Spécification de bibliothèques pour la synthèse de circuits asynchrones

Jean-Baptiste Rigaud

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

T H E S E

pour obtenir le grade de

DOCTEUR DE l'INPG

Spécialité : Microélectronique

préparée au laboratoire **TIMA** dans le cadre de
l'Ecole Doctorale d'« Electronique, Electrotechnique, Automatique, Télécommunications, Signal »

présentée et soutenue publiquement

par

Jean-Baptiste RIGAUD

Le 23 décembre 2002

Titre :

SPECIFICATION DE BIBLIOTHEQUES
POUR
LA SYNTHESE DE CIRCUITS ASYNCHRONES

Directeur de Thèse : Marc Renaudin
Codirecteur : Gilles Sicard

JURY

M. Michel Robert	, Président
M. Christian Piguet	, Rapporteur
M. Eric Martin	, Rapporteur
M. Marc Renaudin	, Directeur
M. Gilles Sicard	, Codirecteur
M. Philippe Matherat	, Examinateur

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__/__/__/__/__/__/__/__/__/__|

T H E S E

pour obtenir le grade de

DOCTEUR DE l'INPG

Spécialité : Microélectronique

préparée au laboratoire **TIMA** dans le cadre de
l'Ecole Doctorale d'« Electronique, Electrotechnique, Automatique, Télécommunications, Signal »

présentée et soutenue publiquement

par

Jean-Baptiste RIGAUD

Le 23 décembre 2002

Titre :

SPECIFICATION DE BIBLIOTHEQUES
POUR
LA SYNTHESE DE CIRCUITS ASYNCHRONES

Directeur de Thèse : Marc Renaudin
Codirecteur : Gilles Sicard

JURY

M. Michel Robert	, Président
M. Christian Piguet	, Rapporteur
M. Eric Martin	, Rapporteur
M. Marc Renaudin	, Directeur
M. Gilles Sicard	, Codirecteur
M. Philippe Matherat	, Examinateur

Résumé

Ce travail de thèse s'intègre dans le développement de l'outil de conception automatique de circuits asynchrones TAST (« TIMA Asynchronous Synthesis Tool »). C'est un environnement de conception principalement composé d'un compilateur et d'un synthétiseur offrant la possibilité de cibler plusieurs formats de sortie (description comportementale en VHDL, langage C, description structurelle au niveau porte en VHDL) à partir de descriptions de haut niveau décrites en langage CHP. Le résultat produit par le synthétiseur est une description au niveau porte qui utilise une bibliothèque de cellules génériques.

Cette thèse s'attache à la réalisation de l'interface entre le « front-end » de l'outil TAST et le « back-end » intégrant les outils commerciaux du flot de conception traditionnel. Pour cela, la bibliothèque générique qui permet de décrire le résultat de la synthèse a été spécifiée puis développée. Cette bibliothèque est une description fonctionnelle des portes qui est indépendante de la technologie. De plus, cette interface comprend des bibliothèques de cellules spécifiques qui permettent de concevoir un circuit de type ASIC ou de type FPGA. La spécification de la bibliothèque de cellules génériques a nécessité une étude approfondie sur les communications à travers un canal. Cette étude présente d'une manière unifiée les différents moyens pour réaliser ces communications et les différentes façons de les implanter au niveau circuit. Pour compléter le développement de la bibliothèque, le problème de la réalisation d'arbitres est abordé dans le cadre élargi de la conception de réseaux de communication sur puce (« Network on Chip »).

Mots-clés : Circuits asynchrones, synthèse de circuits asynchrones, langage de processus communicants, réseau de Pétri, protocoles de communications, arbitrage, bibliothèques, cellules standard, portes de Muller.

Abstract

This PhD thesis deals with the development of the TAST automated tool suite (« TIMA Asynchronous Synthesis Tool »). It is a design environment mainly made up of a compiler and a synthesizer targeting several output formats (VHDL behavioral description, C language description, VHDL gate level netlist) starting from a high level description language (CHP). The synthesizer produces is a gate netlist which uses a generic cell library.

This work presents the interface between the front-end of TAST and its back-end integrating commercial tools of the standard design flow. To do that, this generic library, used to describe the result of the synthesis, was specified and developed. This library includes a technology independent functional description of the gates. Moreover, the interface includes specific libraries for ASIC and FPGA. The generic library specification required an in-depth understanding of the message passing mechanisms through channel. Therefore this work adopted a unified way to study the communication protocols and their corresponding implementations. To complete the library specification and development, arbiter design was considered as a contribution to network on-chip implementation.

Keywords : Asynchronous circuits, asynchronous circuit synthesis, communicating process language, Petri net, communication protocols, arbitration, cell libraries, standard cells, Muller gates.

Table des matières

Liste des figures

May the force be with you!

Remerciements

Les travaux de cette thèse ont été réalisés au sein du laboratoire TIMA, sur le site Viallet de l'Institut National Polytechnique de Grenoble. Que son directeur Bernard Courtois trouve ici tous mes remerciements pour son accueil au sein de son laboratoire.

J'exprime toute ma gratitude, à Marc Renaudin, mon directeur de Thèse, qui a accepté en 1999 de me prendre comme stagiaire DEA. Merci de m'avoir fait découvrir et converti au monde de l'asynchrone. Ainsi j'ai eu la chance de partager les débuts du groupe CIS qu'il dirige d'une manière remarquable. Je le remercie aussi de m'avoir dirigé sur ce sujet, du temps et de l'intérêt qu'il m'a accordé tout au long de ces années.

Je remercie Christian Piguet, directeur du Centre Suisse d'Electronique et de Microélectronique de Neuchâtel, et Eric Martin directeur du Laboratoire d'Electronique des Systèmes Temps Réel de Lorient, qui ont accepté d'être les rapporteurs de mes travaux de thèse.

Merci à Michel Robert, Professeur au Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier, de m'avoir fait l'honneur de présider mon jury de thèse.

Je remercie Philippe Matherat, chercheur dans la division systèmes et communication de l'ENST à Paris, qui a accepté de faire partie de ce jury, et pour l'intérêt qu'il a manifesté suite à la lecture du manuscrit.

Merci à Gilles Sicard, pour son aide précieuse, sa patience et ses bons conseils qui m'ont permis de surmonter bien des difficultés.

Je suis particulièrement redevable envers tous les membres du jury pour avoir accepté de se déplacer et de se réunir à une date si tardive dans l'année et si proche des fêtes de Noël.

Je tiens aussi à avoir une attention toute particulière pour Pierre Gentil, directeur de l'école doctorale EEATS qui a accepté que je poursuive mes études à Grenoble au sein du DEA de Microélectronique.

Enfin ces remerciements s'adressent au personnel du CIME (Alexandre Chagoya, Bernard Bosc…) qui a toujours mis à ma disposition les outils nécessaires à mes travaux ; aux administrateurs réseau et aux secrétaires du laboratoire qui ont toujours été d'une grande disponibilité et d'une patience exemplaire.

Le laboratoire TIMA, en plus d'être mon lieu de travail depuis maintenant presque quatre ans, est aussi devenu un lieu où beaucoup d'amitiés et de complicités se sont créées avec les membres du groupe et même les membres des autres équipes de recherche. Je leur adresse un très grand merci pour tous ces bons moments, de travail, de discussions, de détentes et surtout pour avoir supporté mon caractère et mes humeurs difficiles.

J'attache une dédicace un peu plus particulière au « noyau dur », devenu plus que des collègues de travail, toujours amateurs de soirées à thème : « Club », « Star Wars », « Les Bronzés », « Halloween »…

Damien Lyonnard, alias « le gamin », mon modèle en matière de légèreté d'humour, de ski et de vélo.

Emmanuel Allier, alias « Manuel », le stéphanois le plus coquet et le plus sportif de Grenoble.

Gislain, alias « Vitrrrrriiiine », alias « Drrrrrrrrrr », et accessoirement Fraidy Bouesse, le congolais le plus patient du monde avec les plus belles chaussures de paille et baskets blanches cirées du monde.

Jérôme Quartana, alias « Joli Q », mon amnésique préféré.

Dhanistha Panyasak, alias « la petite », qui m'a souvent énervé mais qui est d'une gentillesse infinie et d'une très grande patience dans ce groupe majoritairement masculin. Si elle n'existait pas il faudrait l'inventer !

Anh-Vu Dinh-Duc, alias « Frère Bob », avec qui j'ai commencé la thèse et qui a été mon compagnon de bureau pendant ces trois années.

Et puis il y a tous les autres : Laurent, Joaõ (champion ou champignon), Kamel, Amine, Mohammed…

J'adresse de très très grands mercis et toute mon affection à mon père, Solange, Marc, les « monstres » (Marine et Alexandre) et toute le reste de ma famille qui a toujours été d'un très grand soutien tout au long de ces années de travail.

Je souhaite terminer ces remerciements par mes pensées et mes sentiments les plus forts pour Christelle que j'embrasse de tout mon cœur.

Introduction

L'étude et la conception des circuits numériques asynchrones (sans horloge, synchronisation locale) remonte aux années soixantes du siècle dernier. Ces études ont été effectuées parallèlement à celles dévolues aux circuits numériques synchrones. Cependant, à l'époque, ces derniers présentaient une certaine facilité de conception en rapport aux technologies utilisées et ont donc pris l'ascendant sur les systèmes asynchrones.

De ce fait, l'évolution de la recherche des outils de conception s'est uniquement axée sur les circuits synchrones. Au cours de ces vingt dernières années, des groupes de recherches se sont focalisés sur l'asynchrone afin de proposer une alternative aux problèmes directement liés à l'évolution rapide des technologies submicroniques (consommation, bruit, distribution d'horloge). Ces études ont montré par l'intermédiaire de circuits prototypes (microprocesseurs, microcontrôleurs) la pertinence de cette approche.

Aujourd'hui, le développement d'outils de conception automatiques pour les circuits asynchrones constitue un verrou très important et difficile à lever. Ces outils doivent être du niveau de performance et de maturité des outils communément utilisés pour la conception des circuits synchrones. La maîtrise de méthodes et le développement d'outils automatiques pour la conception de circuits asynchrones sont essentiels pour, d'une part exploiter leur potentiel et, d'autre part favoriser leur diffusion et leur acceptation dans l'industrie.

Ce travail de thèse se situe dans le cadre du développement de l'outil de conception automatique de circuits asynchrones TAST. TAST est l'acronyme de « TIMA Asynchronous Synthesis Tools ». C'est un environnement de conception dédié à la synthèse de circuits asynchrones développé au sein du groupe de recherche CIS (« Concurrent Integrated Systems ») du Professeur Marc Renaudin au laboratoire TIMA.

Il est principalement composé d'un compilateur et d'un synthétiseur offrant la possibilité de cibler plusieurs formats de sorties (description comportementale en VHDL, langage C, description structurelle au niveau porte en VHDL) à partir de descriptions de haut niveau décrites dans un langage proche du CHP (« Communicating Hardware Processes ») lui même issu du CSP (« Communicating Sequential Processes »).

Le flot de synthèse est organisé autour d'un format intermédiaire basé sur les réseaux de Pétri et des graphes de flot de données. La synthèse de circuits asynchrones est basée sur la spécification DTL (« Data Transfert Level ») [DINH 02b]. Celle-ci fournit un ensemble de règles qui garantissent que le programme écrit en CHP est synthétisable par des circuits asynchrones.

TAST permet de générer les formats suivant à partir de la description sous forme de processus communicants (CHP) :

- Une description comportementale en langage C,
- Une description comportementale en VHDL.

- Les circuits quasiment insensibles aux délais sous la forme d'une netlist de portes décrites en VHDL.

- Les circuits micro pipeline sous la forme d'une description VHDL au niveau porte.

Le flot de TAST est présenté figure 1:

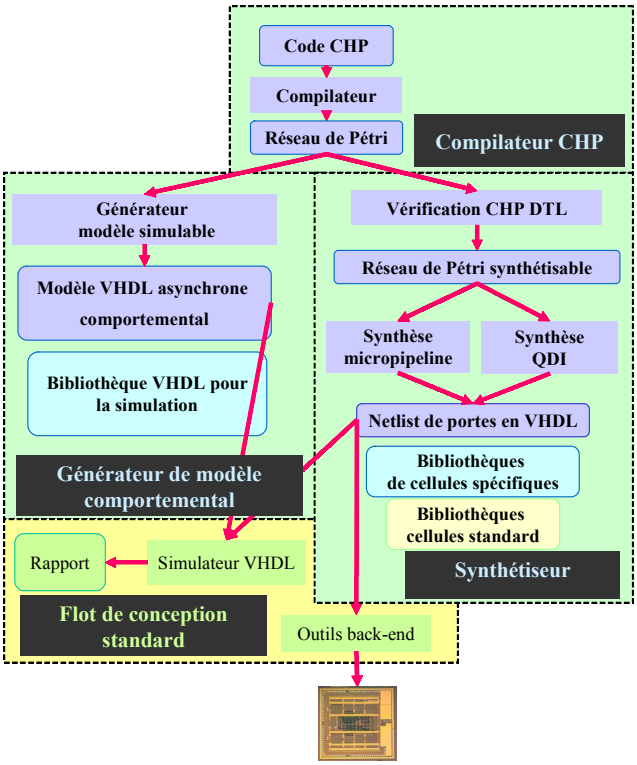


figure 1. Flot de synthèse de l'outil TAST.

TAST accepte en entrée des systèmes décrits en CHP. Le CHP est un langage de description haut niveau de circuits basé sur les processus communicants. TAST lie les différents points suivants : les protocoles de communication, le codage des données, la spécification de l'indéterminisme, la hiérarchie, la gestion de projet ([TAST 02]).

Le programme CHP est compilé et transformé en un ensemble de réseaux de Pétri (structures de contrôle et de choix, séquentialité, parallélisme) et de graphes de données (instructions associées aux places des réseaux de Pétri). De là, la spécification est transformée en un modèle VHDL comportemental afin de pouvoir vérifier par simulation la correction fonctionnelle du circuit obtenu. Le programme VHDL est alors utilisé dans les outils standard de simulation. Deux paquetages de fonctions sont ici utilisés pour réaliser les opérations arithmétiques et les conversions des types de données.

Une fois la validation de la description CHP effectuée, il faut vérifier qu'elle est compatible avec les règles de spécification DTL. Si cela n'est pas le cas, le concepteur transforme le code CHP [DINH 02b]. La synthèse permet le choix entre deux classes de

circuits asynchrones : la classe de circuits Micropipeline et la classe de circuits quasiment insensibles aux délais (QDI).

Le résultat de la synthèse est une description au niveau porte qui utilise une bibliothèque de cellules génériques. Le but est d'arriver au circuit final, en faisant le lien entre la description générée par TAST et les outils d'un flot de conception traditionnel (étapes de simulation et de placement routage).

Le travail de cette thèse est de réaliser l'interface entre le « front-end » et le « back-end » de l'outil TAST. Pour cela il faut spécifier et développer la bibliothèque générique qui permet de décrire le résultat de la synthèse. Cette bibliothèque est une description fonctionnelle des portes qui est indépendante de la technologie. De plus, cette interface comprend des bibliothèques de cellules spécifiques qui permettent d'arriver à un circuit final de type ASIC (« Application Specific Integrated Circuit ») ou de type FPGA (« Field Programmable Gate Array »).

La spécification de la bibliothèque de cellules génériques a nécessité une étude approfondie sur les communications à travers un canal. En effet, le fonctionnement des circuits asynchrones repose sur ces canaux de communication. Cette étude présente d'une manière unifiée les différents moyens pour réaliser cette communication et les différentes façons de les implémenter au niveau circuit. Pour compléter le développement de la bibliothèque, le problème de la réalisation d'arbitres est abordé dans le cadre de la conception de réseaux de communication sur puce (« Network on Chip »).

Le premier chapitre du manuscrit présente les concepts fondamentaux qui caractérisent les circuits asynchrones et qui permettent de les différencier. Contrairement aux circuits synchrones, ils possèdent une synchronisation locale réalisée par une communication particulière locale entre tous les éléments constituants le circuit. Ce mécanisme de communication a la propriété principale de rendre le système indépendant du temps de calcul des opérations dans chaque bloc. De ce fait, d'un point de vue conception, la logique constituant les circuits asynchrones doit être sans aléa. Les hypothèses simplificatrices pour la conception de circuits asynchrones sans aléa sont faites sur le modèle de délai appliqué aux portes et aux interconnexions, ainsi qu'au mode de fonctionnement (comportement du circuit vis-à-vis de son environnement). Ces hypothèses permettent finalement d'effectuer un classement des différents types de circuits asynchrones.

Le deuxième chapitre revient sur l'élément clé des circuits asynchrones : le canal de communication. L'objectif est d'avoir une vision unifiée de l'ensemble des moyens pour réaliser la communication entre plusieurs blocs d'un circuit. Comment spécifier l'échange d'informations à travers un canal (protocole), quels sont les formalismes de représentation des étapes de communication dans un canal, et enfin quels sont les circuits qui implémentent ces protocoles (contrôleurs).

Suite à l'étude du chapitre II, l'intérêt s'est naturellement porté sur les méthodes qui permettent la synthèse des contrôleurs. Deux méthodes sont abordées au chapitre III. L'une est basée sur les graphes de transitions de signaux et l'autre sur l'expansion des communications. Elles sont appliquées sur les protocoles disponibles dans l'outil de synthèse TAST.

Le quatrième chapitre présente les structures de base permettant de réaliser tout type de circuits asynchrones et propose les circuits correspondants. De cette présentation se dégage, un premier ensemble de cellules spécifiques (portes de Muller) indispensables à l'implantation de ces structures et par conséquent à la synthèse de circuits asynchrones.

Le chapitre V décrit les différents types de bibliothèques développés pour l'outil TAST. Une nomenclature est d'abord définie pour nommer les cellules de façon unique quelque soit la bibliothèque. Ensuite, la description des différents modèles est effectuée :

- un modèle fonctionnel indépendant de la technologie pour une première validation par simulation du circuit synthétisé,
- un modèle structurel à base de cellules standard pour les circuits de type ASIC,
- un modèle structurel à base de LUT (« Look Up Table ») pour les circuits de type FPGA.

Le dernier chapitre aborde les problèmes d'arbitrage dans les bus de communication des systèmes mono puce (SoC). La logique asynchrone se prête bien à la modélisation de ces systèmes d'arbitrage. En effet, le langage CHP permet de spécifier des comportements indéterministes utilisés pour modéliser les arbitres. Différents algorithmes d'arbitrage sont modélisés en CHP et synthétisés. Les circuits qui en découlent présentent une structure très modulaire qui repose sur deux éléments clés que sont le synchroniseur et le bloc de mutuelle exclusion. Au final, ces éléments viennent enrichir les bibliothèques de cellules asynchrones.

Chapitre I

Concepts fondamentaux

La conception de circuits synchrones repose fondamentalement sur la présence d'un signal unique qui gère les communications dans tout le système de manière globale : l'horloge. Ce signal détermine quand les unités du chemin de données doivent fonctionner. La fréquence de cette horloge est calculée en connaissance du délai le plus long des opérations qui peuvent être lancées dans un cycle. De ce fait, on ne sait pas quand une opération est terminée, mais par contre, on est sûr qu'à la fin du cycle, elle est réellement terminée. Il y a donc une synchronisation implicite de toutes les unités à la fin de chaque cycle.

Par opposition, les circuits asynchrones, n'utilisent pas de signal d'horloge. La gestion des communications se fait de manière locale par une synchronisation entre blocs fonctionnels. Un des concepts les plus importants dans les circuits asynchrones est le contrôle distribué. Cela signifie que chaque unité se synchronise seulement avec les blocs dont la synchronisation a une signification fonctionnelle. De plus, elle se fait uniquement quand elle doit avoir lieu, indépendamment du reste du système. C'est pour cela que chaque bloc d'un circuit asynchrone possède des signaux explicites de synchronisation qui exécutent des protocoles de type requête/acquittement ou « poignée de main » avec les blocs voisins (figure 2).

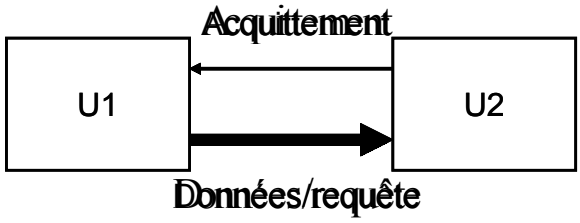


figure 2. Exemple de communication de type « poignée de main ».

Une traduction possible du dialogue qui se fait entre les deux unités serait celle-ci :

U2 : Je suis prêt pour recevoir des données.

U1 : J'émets des données.

Silence, pendant un certain temps.

U2 : J'ai fini, d'utiliser les données.

U1 : Très bien, je les enlève.

Par ce mécanisme, le bon fonctionnement du système devient indépendant du temps de calcul de chaque opération dans les blocs. Plus précisément, le circuit ne dépendra pas de la

distribution des retards dans les portes et les connexions. Cependant, la synchronisation dépend de certaines hypothèses simplificatrices d'un point de vue fonctionnel. Ces hypothèses traduisent le mode de fonctionnement du circuit asynchrone.

Les modes de fonctionnement gouvernent l'interaction entre un bloc asynchrone et son environnement. Certaines hypothèses sur ces modes peuvent simplifier la conception du circuit. De plus ces modes de fonctionnement vont permettre de classifier les circuits asynchrones.

D'un point de vue conception, le principal challenge est de concevoir un système sans aléa. En effet, dans un circuit synchrone, la discrétisation du temps, implémentée par un échantillonnage par le signal d'horloge, permet d'ignorer tous les phénomènes transitoires qui peuvent survenir dans les parties logiques combinatoires (« glitchs »). En asynchrone, par contre, toute transition d'un signal peut être interprétée comme un événement porteur d'information. La principale difficulté réside donc dans la conception de parties combinatoires ou séquentielles sans aléa ou « hazard free » [CORT 02]. Tout changement superflu ou un ensemble de changements superflus est considéré comme un aléa.

Pour analyser et éviter ces aléas, un modèle de délais doit être utilisé. Ce modèle doit justement et précisément capturer le comportement physique du circuit au niveau des portes et des interconnexions.

Ce chapitre présente les concepts fondamentaux qui régissent le fonctionnement des circuits asynchrones et qui permettent de les différencier. Nous présenterons dans un premier temps, les différents types d'aléas que l'on devra prendre en compte au cours de la synthèse. Ensuite nous définirons la notion de délais et la notion de mode de fonctionnement. Cela nous permettra d'effectuer une classification des circuits asynchrones.

1. Génération d'aléas

Un circuit qui se trouve dans un état stable ne produit pas spontanément d'aléas sur ses sorties. La génération d'aléas est liée à l'évolution du circuit. Elle est donc liée à la dynamique des signaux d'entrées et aux délais des constituants du circuit (les portes et les fils).

Supprimer ou plutôt éviter les aléas nécessite donc de définir le mode d'interaction entre le circuit et l'environnement (dynamique des signaux d'entrée) et de caractériser les délais dans les constituants du circuit.

La présence d'aléas dans un circuit peut être détectée à différents niveaux de la conception. Un séquenceur peut, par exemple, contenir des aléas fonctionnels qui trouvent leur origine dans la spécification de la fonction elle-même. Si ce problème est réglé, il peut encore apparaître des aléas au niveau de l'implémentation. Cela signifie que la conception d'un circuit asynchrone ne s'arrête pas après la spécification. Il faut encore s'assurer que les techniques de réalisation choisies sont exemptes d'aléas.

Les paragraphes suivants présentent les différents types d'aléas dont tout circuit asynchrone ne faisant aucune hypothèse temporelle doit se prémunir.

1.1. Les aléas combinatoires logiques

Ce sont des aléas dont l'apparition dépend de la distribution des délais dans la logique. C'est le cas des aléas combinatoires logiques si l'apparition de l'aléa dépend de l'implémentation de la fonction logique. L'implémentation est donc de toute première importance en asynchrone et nécessite la conception d'outils qui intègrent la notion d'aléas. En particulier l'étape de projection technologique « Technology Mapping » est spécifique à la conception de circuits asynchrones. L'implémentation d'une fonction logique à l'aide de portes logiques de base doit être contrôlée pour ne pas générer d'aléas combinatoires logiques. Deux types d'aléas se présentent dans cette catégorie : les aléas statiques et les aléas dynamiques.

1.1.1. Les aléas statiques

Lorsqu'un signal doit rester constant et qu'il change de niveau deux fois successivement on parle d'aléa statique. Cet aléa est aussi appelé « spike » dans le jargon du concepteur. Par exemple, toute porte « ET » et « OU » est susceptible de générer ce type d'aléas si deux entrées changent simultanément (cf. figure 3).

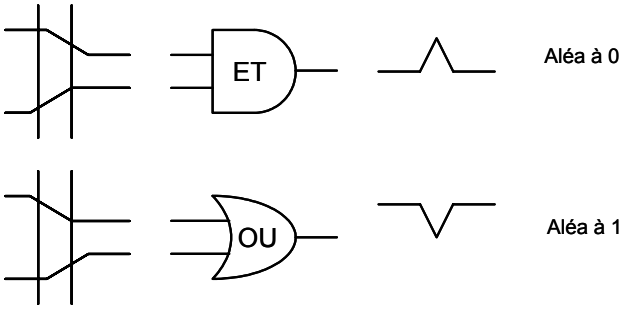


figure 3. Aléas statiques réalisés par des portes logiques ET-OU.

1.1.2. Les aléas dynamiques

Un signal devant changer une seule fois et changeant de niveau plusieurs fois subit des aléas dynamiques. La figure 4, illustre un signal passant à '0', respectivement à '1', qui subit un aléa dynamique à la descente, respectivement à la monté.

figure 4. Aléas dynamiques.

1.2. Les aléas combinatoires fonctionnels

Ce sont des aléas qui peuvent être détectés et supprimés à un niveau plus élevé de la conception en étudiant et en modifiant la spécification logique de la fonction. Il faut une spécification exempte d'aléa fonctionnel pour obtenir une réalisation sans aléa si aucune hypothèse n'est faite sur les délais des éléments.

1.3. Les aléas séquentiels

Ces aléas trouvent leur origine dans les signaux bouclés. Ces aléas sont détectables lors de la spécification d'un problème, par exemple lors de l'étude d'une table de flots. On cite souvent les aléas séquentiels qui peuvent être détectés en faisant changer la même entrée une fois puis trois fois. Il y a aléa séquentiel si l'état final n'est pas le même dans les deux cas.

En conclusion, il est à remarquer que les circuits asynchrones requièrent une conception attentive et fine de toutes les parties (combinatoires et séquentielles) du circuit. Par la suite, il est vu que l'indépendance aux délais et la contrainte de conception sans aléa, sont plus ou moins respectées en fonction de la classe de circuits asynchrones à laquelle on s'intéresse.

1.4. Les aléas de type SIC

Ici, l'environnement effectue un changement unique sur une seule des entrées du circuit et attend que la sortie se stabilise avant tout autre changement. Les aléas se produisant dans ce type de comportement sont de type SIC, de l'anglais « Single Input Change ».

Soit le tableau de Karnaugh suivant, figure 5. Le but ici est de réaliser le circuit logique correspondant.

bc \ a	0	1
00	0	0
01	0	1
11	1	1
10	1	0

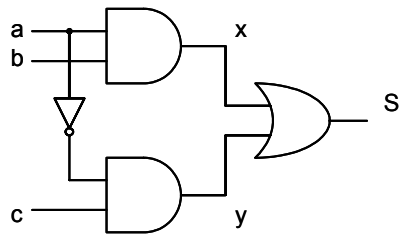


figure 5. $S = a.c + \overline{a}.b$.

Dans cet exemple, les regroupements sont suffisants pour implémenter un circuit à deux niveaux de type ET-OU (somme de produits). Mais le passage de '0' à '1' de l'entrée « a » alors que les autres entrées sont à '1' peut provoquer un aléa statique à '1'. Le chronogramme de la figure 6 présente un aléa statique à '1'.

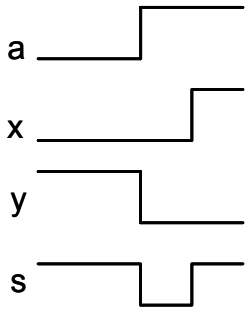


figure 6. Aléa statique à ‘1’.

La solution, dans ce cas, pour éviter cet aléa, est de compléter les regroupements précédents par le couple « bc ». A ce moment là, la sortie S ne dépend plus de l’ordre des changements de x et de y car lorsque b = c = ‘1’, alors, S = ‘1’ et reste à ‘1’ (figure 7). Il est vrai que « bc » est redondant vis-à-vis de la fonction S.

bc \ a	0	1
00	0	0
01	0	1
11	1	1
10	1	0

figure 7. S = $a.c + \overline{a}.b + b.c$.

En conclusion, il a été montré [WAKE 94] que pour une implémentation quelconque sous forme ET-OU, aucun aléa ne peut survenir lors des transitions $0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$ sur la sortie. Ainsi, la synthèse de circuits ET-OU sans aléa nécessite seulement d’éliminer les aléas statiques à ‘1’.

1.5. Aléas de type MIC

L’appellation MIC provient de l’acronyme anglais « Multiple Input Change ». Lorsque plusieurs entrées changent, les aléas statiques et dynamiques peuvent être rencontrés.

Soit M la valeur de départ du vecteur des entrées et N la valeur d’arrivée du vecteur des entrées. Une transition est le passage des entrées de l’état M à l’état N. Et les changements sur les entrées sont supposés monotones.

Considérons le tableau de Karnaugh de la figure 8 comme exemple :

cd \ ab	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	0	0

figure 8. $S = \overline{c}.\overline{d} + \overline{a}.\overline{d} + b.d$.

La transition de M = ''0100'' vers N = ''0111'' peut provoquer un aléa statique à '1' (cf. figure 9).

La solution, pour éviter ce type d'aléa avec changement multiple des entrées, est de rajouter le regroupement « ¬a.b ». Dans les circuits logiques ET-OU à deux niveaux il n'y a pas d'aléa statique à '0' avec présence de changements multiples.

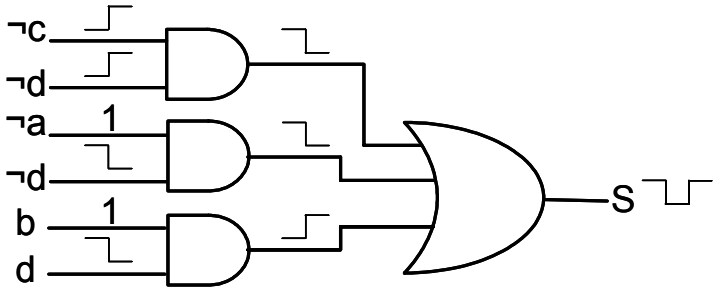


figure 9. Aléa statique à '1'.

Maintenant considérons une nouvelle transition sur le circuit corrigé. A la figure 10, la transition de départ est M = ''0111'' et le vecteur N = ''1110'' est appliqué aux entrées.

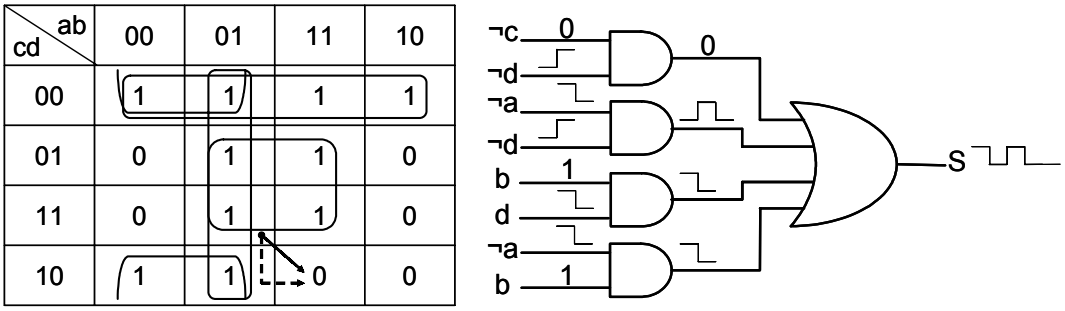


figure 10. $S = \overline{c}.\overline{d} + \overline{a}.\overline{d} + b.d + \overline{a}.b$.

C'est le cas d'un aléa dynamique à '0' car cette transition va activer le terme « ¬a.¬d » et le désactiver.

La solution pour corriger cet aléa est de renforcer le monôme « ¬a.¬d » avec « ¬b ». La transition considérée changent les entrées « a » et « d » le monôme « ¬a.¬d » est susceptible de générer cet aléa dynamique et le renforcement proposé force le monôme à '0' quelque soit l'ordre du changement. Le circuit synthétisé serait alors celui de la figure 11.

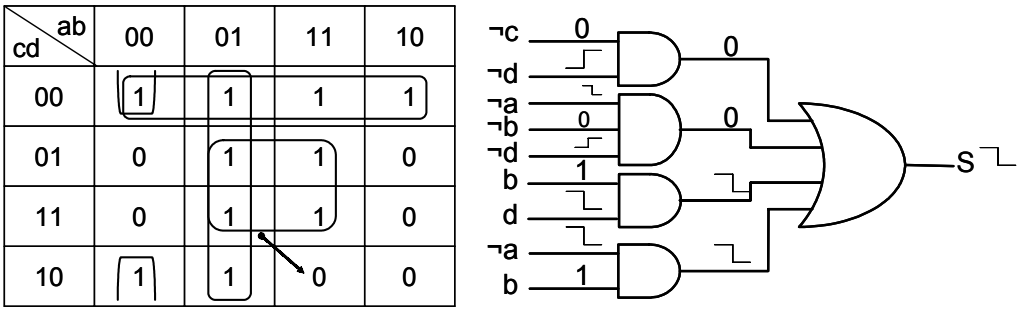


figure 11. $S = \overline{c}.\overline{d} + \overline{a}.\overline{b}.\overline{d} + b.d + \overline{a}.b$.

Pour conclure, la suppression d'aléas est donc un problème de couverture du tableau de Karnaugh :

- Les transitions $1 \rightarrow 1$ doivent être couvertes.
- Les transitions $1 \rightarrow 0$ et $0 \rightarrow 1$ un produit qui rencontre la transition doit contenir le produit de départ ou le produit d'arrivée.
- Les transitions $0 \rightarrow 0$ ne génèrent pas d'aléas dans les réalisations ET-OU.

Ces conditions suffisent pour éliminer n'importe quel aléa de type MIC. Si il existe plusieurs aléas de type MIC, le problème de couverture peut très bien ne pas avoir de solution.

Pour un ensemble de transitions MIC (fonctions de transitions d'une machine à état) l'existence d'une couverture de Karnaugh conduisant à une réalisation ET-OU sans aléa insensible aux délais n'est pas assurée.

2. Les modèles de délais, de circuits et d'environnements

2.1. Modèles de délais

Les modèles qui régissent le comportement des délais interviennent dans la caractérisation des circuits asynchrones.

Les délais sont communément caractérisés par un comportement dit « pur » ou « inertiel ». Un délai « pur » décale dans le temps toute forme d'onde d'un signal de la valeur de ce retard. En VHDL, ce type de délais est appelé « transport delay ». Cependant cela n'est pas très réaliste car cela implique que les portes et les pistes du circuit possèdent une bande passante infinie. Un modèle de délai plus réaliste est le délai « inertiel ». En plus de décaler la forme d'onde du signal retardé, celui-ci supprime les impulsions courtes (certains aléas). Le

modele de délai inertiel utilisé en VHDL se caractérise par deux paramètres : le temps de décalage (« delay time ») et le temps de rejet (« reject time »). Ainsi les impulsions plus courtes que le temps de rejet seront alors filtrées. Ce modèle de délais est celui utilisé par défaut en langage VHDL.

Les délais peuvent aussi être caractérisés vis à vis du temps.

Le plus simple est le délai « fixe ». Il possède alors une valeur constante. Ensuite, on trouve le délai « borné ». Ce dernier est compris entre une limite supérieure et une limite inférieure qui sont connues ($T_{min} \leq$ délai $\leq T_{max}$). Enfin, le délai peut être « non borné » où sa valeur est comprise dans un intervalle dont les limites sont inconnues. Cela dit, sa valeur est finie mais non connue ($0 <$ délai $< \infty$). Par exemple ce modèle de délai est utilisé pour les portes dans les circuits indépendants de la vitesse.

2.2. Modes de fonctionnement

En plus des délais dans les portes et les pistes, il est aussi nécessaire de caractériser le comportement du circuit à synthétiser dans son environnement.

2.2.1. Mode fondamental

Il est basé sur le travail de Huffman [HUFF 64] et Unger [UNGE 69]. Un circuit et son environnement, qui généralement représentent les autres circuits modélisés de façon arbitraire, suivent un protocole deux phases qui est une réminiscence du mode de fonctionnement d'un circuit synchrone. D'abord la phase communication se réalise : quand l'environnement reçoit les sorties du circuit, il lui envoie de nouvelles données. Ensuite c'est la phase de calcul, lorsque le circuit réagit aux nouvelles entrées en produisant de nouvelles sorties. Pendant ce temps, l'environnement n'est pas autorisé à produire de nouvelles entrées pour le circuit. Il existe plusieurs définitions exprimant le fait qu'une phase se termine et que l'autre se poursuit.

Voici le fonctionnement d'un circuit en mode fondamental. Le circuit est dans un état stable : toutes les entrées, les sorties, les signaux externes sont stables. Dans cet état, l'environnement est autorisé à changer une entrée et une seule. C'est le cas d'aléas de type SIC. L'environnement ne peut changer une entrée à nouveau que lorsque le circuit est revenu dans un état stable après avoir produit une sortie.

Un problème se pose : Comment informer l'environnement de la stabilité du circuit ?

Une hypothèse temporelle est faite : le plus long délai pour stabiliser le circuit est calculé et l'environnement doit respecter ce délai. Dans ce cas le modèle de délai est constant et fixe. Le fait de calculer le plus long délai implique de borner les délais des portes et des pistes. Les circuits qui correspondent à ce type de fonctionnement sont les circuits de Huffman qui datent des années 50 (hasard SIC et modèle de délai borné).

Généralement, les circuits fonctionnant en mode fondamental sont spécifiés au moyen de machines d'états finis (tables de flot). Ces machines à états sont très proches de celles utilisées pour la conception synchrone.

L'extension de ce type de circuits à des changements multiples des entrées et des sorties sont des circuits qui fonctionnent en mode rafale (« Burst Mode »). Le modèle de délai reste borné pour les circuits en mode « BURST ».

2.2.2. Mode entrée/sortie

Ce mode de fonctionnement est totalement différent du précédent. Les phases de communication et de calcul peuvent indifféremment se recouvrir arbitrairement. La seule contrainte est l'ensemble des protocoles entre l'environnement et le circuit.

Pour les circuits fonctionnant en mode entrée/sortie, on suppose que le circuit est dans un état stable. L'environnement peut changer les entrées ; quand le circuit a produit les sorties correspondantes, l'environnement est autorisé à changer les entrées à nouveau. Il n'y a pas d'hypothèses sur les signaux internes, et les changements des entrées peuvent survenir alors que le circuit n'est pas encore stabilisé. Les seules restrictions que l'on applique sur l'environnement sont des relations de causalité entre des transitions sur les entrées et les sorties. Pour cette raison les circuits sont souvent spécifiés par des méthodes basées sur les graphes dans lesquels le concepteur précise toutes les séquences possibles des signaux d'entrée et de sortie qui peuvent être observées au niveau de l'interface du circuit. Ces circuits sont dits insensibles aux délais et sont connus sous le nom de circuits de Muller (1950).

3. Les classes de circuits asynchrones

La plupart des études sur les circuits asynchrones visent à établir '1' compromis entre robustesse et complexité [VIVE 01]. La figure 12 présente la terminologie habituellement utilisée pour qualifier les circuits asynchrones. Cette classification est effectuée suivant le modèle de délai et le modèle d'environnement choisis.

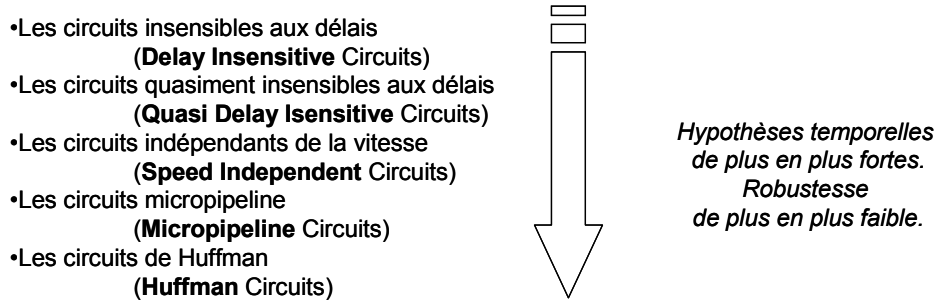


figure 12. Terminologie des différentes classes de circuits asynchrones.

Le terme de circuits auto séquencés (« self timed ») est souvent utilisé pour désigner les circuits asynchrones mais il ne correspond en fait à aucune définition précise. Il désigne simplement des circuits dont l'enchaînement des étapes est déterminé par les signaux internes

du circuit plutôt que par un signal d'horloge. Il qualifie donc les circuits non synchrones plutôt qu'il ne renseigne sur la technique asynchrone mise en œuvre.

3.1. Les circuits insensibles aux délais

Cette classe de circuits utilise un mode de fonctionnement purement asynchrone. Aucune hypothèse temporelle n'est introduite, c'est-à-dire qu'ils sont fonctionnellement corrects indépendamment des délais introduits par les fils ou les éléments logiques. D'un point de vue théorique, cela signifie qu'ils sont basés sur un modèle de délai pour les fils et les éléments qui est « non borné ».

Ainsi, il est supposé qu'un circuit répondra toujours correctement à une sollicitation externe pourvu qu'il ait assez de temps pour calculer. Ceci impose donc au récepteur d'un signal de toujours informer l'expéditeur que l'information a été reçue. Les circuits récepteurs doivent donc être capables de détecter la réception d'une entrée et/ou la fin de son traitement. Les circuits émetteurs doivent attendre un compte rendu avant d'émettre une nouvelle donnée. Les contraintes de réalisation pratique, qu'impose l'utilisation de ce modèle, sont très fortes. De plus, la plupart des circuits conçus aujourd'hui utilisent des portes logiques à une seule sortie. L'adoption du modèle de délai non borné ne permet pas leur utilisation. En effet, les portes logiques standard ont leurs sorties qui peuvent changer si une seule de leurs entrées change. Comme nous l'avons souligné, toutes les composantes de ce type de circuits doivent s'assurer du changement des entrées avant de produire une sortie. Si la sortie change alors qu'une seule des entrées change, seul le changement de l'entrée active est acquittée, sans pouvoir tester l'activité des autres entrées. La seule porte à une sortie qui respecte cette règle est la porte de Muller. Malheureusement, les fonctions réalisables avec seulement des portes de Muller sont très limitées.

La seule solution est d'avoir recours à un modèle de circuits de type « portes complexes » pour les composants élémentaires. Dans ce cas la construction de ces circuits se fait à partir de composants standard plus complexes que de simples portes logiques et qui peuvent posséder plusieurs entrées et plusieurs sorties [HAUC 95], [EBER 91].

3.2. Les circuits quasiment insensibles aux délais

Cette classe de circuits adopte le même modèle de délai non borné pour les connexions mais la notion de fourche isochrone (« isochronic fork ») est ajoutée.

Une « fourche » est un fil qui connecte un expéditeur unique à deux récepteurs. Elle est qualifiée d'isochrone lorsque les délais entre l'expéditeur et les récepteurs sont identiques. Cette hypothèse à des conséquences importantes sur le modèle et les réalisations possibles. Elle résout notamment le problème de l'utilisation de portes logiques à une seule sortie. Car si les fourches sont isochrones, il est permis de ne tester qu'une branche d'une fourche en supposant que le signal s'est propagé de la même façon dans l'autre branche. En conséquence, on peut autoriser l'acquittement d'une des branches de la fourche isochrone.

A.J.Martin a montré [MART 93] que l'hypothèse temporelle de fourche isochrone est la plus faible à ajouter aux circuits insensibles aux délais pour les rendre réalisables avec des portes à plusieurs entrées et une seule sortie. Ainsi les circuits quasi insensibles aux délais se caractérisent par l'adoption d'un modèle de délais pour les connexions qui est de type non borné. En plus, l'hypothèse de fourche isochrone, et un modèle de type porte simple à délai non borné pour les composants élémentaires du circuit. Les circuits quasi insensibles aux délais sont donc réalisables avec des cellules standard telles qu'on les utilise pour la conception de circuits synchrones.

En pratique, la contrainte de fourche isochrone est assez faible, et est facilement satisfaite par une conception soignée, en particulier au niveau du routage et des seuils de commutation. Il suffit, finalement, que la dispersion des temps de propagation jusqu'aux extrémités de la fourche soit inférieure aux délais des opérateurs qui lui sont connectés (au minimum une porte et un fil dont une sortie interagit avec la fourche [MART 90], [BERK 93]).

3.3. Les circuits indépendants de la vitesse

Les circuits indépendants de la vitesse font l'hypothèse que les délais dans les fils sont négligeables, tout en conservant un modèle non borné, pour les délais dans les portes. Ce modèle, qui n'est pas réaliste dans les technologies actuelles, est en pratique équivalent au modèle QDI à l'exception des fourches qui sont toutes considérées comme isochrones. Même si pendant longtemps la communauté a tenté de cerner les différences entre ces deux modèles, il y a aujourd'hui un consensus pour considérer les modèles QDI et SI équivalents. Hauck [HAUC 95] montre avec le schéma de la figure 13 comment une fourche isochrone peut être représentée par un circuit indépendant de la vitesse.

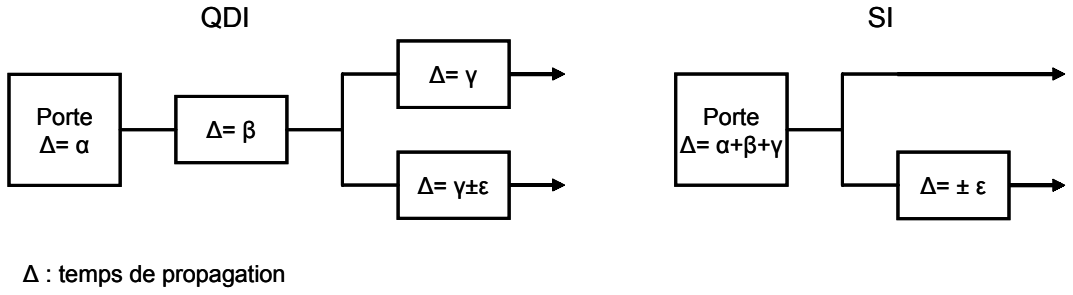


figure 13. Equivalence entre les modèles QDI et SI.

Cependant, il semble plus pertinent d'utiliser le modèle QDI car celui-ci identifie les fourches isochrones de celles qui ne le sont pas. Ceci offre le moyen de vérifier les connexions du circuit qui ne sont pas insensibles aux délais au cours des différentes étapes de conception, et seulement celles-ci.

3.4. Les circuits Micro pipeline

La technique micro pipeline a été introduite par Ivan Sutherland [SUTH 89]. Les circuits de cette classe sont composés de parties de contrôle insensibles aux délais qui commandent des chemins de données conçus en utilisant le modèle de délais bornés. La structure de base de cette classe de circuits est le contrôle d'une file (FIFO) (figure 14). Elle se compose d'éléments identiques connectés tête-bêche. Les opérateurs notés « C » sont des portes de Muller dont la sortie est une copie des niveaux d'entrée lorsqu'ils sont identiques, et une copie du niveau de sortie précédent lorsque les entrées sont différentes.

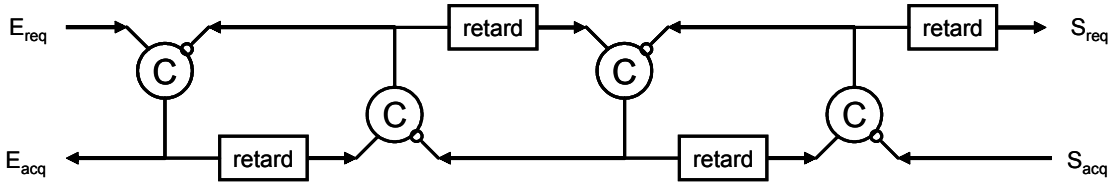


figure 14. Structure de base des circuits micro pipelines.

Le circuit réagit à des transitions de signaux et non pas à des états (protocole deux phases). On parle également d'une logique à évènements. Chaque transition étant associée à un évènement. Ainsi, tous les signaux sont supposés à '0' initialement. Une transition positive sur E_{req} provoque une transition positive sur E_{acq} qui se propage également à l'étage suivant. Le deuxième étage produit une transition positive qui d'une part, se propage à l'étage suivant mais qui d'autre part, revient au premier étage l'autorisant à traiter une transition négative cette fois. Les transitions de signaux se propagent donc dans la structure tant qu'elles ne rencontrent pas une cellule « occupée ». C'est un fonctionnement de type FIFO.

Cette structure peut être enrichie d'opérateurs de mémorisation ou pipeline, et d'opérateurs de traitement combinatoires [SUTH 89]. La figure 15 illustre un micro pipeline avec traitement.

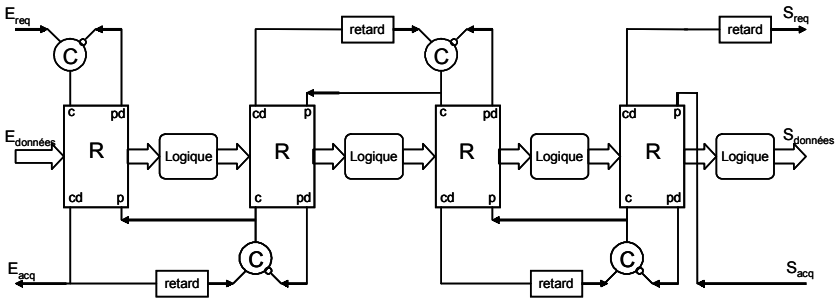


figure 15. Structure micro pipeline avec traitement et registre.

Dans cette structure, les opérateurs dénotés « R » sont des registres qui capturent la donnée entrante sur l'occurrence du signal C. Ils produisent le signal « CD » lorsque la donnée est mémorisée. En pratique, CD est une version retardée du signal C. Durant cette phase, la donnée précédemment mémorisée dans le registre est maintenue en sortie. Lorsque P est actif, le registre laisse passer la donnée d'entrée à la sortie. Le signal « Pd » signale que le registre est bien transparent. La structure figure 15, dépouillée de la logique, réalise une FIFO

(initialement les registres laissent passer les données). Avec la logique la structure est celle d'un circuit asynchrone « pipeliné ».

