out of 4096 **clock** cycles is launched.

CH2 . . . . The deserializer is calibrated and has received a start_request from the HSL it is connected to. A start_acknowledge is returned. A 4096 **clock** cycles time out is launched.

OK . . . . If a start_acknowledge is received in answer to the start_request, that means that the two HSL can exchange data, the next state is then LAST. If a start_request is received, the next state is CH2 where a start_acknowledge is returned. If no start_acknowledge nor start_request is received before the time-out is finished, the next state is CH1 where another start_request is sent.

LAST . . . . A start_acknowledge is sent to close the boot protocol.

RUN . . . . The HSL is ready to send data. The signal **linkstatus** which means that the HSL is functional raises.

D.5 HSL character format

The HSL uses an 8b/12b DC balanced code. This code provides the following features:

- A positive going synchronization transition at the beginning of every character, to enable clock recovery.
- an odd parity bit per character, enabling the detection of single bit errors.
- DC balanced with a maximum disparity of 8 bits.

Define D[7:0] to be the byte to be transmitted, where D[7] is the most significant bit. To each byte, an odd parity, AP, and an inversion bit, I, are added to form the encoded character on 10 bits. The transmission character on 12 bits is the encoded character plus the Start ('1') and Stop ('0') bits which provide the positive going synchronisation transition.

The transmission character therefore has the form:

Start('1') AP e[0] e[1] e[2] e[3] e[4] e[5] e[6] e[7] I Stop('0')

where e[7:0] is d[7:0] or $\bar{d}$[7:0] as defined below.
The AP bit is defined as follows:

- AP = 1 if there is an even number of 1's in e[7:0] and I.
- AP = 0 if there is an odd number of 1's in e[7:0] and I.

D.5.1 DC balance

A code that is DC balanced is one that has a constant DC component regardless of the data patterns. This provides many advantages for high data rate transmission on fibre optic and long distance copper media.

Assigning +1 and -1 to bit levels 1 and 0 respectively provides a measure of the DC component in each transmitted character. The *disparity* D, of a data word is the difference between the number of 1's and the number of 0's in the word - positive and negative disparities refer to an excess of 1's and 0's respectively. The *running disparity*, RD, is the running (cumulative) sum of the disparities of all previous characters. In a code that's ideally DC balanced each transmitted character has a D=0 and thus RD is always equal to 0. However, perfect DC balance is not required in practice.

To maintain DC balance, the RD has always to tend towards zero. For each character to be sent, the following algorithm is used: calculate the disparity of the encoded character (since the start and Stop bits are 1 and 0 respectively, they do not affect the disparity) with I=0. If the result will reduce the absolute value of RD, then transmit the character as it is (i.e. e[7:0] = d[7:0], I = 0). If the result will increase the absolute value of the RD, then invert the encoded character and set I = 1 (i.e. e[7:0] = $\bar{d}$[7:0], I = 1), so that the inverted character reduces or leaves unchanged the absolute value of RD. The receiver uses the bit I to determine wether the character has been inverted or not. By its definition, the AP is also inverted if d[7:0] and I are inverted. In summary:

- if RD $\leq$ 0 and D > 0 then I=0, e[7:0] = d[7:0] (character non-inverted)
- if RD < 0 and D $\leq$ 0 then I=1, e[7:0] = $\bar{d}$[7:0] (character inverted)
- if RD $\geq$ 0 and D < 0 then I=0, e[7:0] = d[7:0] (character non-inverted)
- if RD > 0 and D $\geq$ 0 then I=1, e[7:0] = $\bar{d}$[7:0] (character inverted)

D.5.2 Control character

When the disparity of the encoded character is zero, it does not affect the RD, and can therefore be transmitted either non-inverted or inverted. This property enables the transmission of control characters[3]. As defined above, if D = 0, the character is

[3] Refer to the IEEE 1355 standard for further informations on control characters

transmitted inverted. Control characteres are therfore defined as characters which have D = 0 but are transmitted non-inverted.

