Université Pierre et Marie Curie – Paris VI

Mémoire d'Habilitation à diriger des Recherches

Spécialité Informatique

# Contributions pour la conception et la vérification de systèmes matériels embarqués

Emmanuelle Encrenaz-Tiphène

LIP6 / équipe ALSOC – Université Paris VI & CNRS

LSV - ENS de Cachan & CNRS

soutenue le lundi 18 juin 2007

devant le jury composé de

| | | |
|---|---|---|
| Dominique Borrione | Rapporteur, | professeur à l'Université Joseph Fourier, Grenoble |
| Jordi Cortadella | Rapporteur | professeur à l'Université Polytechnique de Catalogne, Barcelone |
| Claude Jard | Rapporteur | professeur à l'ENS-Cachan, Rennes |
| Paul Feautrier | Examinateur | professeur à l'ENS-Lyon |
| Laurent Fribourg | Examinateur | Directeur de recherche CNRS au LSV, Cachan |
| Alain Greiner | Examinateur | professeur à l'Université Pierre et Marie Curie, Paris |
| Oded Maler | Examinateur | Directeur de recherche CNRS au VERIMAG, Grenoble |

# Table des matières

**Table des matières** **3**

**Introduction** **1**

**I Conception incrémentale et vérification de composants** **3**

**1 Une démarche de conception incrémentale** **5**
1.1 Motivations . . . . . 5
1.1.1 Tour d'horizon d'autres méthodes de conception . . . . . 6
1.2 Définition d'un composant et de sa spécification . . . . . 8
1.3 Définition de la notion d'incrément . . . . . 8
1.3.1 Que modélise-t-on avec ces types d'incrément ? . . . . . 11
1.4 Couplage conception et vérification incrémentales . . . . . 11
1.4.1 L'interaction conception-vérification . . . . . 11
1.4.2 Relations entre incrément et transformations de propriétés CTL . . . . . 12
1.5 Incréments représentant les ruptures de progression d'un pipeline . . . . . 16
1.5.1 Les incréments *stall* et *kill* . . . . . 17
1.5.2 Conséquences sur les transformations de propriétés CTL . . . . . 18
1.6 Application à la conception de convertisseurs de protocoles . . . . . 18
1.6.1 Convertisseurs de protocoles entre VCI et les bus PI et AMBA . . . . . 19
1.6.2 Conception incrémentale du wrapper maître VCI-PI . . . . . 20
1.6.3 Vérification incrémentale du wrapper maître . . . . . 21
1.7 Conclusion . . . . . 24

**2 Abstraction de composants à partir de leur spécification** **25**
2.1 Vue générale de la démarche proposée . . . . . 27
2.2 Algorithme d'abstraction . . . . . 28
2.3 Premiers résultats expérimentaux . . . . . 29
2.4 Conclusion et Perpectives . . . . . 30

**II Algorithmique pour la vérification** **31**

**3 Vérification de systèmes décrits à un niveau abstrait** **33**
3.1 Préliminaires . . . . . 33
3.1.1 Description des systèmes : un sous-ensemble statique de Promela . . . . . 34
3.1.2 Techniques de vérification par modèle – description et limites . . . . . 35

3.2 DDD plats et hiérarchiques . . . 37
3.2.1 Définition structurelle . . . 37
3.2.2 Introduction d'un ordre sur les variables . . . 38
3.2.3 Opérations sur les séquences et sur les ensembles de séquences . . . 39
3.2.4 Transformation d'ensembles : la notion d'homomorphisme . . . 39
3.2.5 Mécanismes d'accès propres aux DDD hiérarchiques . . . 40
3.3 Application aux systèmes décrits dans le sous-ensemble statique de Promela . . . 41
3.3.1 Représentation d'un état sous forme de DDD hiérarchique . . . 41
3.3.2 Homomorphismes associés aux instructions du langage Promela . . . 43
3.3.3 HaDDock, un model-checker symbolique . . . 43
3.3.4 Résultats expérimentaux . . . 44
3.4 Prolongements possibles . . . 47

**4 Détection automatique de deadlocks dans les NoCs 49**
4.1 Détection de deadlock dans les NoCs - état de l'art . . . 50
4.1.1 Hypothèses et résultats classiques . . . 50
4.1.2 Limitations . . . 51
4.1.3 Une solution pratique . . . 51
4.2 Prise en compte des dépendances liant plusieurs types de messages . . . 52
4.2.1 Hypothèses supplémentaires . . . 52
4.2.2 Théorème de Duato . . . 52
4.2.3 Extension à plusieurs types de messages . . . 53
4.2.4 Algorithme de détection de deadlock . . . 54
4.3 Notre algorithme de preuve d'absence de deadlock . . . 54
4.4 Résultats expérimentaux . . . 55
4.4.1 Élements de comparaison . . . 56
4.4.2 Ensemble de test . . . 56
4.4.3 Résultats et discussion . . . 57
4.5 Prolongements possibles . . . 57

**III Vérification des contraintes temporelles de composants 59**

**5 Vérification des contraintes temporelles de composants mémoires embarqués 61**
5.1 Détermination des caractéristiques temporelles d'un composant embarqué . . . 62
5.2 Analyse du problème TSE inverse paramétré . . . 66
5.2.1 Problème TSE inverse . . . 66
5.2.2 La méthode . . . 67
5.2.3 Optimisation . . . 67
5.2.4 Implémentation . . . 68
5.3 Application à la mémoire embarquée SPSMALL . . . 68
5.4 Conclusion et prolongements envisagés . . . 74

**IV Conclusion et Perspectives 77**

**Bibliographie 81**

# Introduction

Ce mémoire est une synthèse des travaux de recherche auquels j'ai participé durant ces dernières années. La thématique concerne le développement et l'utilisation des méthodes formelles pour la conception et la vérification de composants matériels.

Le contexte très concurrentiel des SoC (system on Chip) nécessite le développement de nouveaux produits - extrêmement coûteux - à une cadence effrénée ; la complexité croissante des systèmes matériels rend leur vérification fonctionnelle de plus en plus nécessaire, mais également de plus en plus difficile. En pratique, ces systèmes sont essentiellement vérifiés par simulation : c'est actuellement la principale méthode qui permette de fournir des résultats -même très fragmentaires- sur des systèmes composés d'un ou plusieurs processeurs, de mémoires, de co-processeurs spécialisés, d'interfaces numériques ou analogiques, tous regroupés sur un seul circuit. Le besoin de méthodes de vérification automatiques s'appliquant sur des abstractions de ces systèmes est de plus en plus pressant : ceci explique le développement considérable des activités de recherche dans le domaine de la vérification des systèmes embarqués (ou systèmes sur puce) ces vingt dernières années.

Les techniques de vérification automatique (model-checking) ont connu un essor dans le domaine des systèmes matériels (composants électroniques) avec le développement des *méthodes symboliques* au début des années 90. En pratique, *l'equivalence checking* (comparaison de deux descriptions) de circuits combinatoires ou de certains circuits séquentiels est maintenant bien implantée dans le processus de conception. Par contre, les techniques de vérification de propriétés temporelles peinent à trouver leur place. Ceci s'est longtemps expliqué par deux facteurs : l'explosion combinatoire de l'ensemble de tous les états à examiner pour décider si une propriété est satisfaite ou non, et la difficulté pour écrire des propriétés complexes dans une logique temporelle. La seconde difficulté est en passe d'être contournée. L'introduction des langages d'assertion tels PSL ou SVA simplifie l'écriture des propriétés de logique temporelle, et la vérification par simulation de ces assertions est en plein essor, mais l'analyse n'est pas exhaustive. L'introduction de ces langages d'assertions familiarise les concepteurs avec l'écriture de propriétés logiques. La première difficulté est plus ardue. Si les méthodes de *Bounded Model-Checking* ont permis de repousser la taille des systèmes analysés, ceci s'est également fait au détriment de l'exhaustivité de l'analyse.

Dans ce contexte, je cherche à développer l'utilisation des techniques de vérification formelle dans le processus de conception des circuits intégrés, en confrontant les techniques actuelles à des exemples issus d'applications réelles et en proposant des améliorations à ces techniques.

Les principales contributions de ces travaux sont structurées en trois parties.

**Partie 1.** Elle décrit une méthode de conception incrémentale de composants de contrôle de flux et son implication sur le processus de vérification. La méthode de conception, les transformations induites sur la spécification ainsi que son application à la réalisation d'un convertisseur de protocole sont décrits dans le chapitre 1. Le chapitre 2 présente les prolongements naturels vers une méthode de vérification compositionnelle, permettant de combattre les problèmes d'explosion combinatoire.

**Partie 2.** Elle présente deux voies possibles pour améliorer l'analyse de systèmes embarqués.

Le chapitre 3 montre comment améliorer les techniques de model-checking symbolique en tirant partie de la structuration de la description du système à analyser . Elle s'applique à des descriptions abstraites de systèmes, et utilise les diagrammes de décision de données (DDD) pour représenter les ensembles d'états du système. Les résultats expérimentaux montrent l'intérêt de cette approche pour la vérification de systèmes présentant un fort degré de concurrence. le chapitre 4 se focalise sur l'analyse de deadlocks dans les réseaux (sur puce) : les résultats théoriques classiques sont étendus pour prendre en considération plusieurs types de messages interdépendants, et un algorithme de détection de deadlock très efficace en pratique est proposé.

**Partie 3.** Cette partie porte sur l'analyse des caractéristiques *temporelles* de systèmes. Le chapitre 5 présente une utilisation des méthodes de model-checking de systèmes temporisés pour déterminer des plages de bon fonctionnement du système. Cette méthode a été appliquée pour optimiser les temps de *setup* des signaux d'entrée d'une mémoire embarquée conçue et commercialisée par STMicroelectronics.