Le protocole de communication utilisé ici, est de type « données groupées » ([SUTH 89]). Lors d'une occurrence de $E_{req,}$ les données d'entrée sont stockées dans le premier étage. E_{req} est propagé vers l'étage suivant qui conserve le résultat transmis par la logique du premier étage. La propagation de E_{req} est retardée de façon à s'assurer que la logique combinatoire a bien convergé avant la capture du résultat dans le deuxième étage. La capture du deuxième étage étant effectuée, le premier registre est rendu passant ce qui permet le traitement de la donnée présente dans le premier étage et autorise la prise en compte d'un nouvel évènement sur E_{req} du premier étage.

La motivation première pour le développement de cette classe de circuits était de permettre un pipeline élastique. En effet, le nombre de données présentes dans le circuit peut être variable, les données progressant dans le circuit aussi loin que possible en fonction du nombre d'étages disponibles ou vides. Cependant, ce type de circuits révèle un certain nombre d'inconvénients.

Tout d'abord, il faut remarquer que les problèmes d'aléas ont été écartés en ajoutant des retards sur les signaux de contrôle. Cela permet en fait de se ramener à un fonctionnement en temps discret dans lequel il est autorisé la mémorisation des données seulement lorsqu'elles sont stables (à la sortie des portes combinatoires). Les délais étant de durée fixe dans la proposition initiale de Sutherland, ces circuits effectuent les traitements en pire cas. Il n'est donc pas possible de tirer parti de la variation dynamique de la chaîne critique des opérateurs de traitement.

Il est possible cependant de s'affranchir de cette contrainte en utilisant des registres et des opérateurs combinatoires capables de générer leur propre signal de fin de calcul sans avoir recours à des délais fixes. Alors il est obtenu des structures du type de celles implémentées figure 16 dans laquelle les délais fixes sont remplacés par des délais variables implémentés dans les registres et la logique combinatoire. Ici, le temps de calcul peut varier en fonction de données. On sort alors du cadre des circuits utilisant un modèle de délai borné et le circuit peut être rendu indépendant de la vitesse puisque la seule hypothèse temporelle consiste à assurer que l'occurrence du signal de contrôle, E_{req}, succède les données. Cette hypothèse peut être rendue équivalente à supposer des délais nuls dans les connexions entre les étages.

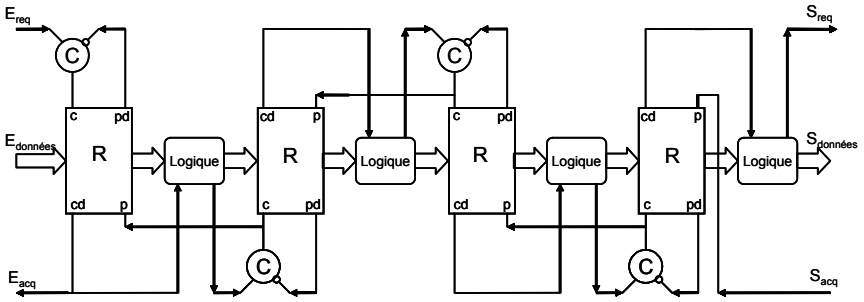


figure 16. Structure micro pipeline indépendant de la vitesse.

3.5. Les circuits de Huffman

Les circuits de cette classe utilisent un modèle de délai identique aux circuits synchrones. Ils supposent que les délais dans tous les éléments du circuit et dans les connexions sont bornés ou même de valeurs connues. Les hypothèses temporelles sont donc du même ordre que pour la conception de circuits synchrones. Leur conception repose sur l'analyse des délais dans tous les chemins et boucles de façon à dimensionner les signaux de contrôles locaux qui s'apparentent d'avantage ici à des horloges locales. Ces circuits sont d'autant plus difficiles à concevoir et à caractériser qu'une faute de conception ou de délai les rend totalement non fonctionnels.

4. Conclusion

La présentation des différentes classes de circuits asynchrones vient d'être faite. Celles-ci proposent, dans leur mode de fonctionnement, des hypothèses temporelles plus ou moins fortes. Plus cette contrainte est relâchée, moins le circuit est robuste d'un point de vue comportemental. Mais, l'ensemble des fonctionnalités implantables est plus grand. Pour chacune de ces classes, il peut être associé une ou plusieurs méthodologies de conceptions développées par différents groupes de recherche de la communauté asynchrone. Ceci est le reflet du vaste spectre de solutions qu'offre le relâchement des synchronisations. A ce jour, il n'existe pas de vision unificatrice de ces différentes méthodologies de conceptions. Ainsi, il n'y a pas de consensus sur le type de circuits asynchrones le plus approprié : cela dépend des objectifs et des moyens visés. Si les avantages présentés sont considérés dans ce chapitre les circuits QDI, SI et micro pipeline semblent les mieux adaptés pour concevoir des systèmes robustes dans les technologies actuelles tout en visant de bonnes performances.

Dans le flot de conception TAST seule les circuits quasiment insensibles aux délais et micro pipeline sont ciblés. Dans la suite du manuscrit, seules ces classes de circuits sont concernées.

Chapitre II

Canaux de communication : état de l'art

Quelque soit la classe de circuits asynchrones visée, la synchronisation entre les blocs fonctionnels du système est primordiale. L'élément qui permet cette synchronisation est le canal de communication. Il est, de ce fait, l'élément fondamental des systèmes asynchrones. La façon de communiquer dans le canal est définie par un protocole qui spécifie l'échange d'informations entre deux blocs.

Il existe dans la littérature un ensemble de protocoles dédiés aux circuits asynchrones. Dans ce chapitre, une présentation détaillée d'un point de vue modèle de spécification et l'implémentation des canaux est faite. Une attention particulière est portée à l'unification des différentes approches proposées dans la littérature.

1. Définitions générales

Ce paragraphe introduit les notions qui permettent de définir quels sont les différents moyens de caractériser et d'implémenter les canaux de communication.

1.1. Le canal de communication

Un canal de communication permet à deux entités fonctionnelles, ou processus d'échanger des données par une connexion point à point (il est également possible de concevoir des schémas de communication multi points plus complexes). Afin de maintenir une certaine cohésion de fonctionnement vis-à-vis de la spécification initiale, l'échange des informations à travers ce canal doit respecter certaines règles : le protocole de communication.

1.1.1. Définition du canal de communication

Un processus peut communiquer avec son environnement par l'intermédiaire de canaux de communication. Le canal de communication est l'ensemble :

- des signaux de contrôle utiles pour réaliser la communication (signal de requête et signal d'acquittement)
- du chemin de données qui contient les informations. Le chemin de données peut se contenter de véhiculer les données ou bien il peut transformer celles-ci par l'intermédiaire des blocs de traitement.

Le canal sert non seulement à échanger des données mais aussi à synchroniser les processus communicants entre eux. L'organisation des échanges de données est gérée par le protocole de communication.

1.1.2. Activité sur un canal

La notion d'activité pour un canal de communication définit qui de l'émetteur ou du récepteur est l'initiateur d'un échange d'informations. Cette notion d'activité dépend du côté où l'utilisateur se place (émetteur ou récepteur).

En effet, du point de vue de l'émetteur, si celui-ci est à l'origine d'une communication, le canal est considéré comme actif. Si c'est le récepteur qui demande l'échange, le canal côté émetteur est dit passif. A l'inverse, du point de vue du récepteur, un échange commencé par l'émetteur caractérise le canal comme passif et actif dés que le récepteur demande une communication.

La règle de base est que pour un canal de communication donné, un des deux ports terminaux est choisi « actif » alors que l'autre est choisi « passif ». Cette règle assure que deux processus connectés par un canal ne sont pas tous les deux en attente d'une donnée. Il y aurait alors blocage.

1.2. Notions de pipeline

1.2.1. Définition d'un pipeline

Un pipeline est un procédé qui décompose une opération en plusieurs étages concurrents pour augmenter le débit de sortie du système au prix d'un faible coût en surface. Un pipeline traite et mémorise des données. Tout au long d'un pipeline il y a alternance entre un élément de mémorisation (registre) et un élément de traitement (bloc logique).Un pipeline peut être implémenté de façon synchrone ou asynchrone.

1.2.2. Caractérisation du pipeline

La « latence » est le temps que met une donnée pour traverser tous les étages d'un pipeline.

Le « temps de cycle » est le temps qui s'écoule entre l'arrivée d'une nouvelle donnée dans un étage du pipeline, son traitement et le retour de l'étage du pipeline dans son état initial avant le traitement d'une nouvelle donnée. C'est le temps nécessaire que met un étage pour effectuer une communication entière sur ses entrées et sur ses sorties (exécution du cycle du protocole).

La figure 17 représente le temps de cycle dans un pipeline en considérant le nième étage d'un pipeline. Ici le temps de cycle est la somme des quatre temps de propagation de : T_{cycle} = T1+T2+T3+T4.

$$T1 = E_{req}+ \text{ à } S_{req}+ \text{ (étage n)}$$

$$T2 = E_{req}+ \text{ à } E_{acq}- \text{ (étage n+1)}$$

$$T3 = S_{acq}- \text{ à } E_{acq}+ \text{ (étage n)}$$

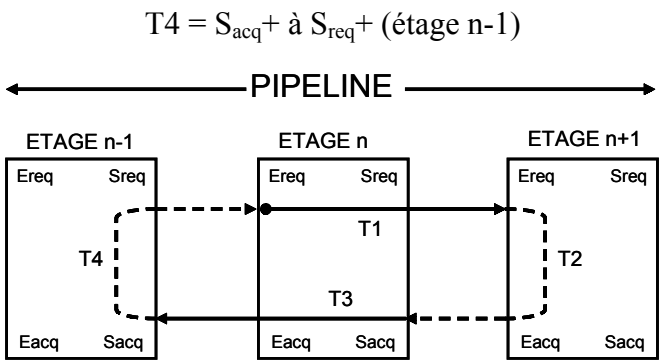


figure 17. Temps de cycle d'un pipeline quatre phases.

Le « rendement » ou « sortance » d'un pipeline est la mesure de la quantité d'informations qui peut traverser le pipeline par unité de temps.

1.2.3. Pipeline synchrone

Dans un pipeline synchrone (cf. figure 18), la communication des données entre les étages s'effectue à l'aide d'un signal global : « l'horloge ». Les données sont communiquées entre les étages simultanément. Le calcul d'un étage doit s'effectuer dans un temps inférieur à celui qui s'écoule entre deux fronts montants d'horloge (la période de l'horloge).

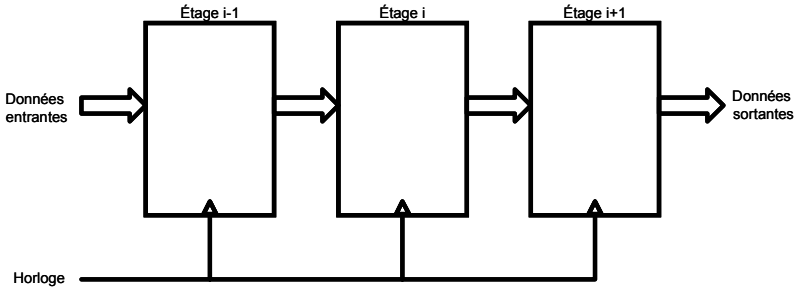


figure 18. Pipeline synchrone.

1.2.4. Pipeline asynchrone

Dans un pipeline asynchrone, figure 19, la communication des données entre les étages est gérée par des signaux locaux par opposition au signal global dans le pipeline synchrone. Quand un étage possède une donnée qu'il souhaite communiquer avec son voisin, il lui envoie un signal de requête. Lorsque le voisin accepte cette nouvelle donnée, il la recueille et envoie un signal d'acquittement à l'émetteur.

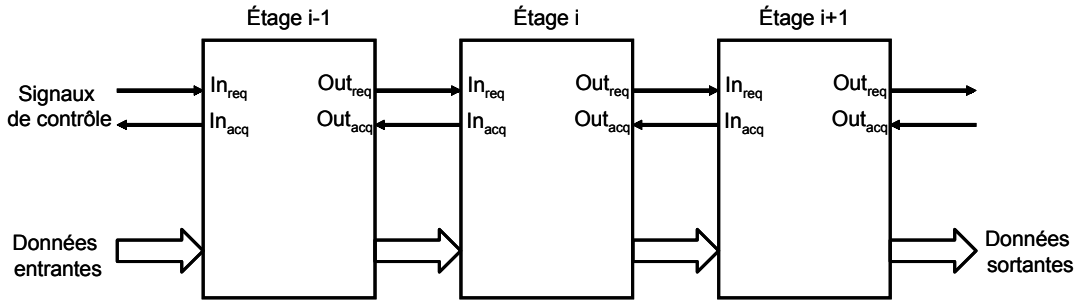


figure 19. Pipeline asynchrone.

Par la suite, les travaux se focalisent sur les pipelines asynchrones. Et chaque étage de pipeline est considéré comme un opérateur asynchrone.

1.2.5. Pipeline linéaire

Un pipeline linéaire est une succession régulière d'étages avec ou sans traitement entre chaque étage.

1.2.6. Pipeline non linéaire

Les pipelines non linéaires possèdent des structures de choix ou de branchement avec ou sans structure de contrôle. Les structures de base sont la fourche, la jonction, le multiplexage et le démultiplexage. Ces différents blocs sont présentés dans la suite du mémoire (chapitre IV). Les pipelines non linéaires sont nécessaires pour réaliser tous les types de circuits asynchrones ([DINH 02b]).

1.2.7. Pipeline inélastique

Un pipeline peut être inélastique : le débit de données rentrant est rigoureusement identique au débit de données sortant. Un registre à décalage est un exemple de pipeline inélastique. De plus les données progressent dans le pipeline de façon totalement synchronisée, elles sont toutes en phases.

1.2.8. Pipeline élastique

Les autres types de pipelines seront considérés comme élastiques : la quantité d'information à l'intérieur d'un pipeline peut varier. Le débit de données en entrée peut donc être différent du débit de données sortant. Une file de type « first in first out » est considérée comme un pipeline élastique. Nous montrerons qu'en fonction de la communication réalisée entre les différents étages d'un pipeline, la quantité d'information que peut contenir celui-ci varie.

1.3. Définition de protocoles

Le protocole est l'ensemble des règles qui régit la communication par un canal entre deux processus, ou entre un circuit et son environnement. Ces protocoles sont classés en fonction du nombre d'étapes que comporte un cycle de communication. On distingue les protocoles deux phases et les protocoles quatre phases.

1.3.1. Protocole deux phases

Ivan Sutherland [SUTH 89] utilise un protocole à deux phases pour contrôler le transfert des données dans un pipeline. Il présente une des premières structures de pipeline asynchrone. Un micro pipeline est un pipeline auto séquencé (asynchrone) sensible aux transitions (suivant initialement un protocole deux phases), avec ou sans traitement. Les points clés de la stratégie du micro pipeline sont la notion de données groupées et le protocole de communication (requête/acquittement) qui gère le transfert de données entre les étages.

L'approche pour la communication est le type « données groupées » :

- Un fil par bit : l'ensemble de ces bits est le chemin de données.
- Deux fils de contrôle : un signal de requête de l'émetteur vers le récepteur et un signal d'acquittement du récepteur vers l'émetteur.

Dans ce type de communication une transition montante ou descendante sur les signaux de contrôle ont la même signification : elles sont appelées évènements (figure 20).

figure 20. Deux transitions correspondent à un seul événement.

La requête émet un évènement quand les données sont valides.

L'acquittement transmet un évènement lorsque les données ont été utilisées.

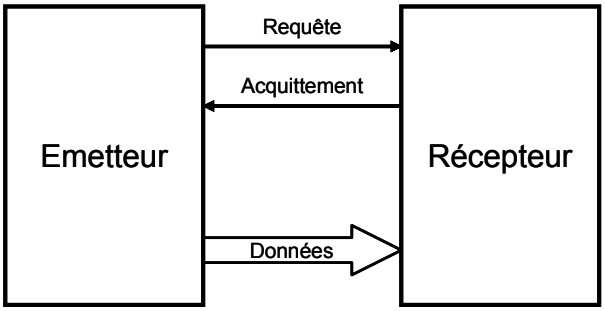


figure 21. Interface de type « données groupées ».

Dans la figure 21, l'émetteur place une donnée valide sur le chemin de données. Ensuite, il produit un évènement sur le fil de requête. Le récepteur prend la donnée et produit un évènement sur le fil d'acquittement [SUTH 89].

Le protocole deux phases a été nommé ainsi par Sutherland. La convention de « données groupées deux phases » signifie que seulement deux phases d'opération sont distinguées :

- La phase active de l'émetteur. Cette phase est terminée par un évènement sur le fil de requête. Elle émet de nouvelles données au récepteur lorsqu'elles sont prêtes.
- La phase active du récepteur. Elle est terminée par un évènement sur le fil d'acquittement. Elle détecte l'arrivée de nouvelles données ainsi qu'un évènement sur le fil de requête.

La figure 22 représente le diagramme temporel du protocole deux phases.

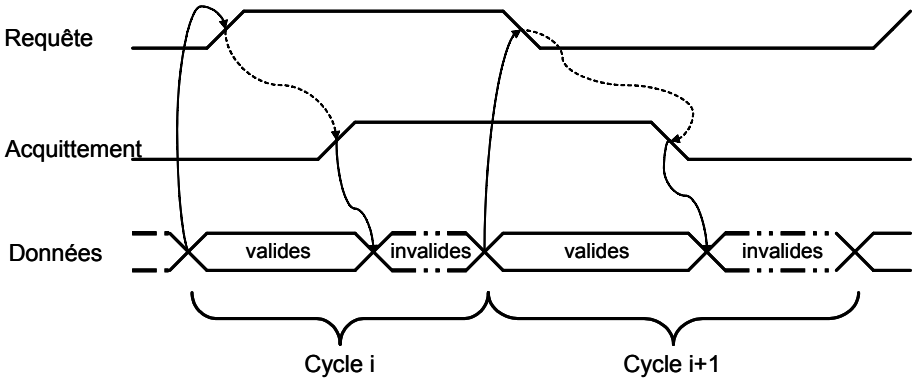


figure 22. Protocole deux phases.

Un évènement termine chaque phase du protocole. Pendant la phase active de l'émetteur, celui-ci est libre de changer les données. Il doit les maintenir pendant la phase active du récepteur (jusqu'à un évènement sur l'acquittement).

Le terme « données groupées » est défini dans [SUTH 89] : il traduit le traitement groupé du chemin de données et du signal de requête. Le délai, dans la transmission de données, doit être inférieur au délai dans la transmission de l'évènement sur la requête. Le signal d'acquittement n'a pas besoin d'être borné.

1.3.2. Protocole quatre phases

La différence principale avec un protocole deux phases est que chaque signal de contrôle à besoin d'une phase de réinitialisation. La communication est maintenant sensible à des niveaux logiques et non pas à des transitions. Par contre, le type de transition (montante ou descendante) doit être distingué et possède une signification différente en fonction du protocole que l'on souhaite appliquer pour une communication.

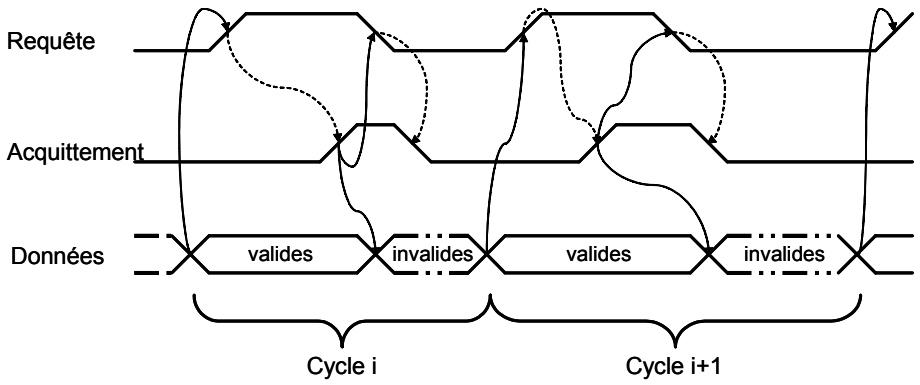


figure 23. Protocole quatre phases.

- Première phase : Le récepteur détecte une nouvelle donnée et le signal de requête. Il la traite et génère un signal d'acquittement.
- Deuxième phase : L'émetteur détecte le signal d'acquittement et invalide la donnée (retour à zéro du signal de requête).
- Troisième phase : Le récepteur détecte le retour à zéro du signal de requête. Il remet le signal d'acquittement dans son état invalide.
- Quatrième phase : L'émetteur après avoir reçu l'invalidité de l'acquittement peut envoyer une nouvelle donnée et un nouveau signal de requête.

Il existe d'autres variantes du protocole quatre phases. Ces différentes variantes dépendent soit de la durée de validité de la donnée au cours de la communication, soit de la façon de gérer les phases d'échange de l'information et de réinitialisation de la communication. Pour chacun de ces différents protocoles correspond un circuit de contrôle particulier.

1.4. Modèles de spécification d'un protocole

La communication dans un canal dépend donc d'un protocole. Les différentes étapes qui gèrent un protocole peuvent être décrites en utilisant différents formalismes. Ici la présentation des formalismes et non les règles qui garantissent que l'implémentation du protocole est correcte, est abordée. Ces règles sont décrites dans le chapitre sur les différentes méthodes de synthèse des circuits asynchrones (chapitre III).

1.4.1. Diagramme des temps

C'est le plus ancien moyen de spécifier un protocole. Il est encore largement utilisé par les concepteurs. Les transitions sont des fronts (montant ou descendant) et des flèches expriment la causalité entre les transitions. Ce formalisme représente la communication sous la forme d'un chronogramme. La figure 23 est un exemple de diagramme des temps.

1.4.2. Graphe de transition des signaux

Le graphe de transition des signaux ou STG (« Signal Transition Graph » en anglais) a été introduit par Chu, Leug et Wanuga [CHU 85] [CHU87]. Ce type de graphe est très utilisé dans la synthèse de circuit indépendant de la vitesse. Un STG est un réseau de Pétri interprété. Chaque place dans les réseaux de pétri qui est utilisé pour exprimer une opération de choix est éliminée dans le STG correspondant si cette place possède seulement une transition entrante et une transition sortante unique. Les deux transitions sont alors connectées entre elles par un arc dirigé (une flèche). Un arc entre deux transitions signifie qu'il existe une relation de causalité entre elles. Si cette flèche est pleine la causalité est issue du contrôleur. Une flèche en trait interrompu fin représente une causalité issue de l'environnement du circuit. De plus, les flèches marquées d'un point représentent l'état initial du circuit. Une transition dans un STG est considérée comme la transition montante ou descendante d'un signal.

Il existe trois types de signaux dans un STG : les entrées, les sorties et les signaux internes (variables d'états). Les signaux d'entrées et de sorties sont extérieurs aux circuits il dépendent d'un protocole bien défini. Les entrées viennent de l'environnement du circuit et les sorties vont vers l'environnement. Chaque transition est représentée par le nom du signal et un symbole « + » pour une transition montante, et un « - » pour une transition descendante.

Par exemple, la relation « x+ $\rightarrow$ y- » signifie que la transition montante du signal x est suivi d'une transition descendante du signal y. Ceci est interprété comme une spécification du comportement du circuit, si y est un signal de sortie du circuit, ou bien comme une spécification du comportement de l'environnement si y est un signal d'entrée. Aucune information n'est donnée sur le temps qui s'écoule entre les deux transitions. Ce temps doit seulement être positif et fini.

Voici, à la figure 24, un exemple de STG pour la porte de Muller qui respecte les propriétés énoncées un peu plus haut. Ici, la sortie de la porte est supposée à '0' à l'état initial.

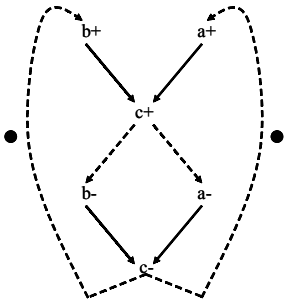


figure 24. STG de la porte de Muller.

1.4.3. Expansion des communications

Ce dernier modèle de spécification se retrouve dans la méthode de synthèse des circuits asynchrones quasiment insensibles aux délais d'Alain Martin [MART 93], est l'expansion des communications. Il s'agit d'une représentation intermédiaire entre la communication au niveau des canaux et les règles de production [MART 93]. L'expansion des communications

ou HSE (de l'anglais « HandShaking Expansion ») est textuelle. L'expansion des communications remplace chaque action de communication en un programme avec une implémentation en terme d'actions élémentaires et chaque canal en un couple d'opérateurs. La propriété de synchronisation des primitives de communication est seulement présentée.

Le formalisme de représentation de l'expansion de communication est le suivant :

La transition montante, respectivement descendante, d'un signal X sera noté « X+ », respectivement « X- ». Il s'agit de l'affectation d'un signal à une valeur. L'action d'attente d'un signal X à '1', respectivement à '0', se note [X], respectivement [¬X]. Les liens entre différentes affectations ou actions d'attente sont séquentiels (représentés par un point virgule « ; ») ou bien concurrents (représentés par une virgule « , »). Tout comme le STG qui s'exécute dans notre cas de façon infinie, cette notion de boucle infinie est représentée par l'étoile : *[liste d'instructions]. La liste des signaux ainsi que leur valeur initiale sont présentées dans une ligne déclarative de la forme : « Init(a, b)=0,0 ».

Reprenons l'exemple de la porte de Muller dans ce format de spécification : à l'initialisation les entrées a et b sont nulles (Init (a, b)=0,0).

$$*[[a],[b];c+;[\neg a],[\neg b];c-]$$

1.5. Représentation de l'information

Contrairement à la logique synchrone, où données et validité sont totalement décorrélées, la logique asynchrone a pour objectif de grouper chaque donnée (bit ou digit) avec l'information de sa validité. Le but est de toujours assurer la synchronisation entre les différents chemins de données. Ce principe implique nécessairement une nouvelle représentation de l'information dans le circuit, c'est-à-dire un nouveau codage. Cette représentation dépend du protocole que l'on vise (deux phases, quatre phases).

1.5.1. Le codage « données groupées »

La façon la plus évidente de caractériser la validité des données consiste simplement à ajouter un fil aux bus de données afin de spécifier cette information de validité (figure 25). Ce type de codage se retrouve principalement dans la réalisation de circuit de type micro pipeline. Les protocoles de communication deux et quatre phases pourront être indifféremment utilisés avec ce type de codage. Le fil de validité peut respecter un protocole deux phases ou quatre phases.

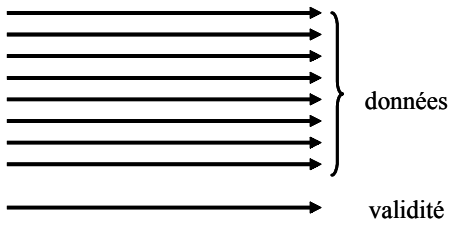


figure 25. Modèle de codage « données groupées ».

1.5.2. Le codage « quatre états »

Dans ce codage, chaque bit de données est représenté par deux fils. Parmi les quatre états possibles pour ces deux fils, la moitié est réservée à la valeur '0' l'autre à la valeur '1'. L'émission d'une nouvelle donnée se traduit par le changement d'un fil.

Deux conventions sont possibles :

- Un fil est dédié à une valeur et tout évènement sur un des fils signifie qu'une nouvelle valeur est valide. Avec ce codage le même code sur les deux fils peuvent représenter une valeur binaire différente. La valeur précédente émise servant dans la détermination de la nouvelle valeur à transmettre (cf. figure 26).

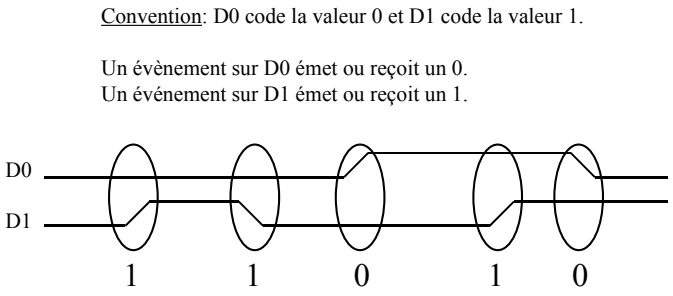


figure 26. Codage quatre états du canal D.

- La deuxième convention consiste à allouer un bit à la valeur et l'autre bit à la parité. Ainsi tout changement d'un bit conduit à un changement de parité. Si le bit de donnée change, la valeur change. Si le bit de parité change la valeur ne change pas. Cette convention nécessite des conditions initiales définies clairement au moment de la spécification du circuit. La validité des données est donc assurée par le changement de parité du couple de fils (cf. figure 27).

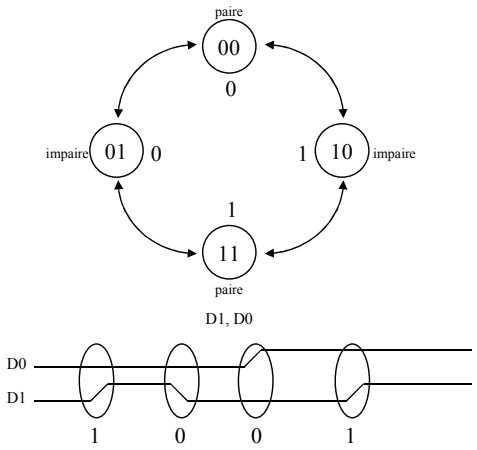


figure 27. Codage quatre état cyclique d'un canal D.

Le protocole quatre états se prête bien au protocole deux phases : chaque événement sur un des deux fils qui codent le bit transporte une information.

1.5.3. Le codage « trois états »

De même que le codage « quatre états », chaque bit est ici représenté par deux fils (figure 28). En revanche, les valeurs représentées ne sont pas dupliquées : une seule combinaison représente chaque valeur, tandis que la troisième indique l'état invalide et la quatrième reste inutilisée (valeur interdite sur le canal). Le passage d'une valeur valide à l'autre se traduit nécessairement par le passage à l'état invalide. Le codage employé ici est un codage 1 parmi n et peut être facilement utilisé pour coder n valeurs distinctes.

On retrouve ce type de codage dans l'implémentation de circuits quasiment insensibles aux délais ou de façon ponctuelle dans certains circuits de contrôle de pipeline (cf. chapitre V).

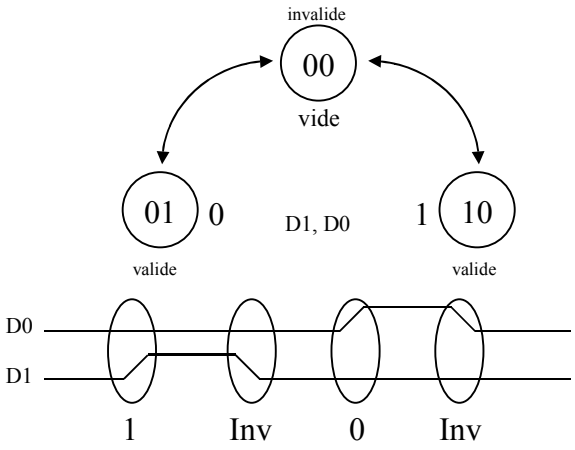


figure 28. Codage trois états du canal double rails D.

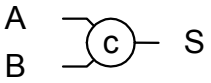
Le codage trois états est particulièrement bien adapté aux protocoles quatre phases. Il permet de coder en même temps la validité d'une donnée et sa valeur et de distinguer le passage d'une valeur à l'autre par le passage obligatoire à l'état invalide. Ce codage est lié à une logique à niveau qui est la plus communément employée.

Le codage quatre états cible des circuits dont la logique et la communication sont sensibles aux transitions. Il s'utilise donc plutôt pour les protocoles deux phases.

Le codage « données groupées » peut être aussi bien utilisé avec un protocole quatre phases ou un protocole deux phases avec les avantages et inconvénients inhérents à la logique à niveaux ou à fronts.

1.6. Portes de Muller

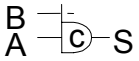
Ce paragraphe aborde rapidement le fonctionnement des portes de Muller ou « C élément ». Elles apparaissent tout au long du manuscrit. La porte de Muller réalise le rendez-vous entre plusieurs signaux. La sortie copie la valeur de ses entrées lorsque celles-ci sont identiques, sinon elle mémorise la dernière valeur obtenue. En voici une représentation accompagnée de sa table de vérité :



A\B	0	1
0	0	S^{-1}
1	S^{-1}	1

figure 29. Symbole et table de vérité de la porte de Muller à deux entrées.

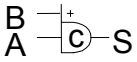
Dans certains cas toutes les entrées de la porte n'interviennent pas dans le passage à '0' ou à '1'de la sortie. Dans ce cas la porte de Muller est dite dissymétrique. Voici deux exemples de portes dissymétriques avec la table de vérité associée pour mieux comprendre son fonctionnement.



A\B	0	1
0	0	S^{-1}
1	1	1

figure 30. Exemple de porte de Muller dissymétrique.

Ici, figure 30, les entrées A et B font commuter la sortie à '0' quand elles valent '0' mais seule l'entrée A permet le passage à '1' de la porte.



A\B	0	1
0	0	0
1	S^{-1}	1

figure 31. Un autre exemple de Muller dissymétrique.

Dans cet exemple, figure 31, la sortie est à '1' lorsque A et B sont à '1'. S est nul dès que A est nul. Dans les autres cas, la porte mémorise sa dernière valeur.

2. Des communications gérées par un protocole

2.1. La communication dans un pipeline

Ce paragraphe est une présentation des protocoles de communication tels qu'ils ont été présentés tout au long de l'évolution de la recherche sur les circuits asynchrones. Seuls les protocoles quatre phases sont évoqués ici. Pour cette étude on se limite à caractériser la relation entre le protocole d'un canal d'entrée E et le protocole d'un canal de sortie S. Les structures plus complexes sont abordées par la suite comme exemples sur les protocoles choisis pour la synthèse avec TAST.

2.2. Protocole deux phases

Le contrôle des communications utilisant un protocole deux phases, sont sensibles aux évènements. Un événement sur le signal de requête ou sur le signal d'acquittement est interprété comme une nouvelle information.

Il faut donc utiliser des blocs logiques qui vont interpréter chaque transition comme une information. De façon générale, la logique à évènement est plus complexe que la logique qui fonctionne sur des niveaux. Par exemple la bascule de type D fonctionnant sur les deux fronts comporte deux fois plus de transistors qu'une bascule standard ([YUN 96b]).

2.3. Protocoles quatre phases : une synthèse

L'objet de ce paragraphe est de présenter l'ensemble des protocoles de communication quatre phases tel qu'ils sont rencontrés dans la littérature de la communauté asynchrone. Ils sont spécifiés sous forme de diagramme temporel, de STG ou de HSE. Le but est ici de comprendre leur fonctionnement, identifier leurs différences, et essayer de réaliser une classification. La différence entre certains de ces protocoles se base sur la durée de validité des données dans un cycle de communication. Pour d'autres les phases montantes et descendantes ainsi que leur point de synchronisation sont distingués. Pour chacun d'eux on présente l'implémentation pour des canaux simple rail.

Pour des raisons de clarté, seule la partie de contrôle d'un micro pipeline est considérée. Pour tous les protocoles présentés, les canaux portent les mêmes noms et la convention sur les signaux de contrôle reste, sauf indication contraire, la même pour toute la suite du document. La polarité des signaux d'acquittement est inversée par rapport à celle des signaux de requête. Une requête est signalée par une transition montante de '0' à '1'. Un acquittement est marqué par une transition descendante de '1' à '0'. A l'état initial, tous les acquittements sont à '1' qu'ils soient fournis par le circuit de contrôle (comme une sortie) ou par l'environnement (comme une entrée). Les signaux de requête sont à '0' à l'initialisation.

Un des objectifs est d'avoir une vision unifiée sur les protocoles quatre phases.

2.3.1. Protocole standard

Ce protocole est le premier et le plus intuitif des protocoles quatre phases. Il est appelé par Andrew Lines WCHB [LINE 95] pour « weak condition half buffer ». Chez Furber, il est appelé protocole de type "early" [FURB 96a]. La validité des données commence sur le front montant du signal de requête et se termine sur le front descendant du signal d'acquittement (dans la convention choisie, il s'agit du front actif pour le signal d'acquittement).

La spécification de ce protocole est donnée figure 32 sous la forme d'un diagramme temps. De celle-ci, il est possible d'extraire un STG ou une expansion des communications. En respectant la convention de données groupées, une donnée valide se présente sur le canal accompagnée d'une requête. Une fois les données consommées par le récepteur, on attend l'arrivée de l'acquittement qui remet à '0' la requête et invalide les données. Enfin, l'acquittement retourne à '1'.

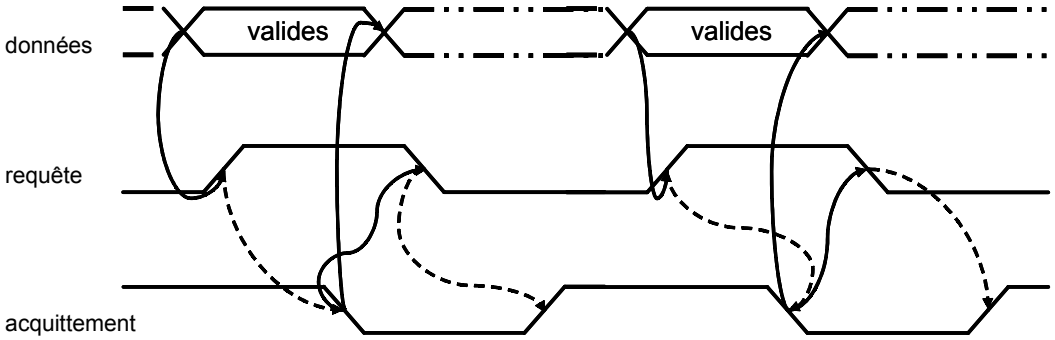


figure 32. Diagramme des temps du protocole « early ».

2.3.2. Protocole « broad »

Le protocole « broad », figure 33 offre une validité des données différente du protocole standard. Les données sont valides sur le front actif du signal de requête. La fin de validité des données sur le canal est signalée par le front inactif (de remise à '1') du signal d'acquittement. C'est lui qui offre la plus grande plage de validité des données.

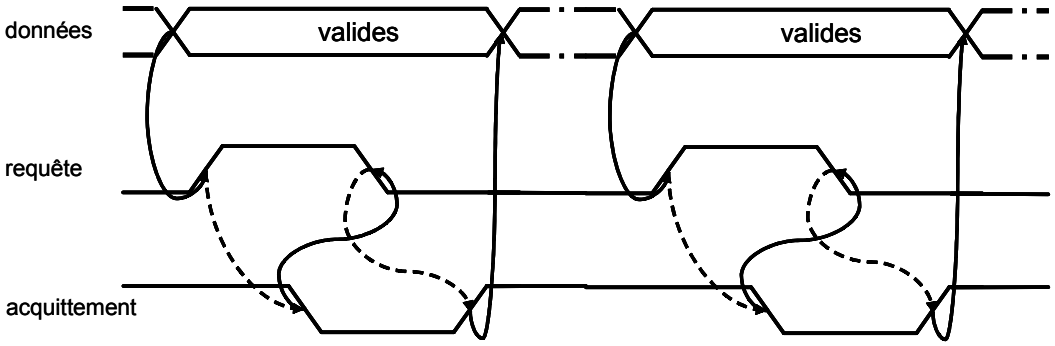


figure 33. Diagramme des temps du protocole de type « broad ».

2.3.3. Protocole « broadish »

La figure 34 décrit le protocole « broadish ». Ici la validité des données commence sur le front actif de la requête et se termine sur le front inactif de ce même signal de contrôle. Cette validité des données n'est donc pas dépendante du signal d'acquittement. Il faut attendre la réinitialisation de l'acquittement avant de recommencer un cycle.

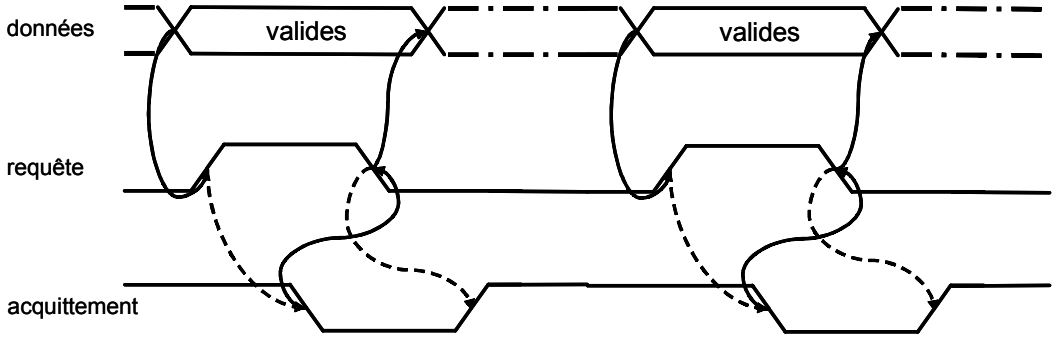


figure 34. Diagramme des temps du protocole de type « broadish ».

2.3.4. Protocole « late »

Ce dernier protocole prend en compte les deux fronts inactifs des signaux de requête et d'acquittement (figure 35). Dans ce cas les données sont actives sur le front descendant de la requête et la fin des données valides est sur le front montant de l'acquittement dans notre convention. C'est le symétrique du protocole de type « early » mais, il retarde au maximum la validité des données dans le cycle.

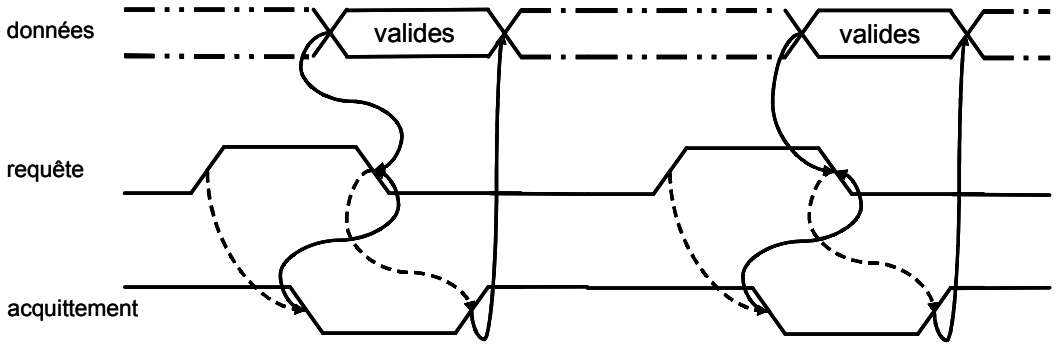


figure 35. Diagramme des temps du protocole de type « late ».

2.3.5. Protocole WCHB

Les trois protocoles suivant sont issus des solutions de réordonnancement des étapes d'une communication entre un canal de sortie et un canal d'entrée. Cette étude repose sur l'expansion des communications dans les canaux conformément à un protocole quatre phases. Elle a été menée initialement par Steve Burns en 1995([LINE 95]). Toutes ces expansions des

communications permettent une implémentation mais ne sont pas efficaces en termes de vitesse et de complexité. On présente ici les plus performant dans l'ordre du protocole le plus séquentiel (WCHB) au plus concurrent (PCFB).

Le protocole WCHB (« Weak Conditionning Half Buffer ») est identique au protocole « early ». Il y a pour ce protocole synchronisation des phases montantes et des phases descendantes entre les canaux d'entrée et de sortie. Ce protocole est présenté sous la forme d'une expansion des communications. La spécification est la suivante :

$$*\left[\left[S_{acq} \wedge E_{req}\right]; S_{req}+; E_{acq}-; \left[\neg S_{acq} \wedge \neg E_{req}\right]; S_{req}-; E_{acq}+\right]$$

Les deux protocoles suivant ont aussi été proposés par A. Lines dans le but d'optimiser les performances en terme de vitesse des communications au sein des canaux.

2.3.6. Protocoles PCHB

L'acronyme PCHB vient de l'anglais « precharge logic half buffer ».

Les phases montantes du protocole sont identiques au WCHB. Au niveau des remises à '0' des signaux de requête et d'acquittement, il y a désynchronisation pour autoriser la communication au niveau de la sortie indépendamment des entrées et ainsi gagner en vitesse dans le pipeline. Pour cela, il y a d'abord remise à zéro du canal de sortie puis remise à zéro du canal d'entrée. Ainsi le gain de vitesse devient significatif quand le nombre de canaux d'entrées est important. Il n'est plus obligatoire d'attendre la remise à zéro des requêtes d'entrée pour terminer les autres protocoles en sortie.

L'expansion des communications pour ce protocole est la suivante :

$$*\left[\left[S_{acq} \wedge E_{req}\right]; S_{req}+; E_{acq}-; \left[\neg S_{acq}\right]; S_{req}-; \left[\neg E_{req}\right]; E_{acq}+\right]$$

2.3.7. Protocoles PCFB

Le dernier protocole abordé ici est appelé « precharge logic full buffer ». Ici on découple totalement les canaux d'entrée et de sortie dans la phase de remise à zéro. Les deux communications sur les entrées et les sorties se font de façons totalement concurrentes. La première partie du protocole reste inchangée. Cela offre les meilleures performances au niveau vitesse mais l'implémentation de ce protocole est coûteuse en circuit. En effet afin de respecter les conditions de codage complet il faut insérer une variable d'état afin de pouvoir distinguer dans quelle partie du protocole on se trouve.

L'expansion des communications pour le protocole est la suivante :

$$*\left[\left[S_{acq} \wedge E_{req}\right]; S_{req}+; E_{acq}-; En-; [\neg En]; \left(\left[\neg S_{acq}\right]; S_{req}-\right) \left(\left[\neg E_{req}\right]; E_{acq}+\right) En+; [En]\right]$$

3. Implémentation des protocoles quatre phases

Les explications suivantes reprennent celles du modèle STG présentées en 1.4.2 adaptées à nos exemples de contrôleurs. Les spécifications pour les contrôleurs de pipeline quatre phases sont présentées sous forme de diagramme de transitions ou sous forme d'expansion des communications. En plus du canal d'entrée E et du canal de sortie S, ces contrôleurs possèdent un signal interne « Le » pour déclencher la mémorisation des données. Les registres sont composés de verrous transparents (« latchs »). Les flèches en traits interrompus fins indiquent l'ordre des transitions qui doit être respecté par l'environnement du contrôleur (les contrôleurs adjacents ou les circuits communiquant avec le pipeline). Les flèches pleines correspondent à l'ordre des transitions que doit assurer le circuit lui-même. Le point près des arcs représente une condition initiale nécessaire pour parcourir le graphe de transitions. Une transition ne peut se produire dans un graphe que si chaque arc possède un point. Quand une transition se produit on place un point sur tous les arcs de sortie et on en enlève un sur les arcs en amont. Ainsi, il est possible de parcourir le graphe et de voir comment s'enchaînent les étapes du protocole.

Les STG figure 36, représentent les séquences de communication des canaux d'entrée et de sortie. On retrouve aussi quelques propriétés comportementales.

- S_{req}+ indique que les données du côté des sorties du circuit sont prêtes et donc doit suivre l'action E_{req}+.
- Quand de nouvelles données sont présentes sur les entrées (E_{req}+) les verrous peuvent se fermer (Le+) et ensuite les entrées peuvent être acquittées (E_{acq}-).
- Une fois que les données de sorties ont été consommées par l'étage suivant (S_{acq}-) les verrous peuvent de nouveau être ouverts (Le-).
- Les verrous sont alternativement ouverts puis fermés.

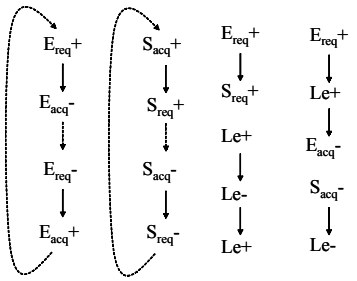


figure 36. Spécifications sous forme de STG d'un contrôleur de pipeline quatre phases.

3.1. Contrôleur simple

Le contrôleur simple met en œuvre le protocole standard. Sa représentation, sous la forme d'un graphe de transitions des signaux, est faite au paragraphe suivant. On retrouve dans ces spécifications le canal d'entée E et le canal de sortie S. Le signal Le est le signal de contrôle du registre qui est composé de verrous transparents.

3.1.1. Spécification du contrôleur simple sous la forme d'un STG

La figure 37 présente le STG du protocole « early ».

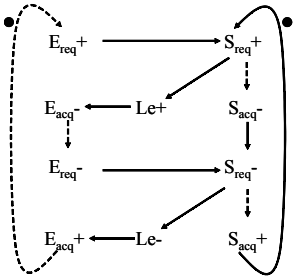


figure 37. STG du contrôleur simple quatre phases.

3.1.2. Circuit du contrôleur simple quatre phases

Le circuit résultant est présenté figure 38 :

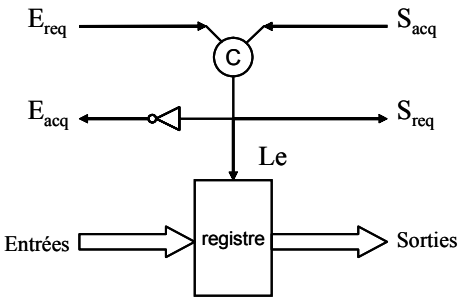


figure 38. Contrôleur simple quatre phases.

3.2. Contrôleur semi découplé

Pour augmenter le découplage entre l'entrée et la sortie du verrou, il est possible de modifier le diagramme de transitions afin de permettre au verrou de se fermer (de mémoriser la donnée) avant que le protocole côté sortie du contrôleur soit terminé. La nouvelle spécification du contrôleur sous forme de STG est figure 39.

3.2.1. Spécification du contrôleur semi découplé sous la forme d'un STG

Pour obtenir le découplage souhaité une variable interne, noté A, est introduite pour mémoriser que le canal d'entrée est prêt à recevoir une nouvelle requête. Maintenant le verrou de l'étage courrant peut se fermer avant que le verrou du contrôleur voisin (côté sortie) soit ouvert. Cela est possible car maintenant la transition Le+ est concurrente avec la transition S_{acq} +.

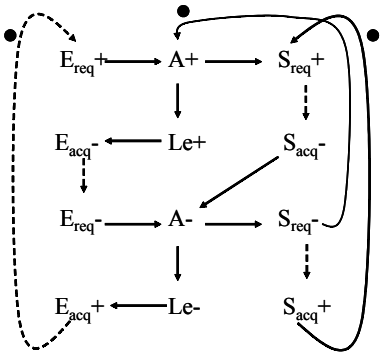


figure 39. STG du contrôleur semi découplé quatre phases.