D.6 Loop-back mode

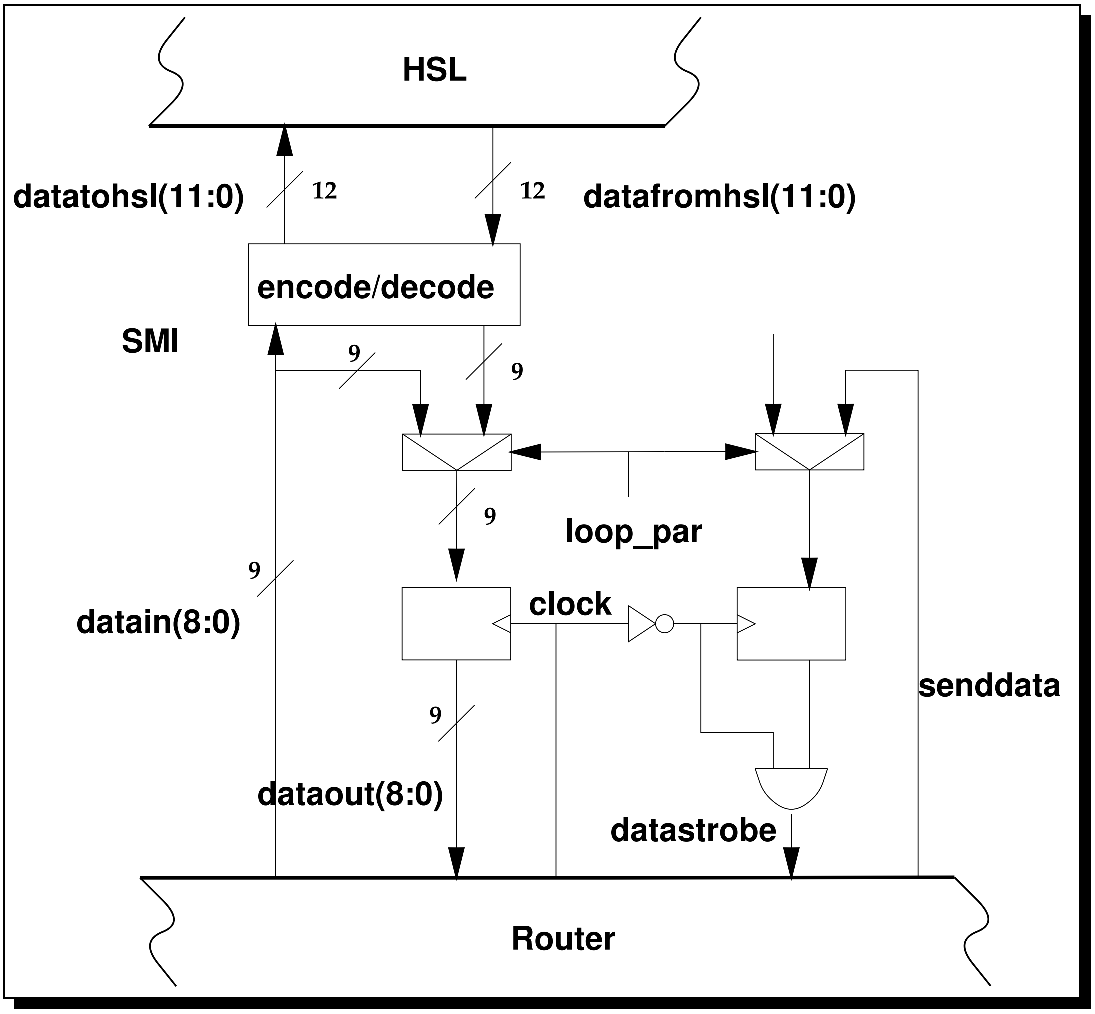


Figure D.7: Loop back mode

For test purpose, a parallel loop back mode has been implemented inside SMI. When enabled, this mode disconnects every signal coming from the HSL and forces the serial output at '0'. When the signal **loop_mac_n** is active, the data **datafromcore(8:0)** is latched into a flip-flop barrier on the falling edge of **clock** and then connected to **datatocore(8:0)**. The signal **datastrobe** is produced according to the value of the input signal **senddata**. When **loop_mac_n** is active, signals **datavalid** and **linkstatus** are forced at '1'.