La conclusion récapitule les contributions et dresse les perspectives à moyen terme de ce travail.

# Première partie

# Conception incrémentale et vérification de composants

# Chapitre 1

# Une démarche de conception incrémentale

Les travaux décrits dans ce chapitre ont été menés dans le cadre du CERME (collaboration STMicroelectronics - LIP6 entre 2000 et 2005). Ils ont été en partie réalisés lors des travaux des stages de maitrise puis de DEA de J. Ossima-Khéba (2002 et 2003), le stage de DEA puis la thèse de C. Braunstein (2003 et 2003-2007).

## 1.1 Motivations

Le développement de composants de SoC est de plus en plus basé sur la réutilisation et l'assemblage de composants pré-existants. Après une pré-étude réalisée sur un modèle décrit au niveau transactionnel, les composants sont décrits par des modèles précis au cycle près et au bit près (ou *CABA cycle accurate, bit accurate)*, dans des langages de description permettant leur simulation (et dans certains cas leur synthèse), par exemple SystemC, VHDL, Verilog,... Dans la très grande majorité des cas, ces modèles sont vérifiés par simulation. Le développement d'outils de vérification basés sur des méthodes formelles conduit les concepteurs à adjoindre à leur composants une spécification, constituée d'un ensemble de formules logiques. PSL ou SVA sont les langages d'assertion les plus couramment utilisés dans le cadre de la simulation ; les logiques temporelles LTL et CTL sont elles communément utilisées pour la vérification par model-cheking.

Dans ce contexte, nous nous sommes intéressées à deux aspects :
- Comment ajouter une fonctionnalité à un composant déjà développé, et comment propager cet ajout sur la spécification du composant ?
- Comment vérifier la bonne fonctionnalité d'un assemblage de composants pré-existants (et dont le comportement, pris isolément, a été vérifié par ailleurs) ?

Ce chapitre présente une démarche de conception de composants incrémentale. Elle offre un cadre de conception garantissant une compatibilité ascendante lors d'ajout de nouvelles fonctionnalités à un composant. Cette démarche de conception est couplée à une démarche de vérification par model-checking. A chaque composant est associée une spécification, définie comme un en-

semble de propriétés décrites dans la logique CTL. Nous montrons comment la spécification d'un composant doit être modifiée et enrichie au fur et à mesure des ajouts fonctionnels de ce composant. Enfin la spécification d'un composant peut être utilisée comme une abstraction de ce composant lors de la vérification d'un assemblage de composants.

La démarche de conception incrémentale que nous proposons est pragmatique ; elle s'inspire de la façon dont les concepteurs conçoivent leurs composants, notamment pour réaliser des mécanismes de contrôle de flux : après avoir esquissé grossièrement le chemin de données du composant et son contrôle de flux nominal (sans à-coup ou rupture), le concepteur identifie un à un les événements pouvant perturber ce fonctionnement nominal, et complexifie son système (chemin de données et automate de contrôle) de telle sorte que ce dernier réagisse conformément à la nouvelle fonctionnalité introduite.

En focalisant l'attention du concepteur sur la prise en compte d'un événement extérieur à la fois, nous sérialisons les difficultés de conception. De plus, notre approche garantit *par construction* la compatibilité de la fonctionalité ajoutée avec l'ensemble des comportements pré-existants dans le composant. Elle permet en outre d'automatiser les transformations de la spécification au fur et à mesure des transformations imposées au composant.

### 1.1.1 Tour d'horizon d'autres méthodes de conception

Cette approche est à mettre en perspective avec d'autres démarches de conception de systèmes proposées dans la littérature, et pour certaines utilisées dans un contexte industriel.

La plupart des méthodes de conception basées sur des modèles formels exploitent la notion de *raffinement*. Dans cette voie, la méthode la plus communément répandue est la méthode B [89]. Celle-ci part de la spécification abstraite d'un système –décrite sous la forme d'une architecture très simplifiée, dans laquelle de nombreux choix (de types de données, ou technologiques) n'ont pas été effectués, et sont indéterministes–. Cette architecture vérifie un ensemble de propriétés globales. La méthode B permet de raffiner cette architecture, en définissant puis en concrétisant les composants qui la constituent, en fixant des choix qui étaient laissés indéfinis ou indéterministes. De ce fait, cette méthode peut être vue comme une démarche de *spécialisation* d'une architecture abstraite en un ensemble de composants concrets, qui implémentent un sous-ensemble des comportements qui étaient possibles dans l'architecture initiale. Si le raffinement est bien défini, alors le système raffiné vérifie *par construction* les propriétés globales vérifiées par le système abstrait. Cette démarche ne nécessite pas de vérification *a posteriori*, et est donc une alternative aux techniques de vérification par model-checking. Elle n'est toutefois pas automatique : il faut définir manuellement chaque étape de raffinement, et prouver que la transformation (manuelle) est un raffinement bien défini. Cette dernière étape est en général réalisée avec l'aide d'un assistant de preuve, mais elle requiert une grande interactivité. Cette méthode par raffinement est très intéressante pour concevoir des systèmes "clôts", dans lequel les composants interragissent dans un environnement confiné, qui n'est pas amené à s'étendre (ou bien alors l'extension de l'environnement n'induit pas de nouveaux traitements, qui n'auraient pas été définis dans la spécification abstraite). Elle est effectivement utilisée dans l'industrie, pour le développement de systèmes automatisés notamment (métro Meteor [23]). Sur le même principe, l'outil *Cadence-SMV* [4] développé par K. McMillan permet de concevoir et prouver des

architectures de processeurs complexes par raffinements successifs. Cette démarche me semble bien adaptée lorsque l'environnement dans lequel le système évolue est figé. En revanche, elle ne me semble pas simplement applicable pour modéliser l'ajout de nouvelles fonctionalités, qui ne sont pas des spécialisations de comportements pré-définis : il faut alors étendre la spécification initiale.

L'ajout de nouvelles fonctionalités à un composant a été largement étudié dans le contexte de *l'intégration de nouveaux services*. Ce problème s'est révélé avec le développement des services téléphoniques notamment. Dans ce contexte, plusieurs approches ont été proposées dans la littérature. Un outil automatisant l'intégration de services a été proposé par Plath et Ryan dans [109]. Cet outil nommé SFI permet de décrire et de combiner des services. Un service est lui même décrit sous la forme d'un programme, possédant des variables locales et réalisant des affectations sur ces variables et sur d'autres, et permettant de changer le comportement prédéfini par d'autres services. Les services ainsi décrits peuvent être interdépendants. Les éventuelles incompatibilités de services sont détectées *a posteriori*, par model-cheking. Ce mécanisme d'intégration de services est couplé à différents model-checkers (SMV, SPIN) ; une configuration initiale de services respecte une certaine spécification. Si l'ajout d'un service (ou d'une combinaison de services) invalide cette spécification initiale, les services ajoutés sont dits incompatibles. Cette méthode d'intégration de services, initialement proposée pour les systèmes clôts, a été étendue aux systèmes ouverts, et portée vers le model-cheker MOCHA [40]. Par ailleurs, Cansell et Mery, dans [39] proposent d'utiliser la méthode B pour déterminer l'incompatibilité de nouveaux services, en étendant la spécification initiale, puis en identifiant les incompatibilités à des obligations de preuve non prouvables.

Une démarche assez proche concerne la conception de systèmes par assemblage de composants, développés indépendemment les uns des autres. La problématique posée ici concerne la *compatibilité des composants à assembler* (et non pas l'extension des fonctionalités d'un composant).

Dans le contexte de la synthèse automatique de convertisseurs de protocoles, R. Passerano et al. proposent dans [105] et [104], de réaliser l'intersection des langages des protocoles à adapter, et en dériver un automate synthétisable. Si l'intersection est vide, les protocoles sont incompatibles. Par ailleurs, T. Henzinger et L. De Alfaro développent une approche similaire pour les ensembles de composants logiciels, décrits comme les agents d'un système ouvert, en introduisant le concept de *Compatible Interfaces* [62], puis plus tard de *Sociable Interfaces* [61]. Deux composants logiciels sont composables si l'environnement (dans lequel les deux composants sont plongés) dispose d'une stratégie permettant de faire collaborer les composants logiciels. Cela revient à dire que la compatibilité des composants dépend de l'environnement dans lequel on les place. Si l'environnement restreint suffisamment les comportements de chacun des composants à composer –jusqu'à exclure leurs interactions incompatibles–, alors les composants peuvent être assemblés *dans cet environnement particulier*. L'outil *ticc* [12] permet de définir des interfaces et de les composer.

Dans ce contexte, notre contribution peut se décrire comme suit. Nous proposons un cadre de conception pour des composants matériels synchrones, permettant d'ajouter de nouveaux comportements en réponse à une extension de l'environnement du composant. Les comportements ajoutés n'interagissent pas avec l'ensemble des comportements pré-existants dans le composant, ce qui nous distingue de Plath-Ryan, mais garantit la non régression (i.e. l'ensemble des com-

portements réalisables avant l'ajout est toujours réalisable après l'ajout). La problématique de composition de composants est différente de la nôtre : nous souhaitons étendre les comportements d'un composant, et faire évoluer sa spécification en conséquence, prenant en considération toutes les évolutions de son environnement.

Les ajouts que nous avons considérés permettent de modéliser des composants régulant le flux d'informations ou de traitements, tels des convertisseurs de protocoles, ou l'automate contrôlant le pipeline d'un processeur simple (MIPS R3000). Ils permettent également (et même surtout) de transformer la spécification du composant au fil du développement du composant, de façon automatique. Ce dernier point distingue notre méthode des approches d'ajout ou de composition citées précédemment. Il me semble que c'est un point très important que d'attacher une spécification à chaque étape du développement d'un composant.