3.2.2. Circuit du contrôleur semi découplé quatre phases

Voici figure 40, le circuit résultant de la synthèse du protocole semi découplé.

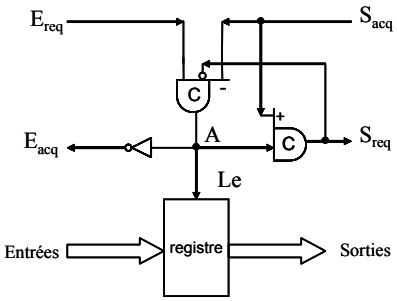


figure 40. Contrôleur semi découplé quatre phases.

Pour la première fois dans le contrôleur de la figure 40, il y a des portes de Muller non symétriques. Elles caractérisent la dissymétrie exprimée par la spécification du contrôleur. Ces portes sont introduites chapitre I et sont étudiées dans le chapitre sur les bibliothèques de cellules asynchrones (chapitre V).

3.3. Contrôleur découplé

3.3.1. Spécification du contrôleur découplé sous la forme d'un STG

Ce troisième contrôleur quatre phases offre un découplage total entre ses entrées et ses sorties. Et cela que ce soit pour les phases montantes ou descendantes du protocole quatre phases. Pour cela, une deuxième variable interne B est introduite. Maintenant, dès que l'étage reçoit une donnée en entrée et qu'elle a été mémorisée par le registre (Le +), la communication peut se terminer sur les entrées et les sortie pour ensuite revenir dans l'état initial.

Le STG proposé à la figure 41 est fortement concurrent.

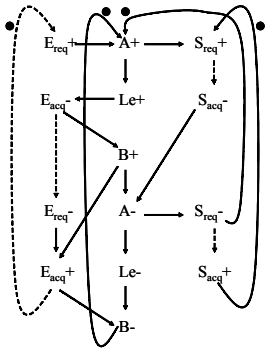


figure 41. STG d'un contrôleur quatre phases découplé.

3.3.2. Circuit du contrôleur découplé quatre phases

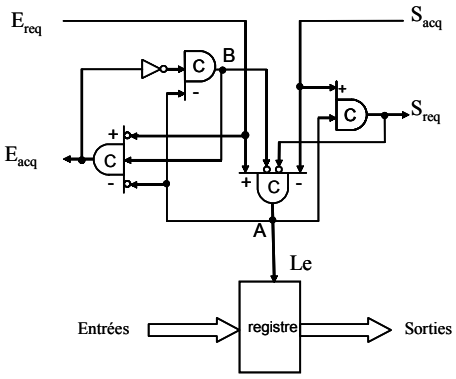


figure 42. Schéma du contrôleur découplé quatre phases.

<u>Remarque</u> : parfois il peut être utile que le verrou maintienne les données jusqu'à ce que S_{acq} revienne à '1', plutôt que de le relâcher quand S_{acq} passe à '1' comme le fait le circuit précédent [FURB 96a]. Si par exemple, un verrou contient l'adresse pour un banc de registre, alors S_{req} peut être utilisé pour valider l'adresse du décodeur. S_{acq}- indique que la donnée a été lue du banc de registres et S_{acq}+ indique que le décodeur a été invalidé. En voici, figure 43, une spécification sous forme d'un STG suivi du contrôleur correspondant (figure 44).

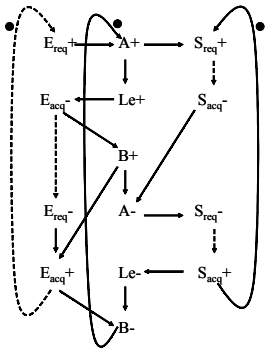


figure 43. STG du contrôleur découplé quatre phases avec maintien des données.

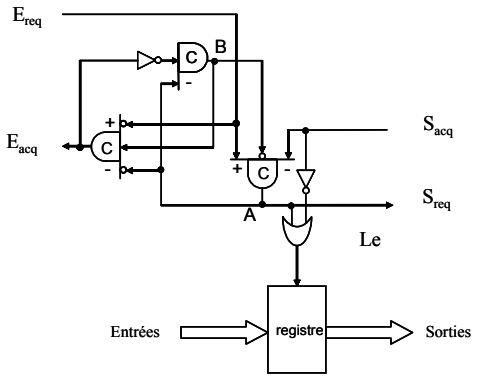


figure 44. Schéma du contrôleur découplé avec maintien des données.

Le protocole élémentaire ainsi que deux de ses variantes viennent d'être proposés. Le premier est aujourd'hui très souvent utilisé pour des raisons de simplicité d'implémentation bien que ses performances ne soient pas toujours optimales. Les variantes offrent un découplage plus ou moins important dans l'échange des données dans un but d'augmenter les performances au détriment de la complexité des circuits de contrôle.

3.4. Contrôleur de type « broad » quatre phases

Les deux contrôleurs suivants implémentent les protocoles de type « broad » et de type « broadish » qui sont basés sur une validité des données différente. Les données sont stables dès la requête jusqu'à la remise à '1' de l'acquittement pour le premier et jusqu'à la remise à zéro de la requête pour le second.

3.4.1. Spécification du contrôleur de type « broad »

Pour le contrôleur de protocole, les données sont maintenues stables dès que la requête est active ($E_{req}+ \rightarrow A+ \rightarrow Le+$), jusqu' au retour à l'état invalide du signal d'acquittement ($Sacq- \rightarrow Le-$). C'est le protocole qui maintient les données valides le plus longtemps.

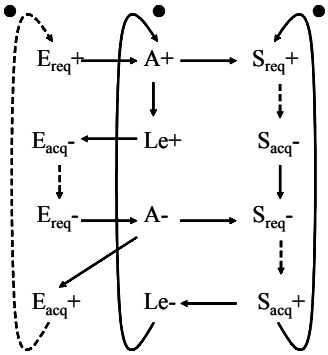


figure 45. STG du protocole de type « broad ».

3.4.2. Circuit du contrôleur de type « broad »

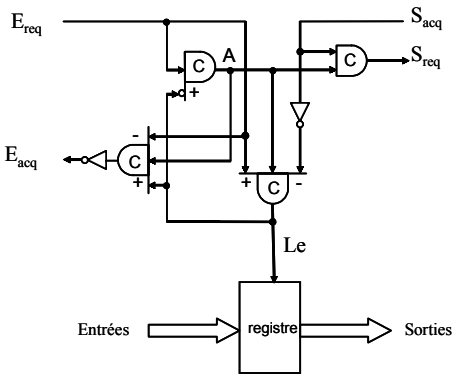


figure 46. Circuit du contrôleur quatre phases avec protocole « broad ».

3.5. Contrôleur « broadish » quatre phases

3.5.1. Spécifications du protocole de type « broadish »

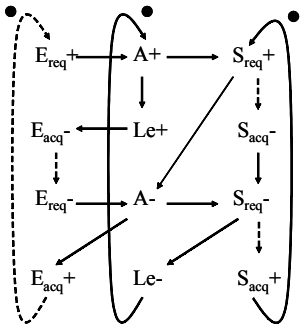


figure 47. STG du protocole de type « broadish ».

La phase active des données correspond à la phase active de la requête (de E_{req}+ jusqu'à S_{req}-). La spécification se retrouve figure 47.

3.5.2. Circuit du contrôleur de type « broadish »

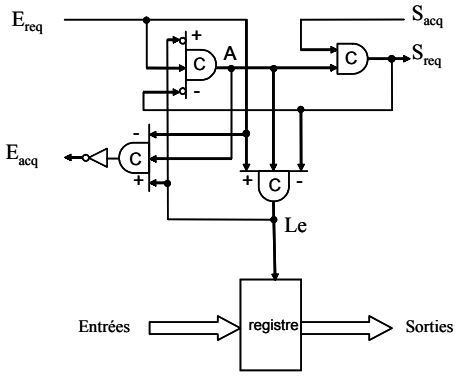


figure 48. Circuit du contrôleur quatre phases avec protocole « broadish ».

L'acquittement du canal de sortie S_{acq} n'intervient pas dans la mémorisation des données.

3.6. Autres contrôleurs

Voici les contrôleurs qui appliquent les protocoles proposés par Andrew Lines. La méthode de synthèse de ces protocoles est présentée dans le chapitre III. Ici, seul le circuit est montré. Il est le résultat de l'application de l'algorithme de synthèse d'Alain Martin sur la dérivation d'un jeu de règles de production à partir de l'expansion des communications d'un programme CHP [MART 97].

3.6.1. Contrôleur WCHB

Le contrôleur pour le protocole WCHB est strictement identique au protocole de type « Early » vu plus tôt dans le chapitre (2.3.1). La figure 49 ne montre que le schéma correspondant à l'expansion des communications suivantes :

$$*\left[\left[S_{acq} \wedge E_{req}\right]; S_{req}+; E_{acq}-; \left[\neg S_{acq} \wedge \neg E_{req}\right]; S_{req}-; E_{acq}+\right]$$

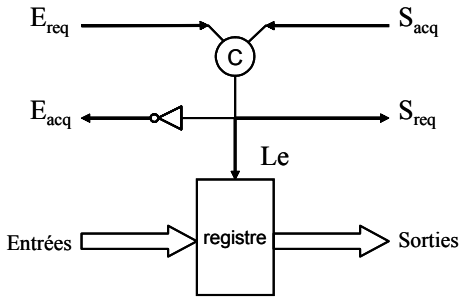


figure 49. Contrôleur de type WCHB.

Les deux circuits de contrôleurs suivants implémentent les protocoles PCHB et PCFB. Ils sont obtenus par un changement de l'ordre des opérations de l'expansion des communications du protocole WCHB. Ces modifications sont faites pour optimiser les performances des communications en vitesse. Ces évolutions doivent respecter la succession des transitions pour qu'il n'y ait pas de blocage fonctionnel.[MART 93].

3.6.2. Contrôleur PCHB

Le protocole PCHB est représenté sous la forme d'une expansion des communications : la première partie du protocole est identique au protocole WCHB mais dans la phase de remise à '0' il y a désynchronisation entre la sortie et l'entrée. On exécute d'abord la réinitialisation de la sortie puis la remise à zéro de l'entrée.

$$*\left[\left[S_{acq} \wedge E_{req}\right]; S_{req}+; E_{acq}-; \left[\neg S_{acq}\right]; S_{req}-; \left[\neg E_{req}\right]; E_{acq}+\right]$$

Le circuit correspondant est présenté en figure 50 :

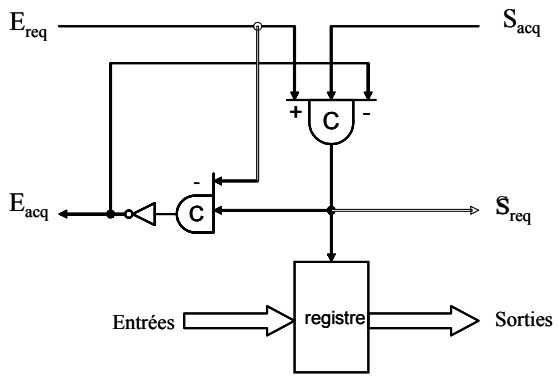


figure 50. Contrôleur de pipeline PCHB.

3.6.3. Contrôleur PCFB

C'est la version totalement découplée du protocole WCHB. La première phase du protocole est identique mais la remise à '0' se fait en parallèle sur l'entrée et sur la sortie.

$$*\left[\left[S_{acq} \wedge E_{req}\right]; S_{req}+; E_{acq}-; En-; [\neg En]; \left(\left[\neg S_{acq}\right]; S_{req}-\right) \left(\left[\neg E_{req}\right]; E_{acq}+\right); En+; [En]\right]$$

Le circuit du contrôleur correspondant au protocole PCFB est présenté figure 51.

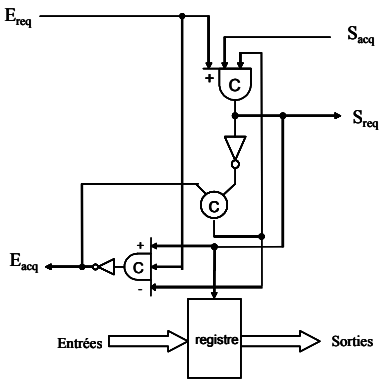


figure 51. Contrôleur de pipeline PCFB.

Pour ces trois autres protocoles le calcul est plus rapide. En effet, si le canal de sortie est disponible (S_{acq}), la donnée (E_{req}) est transmise par S_{req}+ puis est aussitôt acquittée par E_{acq}- et ce sans attendre l'acquittement de la sortie $\neg S_{acq}$. Ceci permet de commencer la remise à zéro du protocole du canal d'entrée indépendamment du temps de calcul sur S. Pour ces trois protocoles, il y a un rendez-vous au sein même du contrôleur entre la disponibilité de l'étage suivant (S_{acq}) et l'arrivée d'une nouvelle donnée (E_{req}). C'est cette condition qui autorise le renouvellement des actions entre les canaux E et S.

3.7. Des contrôleurs pour des performances ciblées

La notion de découplage introduite à partir du contrôleur standard a pour objet d'optimiser les performances en vitesse des contrôleurs. Le but est ici de diminuer leur temps de cycle. Cela est démontré avec les contrôleurs semi découplé et découplé. Ils permettent d'autoriser le remplissage complet d'un pipeline.

3.7.1. Propriétés du contrôleur simple

Bien que ce contrôleur fonctionne correctement, il possède des propriétés indésirables. La plus importante est lorsqu'on juxtapose plusieurs étages de contrôleurs simples afin de former une file de type FIFO, au mieux un étage sur deux peut être occupé en même temps (figure 52). Cela parce que S_{acq} doit être à '1' (et par conséquent le prochain verrou ouvert) avant que ''Le'' puisse passer à son tour à '1' (et ainsi que ce verrou soit fermé).

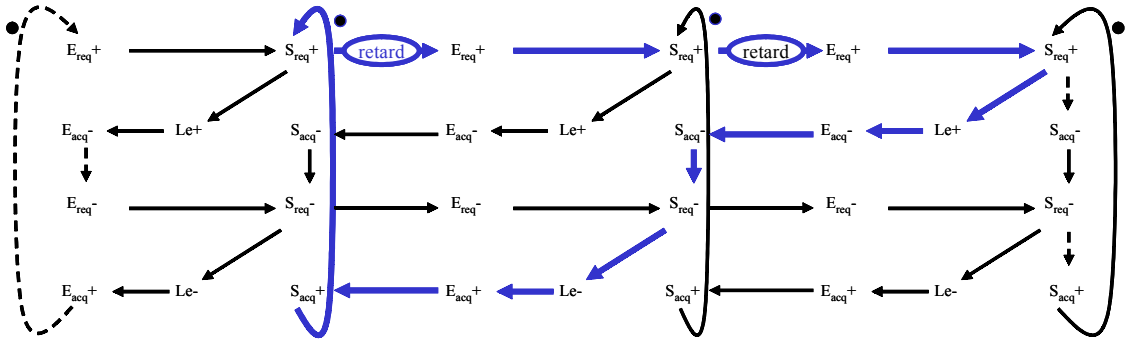


figure 52. Trois étages du contrôleur simple avec calcul.

D'autres problèmes se posent aussi au niveau du nombre de verrous que doit charger le signal Le. On suppose que ce signal doit être correctement piloté afin de pouvoir charger le nombre de verrous nécessaires. Dans la figure 52, le chemin de E_{req} à E_{acq} montre le besoin au verrou de se fermer avant que les données en entrée soient enlevées. Le pilote doit donc être positionné sur ce chemin. Par contre il n'est pas nécessaire de positionner un délai sur le chemin de E_{req} à S_{req}, puisqu'il n'y a pas besoin pour le verrou d'être fermé avant que S_{req} soit signalé aussi longtemps que la donnée se propage à travers le registre. Cependant pour un fonctionnement correct du circuit le temps de traversé de la porte de Muller ne doit pas être plus court que celui d'un verrou.

3.7.2. Propriétés du contrôleur semi découplé

Le temps de cycle de ce contrôleur est montré figure 53 par la représentation de trois étages successifs de pipeline. Entre chacun des étages on introduit un temps de retard correspondant au temps critique de calcul du bloc de traitement entre deux étages.

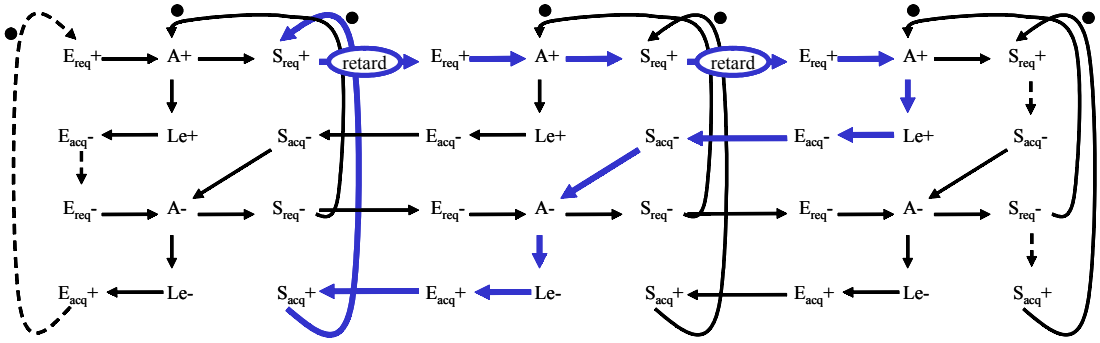


figure 53. Trois étages de pipeline semi découplé quatre phases.

Ce contrôleur semi découplé quatre phases offre de très bonnes performances et chaque étage du pipeline pourra être rempli si on bloque la communication en sortie. Cependant les phases de réinitialisation du protocole du côté des entrées et des sorties d'un même contrôleur sont toujours liées: E_{req} ne peut pas retourner à '0' tant que S_{acq} n'est pas à '1'. Dans une simple FIFO cela ne pose pas de problème, mais si on introduit des blocs de traitement entre les étages, il peut y avoir des pertes de performances dues à ce couplage partiel.

3.7.3. Propriétés du contrôleur découplé

A partir du diagramme de transitions du protocole découplé on a de nouveau mis en série trois étages pour visualiser le temps de cycle de ce nouveau circuit. La figure 54 permet aussi de se rendre compte que grâce à son exécution fortement concurrente, le protocole sur les entrées peut s'effectuer indifféremment de la sortie du contrôleur. Par contre on ne pourra pas en recommencer un nouveau si le protocole du côté de la sortie n'est pas terminé (arc de S_{acq}+ à A+). Ce protocole offre une meilleure performance au niveau de la vitesse mais demande la présence de deux variables internes ce qui devient coûteux au niveau de la complexité du contrôleur.

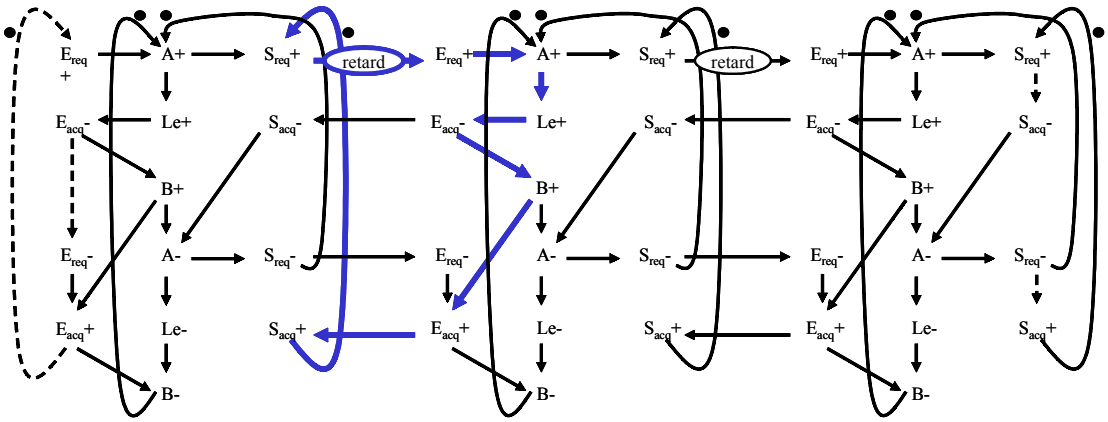


figure 54. Trois étages de pipeline découplé quatre phases.

3.8. Conclusion

Une présentation des différents protocoles qui gèrent la communication dans un canal et leur circuit de contrôle associé vient d'être faite. Elle repose sur des moyens de spécifications et de représentation commun à tous les protocoles (nom des signaux, polarité des signaux de contrôle, modèles de représentation, logique de représentation des circuits de contrôle). Cet état de l'art propose une vision unifiée des protocoles de communication.

De cette étude, il ressort que les différents protocoles de communication se distinguent entre eux suivant deux approches :

- la durée de validité des données (early, découplé, semi-découplé, broad, broadish, late),
- l'enchaînement des phases du protocole (WCHB, PCHB, PCFB).

Le protocole standard (« early » chez Furber) est identique au protocole WCHB. Il synchronise, entre les entrées et les sorties, les phases de communication.

Le protocole semi-découplé autorise la remise à zéro des entrées avant celles des sorties. C'est une version améliorée (augmentant le taux de remplissage d'une file) du protocole standard. Le protocole PCHB réalise la remise à zéro côté des sorties avant celle des entrées. Tous les deux offrent de meilleures performances en vitesse lorsque le nombre d'entrées ou de sorties devient important.

Deux protocoles proposent un découplage total entre les entrées et les sorties et permettent toujours d'améliorer la performance en vitesse des communications au sein d'un pipeline. Mais la concurrence ajoutée dans la communication se fait au détriment de la complexité des contrôleurs associés. Le protocole à besoin de deux variables internes, donc deux points de mémorisation contre un pour le protocole PCFB.

Finalement, les protocoles choisis ont été sélectionnés comme les plus performants et les plus représentatif :

- protocole WCHB : simplicité et symétrie des communications.
- protocole PCHB : désynchronisation des communications pour augmenter les performances en vitesse au détriment de la complexité du contrôleur.
- protocole PCFB : le plus rapide et le plus concurrent mais le plus complexe des protocoles retenus. Il nécessite une variable interne supplémentaire pour mémoriser l'évolution des phases.

Ces protocoles sont proposés lors de la phase de synthèses de TAST.

Chapitre III

Une vision unifiée de la synthèse des canaux de communication

L'objet de ce chapitre est de présenter à travers plusieurs méthodes de synthèse l'implémentation des protocoles sélectionnés au chapitre précédent (WCHB, PCHB, PCFB).

Dans le cadre du développement de la partie « back-end » d'un outil de synthèse de circuits asynchrones, l'objectif, ici, est double :

- Vérifier la convergence des méthodes de synthèse vers une implémentation correcte des canaux de communication.
- Identifier les besoins de cellules spécifiques qui résultent de ces synthèses.

Pour cela, deux méthodes de synthèse de circuits asynchrones sont présentées. Elles partent d'un formalisme de description des spécifications différent et permettent d'obtenir des classes de circuits différentes.

La première méthode, proposée par Alain Martin, part du formalisme d'expansion des communications (HSE) et cible les circuits quasiment insensibles aux délais (QDI). Nous aborderons ensuite, la méthode basée sur les graphes de transitions des signaux (STG). Les circuits générés alors, seront de la classe des circuits indépendants de la vitesse.

Pour la méthode proposée par Martin, un outil a été réalisé au cours de la thèse et est détaillé ici. Il fourni les règles de production des variables internes et des sorties. L'objectif de cette application est, dans le cadre de l'étude des contrôleurs, d'obtenir rapidement des règles de production qui servent de base pour une implémentation de contrôleurs particuliers.

Après la présentation de ces méthodes de synthèse, une grande partie de ce chapitre est consacrée aux étapes de réalisation de chacune de ces méthodes pour les protocoles retenus au chapitre II.

1. Présentation des méthodes de synthèse

1.1. Méthode de synthèse de Martin : présentation de l'algorithme à travers une lecture écriture séquentielle.

Au cours de ce paragraphe, la méthode d'extraction de règles de production d'un programme décrit dans un langage de haut niveau (CHP) est abordée. Le formalisme de départ est l'expansion des communications d'un programme CHP. Cette étape fait partie de la

méthode de compilation du CHP d'Alain Martin [MART 93]. Cette synthèse est décrite à travers un exemple simple de programme linéaire. Elle se décompose en trois étapes :

- l'affectation de variables d'état pour obtenir un programme avec un codage complet de ses états CSC,
- le renforcement des gardes (conditions d'affectation d'un signal),
- la réduction des opérateurs qui donnera un réseau de portes standard à partir des règles de production. Cette dernière étape est aussi appelée symétrisation.

1.1.1. Un programme linéaire

L'exemple pour présenter l'extraction des règles de production est un programme CHP réalisant le séquencement entre deux canaux E et S. E est un canal d'entrée passif. Il est en attente d'une requête. S est un canal de sortie actif. Il initialise une communication par l'envoie d'une requête. Le programme CHP est le suivant : $*[E?;S!]$. Il réalise une lecture sur le canal E (le point d'interrogation en CHP) puis une écriture sur le canal S (le point d'exclamation en CHP). Ce processus est le programme de base pour réaliser une séquence entre la communication sur E et la communication sur S.

Le protocole quatre phases appliqué sur le canal E est le suivant sachant qu'on lui associe deux fils E_{req} et E_{acq} qui sont respectivement les fils de requête et d'acquittement. Ici E attend que le canal reçoive une requête provenant de l'environnement.

$$E \equiv [Ereq]; Eacq-; [\neg Ereq]; Eacq+$$

De même pour le canal de sortie S, la communication commence par l'envoie d'une requête à l'environnement (le canal est déclaré actif).

$$S \equiv Sreq+; [\neg Sacq]; Sreq-; [Sacq]$$

<u>Remarque</u> : la convention adoptée pour la polarité des signaux de contrôle est la suivante : le fil de requête sera '0' à l'état initial. Le fil d'acquittement possède une polarité inversée. A savoir l'acquittement est à '1' pendant l'état initial.

L'expansion des communications de ce programme est alors la suivante :

$$*[[Ereq]; Eacq-; [\neg Ereq]; Eacq+; Sreq+; [\neg Sacq]; Sreq-; [Sacq]]$$

C'est la traduction directe d'une communication sur le canal E suivi d'une communication sur le canal S. Elle peut être réécrite sous la forme suivante :

$$*[[Ereq]; Sreq+; [\neg Sacq]; Sreq-; [Sacq]; Eacq-; [\neg Ereq]; Eacq+]$$

Ce HSE représente la spécification du circuit à synthétiser. Par la suite, les autres exemples seront présentés sous cette forme. Chaque implémentation de ce programme doit satisfaire l'ordre défini par la spécification ci-dessus. La prochaine étape est de construire un ensemble de règles de production qui satisfait cet ordre. La première étape est la déduction

d'un premier jeu de règles de production qui est directement dérivé de l'expansion des communications.

$$Ereq \rightarrow Sreq+$$

$$Sacq \rightarrow Eacq-$$

$$\neg Sacq \rightarrow Sreq-$$

$$\neg Ereq \rightarrow Eacq+$$

Puisque le programme considéré est sans blocage fonctionnel, l'exécution effective des règles de production dans l'ordre du programme est toujours possible. Cependant, il existe d'autres ordres d'exécution. Un ensemble de règles de production satisfait la spécification de l'expansion des communications si, et seulement si, le seul ordre d'exécution possible est l'ordre du programme. Si des ordres d'exécution, autre que l'ordre du programme, sont possibles pour le jeu de règles de production, les gardes de certaines règles sont renforcées de façon à supprimer ces ordres d'exécution.

Dans le programme, l'ordre du programme n'est pas le seul ordre d'exécution pour le jeu des règles de production. S_{acq} étant vrai dès l'initialisation, la troisième règle de production ($Sacq \rightarrow Eacq-$) peut être exécutée en premier. C'est aussi vrai pour la quatrième ($\neg Ereq \rightarrow Eacq+$).Toutes les variables de la communication du canal E étant de retour à leur valeur initiale quand E se termine, il n'est pas possible de trouver une garde pour l'affectation S_{req}+ qui soit maintenue vraie seulement comme précondition de S_{req}+ dans l'expansion des communications. Ne pouvant distinguer l'état suivant E et l'état précédent E, notre programme ne vérifie pas la condition de codage complet des états.

Cette propriété permet de vérifier que le codage de chacun des états de l'expansion des communications est unique. Dans le but de remplir la condition d'exécution séquentielle, nous devons garantir que chacun des états du HSE est unique. C'est à dire qu'il existe un prédicat en terme de variable du programme qui est vrai dans cet état. La tâche de transformer le programme pour rendre chacun de ces états unique est appelée affectation d'états.

1.1.2. Affectation d'états avec variables d'états

Une méthode pour déterminer de façon unique l'état dans lequel la transition S_{req}+ doit prendre place consiste en l'introduction d'une variable d'état, appelé ici x. L'expansion des communications initiales devient la suivante.

$$*[[Ereq];Sreq+;[\neg Sacq];x+;[x];Sreq-;[Sacq];Eacq-;[\neg Ereq];x-;[\neg x]Eacq+]$$

Cette nouvelle expansion des communications est sémantiquement équivalente à la précédente car les deux séquences ajoutées correspondent à un « Skip » (une commande qui ne fait rien). Il existe plusieurs places où les deux affectations de la variable d'état peuvent être introduites. Les différentes heuristiques qui sont utilisées dans le placement des variables ne sont pas abordées ici.

Il est important de remarquer que la minimisation du nombre de variables d'état n'est pas un bon critère de choix de l'affectation d'un état. Ce qui compte, c'est le nombre de transitions sur les variables d'état et la taille de la garde des règles de production. En effet, si la règle de production comporte un trop grand nombre d'éléments, l'étape suivante qui consiste à regrouper les règles entre elles afin d'obtenir des portes logiques devient une tache difficile.

1.1.3. Algorithme de base pour l'extraction des règles de production

Le point de départ pour présenter cet algorithme est une expansion des communications linéaire (sans choix) dont l'affectation des variables a déjà été faite. Une fois que chaque état du HSE est unique, il est ensuite possible de générer des règles de production qui ont un sens similaire au HSE. Chaque affectation à une variable est telle que x+ et x- se produit au moins une fois et une seule fois. Cette condition est facilement renforcée en se rappelant que dans le cas d'un programme de cette forme :

$$P \equiv \cdots x+;\cdots;x-;\cdots;x+;\cdots;x-;\cdots$$

La variable x est renommée comme :

$$P' \equiv \cdots x1+;\cdots;x1-;\cdots;x2+;\cdots;x2-;\cdots$$

D'abord l'extraction des règles de production est faite sur P'. Comme $\neg x1 \vee \neg x2$ sont vraies à chaque instant, il est possible de combiner x1 et x2 par ces deux règles :

$$x1 \vee x2 \rightarrow x+$$

$$\neg x1 \vee \neg x2 \rightarrow x-$$

Toutes les gardes des règles de production sont des conjonctions. Les disjonctions sont ensuite introduites dans l'étape de symétrisation.

1.1.3.1. Première méthode : affaiblissement des gardes fortes

Puisque chaque état de l'expansion des communications est défini de façon unique, l'ensemble des règles de production, dans lequel chaque garde est le prédicat le plus robuste de cet état, est ordonné.

Le jeu des gardes les plus fortes est construit mécaniquement : chaque état de l'expansion est codé par la valeur binaire de tous les signaux qui interviennent dans le programme. Le prédicat, le plus robuste, dans chaque état est la conjonction de toutes les variables qui sont vraies. Ensuite, les gardes des règles de production sont simplifiées par l'utilisation des propriétés de la forme : P => R est vraie comme précondition de la règle de production pour remplacer P et R par P. Cette méthode a été proposée et utilisée par Huubs Schols.

1.1.3.2. Deuxième méthode : renforcement des gardes faibles

La deuxième méthode, qui est la plus souvent utilisée, part du jeu de règles de production les plus faibles et les renforce jusqu'à que les règles de production soient ordonnées (rendre l'ordre d'exécution unique pour la spécification).

Pour chaque affectation, la garde initiale de la règle de production est l'action d'attente qui précède cette affectation dans l'expansion des communications. Quand l'affectation, appelée S, est précédée par une autre affectation, une action d'attente est introduite issue de l'affectation précédente S. Par exemple : $x+;S$ est remplacé par $x+;[x];S$ ou bien $x-;S$ est remplacé par $x-;[\neg x];S$.

Pour chacune des affectations on définit deux ensembles d'états :

- « Firing set » d'un signal X, noté FS(X), est l'ensemble de tous les états dans lequel la garde de l'affectation est vraie.

- « Conflicting set » d'un signal X, noté CS(X), est l'ensemble de tous les états dans lequel le lancement de l'affectation doit être interdit. Pour l'affectation S, soit S' l'affectation complémentaire. CS(S) est l'ensemble des états successifs commençant à l'état précédant S' et finissant à l'état qui précède l'affectation qui précède S.

La fenêtre de S est l'intersection du « firing set » et du « conflicting set » de S. Pour qu'une règle de production soit ordonnée il faut que la fenêtre de S soit vide. On dit aussi que la fenêtre est fermée. Si ce n'est pas le cas, il faut réduire le « firing set » par un renforcement de la précondition de S jusqu'à ce que l'intersection soit vide.

Parce que chaque état peut être caractérisé de façon unique en terme de variable de programme, il est toujours possible de fermer une fenêtre pour chacune des affectations. Il existe plusieurs façons de renforcer une garde. On choisit celle qui est la plus simple (un nombre minimal de variables) et celle qui est la plus souhaitable pour la symétrisation des règles de production.

Comme exemple d'application de cet algorithme, voici la démonstration d'un théorème qui identifie les règles de production standard ne nécessitant pas d'être renforcées. Ce résultat réduit considérablement le nombre de cas à considérer.

1.1.3.3. Théorème

La règle de production $\neg xacq \rightarrow xreq-$, d'une expansion des communications active X, et la règle de production $\neg xreq \rightarrow xacq+$, d'une expansion des communications passive X, sont toujours ordonnées.

1.1.3.4. Démonstration

L'expansion des communications active d'un canal X est :

$$X \equiv Xreq+;[\neg Xacq];Xreq-;[Xacq]$$

Pour $\neg Xacq \rightarrow Xreq-$, le « firing set » commence de la précondition Xreq- et se termine à la post condition de Xreq-. Le « conflicting set », lui, commence à la précondition de Xreq+ et se termine à la post condition de Xreq+. Même avec réordonnancement (changement de l'ordre des affectations et des gardes de la spécification sans en changer le sens et sans introduire de blocage fonctionnel) ces deux ensembles disjoints : la fenêtre est donc fermée. Il en est de même pour une communication passive.

Si nous reprenons l'exemple précédent :

$$*[[Ereq];Sreq+;[\neg Sacq];x+;[x];Sreq-;[Sacq];Eacq-;[\neg Ereq];x-;[\neg x];Eacq+]$$

Les règles de production avant renforcement des gardes sont les suivantes :

$$Ereq \rightarrow Sreq+$$

$$\neg Sacq \rightarrow X+$$

$$Sacq \rightarrow Eacq-$$

$$X \rightarrow Sreq-$$

$$\neg Ereq \rightarrow X-$$

$$\neg X \rightarrow Eacq+$$

Les règles de production ont une fenêtre fermée sans avoir à renforcer les gardes sauf pour S_{req}+ et E_{acq}-. Appliquons l'algorithme pour ces deux règles de production

1.1.3.5. Règle de production de Sreq+

Le « firing set » est le suivant :

FS(S_{req}+)=$[Ereq];Sreq+;[\neg Sacq];x+;[x];Sreq-;[Sacq];x-;[\neg x];Eacq-;$

Le « conflicting set » donne :

CS(S_{req}+)=$[x];Sreq-;[Sacq];x-;[\neg x];Eacq-;[\neg Ereq]$

Les deux ensembles d'états ne sont pas disjoints, il faut, alors,renforcer la garde de la règle de production et recalculer le « firing set ». Après le renforcement par ¬x de S_{req}+, le « firing set » devient :

FS(S_{req}+)=$[Ereq]; Sreq+; [\neg Sacq]; x+; [x]; Sreq-; [Sacq];$

Cela ferme maintenant la fenêtre car l'intersection entre le nouveau « firing set » et le « conflicting set » est vide. De même, en renforçant par x la dernière règle de production, cela ferme aussi la fenêtre. Après ces deux renforcements de gardes, les règles de production suivantes sont toutes ordonnées :

$$\neg X \wedge Ereq \rightarrow Sreq+$$

$$X \rightarrow Sreq-$$

$$\neg X \rightarrow Eacq+$$

$$X \wedge Sacq \rightarrow Eacq-$$

$$\neg Sacq \rightarrow X+$$

$$\neg Ereq \rightarrow X-$$

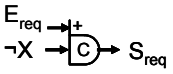
1.1.4. Réduction des opérateurs

La dernière étape de la compilation est la réduction d'opérateurs. Elle groupe les règles de production qui affectent la même variable. Ces règles de production sont implémentées alors par un opérateur (une porte).

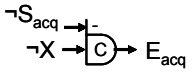
Comme on a renforcé la stabilité de chaque règle et on n'interfère pas entre deux règles complémentaires, on peut implémenter n'importe quel jeu de règles de production. Pour des raisons d'efficacité, il faut vérifier que la garde ne contient pas trop de termes dans une conjonction. Cela signifierait trop de transistors en série. L'implémentation de ces deux jeux de règles de production peut aussi aboutir à une décomposition de celles-ci par l'introduction de nouvelles variables.

L'implémentation directe en portes de Muller généralisées (ces portes sont abordées au chapitre V) du jeu de règles de production donne les portes suivantes :

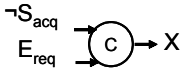
Sreq= Muller2D1P($\neg$X, E_{req})



Eacq=Muller2D1N($\neg$X, $\neg S_{acq}$)



X=Muller($\neg S_{acq}$, E_{req})



La représentation en logique dynamique de ce circuit est la plus simple. Pour une représentation statique, une solution possédant moins d'éléments mémorisants doit être envisagée. Ainsi l'opération de symétrisation permet de diminuer le nombre d'éléments mémorisants.

1.1.5. Symétrisation

La symétrisation est réalisée sur les gardes des règles de production $g1 \rightarrow z+$ et $g2 \rightarrow z-$, quand une des deux gardes appelées ici g1 est déjà de la forme $x \wedge \neg g2$. Si on remplace g2 par $\neg x \vee g2$, alors les deux gardes sont complémentaires entre elles, c'est-à-dire l'opérateur est combinatoire. Bien sur, affaiblir la garde g2 est une transformation dangereuse car cela risque d'introduire un nouvel état où la garde serait vraie.

Il faut, alors, vérifier la chose suivante : Etant donné la nouvelle règle de production $\neg x \vee g2 \rightarrow z-$, $\neg z$ doit être vrai dans chacun des états où $x \wedge \neg g2$ est vrai, c'est-à-dire que il faut vérifier l'invariance de la vérité $\neg x \vee g2 \vee \neg z$.

$$\neg X \wedge Ereq \rightarrow Sreq +$$

$$(\neg Ereq \vee) X \rightarrow Sreq -$$

$$\neg Sacq \vee \neg X \rightarrow Eacq +$$

$$X \wedge Sacq \rightarrow Eacq -$$

$$\neg Sacq \rightarrow X +$$

$$\neg Ereq \rightarrow X -$$

La porte associée à S_{req} est le AND($\neg$x, E_{req}).

La porte associée à E_{acq} est le NAND(x, S_{acq}).

La porte associée à X est une bascule de type R/S qui peut être remplacée par une porte de Muller symétrique.

La figure 55 représente le circuit final. Il est aussi appelé par Martin le Q-élément [MART 93].

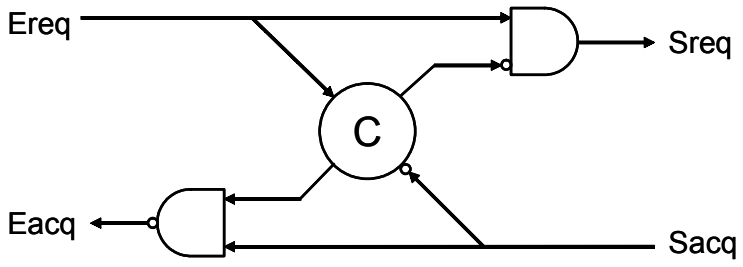


figure 55. Schéma du circuit Q-élément.

1.1.6. Fourches isochrones

On suppose que les fourches introduites dans le circuit sont isochrones et que la différence de délais dans les deux branches de la fourche est plus courte que les délais dans les opérateurs auxquels une branche de la fourche est une entrée. Donc quand une transition sur

une sortie est acquittée, donc complétée, la transition sur l'autre sortie est aussi acquittée et donc complétée. C'est la seule hypothèse temporelle qui doit être remplie. Cependant, la condition d'isochronisme est difficile à remplir quand une entrée négative introduit un inverseur sur une branche de la fourche, puisque le délai de traversé d'un inverseur est du même ordre de grandeur que celui des autres opérateurs. Il a été prouvé, dans [BERK 93], que ces inverseurs peuvent toujours être éliminé des fourches isochrones par de simples transformations.

Dans [MART 90], il a été prouvé que la classe des circuits entièrement insensibles aux délais est très limitée : pratiquement tous les circuits susceptibles d'être intéressant ne font pas partie de cette classe. La notion de fourche isochrone est le compromis le plus simple pour implémenter des circuits insensibles aux délais. Les fourches qui doivent être isochrones peuvent être identifiée en examinant l'ensemble des règles de production.

1.1.7. Un outils de génération de circuits pour la synthèse de contrôleurs

L'étude sur la méthode de synthèse d'Alain Martin a conduit au développement d'un outil réalisant de façon automatique les règles de production à partir d'un programme décrit sous la forme d'une expansion des communications. Le but est, par la maîtrise de cette technique de synthèse, d'avoir rapidement un résultat de synthèse pour faire de l'investigation sur des circuits de complexité plus grande.

1.1.7.1. Le formalisme d'entrée

Le format d'entrée de ce programme est un fichier texte comportant la spécification du contrôleur à synthétiser sous forme d'une expansion des communications (HSE). Le HSE est précédé d'une partie déclarative qui comprend les signaux de requête et d'acquittement des canaux ainsi que les variables d'états. Cette déclaration précise également les conditions initiales des signaux.

Pour le HSE, la composition parallèle et séquentielle des affectations et des gardes est autorisée.

Voici un exemple :

Init(E_{req}, E_{acq}, S_{acq}, S_{req})="0110"

*[

[E_{req}],[S_{acq}]; S_{req} +; E_{acq} -; [/E_{req}],[/ S_{acq}]; S_{req} -; E_{acq} +

]

Un programme d'entrée possède donc une partie déclaration et conditions initiales :

Init(listes des signaux) = « valeurs initiales »

L'expansion des communications est encadrée par *[] pour évoquer la répétition comme en CHP. Ensuite nous avons une succession de gardes sur les signaux (marquée par []) et d'affectations séparées par des points virgules (composition séquentielle) ou des virgules (composition parallèle). La virgule est prioritaire sur le point virgule. Les parenthèses permettent des compositions particulières.

Le signe « + », respectivement « - », est utilisé pour une affectation positive, négative.

La négation d'un signal est notée par le symbole « / ».

1.1.7.2. Vérifications avant synthèse

Pour qu'une description soit correcte il faut que pour un signal donné, il y ait alternance des transitions et ceci quelque soit le chemin de parcours du HSE.

En plus de l'alternance (positive, négative) des affectations, avant d'entamer toute synthèse, une vérification de la propriété de codage complet des états est réalisée.

En partant de l'état initial des signaux, le programme calcule tous les états atteignables de la spécification. Tous les parcours des branches possibles sont pris en compte quand il y a des compositions parallèles. Chaque état atteint est codé par la valeur binaire des signaux du circuit. A la fin du parcours, si plusieurs états possèdent le même codage, on retourne une erreur car la spécification n'est pas CSC (« Complete State Coding » : codage complet des états).

Cet outil ne prend pas en compte l'insertion automatique de variables pour résoudre les conflits de codage complet des données. Dans le cas où cette propriété n'est pas respectée, il faut ajouter de façon manuelle une variable interne ou on procède à un réordonnancement des phases de communication.

1.1.7.3. Règles de production initiales

Dans cette étape, pour chaque affectation, une première règle de production est calculée en transformant la garde ou l'affectation qui précède l'affectation considérée. Soit le morceau de programme suivant :

$$\ldots\ [/E_{req}],[/\ S_{acq}];\ S_{req}\ -;\ldots$$

La règle de production initiale de l'affectation S_{req}- est la condition d'attente qui la précède. Ici, il s'agit de $[/E_{req}],[/\ S_{acq}]$.

Cette étape se résume à la transformation en équation booléenne de la garde ou de l'affectation qui précède l'affectation traitée.

Le résultat de cette étape est un premier jeu de règles de production pour le programme à synthétiser. Ces règles doivent être renforcées pour conduire à une implémentation correcte du circuit.

1.1.7.4. Renforcement des gardes

Pour cela il faut déduire du programme les deux ensembles d'états correspondant au « firing set » et au « conflicting set » définit par Martin [MART 93]. La garde ou l'affectation sont marquées d'un attribut « f » pour le « firing set » du signal et « c » pour le « conflicting set » du signal.

Après le calcul de l'intersection de ces deux ensembles d'états deux cas se présentent :

- l'intersection est vide. Cela signifie que la fenêtre est fermée. La règle de production est minimale et l'affectation suivante est traitée de la même façon.

- l'intersection est non vide. Il faut renforcer la garde de la règle de production considérée en prenant dans le programme un état supplémentaire. Cela entraîne le calcul du nouveau « firing set ». On recalcule la nouvelle intersection des deux ensembles.

Le renforcement des gardes n'est pas optimisé dans la version utilisée de l'outil. On s'est contenté de renforcer les gardes sans chercher à les optimiser. Une solution optimisée serait de rechercher le terme minimal permettant de rendre l'intersection entre le « firing set » et le « conflicting set » vide. Cela dit, la solution optimisée est toujours un sous ensemble de la règle de production obtenue par un renforcement systématique. Cette optimisation n'a pas été faite car elle demandait un développement trop long pour l'automatiser. Par contre elle a été faite par l'utilisateur lorsque l'équation de la règle de production n'était pas optimisée.

1.1.7.5. Règles de production finales

Une fois toutes les affectations traitées, le programme retourne le jeu de règles de production qui sont minimales pour implémenter le programme.

Les étapes de réduction des opérateurs et de symétrisation n'ont pas été implémentées dans cet outil. Les circuits à synthétiser restant simple dans l'ensemble il était assez facile de les exécuter de façon manuelle. Enfin, on s'est limité à des circuits implémentés à base de portes de Muller généralisées ou des porte logiques simples.

1.1.7.6. Conclusion sur l'outil

Cet outil de génération sert de support pour obtenir rapidement un jeu de règes de production pertinent sur un ensemble de programme. Certains points peuvent être enrichis ou optimisés mais ce n'était pas le but et cela aurait pris un temps de développement trop long.

Il permet de faire une investigation sur des programmes de contrôleurs abordés dans ce chapitre ou des structures plus complexes.

1.2. Méthode STG

Le formalisme des STG présenté dans le paragraphe 1.4.2 du chapitre I est utilisé pour représenter les protocoles d'Andrew Lines.

1.2.1. Conditions de synthèse pour un STG

Un STG décrivant un circuit doit respecter certaines propriétés. Les algorithmes de synthèse à partir de STG nécessitent des propriétés supplémentaires. Elles sont énumérées ici.

1. L'entrée libère le choix : le choix parmi les transitions doit seulement être commandé par des entrées mutuellement exclusives.
2. Un jeton unique par place.
3. Le STG doit être sans blocage fonctionnel.

Pour que le STG soit en plus de la classe des circuits indépendants de la vitesse :

4. Une affectation consistante des états : Les transitions d'un signal doivent strictement alternées entre une transition positive (+) et une transition négative (-). Et cela quelque soit l'exécution du STG.
5. Persistance : Si la transition d'un signal est validée, elle doit ensuite prendre une place : c'est à dire qu'une transition ne doit pas être invalidée par une autre transition de signal. Cette propriété doit être garantie dans les STG pour les signaux internes (variables d'états) et pour les signaux de sortie, et par conséquent c'est à l'environnement de garantir la persistance des signaux d'entrée.

Pour synthétiser un circuit une dernière condition est nécessaire :

6. Codage complet des états (complete state coding : CSC) : Deux ou plusieurs endroits dans un STG ne doivent pas avoir le même codage. Si cette condition est violée, il faut introduire une variable d'état supplémentaire pour distinguer les codages identiques. On entend par codage la représentation de tous les états atteignables d'un circuits en affectant à une valeur binaire tous les signaux de ce circuit (entrées, sorties, variables internes).

1.2.2. Etapes de synthèse à partir d'un STG

La synthèse à partir d'une telle description se décompose suivant les étapes suivantes :

- Décrire le comportement de la fonction à implémenter et de son environnement sous forme d'un STG.

- Générer le graphe d'états correspondant, et rajouter les variables d'état si nécessaire. L'ajout de variables internes dépend du respect de la propriété du codage complet des états. Une variable interne peut être utile pour respecter aussi le bon fonctionnement du circuit désiré.

- Extraire les équations booléennes pour les variables d'état et les sorties. Cette étape se base sur les tableaux de Karnaugh dérivés du graphe d'état, et le calcul des régions d'excitations et de repos pour chaque transition des sorties et des variables d'états.

Chacune des équations booléennes obtenues peut l'être sous la forme de trois architectures différentes : portes complexes, portes de Muller généralisées ou portes complexes à base de portes de Muller symétriques. Ces deux dernières implémentations reposent sur le calcul des fonctions de « Set et Reset » des signaux (équations booléennes excitant les sorties du circuit).

Petrify est un outil qui réalise automatiquement la synthèse de contrôleurs à partir de STG [CORT 96]. Le but ici est de maîtriser toutes les étapes de cette synthèse. Elles sont donc présentées en détails pour chaque protocole.

2. Synthèse de protocoles

Voici l'application des méthodes de synthèses à base de STG et de HSE sur les quatre protocoles suivants : séquentiel, WCHB, PCHB, PCFB. Ce sont les protocoles que propose l'outil TAST pour la synthèse des programmes CHP.

Ils sont le résultat du changement de l'ordre des séquences dans l'expansion des communications d'un programme réalisant une communication séquentielle entre un canal d'entrée et un canal de sortie [LINE 95] :

$$*[[Ereq]; Sreq+; [\neg Sacq]; Sreq-; [Sacq]; Eacq-; [\neg Ereq]; Eacq+]$$

Cette étape est appelée réordonnancement. C'est une alternative pour rendre un programme « CSC » (codage complet des états). Cependant, elle ne doit pas introduire de problème de blocage de fonctionnement du processus.