D.7 Direct parallel access mode

SMI provides a Direct Parallel Access (DPA) that enables the user to write data directly into the router parallel input port (fig.D.8).

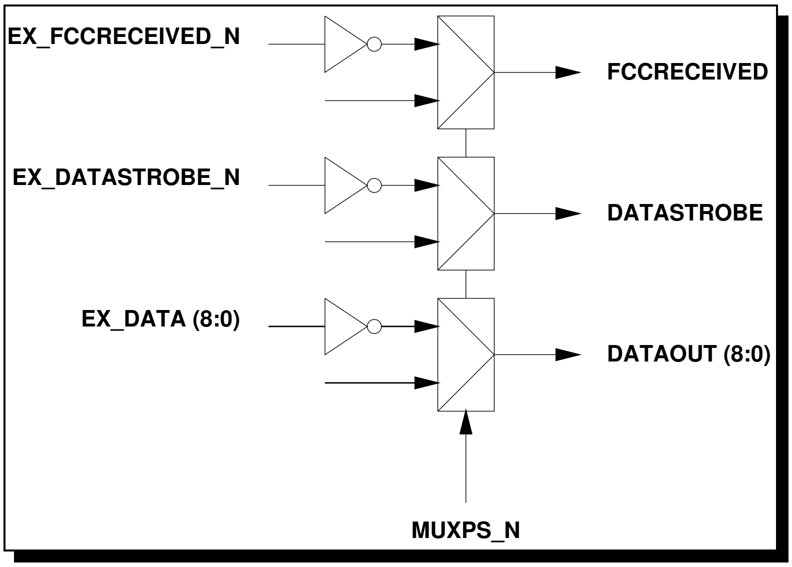


Figure D.8: Direct Parallel Access mode

D.8 Test

This scanpath contains 46 bits. Bit 1 is written first. The signal **loop_mac_n** must be active to enable the scanpath content shifting.

1	smilogic.mux.reg_11	24	smilogic.automate.regs_7
2	smilogic.mux.reg_10	25	smilogic.automate.regs_8
3	smilogic.mux.reg_9	26	smilogic.automate.regs_9
4	smilogic.mux.reg_8	27	smilogic.huitversdouze.rbo_0
5	smilogic.mux.reg_7	28	smilogic.huitversdouze.rbo_1
6	smilogic.mux.reg_6	29	smilogic.huitversdouze.rbo_2
7	smilogic.mux.reg_5	30	smilogic.huitversdouze.rbo_3
8	smilogic.mux.reg_4	31	smilogic.huitversdouze.rbo_4
9	smilogic.mux.reg_3	32	smilogic.quatremille.carry_l1
10	smilogic.mux.reg_2	33	smilogic.quatremille.reg_1_5
11	smilogic.mux.reg_1	34	smilogic.quatremille.reg_1_4
12	smilogic.mux.reg_0	35	smilogic.quatremille.reg_1_3
13	smilogic.mux.poub	36	smilogic.quatremille.reg_2_2
14	smilogic.paraloop.fcc	37	smilogic.quatremille.reg_1_1
15	smilogic.paraloop.link_etat	38	smilogic.quatremille.reg_1_0
16	smilogic.paraloop.ap_reg	39	smilogic.quatremille.carry_l2
17	smilogic.automate.regs_0	40	smilogic.quatremille.reg_2_5
18	smilogic.automate.regs_1	41	smilogic.quatremille.reg_2_4
19	smilogic.automate.regs_2	42	smilogic.quatremille.reg_2_3
20	smilogic.automate.regs_3	43	smilogic.quatremille.reg_1_2
21	smilogic.automate.regs_4	44	smilogic.quatremille.reg_2_1
22	smilogic.automate.regs_5	45	smilogic.quatremille.reg_2_0
23	smilogic.automate.regs_6	46	smilogic.quatremille.counterstop

Table D.5: SMI scanpath description