## 1.2 Définition d'un composant et de sa spécification

Un composant $C_i$ est un dispositif matériel synchrone, muni d'une interface (un ensemble de signaux d'entrée $I$ et de sorties $O$) et dont l'ensemble de comportements est modélisé sous la forme d'une machine de Moore $M_i$. Il est muni d'une spécification $SPEC_i$, composée d'un ensemble de formules CTL.

**Définition 1** *Une Machine de Moore est un tuple* $M = \langle S, s_0, I, O, \delta, \gamma \rangle$ *dans lequel :*

$S$ *: un ensemble fini d'états ;*

$s_0$ *: un état initial*

$I$ *: un ensemble fini de signaux d'entrée ;*

$O$ *: un ensemble fini de signaux de sortie ;*

$\delta$ *: la fonction de transition :* $\delta : S \times 2^I \rightarrow S$.

$\gamma$ *: la fonction de génération :* $\delta : S \rightarrow 2^O$.

La spécification associée à un composant est un ensemble des propriétés, décrites dans la logique CTL, vérifiées par le composant $M_i$ (plus exactement, vérifiées dans les états initiaux de la structure de Kripke $K_i$ déduite de la machine de Moore, cf. Sec 1.4.2).

Le comportement d'un système combinant plusieurs composants est représenté par le produit synchronisé des machines de Moore des composants.

## 1.3 Définition de la notion d'incrément

Un incrément correspond à l'adjonction d'une nouvelle fonctionalité à un composant préexistant $C_i$. Un incrément est composé des trois éléments suivants :

- Un événement déclencheur $e$, matérialisé par un nouveau signal d'entrée, et la donnée de sa valeur d'activation (exprimant la survenue du nouvel événement), et sa valeur muette.

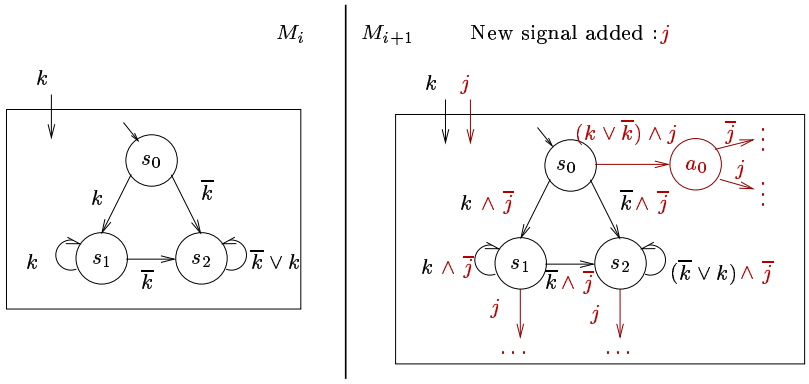


FIG. 1.1 – Extension de l'interface et des comportements d'un composant

- Une machine de Moore $M_{INC}$ modélisant le traitement à réaliser lorsque l'événement déclencheur a été activé.
- Un mode de connexion, précisant les points d'accès à la nouvelle fonctionalité, et classifiant les différentes possibilités de terminaisons du traitement ajouté.

L'événement déclencheur est défini comme suit.

**Définition 2** *Un* événement $e = \langle I_+, C_{ACT}(I_+), C_{QT}(I_+)\rangle$ *est défini tel que :*

$I_+$ **:** *L'ensemble des nouveaux signaux d'entrée et leur domaine de définition. Cet ensemble est disjoint de l'interface du composant à incrémenter.* $I \cap I_+ = \emptyset$.

$C_{ACT}(I_+)$ **:** *L'ensemble des configurations sur* $I_+$ *représentant l'occurrence du nouvel événement. Lorsque l'ensemble des signaux de* $I_+$ *est dans une telle configuration, l'événement est dit* actif. *On note* $c_act$ *une configuration de* $C_{ACT}$.

$C_{QT}(I_+)$ **:** *L'ensemble des configurations sur* $I_+$ *représentant l'absence d'occurrence du nouvel événement. Lorsque l'ensemble des signaux de* $I_+$ *est dans une telle configuration, l'événement est dit* muet. *On note* $c_qt$ *une configuration de* $C_{QT}$.

*Les ensembles de configurations actives et muettes forment une partition de l'ensemble des configurations sur* $I_+$. $C_{ACT}(I_+) \cup C_{QT}(I_+) = C(I_+)$ *et* $C_{ACT}(I_+) \cap C_{QT}(I_+) = \emptyset$. *On note abusivement* $\neg c_act \in C_{QT}$ *et* $\neg c_qt \in C_{ACT}$.

L'extension de l'interface du composant, ainsi que la prise en compte du nouvel événement sont illustrées sur la figure 1.1.

L'ensemble des comportements à ajouter lors de la survenue du nouvel événement est modélisé par une Machine de Moore[1] $M_{INC} = \langle S_{INC}, s0_{INC}, I+, O+, \delta_{INC}, \gamma_{INC}\rangle$, disjointe de celle de $M_i$ (i.e. les ensembles de signaux et d'états de $M_i$ et $M_{INC}$ sont disjoints).

La fonctionnalité à ajouter peut être munie d'une spécification sous la forme d'un ensemble de formules CTL vérifiées pour tous les états initiaux de $K_{INC}$ *lorsque* $K_{INC}$ *n'est pas connectée à* $K_i$. On note $K_{INC} \models \varphi$ iff $\forall s_0 \in S_{0_{INC}}, K_{INC}, s_0 \models \varphi$.

[1] Ce choix ne permet pas de représenter des incréments n'ajoutant que des transitions. Dans [35], nous donnons une définition de l'incrément qui ne souffre pas de cette restriction (mais pour laquelle les résultats de transformation de propriété CTL exposés en section 1.4.2 sont moins forts).

La connexion de $M_i$ et $M_{INC}$ peut être restreinte; Les relations $R_{i\to INC}$ et $R_{INC\to i}$ établissent la liste des états de $M_i$ et $M_{INC}$ à relier lors de l'ajout de l'incrément.

**Définition 3** Increment
*Soit $M_i$ la Machine de Moore caractérisant un composant initial, un incrément $INC$, déclenché sur l'événement $e$, appliqué à $M_i$, est un tuple*
$INC = \langle e, M_{INC}, R_{i\to INC}, R_{INC\to i}\rangle$ *tel que*

*$e$ est l'événement déclenchant,*

*$M_{INC}$ est l'ensemble des nouveaux comportements (ils peuvent faire intervenir des signaux présents dans $M_i$),*

*$R_{i\to INC} \subseteq S_i \times S_{INC}$ définit la connexion de $M_i$ vers $M_{INC}$*

*$R_{INC\to i} \subseteq S_{INC} \times S_i$ définit la connexion de $M_{INC}$ vers $M_i$*

L'ajout de l'incrément $INC$ au composant $C_i$ dont le comportement est modélisé par $M_i$ conduit à la définition d'un composant $C_{i+1}$ dont le comportement est défini par la machine $M_{i+1}$, construite suivant les rêgles décrites ci-après.

1. l'ensemble des états de $M_{i+1}$ est l'union des ensembles des états de $M_i$ et de $M_{INC}$.
2. L'état initial de $M_{i+1}$ est l'état initial de $M_i$.
3. L'ensemble des signaux d'entrée (resp. de sortie) de $M_{i+1}$ est l'union des ensembles des signaux d'entrées (resp. de sortie) de $M_i$ et de $M_{INC}$.
4. La fonction de transition est définie comme suit :
   (a) Les transitions qui étaient dans $M_i$ sont conservées, et la configuration de déclenchement associée est étendue par $e_qt$ la valeur muette de l'événement.
   (b) Les transitions de $R_{i\to INC}$ sont ajoutées. La configuration de déclenchement associée est $e_act$ la valeur active de l'événement.
   (c) les transitions de $R_{INC\to i}$ sont ajoutées. La configuration de déclenchement associée est *a priori* quelconque.
5. La fonction de génération de $M_{i+1}$ est l'extension des fonctions de génération de $M_i$ et de $M_{INC}$.

Cette définition correspond à **l'incrément général**. Plusieurs hypothèses portants sur $R_{i\to INC}$ et $R_{INC\to i}$ et $M_{INC}$ ont été définies, permettant d'affiner les relations entre les comportements décrits par $M_i$ et ceux décrits par $M_{INC}$.

**Hypothèses A-1** : $R_{i\to INC} \in S_i \times S_{0_{INC}}$ dans ce cas, les états de la frontière de $M_i$ vers $M_{INC}$ mènent *tous* vers l'état initial de $M_{INC}$ dans $M_{i+1}$. Cela modélise l'invocation préemtive d'une fonction (sans paramètres), ou un départ en exception.

**Hypothèse R-1** : $R_{INC\to i} = \emptyset$ dans ce cas, dans $M_{i+1}$, il n'y a pas de retour possible de $M_{INC}$ vers $M_i$. Cela modélise l'ajout d'un comportement entièrement nouveau et permanent, qui annihile les comportements précédemment définis (en ce sens qu'une fois le nouvel événement survenu, les anciens comportements ne pourront plus être atteints).

**Hypothèse R-2** : $R_{INC \to i} \in S_i \times S_{INC}$ et de plus, les transitions des états de $M_{INC}$ menant vers des états de $M_i$ sont tous étiquetés par une valeur particulière $e_rtn$. Cela modélise la terminaison d'un traitement avec signalisation explicite.