Les circuits sont donnés en portes de Muller généralisées. On conserve les mêmes conventions que celles adoptées au chapitre précédent, tant pour les signaux que pour les protocoles.

2.1. Premier protocole : une version séquentielle

Dans l'étude de Lines [LINE 95], cette description est la plus séquentielle. La requête E_{req} est transmise de l'entrée vers la sortie puis est acquittée une fois que la sorite est acquittée à son tour. Cette séquence est valable aussi bien pour la phase de calcul que pour la phase de remise à zéro.

2.1.1. Méthode de Martin

Partant de l'expansion des communications initiale, la seconde partie du HSE de S est retardée. C'est-à-dire la séquence $Sreq-;[Sacq]$ est placée après la condition d'attente $[\neg Ereq]$. La séquence devient :

$$*\left[\left[E_{req}\right];S_{req}+;\left[\neg S_{acq}\right];E_{acq}-;\left[\neg E_{req}\right];S_{req}-;\left[S_{acq}\right];E_{acq}+\right]$$

Dans cet exemple les règles de production minimales sont déjà ordonnées.

$$E_{req} \rightarrow S_{req}+$$

$$\neg E_{req} \rightarrow S_{req}-$$

$$S_{acq} \rightarrow E_{acq}+$$

$$\neg S_{acq} \rightarrow E_{acq}-$$

Réduction des opérateurs : la première et la deuxième règle correspondent à un simple fil. De même pour la troisième et la quatrième règle. Dans cet exemple, le circuit final sera juste deux fils. Chaque sortie du circuit est une combinaison des entrées, il n'y a pas de mémorisation. La figure 56 représente un circuit séquentiel et non pipeliné de la séquence entre les canaux E et S (sans élément de mémorisation).

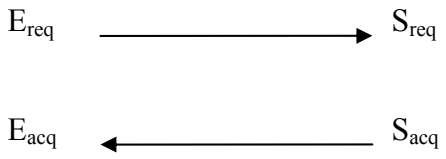


figure 56. Circuit issu du premier HSE.

2.1.2. Méthode STG

2.1.2.1. STG du protocole séquentiel

Le graphe de transition des signaux, figure 57, spécifie le circuit à réaliser pour le protocole séquentiel. Il sert de support à l'application des étapes de synthèse. A partir de celui–ci, le graphe d'états du circuit est dérivé.

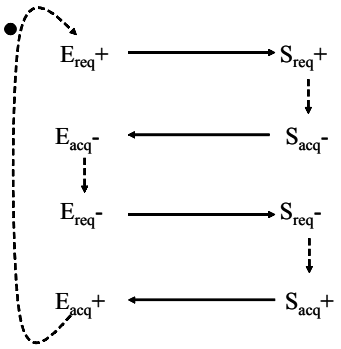


figure 57. STG du protocole combinatoire.

2.1.2.2. Graphe d'états

La construction du graphe d'états se fait en codant sous forme de vecteur binaire l'ensemble des états atteignables par le circuit. En partant de l'état initial, il faut parcourir le STG. Après chaque transition, il y a un nouvel état. Toute branche concurrente entraînera une structure en diamant pour représenter tous les chemins possibles. Il est évident qu'on se limite à des exemples simples pour faciliter la représentation (un petit nombre d'entrées et de sorties). C'est cette explosion possible du nombre d'états qui est un facteur limitant de cette méthode pour la synthèse de circuits asynchrones complexes.

Le vecteur codant chaque état considère les signaux dans cet ordre : « E_{req} E_{acq} S_{acq} S_{req} » L'état initial sera donc « 0110 ». On note par « * » un signal qui va changer de valeur à l'état suivant. D'où la figure 58 :

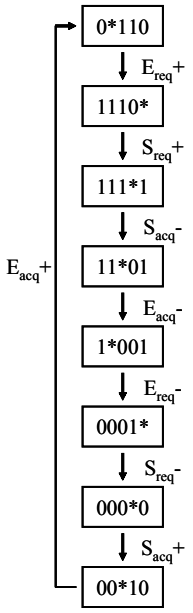


figure 58. Graphe d'état du protocole combinatoire.

Ici, le codage des états est complet, il n'existe pas d'états possédant le même code. La propriété de CSC est vérifiée. La synthèse de ce circuit est donc possible.

2.1.2.3. Fonction d'état suivant

La fonction d'état suivant comprend deux régions qui sont des ensembles d'états. Elle est calculée pour toutes les sorties et toutes les variables internes et est composée de deux sous fonctions :

- La région d'excitation du signal X (en anglais « Excitation Region ») se note ER(X). Elle correspond à l'ensemble des états où X passe à '1' (noté ER+(X)) et l'ensemble des états où X passe à '0' (noté ER-(X)).

- La région de quiétude du signal (en anglais « Quiescent Region ») se note QR(X). Elle regroupe l'ensemble des états où X est à '1' et reste à '1' (noté QR+(X)) et l'ensemble des états où X est à '0' et reste à '0' (noté QR-(X)).

La figure 59 définit ces deux régions pour les sorties E_{acq} et S_{req}.

Rappel du vecteur des signaux : « E_{req} E_{acq} S_{acq} S_{req} »

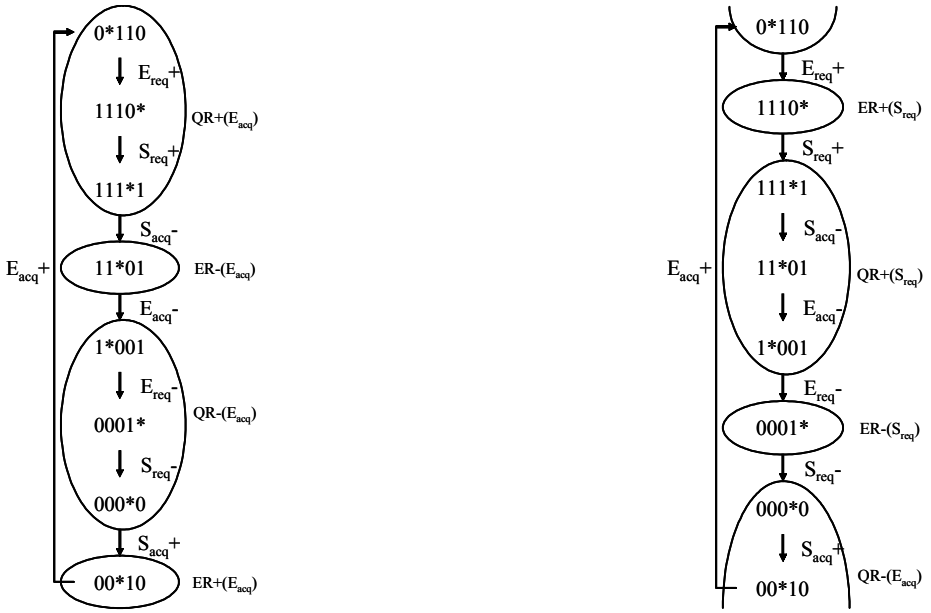


figure 59. Fonctions d'état suivant du protocole séquentiel.

A partir de ces fonctions, il faut déterminer les tableaux de Karnaugh qui permettent d'extraire les règles de production du circuit.

2.1.2.4. Tableaux de Karnaugh des sorties E_{acq} et S_{req}

On retranscrit dans un tableau de Karnaugh à quatre entrées l'évolution des sorties du circuit à partir des graphes d'états précédents. Les états non affectés sont notés par le signe moins « - ». Les régions d'excitation sont notées par une étoile dans les tableaux de Karnaugh : ER+(X) = {0*} et ER-(X) = {1*}. Enfin les régions de quiétude sont l'ensemble des zéros ou des uns : QR+(X) = {1} et QR-(X) = {0}.

A partir de la figure 59 il vient les tableaux de la figure 60.

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	0	-	-	-
01	0	-	1*	0
11	-	-	1	-
10	0*	1	1	-

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	0	-	-	-
01	1*	-	1	1
11	-	-	1	-
10	0	0	0*	-

figure 60. Tableaux de Karnaugh du protocole combinatoire.

De ces tableaux, l'objectif est de synthétiser des circuits à base de portes de Muller généralisées. Pour cela il faut déterminer les équations de « set » qui permet le passage à '1' de la sortie (noté pour un signal x, S(x)) et de « reset » qui permet le passage à '0' (noté pour un signal x, R(X)) des sorties en regroupant la région d'excitation de la transition montante avec le maximum de uns et la région d'excitation de la transition descendante avec le maximum de zéros.

Trois règles sont à observer pour extraire ces équations de « set » et de « reset » :

- Les équations de set et de reset doivent être mutuellement exclusives.
- S(X) doit couvrir ER+(X) et ne doit pas rencontrer ER-(X) ∪ QR-(X)
- R(X) doit couvrir ER-(X) et ne doit pas rencontrer ER+(X) ∪ QR+(X)

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	0	-	-	-
01	0	-	1*	0
11	-	-	1	-
10	0*	1	1	-

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	0	-	-	-
01	1*	-	1	1
11	-	-	1	-
10	0	0	0*	-

De ces deux tableaux on en déduit que :

$$S_{acq} \rightarrow E_{acq}+$$

$$\neg S_{acq} \rightarrow E_{acq}-$$

$$E_{req} \rightarrow S_{req}+$$

$$\neg E_{req} \rightarrow S_{req}-$$

Ici, le circuit obtenu est extrêmement simple puisqu'il se résume à deux fils.

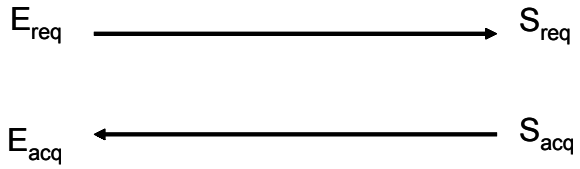


figure 61. Circuit du protocole séquentiel.

2.2. Protocole WCHB

2.2.1. Méthode de Martin

Appliquons l'algorithme d'extraction de règles de production au HSE suivant :

$$*\left[\left[S_{acq} \wedge E_{req}\right];S_{req}+;E_{acq}-;\left[\neg S_{acq} \wedge \neg E_{req}\right];S_{req}-;E_{acq}+\right]$$

Après avoir vérifier qu'il était sans conflit de codage, déterminons les règles initiales de production avant un éventuel renforcement de celles-ci.

$$E_{req} \wedge S_{acq} \rightarrow S_{req}+$$
$$\neg E_{req} \wedge \neg S_{acq} \rightarrow S_{req}-$$
$$\neg S_{req} \rightarrow E_{acq}+$$
$$S_{req} \rightarrow E_{acq}-$$

Pour savoir ci ces règles de production sont suffisantes on vérifie si les fenêtres de ces affectations sont fermées :

$FS(S_{req}+)$: $\left[S_{acq} \wedge E_{req}\right];S_{req}+;E_{acq}-$

$CS(S_{req}+)$: $\left[\neg S_{acq} \wedge \neg E_{req}\right];S_{req}-;$

Ici, la fenêtre est effectivement fermée. De même pour les autres affectations de cet exemple, les fenêtres sont fermées, aucune garde n'est à renforcer.

La réduction des opérateurs est ici simple car les règles de production sont symétriques. Ainsi pour ce protocole, le contrôleur correspondant donnera le circuit de la figure 62 :

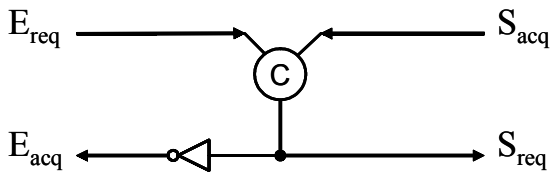


figure 62. Circuit de contrôle du protocole WCHB.

2.2.2. Méthode STG

2.2.2.1. STG du protocole WCHB

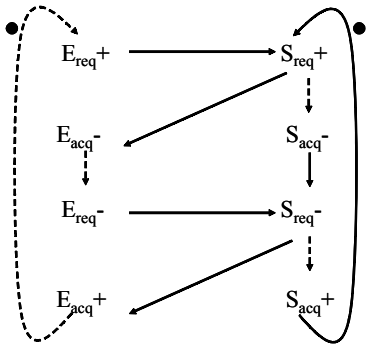


figure 63. STG du protocole WCHB.

Dans la figure 63, le protocole synchronise les deux phases d'affectation des sorties suivant un schéma très symétrique.

2.2.2.2. Graphe d'états du protocole WCHB

La correspondance des signaux dans le codage des données est : « E_{req}, E_{acq}, S_{acq}, S_{req} ».

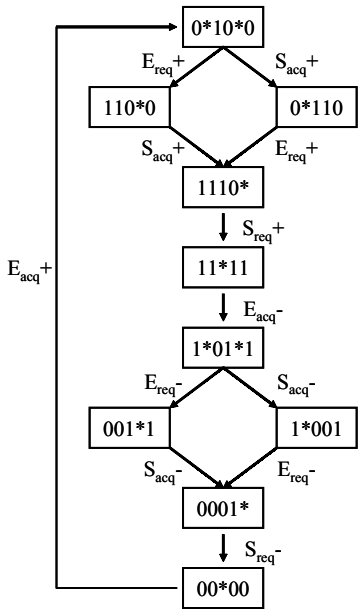


figure 64. Graphe d'états du protocole WCHB.

Les deux fourches rappellent (cf. figure 64) qu'il n'y a pas, à priori, d'ordre d'arrivée des entrées E_{req} et S_{acq} pour produire la sortie S_{req}.

2.2.2.3. Fonctions d'état suivant

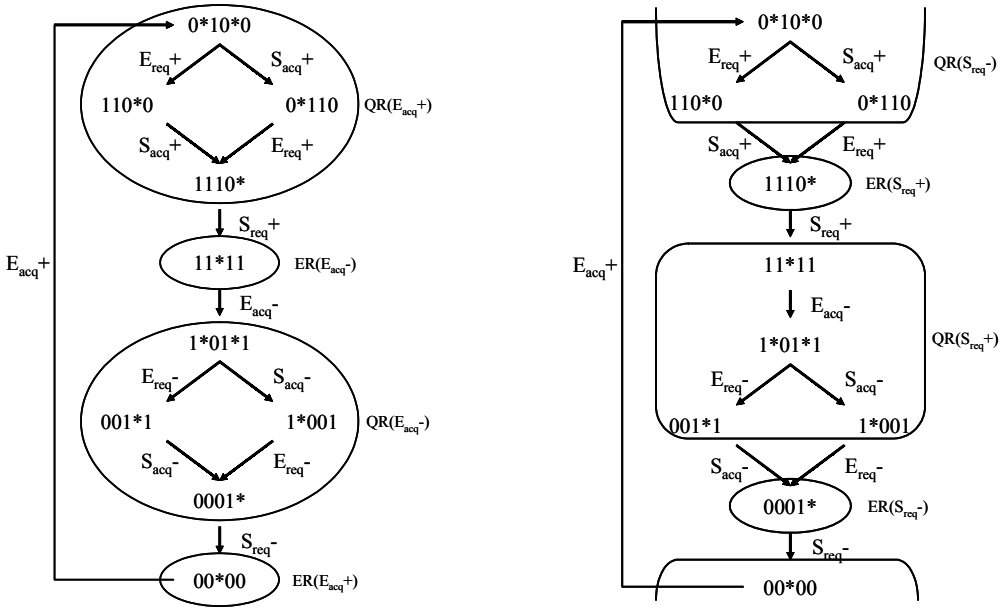


figure 65. Fonctions d'état suivant du protocole WCHB.

Les fonctions d'état (cf. figure 65) suivant de S_{req} et de E_{acq} sont représentées par les tableaux de Karnaugh suivant.

2.2.2.4. Tableaux de Karnaugh pour le protocole WCHB

S_{acq} S_{req} \ E_{acq} E_{req}	00	01	11	10
00	0*	1	1	-
01	0	-	-	0
11	0	-	1*	0
10	-	1	1	-

S_{acq} S_{req} \ S_{req} E_{req} E_{acq}	00	01	11	10
00	0	0	0	-
01	1*	-	-	1
11	1	-	1	1
10	-	0	0*	-

figure 66. Tableaux de Karnaugh du protocole WCHB.

Enfin il reste à extraire les équations de validation et d'invalidation des signaux de sorties S_{req} et E_{acq} à partir des tableaux de la figure 66 en choisissant les regroupements judicieux qui respectent les règles énoncées plus haut.

2.2.2.5. Règles de production et circuit pour le protocole WCHB

S_{acq} S_{req} \ E_{acq} E_{req}	00	01	11	10
00	0*	1	1	-
01	0	-	-	0
11	0	-	1*	0
10	-	1	1	-

S_{acq} S_{req} \ S_{req} E_{req} E_{acq}	00	01	11	10
00	0	0	0	-
01	1*	-	-	1
11	1	-	1	1
10	-	0	0*	-

Ainsi les règles de production pour la sortie E_{acq} sont :

$$\neg S_{req} \rightarrow E_{acq}+$$

$$S_{req} \rightarrow E_{acq}-$$

D'où,il ressort que E_{acq} est le signal inversé de S_{req}.

Les règles de production de S_{req} sont :

$$E_{req} \wedge S_{acq} \rightarrow S_{req}+$$

$$\neg E_{req} \wedge \neg S_{acq} \rightarrow S_{req}-$$

Elles correspondent à la porte de Muller symétrique à deux entrées.

Le circuit final, figure 67, sera donc le même circuit que celui déjà présenté dans l'étude des différents protocoles et de leur circuit de contrôle associé. Ce contrôleur montre bien en effet la correspondance entre le protocole de type « early » et le protocole WCHB.

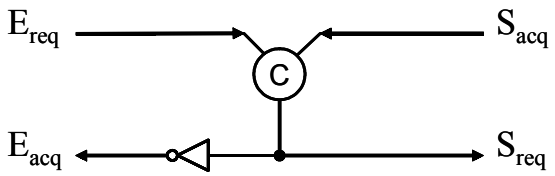


figure 67. Circuit de contrôle du protocole WCHB.

Une remarque sur l'initialisation de ce circuit : sachant que l'environnement fournit un acquittement à '1' pour le canal S et un signal de requête à '0' pour le canal d'entrée E, il est nécessaire de forcer la sortie de la porte de Muller à '0'. Pour cela la porte de Muller est dotée d'une remise à zéro commandée par un signal noté ResetB.

2.3. Protocole PCHB

2.3.1. Méthode de Martin

L'expansion des communications pour ce protocole est la suivante :

$$*\left[\left[S_{acq} \wedge E_{req}\right];S_{req}+;E_{acq}-;\left[\neg S_{acq}\right];S_{req}-;\left[\neg E_{req}\right];E_{acq}+\right]$$

Pour ce protocole une attention plus particulière est à porter sur sa spécification. Entre l'environnement et le circuit il y a, à priori, aucune information sur la causalité de l'affectation E_{acq}- et la garde [$\neg S_{acq}$]. Ce problème se retrouve dans l'établissement de la règle de production minimale pour S_{req}-. Les autres ne présentent aucune difficulté.

Pour la règle de production : $\neg S_{acq} \rightarrow S_{req}-$

FS(S_{req}-) : $\lfloor \neg S_{acq} \rfloor S_{req}-;\lfloor \neg E_{req} \rfloor E_{acq}+;\lfloor E_{req} \rfloor$

CS(S_{req}-) : $\lfloor S_{acq} \wedge E_{req} \rfloor S_{req}+;$

L'intersection de ces deux ensembles d'états n'étant pas nulle, il faut renforcer la garde de S_{req}-. Ici, $\neg E_{acq}$ renforce la garde $\neg S_{acq}$ car il faut être sûr d'avoir acquitter l'entrée avant de remettre à zéro le canal de sortie S. Le nouveau FS(S_{req}-) devient :

$\neg E_{acq} \wedge \neg S_{acq} \rightarrow S_{req}-$

FS(S_{req}-) : $\lfloor \neg S_{acq} \rfloor S_{req}-;\lfloor \neg E_{req} \rfloor$

Avec le même « conflicting set » CS(S_{req}-), la fenêtre de S_{req} – est fermée.

Une remarque concernant la règle de production E_{acq}+. La spécification demande que la remise à '0' du canal E se fasse après la remise à '0' du canal de sortie. Pour obtenir cette

contrainte, il faut renforcer la garde par $\neg S_{req}$. Ainsi, même si dans certains cas l'entrée répond plus vite que la sortie, il faut s'assurer que ce point du protocole est bien respecté.

Au final les règles de production obtenues sont :

$E_{req} \wedge S_{acq} \rightarrow S_{req}+$
$\neg E_{acq} \wedge \neg S_{acq} \rightarrow S_{req}-$
$\neg E_{req} \wedge \neg S_{req} \rightarrow E_{acq}+$
$S_{req} \rightarrow E_{acq}-$

Le circuit obtenu est alors celui de la figure 68 :

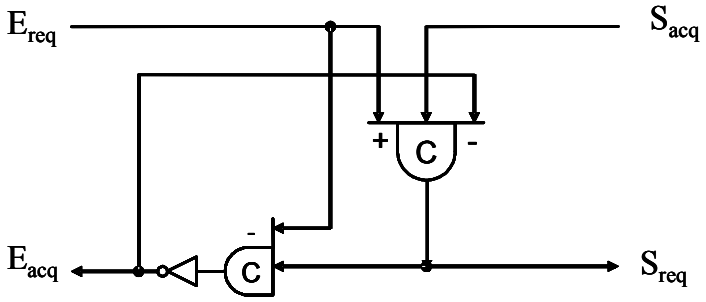


figure 68. Circuit d'un contrôleur PCHB.

2.3.2. Méthode STG

Il s'agit de la même spécification qu'au paragraphe 3.6.2 du chapitre II.

Ce protocole permet de terminer la communication sur la sortie S_{req}- dès que celle–ci est acquittée (S_{acq}-) et sans attendre l'invalidité de l'entrée (E_{req}-). Ensuite, on repositionne à '1' l'acquittement de l'entrée (E_{acq}+) une fois que les données sont invalidées (E_{req-}). Ce protocole sera plus rapide que le précédent si la sortie de l'étage est plus rapide que son entrée. Dans ce cas la sortie sera remise à zéro sans attendre la remise à zéro de l'entrée. On peut exploiter cet avantage lorsqu'il y a un grand nombre d'entrées par rapport au nombre de sorties [MART 97].

2.3.2.1. STG du protocole PCHB

Voici figure 69 la traduction sous forme de STG de la spécification du protocole PCHB.

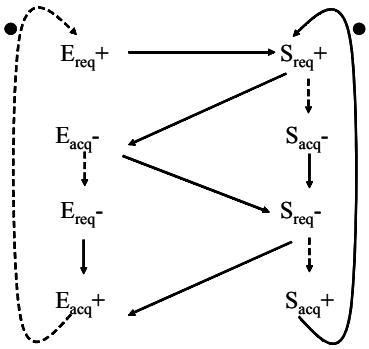


figure 69. STG du protocole PCHB.

2.3.2.2. Graphe d'états du protocole PCHB

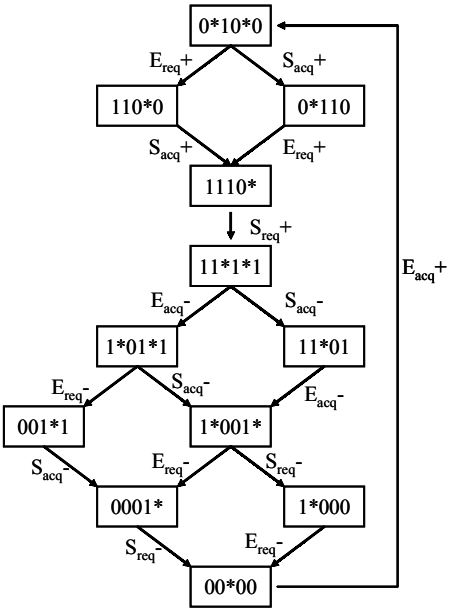


figure 70. Graphe d'états du protocole PCHB.

« E_{req} E_{acq} S_{acq} S_{req} » est le vecteur des signaux.

Le graphe de la figure 70 reste CSC. On cherche maintenant les fonctions d'état suivant. On utilise toujours ce même graphe.

2.3.2.3. Fonction d'état suivant

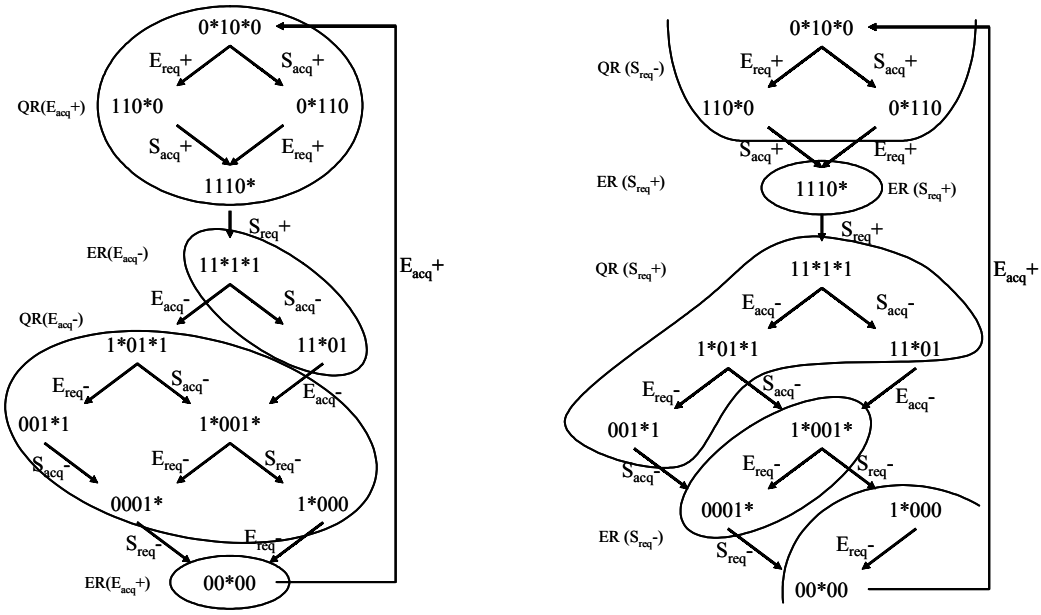


figure 71. Régions d'excitation et de repos des sorties du protocole PCHB.

2.3.2.4. Tableaux de Karnaugh pour le protocole PCHB

S_{acq} S_{req} \ E_{acq} (Ereq Eacq)	00	01	11	10
00	0*	1	1	0
01	0	-	1*	0
11	0	-	1*	0
10	-	1	1	-

S_{acq} S_{req} \ S_{req} (Ereq Eacq)	00	01	11	10
00	0	0	0	0
01	1*	-	1	1*
11	1	-	1	1
10	-	0	0*	-

figure 72. Tables de Karnaugh des sorties pour le protocole PCHB.

2.3.2.5. Règles de production et circuit pour le protocole PCHB

S_{acq} S_{req} \ E_{acq} (Ereq Eacq)	00	01	11	10
00	0*	1	1	0
01	0	-	1*	0
11	0	-	1*	0
10	-	1	1	-

S_{acq} S_{req} \ S_{req} (Ereq Eacq)	00	01	11	10
00	0	0	0	0
01	1*	-	1	1*
11	1	-	1	1
10	-	0	0*	-

Les règles de production afin de dériver les portes de Muller généralisées sont :

$$\neg E_{req} \wedge \neg S_{req} \rightarrow E_{acq}+$$

$$S_{req} \rightarrow E_{acq}-$$

La porte qui découle de ces deux règles de production est une porte de Muller dont la dissymétrie se trouve sur l'affectation à '0' de E_{acq}. En effet, seule la requête du canal S provoque l'acquittement du canal d'entrée E. Cette porte présenterait deux inverseurs sur ses entrées. Une solution simple consiste à translater ces inverseurs sur la sortie de la porte. Sa fonctionnalité s'en trouve un peu modifiée. On applique cette opération sur le circuit final.

$$E_{req} \wedge S_{acq} \rightarrow S_{req}+$$

$$\neg E_{acq} \wedge \neg S_{acq} \rightarrow S_{req}-$$

Cette porte comporte deux équations pour le passage à un et à zéro de sa sortie. Seule l'entrée S_{acq} intervient pour les deux affectations. Voici le circuit résultant de cette synthèse figure 73.

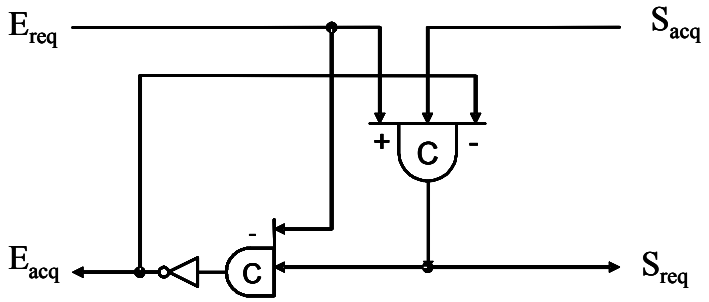


figure 73. Circuit d'un contrôleur PCHB.

2.4. Protocole PCFB

2.4.1. Méthode de Martin

Ce dernier protocole, étudié avec la méthode de synthèse de Martin ajoute une variable interne afin de différencier les phases montantes des deux canaux mis en jeu de leur phase descendante (remise à zéro). Ces dernières s'exécutant en parallèle il faut pouvoir distinguer le fait que les deux protocoles soient terminés. Cette variable est appelé ici En.

$$*\left[\left[S_{acq} \wedge E_{req}\right];S_{req}+;E_{acq}-;En-;\left[\neg En\right];\left(\left[\neg S_{acq}\right];S_{req}-\right),\left(\left[\neg E_{req}\right];E_{acq}+\right);En+\right]$$

Le rôle de En se justifie ainsi : reprenons ce HSE sans cette variable :

$$*\left[\left[S_{acq} \wedge E_{req}\right];S_{req}+;E_{acq}-;\left(\left[\neg S_{acq}\right];S_{req}-\right),\left(\left[\neg E_{req}\right];E_{acq}+\right)\right].$$

Si le canal S très rapide par rapport à E, dès l'affectation de $S_{req}+$, le protocole sur S (c'est-à-dire $[\neg S_{acq}]$; $S_{req}-$) peut très bien terminer sans avoir eu $E_{acq}-$. Dans ce cas extrême, un

protocole sur S peut très bien recommencer sans qu'il y ait eu un échange sur E. C'est pour cela qu'il faut s'assurer à l'intérieur du contrôleur que les quatre phases s'exécutent une seule fois par cycle pour les canaux concernés.

Les règles de production initiales du protocole PCFB avec En sont alors :

$$S_{acq} \wedge E_{req} \rightarrow S_{req}+$$

$$\neg S_{acq} \rightarrow S_{req}-$$

$$\neg E_{req} \rightarrow E_{acq}+$$

$$S_{req} \rightarrow E_{acq}-$$

$$E_{acq} \wedge \neg S_{req} \rightarrow En+$$

$$\neg E_{acq} \rightarrow En-$$

En appliquant l'algorithme de Martin, les règles de production suivantes offrent une fermeture de fenêtre pour chacune des affectations :

$$S_{acq} \wedge E_{req} \rightarrow S_{req}+$$

$$\neg S_{acq} \wedge \neg En \rightarrow S_{req}-$$

$$\neg E_{req} \wedge \neg En \rightarrow E_{acq}+$$

$$S_{req} \wedge E_{req} \rightarrow E_{acq}-$$

$$E_{acq} \wedge \neg S_{req} \rightarrow En+$$

$$\neg E_{acq} \wedge S_{req} \rightarrow En-$$

Ces règles donnent le circuit de la figure 74 après l'étape de réduction des opérateurs:

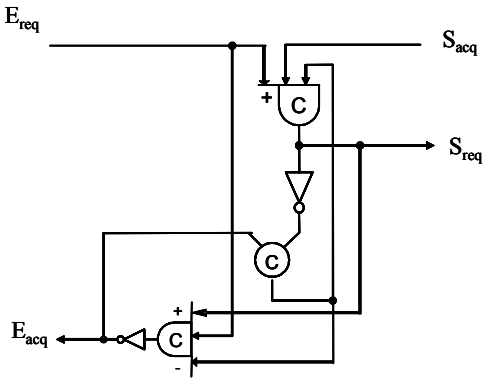


figure 74. Contrôleur du protocole PCFB.

Remarque : une symétrisation a été faite ici pour S_{req}. La variable En intervient aussi dans le passage à '1' de S_{req}.

2.4.2. Méthode STG

2.4.2.1. STG du protocole PCFB

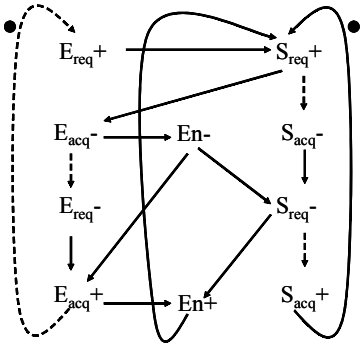


figure 75. STG du protocole PCFB.

2.4.2.2. Graphe d'états du protocole PCFB

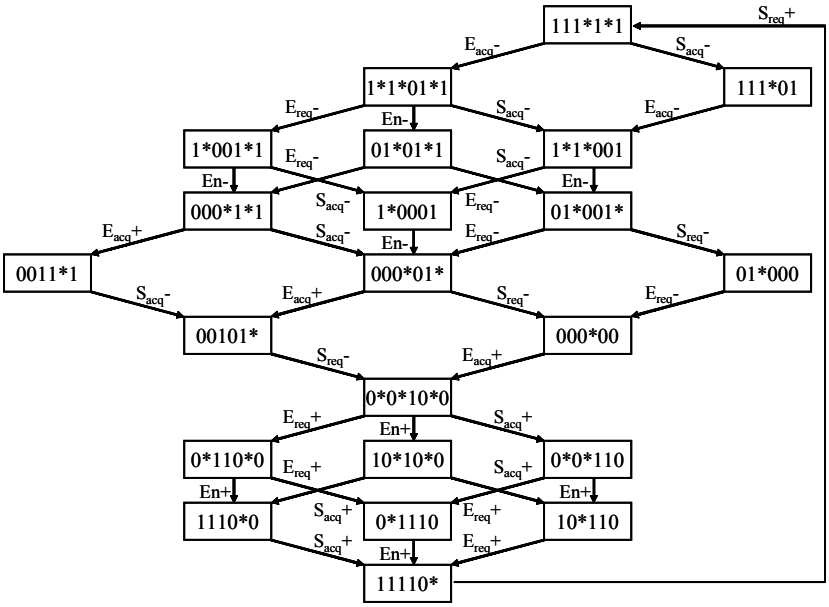


figure 76. Diagramme d'états du protocole PCFB.

Rappel : « E_{req} E_{acq} S_{acq} S_{req} »

2.4.2.3. Fonctions d'état suivant

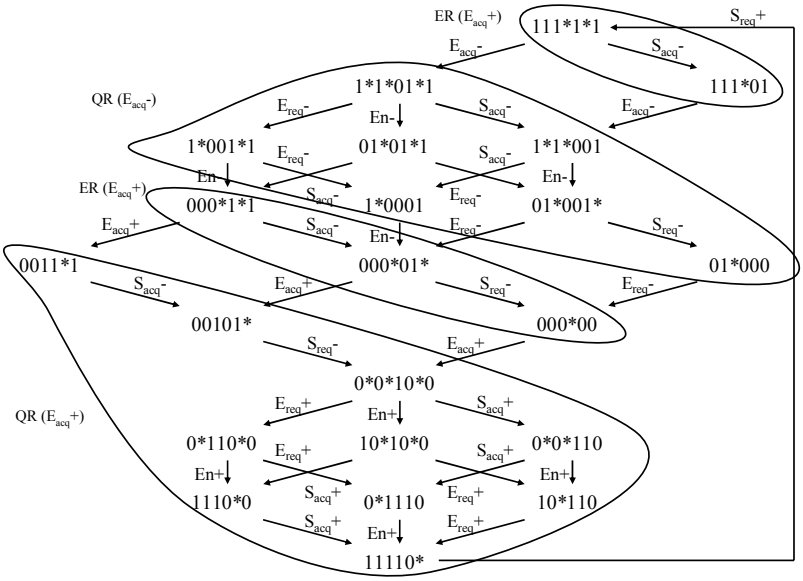


figure 77. ER et QR de la sortie E_{acq}.

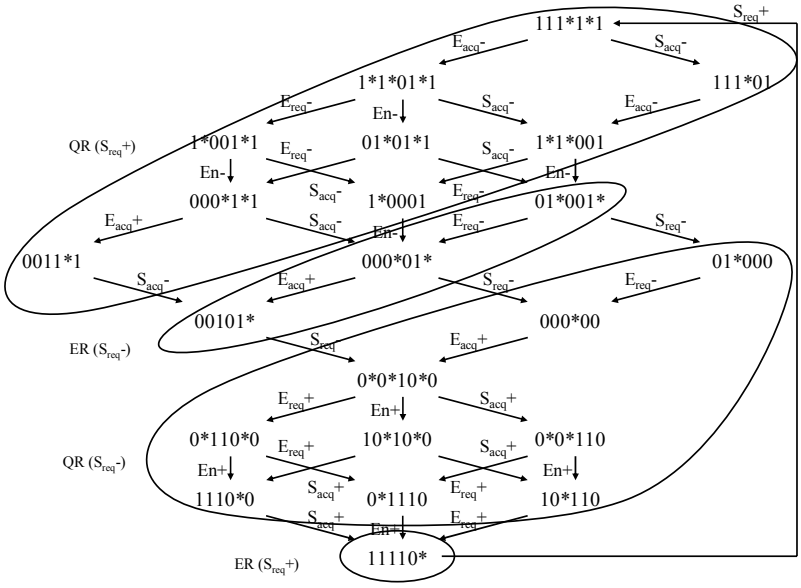


figure 78. ER et QR de la sortie S_{req}.

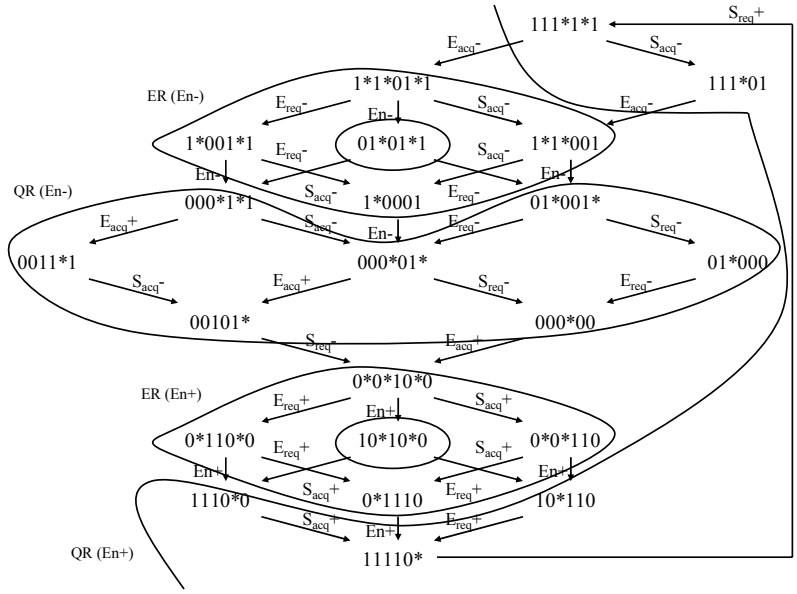


figure 79. ER et QR de la sortie En.

Ici, il y a deux sorties et une variable d'état (En) à synthétiser.

2.4.2.4. Tableaux de Karnaugh pour le protocole PCFB

S_{acq} S_{req} \ E_{acq} E_{req}	00	01	11	10
00	0*	1	1	0
01	0*	1	-	0
11	0*	1	-	0
10	-	1	1	-

En=0

S_{acq} S_{req} \ E_{acq} E_{req}	00	01	11	10
00	-	1	1	-
01	0	-	1*	0
11	0	-	1*	0
10	-	1	1	-

En=1

figure 80. Tableaux de Karnaugh pour la sortie E_{acq}.

S_{acq} S_{req} \ S_{req} E_{req} E_{acq}	00	01	11	10
00	0	0	0	0
01	1*	1*	-	1*
11	1	1	-	1
10	-	0	0	-

En=0

S_{acq} S_{req} \ S_{req} E_{req} E_{acq}	00	01	11	10
00	-	0	0	-
01	1	-	1	1
11	1	-	1	1
10	-	0	0*	-

En=1

figure 81. Tableaux de Karnaugh pour la sortie S_{req}.

En	00	01	11	10
00	0	0*	0*	0
01	0	0	-	0
11	0	0	-	0
10	-	0*	0*	-

En=0

En	00	01	11	10
00	-	1	1	-
01	1*	-	1	1*
11	1*	-	1	1*
10	-	1	1	-

En=1

figure 82. Tableaux de Karnaugh pour la variable En.

2.4.2.5. Règles de production et circuit pour le protocole PCFB

Commençons par le signal E_{acq} dont voici les regroupements.

E_{acq}	00	01	11	10
00	0*	1	1	0
01	0*	1	-	0
11	0*	1	-	0
10	-	1	1	-

En=0

E_{acq}	00	01	11	10
00	-	1	1	-
01	0	-	1*	0
11	0	-	1*	0
10	-	1	1	-

En=1

figure 83. Dérivation des règles de production de E_{acq}.

L'équation de set pour E_{acq} est unique : $\neg E_{req} \wedge \neg En \rightarrow E_{acq}+$.

Par contre, pour le reset, deux regroupements respectant les conditions sont possibles : $S_{req} \wedge En \rightarrow E_{acq}-$ ou, $S_{req} \wedge E_{req} \rightarrow E_{acq}-$.

Cependant, pour effectuer l'acquittement du canal d'entrée, il faut s'assurer qu'une requête sur ce canal a bien été faite. Cela oriente, dans ce cas, le choix sur la deuxième règle de production. Finalement, la porte que l'on utilise est une Muller dissymétrique à trois entrées comportant deux dissymétries. La sortie est complémentée.

La dérivation pour les deux autres signaux est plus simple :

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	0	0	0	0
01	1*	1*	-	1*
11	1	1	-	1
10	-	0	0	-

En=0

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	-	0	0	-
01	1	-	1	1
11	1	-	1	1
10	-	0	0*	-

En=1

figure 84. Dérivation des règles de production de S_{req}.

Cela donne :

$S_{acq} \wedge E_{req} \wedge En \rightarrow S_{req}+$.

$\neg S_{acq} \wedge \neg En \rightarrow S_{req}-$.

Le signal En est important dans la première règle de production, car entre deux cycles du protocole, on peut avoir une nouvelle requête sur E et le canal de sortie S peut être prêt pour une nouvelle communication, alors que le cycle précédent n'est pas terminé. Ici, il faut donc s'assurer que la variable En est bien à '1' avant de commencer une nouvelle communication.

La variable d'état En est affectée par les règles de production suivantes :

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	0	0*	0*	0
01	0	0	-	0
11	0	0	-	0
10	-	0*	0*	-

En=0

$S_{acq}S_{req}$ \ $E_{req}E_{acq}$	00	01	11	10
00	-	1	1	-
01	1*	-	1	1*
11	1*	-	1	1*
10	-	1	1	-

En=1

figure 85. Dérivation des règles de production de En.

La symétrie des regroupements offre sans surprise des règles de production symétriques qui donne une porte de Muller symétrique à deux entrées.

$E_{acq} \wedge \neg S_{req} \rightarrow En+$.

$\neg E_{acq} \wedge S_{req} \rightarrow En-$.

Le circuit final de cette synthèse donne donc celui de la figure 86:

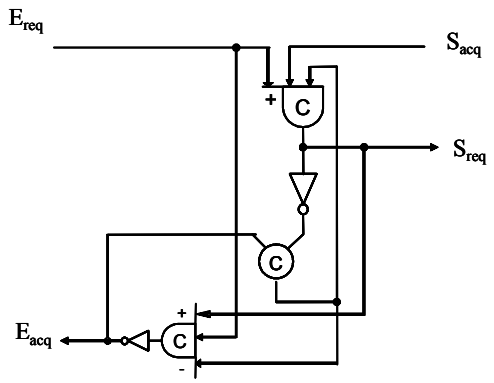


figure 86. Contrôleur du protocole PCFB.

Il est nécessaire d'initialiser ces portes à '0' à l'aide d'un signal ResetB.

3. Conclusion

Pour les protocoles séquentiel, WCHB, PCHB, PCFB deux types de synthèses ont été détaillés. Ces synthèses conduisent à des circuits (les contrôleurs) qui sont fonctionnellement corrects et respectent les protocoles spécifiés sous la forme d'une expansion des communications ou d'un graphe de transition des signaux.

Il est à noté qu'initialement des protocoles comme PCHB et PCFB ont été introduit sous le formalisme HSE. Leur synthèse par la méthode STG a nécessité la réécriture du protocole sous la forme d'un diagramme de transitions des signaux. Les difficultés de cette transformation ont été explicitées et résolus.

Il ressort de ces contrôleurs, le besoin de portes logiques simples et spécifiques aux circuits asynchrones intégrant la mémorisation d'une information et un comportement dissymétrique dû aux spécifications des protocoles. L'ensemble de ces cellules est développé au chapitre V.

Chapitre IV

Etude des structures de pipelines non linéaires

La communication entre différents processus se fait par l'intermédiaire du canal de communication. L'ensemble des processus et des canaux forme le pipeline. Dans sa configuration la plus simple, le pipeline est linéaire, c'est-à-dire qu'un étage émetteur est relié à un seul étage récepteur. Les possibilités qu'offre ce schéma de circuit ne permettent pas de couvrir l'ensemble des circuits réalisables. Pour associer plusieurs pipelines entre eux, des blocs spécifiques sont indispensables.

Les différents circuits dans ce chapitre servent de blocs de base à la réalisation de circuits asynchrones complexes [SPAR 01]. Ils sont présentés ici sous forme élémentaire, mais ils peuvent être étendus à un nombre quelconque d'entrées et de sorties. Ces structures de contrôle mettent en œuvre, des fourches, des jonctions, des choix ou des sélections afin de réaliser du multiplexage ou du démultiplexage.

Toutes ces structures sont réalisées pour les protocoles quatre phases séquentiel, WCHB, PCHB et PCFB. Leur fonctionnalité est spécifiée en CHP pour des canaux de type simple rail (cf. CHP dans les annexes). Les canaux sont implémentés par le couple de signaux requête/acquittement. On ne s'intéresse qu'à la partie de contrôle (on ne représente pas la partie chemin de données). L'extension au type multi rails est du ressort de l'outil TAST.

1. La fourche

La fourche est l'opérateur de composition parallèle. Un programme simple en CHP qui met en œuvre ce type de structure est le suivant : $*[E?;S1!,S2!]$. Il faut attendre la présence d'une requête sur le canal d'entrée E pour diffuser une donnée ou commencer une communication sur les canaux de sorties S1 et S2. Une fois les deux communications terminées sur S1 et S2, il faut acquitter E pour autoriser une nouvelle exécution du programme. La figure 87 représente le symbole d'une fourche et le réseau de pétri équivalent du programme CHP.

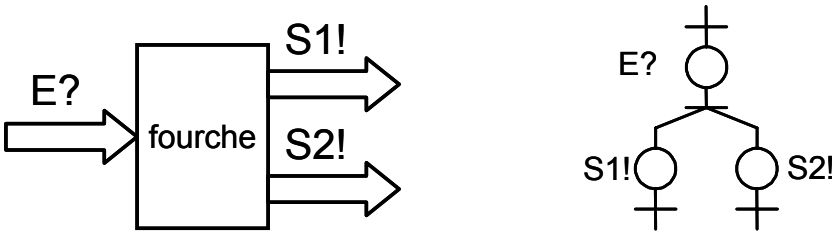


figure 87. Schéma et réseau de Pétri d'une fourche.

1.1. La fourche pour le protocole séquentiel

La requête se résume à un simple fil pour la requête qui est distribuée au deux canaux de sorties S1 et S2.

L'acquittement est un rendez-vous entre les signaux d'acquittement des canaux S1 et S2.Le circuit correspondant est décrit figure 88.

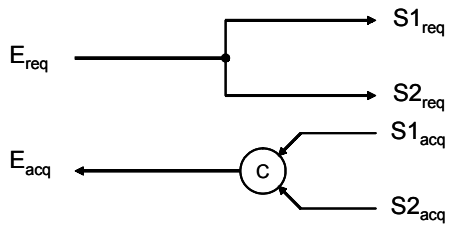


figure 88. La fourche pour le protocole séquentiel.

1.2. La fourche pour le protocole WCHB

La requête du canal d'entrée est propagée sur toutes les Muller à deux entrées pour chacun des canaux de sortie (S1 etS2). L'acquittement du circuit est la Muller complémentée des signaux de requête générés. L'inversion est due à la polarité inversée de l'acquittement. Le circuit est décrit figure 89.

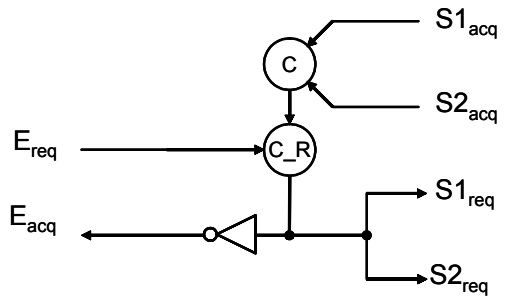


figure 89. Circuit d'une fourche en WCHB.

1.3. La fourche pour le protocole PCHB

C'est le premier protocole qui dans le contrôleur initial présentait des portes de Muller dissymétriques. Elles se retrouvent pour la génération des requêtes au niveau des canaux de sortie et pour l'acquittement du canal d'entrée E. La Muller symétrique correspond toujours au rendez-vous des signaux de requête pour créer l'acquittement E_{acq}. Elle possède autant d'entrées que de canaux de sortie (cf. figure 90).

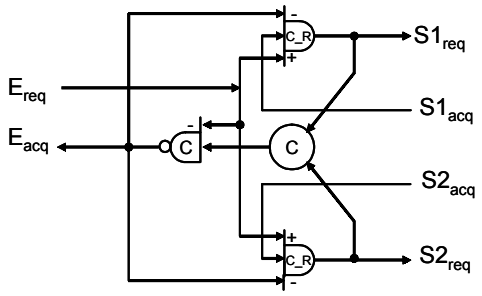


figure 90. Schéma d'une fourche en PCHB.