**Hypothèse K-1** : Il existe une relation d'équivalence entre les états de $M_i$ et les états de $M_{i+1}$. Nous nous intéressons aux équivalences de type *bisimulation faible*, décrivant des comportements *retardés* dans $M_{i+1}$ par rapport aux comportements décrits dans $M_i$.

### 1.3.1 Que modélise-t-on avec ces types d'incrément ?

Avec cet incrément, nous pouvons modéliser des ajouts de comportements préemptifs et n'introduisant pas de régression.

**Non régression.** L'incrément tel qu'il est défini garantit que les comportements ajoutés ne perturbent pas les comportements pré-existants. Il assure la non-régression des comportements lors des ajouts. Ceci vient du fait que la composition des machines de Moore est additive uniquement : il n'est pas possible de supprimer un comportement existant, ni de remplacer un comportement existant par un autre.

**Préemption.** La survenue d'un nouvel événement est prise en compte de façon préemptive : dès que le nouvel événement devient actif, la Machine $M_{i+1}$ quitte l'ensemble des états de $M_i$ pour se rendre dans les comportements définis par $M_{INC}$. Ce mécanisme d'incrément introduit une forme de priorité entre les événements induisant des incréments ajoutés à des étapes différentes : les derniers incréments ajoutés sont les plus prioritaires.

Par exemple, considérons une machine initiale $M_0$. Soit l'incrément $INC_1$, déclenché par l'événement $evt_1$, appliqué à $M_0$ et conduisant à la machine $M_1$. Soit l'incrément $INC_2$, déclenché par l'événement $evt_2$, appliqué à $M_1$ et produisant la machine $M_2$. Si, à un instant donné, $evt_1$ et $evt_2$ sont actifs simultanément, la machine $M_2$ exécutera les comportements définis par $M_{INC_2}$ et non pas ceux définis par $M_{INC_1}$. (Néanmoins, les comportements de $M_{INC_2}$ peuvent reproduire tout ou partie des comportements de $M_{INC_1}$.)

Si l'on veut adjoindre deux incréments au même niveau de priorité, on peut construire un événement représent le produit des deux incréments, et adjoindre ce dernier au composant.

## 1.4 Couplage conception et vérification incrémentales

### 1.4.1 L'interaction conception-vérification

La démarche de conception que nous proposons est couplée à une démarche de vérification par model-checking. Dans ce cadre, à chaque composant est associé une spécification. Cette spécification caractérise l'environnement admissible pour le composant (i.e. l'ensemble des événements auxquels le composant doit réagir), ainsi que l'ensemble des propriétés, exprimées dans la logique CTL, que doit vérifier le composant lorsqu'il est plongé dans son environnement.

Il est bien naturel de chercher à caractériser l'impact des modifications, introduites par l'incrément sur le composant, sur sa spécification. Par construction, l'ajout d'un incrément n'introduit pas de régression du composant $C_{i+1}$ par rapport au composant $C_i$. Cette propriété se traduit par un premier ensemble de règles de transformation automatiques, de la spécification de $C_i$ en une partie de la spécification de $C_{i+1}$. De plus, selon la nature de l'incrément (identifiée par les hypothèses A-1,R-1,R-2 et K-1 introduites précédemment), d'autres ensembles de rêgles de transformations, ou de préservation, peuvent être proposées. C'est l'objet de la présente section.

### 1.4.2 Relations entre incrément et transformations de propriétés CTL

La sémantique des formules CTL est définie par rapport à l'arbre d'exécution infini engendré par une structure de Kripke (un automate fini pour lequel les informations étiquettent les états uniquement). Du fait des configurations d'entrées étiquetant les transitions de la machine de Moore, cette dernière n'est pas une structure de Kripke. Une transformation classique, définie dans [93] permet de construire la structure de Kripke $K_i$ représentant les comportements de la machine de Moore $M_i$. Cette transformation incorpore les configurations d'entrées dans (des copies) des état source de la transition.

#### Propriétés de $K_{i+1}$ par rapport à $K_i$

A un composant $C_i$, on peut associer une structure de Kripke $K_i$. Soit $C_{i+1}$ le composant obtenu par ajout d'un incrément à $C_i$, on lui associe la structure de Kripke $K_{i+1}$. Nous avons montré dans [35] que $K_{i+1}$ *simule la structure* $K_i$.

Cela revient à dire que $K_i$ est isomorphe à un sous-graphe $G$ de $K_{i+1}$, et G est accessible à partir d'un état initial de $K_{i+1}$. Plus précisément, dans la structure $K_{i+1}$, $G$ comprend tous les états de $K_i$ étendus par $e_qt$, et accessibles à partir d'un état initial en suivant des chemins composés uniquement d'états étiquetés par $e_qt$. La frontière de $G$ dans $K_{i+1}$ est marquée par les états (dans $K_{i+1}$) étendant les états de $K_i$ par $e_act$. Les chemins dans $K_{i+1}$ issus d'un état de la frontière correspondent à des comportements ajoutés.

Cette délimitation de $G$ dans $K_{i+1}$ est représentée sur la figure 1.2. Elle correspond à l'ajout d'un incrément général (i.e. sans hypothèses supplémentaires). Une structure $K_i$ à quatre états (numérotés $1, 2, 3, 4$) est représentée sur la partie gauche de la figure. Une partie de la structure $K_{i+1}$ résultant de l'adjonction à $K_i$ d'un incrément $INC$ est représentée sur la partie droite de la figure. Elle met en évidence l'extension des états de $K_i$ aux configurations $e_qt$ et $e_act$ de l'increment. Ainsi, à l'état 1 de $K_i$ correspondent deux états dans $K_{i+1}$, représentés dans l'ovale 1. Un des états est étiqueté par $e_qt$ (en bleu) et l'autre étiqueté par $e_act$ (en rouge). $G$, le sous-graphe de $K_{i+1}$ isomorphe à $K_i$ est le sous-graphe connexe maximal de $K_{i+1}$ contenant uniquement des états étiquetés par $e_qt$ (le sous-graphe bleu). Les états de $K_{i+1}$ étendant les états de $K_i$ par $e_act$ sont tous désignés en rouge. Ils marquent la frontière entre $G$ et les nouveaux comportements, représentés par des arcs et états noirs.

L'invocation préemptive d'une fonction, et différentes possibilités de retours sont représentées sur la figure 1.3. La partie gauche de la figure présente l'allure d'une structure $K_{i+1}$ lorsque les

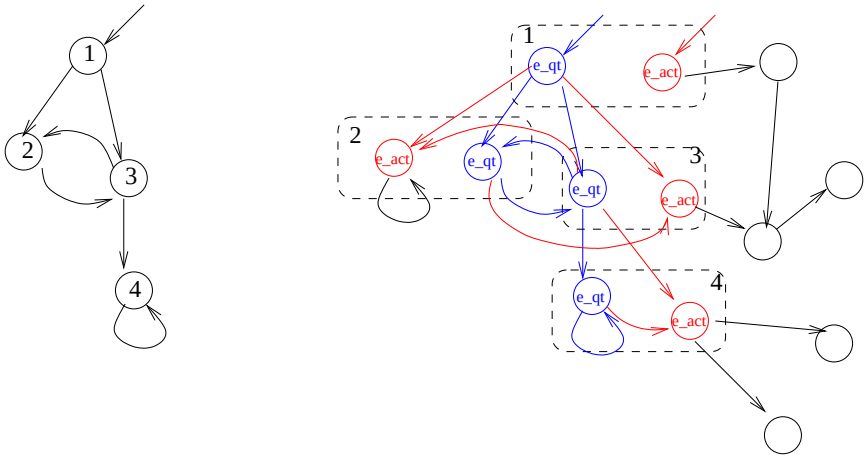


Fig. 1.2 – Frontière délimitant le sous-graphe isomorphe à $K_i$ dans $K_{i+1}$

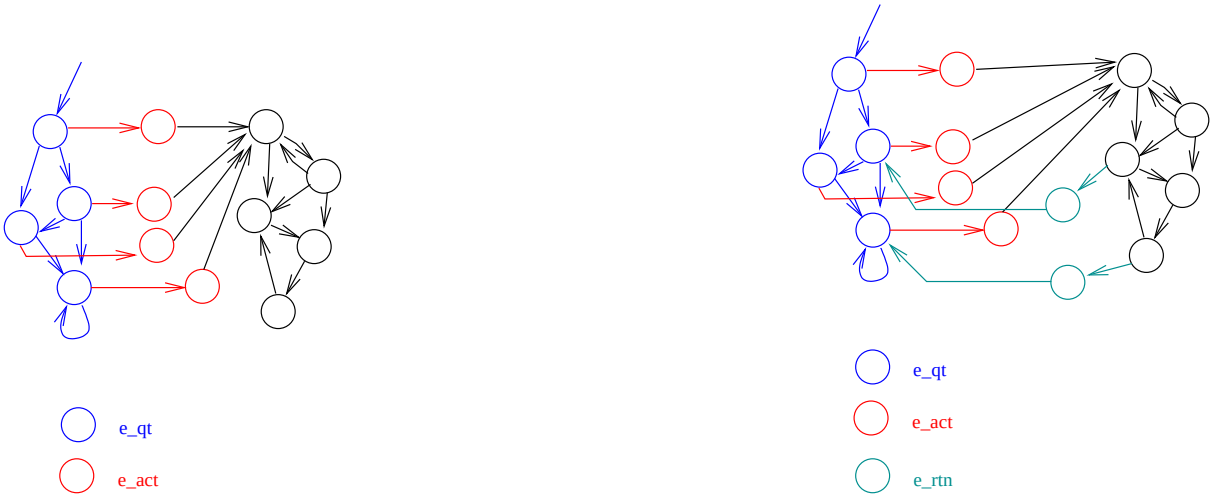


Fig. 1.3 – Différents cas de retour (de l'extension de $K_{INC}$ vers l'extension de $K_i$) dans $K_{i+1}$

hypothèses A-1 et R-1 sont vérifiées : de l'état initial étendu à $e_qt$, on reste dans $G$ tant que $e_qt$ reste vrai. Dès que l'événement est activé, on passe dans un état frontière (étiqueté par $e_act$) puis l'on se rend dans un état correspondant à l'extension de l'état initial de $K_{INC}$. Une fois dans l'ensemble des états étendants $K_{INC}$, on ne peut pas ressortir de cet ensemble. La partie droite de la figure montre une situation de terminaison différente : une fois que l'état étendant l'état initial de $K_{INC}$ est atteint, on reste dans les états étendants $K_{INC}$ tant que l'on n'atteint pas d'état étiqueté par la configuration $e_rtn$. De tels états forment la frontière de retour de $K_{INC}$ vers $K_i$.

### Résultats généraux

Si l'on ne dispose pas d'hypothèse particulière, l'ajout de nouveaux comportements à un composant existant ne permet pas de dire grand-chose quant à l'ensemble des propriétés vérifiées par le nouveau composant. Les résultats concernant la préservation de propriétés CTL entre deux composants reliées par une relation de simulation sont les suivants [72] [92] :

- Si $K_{i+1}$ simule $K_i$, et si $< K_i, s_0 \models \phi_i >$ avec $\phi_i \in ECTL$, alors $< K_{i+1}, s_0 \models \phi_i >$.
- Si $K_{i+1}$ simule $K_i$, et si $< K_{i+1}, s_0 \models \psi_{i+1} >$ avec $\psi_{i+1} \in ACTL(AP_i)$, alors $< K_i, s_0 \models \psi_{i+1} >$.

Intuitivement, le premier item signifie qu'il existe un comportement de $K_i$ validant la propriété $\phi_i$, et que ce comportement est bien présent dans $K_{i+1}$ (car $K_{i+1}$ simule $K_i$), et donc il valide la propriété $\phi_i$ dans $K_{i+1}$. La réciproque est fausse : s'il existe un comportement dans $K_{i+1}$ qui valide une propriété $\phi_i (\in ECTL)$, rien ne guarantit que ce comportement est présent dans $K_i$ (ce peut être un comportement qui a été ajouté par l'incrément). Symétriquement, le second item stipule que si tous les comportements de $K_{i+1}$ valident la propriété $\psi_{i+1}$, alors tous les comportements de $K_i$ la valident également (puisqu'ils forment un sous-ensemble des comportements de $K_{i+1}$). La réciproque est fausse : si tous les comportements de $K_i$ vérifient $\psi_i$, le comportement supplémentaire dans $K_{i+1}$ peut ne pas vérifier la propriété $\psi_i$.

### Expression de la non-regression

Les incréments que nous avons définis ne sont pas quelconques : ils induisent une frontière entre les comportements de $K_i$ et les comportements de $K_{i+1}$, et garantissent que tous les comportements de $K_i$ sont bien présents dans $K_{i+1}$. Nous tirons parti de ces spécificités pour proposer un ensemble de règles de transformations de la spécification de $K_i$ afin de dériver automatiquement une partie de la spécification de $K_{i+1}$.

Les propriétés qui sont vraies dans le composant $C_i$ sont transformées en un nouvel ensemble de propriétés vérifiées *par construction* sur le modèle $C_{i+1}$. Cette transformation s'appuie sur la notion de frontière; lors de l'évaluation de la formule $\psi_{i+1}$, elle contraint le parcours de la structure de Kripke de $C_{i+1}$ au sous-ensemble présent dans $C_i$ (il s'agit du sous-graphe, accessible à partir de l'état initial, pour lequel les états ne sont pas étiquetés par la valeur active du signal d'incrément). Le théorème suivant, présenté dans [32] puis dans [35], formalise cette transformation, incorporant la frontière d'un événement pour limiter l'évaluation de la formule.

**Théorème 1** *Soient $s \in K_i$ et $s' \in K_{i+1}$ tels que $s'$ enrichit[2] $s$ avec $e_qt$.*
*Pour toute proposition atomique $p \in AP_i$ et pour toutes formules CTL $\Phi$, $\chi$ et $\Psi$ (ayant leurs propositions atomiques dans $AP_i$),*

$$K_i, s \models \Phi \Leftrightarrow K_{i+1}, s' \models \Phi'$$

*et $\Phi'$ est obtenue par application récursive des transformations suivantes :*

*1* $\Phi = p \quad \Leftrightarrow \Phi' = p.$
*2* $\Phi = \neg\Psi \quad \Leftrightarrow \Phi' = \neg\Psi'.$

[2] la notion d'enrichissement est définie dans [35]; intuitivement, $s'$ enrichit $s$ par $e_qt$ si $s'$ (dans $K_{i+1}$) est l'équivalent de $s$ (dans $K_i$), étendu par la valeur $e_qt$. Alors $s'$ simule $s$ et le sous-arbre d'exécution engendré par $s'$ étiqueté par $e_qt$ est isomophe au sous-arbre d'exécution de s dans $K_i$.

*3* $\Phi = \Psi \vee \chi \quad \Leftrightarrow \Phi' = \Phi' \vee \chi'.$
*4* $\Phi = EX\Psi \quad \Leftrightarrow \Phi' = e_qt \wedge EX\Psi'.$
*5* $\Phi = EF\Psi \quad \Leftrightarrow \Phi' = E(e_qtU\Psi').$
*6* $\Phi = EG\Psi \quad \Leftrightarrow \Phi' = EG(e_qt \wedge \Psi').$
*7* $\Phi = E[\Psi U \chi] \quad \Leftrightarrow \Phi' = E[(e_qt \wedge \Psi')U\chi'].$
*8* $\Phi = E[\Psi W \chi] \quad \Leftrightarrow \Phi' = E[(e_qt \wedge \Psi')W\chi'].$
*9* $\Phi = AX\Psi \quad \Leftrightarrow \Phi' = e_qt \Rightarrow AX\Psi'.$
*10* $\Phi = AF\Psi \quad \Leftrightarrow \Phi' = AF(e_act \vee \Psi').$
*11* $\Phi = AG\Psi \quad \Leftrightarrow \Phi' = A[\Psi' W(e_act \wedge \Psi')].$
*12* $\Phi = A[\Psi U \chi] \quad \Leftrightarrow \Phi' = A[\Psi' U((e_act \wedge \Psi') \vee \chi')].$
*13* $\Phi = A[\Psi W \chi] \quad \Leftrightarrow \Phi' = A[\Psi' W((e_act \wedge \Psi') \vee \chi')].$

La preuve de ce théorème se fait par induction sur la structure de la formule, pour une structure $K_{i+1}$ relativement à une structure $K_i$ . $K_{i+1}$ est dérivée d'un composant $C_{i+1}$ obtenu par incrément général d'un composant $C_i$ (induisant une structure $K_i$. La preuve est donnée dans [31]. Ce théorème permet de construire automatiquement la partie de la spécification de $C_{i+1}$ correspondant à la *non-regression* par rapport à $C_i$. Des hypothèses supplémentaires quant à la nature du comportement ajouté permettent de construire des jeux de propriétés supplémentaires pour $C_{i+1}$.

**Prise en considération de la spécification (CTL) de l'incrément**

Supposons que l'incrément réponde à l'hypothèse A-1 (modélisant l'invocation préemptive de fonction). Considérons de plus que l'incrément (pris isolément) vérifie un ensemble de propriétés $\chi$ (i.e. pour toute formule $\chi_k$ de $\chi$, pour tout état initial de $s_{0_{INC}}$ de $K_{INC}$, $K_{INC}, s_{0_{INC}} \models \chi_k$).

L'intégration de $\chi$ dans la spécification de $C_{i+1}$ dépend de la connexion de $K_i$ et $K_{INC}$, plus précisément du mode de retour de $K_{INC}$ vers $K_i$ [34] :

- si $K_{INC}$ ne retourne pas vers $K_i$ (hypothèse R-1), alors les états formant la frontière de $K_i$ vers $K_{INC}$ (étiquetés par $e_act$) ont tous leurs successeurs qui vérifient $\chi_k$. Ceci est formalisé dans le théorême 2.
- si $K_{INC}$ retourne vers $K_i$ avec une valeur de retour déterminée (hypothèse R-2), alors cette valeur de retour définit une frontière de $K_{INC}$ vers $K_i$. Symétriquement, les propriétés $\chi_k$ vérifiée dans les états initiaux de $K_{INC}$ sont transformées en propriétés $\chi'_k$ vraies dans les états initiaux de $K_{INC}$ lorsque celle-ci est connectée à $K_i$ au travers d'états frontière étiquetés par $e_rtn$. Dès lors, les propriétés $\chi_k$ (vraies dans les états initiaux de $K_{INC}$) peuvent être transformées en propriétés $\chi'_k$ par un principe similaire à celui présenté dans le théorême 1, selon la frontière délimitée par les états d'étiquette $e_rtn$. Ces propriétés $\chi'_k$ sont vraies dans les états de $K_{i+1}$ étendant les états initiaux de $K_{INC}$ ; ils sont immédiatement accessibles à partir des états marquant la frontière entre $K_i$ et $K_{INC}$. Les théorêmes relatifs à ces transformations sont des variantes des théorêmes 1 et 2 ; ils sont présentés dans [34], mais afin de ne pas alourdir le propos, ils ne sont pas donnés ici.