1.4. La fourche pour le protocole PCFB

Dernier protocole utilisé par l'outil TAST, il offre la communication la plus concurrente entre les canaux d'entrées et de sortie. La partie fixe de la fourche pour ce protocole comprend la variable d'état En qui correspond au rendez-vous de l'acquittement de E avec les requêtes des canaux de sortie. La représentation de la fourche est montrée figure 91.

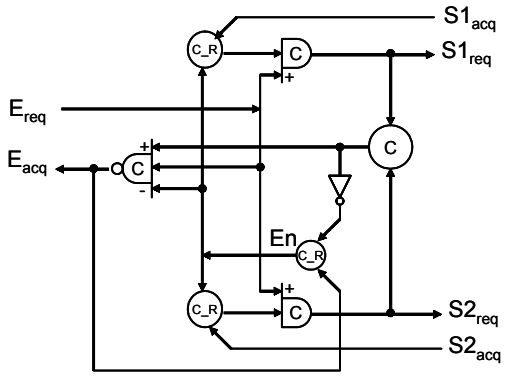


figure 91. Schéma d'une fourche en PCFB.

2. La jonction

Ce bloc est le schéma dual de la fourche. En effet, elle propose deux canaux en entrée et un en sortie. Une fois que les deux entrées reçoivent une requête on réalise une communication sur la sortie. La jonction est l'opérateur rendez-vous au niveau du canal. Il permet la synchronisation entre plusieurs canaux, ou entre plusieurs processus concurrents. Le programme CHP correspondant est : $*[E1?, E2?; S!]$. La figure 92 présente le symbole de la jonction et son réseau de pétri.

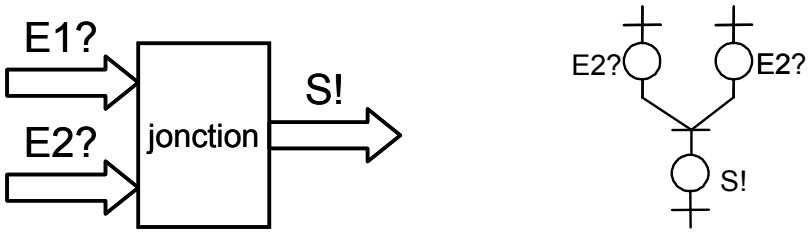


figure 92. Schéma bloc et réseau de Pétri de la jonction.

2.1. Protocole séquentiel

La spécification de la jonction au niveau du contrôle se résume à établir le rendez-vous entre deux requêtes. L'implémentation matérielle est une porte de Muller symétrique avec autant d'entrées que de canaux d'entrée et l'acquittement un simple fil. La figure 93 montre bien la dualité avec la fourche de la figure 88.

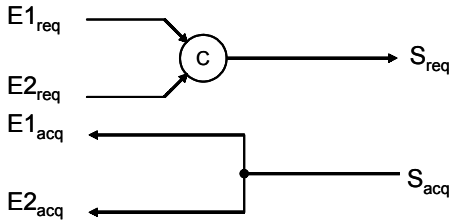


figure 93. Fonction jonction pour le protocole séquentiel.

2.2. Protocole WCHB

La jonction pour le protocole WCHB met en œuvre le contrôleur simple vu au chapitre précédent. Ce contrôleur est commandé par la présence d'une requête sur chacun des canaux d'entrée. Il vient le circuit de la figure 94.

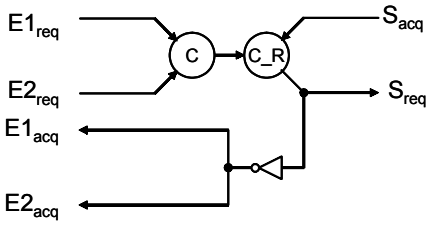


figure 94. Fonction jonction en WCHB.

2.3. Protocole PCHB

Les mêmes remarques que pour le protocole WCHB sont à faire pour les figure 95 et figure 96 qui implémente la jonction dans les deux derniers protocoles. Une fois que les deux requêtes $E1_{req}$ et $E2_{req}$ sont à '1', le protocole s'applique.

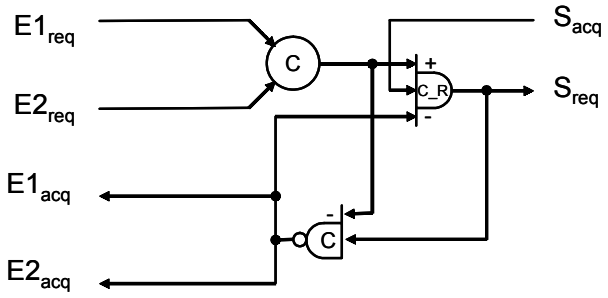


figure 95. Fonction jonction en PCHB.

2.4. Protocole PCFB

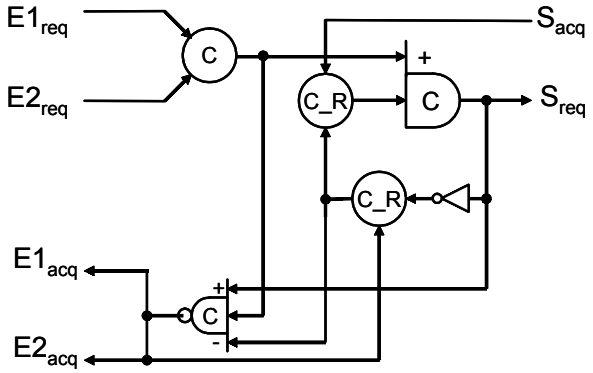


figure 96. Fonction jonction en PCFB.

3. Le multiplexage

Le multiplexeur propage sur une de ses sorties la valeur reçue sur son entrée. Le choix de la sortie dépend de la valeur du sélecteur. Au niveau de la communication on attendra une requête du sélecteur et de l'entrée du multiplexeur avant de réaliser l'envoie de la donnée sur la sortie voulue. Le multiplexeur est une "jonction contrôlée" ou conditionnelle. Le multiplexage permet de réaliser des choix commandés par un canal ou de façon plus générale par une expression booléenne. Les structures abordées ne mettent en jeu que des choix qui sont exclusifs. Dans l'exécution du programme CHP, une seule des branches d'une commande gardée est vraie en même temps.

Voici le programme CHP du multiplexage de deux canaux E1 et E2 sur un canal de sortie S et contrôlé par le canal C : $*[C?c; c=0 \rightarrow [\ E1?; S!\ ] @ c=1 \rightarrow [\ E2?; S!\ ]\ ]$

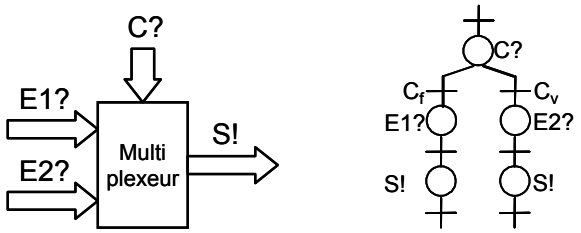


figure 97. Schéma bloc et réseau de Pétri du multiplexeur.

La figure 97 présente le symbole du multiplexeur accompagné du réseau de Pétri correspondant au programme CHP.

3.1. Protocole séquentiel

Les variables C_v et C_f sont les valeurs logiques du canal C lorsque celui-ci est actif (présence d'une requête) : C_v pour c='1' et Creq='1' ; C_f pour c='0'et Creq='1'). Elles permettent l'aiguillage des entrées vers les sorties. Elles sont présentes sur les circuits proposés par la suite.

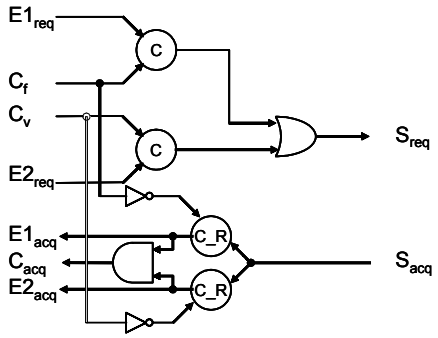


figure 98. Multiplexeur pour le protocole séquentiel.

La figure 98 présente un multiplexeur pour le protocole séquentiel. Une fois que le canal C est actif, en fonction de sa valeur, le canal E1 ou E2 est utilisé et communique avec le canal de sortie S. L'exclusivité du choix est assurée par C_v et par C_f qui n'est jamais vrai en même temps.

3.2. Protocole WCHB

Voici figure 99, le multiplexeur deux voies qui suit le protocole WCHB. Le canal S est prêt. Une fois que C est actif, la valeur qu'il transporte sélectionne le canal d'entrée qui doit communiquer et envoyer une donnée via le canal S.

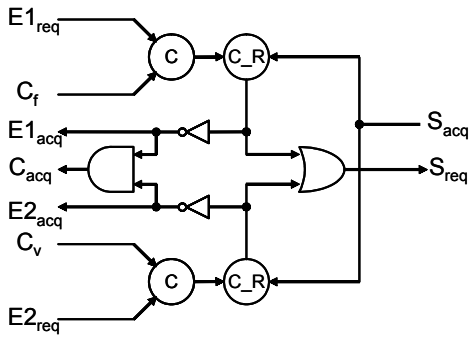


figure 99. Multiplexeur pour le protocole WCHB.

3.3. Protocole PCHB

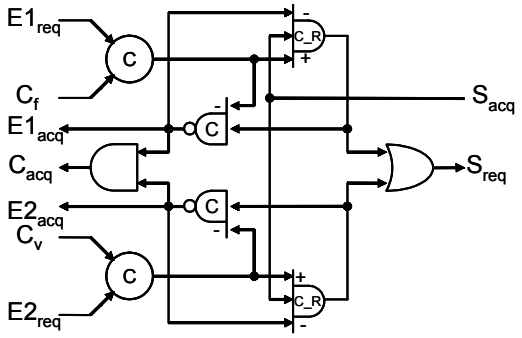


figure 100.Multiplexeur pour le protocole PCHB.

3.4. Protocole PCFB

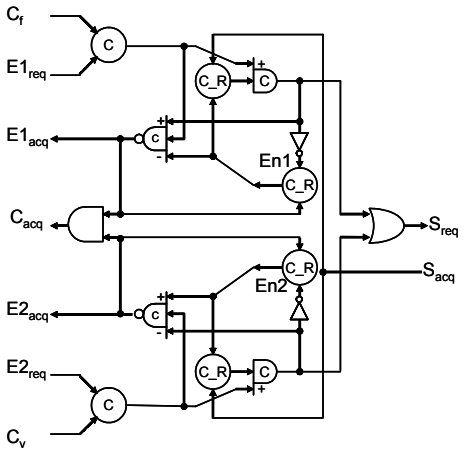


figure 101.Multiplexeur pour le protocole PCFB.

L'extension du multiplexeur à un nombre plus grand d'entrées se fait par la copie du contrôleur (propre au protocole) pour chaque canal d'entrée. Les portes « OU » et « ET » ont un nombre d'entrées égal au nombre de canaux d'entrée.

4. Le démultiplexage

Le démultiplexeur joue le rôle d'une fourche contrôlée. Le contrôle se porte sur les sorties du circuit. Le canal C sélectionne le canal de sortie qui doit réaliser une communication avec son environnement. Cette structure est utilisée pour la diffusion d'une donnée venant d'un canal d'entrée unique vers plusieurs canaux en sortie.

Voici le programme CHP d'un démultiplexeur qui met en jeu deux canaux d'entrées (la source E, le contrôle C) et deux sorties S1 et S2.

$$*[C?c; c=0 \rightarrow [\ E?; S1!\ ] @ c=1 \rightarrow [\ E?; S2!\ ]].$$

Le symbole et le réseau de Pétri du démultiplexeur sont présentés figure 102.

Le canal C permet de réaliser la sélection du canal de sortie. Son type en CHP est du double rails (1 parmi 2). Les variables C_v (C=1) et C_f (C=0) se retrouvent dans les circuits des contrôleurs pour les quatre protocoles étudiés.

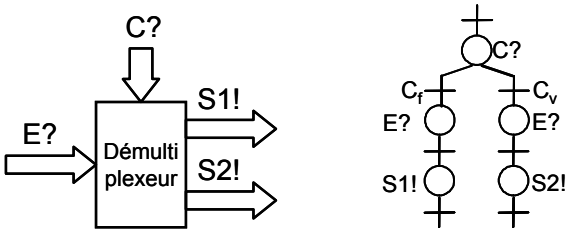


figure 102.Schéma bloc et réseau de Pétri du démultiplexeur.

4.1. Protocole séquentiel

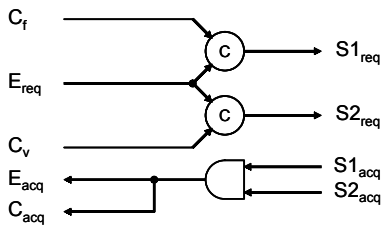


figure 103. Multiplexeur pour le protocole séquentiel.

4.2. Protocole WCHB

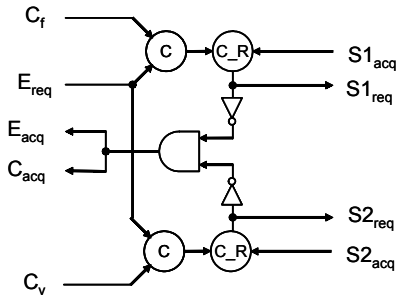


figure 104. Multiplexeur pour le protocole WCHB.

4.3. Protocole PCHB

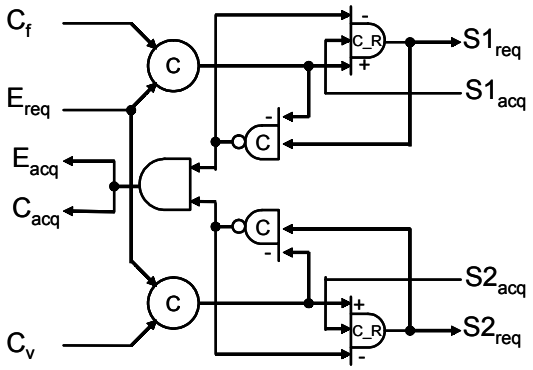


figure 105. Multiplexeur pour le protocole PCHB.

4.4. Protocole PCFB

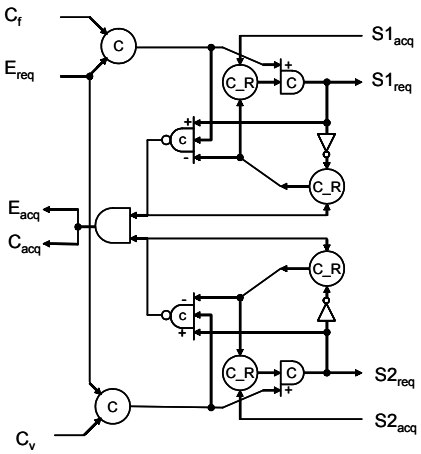


figure 106. Multiplexeur pour le protocole PCFB.

Tout comme le multiplexeur il est possible d'étendre cette structure de contrôle à un nombre plus important de sorties. Le canal de contrôle suit un codage un parmi n, l'acquittement est le ET de chacun des acquittements des canaux de sorties.

5. Conclusion

La réalisation de tout type de circuits asynchrones passe par l'utilisation de fonctions de base qui permettent d'implémenter toute sorte de structures de contrôles. Ces fonctions permettent de réaliser la communication d'un bloc vers plusieurs (fourche), de synchroniser plusieurs sources indépendantes (jonction) entre elles ou de contrôler une communication par la sélection de plusieurs entrées (multiplexage), ou plusieurs sorties (démultiplexage).

Le chapitre a présenté toutes ses fonctions spécifiées en CHP et synthétisées suivants les protocoles séquentiel, WCHB, PCHB, PCFB. Le but est d'identifier toutes les cellules nécessaires pour implémenter ces circuits de contrôle. Tous les circuits utilisent des canaux de communication simple rail (requête/acquittement). L'extension des canaux au type multi-rails est prise en compte par l'outil TAST. Les contrôleurs mettent en jeu un nombre réduit de canaux en entrée comme en sortie. Il est possible d'augmenter ce nombre et de dériver à partir des circuits exposés dans cette partie le contrôleur étendu correspondant. Cela implique dans certains cas un agrandissement du nombre d'entrées pour les portes mises en jeu (logique combinatoire, Muller).

Au final les cellules spécifiques nécessaires pour la bonne réalisation des circuits sont des portes de Muller symétriques et dissymétriques. Le chapitre suivant se consacre à la spécification de bibliothèques qui mettent en œuvre ces cellules.

Chapitre V

Bibliothèques de cellules spécifiques basées cellules standard et LuTs de FPGA

L'étude menée sur les différents systèmes de communication asynchrones du chapitre II, les moyens de les synthétiser au chapitre III et leurs mises en forme dans les structures de base des pipelines du chapitre IV montrent le besoin d'un certain nombre de cellules spécifiques indispensables pour la réalisation concrète de circuits asynchrones complexes : les cellules de Muller symétriques et dissymétriques.

Le travail de cette thèse a pour but de mettre en oeuvre la couche « back-end » de l'outil de synthèse de circuits asynchrones TAST, développé au laboratoire. Il s'agit, en effet, de créer le lien entre TAST et les outils standard de simulation (électrique ou fonctionnelle) et de placement routage. Cette interface est constituée des bibliothèques de cellules spécifiques aux circuits asynchrones décrites sous différentes vues :

- Fonctionnelle : modèle VHDL comportemental des portes.
- Structurelle à base de cellules standard : pour des circuits à applications spécifiques.
- Structurelle à bases de LUT (« Look Up Table ») : pour des circuits ciblés FPGA.

Elles sont utilisées par tous les outils qui interviennent dans le flot de conception.

La présentation et les étapes du flot de conception TAST sont reprises ici pour justifier les différentes vues des bibliothèques nécessaires dans le flot de conception. Le point commun de ces bibliothèques est la nomenclature unique. Une fois celle-ci présentée, chaque modèle de bibliothèque est décrit.

1. TAST : un outil de synthèse automatique de circuits asynchrones

TAST (« TIMA Asynchronous Synthesis Tool ») est un ensemble d'applications dédié à la conception et à la synthèse de circuits asynchrones. Il est composé d'un compilateur et d'un synthétiseur offrant la possibilité de cibler plusieurs formats de sorties à partir d'un langage de description haut niveau : le CHP (cf. annexes). Le flot de synthèse est organisé autour des réseaux de Pétri et des graphes de données. La synthèse de circuits asynchrones est basée sur la spécification DTL (« Data Transfert Level ») ([DINH 02c]). La spécification DTL fournit un ensemble de règles qui garantissent que le réseau de Pétri et les graphes de données sont synthétisables vers des circuits asynchrones.

TAST cible plusieurs types de circuits à partir de la description sous la forme de processus communicants :

- Une description comportementale décrite en VHDL simulable.
- Les circuits quasiment insensibles aux délais sous la forme d'une description VHDL au niveau porte.
- Les circuits micro pipeline sous la forme d'une description VHDL au niveau porte.

Le flot de TAST est présenté figure 107 :

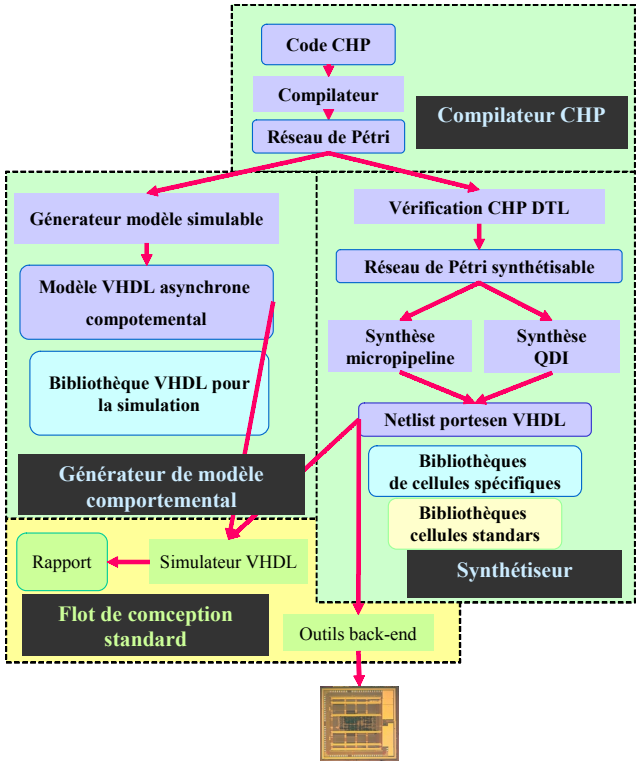


figure 107. Flot de synthèse de l'outil TAST.

TAST accepte en entrée des systèmes décrits en CHP. Le CHP, de l'anglais « Communicating Hardware Processes », est un langage de description haut niveau de circuits basés sur les processus communicants. TAST lie les différents points suivants : les protocoles de communication, le codage des données, la spécification de l'indéterminisme, la hiérarchie, la gestion de projet (TAST 02).

Le programme CHP est compilé et transformé en un ensemble de réseaux de Pétri (structures de contrôle et de choix, séquentialité, parallélisme) et de graphes de données (instructions associées aux places des réseaux de Pétri). De là, la spécification en transformée en un modèle VHDL comportemental afin de pouvoir vérifier par simulation la correction fonctionnelle du circuit obtenu. Le programme VHDL est alors utilisé dans les outils standard

de simulation. Deux paquetages de fonctions sont ici utiliser pour réaliser les opérations arithmétiques et les conversions des types de données.

Une fois la validation de la description CHP effectuée, il faut vérifier qu'elle est compatible avec les règles de spécification DTL. Si cela n'est pas le cas, Le concepteur transforme le code CHP ([DINH 02b]). La synthèse permet le choix entre deux classes de circuits asynchrones : la classe de circuits micro pipeline et quasiment insensibles aux délais (QDI).

Le circuit, une fois synthétisé, se présente sous la forme d'une description VHDL au niveau porte (« netlist ») composées de portes de Muller et de cellules standard. La netlist obtenue doit être à son tour validée. A ce niveau du flot de conception, la connaissance de la technologie cible n'étant pas obligatoire. Pour simuler le circuit synthétisé, une bibliothèque de cellules décrites en VHDL comportemental est disponible. Elle est présentée au paragraphe 5.

Enfin, le lien avec les étapes back-end du flot est réalisé par une nouvelle bibliothèque de cellules décrites cette fois de façon structurelle (à base de cellules standard). La connaissance de la technologie cible est maintenant nécessaire pour utiliser les outils standard associés qui permettent entre autre de faire de la simulation électrique et du placement routage.

TAST permet de cibler également les techniques de prototypage rapide et de mapper les circuits synthétisés sur les FPGA standard (circuits synchrones). Pour parvenir à cela, une dernière version de bibliothèque de cellules est disponible (§7). Elles sont construites sur la base de LUT (« Look-Up Table »).

Pour mener à bien la conception et la réalisation d'un circuit asynchrone avec TAST, trois bibliothèques génériques sont nécessaires :

- une bibliothèque fonctionnelle pour la simulation indépendante de la technologie cible.
- une bibliothèque structurelle pour la réalisation d'ASIC construite à partir de cellules standard.
- Une bibliothèque structurelle pour la réalisation de FPGA construite à partir de LUTs.

2. Organisation des bibliothèques

Pour chacune des bibliothèques l'organisation est faite de la façon suivante :

On distingue les portes de Muller symétriques et dissymétriques. Toutes les portes sont avec une sortie simple ou complémentée (inverseur en sortie), avec un signal d'initialisation à '1' (Set) ou à '0' (Resetb). Cela représente six modèles différents pour une porte avec un nombre donné d'entrées.

Les portes de Muller symétriques et dissymétriques à deux et trois entrées sont incluses dans les bibliothèques.

Les différents modèles de bibliothèques disponibles sont les suivants :

Une version, appelée « fonctionnelle », permet de prendre le résultat de synthèse de TAST et d'en faire une validation fonctionnelle. Cette version de la bibliothèque comprend des cellules décrites en VHDL fonctionnelle. Elle ne cible aucune technologie particulière. Elle est utilisable par n'importe quel outil de simulation VHDL après avoir pris la précaution de la compiler avec le compilateur adéquat. L'ensemble des cellules est réuni dans un paquetage de cellules VHDL.

Une version, appelée « structurelle », permet de réaliser une simulation après la synthèse en tenant compte du temps de traversée de chacune des portes du circuit. Cette bibliothèque permet d'importer directement la description au niveau porte des circuits directement dans les outils de « back-end » standard tel que Cadence afin de réaliser des simulations électriques, et faire le placement routage du circuit. Cette description décrit les portes de Muller à base de cellules logiques standard.

Une dernière version, cible les systèmes de prototypage rapide (FPGA). Cette bibliothèque comprend les portes de Muller implémentées à base de cellules élémentaires de circuit FPGA (LUT).

3. Nomenclature et symboles des portes de Muller

Afin de pouvoir passer d'un niveau de description à un autre sans avoir à appliquer de transformations sur les circuits produits par TAST, une nomenclature générique a été définie et mise en place pour nommer les cellules spécifiques asynchrones.

Le but de cette nomenclature est de pouvoir identifier la porte de Muller facilement. Le nom générique de la porte de Muller est donc :

TAST_MULLERM(D)(XP)(YN)(B)(_R)(_S)

Les paramètres entre parenthèses sont optionnels. Les paragraphes suivants décrivent chacun de ces paramètres.

3.1. Préfixe

TAST_MULLERM(D)(XP)(YN)(B)(_R)(_S)

Le préfixe TAST évoque ici l'outil de synthèse développé au sein du laboratoire TIMA présenté plus haut au (§1). La description des circuits au niveau porte présente la liste des portes utiles à la constitution du circuit. Afin de pouvoir utiliser cette description sur plusieurs outils toutes les instances de cellules possèdent ce préfixe et indiquen clairement qu'on fait appel à une bibliothèque de cellules génériques ne ciblant pour le moment aucune technologie ou modèle de fonctionnement.

3.2. Nombre d'entrées

Le nombre d'entrées des portes de Muller correspond au chiffre « M » qui suit le préfixe de TAST_MULLER dans la nomenclature des portes de Muller : ((TAST_MULLER**M**(D)(XP)(YN)(B)(_R)(_S)). Ce nombre d'entrées ne tient pas compte d'un éventuel signal Set ou ResetB. Une porte de Muller à 3 entrées est donc appelée TAST_MULLER3 (figure 108).

Sur la figure 108, le C à l'intérieur de chaque symbole permet d'identifier qu'il s'agit d'une porte de Muller. En effet, celle-ci est aussi appelée « C-element » [MULL 65].

3.3. Politique d'initialisation

Deux valeurs d'initialisation de la sortie des portes de Muller sont disponibles : une initialisation à '0' ou à '1'. Pour cela, une entrée à la cellule est ajoutée. Elle permet, pendant la phase de mise en route du circuit, de forcer à sa valeur d'initialisation les portes souhaitées. Ce signal porte un nom différent et fonctionne suivant une certaine polarité en fonction de la valeur que l'on désire en sortie de la porte.

3.3.1. Initialisation à zéro : ResetB

Dans ce cas, la sortie de la porte de Muller est forcée à '0'. Le nom de l'entrée additionnelle est ResetB. Cette spécificité de la porte se retrouve dans son nom en ajoutant le suffixe « _R » (TAST_MULLERM(D)(XP)(YN)(B) **_R** (_S)).

Le signal d'initialisation s'appelle « Reset » pour dire que la valeur imposée est zéro et la lettre B signifie que l'entrée « ResetB » est active à l'état bas. Au démarrage du circuit, l'entrée ResetB sera à '0' et forcera à '0' la sortie de toutes les portes qui sont connectées à ce signal. Une fois de retour à '1', les sorties changent en fonction des entrées.

Par exemple, la figure 108 montre une porte de Muller à 3 entrées avec initialisation à '0' : TAST_MULLER3_R.

3.3.2. Initialisation à un : Set

Le signal Set est utilisé pour forcer la sortie des portes à '1'. Le signal est actif à l'état haut. Au niveau du noms des portes de bibliothèques ceci est signalé par le suffixe « _S » (TAST_MULLERM(D)(XP)(YN)(B)(_R) **_S**).

Le signal Set au niveau circuit retourne à '0' après la phase d'initialisation du circuit. Si on compare la forme d'onde du signal Set, c'est la forme inversée du signal ResetB. En reprenant la même porte que précédemment avec une affectation à '1' de la sortie : TAST_MULLER3_S.

Sur la figure 108, les deux suffixes _R et _S se retrouvent dans les symboles indiquant à quelle valeur on initialise les portes. Au niveau des bibliothèques de cellules l'entrée d'initialisation est présente sur les portes.

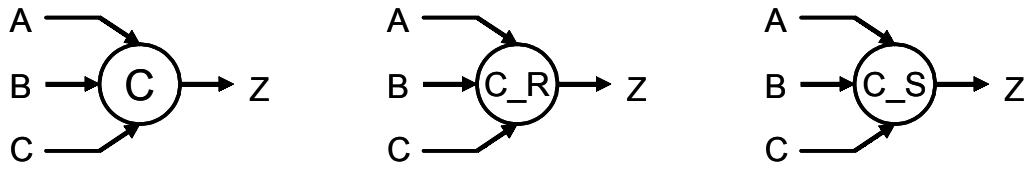


figure 108. Exemples de portes de Muller à trois entrées.

3.4. Sortie inversée

Le moyen de connaître si la sortie de la porte est inversée peut aussi se lire dans le nom des portes de Muller. Le suffixe est un « B » (TAST_MULLERM(D)(XP)(YN) **B** (_R)(_S)). Ce « B » exprime la présence d'un inverseur en sortie de la porte de Muller. La même convention d'initialisation que pour les portes non complémentées est appliquée. A savoir que la sortie de la porte TAST_MULLR3B_R est initialisée à '0' pendant que le signal « ResetB » est nul. Au niveau du symbole, cela se traduit par une bulle positionnée sur la sortie (figure 109).

figure 109. TAST_MULLER3B et TAST_MULLER3B_R.

3.5. Expression de la dissymétrie

Les portes de Muller dissymétriques rencontrées dans les contrôleurs de pipeline possèdent aussi une appellation particulière. Cette dissymétrie s'applique sur les conditions d'affectation de la sortie. Trois types de dissymétrie existent :

- une dissymétrie pour le passage à '1' de la sortie (appelée positive),
- une dissymétrie pour le passage à '0' de la sortie (appelée négative),
- une double dissymétrie (à la fois positive et négative).

Dans tous ces cas, la lettre « D », pour dissymétrie, vient suivre le nombre total d'entrées dans la nomenclature ((TAST_MULLERM **D** (XP)(YN)(B)(_R)(_S)).

3.5.1. Dissymétrie positive de la porte de Muller

Ici, la porte de Muller fait intervenir un nombre d'entrées plus important pour faire commuter la sortie à '1'. Certaines de ces entrées n'interviennent donc que pour affecter la sortie à '1'. C'est précisé au niveau de l'appellation en renseignant le paramètre « XP ». « X désignant le nombre d'entrées qui ne servent que pour faire passer la sortie à '1'. Le « P » pour dire qu'il s'agit d'une dissymétrie positive.

Au niveau du symbole, c'est le nombre d'entrées excentrées du corps de la porte ((TAST_MULLERM **D** **XP** (YN)(B)(_R)(_S)). Le signe « + » pour positif. Le nombre d'entrées intervenant dans les deux affectations de la sortie est déterminé en soustrayant au nombre total d'entrées M le nombre d'entrées décentrées X. Par exemple, la porte TAST_MULLER2D1P (figure 110) est une porte de Muller dissymétrique à deux entrées avec les deux entrées qui interviennent pour faire passer la sortie à '1' et une entrée seulement intervient pour refaire passer la sortie à '0'.

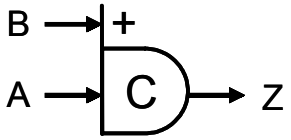


figure 110. Porte de Muller dissymétrique : TAST_MULLER2D1P.

3.5.2. Dissymétrie négative de la porte de Muller

Le principe est le même que pour la dissymétrie positive, mais le paramètre « YN » renseigne sur le nombre d'entrées faisant passer la sortie à '0' ((TAST_MULLERM **D** (XP) **YN** (B)(_R)(_S)).

Au niveau symbole, le nombre d'entrées « Y » décentré se retrouve sur la porte avec un signe moins pour rappeler la dissymétrie négative. Exemple : TAST_MULLER2D1N (figure 111) est une Muller dissymétrique négative avec une entrée dissymétrique.

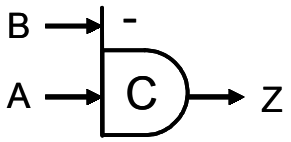


figure 111. Porte de Muller dissymétrique : TAST_MULLER2D1N.

3.6. Convention de nommage

Le nombre d'entrées pour ces portes de Muller étant très variable et leur fonction n'étant pas constante, il n'est pas forcément aisé de déterminer si une entrée intervient pour la validation ou le retour à zéro de la porte. Ainsi, la politique pour nommer les entrées est définie de la façon suivante :

- Les noms des entrées suivent l'ordre alphabétique.
- Le nommage commence par les entrées appartenant au corps principal de la porte (intervenant pour le passage à '1' et à '0' de la sortie).
- Ensuite, ce sont les entrées n'intervenant que pour faire passer la sortie à '1' (les entrées décentrées côté « plus » de la porte).
- Enfin les entrées intervenant pour la remise à zéro de la sortie sont affectées (entrées décentrées côté « moins » de la porte).

Ainsi, la porte TAST_MULLER3D1P1N, figure 112, possède trois entrées. Sur le corps central se trouve l'entrée A, sur le côté « plus », l'entrée B , et sur le côté « moins », l'entrée C. La sortie est appelée Z. Les entrées d'initialisation ont un nom invariable et suivent la convention vue plus haut. Ces deux derniers signaux sont toujours déclarés dans l'entité VHDL avant la déclaration de la sortie (Z).

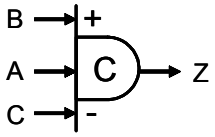


figure 112. Symbole de la TAST_MULLER3D1P1N.

Cette nomenclature, de par sa généricité, permet d'étendre, si besoin est, le nombre d'entrées. Cette extension est d'autant plus facile pour la description fonctionnelle de chacune des portes. Pour les autres modèles de bibliothèque, l'extension du nombre d'entrées est plus difficile et moins automatique. Cependant une porte avec un grand nombre d'entrées peut se décomposer en porte de Muller à deux entrées.

4. Décomposition des portes de Muller

L'objet de ce paragraphe, est de montrer qu'une porte de Muller à M entrées peut se décomposer en un arbre de portes de Muller élémentaires qui possèdent un nombre d'entrées plus petit. Dans le cadre de ce travail, les portes de Muller élémentaires se limitent à deux et trois entrées. La démonstration est faite dans un premier temps pour les Muller symétriques, puis pour les Muller dissymétriques.

Dans un premier temps, seule la porte de Muller à deux entrées est l'élément de base de la décomposition.

E = {entrées de la porte de Muller}. Le nombre d'éléments est égal à M.

$E \subset$ {lettre de l'alphabet} \ {Z}. Z est réservé pour la sortie.

E_i est un élément de E. E_1 correspond à A, E_2 à B, etc.

Les règles de production d'une Muller à M entrées sont les suivantes :

$$E_1 \wedge E_2 \wedge \cdots \wedge E_{M-1} \wedge E_M \rightarrow Z+$$

$$\neg E_1 \wedge \neg E_2 \wedge \cdots \wedge \neg E_{M-1} \wedge \neg E_M \rightarrow Z-$$

Pour une porte de Muller à deux entrées il vient :

$$E_1 \wedge E_2 \rightarrow Z+$$

$$\neg E_1 \wedge \neg E_2 \rightarrow Z-$$

En regroupant deux à deux les termes E_i intervenant dans les règles de production et en les renommant, il y a réduction du nombre d'éléments dans la règle de production. Cela se traduit au niveau circuit par un premier étage de portes à deux entrées. Le nombre d'entrées passe de M à M/2 ou M/2+1 si il y a un nombre impair d'entrées. L'équation générale devient (les sorties du premier étage sont appelées S_i).

$$S_1 \wedge S_2 \wedge \cdots \wedge S_{\frac{M}{2}-1} \wedge S_{\frac{M}{2}} \rightarrow Z+$$

$$\neg S_1 \wedge \neg S_2 \wedge \cdots \wedge \neg S_{\frac{M}{2}-1} \wedge \neg S_{\frac{M}{2}} \rightarrow Z-$$

Ensuite, il faut renouveler l'étape de décomposition précédente jusqu'à obtenir deux éléments par règle de production.

Par exemple, une porte de Muller à quatre entrées, figure 113, se décompose en trois portes de Muller à deux entrées. Dans ce cas, la structure d'arbre est composée de deux étages. La même démonstration est possible en incorporant comme élément de décomposition supplémentaire, une Muller à trois entrées. Les bibliothèques développées se limitent aux portes à deux et trois entrées.

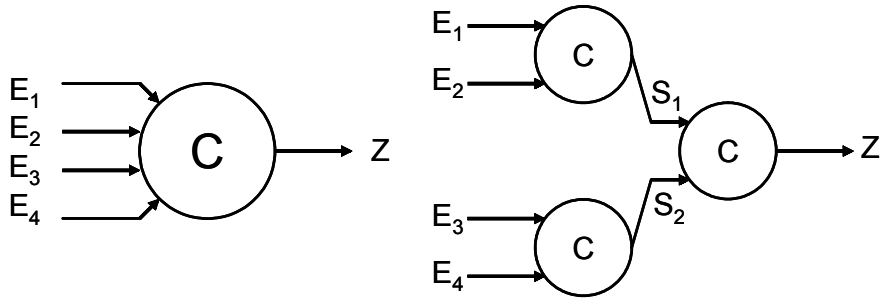


figure 113.Décomposition d'une TAST_MULLER4.

Pour les portes de Muller dissymétriques, plusieurs situations sont à prendre en considération. Soit, sur l'ensemble des entrées, une seule entrée intervient dans la dissymétrie positive ou négative ou dans les deux. C'est le cas qui a été rencontré dans les circuits étudiés. Soit plus d'une entrée interviennent dans les dissymétries positives ou négatives.

Dans le premier cas, pour une dissymétrie positive (TAST_MULLERMD1P), les règles de production de la porte de Muller sont :

$$E_1 \wedge E_2 \wedge \cdots \wedge E_{M-1} \wedge E_M \rightarrow Z+$$

$\neg E_1 \wedge \neg E_2 \wedge \cdots \wedge \neg E_{M-1} \rightarrow Z-$

Les M-1 premières entrées interviennent pour le passage à '0' et à '1' de la sortie. Il est donc possible de réécrire les règles de production ainsi :

$S \wedge E_M \rightarrow Z+$

$\neg S \rightarrow Z-$

S est la sortie intermédiaire d'une porte de Muller symétrique à M-1 entrées. Il reste dans les règles de production ci-dessus l'expression d'une TAST_MULLER2D1P qui fait parti de l'ensemble des portes de base de la bibliothèque (cf. figure 114). Pour une dissymétrie négative (cf. figure 115) et double (cf. figure 116), cela se décompose de la même façon en se ramenant à une porte TAST_MULLER2D1N pour la dissymétrie négative et à une TAST_MULLER3D1P1N pour la double dissymétrie.

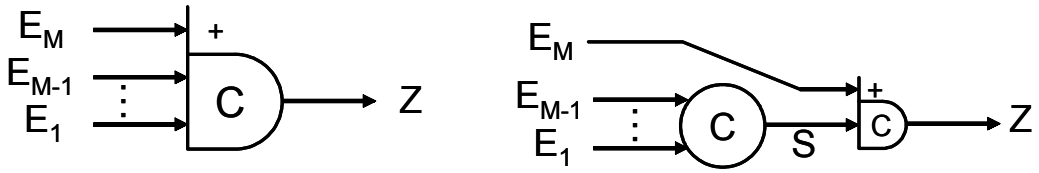


figure 114.Décomposition d'une TAST_MULLERMD1P.

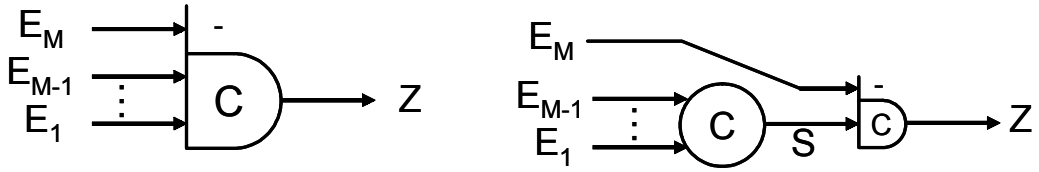


figure 115.Décomposition d'une TAST_MULLERMD1N.

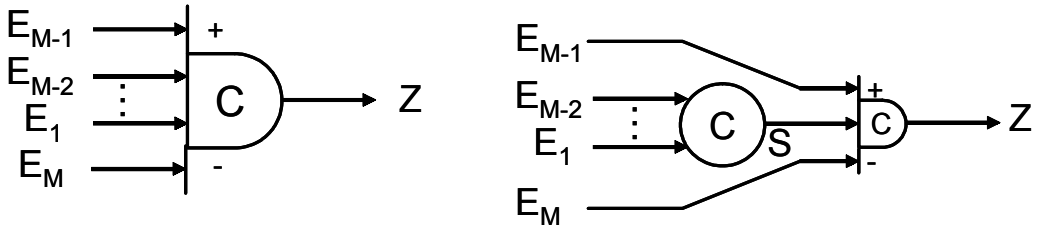


figure 116.Décomposition d'une TAST_MULLERMD1P1N.

Cas où plusieurs entrées interviennent dans la dissymétrie d'une porte de Muller. Soit les données suivantes pour une dissymétrie positive:

X est le nombre d'entrées qui intervient uniquement pour la transition montante de la porte.

W est le nombre d'entrées qui intervient pour les deux transitions. W= M - (X).

Les règles de production d'une TAST_MULLERMDXP sont :

$$E_1 \wedge E_2 \wedge \cdots E_{W-1} \wedge E_W \wedge \cdots \wedge E_{M-1} \wedge E_M \rightarrow Z+$$

$$\neg E_1 \wedge \neg E_2 \wedge \cdots \wedge \neg E_{W-1} \wedge \neg E_W \rightarrow Z-$$

Elles sont équivalentes à :

$$S \wedge \cdots \wedge E_{M-1} \wedge E_M \rightarrow Z+$$

$$\neg S \rightarrow Z-$$

S étant la sortie d'une porte de Muller symétrique à W entrées. Cette porte peut facilement être décomposée en portes élémentaires à deux entrées.

Pour se ramener à une porte de Muller à dissymétrie négative de deux entrées, les X entrées de la dissymétrie doivent être réunies dans une porte ET logique standard.

Au final il faut trois portes pour décomposer cette porte (cf. figure 117) : une Muller symétrique à W entrées, une porte ET à X entrées et une TAST_MULLER2D1P.

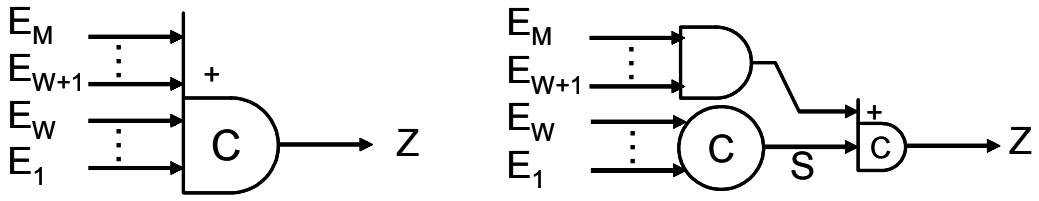


figure 117. Décomposition d'une TAST_MULLERMDXP.

Pour la dissymétrie négative une porte OU sur la branche « - » de la porte remplace la porte ET des portes de Muller avec dissymétrie positive.

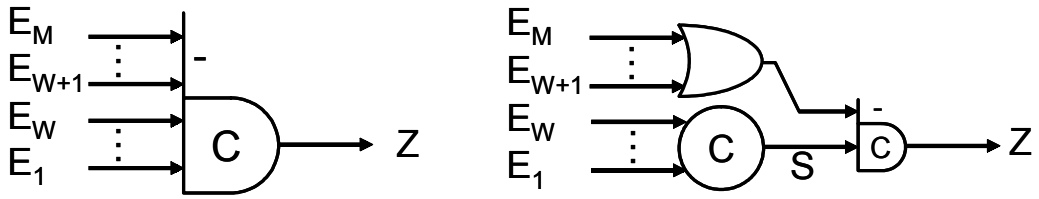


figure 118.Décomposition d'une TAST_MULLERMDXN.

Le cas de la double dissymétrie est une généralisation des deux cas précédents.

Ce paragraphe montre que toute porte de Muller est décomposable en portes de Muller élémentaires à deux entrées pour les Muller symétriques (TAST_MULLER2, TAST_MULLER3); à deux et trois entrées pour les portes de Muller dissymétriques (TAST_MULLER2D1P, TAST_MULLER2D1N, TAST_MULLER3D1P1N). Dans les structures de contrôle, seules les portes avec une dissymétrie d'une seule entrée (TAST_MULLERMD1P, TAST_MULLERMD1N, TAST_MULLERMD1P1N) sont

utilisées. Cela justifie le nombre limité de cellules qui composent les bibliothèques développées dans la suite.

5. Une bibliothèque pour la simulation fonctionnelle

L'outil de synthèse TAST fournit, après synthèse des programmes écrits en CHP, une description au niveau portes du circuit. Avant de poursuivre le flot standard de conception, il faut tout d'abord vérifier la conformité du circuit produit (VHDL comportemental) avec la spécification initiale. Pour toutes les portes définies au paragraphe 2, une description comportementale en VHDL est disponible permettant d'effectuer très vite une validation des programmes avant d'avoir choisi la technologie cible.

5.1. Les portes de Muller symétriques

5.1.1. Portes avec sortie simple

La fonctionnalité de la porte de Muller est la suivante : la valeur de sortie de la porte recopie la valeur de l'entrée lorsque celles-ci sont égales. Dans les autres cas, la porte mémorise sa valeur courante. Le modèle des portes de Muller symétriques possède donc la forme générique suivante.

Reprenons la nomenclature générale pour une porte de Muller symétrique :

TAST_MULLERM(B)(_R)(_S).

E = {entrées de la porte de Muller}. Le nombre d'éléments est égal à M.

$E \subset$ {lettre de l'alphabet} \ {Z}. Z est réservé pour la sortie.

E_i est un élément de E. E_1 correspond à A, E_2 à B, etc.

RESETB : signal d'initialisation à '0' des portes.

SET : signal d'initialisation à '1' des portes.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;


ENTITY Nom_de_la_porte IS
  PORT(
    Z : OUT STD_ULOGIC;              --déclaration de la sortie de la porte
    E1 : IN  STD_ULOGIC;             --déclaration des entrées :autant de fois que la porte
…   EM : IN  STD_ULOGIC              --comporte d'entrées (M fois).
    );
END Nom_de_la_porte;


ARCHITECTURE behaviour OF Nom_de_la_porte IS

  SIGNAL S_Z : STD_ULOGIC;           --Signal interne pour la mémorisation de la sortie
```

```
BEGIN

  S_Z <= '0' AFTER 100 PS WHEN AND_{i∈[1,M]}(E_i ='0') ELSE

         '1' AFTER 100 PS WHEN AND_{i∈[1,M]}(E_i ='1') ELSE

         S_Z;

  Z <= S_Z;                          --Affectation de la sortie avec le signal interne

END behaviour;
```

figure 119.Modèle général d'une porte de Muller décrite en VHDL comportemental.

La description de chaque porte est réalisée au moyen de deux instructions concurrentes (figure 119). La première donne les conditions d'affectation du signal interne « S_Z » en fonction le la valeur des entrées (instruction d'affectation conditionnelle). Le délai permet de rappeler que la traversée de la porte n'est pas instantanée. Ce signal interne correspond en fait à la sortie de la porte. La seconde est l'affectation de la sortie par ce signal interne.

La figure 120 décrit un exemple de porte à trois entrées (TAST_MULLTER3).

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY TAST_MULLER3 IS
  PORT(
    Z : OUT STD_ULOGIC;
    A : IN  STD_ULOGIC;
    B : IN  STD_ULOGIC;
    C : IN  STD_ULOGIC
    );
END TAST_MULLER3;

ARCHITECTURE behaviour OF TAST_MULLER3 IS

  SIGNAL S_Z : STD_ULOGIC;

BEGIN

  S_Z <= '0' AFTER 100 PS WHEN (A = '0') AND (B = '0') AND (C = '0') ELSE

         '1' AFTER 100 PS WHEN (A = '1') AND (B = '1') AND (C = '1') ELSE

         S_Z;
  Z <= S_Z;
END behaviour;
```

figure 120.Description en VHDL fonctionnel de la porte TAST_MULLER3.

5.1.2. Portes avec sortie inversée

Pour les portes avec une sortie inversée le modèle est similaire au précédent. La différence réside dans la valeur à laquelle le signal interne est affecté. Le signal « S_Z » sera affecté à '0', réciproquement à '1', quand toutes les entrées seront à '1', réciproquement à '0'. Voici l'architecture du modèle correspondant (figure 121). La structure de l'entité ne change pas.

```
ARCHITECTURE behaviour OF Nom_de_la_porte IS
```

```
  SIGNAL S_Z : STD_ULOGIC;
BEGIN

  S_Z <= '1' AFTER 100 PS WHEN AND_{i∈[1,M]}(E_i ='0') ELSE

         '0' AFTER 100 PS WHEN AND_{i∈[1,M]}(E_i ='1') ELSE

         S_Z;

  Z <= S_Z;
END behaviour;
```

figure 121.Modèle de la porte de Muller avec une sortie inversée.

5.1.3. Portes avec entrée d'initialisation

Les deux modèles suivants (figure 122, figure 123), présentent les portes qui doivent être initialisées à '0' ou à '1'. Dans l'entité, apparaît du signal d'initialisation (« RESETB » ou « SET »). Dans l'architecture VHDL, il y a une condition supplémentaire qui force la sortie à '0' si RESETB='0', respectivement à '1' si SET='1'et ce, quelque soit la valeur des entrées.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY Nom_de_la_porte IS
  PORT(
    RESETB : IN  STD_ULOGIC;          --signal d'initialisation
    Z : OUT STD_ULOGIC;               --déclaration de la sortie de la porte
    E1 : IN  STD_ULOGIC;              --déclaration des entrées :autant de fois que la porte
…   EM : IN  STD_ULOGIC               --comporte d'entrées (M fois).
    );
END Nom_de_la_porte;

ARCHITECTURE behaviour OF Nom_de_la_porte IS

  SIGNAL S_Z : STD_ULOGIC;

BEGIN

  S_Z <= '0' AFTER 100 PS WHEN RESETB = '0' ELSE

         '0' AFTER 100 PS WHEN (RESETB = '1') AND AND_{i∈[1,M]}(E_i ='0') ELSE

         '1' AFTER 100 PS WHEN (RESETB = '1') AND AND_{i∈[1,M]}(E_i ='1') ELSE

         S_Z;

  Z <= S_Z;

END behaviour;
```

figure 122.Modèle de porte de Muller avec initialisation à '0' de la sortie.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY Nom_de_la_porte IS
  PORT(
    SET : IN  STD_ULOGIC;             --signal d'initialisation
    Z : OUT STD_ULOGIC;               --déclaration de la sortie de la porte
```

```
    E1 : IN  STD_ULOGIC;            --déclaration des entrées :autant de fois que la porte
…   EM : IN  STD_ULOGIC             --comporte d'entrées (M fois).
    );
END Nom_de_la_porte;

ARCHITECTURE behaviour OF Nom_de_la_porte IS

  SIGNAL S_Z : STD_ULOGIC;

BEGIN

  S_Z <= '1' AFTER 100 PS WHEN SET = '1' ELSE

         '0' AFTER 100 PS WHEN (SET = '0') AND AND_{i∈[1,M]}(E_i ='0') ELSE

         '1' AFTER 100 PS WHEN (SET = '0') AND AND_{i∈[1,M]}(E_i ='1') ELSE
         S_Z;

  Z <= S_Z;

END behaviour;
```

figure 123.Modèle de porte de Muller avec initialisation à '0' de la sortie.

L'exemple de la figure 124 présente le code VHDL pour une porte de Muller à deux entrées, sortie complémentée et avec un signal d'initialisation SET.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;


ENTITY TAST_MULLER2B_S IS
  PORT(
    SET : IN  STD_ULOGIC;
    Z   : OUT STD_ULOGIC;
    A   : IN  STD_ULOGIC;
    B   : IN  STD_ULOGIC
    );
END TAST_MULLER2B_S;

ARCHITECTURE behaviour OF TAST_MULLER2B_S IS

  SIGNAL S_Z : STD_ULOGIC;

BEGIN

  S_Z <= '1' AFTER 100 PS WHEN SET = '1' ELSE

         '1' AFTER 100 PS WHEN (SET = '0') AND (A = '0') AND (B = '0') ELSE

         '0' AFTER 100 PS WHEN (SET = '0') AND (A = '1') AND (B = '1') ELSE

         S_Z;

  Z <= S_Z;

END behaviour;
```

figure 124. TAST_MULLER2B_S.

5.2. Les portes de Muller dissymétriques

Le modèle VHDL qui est présenté ici est de nouveau très général.

La façon de modéliser les portes à sortie complémentée avec ou sans entrée d'initialisation reste identique pour les portes de Muller dissymétriques. C'est pourquoi dans ce paragraphe, seules les portes de Muller dissymétriques avec une sortie simple sont présentées.

A partir de la nomenclature :

TAST_MULLERMD(XP)(YN)(B)(_R)(_S)

M est le nombre d'entrées de la porte.

X est le nombre d'entrées qui intervient uniquement pour la transition montante de la porte.

Y est le nombre d'entrées qui intervient uniquement pour la transition descendante de la porte.

W est le nombre d'entrées qui intervient pour les deux transitions. W= M - (X + Y).

F, G, H sont les ensembles des entrées associés à W, X, Y.

$F = \{E_1..E_W\}$:

$G = \{E_{W+1}..E_{W+X}\}$

$H = \{E_{W+X+1}..E_M\}$

Par exemple, si M=6, X=1, Y=3 alors : F={E_1, E_2}={A, B}, G={E_3}={C}, H={E_4..E_6}={D, E, F}.

Ces trois ensembles sont utiles pour définir l'architecture du modèle général comportemental d'une porte de Muller dissymétrique figure 125).

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;


ENTITY Nom_de_la_porte IS
  PORT(
    Z : OUT STD_ULOGIC;              --déclaration de la sortie de la porte
    E1 : IN  STD_ULOGIC;             --déclaration des entrées :autant de fois que la porte
…   EM : IN  STD_ULOGIC              --comporte d'entrées (M fois).
    );
END Nom_de_la_porte;


ARCHITECTURE behaviour OF Nom_de_la_porte IS

  SIGNAL S_Z : STD_ULOGIC;           --Signal interne pour la mémorisation de la sortie

BEGIN

  S_Z <= '0' AFTER 100 PS WHEN AND_{i∈F}(Ei ='0') AND AND_{i∈H}(Ei ='0') ELSE
```

```
        '1' AFTER 100 PS WHEN AND_{i∈F}(E_i ='1') AND AND_{i∈G}(E_i ='1') ELSE
        S_Z;
  Z <= S_Z;                          --Affectation de la sortie avec le signal interne
END behaviour;
```

figure 125. Modèle général d'une porte de Muller dissymétrique.

Finalement, la sortie est affectée à '0' lorsque toutes les entrées de l'ensemble F et H sont à '0'. Elle vaut '1' quand toutes les entrées de l'ensemble F et G sont à '1'.

La figure 126 regroupe toutes les portes de Muller dissymétriques qui ont été effectivement développé pour le « back-end ».

TAST_MULLER2D1P	TAST_MULLER2D1N	TAST_MULLER3D1P1N
TAST_MULLER2D1PB	TAST_MULLER2D1NB	TAST_MULLER3D1P1NB
TAST_MULLER2D1P_R	TAST_MULLER2D1N_R	TAST_MULLER3D1P1N_R
TAST_MULLER2D1PB_R	TAST_MULLER2D1NB_R	TAST_MULLER3D1P1NB_R
TAST_MULLER2D1P_S	TAST_MULLER2D1N_S	TAST_MULLER3D1P1N_S
TAST_MULLER2D1PB_S	TAST_MULLER2D1NB_S	TAST_MULLER3D1P1NB_S

figure 126.Portes de Muller dissymétriques pour la bibliothèque fonctionnelle.

6. Bibliothèque structurelle à base de cellules standard

Le problème de la synthèse des portes de Muller se pose. Ou bien une bibliothèque de cellules au niveau transistor doit être utilisée mais cela implique un temps de développement long pour chaque technologie visée, ou bien les portes sont construites avec les cellules standard mises à disposition par le fondeur. Dans ce cas, chaque porte de la bibliothèque est composée d'une ou de plusieurs portes.

Dans ce travail, le développement de la bibliothèque s'est orienté vers cette deuxième solution. Ainsi chaque porte de Muller est basée sur l'utilisation des portes disponibles dans la bibliothèque du fondeur. Les problèmes d'implémentation des portes de Muller sans aléa (chapitre I) ont été considérés pendant le développement de ces cellules.

Chaque porte est décrite dans un fichier VHDL. Il comprend en en-tête la bibliothèque du fondeur, l'ensemble des cellules standard utilisées ainsi que la façon dont elles sont interconnectées. Le nom des cellules instanciées dans une porte dépend de la bibliothèque du fondeur.

6.1. Les portes de Muller symétriques

Pour chacune des cellules de la bibliothèque, une solution structurelle exempte d'aléa et à base de portes complexes (ET-OU) est proposée. Chaque porte se décline en six versions : sortie simple, inversée, avec signal Set et/ou ResetB.

6.1.1. Muller symétrique à deux entrées

La porte majorité est la cellule la mieux adaptée à la réalisation en porte complexe CMOS de la Muller à deux entrées. La mémorisation de la donnée se fait par le rebouclage de la sortie sur certaines des entrées. Le tableau de la figure 127 montre les versions structurelles de la porte de Muller2. Sa représentation en porte complexe est issue de l'expression booléenne de sa sortie : Z = A.B + A.Z + B.Z

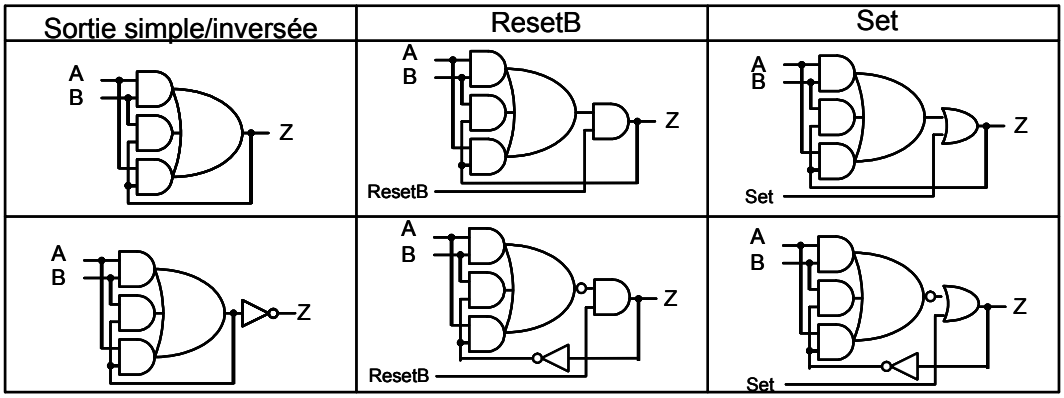


figure 127.Circuit des portes TAST_MULLER2 et ses dérivés.

6.1.2. Muller symétrique à trois entrées

La figure 128 décrit une solution pour implémenter les portes de Muller à trois entrées. L'équation booléenne est : Z = A.B.C + A.Z + B.Z + C.Z .

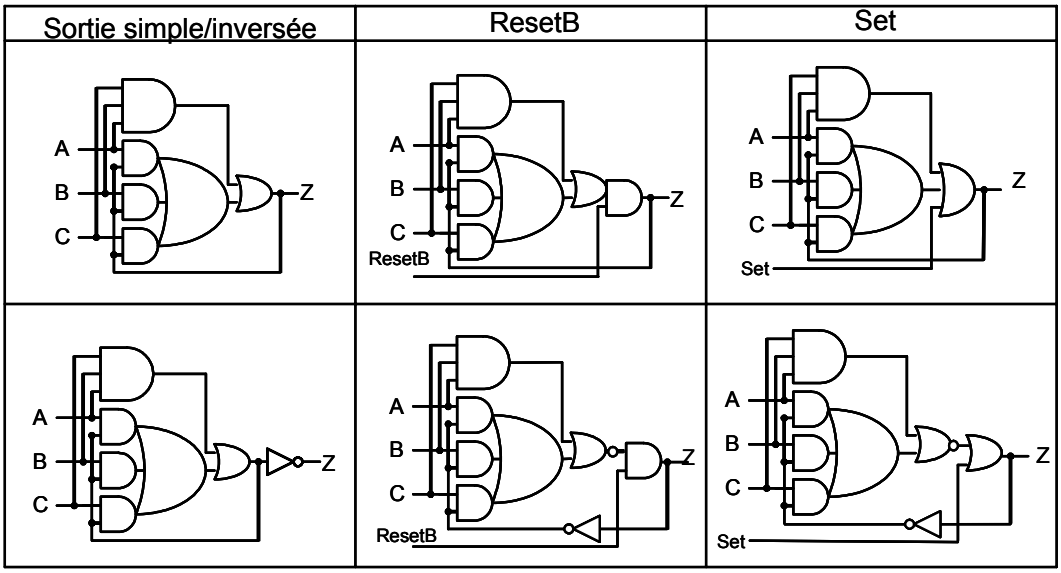


figure 128.Circuit de la porte TAST_MULLER3 et ses dérivés.

Une autre solution, envisagée au cours de cette étude, est de décomposer la porte de Muller à trois entrées en portes de Muller élémentaires à deux entrées. Cette solution permet de réduire considérablement la diversité des cellules nécessaires. Elle a été choisie pour réaliser des portes supérieures à trois entrées tant que l'outil TAST ne fournissait pas des circuits composés uniquement de portes de Muller à deux et trois entrées.

6.2. Les portes de Muller dissymétriques positives

Le tableau de la figure 129 présente les six montages structurels qui implémentent la porte de Muller dissymétrique à deux entrées avec dissymétrie « positive » pour faire passer la sortie à '1'. Le symbole est rappelé figure 110.

L'équation logique est : Z = A.B + A.Z .

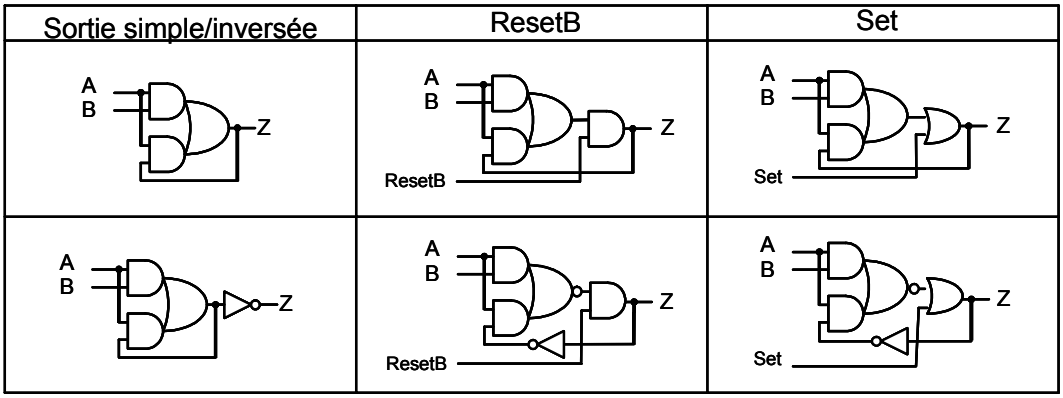


figure 129.Circuit des portes TAST_MULLER2D1P et ses dérivés.

6.3. Les portes de Muller dissymétriques négatives

Le tableau de la figure 130 présente les six montages structurels qui implémentent la porte de Muller dissymétrique à deux entrées avec dissymétrie « négative » pour faire passer la sortie à '0'. Le symbole correspondant à la porte dans sa version la plus simple est rappelé figure 111. L'équation booléenne est : Z = A+B . A+Z . Contrairement aux portes précédentes, le cœur de cette porte est une porte complexe OU/ET.

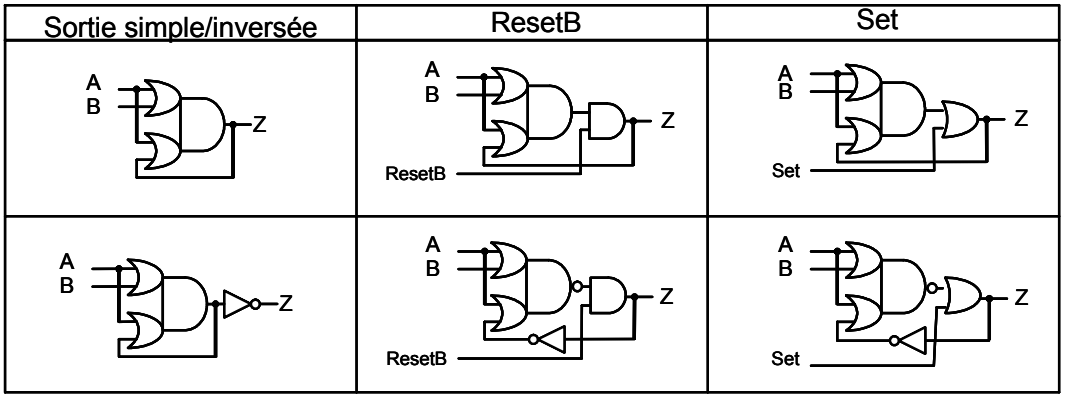


figure 130.Circuit des portes TAST_MULLER2D1N et ses dérivés.

6.4. Les portes de Muller dissymétriques positives/négatives

Ce titre désigne la porte de Muller TAST_MULLER3D1P1N

C'est la seule porte dissymétrique à trois entrées construite dans les bibliothèques spécifiques de cellules asynchrones. Cette porte est l'union des deux portes de Muller « TAST_MULLER2D1P » et « TAST_MULLER2D1N » vu un peu plus tôt. D'autres portes sont évidemment réalisables mais elles se résument à des compositions de portes de Muller symétriques et dissymétriques plus simples (deux entrées). C'est la raison pour laquelle il n'y a pas d'autres portes de Muller dissymétriques dans les bibliothèques.

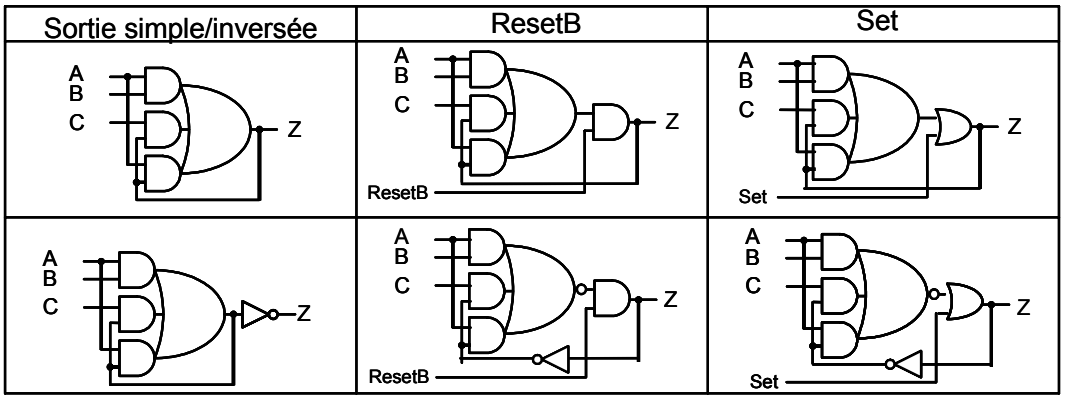


figure 131.Circuit des portes TAST_MULLER3D1P1N et ses dérivés.

Finalement, la bibliothèque structurelle de Muller comporte une trentaine d'éléments. Elle permet de réaliser la projection (« mapping ») des circuits asynchrones synthétisés par TAST sur des bibliothèques de cellules standard de fondeurs particuliers. Cette bibliothèque a spécialement été utilisée dans la réalisation de circuits ciblant la technologie HCMOS8D de ST Microelectronics. Mais celle-ci à été également utilisée pour les technologie 0.18 micron de TSMC. Le point limitant pour cette dernière technologie est que le nombre de cellules de base est plus petit. Toutes les cellules standard complexes possèdent des sorties inversées. Une étape de « Bubble Reshuffling » (injection d'inverseurs dans les portes) a donc été effectuée pour construire la bibliothèque correspondante.

7. Bibliothèque structurelle à base de LUTs

Pendant ce travail de thèse, l'étude sur le développement d'une bibliothèque de cellules spécifiques s'est orientée vers le prototypage de circuits asynchrones sur des composants programmables. Les travaux ont été fait sur les circuits asynchrones quasiment insensibles aux délais. L'idée est, tout en utilisant l'outil TAST, d'explorer les possibilités dune implémentations rapides sur FPGA de tels circuits.

Les précautions à prendre pour concevoir de circuits asynchrones sur FPGA sont les suivantes :

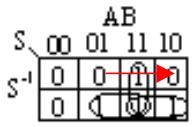
- la logique doit être sans aléa :
- les délais dans les fourches doivent être équilibrés.

Cela soulève deux problèmes que sont :

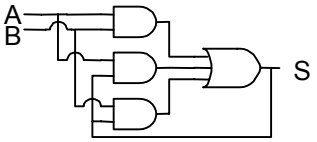
- le mapping des portes de Müller dans un FPGA. Suite à l'analyse des aléas dans une porte de Muller, comment réaliser ces portes pour qu'elles présentent un comportement sans aléa. Dans la plus part des architectures FPGA standard il n'est pas possible de relier la sortie à l'entrée à l'intérieur d'une même cellule logique programmable. Ce point implique que le « mapping » des portes de Muller doit être forcé dans un bloc logique configurable (CLB).
- le routage dans les FPGA. Les temps de propagation doivent être pris en compte lorsqu'on cible les FPGA. Ils ne sont pas négligeables car les connexions entre 2 nœuds traversent un ou plusieurs points programmables. Le routage des branches isochrones doit être équilibré pour éviter les hasards et minimiser les délais.

7.1. Analyse des aléas dans une porte de Muller

La porte Muller symétrique avec 2 entrées réalise le rendez-vous entre 2 signaux. Une implémentation sans aléa d'une porte de Muller à deux entrées est donnée dans la figure 132:



a. table de vérité de la porte MULLER2
transitions interdites : AB= 01→10 et 10→01



b. décomposition de la porte MULLER2 en réseau à deux niveaux AND, OR.

figure 132.Tableau de Karnaugh et circuit d'une porte de Muller à deux entrées.

La sortie de rétroaction de la porte Muller 2 est considérée comme une entrée. La minimisation des monômes canoniques dans les trois termes recouvrants AB , AS^{-1} , BS^{-1} rend impossible les aléas statiques dépendants de la distribution des délais quand une entrée change. Ainsi, cette implémentation est sans aléa quand une seule entrée change d'état. Le changement de plusieurs entrées est possible, excepté pour la transition AB = 01 → 10 et AB=10 → 01 qui produit un aléa lorsque la distribution des délais dans les signaux A et B n'est pas équilibrée (l'un change avant l'autre). La flèche dans le tableau de Karnaugh de figure 132, montre une transition pouvant provoquer un aléa.

TAST produit des circuits qui ne montrent pas ce type de comportement. Les transitions simultanées AB = 01 → 10 et AB=10 → 01 ne joue pas un rôle limitant dans l'analyse d'aléas.

En conclusion, la méthodologie TAST permet d'éviter l'apparition d'aléas dans les portes de Muller.

7.2. Implémentation sans aléa dans une CLB (bloc logique configurable)

7.2.1. Les cellules de bases : La LUT à portes de transmission

L'élément de base qui existe dans les cellules logiques élémentaires d'Altera (LE) et de Xilinx (CLB) est le générateur de fonctions à trois ou quatre entrées, ou autrement dit 3-LUT, et 4-LUT.

Un multiplexeur 8 → 1 peut se construire avec des portes de transmission comme nous pouvons le voir dans la figure 133.

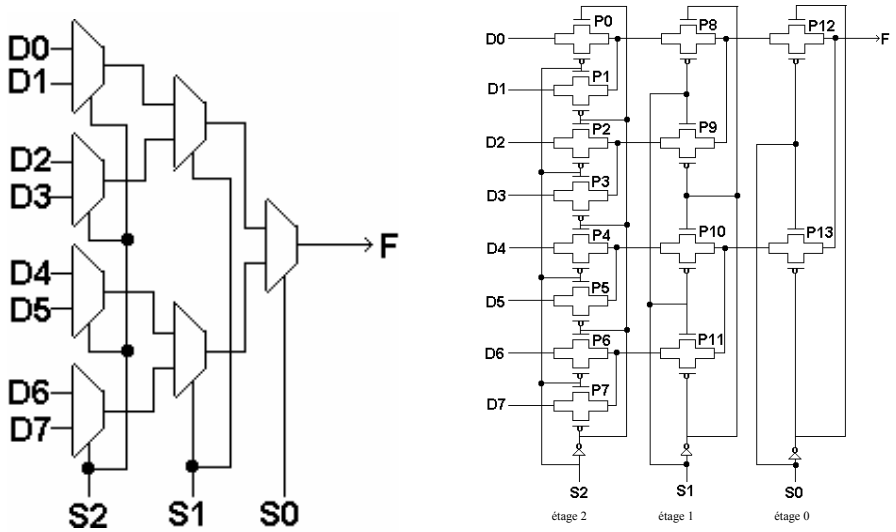


figure 133. Multiplexeur 8 entrées vers 1 sortie.

7.2.2. L'interconnexion des cellules de base

La figure 134 est une représentation simplifiée, de l'architecture hétérogène des composants d'Altera. Elle met en évidence le canal de routage en omettant les composants spécifiques conçus pour les circuits synchrones.

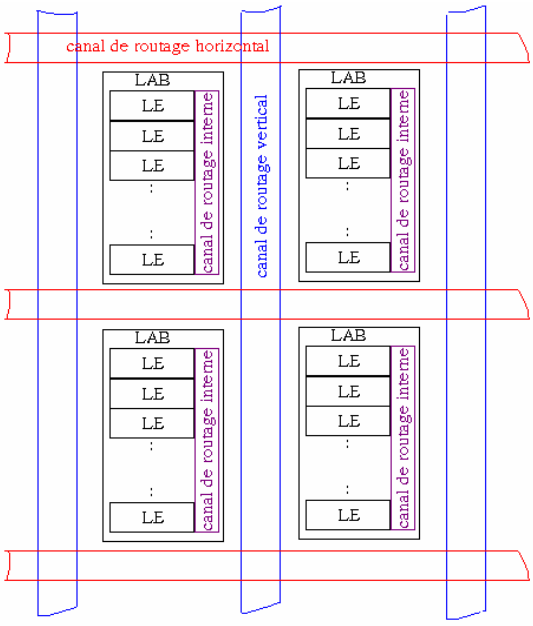


figure 134. Architecture de routage des composants d'Altera.

Le nombre de LEs (« Logic Element ») dans chaque LAB (« Logic Array Block ») varie selon les familles.

Dans les composants Altera, le canal de routage se compose de lignes directes entre les cellules logiques de base (LEs). Ainsi, le temps de propagation entre les LEs dans un même LAB est identique (tINT). De même, le temps de propagation entre les LABs d'une même

ligne de LABs est identique (tHOR), et le temps de propagation entre les LABs d'une même colonne de LABs est identique (tVER). Dans tous les cas, les interconnexions entre deux cellules logiques (LEs) ne traversent que deux points programmables.

7.2.3. Implémentation sans aléa de la porte Muller 2

Les LUT des composants Xilinx disposent de quatre entrées. Celles-ci limitent donc l'implémentation de portes de Muller d'au maximum un nombre identique d'entrées. Chez Altera ce nombre se limite à trois.

La porte MULLER2 peut être représentée à l'aide d'un LUT à quatre entrées dont le « mapping » des entrées/sorties est décrit sur la figure 135.

Soient F1, F2, F3 les entrées et F' la sortie du 4-LUT. La table de vérité de la porte MULLER2 à base du 4-LUT peut être représentée de la manière suivante (figure 135):

F1 S^{-1}	F2 A	F3 B	F4 x	F' S
0	0	0	x	0
0	0	1	x	0
0	1	0	x	0
0	1	1	x	1
1	0	0	x	0
1	0	1	x	1
1	1	0	x	1
1	1	1	x	1

figure 135. Table de vérité d'une Muller 2 à base de 4-LUT.

Les entrées A, B et la sortie de la porte sont mappées respectivement sur F2, F3, F' du 4-LUT. Le signal rebouclé S-1 est mappé sur F1. L'équation du générateur de fonctions F' est ainsi : F2F3 + F2F1 + F3F1. Les F1, F2, F3 sont les signaux de sélection du multiplexeur du 4-LUT. Pour chaque code de F1, F2, F3, la sortie est multiplexée à un bit de configuration. Ce bit de configuration est auparavant fixé par la mémoire de configuration du 4-LUT.

Dans le cas du changement d'un seul bit, les propriétés du générateur de fonction d'Altera assurent qu'elles ne déclenchent pas d'aléa quand une seule entrée de sélection change, même s il s'agit d'un décodage comportant des termes non recouvrants. Il n'y a, par conséquent, pas d'aléas quand une seule entrée change.

Considérons le cas de changements multiples. Il existe une différence d'implémentation logique utilisant une mémoire LUT comparée à un circuit composé de portes ET-OU standard. La mémoire LUT est constituée d'un multiplexeur et d'une configuration de bits correspondant au codage d'entrée. Les bits de configuration d'un générateur de fonction et les entrées d'une porte de Muller sont respectivement utilisés comme des informations d'entrées et des entrées de sélection du multiplexeur.

Par conséquent, les données d'entrées sont seulement affectées par des bits de configuration et ne changent pas dynamiquement. Les LUT de Xilinx et Altera ont été

soigneusement réalisées de telle manière que le délai de propagation de ses entrées vers la sortie soit équilibré. La sortie est retenue stable pendant le changement d'un codage des entrées d'une sélection à l'autre. Ainsi, si un codage intermédiaire qui cause un « glitch » à la sortie ne survient pas, il n'y a pas d'aléa. Cette supposition est assurée car les entrées de la porte Muller sont monotones.(Les entrées ont des transitions dans le même sens, c'est à dire qu'elle ont : soit toutes une transition'0'-> '1' soit toutes une transition '1'->'0').

Ce paragraphe expliquait l'implémentation sans aléa utilisant des générateurs de fonction où le comportement de la mémoire d'une porte Muller est contrôlé par le signal de rétroaction.

7.2.4. Analyse des aléas dans les portes à transmission du multiplexeur

Cette partie clarifie l'implémentation sans aléa à l'aide de LUT dont le multiplexeur est basé sur les portes de transmission (voir figure 136). La porte Muller 2 peut être implémentée en utilisant une LUT à 3 entrées.

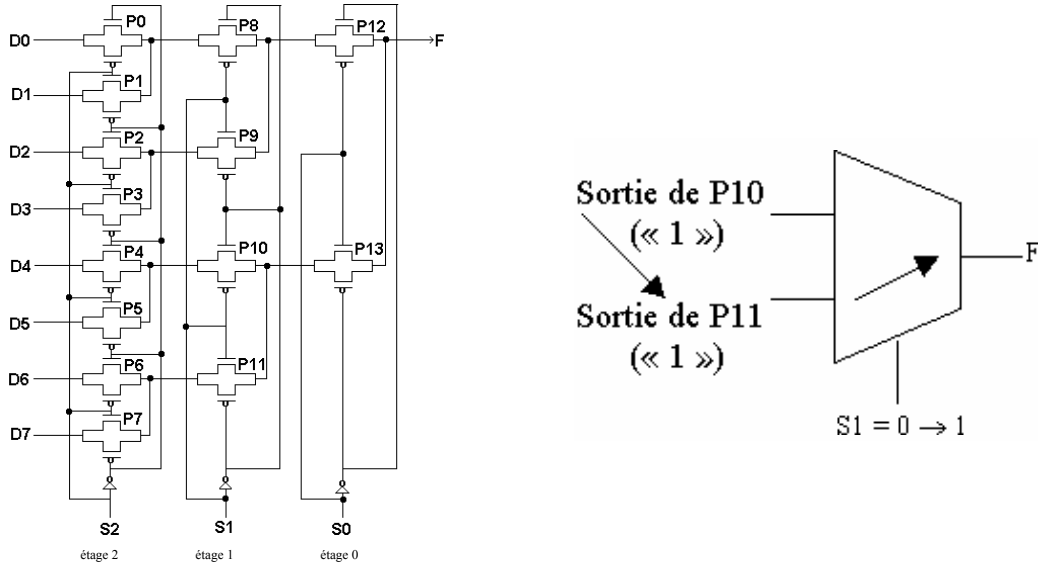
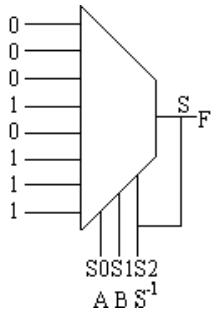


figure 136. Analyse d'aléas dans un multiplexeur à portes à transmission.

Supposons le changement d'un codage à un autre avec deux termes non–recouvrants, par exemple : S0S1S2 = 101 → 111.

Dans ce cas, seul S1 change de '0' à '1'. Ainsi, seul l'étage 1 change P8, P10 ON/ P9, P11 OFF → P8,P10 OFF/ P9,P11 ON. Puisque S0 est toujours à '1', P12 est OFF/ P13 est ON. C'est pour cela qu'aucun aléa n'apparaît. Dans le cas de multiples changements d'entrées, le changement de sélection d'une entrée à une autre peut provoquer l'apparition d'une ou de plusieurs sélections d'entrées intermédiaires. En appliquant l'analyse précédente pour un seul changement d'entrée, il ne peut pas y avoir de « glitch » en sortie entre deux valeurs appliquées en entrée si l'équation logique contient seulement des termes recouvrants.

En conclusion, la porte Muller, MULLER2 est sans aléa en utilisant les LUT du FPGA d'Altera et Xilinx (figure 137) si elle est insérée dans une cellule logique configurable et si son équation logique est optimisée pour le recouvrement de termes. Le comportement mémoire de la porte de Muller est contrôlé par le signal rebouclé. Finalement, une librairie de portes Muller intégrés dans les CLB a été réalisée.



S0	S1	S2	F
0	0	0	0 D0
0	0	1	0 D1
0	1	0	0 D2
0	1	1	1 D3
1	0	0	0 D4
1	0	1	1 D5
1	1	0	1 D6
1	1	1	1 D7

figure 137.Implémentation de la Muller 2 avec une 3-LUT.

7.2.5. Le placement des portes Muller dans les cellules logiques configurables

Dans cette section, les outils de synthèse et de placement routage d'Altera et de Xilinx sont considérés. Il est expliqué comment la bibliothèque a été réalisée afin de s'assurer du comportement sans aléa.

Les outils de synthèse commerciaux ne permettent pas d'automatiser la synthèse des portes de Muller. En effet, il n'est pas assuré que les équations logiques soient correctement synthétisées en terme d'aléas dans les cellules logiques configurables. La mémoire des portes de Muller est réalisée à l'aide d'un signal rebouclé. Pour des raisons de distribution de délais, il peut y avoir des aléas si on place et route automatiquement les portes de Muller dans un FPGA, même si son équation booléenne est correctement optimisée. Pour cela l'utilisation des outils de placement routage sont utilisés. La méthode est différente d'une famille de FPGA à l'autre et dépendent de ces outils.

Les outils de placement routage peuvent déplacer les termes redondants dans une équation logique durant la phase d'optimisation de l'espace disponible. Ces outils n'assurent pas que les équations des portes de Muller restent correctes. Ainsi, le flot de conception traditionnel des FPGA n'est pas utilisable. Il est, en premier lieu, nécessaire de veiller à la bonne conception d'une bibliothèque de portes de Muller exempte d'aléa. Cette approche résout deux questions : Premièrement, l'équation logique est sans aléa, deuxièmement, elle est intégrée dans un unique CLB, dont le délai d'interconnexion est connu et ne peut pas introduire d'aléas. Pour ce faire, chaque porte est modélisée en utilisant une netlist (EDIF, VHDL, AHDL, XNF) qui spécifie l'encapsulation à réaliser dans une CLB.

7.2.6. Implémentation dans un FPGA de Xilinx

Le flot de conception pour la cible FPGA de Xilinx est présenté figure 138 :

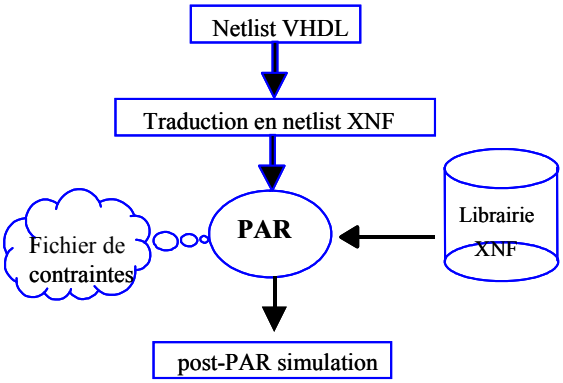


figure 138. Flot de conception Xilinx.

Les cellules de la bibliothèque sont d'écrites en format XNF. La description de chaque cellule contient les informations pour forcer le mapping sur FPGA.

La netlist VHDL du circuit est d'abord traduite au format propriétaire de Xilinx (XNF : Xilinx Netlist Format). Cette étape est obligatoire car l'outil de placement routage n'accepte que ce format. La description des contraintes du circuit et de placement routage se fait dans le fichier UCF (User Constraint File). Ainsi, ces contraintes sont appliquées sur les instances et les pistes d'interconnexions. Les cellules spécifiques sont considérées à cette étape du placement routage comme des « Macro-softs ». L'utilisation des informations de routage (ces « Macro-softs »), augmente la dynamique du routage et le taux d'occupation des cellules logiques de Xilinx (CLB). En effet, il permet de placer plus d'une cellule de librairie dans un seul CLB. La description en « Macro-hard » avec les informations de routage implique qu'une cellule de la bibliothèque dont le nombre des entrées est petit occupe tout le CLB.

Le fichier « User Constraint File » décrit les contraintes de placement routage pour garantir que les fourches soient bien isochrones. Ces contraintes sont : une relation spatiale entre les instances contenant les fourches isochrones. Un délai borné de l'interconnexion de la piste qui connecte les instances destinataires de la fourche.

L'outil de placement routage de Xilinx est capable d'augmenter le nombre d'itérations de placement et de routage nécessaires pour satisfaire ces contraintes. Si TAST fournit l'information des fourches isochrone, celles-ci sont placées dans le fichier de contraintes et sont utilisées pour forcer le mapping des cellules de base.

7.2.7. Implémentation dans un FPGA d'Altera

Le flot de conception pour la cible FPGA d'Altera peut être décrit de façon générale comme dans la figure 139.

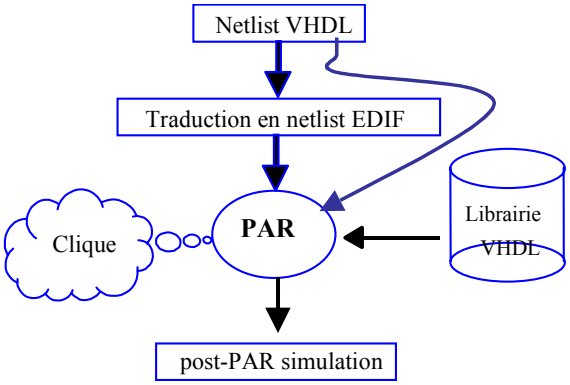


figure 139.Flot de conception d'Altera.

Les cellules de la librairie peuvent être décrites soit en schématique, soit en format particulier défini par Altera AHDL (Altera HDL), soit en VHDL. Le format en VHDL utilisant la librairie spécifique d'Altera est montré dans le flot de la figure 139. Les descriptions VHDL de chaque cellule contiennent des informations afin d'éviter explicitement la duplication logique par l'outil de PAR (Place And Route). Ces cellules sont pré compilées et mises dans une bibliothèque d'utilisateur (User-defined Library) des composants TAST. La librairie est prête pour implémenter la netlist dans un composant d'Altera par l'outil PAR d'Altera.

La netlist finale du circuit est au format EDIF. Il est également possible d'utiliser directement la netlist VHDL avec l'outil PAR (« Place and Route ») d'Altera.

Pour résoudre le problème des fourches isochrones, il suffit de mettre les instances de la fourche (expéditeur et destinataires) dans une « clique » (avec la souris dans l'outil). Défini par Altera, l'outil de PAR d'Altera est capable de mettre les LEs occupés par cette fourche dans un même LAB ou dans une même ligne selon la taille de cette fourche si ces LEs sont mis dans un « clique ». Grâce à l'architecture particulière de ces composants, le modèle de délai de propagation entre LEs d'un même LAB est identique.

Enfin, la simulation post-PAR permet de vérifier le fonctionnement du circuit implémenté.

7.2.8. Conclusion pour ce modèle de bibliothèque

La méthodologie TAST et les outils associés sont dédiés à la conception de circuits asynchrones. Cette structure générale de conception cible la simulation de modèles HDL comportementaux et la synthèse de circuits QDI ou de circuits micro pipelinés. La netlist des portes obtenue par synthèse est exploitée pour cibler des ASIC ou des prototypes à base de FPGA. L'insertion sur un FPGA standard est quasiment automatisée (excepté la génération d'un fichier de contraintes) et ne nécessite pas d'outils de placement routage spécifiques. Seule, la librairie (en format EDIF ,VHDL …) est nécessaire. Une fois que la position et

l'intégration d'une porte Muller sont physiquement planifiées pour satisfaire l'absence d'aléas et qu'elles minimisent l'utilisation des cellules de base dans le FPGA, le placement routage est réalisable automatiquement pour les circuits asynchrones. Cette approche est appropriée pour prototyper rapidement des circuits asynchrones de type QDI. Pour réaliser des circuits micro pipelines, il faut regarder comment implémenter les délais dans un FPGA La synthèse logique (à la main) des portes Muller requiert toutes les attentions mais l'utilisation de la librairie est automatique. Cette méthodologie a été appliquée avec succés pour la conception de systèmes incluant un ADC, un filtre FIR (en QDI) et un DAC dans un Altera de la famille APEX et Xilinx Virtex. Les résultats expérimentaux montrent que les circuits asynchrones testés sont pleinement fonctionnels. La méthodologie TAST développée au TIMA est bien engagée et permet le prototypage rapide et l'évaluation de circuits asynchrones et mixtes (circuits synchrone /asynchrone) sur un FPGA standard.([HO 02], [BAIX 02])

8. Conclusion

Trois bibliothèques de cellules asynchrones (fonctionnelle, structurelle à base de cellules standard et structurelle à base de LUT) ont été présentées dans ce chapitre pour lier la partie « front-end » et « back-end » du flot de conception de TAST. Il met en avant :

- une nomenclature unique.
- une bibliothèque générique indépendante de la technologie pour la description des circuits au niveau porte en sortie de l'outil TAST. Ce modèle fonctionnel permet une validation rapide des circuits synthétisés.
- deux bibliothèques structurelles spécifiques dépendantes de la technologie. Une à base de cellules standard permet de cibler les circuits de type ASIC, et l'autre à base de LUT permet de cibler les circuits de type FPGA.

A l'issue de ce développement, toutes ces bibliothèques ont été validées rapidement à tous les niveaux de la conception par la réalisation de plusieurs circuits chez différents fondeurs (ST Microelectronics, TSMC) et dans différentes technologies.

Le chapitre suivant à pour but de compléter les bibliothèques de cellules à travers l'étude de programmes indéterministes pour la conception de réseaux de communication dans les systèmes mono puce.

Chapitre VI

Synchronisation et arbitrage

Un des points critiques dans la conception d'un système mono puce (« System on-Chip ») est la réalisation du réseau de communication entre les différents modules qui composent le circuit (« Network On-Chip »). Ces réseaux de communication sur puce doivent être très modulables pour interfacer des blocs internes ou externes au composant complet, fournir une large bande passante, une latence faible, une consommation faible, des mécanismes d'arbitrages et de grandes flexibilités de routage. Dans un système mono puce, le bus de communication relie les composants entre eux et alloue dynamiquement un chemin d'un composant vers un autre. Plusieurs blocs fonctionnant en parallèle peuvent avoir besoin de la même ressource au même moment et provoquer une contention. Dans ce cas, un arbitre est nécessaire pour résoudre ce genre de conflits et s'assurer qu'un seul composant utilise cette ressource. Le choix du bloc qui accède à la ressource est fait par le biais d'une priorité affectée à chaque requête.

Le but de ce chapitre est de s'intéresser aux problèmes de synchronisations et d'arbitrages entre différents blocs synchrones ou asynchrones dans la perspective de rendre plus aisée et plus automatisée la conception de systèmes mono puce globalement asynchrones localement synchrones (« GALS ») [MOOR 02], [VILL 02] ou globalement asynchrones localement asynchrones (« GALA »). Dans un premier temps, nous présentons les différents blocs de bases que sont les synchroniseurs et les ME (blocs de mutuelle exclusion) indispensables à la réalisation d'arbitres. Ensuite, sont proposés différents algorithmes d'arbitrage équitables, à priorités fixes et dynamiques. Cette étude permet de compléter les bibliothèques de cellules asynchrones pour permettre la synthèse de systèmes non déterministes insensibles aux délais tels que les arbitres.

1. Spécification de l'indéterminisme en CHP

Le CHP possède une syntaxe et une sémantique pour modéliser l'indéterminisme dans un programme. Cela permet au concepteur de spécifier explicitement les problèmes d'indéterminisme pour ensuite appliquer un outil de synthèse qui dérive correctement le circuit correspondant.

Le CHP est basé sur les commandes gardées. Supposons que G soit une expression booléenne et S une liste de commandes, la commande gardées est alors : G => S. Elle signifie que S s'exécute quand l'expression G est vraie. La garde G peut être calculée par composition logique des canaux de communication, et/ou par la comparaison de variables et/ou de constantes, et/ou de « probe » (« # » teste l'activité sur un canal).

Le CHP propose deux structures: le choix et la répétition. Pour chacune d'entre elles le non déterminisme peut être spécifié. Au final les structures CHP possibles sont : le choix

déterministe (1) et la répétition déterministe (2), le choix non déterministe (3) et la répétition indéterministe (4). Leur syntaxe est la suivante :

(1) [G_1 => S_1 @ ... @ G_n => S_n]

(2) *[G_1 => S_1 @ ... @ G_n => S_n]

(3) [G_1 => S_1 @@ ... @@ G_n => S_n]

(4) *[G_1 => S_1 @@ ... @@ G_n => S_n]

Dans un choix déterministe (1), au moins une garde est vraie. L'exécution est arrêtée jusqu'à ce qu'une garde soit vraie. La commande S correspondant à la garde qui est vraie est exécutée et le choix se termine.

Dans une répétition déterministe (2) (note par « * »), au moins une garde est vraie. La répétition continue à exécuter les commandes associées à la garde qui est vraie. Quand aucune garde n'est vraie, la répétition se termine.

Dans le cas où l'exclusion mutuelle ou la stabilité des gardes ne peuvent être garanties [MART 93], le choix (3) et la répétition indéterministes (4) sont utilisés. Cela se traduit au niveau circuit par un problème d'arbitrage. Dans ce cas plusieurs gardes peuvent être vraies en même temps, mais une seule, arbitrairement choisie est sélectionnée et la commande correspondante est exécutée.

2. Blocs de base pour implémenter les problèmes de synchronisations

Le bloc de mutuelle exclusion et le synchroniseur sont les deux structures de bases indispensables pour résoudre les problèmes de synchronisation et d'arbitrage. Ce paragraphe à pour but de montrer comment ils sont modélisés en CHP et comment ils sont synthétisés.

2.1. Le bloc de mutuelle exclusion

Dans la méthode de synthèse de Martin [MART 93] du chapitre III, un ensemble de règles de production a été considéré. Dans ces règles de production, chaque garde était stable et non interférente. Mais, il est possible qu'il faille synthétiser des commandes gardées (choix ou répétitions) dans lesquelles les gardes ne sont pas mutuellement exclusives. C'est pourquoi, il est nécessaire d'avoir au moins un opérateur qui réalise un choix entre deux gardes qui sont potentiellement vraies au même instant. La fonctionnalité de ce bloc est de choisir parmi ses deux entrées celles qu'il faut considérer lorsque elles sont toutes les deux actives.

2.1.1. Spécification en CHP

La synthèse ici exposée fait référence à [MART 93]

La plus simple sélection entre deux gardes non exclusive est de la forme suivante:

$$
\begin{array}{l}
*[\quad [\quad x \rightarrow \cdots \\
\qquad\qquad @@\, y \rightarrow \cdots \\
\qquad\quad] \\
\quad]
\end{array}
$$

Dans lesquelles x et y sont de simples variables booléennes, et les deux gardes sont stables. Dans le but de distinguer les trois états de base du système (aucune garde, x ou y sont vrais), deux sorties appelées u et v sont introduites de la manière suivante:

$$
\begin{array}{l}
*[\quad [\quad x \rightarrow u+;\cdots \\
\qquad\qquad @@\, y \rightarrow v+;\cdots \\
\qquad\quad] \\
\quad]
\end{array}
$$

L'exécution du programme se déroule ainsi : à l'état initial, $\neg u \wedge \neg v$ est vrai codant l'état "aucune garde vraie". Ensuite, lorsque le choix est considéré comme terminé, les variables u et v retournent dans leur état initial.

Le programme devient alors :

$$
\begin{array}{l}
*[\quad [\quad x \rightarrow u+;[\neg x];u- \\
\qquad @@ \quad y \rightarrow v+;[\neg y];v- \\
\qquad\quad] \\
\quad].
\end{array}
$$

Si l'expression $\neg u \wedge \neg v$ est vraie à l'état initial du programme, $\neg u \vee \neg v$ est vrai tout le temps de son exécution.

Ce programme est la description de l'opérateur appelé par Martin arbitre de base ou bloc de mutuelle exclusion. Le choix entre les deux gardes n'est pas équitable.

2.1.2. Synthèse

Afin de respecter les conditions initiales sur les sorties de l'opérateur d'arbitrage ($\neg u \wedge \neg v$), on peut réécrire le programme proposé ci-dessus.

$$
\begin{array}{l}
*[\quad [\quad [x \wedge \neg v];u+;[\neg x];u- \\
\qquad @@ \quad [y \wedge \neg u];v+;[\neg y];v- \\
\qquad\quad] \\
]
\end{array}
$$

L'ensemble des règles de production, issue de la synthèse de cette expansion des communications, contient des règles non stables:

$$x \wedge \neg v \rightarrow u+$$
$$y \wedge \neg u \rightarrow v+$$
$$\neg x \vee v \rightarrow u-$$
$$\neg y \vee u \rightarrow v-$$

Les deux premières règles de production peuvent s'exécuter en même temps et rendre l'état du programme instable. L'état où $x \wedge y \wedge (u = v)$ de l'arbitre est appelé métastable. Une fois entré dans l'état métastable, avec $\neg u \wedge \neg v$, l'ensemble des règles de production qui spécifient le circuit peut produire un nombre indéfini de séquences :

$$*[(u+, v+); (u-, v-)]$$

Les nœuds u et v peuvent se stabiliser à une valeur logique au bout d'un temps qui est non borné. Eventuellement, une certaine dissymétrie due à une réalisation physique de la mutuelle exclusion (impureté, processus de fabrication, bruit thermique…) forcera le circuit dans un des deux états stables où u est différent de v. Mais cela ne se produira que dans un temps non borné. Cela à pour conséquence, qu'il n'est pas possible d'introduire un élément d'arbitrage dans un système avec horloge sans avoir la certitude de rencontrer une erreur de fonctionnement.