**Théorème 2** *Soit $K_{i+1}$ la structure de Kripke obtenue par ajout de l'incrément INC sur la structure $K_i$, sous hypothèses A-1 et R-1. On a :*
$K_{INC} \models \chi_k \Rightarrow \forall s_0 \in S_{0_{i+1}}, K_{i+1}, s_0 \models \mathbf{A}(e_qt\mathbf{W}(e_act \wedge \mathbf{AX}\chi_k))$

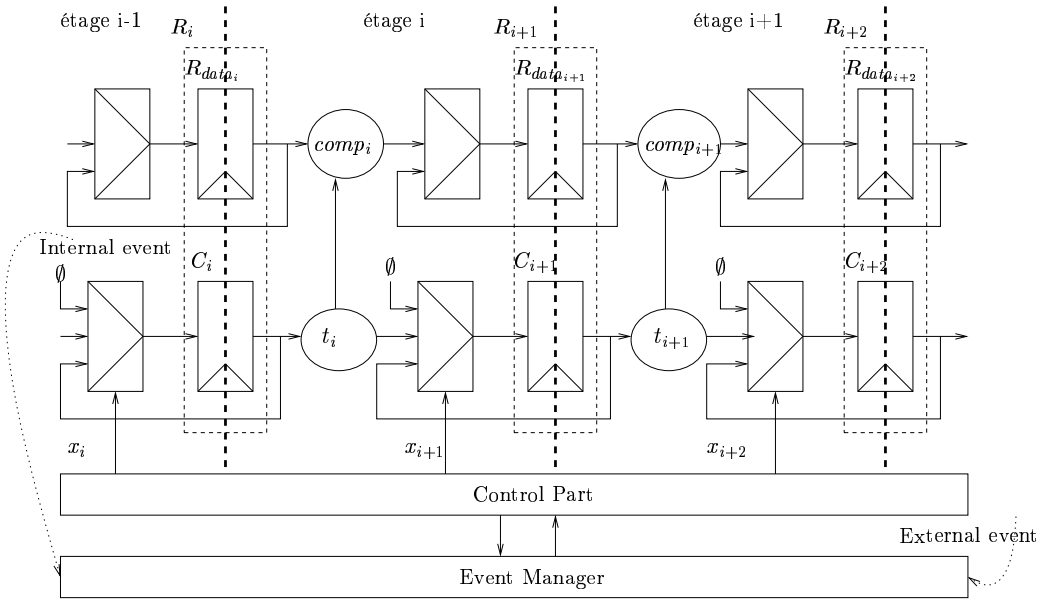


Fig. 1.4 – Architecture du composant à flux pipeline

La preuve est donnée dans [34]. Elle découle directement de la structure de $K_{i+1}$ sous hypothèse A-1 et R-1 (cf. figure 1.3, partie gauche).

## 1.5 Incréments représentant les ruptures de progression d'un pipeline

Nous nous sommes plus particulièrement intéressées aux incréments représentant des équivalences begayantes, qui correspondent à la mise en attente momentanée de partie de traitement dans un pipeline. En partant d'une architecture pipeline munie d'un mécanisme de contrôle de flux optimal, nous avons défini les incréments correspondant au *gel* d'une portion du pipeline (*stall*), ainsi qu'à la *destruction* du résultat du traitement d'un étage du pipeline (*kill*). Ce sont là les opérations de base de gestion du contrôle de flux : dans le pipeline d'un processeur, le stall correspond par exemple à l'attente d'une donnée lors d'un *(cache miss)* ; le kill correspond à la levée d'une exception.

L'architecture générale d'un composant à flux pipeline est décrite sur la figure 1.4. L'automate contrôlant la progression du pipeline (implanté dans la partie contrôle sur la figure 1.4) produit dans chaque état, une commande pour chaque multiplexeur séparant deux étages consécutifs du pipeline. $x_j$, la commande du multiplexeur de rang $j$, peut prendre trois valeurs :

- $x_j = 1$ : progression du flot d'information à l'étage $j$. Le résultat du traitement de l'étage $j-1$ est inséré dans le registre $R_j$.
- $x_j = R$ : gel du flot d'information à l'étage $j$. Il y a réécriture du contenu de $R_j$ dans $R_j$.
- $x_j = 0$ : destruction d'une opération : Le résultat du traitement de l'étage $j-1$ n'est pas propagé, mais remplacé par une opération vide (NOP) dans $R_j$.

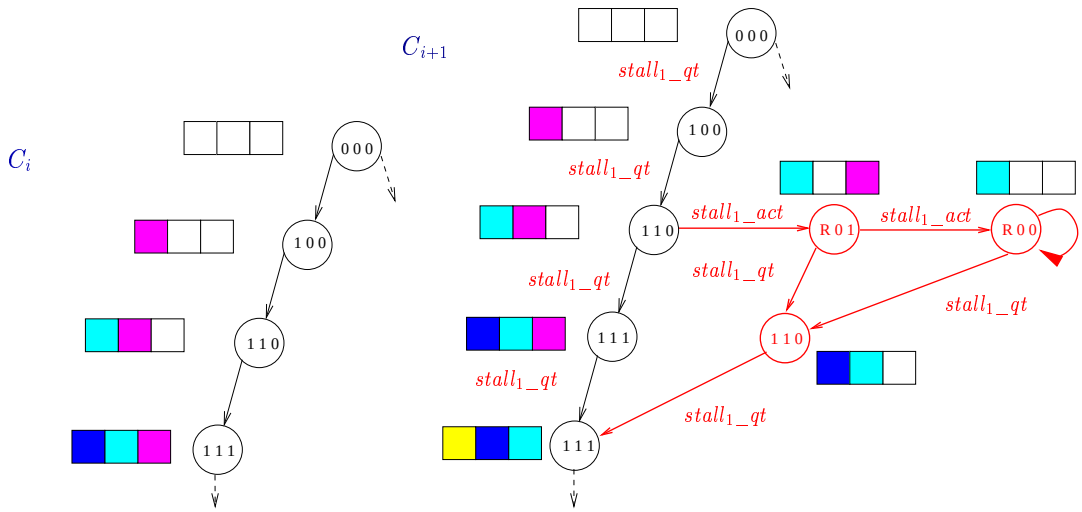


FIG. 1.5 – Flux optimal d'un pipeline de trois étages, et introduction d'un *stall*.

**Définition 4** *Le* mot d'état *d'un pipeline de longueur n est le vecteur regroupant les n commandes des multiplexeurs, noté* $V_0^{n-1} = \{v_0, v_1, ...v_{n-1}\}$ *tel que* $\forall i : v_i \in \{0, 1, R\}$. *Le préfixe de rang l est le sous-mot*$V_0^{l-1}$ *et le suffixe de rang l est le sous-mot* $V_l^{n-1}$

Le flux optimal d'un pipeline de $n$ étages a été défini à partir de ces éléments. Une représentation schématique d'une partie d'un pipeline de trois étages est donnée sur la figure 1.5, à gauche. Les états de l'automate sont étiquetés par le mot d'état correspondant. Afin de visualiser l'effet du mot d'état sur le chemin de données du pipeline, nous avons introduit sur le schéma un tableau de couleurs attaché à chaque état, qui représente la progression des différents traitements.

### 1.5.1 Les incréments *stall* et *kill*

À partir de ce flux optimal, nous avons défini l'incrément $stall_k$ représentant le gel de la progression du pipeline à partir d'un étage $k$. Il s'agit d'un incrément général (répondant à la définition de la section 1.2). Nous avons montré que, par construction, il existe une relation de bisimulation faible entre les préfixes de rang $k$ des mots d'états de $K_i$ et $K_{i+1}$. Plus précisément, chaque séquence $\sigma_{i+1}$ de mots d'états de $K_{i+1}$ peut être mise en relation avec une séquence $\sigma_i$ de mots d'états de $K_i$ pour laquelle :

- il existe une équivalence bégayante pour les préfixes de rang $k$.
- les suffixes de rang $k$ progressent identiquement.

Ces relations sont illustrées sur la figure 1.5. La partie droite de la figure montre la machine de Moore du pipeline lors de l'adjonction d'un incrément *stall* sur l'étage 1, intervenant sur l'état "1,1,0". Si l'on se focalise sur le préfixe de longueur 1 (i.e. sur le premier étage du pipeline) , les états "1,1,0", "R,0,1" et "R,0,0" de $C_{i+1}$ sont en relation avec l'état "1,1,0" de $C_i$ ; l'état "1,1,0" rouge de $C_{i+1}$ est en relation avec l'état "1,1,1" de $C_i$. Leurs successeurs sont identiques.

L'incrément *kill* appliqué à un étage $k$ consiste à substituer dans $R_k$ une opération vide au résultat de l'étage $k-1$. Par la suite, cette opération vide se propage au rythme d'écoulement du pipeline. Nous retrouvons des relations similaires entre $K_i$ et $K_{i+1}$ : équivalence bégayante pour le préfixe de rang $k-1$ (et non plus $k$), et progression du suffixe de rang $k+1$. Par contre, le traitement associé au rang $k$ est définitivement perdu.

### 1.5.2 Conséquences sur les transformations de propriétés CTL

Nous nous intéressons à la spécification CTL attachée à un composant en cours de conception, dans le cas particulier d'une architecture pipeline construite, à partir d'un flot de progression optimale, par adjonction d'incréments *stall* et *kill*.

Nous présentons deux classes de résultats, selon que les propriétés portent sur le fonctionnement macroscopique du pipeline, ou bien sur le fonctionnement interne du pipeline (les propriétés portent sur les relations entre données présentes sur différents étages du pipeline). Ces résultats sont détaillés dans [33].

**Fonctionnement macroscopique du pipeline.** La spécification du fonctionnement macroscopique du pipeline établit des relations de causalité entre des données injectées à l'étage 0 et le résultat produit à l'étage $n-1$.