Les valeurs parasites de u et de v doivent être éliminées pendant l'état métastable puisqu'ils violent la condition $\neg u \vee \neg v$. Pour cela, un étage qui sert de filtre est ajouté. Il prend en entrée u et v et produit en sortie *uf* et *vf* (pour les sorties filtrées). L'effet de cet étage est le suivant:

$$uf, uv := (u \wedge \neg v), (v \wedge \neg u)$$

2.1.3. Schéma bloc et électrique du bloc de mutuelle exclusion

Voici le circuit de synthèse résultant proposé par Alain martin dans [MART 93]

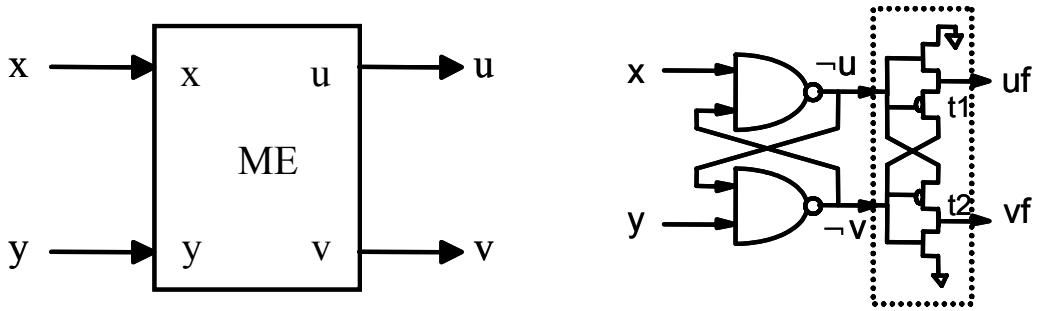


figure 140. La mutuelle exclusion.

Le réalisation de ce filtre en technologie CMOS utilise les tensions de seuil à son avantage: Le canal du transistor T1 ne conduit que si $(u \wedge \neg v)$ est vrai et le canal du transistor T2 ne conduit que si $(v \wedge \neg u)$. Lorsque le bloc de mutuelle exclusion est dans un état métastable, le filtre se comporte comme une mémoire et conserve sur ses sorties (uf et vf) des valeurs stables jusqu'à ce que le premier étage se stabilise.

2.2. Le synchroniseur

Le synchroniseur intervient dans les structures de choix qui implémentent des gardes instables. Par exemple, lorsque l'activité d'un canal intervient dans les différents choix d'un processus par l'intermédiaire de la commande probe en CHP (#).

2.2.1. Spécification en CHP

Le synchroniseur est ainsi défini en CHP:

$$
\begin{array}{lll}
*[& [& b \wedge z \rightarrow u+;[\neg b];u- \\
 & @@ & \neg b \wedge z \rightarrow v+;[\neg b];v- \\
 &] & \\
] & &
\end{array}
$$

La variable b (signal qui échantillonne le canal) peut changer de valeur à chaque instant de faux à vrai. Par conséquent, la garde $\neg b \wedge z$ est instable, alors que la garde $b \wedge z$ est stable et inversement jusqu'à ce qu'une des deux variables u ou v change de valeur.

2.2.2. Synthèse

La spécification du synchroniseur est la suivante:

$$
\begin{array}{lll}
*[& [& [b \wedge z \wedge \neg v];u+;[\neg b];u- \\
 & @@ & [\neg b \wedge z \wedge \neg u];v+;[\neg b];v- \\
 &] & \\
] & &
\end{array}
$$

De cette spécification l'extraction des règles de production faisantsuite à la synthèse donne:

$$
\begin{array}{c}
b \wedge z \wedge \neg v \rightarrow u+ \\
\neg b \wedge z \wedge \neg u \rightarrow v+ \\
\neg z \vee v \rightarrow u- \\
\neg z \vee u \rightarrow v-
\end{array}
$$

L'analyse sur l'état de métastabilité est similaire à celle du circuit de mutuelle exclusion. Elle est $(b \vee \neg b) \wedge z \wedge (u = v)$.

L'expansion est réécrite sans changer le fonctionnement en introduisant une variable x et une variable y pour respectivement la première et la seconde garde.

$$
\begin{array}{l}
*[\quad [\quad [b \wedge z]; x+; [x \wedge \neg v]; u+; [\neg b]; x-; [\neg x]; u- \\
\qquad @@ [\neg b \wedge z]; y+; [y \wedge \neg u]; v+; [\neg b]; y-; [\neg y]; v- \\
\qquad] \\
]
\end{array}
$$

A partir de cette expansion des communications on remarque que la réalisation du synchroniseur peut être considérée comme la juxtaposition d'un bloc de mutuelle exclusion avec un étage de logique booléenne.

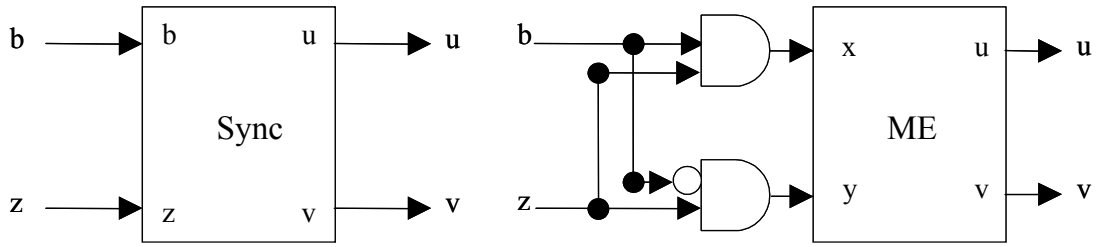


figure 141.Composition d'un synchroniseur à partir de la mutuelle exclusion.

2.3. Conclusion

Dans les systèmes insensibles aux délais, le fonctionnement correct de circuits contenant un élément de mutuelle exclusion ou un synchroniseur est indépendant de la durée de l'état métastable. C'est pourquoi une réalisation relativement simple de ces structures peut être utilisée. Cependant, dans le monde synchrone, l'implémentation de ces blocs doit prendre en compte la contrainte supplémentaire que potentiellement l'état métastable durera plus longtemps que la période d'horloge.

3. Modélisation de structures d'arbitrage quatre phases

A partir des deux éléments de base, que sont la mutuelle exclusion et le synchroniseur; ce paragraphe présente des structures simples utilisant ces deux circuits afin d'arbitrer deux canaux (utilisation de la ME) et/ou d'échantillonner l'activité d'un canal (utilisation du Synchroniseur). Le CHP est utilisé comme langage de spécification des circuits. Nous spécifierons le type d'accès entre les canaux (parallèle ou séquentiel). Le type de communication adopté dans les canaux est le protocole WCHB.

3.1. Structures de mutuelle exclusion quatre phases

Le premier arbitre quatre phases est spécifié par le programme CHP suivant:

$$\text{Arb } 0 \equiv *\left[\left[\begin{array}{l} \#A \rightarrow S!;A? \\ @@\#B \rightarrow S!;B? \end{array} \right] \right]$$

On réalise l'arbitrage entre deux canaux (A et B) en détectant d'abord une activité à l'aide de l'opérateur « probe ». Dans le cas où les deux canaux reçoivent en même temps une requête, un choix doit être fait pour connaître qui de A ou exclusivement de B sera ensuite acquitté. C'est ici le rôle du ME. Avant cet acquittement, une communication est réalisée sur le canal de sortie du circuit (S) . La spécification utilise un ";": il y a une séquence entre la communication sur le canal S et le canal A, avec à l'état initial A_{acq}, B_{acq} et S_{acq} à '1' et A_{req}, B_{req} et S_{req} à '0'.

Pour la synthèse de ce bloc, on déploie les communications du programme CHP.

$$*\left[\left[\begin{array}{l} \left[A_{req}\right];S_{req}+;\left[\neg S_{acq}\right];S_{req}-;\left[S_{acq}\right];A_{acq}-;\left[\neg A_{req}\right];A_{acq}+ \\ @@\left[B_{req}\right];S_{req}+;\left[\neg S_{acq}\right];S_{req}-;\left[S_{acq}\right];B_{acq}-;\left[\neg B_{req}\right];B_{acq}+ \end{array} \right] \right]$$

De là on introduit l' élément de mutuelle exclusion présenté au paragraphe 2.1.

$$*\left[\left[\begin{array}{l} \left[A_{req}\wedge\neg v\right];u+;[u];S1_{req}+;\left[\neg S1_{acq}\right];S1_{req}-;\left[S1_{acq}\right];A_{acq}-;\left[\neg A_{req}\right];u-;[\neg u];A_{acq}+ \\ @@\left[B_{req}\wedge\neg u\right];v+;[v];S2_{req}+;\left[\neg S2_{acq}\right];S2_{req}-;\left[S2_{acq}\right];B_{acq}-;\left[\neg B_{req}\right];v-;[\neg v];B_{acq}+ \end{array} \right] \right]$$

Pour obtenir une implémentation insensible aux délais il faut palier au problème de complétude du codage des états (« CSC »). Dans chacune des branches du choix non déterministe deux variables internes En1 et En2 sont ajoutées.

$$*\left[\left[\begin{array}{l} \left[A_{req}\wedge\neg v\right];u+;[u];S1_{req}+;\left[\neg S1_{acq}\right];En1+;[En1];S1_{req}-;\left[S1_{acq}\right];A_{acq}-;\left[\neg A_{req}\right];u-;[\neg u];En1-;[\neg En1];A_{acq}+ \\ @@\left[B_{req}\wedge\neg u\right];v+;[v];S2_{req}+;\left[\neg S2_{acq}\right];En2+;[En2];S2_{req}-;\left[S2_{acq}\right];B_{acq}-;\left[\neg B_{req}\right];v-;[\neg v];En2-;[\neg En2];B_{acq}+ \end{array} \right] \right]$$

Finalement, le programme à synthétiser est obtenu en enlevant la partie relative au bloc de mutuelle exclusion. A partir d'ici on peut considérer le choix comme exclusif puisqu'il se situe après l'exclusion mutuelle. Une seule des sorties du bloc est vraie en même temps.

$$
*\Big[\quad \Big[\quad [u];S1_{req}+;\big[\neg S1_{acq}\big];En1+;[En1];S1_{req}-;\big[S1_{acq}\big];A_{acq}-;[\neg u];En1-;[\neg En1];A_{acq}+
$$

$$
@[v];S2_{req}+;\big[\neg S2_{acq}\big];En2+;[En2];S2_{req}-;\big[S2_{acq}\big];B_{acq}-;[\neg v];En2-;[\neg En2];B_{acq}+
$$

$$
\Big]
$$

$$
\Big]
$$

Après synthèse, voici l'ensemble des règles de production du circuit final.

$$
\begin{array}{ll}
(\neg S1_{acq} \vee)\neg En1 \rightarrow A_{acq}+ & S1_{acq} \wedge En1 \rightarrow A_{acq}- \\
(u \wedge)\neg S1_{acq} \rightarrow En1+ & (S1_{acq} \wedge)\neg u \rightarrow En1- \\
u \wedge \neg En1 \rightarrow S1_{req}+ & (\neg u \vee)En1 \rightarrow S1_{req}- \\
(\neg S2_{acq} \vee)\neg En2 \rightarrow B_{acq}+ & S2_{acq} \wedge En2 \rightarrow B_{acq}- \\
(v \wedge)\neg S2_{acq} \rightarrow En2+ & (S2_{acq} \wedge)\neg v \rightarrow En2- \\
v \wedge \neg En2 \rightarrow S2_{req}+ & (\neg v \vee)En2 \rightarrow S2_{req}-
\end{array}
$$

Le circuit final est proposé figure 142:

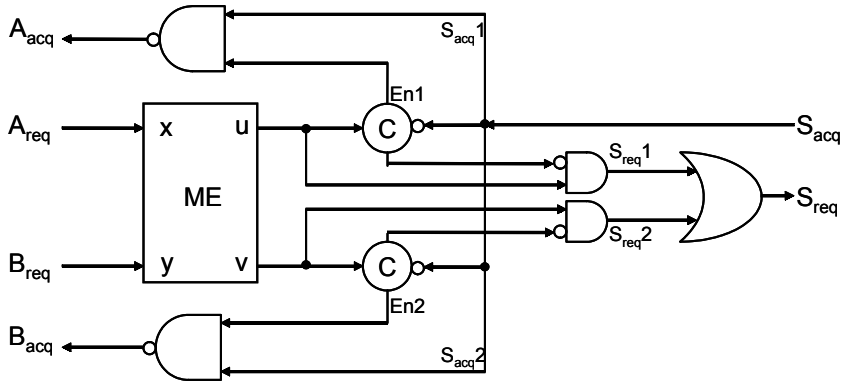


figure 142. Arbitre quatre phases avec séquentialité stricte.

Pendant la phase d'initialisation, le signal ResetB (actif bas) est appliqué sur les deux portes de Muller. L'environnement force A_{req} et B_{req} à '0' et S_{acq} à '1'. De son côté, le circuit force A_{acq}, B_{acq} à '1' et S_{req} à '0'. Après la mise en route du circuit, dès qu'une requête se présente sur un des deux canaux, ou sur les deux en même temps, une des sorties du bloc de mutuelle exclusion passe à '1' (supposons la sortie u). S_{req} envoie vers l'extérieur une requête. La réception d'un acquittement sur le canal S valide En1 et termine le protocole sur S. Dés que S_{acq} repasse à '1'. La communication sur le canal A est achevée en remettant En1 à '0': la porte Nand passe à '0' (En1=1 et S_{acq}=1) et envoie l'acquittement à la requête sur le canal A. Ensuite, la phase de remise à zéro sur le canal A s'effectue.

Ce premier arbitre quatre phases effectue strictement une communication sur S puis sur le canal d'entrée gagne la contention. Un autre circuit d'arbitrage a été proposé par David Dill et Ed Clarke [DILL 86] et amélioré plus tard par Mark Josephs [JOSE 96] pour réaliser des arbitres sous forme d'arbre. Il évite l'insertion d'une variable interne au niveau de l'expansion des communications. Pour cela, il utilise le signal d'acquittement du canal qui ne gagne pas la

contention. La spécification haut niveau en CHP ne change pas. On autorise ici le séquencement des phases de communication sur S et le canal qui gagne la contention : phases montantes de S puis de A et ensuite phases descendantes de S puis de A. L'expansion des communications avec insertion de l'élément de mutuelle exclusion devient :

$$
\begin{array}{l}
*\Big[\quad \Big[\quad \big[A_{req} \wedge \neg v\big]; u+; [u], \big[B_{acq}\big]; S1_{req}+; \big[\neg S1_{acq}\big]; A_{acq}-; \big[\neg A_{req}\big]; u-; [\neg u]; S1_{req}-; \big[S1_{acq}\big]; A_{acq}+ \\
\qquad @@\big[B_{req} \wedge \neg u\big]; v+; [v], \big[A_{acq}\big]; S2_{req}+; \big[\neg S2_{acq}\big]; B_{acq}-; \big[\neg B_{req}\big]; v-; [\neg v]; S2_{req}-; \big[S2_{acq}\big]; B_{acq}+ \\
\qquad\quad \Big] \\
\quad \Big]
\end{array}
$$

Le programme à compiler après l'extraction du bloc de mutuelle exclusion est :

$$
\begin{array}{l}
*\Big[\quad \Big[\quad [u], \big[B_{acq}\big]; S1_{req}+; \big[\neg S1_{acq}\big]; A_{acq}-; [\neg u]; S1_{req}-; \big[S1_{acq}\big]; A_{acq}+ \\
\qquad @\ [v], \big[A_{acq}\big]; S2_{req}+; \big[\neg S2_{acq}\big]; B_{acq}-; [\neg v]; S2_{req}-; \big[S2_{acq}\big]; B_{acq}+ \\
\qquad\quad \Big] \\
\quad \Big]
\end{array}
$$

Les règles de production après synthèse sont:

$$
\begin{array}{cc}
S1_{acq} \wedge \neg S1_{req} \rightarrow A_{acq}+ & \neg S1_{acq} \wedge S1_{req} \rightarrow A_{acq}- \\
u \wedge B_{acq} \rightarrow S1_{req}+ & \left(\neg B_{acq} \vee\right) \neg u \rightarrow S1_{req}- \\
S2_{acq} \wedge \neg S2_{req} \rightarrow B_{acq}+ & \neg S2_{acq} \wedge S2_{req} \rightarrow B_{acq}- \\
v \wedge A_{acq} \rightarrow S2_{req}+ & \left(\neg A_{acq} \vee\right) \neg v \rightarrow S2_{req}-
\end{array}
$$

Le circuit résultant est donné figure 143 :

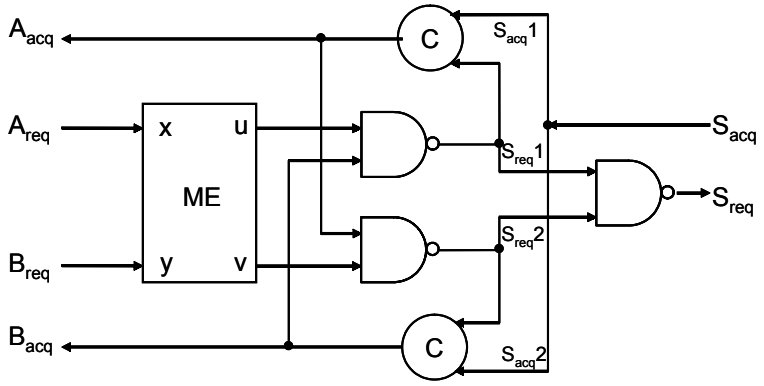


figure 143. Implémentation de D.Dill et E.Clarke.

Après réception d'une requête, celle-ci est propagée par l'intermédiaire de u ou v sur les canaux S et A ou S et B. Une fois les phases montantes du protocole accomplies, il y a synchronisation des phases de réinitialisation (retour à '0' du ME avec retour à '1' de l'acquittement de S).

Une dernière variante d'arbitre est proposée suivant le même modèle en relâchant encore les contraintes au niveau du programme CHP. Maintenant on introduit une concurrence totale entre la lecture sur les canaux d'entrée et l'écriture sur un canal S. La nouvelle spécification haut niveau est alors la suivante:

$$\text{Arb } 1 \equiv \begin{array}{l} *[\quad [\quad \#A \rightarrow S!, A? \\ \qquad @@ \quad \#B \rightarrow S!, B? \\ \qquad] \\ ] \end{array}$$

L'opérateur de concurrence (« , ») remplace l'opérateur de séquentialité (« , »). Le programme CHP est alors synthétisé

$$\begin{array}{l} *[\quad [\quad [A_{req} \wedge S_{acq}]; S_{req}+, A_{acq}-; [\neg S_{acq} \wedge \neg A_{req}]; S_{req}-, A_{acq}+ \\ \qquad @@ [B_{req} \wedge S_{acq}]; S_{req}+, B_{acq}-; [\neg S_{acq} \wedge \neg B_{req}]; S_{req}-, B_{acq}+ \\ \qquad] \\ ] \end{array}$$

Insertion du bloc de mutuelle exclusion pour résoudre les problèmes de métastabilité.

$$\begin{array}{l} *[\quad [\quad [A_{req} \wedge \neg v]; u+; [u \wedge S1_{req} \wedge B_{acq}]; S1_{req}+, A_{acq}-; [\neg A_{req}]; u-; [\neg u \wedge \neg S1_{acq}]; S1_{req}-, A_{acq}+ \\ \qquad @@ [B_{req} \wedge \neg u]; v+; [v \wedge S2_{req} \wedge A_{acq}]; S2_{req}+, B_{acq}-; [\neg B_{req}]; v-; [\neg v \wedge \neg S2_{acq}]; S2_{req}-, B_{acq}+ \\ \qquad] \\ ] \end{array}$$

Une fois l'opérateur extrait de ce HSE le programme à compiler est alors le suivant:

$$\begin{array}{l} *[\quad [\quad [u \wedge S1_{acq} \wedge B_{acq}]; S1_{req}+, A_{acq}-; [\neg u \wedge \neg S1_{acq}]; S1_{req}-, A_{acq}+ \\ \qquad @ [v \wedge S2_{acq} \wedge A_{acq}]; S2_{req}+, B_{acq}-; [\neg v \wedge \neg S2_{acq}]; S2_{req}-, B_{acq}+ \\ \qquad] \\ ] \end{array}$$

La synthèse donne cet ensemble de règles de production :

$$\begin{array}{cc} \neg u \wedge \neg S1_{acq} \rightarrow A_{acq}+ & u \wedge S1_{acq} \wedge B_{acq} \rightarrow A_{acq}- \\ u \wedge S1_{acq} \wedge B_{acq} \rightarrow S1_{req}+ & \neg u \wedge \neg S1_{acq} \rightarrow S1_{req}- \\ \neg v \wedge \neg S2_{acq} \rightarrow B_{acq}+ & v \wedge S2_{acq} \wedge A_{acq} \rightarrow B_{acq}- \\ v \wedge S2_{acq} \wedge A_{acq} \rightarrow S2_{req}+ & \neg v \wedge \neg S2_{acq} \rightarrow S2_{req}- \end{array}$$

Les règles de production conduisent au circuit figure 144 :

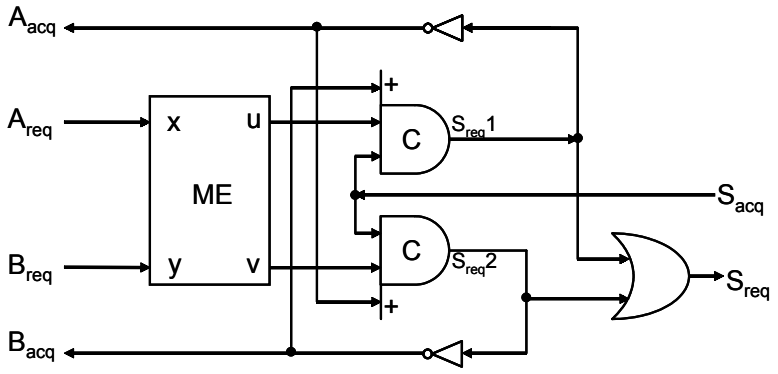


figure 144. Arbitre quatre phases tout parallèle.

Les deux portes de Muller dissymétriques ont besoin d'être initialisées à ‘0’ pour que le circuit fonctionne correctement et puisse forcer à ‘1’ les acquittements des canaux d'entrée et à ‘0’ la requête de sortie. Ensuite chacune des quatre phases du protocole s'exécute en concurrence sur les entrées et les sorties dès la réception d'une requête sur A ou sur B.

Ce paragraphe propose différentes structures d'arbitre quatre phases sur la base d’un bloc de mutuelle exclusion Il montre qu’un réordonnancement judicieux de phases de communication pendant la synthèse conduisent à des circuits avec des performances améliorées.

3.2. Structures de synchronisation quatre phases

Après avoir considéré l'exclusion mutuelle comme élément de base pour réaliser la synthèse de circuits nécessitant un arbitrage entre deux canaux, voyons que le synchroniseur s'utilise dans les cas ou un canal a besoin de tester l'activité d'un autre.

La spécification de ce synchroniseur quatre phases est par la suite utilisé dans l'exemple d'un arbitre équitable dont le programme à été proposé par Alain Martin [MART 93].

$$\text{Synchro } 0 \equiv \begin{array}{l} *[\\ \quad [\quad \#D \wedge \#B \rightarrow B?;D? \\ \quad\quad @@\#D \wedge \neg\#B \rightarrow D? \\ \quad] \\ ] \end{array}$$

Pour ce programme, le canal D est utilisé pour tester l’activité sur B. L'expansion des communications réalisant tout le protocole sur B puis le protocole sur D donne:

$$*\Big[\quad\Big[\quad\Big[D_{req} \wedge B_{req}\Big]; B_{acq}-; \Big[\neg B_{req}\Big]; B_{acq}+; D_{acq}-; \Big[\neg D_{req}\Big]; D_{acq}+$$
$$@@\ \Big[D_{req} \wedge \neg B_{req}\Big]; D_{acq}-; \Big[\neg D_{req}\Big]; D_{acq}+$$
$$\Big]$$
$$\Big]$$

Le signal B_{req} pouvant changer de vrai à faux de façon asynchrone de D, la deuxième garde de ce processus n'est pas stable, sa valeur peut changer à tout moment. Pour rendre les deux gardes de ce processus stables, le synchroniseur est introduit. Comme pour les exemples précédents, le but est de trouver un processus X tel que synchro 0 = X//sync. On introduit donc la spécification du synchroniseur dans le processus synchro 0.

$$*\Big[\Big[\Big[D_{req} \wedge B_{req}\Big]; u+; [u]; B_{acq}-; \Big[\neg B_{req}\Big]; B_{acq}+; D1_{acq}-; \Big[\neg D_{req}\Big]; u-; [\neg u]; D1_{acq}+$$
$$@@\ \Big[D_{req} \wedge \neg B_{req}\Big]; v+; [v]; D2_{acq}-; \Big[\neg D_{req}\Big]; v-; [\neg v]; D2_{acq}+\ \Big]\Big]$$

Finalement, le programme à synthétiser est le suivant :

$$*\Big[\quad\Big[\quad [u]; B_{acq}-; \Big[\neg B_{req}\Big]; B_{acq}+; D1_{acq}-; [\neg u]; D1_{acq}+$$
$$@\ [v]; D2_{acq}-; [\neg v]; D2_{acq}+$$
$$\Big]$$
$$\Big]$$

Pour faciliter la synthèse la transition B_{acq}+ est retardée après la garde $[\neg u]$. Cette modification n'entraîne pas de blocage de fonctionnement car la fin du protocole de D ne dépend pas de celle de B. On obtient le jeu de règle de production suivant:

$$\begin{array}{cc} u \rightarrow B_{acq}- & \neg u \rightarrow B_{acq}+ \\ u \wedge \neg B_{req} \rightarrow D1_{acq}- & \neg u \vee B_{req} \rightarrow D1_{acq}+ \\ v \rightarrow D2_{acq}- & \neg v \rightarrow D2_{acq}+ \end{array}$$

Les seules différences entre le circuit final proposé et celui de [MART 93] sont les conditions initiales sur les signaux des canaux en adéquation avec la convention choisie.

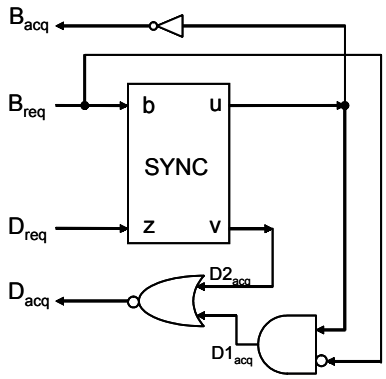


figure 145. Synchroniseur quatre phases d'Alain Martin.

Le fonctionnement du circuit figure 145 est le suivant: le canal D échantillonne le canal B. Si une requête est présente sur B au moment où D teste ce canal, la sortie u du synchroniseur passe à '1', B_{acq} passe à '0', mais $D1_{acq}$ reste nul. Ce n'est qu'au retour à '0' de B_{req}, que l'acquittement de D est envoyé à l'environnement. Le retour à l'état initial de D initialise de nouveau le circuit (D_{req} = 0 => u=0 => B_{acq}=D_{acq}=1).

Une extension immédiate de ce synchroniseur quatre phases est l'ajout d'un canal de sortie pouvant envoyer une donnée sur un canal S. Le programme CHP serait le suivant:

$$\text{Synchro 1} \equiv \begin{array}{l} *[\ [\ \#D \wedge \#B \rightarrow (B?;D?),S! \\ \quad @@\#D \wedge \neg\#B \rightarrow D? \\ \quad] \\ ] \end{array}$$

Après insertion du synchroniseur élémentaire, l'expansion des communications du programme à synthétiser devient:

$$\begin{array}{l} *[\ [\ [u \wedge S_{acq}];B_{acq}-,S_{req}+;[\neg B_{req}];D1_{acq}-;[\neg u \wedge \neg S_{acq}];B_{acq}+,D1_{acq}+,S_{req}- \\ \qquad @\ [v];D2_{acq}-;[\neg v];D2_{acq}+ \\ \quad] \\ ] \end{array}$$

Le processus Synchro1 possède juste une porte de Muller qui permet de synchroniser les deux échanges de protocoles entre les canaux B et S (figure 146).

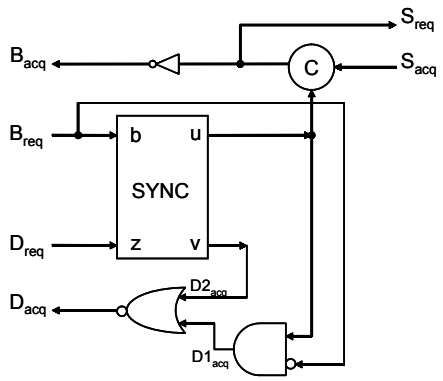


figure 146. Synchroniseur d'Alain Martin étendu avec un canal de sortie.

Ici une fois que le synchroniseur est activé, la porte de Muller synchronise la communication avec le canal S. Le reste du circuit fonctionne comme précédemment. On remarque qu'il faut initialiser cette porte à '0' à « reset ».

Les deux derniers programmes synchronisaient les phases montantes et descendantes du protocole de communication. Cette dernière version de synchroniseur réalise toutes les communications sur les canaux en parallèle. Le nouveau programme CHP, avec un canal de sortie est alors :

$$\text{Synchro } 2 \equiv \begin{array}{l} *[\quad [\quad \#D \wedge \#B \rightarrow B?, D?, S! \\ \qquad @@ \#D \wedge \neg\#B \rightarrow D? \\ \quad] \\ ] \end{array}$$

$$\begin{array}{l} *[\quad [\quad [D_{req} \wedge B_{req}]; u+; [u \wedge S_{acq}]; B_{acq}-, D1_{acq}-, S_{req}-; [\neg D_{req}]; u-; [\neg u \wedge \neg S_{acq}]; B_{acq}+, D1_{acq}+, S_{req}- \\ \qquad @@ [D_{req} \wedge \neg B_{req}]; v+; [v]; D2_{acq}-; [\neg D_{req}]; v-; [\neg v]; D2_{acq}+ \\ \quad] \\ ] \end{array}$$

Le programme à compiler est au final :

$$\begin{array}{l} *[\quad [\quad [u \wedge S_{acq}]; B_{acq}-, D1_{acq}-, S_{req}+; [\neg u \wedge \neg S_{acq}]; B_{acq}+, D1_{acq}+, S_{req}- \\ \qquad @ [v]; D2_{acq}-; [\neg v]; D2_{acq}+ \\ \quad] \\ ] \end{array}$$

Le jeu de règles de production issues de la synthèse est:

$$\begin{array}{ll} u \wedge S_{acq} \rightarrow B_{acq}- & \neg u \wedge \neg S_{acq} \rightarrow B_{acq}+ \\ u \wedge S_{acq} \rightarrow D1_{acq}- & \neg u \wedge \neg S_{acq} \rightarrow D1_{acq}+ \\ v \rightarrow D2_{acq}- & \neg v \rightarrow D2_{acq}+ \\ u \wedge S_{acq} \rightarrow S_{req}+ & \neg u \wedge \neg S_{acq} \rightarrow S_{req}- \end{array}$$

Le circuit se résume à un simple demi-buffer ajouté à la suite du synchroniseur. La poignée de main sur le canal échantillonneur est réalisé pour son acquittement par une porte non-ou. La figure 147 montre le schéma correspondant au synchroniseur quatre phases tout parallèle.

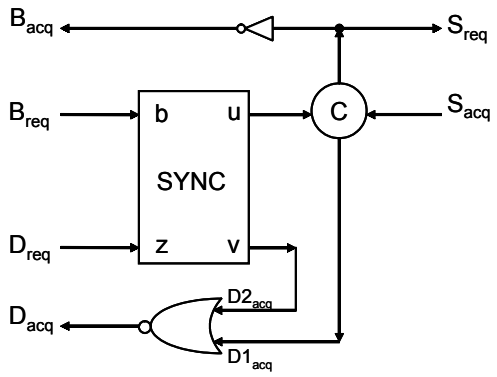


figure 147. Synchroniseur quatre phases « tout parallèle ».

La porte de Muller figure 147 doit être initialisée à ‘0’ pour que le circuit puisse démarrer.

4. Une structure très modulaire pour le partage des ressources

Le reste de ce chapitre a pour but d'introduire une méthode qui permet grâce au CHP de spécifier et de synthétiser tout circuit réalisant l'arbitrage entre différents blocs qui souhaitent accéder à une ressource commune. La décomposition en processus élémentaires des programmes CHP s’avère très puissante pour la synthèse d'arbitres.

Cette partie est aussi consacrée aux différents modèles d'arbitres proposés pour réaliser la synchronisation des blocs. Dans un premier temps, la synthèse d'arbitres équitables est abordée, puis celle des arbitres à priorités fixes enfin celle des arbitres à priorités dynamiques.

5. Les arbitres équitables

Cette première classe d'arbitres ne présente pas forcément un intérêt pour des cas pratiques de système mais elle représente une bonne application aux différents blocs présentés dans ce chapitre, paragraphe 3. En effet, en utilisant la méthode de décomposition en processus proposée par Alain Martin [MART 93], il est aisé d’entreprendre la réalisation de structures complexes par un simple assemblage de circuits élémentaires correspondant aux processus issus de la décomposition d'un programme CHP.

5.1. Définition

Un arbitre équitable est un circuit qui donnera accès à une ressource commune de façon équilibrée entre les différents canaux souhaitant accéder à cette ressource. Alain Martin a proposé une implémentation d'arbitre équitable [MART 93].

Un arbitre est dit fortement équitable lorsque la requête d'une communication en attente est acquittée après un nombre fini et borné de requêtes sur les autres demandeurs.

Un arbitre est dit faiblement équitable lorsque une requête est servie après un nombre fini mais non borné d'autres requêtes.

La suite de cette partie propose différentes solutions d'arbitres équitables en fonction de la décomposition que l'on aura choisie [RENA 00].

5.2. Synthèse d'arbitres équitables

Voici le programme CHP initial proposé par A.Martin accompagné du même programme auquel on a rajouté un canal de sortie correspondant à la ressource commune à laquelle il faut accéder.

$$
\begin{array}{l}
*[\ [\ \#A \rightarrow A? \\
\quad @@ \neg\#A \rightarrow Skip \\
\quad]; \\
\quad [\ \#B \rightarrow B? \\
\quad @@ \neg\#B \rightarrow Skip \\
\quad] \\
]
\end{array}
\qquad
\begin{array}{l}
*[\ [\ \#A \rightarrow A?, S! \\
\quad @@ \neg\#A \rightarrow Skip \\
\quad]; \\
\quad [\ \#B \rightarrow B, S! \\
\quad @@ \neg\#B \rightarrow Skip \\
\quad] \\
]
\end{array}
$$

L'activité de deux canaux A et B est testé par l'intermédiaire de la commande "Probe". Si une requête est détectée sur A ou B, on communique avec le canal de sortie S et la requête sur le canal demandeur est acquittée.

En appliquant les règles de décomposition des processus, le programme se partage en quatre processus qui communiquent entre eux par des canaux.

$$P1 \equiv *[C!; D!]$$

$$
\begin{array}{l}
P2 \equiv *[\ [\ \#C \wedge \#A \rightarrow (S1!, A?); C? \\
\qquad @@ \ \#C \wedge \neg\#A \rightarrow C? \\
\qquad] \\
]
\end{array}
$$

$$
\begin{array}{lll}
P3 \equiv *[& [& \#D \wedge \#B \rightarrow (S2!, B?); D? \\
 & @@ & \#D \wedge \neg\#B \rightarrow D? \\
 &] & \\
] & &
\end{array}
$$

$$
\begin{array}{lll}
P4 \equiv *[& [& \#S1 \rightarrow S!, S1? \\
 & @ & \#S2 \rightarrow S!, S2? \\
 &] & \\
] & &
\end{array}
$$

Le processus P1 est un Q-element (cf. 1.1) Il réalise une séquence forte entre les deux processus P2 et P3 qui sont des synchroniseurs quatre phases (3.2). Chacun de ces éléments sonde l'activité alternativement sur les canaux A et B respectivement par les canaux C et D. Pour chacun de ces synchroniseurs on affecte une sortie S1 (processus P2) et une sortie S2 (processus P3). Le dernier Processus P4 est implicite car il effectue simplement un ou de ces sorties intermédiaires pour les réunir sur le canal S. Une première version de ce circuit utilise le synchroniseur de A.Martin de la figure 148.

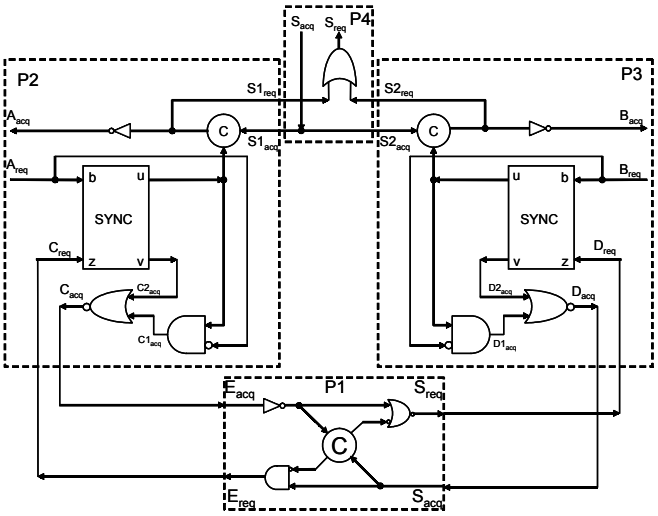


figure 148. Arbitre équitable proposé par A.Martin.

Après l'initialisation, le canal C reçoit une requête du Q-élément. La présence d'une donnée sur le canal A est testée via le premier synchroniseur quatre phases (processus P2). Ensuite, c'est au tour du canal B par l'intermédiaire du canal D. Ainsi de suite et de façon alternative sur P2 puis P3. Dès qu'une demande est faite, cela déclanche une communication sur S. Une fois celle-ci terminée, l'échantillonnage reprend sur les canaux. Toutes les portes de Muller doivent être initialisées à '0' au démarrage du circuit.

Un version optimisée de cet arbitre équitable est d'utiliser le deuxième synchroniseur quatre phases proposé (figure 147). Le programme CHP reste le même, la décomposition en processus est identique : les processus P1 et P4 restent inchangés mais P2 et P3 deviennent :

$$P2 \equiv *[\quad [\quad \#C \wedge \#A \rightarrow S1!, A?, C? \quad @@ \quad \#C \wedge \neg\#A \rightarrow C? \quad] \quad]$$

$$P3 \equiv *[\quad [\quad \#D \wedge \#B \rightarrow S2!, B?, D? \quad @@ \quad \#D \wedge \neg\#B \rightarrow D? \quad] \quad]$$

Le lecture des canaux C et D est parallélisée avec l'accès au canal S par l'opérateur « , ».

En reprenant les circuits équivalents à ces deux processus l'arbitre équitable correspond à la figure 149.

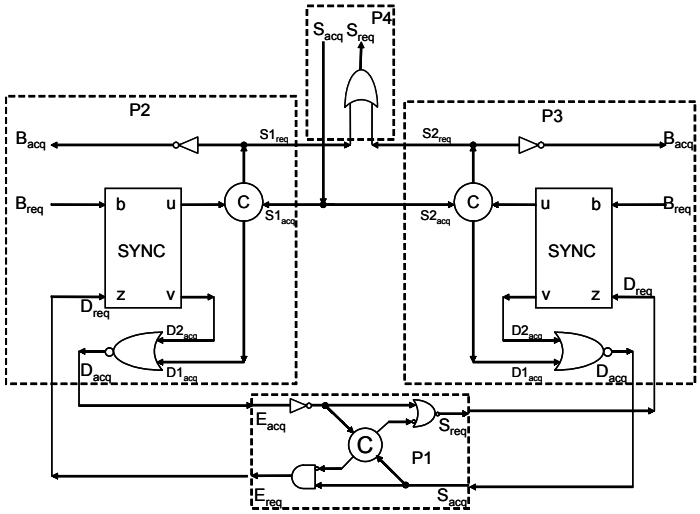


figure 149.Arbitre équitable quatre phases parallélisé.

Un des inconvénients de ces deux arbitres est qu'ils ne sont pas adaptés pour la basse consommation car ils sont en permanence en train de tester l'activité des deux ressources de l'arbitre.

5.3. Conclusion

Ce travail est inspiré de travaux proposés par A. Martin et J. Ebergen qui ont aussi été appliqués pour la réalisation de circuit d'arbitrage [MART 93] [EBER 95]. Cependant la synthèse automatique de tels programmes n'est pas encore implémentée dans l'outil TAST.

Dans le but d'enrichir l'ensemble des circuits synthétisables, cette première étude a exploré l'espace des arbitres basés sur la mutuelle exclusion ou sur le synchroniseur. Elle a

conduit à de nouvelles structures d'arbitres quatre phases. Elles sont maintenant utilisables pour simplifier la réalisation de circuits résolvant des problèmes complexes d'arbitrages par un jeu de décomposition de processus tel que l'ont proposé Martin et Ebergen. L'étape suivante est de poursuivre ce travail afin d'automatiser cette synthèse et de proposer au concepteur un langage de haut niveau sachant implémenter des problèmes complexes d'arbitrage ou des structures comprenant des choix déterministes ou non déterministes.

Le but final est de dispenser le concepteur de l'étape de décomposition des programmes en processus élémentaires. L'idée étant de savoir et de pouvoir isoler les choix indéterministes afin de les traiter dans une étape générale de compilation d'un programme CHP comme une structure de choix bien particulière.

6. Les Arbitres à priorité : présentation

La classification des systèmes d'arbitrages complexes repose sur la façon dont le service d'une ou des ressources communes est effectué. Si on tient compte de l'historique du traitement des demandes, alors l'arbitre exclut un comportement séquentiel tel qu'on le trouve dans les arbitres en anneau, arbres d'arbitres ou arbitres chaînés. On ne se base pas sur de telle considération pour classer les arbitres.

Si le calcul du demandeur à servir se fait en fonction de l'état courant de l'ensemble de ces demandeurs, l'arbitre est dit combinatoire. Cette fonction combinatoire des services à rendre peut être considérée comme une politique de priorité d'arbitrage. Ceci permet d'associer ces arbitres combinatoires à des arbitres dits à priorité. Ils sont alors classés ainsi:

- arbitrage à topologie fixe
- arbitrage statique avec une fonction de priorité arbitraire.
- arbitrage dynamique

6.1. Arbitrage à topologie fixe

Ce sont des arbitres qui ont une structure figée à la conception. L'ordre de test de l'ensemble des requêtes en attente d'un service est fixé par la structure même de l'arbitre. C'est le cas des arbitres en anneau, des arbres d'arbitres ou des arbitres chaînés.

6.2. Arbitrage à priorité fixe

Ici la structure de l'arbitre est unique : elle comprend une partie modifiable à volonté pour obtenir une fonction combinatoire de priorité. L'étage de résolution des contentions et le bloc qui implémente la fonction de priorité à résoudre sont séparés. La différence avec la classe précédente d'arbitre est que n'importe quelle fonction de priorité est réalisable. Elle ne dépend pas de l'ordre d'arrivée des requêtes en attente.

6.3. Arbitrage à priorité dynamique

Pour cette classe d'arbitres, la fonction de priorité prend en compte une information de priorité transportée par les demandeurs. Le rôle de la fonction de priorité est de détecter la requête avec le niveau de priorité le plus important et de la servir.

6.4. Conclusion

Les paragraphes 7 et 8, se propose de modéliser différents algorithmes de priorité au moyen du CHP et de voir quelles en sont les architectures obtenues après synthèse. Tous les arbitres proposés s'orientent sur le partage d'une ressource commune par quatre demandeurs potentiels.

Toutes les implémentations utilisent un protocole quatre phase initial (WCHB). On utilise des canaux simple rail provenant des demandeurs pour les arbitres à topologie ou priorité fixe. Par contre on utilise des canaux double rail pour les arbitres à priorité dynamique. Ceci permet de transporter en plus de la requête une information de priorité. Les canaux des blocs demandant l'accès à la ressource commune sont nommés R0, R1, R2, R3. Le signal de requête correspondant au canal R0 (en codage simple rail) est $R0_{req}$ et son signal d'acquittement est $R0_{acq}$. Pour un canal double rail on aura $R0_0$ et $R0_1$ pour la requête et la valeur transportées par le canal R0. Le signal d'acquittement portera le même nom que pour un canal simple rail.

On rappelle que dans les schémas, les portes marquées avec une lettre C sont des portes de Muller. Un R, respectivement un S, signifie que la sortie doit être initialisée à '0', respectivement à '1' pendant la phase de reset. Un plus sur les portes de Muller dissymétrique spécifie les entrées qui n'interviennent que pour faire passer la sortie de la porte à '1'.

7. Les Arbitres à priorité fixe

Ce paragraphe présente trois structures d'arbitres à priorité: un arbitre chaîné, un arbitre en arbre et un arbitre à échantillonnage parallèle. Cette dernière structure a été proposée dans [BYST 00]. Ici toutes les requêtes d'entrées ont un niveau de priorité qui est fixé matériellement. Une modification de l'ordre de priorité ou de la fonction de priorité entraîne la création d'un nouveau circuit. Pour nos exemple l'ordre de priorité est le suivant: R3>R2>R1>R0.

7.1. Arbitre chaîné à quatre entrées

7.1.1. Spécification en CHP

Voici figure 150 le programme CHP qui implémente cet arbitre à topologie fixe.

```
*[ #(R0∨R1 ∨R2 ∨R3)→[#R3 →S!,R3?
                    @@¬ #R3 →[#R2 →S!,R2?
                              @@¬ #R2 →[#R1 →S!,R1?
                                        @@¬ #R1 →S!,R0?
                                        ]
                              ]
                    ]
 ]
```

figure 150.Code CHP d'un arbitre chaîné à quatre entrées.

Le schéma de priorité ici est séquentiel. Le circuit est par défaut au repos et se réveille chaque fois qu'il détecte une activité sur au moins un des canaux. La fonction de sonde ou probe (le dièse en CHP) surveille l'activité sur tous les canaux (première ligne dans le programme CHP figure 150) et lance le traitement du processus. La priorité est simplement modélisée en testant séquentiellement les requêtes sur les canaux. La plus haute priorité est le canal R3. Il est testé en premier (#R3). S'il est actif, la ressource commune S est attribuée à R3 en écrivant sur S (commande CHP d'écriture : « S! »). Parallèlement (opérateur de composition parallèle « , »), R3 est servi en acquittant le canal (« R3?). Si R3 n'est pas actif (« # ¬R3 ») le canal suivant est analysé de la même façon que R3 et ainsi de suite pour les autres canaux.

La stabilité des gardes #R3 et #¬R3 n'est pas garantie. Le niveau du signal de requête de R3 peut changer au moment où il est évalué. Dans ce cas, un choix non déterministe est utilisé en CHP par le double @@. Au niveau matériel il correspond à l'utilisation d'un synchroniseur qui a pour rôle de résoudre le problème de métastabilité qui peut se produire pendant l'évaluation sur le canal R3.

7.1.2. Schéma de l'arbitre

Le circuit, figure 151, présente la représentation matérielle de l'arbitre chaîné.

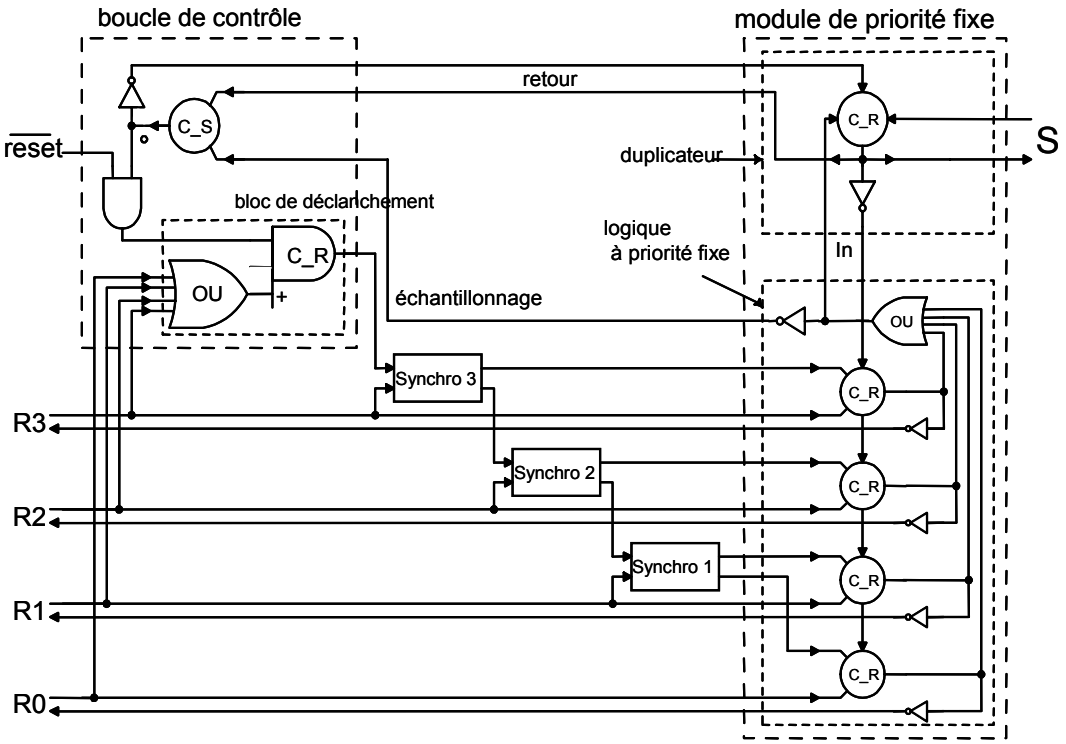


figure 151. Architecture de l'arbitre à priorité chaîné.

Cet arbitre est composé de trois blocs :

- La boucle de contrôle : elle doit réactiver l'arbitre après qu'une requête entrante ait été servie et que la ressource commune ait été accédée. Elle se compose du bloc de déclenchement qui détecte toute activité sur les canaux demandeurs et qui maintient l'arbitre au repos tant qu'il n'y a pas de nouvelles requêtes.
- Le deuxième bloc est la succession de trois synchroniseurs qui échantillonnent séquentiellement les requêtes entrantes.
- Le bloc de logique à priorité fixe : il détermine le canal d'entrée à servir en fonction de l'ordre de priorité choisi. On remarque que dans cette structure, on n'a pas besoin de synchroniseur pour R0. En effet, si ni R3, ni R2 ou R1 n'est actif ce sera forcement R0. Le duplicateur contrôle la ressource partagée S.

7.2. Arbitre en arbre

Le programme CHP, figure 152, montre clairement le schéma d'arbre du processus de sélection de cet arbitre quatre phases.

7.2.1. Spécifications en CHP

```
*[#(R0∨R1∨R2 ∨R3)?      [       [       #R3 ?   s1 := 2                          (1)
                                @@      ¬#R3 ?  [       #R2 ?   s1 := 1
                                                @@      ¬#R2 ?  s1 := 0
                                                ]
                                ],
                                [       #R1 ?   s0 := 2                          (2)
                                @@      ¬#R1 ?  [       #R0 ?   s0 := 1
                                                @@      ¬#R0 ?  s0 := 0
                                                ]
                                ]
                        ];
        [       s1 = 2 ?  R3? , S!                                               (3)
        @       s1 = 1 ?  R2? , S!
        @       s1 = 0 ?  [       s0 = 2 ?  R1 , S!
                          @       s0 = 1 ?  R0 , S!
                          ]
        ]
]
```

figure 152. Code CHP d'un arbitre en arbre à quatre entrées.

Ici encore, la première ligne du programme détecte l'activité sur les canaux demandeurs. Les deux premières parties du programme appelées (1) et (2) s'exécutent en parallèle et résolvent les problèmes de contention entre R3/R2 et R1/R0. La priorité de R3 sur R2 (respectivement de R1 sur R0) est modélisée par une structure linéaire en deux parties : on teste R3, respectivement R1, puis R2, respectivement R0. Ensuite l'opérateur de séquentialité du CHP (le point-virgule) est utilisé pour faire un rendez-vous entre les deux parties qui s'exécutaient en parallèle. Une fois ces deux blocs terminés, on exécute le deuxième étage de l'arbitre (3). Ici on rencontre un choix déterministe car les deux variables internes s1 et s0 sont stables. En tenant compte de la valeur de ces deux variables, le vainqueur est servi et accède à la ressource commune.

7.2.2. Schéma de l'arbitre

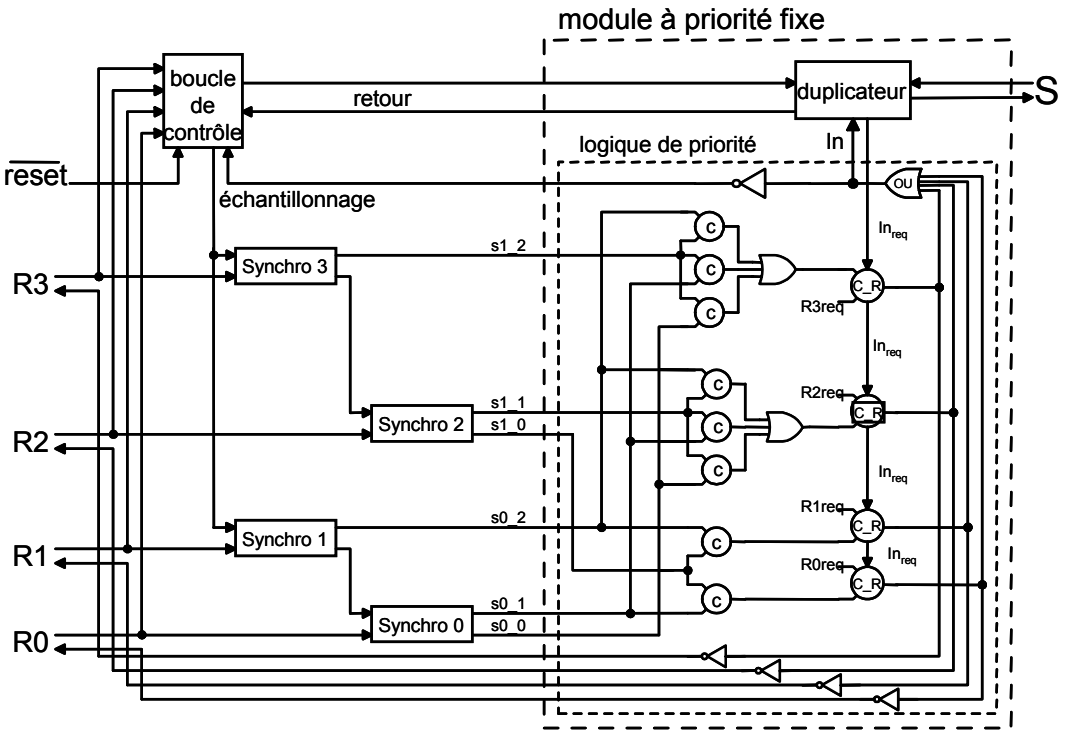


figure 153. Structure de l'arbitre à priorité en arbre.

Le circuit figure 153 comprend deux paires de synchroniseurs en cascade qui analysent les requêtes d'entrée en tenant compte de l'ordre de priorité choisi (R3 sur R2 et R1 sur R0). Ensuite le bloc de logique insensible aux délais implémente une comparaison déterministe dans le but de choisir l'entrée qui doit être servie. La boucle de contrôle et le duplicateur sont identiques et jouent le même rôle que pour le modèle d'arbitre précédent.

7.3. Arbitre à priorité fixe avec échantillonnage parallèle

Cet arbitre a récemment été proposé par Bystrov, Kinniment et Yakovlev dans [BYST 00]. Ils ont introduit une version d'arbitres à priorité qui découple l'échantillonnage des signaux de requête (module de synchroniseurs) et la résolution des contentions entre les demandeurs (module de priorité). Cette structure dépasse les architectures précédemment proposées en terme de complexité et de latence. En plus, elle est fortement modulaire : la fonction de priorité peut être facilement modifiée en ne changeant que le module de priorité

7.3.1. Spécifications en CHP

```
*[#(R0∨R1∨R2 ∨R3)?      [       [       #R3 ?   s3 := 1    -- synchroniseur 3
                                @@     ¬#R3 ?   s3 := 0
                                ],
                                [       #R2 ?   s2 := 1    -- synchroniseur 2
                                @@     ¬#R2 ?   s2 := 0
                                ],
                                [       #R1 ?   s1 := 1    -- synchroniseur 1
                                @@     ¬#R1 ?   s1 := 0
                                ],
                                [       #R0 ?   s0 := 1    -- synchroniseur 0
                                @@     ¬#R0 ?   s0 := 0
                                ]
                        ];                                 --module de priorité
        [       s3 = 1 ?  R3? , S!
        @       s3 = 0 ?  [       s2 = 1 ?  R2 , S!
                          @       s2 = 0 ?  [       s1 = 1 ?  R1 , S!
                                            @       s1 = 0 ?  R0 , S!
                                            ]
                          ]
        ]
]
```

figure 154.Code CHP d'un arbitre à échantillonnage parallèle.

La première ligne du code a toujours la charge de détecter l'activité sur les canaux entrants. Ensuite, quatre sous parties identiques modélisent l'échantillonnage parallèle des signaux de requêtes (commenté dans le code CHP synchroniseur 3 à 0). Les variables s3 à s0 sont utilisées pour mémoriser les échantillons (activité du canal). Le codage de la donnée est du double rails. Les échantillons sont ensuite exploités dans le module de priorité pour déterminer quel sera le canal à servir.

7.3.2. Schéma bloc de l'arbitre à priorité à échantillonnage parallèle.

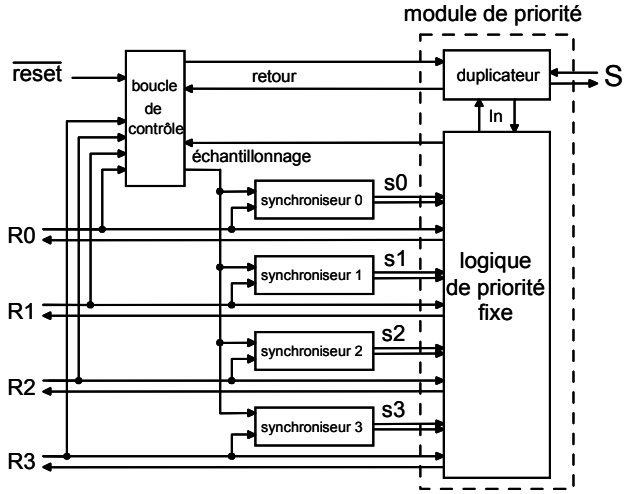


figure 155.Schéma bloc de l'arbitre à priorité quatre voies avec échantillonnage parallèle.

Les quatre synchroniseurs fonctionnent de façon concurrente pour échantillonner les signaux de requête. Le module de priorité détermine le demandeur à servir en accord avec la fonction de priorité codée dans ce bloc. En conservant le même ordre de priorité que les exemples précédents, voici, figure 156, un circuit QDI du bloc de priorité.

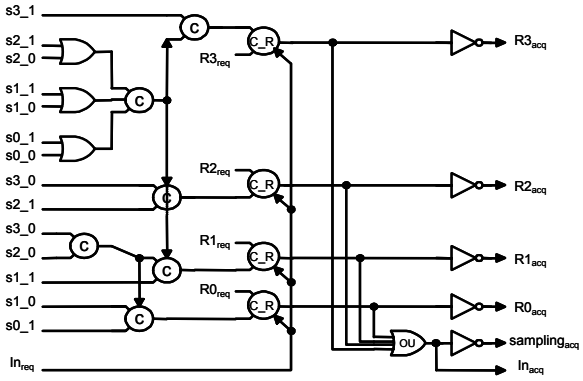


figure 156.Logique de priorité de l'arbitre à échantillonnage parallèle.

8. Les Arbitres à priorité dynamique

Dans les arbitres à priorité dynamique, les canaux d'entrée transportent en plus de la requête une information de priorité. Ici le schéma de fonctionnement est différent des autres. L'arbitre à priorité fixe effectue les comparaisons des requêtes actives et sert celle qui à le niveau de priorité le plus haut. L'entrée qui aura l'index le plus haut est, par convention, sélectionnée en cas de niveaux de priorité identiques. On présente ici la même structure que l'arbitre avec échantillonnage parallèle mais ici l'algorithme de résolution mis en œuvre possède une priorité dynamique.

8.1. Spécifications CHP de l'arbitre

La figure 157 présente le code CHP qui modélise un arbitre quatre voix à priorité dynamique qui fonctionne suivant le principe de l'arbitre à échantillonnage parallèle.

Chaque entrée Ri est maintenant codée grâce à un canal double rails : deux niveaux de priorité sont seulement considérés pour des raisons de simplicités mais il n'y a pas de limitations à ce niveau la dans la modélisation du programme. Ce canal double rails code à la fois la requête et la valeur de priorité : zéro (« 01 » en double rails) qui est plus faible qu'une priorité de un (« 10 » en double rails).

```
* [ #( R0 ∨ R1 ∨ R2 ∨ R3 )                          -- boucle de contrôle
→ [  [      #R3 → s3 := 1 , [  #R3 = 0 → r3 := 0  -- synchroniseur 3
                              @ #R3 = 1 → r3 := 1 ]
     @@ ]#R3 → s3 := 0
     ] ,
     [      #R2 → s2 := 1 , [  #R2 = 0 → r2 := 0 -- synchroniseur 2
                              @ #R2 = 1 → r2 := 1 ]
     @@ ]#R2 → s2 := 0
     ] ,
     [      #R1 → s1 := 1 , [  #R1 = 0 → r1 := 0  -- synchroniseur 1
                              @ #R1 = 1 → r1 := 1 ]
     @@ ]#R1 → s1 := 0
     ] ,
     [      #R0 → s0 := 1 , [  #R0 = 0 → r0 := 0   -- synchroniseur 0
                              @ #R0 = 1 → r0 := 1 ]
     @@ ]#R0 → s0 := 0
   ] ] ;
-- module de priorité dynamique
-- 1st étage: analyseur de requêtes deux voies
-- et comparaison de priorité
   [ [   (s0, s1) = (0, 0) → v1_0 := 0
     @ (s0, s1) = (1, 0) → v1_0 := 1
     @ (s0, s1) = (0, 1) → v1_0 := 2
     @ (s0, s1) = (1, 1) → [   r1 ≥ r0 → v1_0 := 2
                             @ r1 < r0 → v1_0 := 1 ]
     ] ,
     [   (s2, s3) = (0, 0) → v3_2 := 0
     @ (s2, s3) = (1, 0) → v3_2 := 1
     @ (s2, s3) = (0, 1) → v3_2 := 2
     @ (s2, s3) = (1, 1) → [   r3 ≥ r2 → v3_2 := 2
                             @ r3 < r2 → v3_2 := 1 ]
     ] ] ;
-- 1er étage d'acquittement
   [  (v1_0, v3_2) = (1, 0) → comp := 0 , R0? , S!
   @ (v1_0, v3_2) = (2, 0) → comp := 0 , R1? , S!
   @ (v1_0, v3_2) = (0, 1) → comp := 0 , R2? , S!
   @ (v1_0, v3_2) = (0, 2) → comp := 0 , R3? , S!
-- multiplexeur
   @ (v1_0, v3_2) = (1, 1) → comp := 1 , C0 := r0, C1 := r2
   @ (v1_0, v3_2) = (1, 2) → comp := 1 , C0 := r0, C1 := r3
   @ (v1_0, v3_2) = (2, 1) → comp := 1 , C0 := r1, C1 := r2
   @ (v1_0, v3_2) = (2, 2) → comp := 1 , C0 := r1, C1 := r3
   ] ;
-- 2ieme étage: comparateur de priorité deux voies
   [  comp = 1 → [   C1 ≥ C0 → out := 1
                   @ C1 < C0 → out := 0 ]
   @ comp = 0 → skip
   ] ;
-- 2ième étage d'acquittement
   [  comp = 1  → [   out = 1 → [  (v1_0, v3_2) = (1, 1) → R2? , S!
                                 @ (v1_0, v3_2) = (1, 2) → R3? , S!
                                 @ (v1_0, v3_2) = (2, 1) → R2? , S!
                                 @ (v1_0, v3_2) = (2, 2) → R3? , S!
]
                   @ out = 0 → [  (v1_0, v3_2) = (1, 1) → R0? , S!
                                 @ (v1_0, v3_2) = (1, 2) → R0? , S!
                                 @ (v1_0, v3_2) = (2, 1) → R1? , S!
                                 @ (v1_0, v3_2) = (2, 2) → R1? , S!
]
   @ comp = 0 → skip
 ] ] ;
```

figure 157. Spécifications CHP d'un arbitre à priorité dynamique quatre voies.

La première partie est très similaire à celle de l'arbitre à priorité fixe précédent. Elle correspond aux étiquettes synchroniseurs 3 à 0 dans le programme ci-dessus. Par contre, le module de priorité est beaucoup plus compliqué parce qu'il doit effectuer la comparaison des priorités des requêtes qui sont en compétition. Le programme possède une structure de comparaison des priorités à deux étages. D'autres structures pourraient être envisagées entraînant différents compromis entre complexité, vitesse et consommation.

Le premier étage du module de priorité analyse de façon concurrente, les entrées par paires et si nécessaires effectue la comparaison des priorités. La variable v1_0 signifie : pas de requête (valeur 0) ou une requête sur R0 (valeur 1) ou une sur R1 (valeur 2). La requête avec la plus forte priorité est envoyée à l'étage suivant. La variable v3_2 joue le même rôle pour les canaux R2 et R3. Ces deux variables ne peuvent pas être ensemble à zéro car les comparateurs de priorités sont déclenchés sur la détection d'au moins une requête. Si une seule requête franchit le premier étage, le canal correspondant est immédiatement servi et peut accéder à la ressource commune. Si plusieurs requêtes se présentent, elles sont traitées par le deuxième étage de comparaison. Un multiplexeur identifie les requêtes qui sont en comparaison et envoie alors leur propre valeur de priorité dans le deuxième comparateur de priorités. Finalement le résultat de cette comparaison avec le couple d'entrées en contention permet d'identifier le vainqueur (deuxième étage d'acquittement).

8.2. Structure de l'arbitre à priorité dynamique

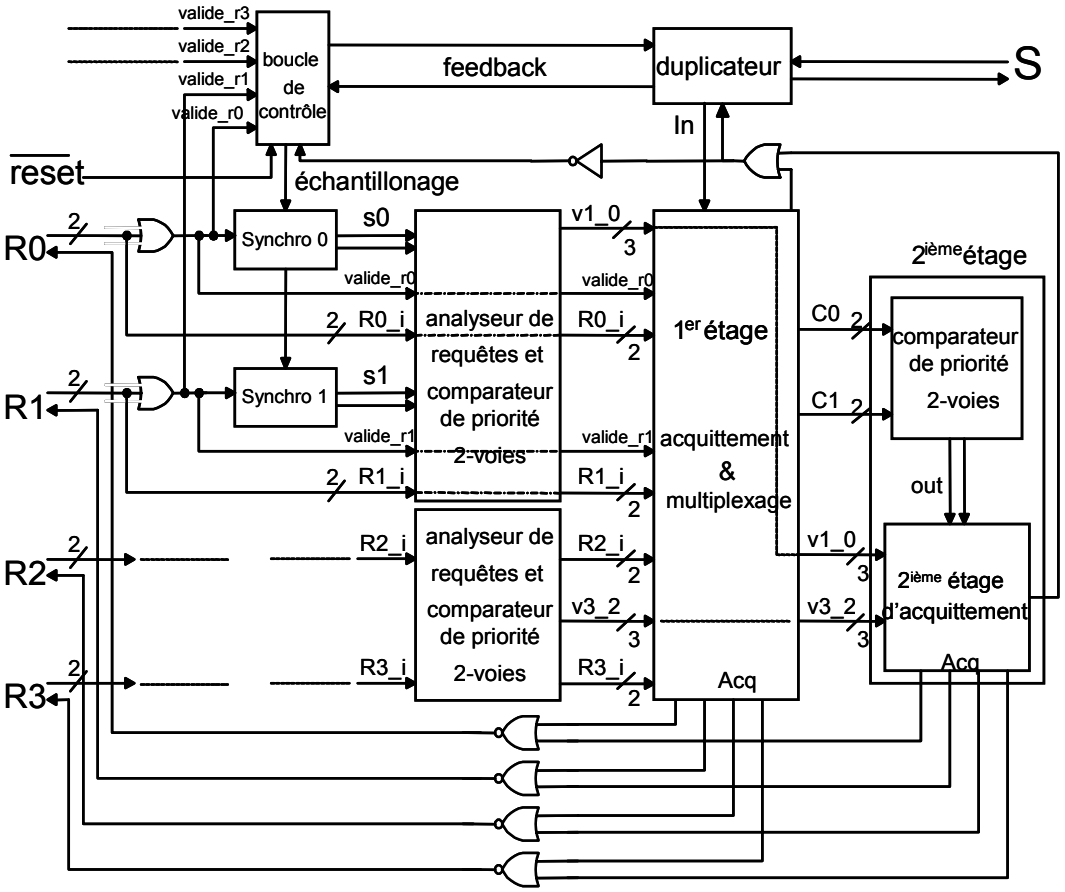


figure 158. Schéma bloc de l'arbitre à priorité dynamique.

Une description plus détaillée des blocs du circuit figure 158 est maintenant abordée.

8.2.1. L'analyseur de requêtes et le comparateur de priorités à deux voies

Ils sont au nombre de deux et chacun compare une paire de requêtes. Le circuit synthétisé est figure 159:

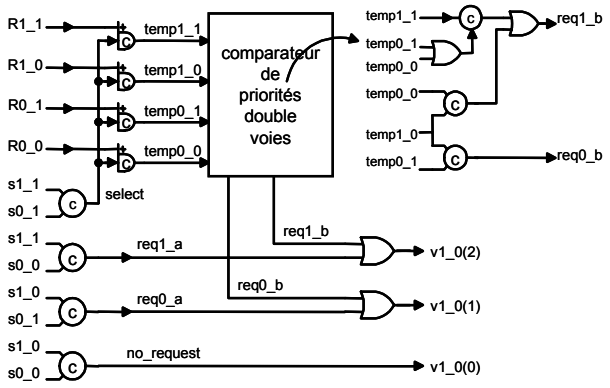


figure 159. Analyseur de priorités et comparateur de priorité.

Ce circuit a pour fonction de libérer le demandeur de priorité le plus fort. On remarque que le comparateur de priorité est activé seulement si il y a deux requêtes présentes dans le bloc (Muller (s1_1, s0_1).

8.2.2. Premier étage d'acquittement et multiplexeurs

Ce bloc détecte si une seule requête est active à la sortie des deux blocs précédents. Dans ce cas favorable le demandeur est immédiatement acquitté et peut accéder en parallèle à la ressource commune. Voici, figure 160, une représentation du circuit :

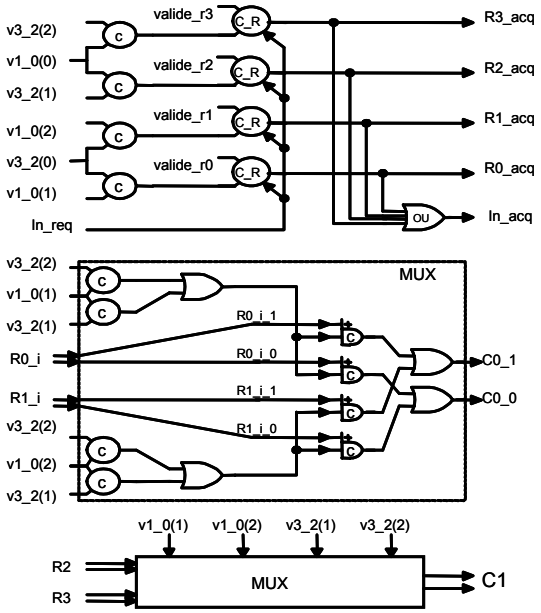


figure 160. Multiplexeurs et premier étage d'acquittement.

Dans le cas où plusieurs requêtes sont en contention après ce premier étage, deux multiplexeurs propagent le niveau de priorité au comparateur de priorités final.

8.2.3. Deuxième étage d'acquittement

La fin de cette comparaison est suivie d'un deuxième étage d'acquittement pour servir le canal vainqueur. Le circuit est le suivant :

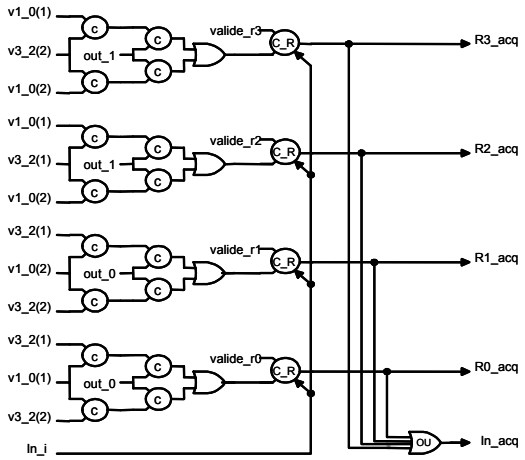


figure 161. Schéma du deuxième étage d'acquittement.

9. Conclusion

La synthèse d'arbitres insensibles aux délais pour la conception de réseaux de communication sur puce (« Network on-Chip ») est abordée dans ce chapitre. Tous les circuits sont spécifiés dans un langage de haut niveau le CHP.

Dans un premier temps, une étude sur les deux différents blocs élémentaires que sont le bloc de mutuelle exclusion et le synchroniseur a été réalisée. Ils sont les éléments qui permettent de résoudre les problèmes de métastabilité. Le synchroniseur et le bloc de mutuelle exclusion ont par conséquent été ajoutés aux bibliothèques de cellules asynchrones.

De nouvelles structures d'arbitres et de synchroniseurs quatre phases sont proposées. Elles montrent, qu'en utilisant la méthode de décompositions des processus CHP, proposée par Alain Martin, la conception d'arbitres asynchrones est très rigoureuse et offre des structures très modulaires.

Par la suite, différents algorithmes d'arbitrages à priorité fixe et dynamique ont été spécifiés en CHP, puis synthétisés. Ces arbitres sont fiables à 100% car ils laissent assez de temps au circuit pour résoudre les problèmes de métastabilité. Sur le point de la consommation, ces circuits ont une activité électrique minimale : contrairement aux circuits synchrones, la consommation électrique des circuits asynchrones insensibles aux délais est proportionnelle à son activité [PIGU 01].

Les systèmes de communication asynchrones proposés sont des blocs totalement autonomes, redimensionnables, qui peuvent être facilement réutilisés dans les architectures des systèmes sur puce complexes et par conséquent diminuer la complexité et le temps de conception.

Ce chapitre montre qu'aujourd'hui, il est possible de mettre en forme et de concevoir de façon rigoureuse des modules d'arbitrages insensibles aux délais, fiables, modulaires et rapides ([RIGA 02]). Il définit aussi les bases d'une automatisation du processus de synthèse pour les problèmes d'arbitrage.

Conclusion

Ce travail de thèse s'insère dans le cadre du développement d'un environnement logiciel pour la conception de circuits asynchrones. La principale raison qui justifie ce projet est un manque général d'outils permettant de mettre en œuvre de tels systèmes pour favoriser l'acceptation et la diffusion dans le monde industriel des concepts de la logique asynchrone comme une alternative aux limitations qui découlent de l'évolution rapide des nouvelles technologies.

L'objectif est le développement d'une couche « back-end » faisant le lien entre l'outil de conception et de synthèse de circuits asynchrones TAST et les outils standard (des circuits synchrones) de simulation et de placement routage.

Cette couche « back-end » est composée de bibliothèques de cellules spécifiques. Leur composition découle de l'étude approfondie de la clé de voûte des circuits asynchrones : le canal de communication. Celui-ci permet de gérer l'échange de données entre deux blocs fonctionnels et par conséquent leur synchronisation locale (par opposition aux circuits synchrones).

Une communication à travers un canal est gérée par un protocole. Il peut se décomposer en deux ou quatre phases. Les travaux présentent l'ensemble des protocoles quatre phases. Dans la communauté asynchrone, la façon de les définir est toujours différente (nom des signaux, polarité des signaux de contrôle, modèle de représentation, logique de représentation des circuits correspondant). Il est de ce fait difficile d'avoir une vision unifiée de ces protocoles. L'état de l'art effectué sur les protocoles et les circuits associés offre une vision unifiée jamais proposée jusqu'à présent. Elle permet d'en tirer certaines similitudes et de les classer en fonction de leurs caractéristiques. A la fin de cette étude, quatre protocoles ont été choisis car ils sont les plus représentatifs et offrent le meilleur compromis performance/complexité : séquentiel, WCHB, PCHB, PCFB. En liaison avec l'outil TAST, ces quatre protocoles sont proposés à l'utilisateur lors de la phase de synthèse.

Dans le contexte de l'étude du canal de communication, la synthèse des protocoles est faite à travers deux méthodes : l'une basée sur un modèle expansion des communications (HSE) initialement proposée par A. Martin et l'autre sur un modèle à base de graphes de transition des signaux (STG). Elles sont appliquées sur les protocoles choisis de façons très détaillées. Cette attention particulière a pour but de bien maîtriser les différentes étapes de la synthèse.

Cette maîtrise a permis d'étendre la synthèse des contrôleurs à l'ensemble des structures de bases indispensables pour la synthèse des circuits asynchrones. La conclusion de ce travail est que pour réaliser cette synthèse, il est nécessaire d'avoir un ensemble de cellules spécifiques autre que les cellules standard de la logique synchrone. Ces cellules sont les portes de Muller symétriques et dissymétriques. Elles permettent de réaliser la synchronisation entre deux signaux à un niveau de granularité de la porte.

Ces cellules sont réunies dans des bibliothèques qui interviennent dans les différentes étapes du flot de conception TAST :

- La bibliothèque générique de cellules décrite en VHDL comportemental, permet de décrire les circuits synthétisés et de les simuler pour valider leur fonctionnement. La description VHDL comportementale est indépendante de la technologie. La simulation des circuits peut être faite indifféremment avec les outils commerciaux Mentor (Modelsim), Synopsys (VSS), Cadence ...

- La bibliothèque structurelle à base de cellules standard cible directement des cellules appartenant à une technologie bien particulière. Ce modèle permet d'avoir des résultats de simulation plus significatifs car il fait appel à des descriptions VITAL des portes (VHDL avec délais). Il permet aussi de réaliser des simulations électriques pré-layout (niveau transistor), d'appliquer les outils de placement routage et d'effectuer des simulations post-layout rétro annotées.

- La bibliothèque structurelle à base de LUT cible les familles de FPGA d'Altera et de Xilinx. L'implantation d'un circuit asynchrone sur un FPGA standard est quasiment automatisée (excepté la génération d'un fichier de contraintes). Seule, la bibliothèque spécifiée (en format EDIF ,VHDL …) est nécessaire. Elle garantit l'absence d'aléa dans les portes. La méthode a été réalisée sur chaque famille de FPGA et a fait l'objet de plusieurs applications concrètes (filtres, DES).

Le développement de ces bibliothèques permet donc de réaliser le lien entre le « front-end » et le « back-end » de l'environnement TAST et d'appliquer un flot complet de conception de systèmes asynchrones.

Le dernier chapitre de la thèse traite du problème de la réalisation d'arbitres dans le cadre de la conception de réseaux de communication sur puce (« Network on Chip »). Les différents algorithmes d'arbitrage présentés ont été modélisés en langage CHP et reposent au niveau de la synthèse sur deux cellules particulières que sont le synchroniseur et le bloc d'exclusion mutuelle. Ces deux dernières cellules sont donc venues compléter les bibliothèques de cellules précédemment décrites. Enfin les arbitres à priorité étudiés montrent :

- qu'il est possible de les modéliser en CHP,

- qu'ils sont très modulaires (séparation de la résolution des contentions et de la fonction de priorité),

- qu'ils peuvent faire l'objet d'une synthèse automatique.

De plus les propriétés générales de la logique asynchrone (faible consommation, indépendance aux délais) sont particulièrement intéressantes pour la conception de ces réseaux de communication sur puce.

L'objectif initial de cette thèse est donc atteint et représente une contribution pour l'évolution de l'outil TAST en particulier et des outils de conception de circuits asynchrones en général.

Les perspectives de ce travail seraient l'exploitation de ces résultats (spécifications de bibliothèques pour la conception de circuits asynchrones en technologies ASIC et FPGA) pour la conception spécifique des bibliothèques en layout optimisant des paramètres tels que la surface, la consommation, la vitesse…Une deuxième perspective concerne le développement d'algorithmes de synthèse logique et de projection technologique (« technology mapping ») utilisant ces bibliothèques.

Bibliographie

[BAIX 02] A.Baixas, Cryptographie sur FPGA standard en technologie asynchrone, rapport de DESS CSINA, Grenoble, France, septembre 2002.

[BERK 93] K. Van Berkel, Beware the isochronic fork, Integration, the VLSI journal, N°. 13, pp. 103-128, 1993.

[BRAC 00] L. E. M. Brackenbury P. A. Riocreux, M. J. G. Lewis. Power reduction in self-timed circuits using early-open latch controllers. Electronics Letters, 36(2):115-116, 2000.

[BURN 88a] Steven M. Burns and Alain J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In J. Allen and F. Leighton, editors, Advanced Research in VLSI, pages 35-50. MIT Press, 1988.

[BURN 88b] Steven M. Burns. Automated compilation of concurrent programs into self-timed circuits. Master's thesis, California Institute of Technology, 1988.

[BYST 00] A. Bystrov, D. J. Kinniment, and A. Yakovlev. Priority arbiters. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 128-137. IEEE Computer Society Press, April 2000.

[CHU 85] T.-A. Chu, C. K. C. Leung, and T. S. Wanuga. A design methodology for concurrent VLSI systems. In Proc. International Conf. Computer Design (ICCD), pages 407-410. IEEE Computer Society Press, 1985.

[CHU 87] Tam-Anh Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In Proc. International Conf. Computer Design (ICCD), pages 220-223. IEEE Computer Society Press, 1987.

[CORT 96] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Technical report, Universitat Politècnica de Catalunya, 1996.

[CORT 02] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Logic Synthesis of Asynchronous Controllers and Interfaces, Springer-Verlag, 2002.

[DAVI 97] Al Davis and Steven M. Nowick. An introduction to asynchronous circuit design. Technical Report UUCS-97-013, Dept. of Computer Science, University of Utah, September 1997.

[DAY 95] Paul Day and J. Viv Woods. Investigation into micropipeline latch design styles. IEEE Transactions on VLSI Systems, 3(2):264-272, June 1995.

[DILL 86] David L. Dill and Edmund M. Clarke, "Automatic verification of asynchronous circuits using temporal logic", IEE Proceedings, Computers and Digital Techniques, Vol. 133, pp. 272-282, September, 1986.

[DINH 02a] A.V. Dinh Duc et al., TAST, appeared in the 8th IEEE Intl. Symposium on Asynchronous Circuits and Systems, Manchester, UK, Apr. 8-11, 2002, ISRN: TIMA-RR--02/04/01--FR

[DINH 02b] A.V Dinh Duc et al, "DTL: the Foundation of a general design methodology for asynchronous circuits", published to ASYNC'02, Manchester, UK

[DINH 02c] A. V. Dinh Duc, L. Fesquet, M. Renaudin, Synthesis of QDI Asynchronous, Circuits from DTL-style Petri Net, 11 th Workshop on Logic and Synthesis (IWLS'02), New Orleans, Lousiana, USA, June 4-7, 2002.

[EBER 90] Jo C. Ebergen. Arbiters: An exercise in specifying and decomposing asynchronously communicating components. Research Report CS-90-29, Computer Science Dept., Univ. of Waterloo, Canada, July 1990.

[EBER 91] J. Ebergen, A formal approach to designing delay-insensitive circuits, Distributed Computing, Vol. 5, N°. 3, pp. 107-119, July 1991.

[EBER 95] Jo C. Ebergen, John Segers, and Igor Benko, "Parallel program and asynchronous circuit design", In Graham Birtwistle and Al Davis, editors, "Asynchronous Digital Circuit Design", Workshops in Computing, pp. 51-103, Springer-Verlag, 1995.

[FURB 96a] Stephen. B. Furber and Paul Day. Four-phase micropipeline latch control circuits. IEEE Transactions on VLSI Systems, 4(2):247-253, June 1996.

[FURB 96b] S. B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. IEEE Computer Society Press, March 1996.

[HAUC 95] Scott Hauck., Asynchronous design methodologies: An overview. Proceedings of the IEEE, 83(1):69-93, January 1995.

[HAUC 99] O. Hauck, M. Garg, and S. A. Huss. Two-phase asynchronous wave-pipelines and their application to a 2D-DCT. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 219-228, April 1999.

[HUFF 64] D. A. Huffman. The synthesis of sequential switching circuits. In E. F. Moore, editor, Sequential Machines: Selected Papers. Addison-Wesley, 1964.

[JOSE 96] Mark B. Josephs and Jelio T. Yantchev, "CMOS design of the tree arbiter element", IEEE Transactions on VLSI Systems, Vol. 4, N° 4, pp. 472-476, December, 1996.

[KESS 00] Joep Kessels, Ad Peeters, Torsten Kramer, Markus Feuser, and Klaus Ully. Designing an asynchronous bus interface. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 108-117. IEEE Computer Society Press, March 2001.

[KOND 98] Alex Kondratyev, Michael Kishinevsky, and Alex Yakovlev. Hazard-free implementation of speed-independent circuits. IEEE Transactions on Computer-Aided Design, 17(9):749-771, September 1998.

[LEWI 99] M. Lewis, J. D. Garside, and L. Brackenbury. Reconfigurable latch controllers for low power asynchronous circuits. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 27-35, April 1999.

[LINE 95] Andrew Matthew Lines, Pipelined Asynchronous Circuits, CS-TR-95-21, 1995.

[MANO 01] Rajid Manohar. An analysis of reshuffled handshaking expansions. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 96-105. IEEE Computer Society Press, March 2001.

[MART 90] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, Advanced Research in VLSI, pages 263-278. MIT Press, 1990.

[MART 93] A.J. Martin, "Synthesis of Asynchronous VLSI Circuits", Internal Report, Caltech-CS-TR-93-28, California Institute of Technology, Pasadena, 1993. or A.J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits", in "Developments in Concurrency and Communication", edited by C.A.R. Hoare, Addison Wesley, pp. 1-64, 1990.

[MART 97] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nystroem, Paul Penzes, Robert Southworth, and Uri Cummings. The design of an asynchronous MIPS R3000 microprocessor. In Advanced Research in VLSI, pages 164-181, September 1997.

[MOOR 02] Simon Moore, George Taylor, Robert Mullins, and Peter Robinson. Point to point GALS interconnect. In *Proc*. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 69-75, April 2002.

[MULL 65] R. E. Muller, Sequential Circuits, Chapter 10, Switching Theory, Vol. 2, N.Y. Wiley, 1965.

[NOWI 91] Steven M. Nowick and David L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In Proc. International Conf. Computer-Aided Design (ICCAD), pages 318-321. IEEE Computer Society Press, November 1991.

[PARK 97] Sung Bum Park and T. Nanya. Synthesis of asynchronous circuits from signal transition graph specifications. IEICE Transactions on Information and Systems, E80-D(3):326-335, March 1997.

[PIGU 98] C. Piguet and J. Zahnd. STG-based synthesis of speed-independent CMOS cells. In Workshop on Exploitation of STG- Based Design Technology, July 1998.

[PIGU 01] C. Piguet, M. Renaudin, T. Omnes, "Special Session on Low-power Systems on Chips", Design Automation and Test in Europe (DATE), Munich, Germany, March 13-16, 2001, pp. 488-494.

[RENA 96] Marc Renaudin, Bachar El Hassan, and Alain Guyot. New asynchronous pipeline scheme: Application to the design of a self-timed ring divider. IEEE Journal of Solid-State Circuits, 31(7):1001-1013, July 1996.

[RENA 98] M. Renaudin, P. Vivet, and F. Robin. ASPRO-216: A standard-cell QDI 16-bit RISC asynchronous microprocessor. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 22-31, 1998.

[RENA 00] M. Renaudin and J. B. Rigaud. Modelling and design/synthesis of arbitration problems. In Alex Yakovlev and Reinder Nouta, editors, Asynchronous Interfaces: Tools, Techniques, and Implementations, pages 121-128, July 2000.

[SPAR 01] J. Sparsø and S. Furber (eds.), Principles of asynchronous circuit design - A systems perspective. Kluwer Academic Publishers, 2001.

[SUTH 89] Ivan E. Sutherland. Micropipelines. Communications of the ACM, 32(6):720-738, June 1989.

[UNGE 69] S. H. Unger. Asynchronous Sequential Switching Circuits. Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[VILL 02] T. Villiger, F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner,Multi-point Interconnect for Globally-Asynchronous Locally-Synchronous Systems, Second ACiD-WG Workshop, Munich, Germany, 28-29 January, 2002

[VIVE 01] P. Vivet, Une méthodologie de conception de circuits intégrés quasi-insensibles aux délais : application à l'étude et à la réalisation d'un processeur RISC 16-bit asynchrones, Thèse, Grenoble, France, juin 2001.

[WAKE 94] John F. Wakerly, "Digital Design, Principles and Pratices", Second Edition, Prentice, 1994.

[YUN 96a] K. Y. Yun, P. A. Beerel, and J. Arceo. High-performance asynchronous pipeline circuits. In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems. IEEE Computer Society Press, March 1996.

[YUN 96b] K. Y. Yun, P. A. Beerel, and J. Arceo. High-performance two-phase micropipeline building blocks: double edge- triggered latches and burst-mode select and toggle circuits. IEE Proceedings, Circuits, Devices and Systems, 143(5):282-288, October 1996.

Liste des publications

Livre (contributions)

[RIGA 02a] Rigaud J.-B., Quartana J., Fesquet L., Renaudin M., Modeling and design of asynchronous priority arbiters for on-chip communication systems, article selectionné suite à la conférence VLSI-SoC'01 et publié dans le livre Soc Design Methodologies, Editeur : Kluwer Academic Publishers, juin 2002.

Communications effectuées à des manifestations d'audience internationale avec comité de sélection et actes

[BORR 02] Borrione D., Boubekeur M., Dumitrescu E. (VDS Group), Renaudin M., Rigaud J.-B., Sirianni A. (CIS Group), An approach to the introduction of formal validation in an asynchronous circuit design flow, Design Correct Circuits (DCC'02), Grenoble, France, April 6-7, 2002.

[BORR 03] Borrione D., Boubekeur M., Dumitrescu E. (VDS Group), Renaudin M., Rigaud J.-B., Sirianni A. (CIS Group), An approach to the introduction of formal validation in an asynchronous circuit design flow, (à paraître) Thirty-Sixth Hawaii International Conference on System Sciences (HICSS-36), Hawaï, USA, January 6-9, 2003.

[HO 02] Ho Q.T., Rigaud J.-B., Fesquet L., Renaudin M., Rolland R., Implementing asynchronous circuits on LUT based FPGAs, 12th International Conference on Field Programmable Logic and Application (FPL'02), La Grande Mote, France, September 2-4, 2002.

[DINH 02d] Dinh Duc A.V., Rigaud J.B., Rezzag A., Sirianni A., Fragoso J., Fesquet L., Renaudin M., TAST CAD Tools, 2nd Asynchronous Circuit Design Workshop of the European Commission's Fifth Framework Programme (ACID'02), Munich, Germany, 28-29 January 2002.

[RENA 00] Renaudin M., Rigaud J.-B., Modeling and design/synthesis of arbitration problems, Asynchronous Interfaces : Tools, Techniques and Implementations Workshop (AINT'2000), TU Delft, The Netherlands, July 19-20 2000 .

[RIGA 01] Rigaud J.-B., Quartana J., Fesquet L., Renaudin M., Modeling and design of asynchronous priority arbiters for on-chip communication systems, 11th IFIP International Conference On Very Large Scale Integration (VLSI-SOC'01), Le Corum, Montpellier, France, December 3-5, 2001, pp424-429.

[RIGA 02b] Rigaud J.-B., Quartana J., Fesquet L., Renaudin M., High-level modeling and design of asynchronous priority arbiters for on-chip communication systems, Design, Automation and Test in Europe (DATE'02), Le Palais des Congrés, Paris, France, March 4-8, 2002, pp1090.

Annexes :

Syntaxe du CHP

Dans cette annexe est présentée la syntaxe du langage CHP. Elle provient du document édité pour le tutorial sur l'outil TAST à l'occasion de l'école d'été ACID'02 organisée par le groupe CIS à Grenoble au mois de juillet 2002 [DINH 02c].

CHP Syntax

In this section, the CHP language is briefly introduced to provide the minimum necessary background to enable the reader to read and understand the programs used through out the tutorial.

1. Literal

Integer constants are noted in fixed precision with the following syntax:
"<digit>.<digit>…" [base][length] which specifies a vector of "length" element represented in base "base".
Each digit belongs to [0, base-1], and "base" and "length" are non-zero natural numbers.
Example:

"1.9.7.9"[10] is obviously number 1979.

"1.2.3"[4] = $1*4^2 + 2*4^1 + 3*4^0 = 27$.

In order to avoid this notation for usual bases such as 10 and 2, standard notations are also supported for integers and binary numbers.

1.1. Unsigned data type

MR[B] : *Multi-Rail in base B*

This type represents a number between 0 and (B – 1), encoded with the "1-of-n" delay insensitive code.
Example:

```
VARIABLE        b : MR [B] ;          -- declaration
                b := "x" [B] ;        -- assignment, with 0 <= x < B
```

MR[B] [L] : *Vector of L Multi-Rails in base B*

This type corresponds to a vector of L elements of type MR [B]. It represents a number between 0 and (B^L - 1).

Example:

```
VARIABLE        b : MR [B] [3] ;        -- declaration
                b := "0.0.0" [B] ;      -- assignment
```

Based on these basic unsigned types, some other types are defined to make designers' life easier. They are:

DR: *Dual Rail* – equivalent to MR[2]

DR[L] : *Vector of L Dual Rail elements* – equivalent to MR[2][L]

BIT: *Binary* – equivalent to MR[2]

BIT[L] : *Vector of Bits* – equivalent to MR[2][L]

BOOLEAN: equivalent to MR[2]

NATURAL[Max] : *MR[2][L] type ranging from 0 to Max value*

L is the superior integer part of (Log_2 [Max+1]). Max is a natural value, which must be specified during the declaration of the element of type NATURAL.

SR: *Single Rail* – equivalent to MR [1]

This type does not carry any value, it allows the specification of synchronizations between communicating processes. The SR type only applies to Channel or Port declaration. It is not relevant for variables.

1.2. Signed data type

All the types previously presented, except SR, have a signed equivalent. The complement to the base B is used to encode negative numbers. Thus, the same representation as unsigned types is used, but the highest order digit is interpreted as a signed number.

SMR[B] : *Signed Multi Rail in base B*

This type represents a number between -(B / 2) and (B / 2) - 1, if B is even or between -(B - 1) /2 and (B - 1) / 2, if B is odd. The representation of a negative value is obtained with the complement to the base B. For instance in base 16, -2 is represented by 14 (16 + (-2) = 14).

Example:

```
VARIABLE        b : SMR [4];      -- declaration
                b := "2" [4];     -- assignment : b = 2 - 4 = -2
```

SMR[B][L] : *Signed Vector of L Multi Rail in base B*

This type corresponds to a signed vector of L elements of type MR [B]. To evaluate the number, only the highest order digit is signed, the other digits keep their positive values. For instance "1.5"[16] = 1*16 + 5 = 21 while "15.5"[16] = (15 - 16)*16 + 5 = -11.

Example:

VARIABLE b : SMR [3] [3]; -- declaration
b := "2.2.2" [3] -- b = $(-1)*3^2 + 2*3^1 + 2*3^0 = -1$

From these basic signed types, other types are defined as abbreviations:

SDR: *Signed Dual Rail* - equivalent to SMR [2]

SDR[L] : *Signed Vector of L Dual Rail values* - equivalent to SMR [2][L]

INTEGER[Max] : *SMR [2][L] type ranging from -(Max+1) to Max.*

2. Operator

The following operators are available in our CHP language. They are all defined for MR and SMR data types.

Comparison:	=, /=, <, <=, >, >=
Arithmetic:	+, -, *, mod, sll, sla, srl, sra, rol, ror
Logical:	not, nand, and, nor, or, xnor, xor
Communication actions:	! (send), ? (receive), # (probe)
Assignment/conversion:	:=
Sequential/parallel:	";" (sequential) and “,” (parallel)

3. Control structure

Deterministic selection. It waits for a unique true guard. Once one of the guards is evaluated to true, it executes the associated bloc and terminates the selection.	[guard1 => bloc1 @ guard2 => bloc2 @ …]

Non-deterministic selection. It waits for one or more true guards. One of the true guards is selected, and the associated bloc executed. The selection then terminates.	[guard1 => bloc1 @@ guard2 => bloc2 @@ …]
Deterministic loop. While a unique guard is true, it executes the corresponding bloc. It terminates when none of the guard is true.	*[guard1 => bloc1 @ guard2 => bloc2 @ …]
Non-deterministic loop. While one or more guards are true, one of the true guards is selected and the corresponding bloc executed. It terminates when none of the guard is true.	*[guard1 => bloc1 @@ guard2 => bloc2 @@ …]

4. Program structure

CHP Component It is made of its communication interface, followed by a declaration part and its body. The communication interface is a directed port list, similar to VHDL. The declaration part declares local objects like channels and constants. The body is made of concurrent processes or component instances. They can all communicate with each other, and also with the component ports. Point-point communication is only allowed.	COMPONENT component_name PORT (port_list) {declaration part} Begin component body End component_name ;
Process It is made of a port list, a declaration part and a body. It is important to mention that a process is a loop. When the last instruction completes, the process restarts the first instruction.	PROCESS process-name PORT (port_list) {declaration part} [instruction_list]

5. Example

As an example, consider the following program which specifies a selector. Because there is only one process, the component port list and the process port list are identical. Channel C is read in the local variable “ctrl” which is tested using a deterministic choice structure. If “ctrl” is ‘0’ then the value read from channel “E” is propagated to channel “S1”. If “ctrl” is ‘1’ then the value read from channel “E” is propagated to channel “S2”. Finally, if “ctrl” is “3” then the value read from channel “E” is propagated to both channels “S1” and “S2” in parallel.

```
COMPONENT Selector
   PORT ( E: in DR;  C: in MR[3][1];
          S1, S2 : out DR )
Begin
PROCESS main
   PORT ( C: in MR[3][1];  E: in DR;
          S1, S2 : out DR )
Variable x : DR;
Variable ctrl : MR[3][1];
[
  *[ C ? ctrl ; [  ctrl = “0”[3] => E ? x; S1 ! x
                 @  ctrl = “1”[3] => E ? x; S2 ! x
                 @  ctrl = “2”[3] => E ? x; S1 ! x, S2 ! x
             ]
    ]
]
End Selector;
```

CHP code of a Selector

RESUME

Ce travail de thèse s'intègre dans le développement de l'outil de conception automatique de circuits asynchrones TAST (« TIMA Asynchronous Synthesis Tool »). C'est un environnement de conception principalement composé d'un compilateur et d'un synthétiseur offrant la possibilité de cibler plusieurs formats de sortie (description comportementale en VHDL, langage C, description structurelle au niveau porte en VHDL) à partir de descriptions de haut niveau décrites en langage CHP. Le résultat produit par le synthétiseur est une description au niveau porte qui utilise une bibliothèque de cellules génériques.

Cette thèse s'attache à la réalisation de l'interface entre le « front-end » de l'outil TAST et le « back-end » intégrant les outils commerciaux du flot de conception traditionnel. Pour cela, la bibliothèque générique qui permet de décrire le résultat de la synthèse a été spécifiée puis développée. Cette bibliothèque est une description fonctionnelle des portes qui est indépendante de la technologie. De plus, cette interface comprend des bibliothèques de cellules spécifiques qui permettent de concevoir un circuit de type ASIC ou de type FPGA. La spécification de la bibliothèque de cellules génériques a nécessité une étude approfondie sur les communications à travers un canal. Cette étude présente d'une manière unifiée les différents moyens pour réaliser ces communications et les différentes façons de les implanter au niveau circuit. Pour compléter le développement de la bibliothèque, le problème de la réalisation d'arbitres est abordé dans le cadre élargi de la conception de réseaux de communication sur puce (« Network on Chip »).

MOTS-CLES

Circuits asynchrones, synthèse de circuits asynchrones, langage de processus communicants, réseau de Pétri, protocoles de communications, arbitrage, bibliothèques, cellules standard, portes de Muller.

TITLE

LIBRARIES SPECIFICATION FOR THE SYNTHESIS OF ASYNCHRONOUS CIRCUITS

ABSTRACT

This PhD thesis deals with the development of the TAST automated tool suite (« TIMA Asynchronous Synthesis Tool »). It is a design environment mainly made up of a compiler and a synthesizer targeting several output formats (VHDL behavioral description, C language description, VHDL gate level netlist) starting from a high level description language (CHP). The synthesizer produces is a gate netlist which uses a generic cell library.

This work presents the interface between the front-end of TAST and its back-end integrating commercial tools of the standard design flow. To do that, this generic library, used to describe the result of the synthesis, was specified and developed. This library includes a technology independent functional description of the gates. Moreover, the interface includes specific libraries for ASIC and FPGA. The generic library specification required an in-depth understanding of the message passing mechanisms through channel. Therefore this work adopted a unified way to study the communication protocols and their corresponding implementations. To complete the library specification and development, arbiter design was considered as a contribution to network on-chip implementation.

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France.

ISBN : **2-913329-95-0 (version brochée)**

ISBNE : **2-913329-96-9 (version électronique)**