- Dans le cas d'une composition d'incréments *stall* intervenant sur différents étages du pipeline, toute formule positive sur CTL\X, pour laquelle les propositions atomiques portent sur les données à l'entrée ou à la sortie du pipeline sont préservées.
- Dans le cas d'un incrément *kill*, nous proposons une transformation incorporant dans la formule la perte du résultat au cours de sa progression dans le pipeline.

**Fonctionnement interne du pipeline.** Dans ce cas, nous établissons des relations de causalité entre des événements intervenant à l'intérieur du pipeline, dans le cas de combinaison d'incréments *stall*. Soit $\phi_k$ (resp. $\phi_l$) une proposition atomique (ou sa négation), associée à l'étage $k$ (resp. $l$) du pipeline. Nous présentons le résultat suivant (détaillé dans [31]) :
Toutes les formules de CTL\X combinant des propositions atomiques $\phi_k$ et $\phi_l$ *portant sur des étages antérieurs à l'étage sur lequel porte le gel* sont préservées.

## 1.6 Application à la conception de convertisseurs de protocoles

La démarche de conception présentée ci-dessus, ainsi que les transformations automatiques de propriétés qui en découlent ont été appliquées lors de la conception de convertisseurs de protocoles de communication dans les SoC. Nous décrivons cette étude de cas dans la présente section.

### 1.6.1 Convertisseurs de protocoles entre VCI et les bus PI et AMBA

La conception de SoC est de plus en plus une activité d'assemblage de composants pré-existants, conçus par d'autres équipes ou proposés par divers fournisseurs. Chaque composant est muni de son interface propre et la connexion de composants ne respectant pas les mêmes protocoles de communication nécessite l'adjonction d'un composant intermédiaire réalisant la conversion de protocole. Il s'agit de convertir les données entre les formats des deux protocoles, mais surtout d'adapter les flux de communication des protocoles différents.

Les protocoles de communication présentent en général plusieurs modes de fonctionnement dédiés à des architectures particulières (par exemple le protocole AMBA [10] distingue les modes de fonctionnement associés au bus système rapide, et au bus périphérique plus lent). De plus, tous les composants connectés n'implantent pas forcément la totalité des traitements admissibles par un protocole. Ces caractéristiques permettent la conception incrémentale de convertisseurs de protocoles : le fonctionnement nominal (transport des informations sans rupture de flux) est le coeur du composant. Différents événements extérieurs peuvent venir perturber ce flux ; l'ensemble des événements perturbateurs reçus par un composant dépend de la complexité de l'environnement dans lequel il est plongé.

Nous avons utilisé la méthode incrémentale pour concevoir une famille de wrapper initiateur réalisant la conversion entre les protocoles VCI [100] et PI [101]. Une démarche très similaire a permis de réaliser d'autres convertisseurs entre les protocoles VCI et AMBA/ASB [10] (i.e. le protocole AMBA pour un bus système).

Le convertisseur de protocole est inséré entre le composant à interface VCI, et le bus PI, support physique du transfert. La terminologie PI distingue le composant initiateur d'une requête de transfert (maître) du composant répondant à cette requête (esclave). Selon le type de transfert qu'il doit réaliser, un composant à interface VCI peut disposer d'un convertisseur maître et d'un convertisseur esclave ou bien uniquement de l'un des deux. Le protocole VCI établissant un transfert d'information entre un initiateur et une cible est basé sur l'échange d'un ensemble de couples de messages <*requête,réponse*>. Le bus PI quant à lui est un bus synchrone, à transfert pipeliné (au même cycle, le bus véhicule l'adresse du transfert $i+1$ ainsi que la donnée et le statut associés au transfert $i$). Un exemple d'architecture combinant des composants à interface VCI communiquant au travers du bus PI est illustré sur la figure 1.6.

L'initiateur VCI envoie une requête au wrapper maître (1), lequel demande l'accès exclusif au bus à l'arbitre de bus (2). Lorsque l'arbitre de bus accorde l'accès au wrapper maître (3), celui-ci transfère les informations vers le wrapper esclave (4,5). Le wrapper esclave traduit les informations reçues du bus PI en cellules VCI et les transmet à la cible VCI (6). Celle-ci élabore la réponse, la transmet au wrapper esclave (7), qui la convertit au format PI et l'envoie sur le bus à destination du wrapper maître (8,9). A réception de cette réponse, le wrapper maître traduit la réponse au format VCI et la transmet à l'initiateur VCI (10). Dans la plupart des cas, le wrapper esclave *anticipe* la réponse de la cible VCI, afin de répondre au wrapper maître en un cycle [102].

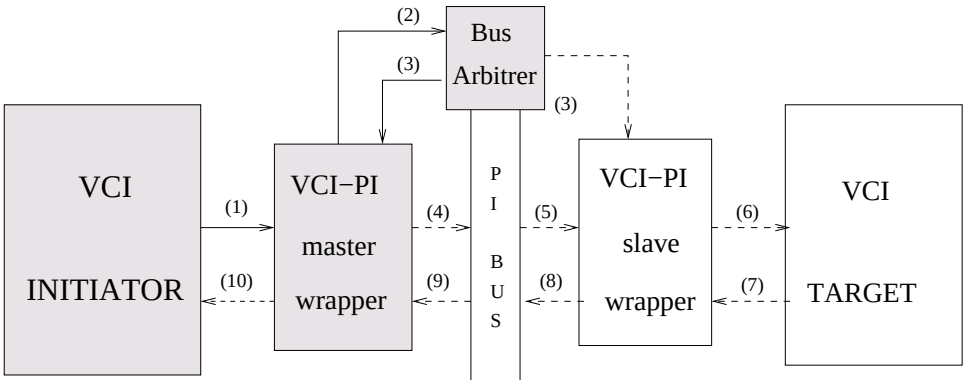


FIG. 1.6 – Communication d'un initiateur et une cible VCI empruntant le support physique (et le protocole) du bus PI.

| Type of event considered | Initiator is alway *ready*<br>cmd_val=1 ; rsp_ack=1 | Initiator may impose *wait states*<br>cmd_val={0,1} ; rsp_ack = {0,1} |
|---|---|---|
| Target is always *ready*<br>pi_rsp=RDY | **A** | **A'** |
| Target may impose<br>*wait states*<br>pi_rsp={RDY,WAIT} | **B** | **B'** |
| Target may impose *retract*<br>pi_rsp={RDY,WAIT,RTR} | **C** | **C'** |

FIG. 1.7 – Hiérarchie des convertisseurs VCI-PI, du plus simple **A** au plus complexe **C'**. Chaque flêche correspond à un incrément. Pour chaque incrément, le domaine de définition d'un ou plusieurs signaux d'interface est étendu.

### 1.6.2 Conception incrémentale du wrapper maître VCI-PI

Le convertisseur de protocole doit convertir les requêtes au format VCI en des transferts au format PI, puis interpréter les réponses au format PI en des réponses au format VCI (cf. Fig. 1.8). Ces différents flux d'information ne progressent pas au même rythme. La difficulté majeure dans la conception du convertisseur est la réalisation d'un mécanisme de contrôle de flux garantissant la progression à vitesse maximale des différents flux entrants et sortants, en considérant différentes causes de retard ou de rupture d'un des flux.

La conception suivant la démarche incrémentale nous a permis de définir un ensemble de six composants, réagissant à diverses combinaisons d'événements extérieurs perturbant un des flux. La hiérarchie des convertisseurs est donnée sur la figure 1.7. Le modèle le plus simple (A) suppose que l'initiateur VCI et la cible PI répondent toujours favorablement en un cycle. Le plus complexe (C'), supporte les attentes et les réinitialisations de transfert provenant du bus PI ainsi que les attentes provenant de l'initiateur VCI.

Le convertisseur le plus simple (**A**) est organisé en un pipeline de trois étages réalisant en un même cycle :

- étage 1 : réception sur l'interface VCI d'une requête $k$.
- étage 2 : émission sur l'interface PI de l'adresse correspondant au transfert de la requête $k-1$,
- étage 3 : réception sur l'interface PI des données et status du transfert $k-2$ (A transférer sur l'interface VCI)

Les modèles plus complexes (A',B,B',C,C') sur la figure 1.7, traitent en outre les ruptures de régularité des flux perturbant la progression du pipeline. Les différentes causes de rupture de flux sont les suivantes :

- La non disponibilité de l'initiateur VCI, qui n'est pas prêt à fournir la requête $k$ ou à recevoir la réponse $k-2$. Ce cas est représenté sur la colonne de droite de la figure (modèles A', B', C').
- La non disponibilité momentanée de l'esclave PI (éventuellement induit par la non disponibilité de la cible VCI), qui retarde la fourniture de la donnée ou du status PI (modèles B et B').
- La non-disponibilité de l'esclave PI qui clôt le transfert PI (le bus est relâché). Le transfert devra être repris ultérieurement, après que le maître ait de nouveau pu accéder au bus (modèles C et C'). Ce type de rupture induit une rupture d'atomicité du transfert.

L'architecture du chemin de données du wrapper présentée sur la figure 1.8 reflête le processus de conception incrémentale. Les additions successives sont représentées dans des zones de couleurs différentes. La zone *PI wait* correspond au gel des étages 1 et 2 du pipeline tant que la données ou le statut de la requête $k-2$ ne sont pas fournis en étage 3 (modèle B et B'). La zone *PI retract* correspond au traitement de la rupture d'atomicité : demande du bus puis l'émission correcte des données des étages 1 et 2 (modèles C et C'). La zone *Initiator wait* correspond à l'attente induite par la non-disponibilité de l'initiateur VCI (modèles A',B',C').

L'automate de contrôle du composant C' est obtenu en partant du modèle A par ajouts successifs d'incréments. La figure 1.9 présente la machine de Moore du wrapper B'. La machine initiale (A) est présentée en blanc, l'incrément $PI_{wait}$ (de type *stall* respectant l'hypothèse K-1) est représenté en jaune, sa composition au modèle A produit le modèle B. Sur ce dernier, on ajoute l'incrément $VCI_{wait}$ (de type *stall*, respectant l'hypothèse K-1), représenté en violet, produisant le modèle B'. La figure 1.10 montre le modèle C' : c'est le modèle B' auquel on a ajouté l'incrément $PI_{retract}$ (de type *appel fonctionnel*, respectant les hypothèses A-1 et R-2).

### 1.6.3 Vérification incrémentale du wrapper maître

La vérification du wrapper maître a été réalisée en immersion dans la plate-forme décrite sur la figure 1.6, implantée en Verilog synchrone, et vérifiée avec l'outil de model-checking VIS. Environ 80 propriétés CTL ont été écrites pour la plate-forme B. Elles sont classées en différentes catégories :

- description des comportements attendus du wrapper maître B (dans un environnement B),
- description des comportements attendus d'un esclave B,
- description du transfert d'un paquet VCI complet enchaînant le transfert de plusieurs mots.

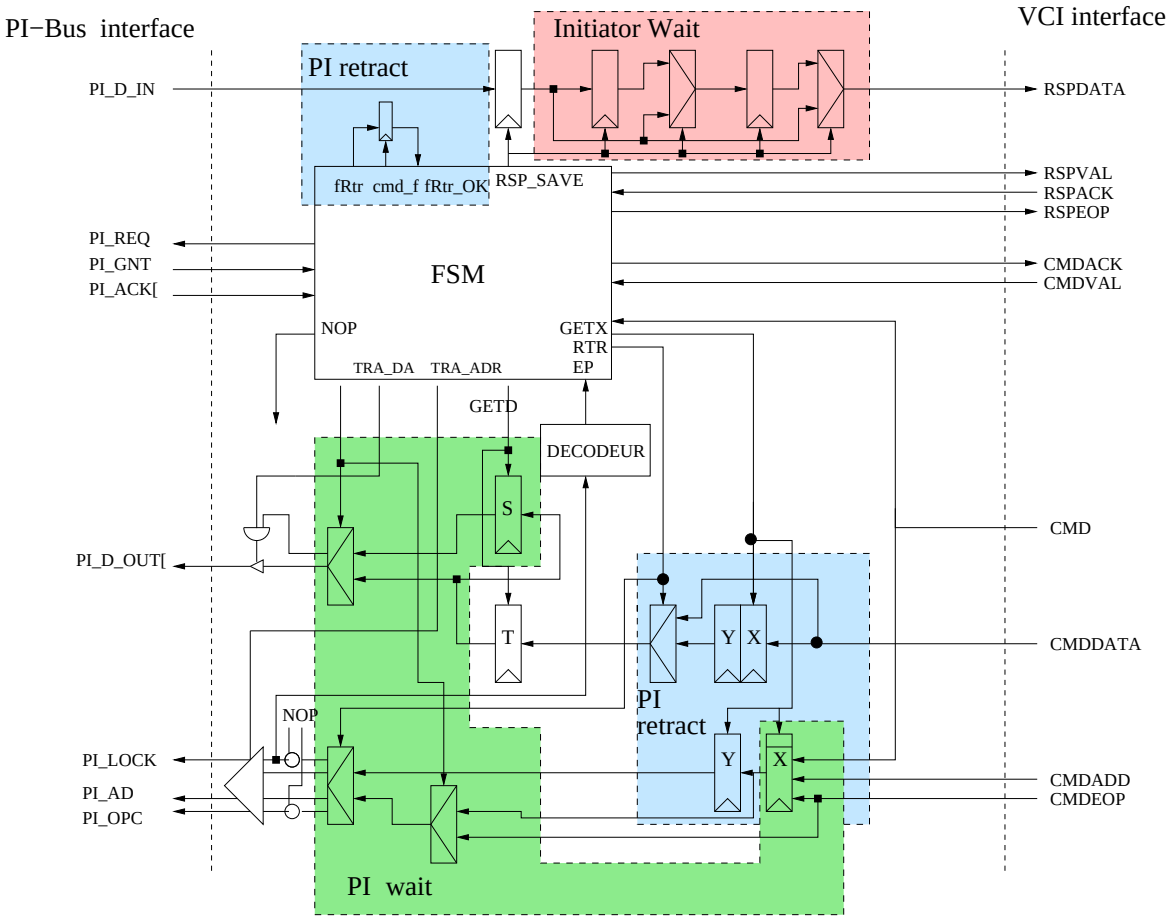


FIG. 1.8 – Architecture du convertisseur maître C'.

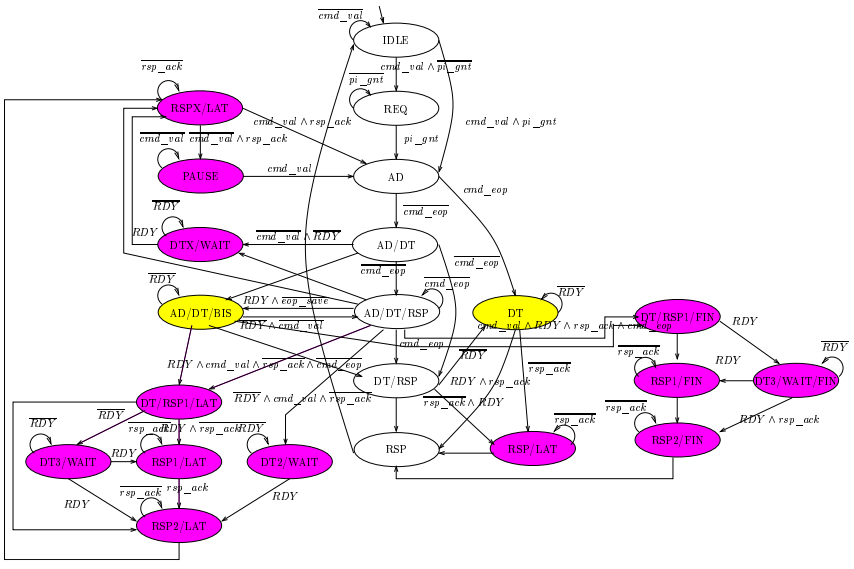


FIG. 1.9 – Machine de Moore du wrapper B' mettant en évidence deux incréments de type *stall*.

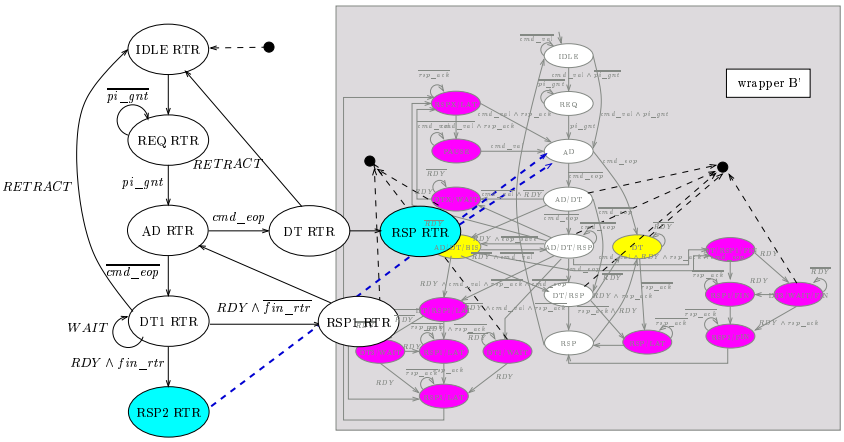


FIG. 1.10 – Machine de Moore du wrapper C' mettant en évidence l'incrément de type *appel de fonction*.

Ces propriétés sont décrites sous hypothèses globales d'équité des composants annexes : l'initiateur VCI finira toujours par envoyer la suite (et la fin) du paquet à transmettre, la cible VCI finira toujours par répondre, l'arbitre de bus finira toujours par autoriser le wrapper maître à accéder au bus.

Cet ensemble forme la spécification de la plate-forme B : lorsque les hypothèses d'équité sont respectées, les comportements de la plate-forme B satisfont chaque propriété de cet ensemble. La spécification de la plate-forme B n'est pas une spécification pour la plate-forme B' : certaines propriétés vérifiées sur la plate-forme B ne le sont pas sur la plate-forme B'. Ce sont celles construites avec le quantificateur A, avec l'opérateur X, ou à partir de conjonctions de propositions atomiques relatives à différents étages du pipeline.

Non régression : l'ensemble des propriétés vérifiées sur la plate-forme B a été automatiquement transformé (d'après le théorème 1, sur l'événement $VCI_{wait}$) en un ensemble de propriétés à vérifier pour la plate-forme B'.

Equivalences begayantes : l'incrément $VCI_{wait}$ induit une équivalence bégayante entre les états de B et de B'. Les propriétés de B ont été assouplies suivant les règles associées au stall, pour être adjointes à la spécification de la plate-forme B'.

Cet ensemble de propriétés CTL forme la spécification de la plate forme B'. Par construction, cet ensemble de propriétés est vérifié sur la plate-forme B'. En pratique, nous avons vérifié que c'était bien le cas.

Une démarche similaire est appliquée de B'à C'. Des propriétés supplémentaires ont été ajoutées manuellement pour spécifier le comportement du wrapper C' lors de la survenue d'un événement $PI_{retract}$.

Avec l'outil VIS, les propriétés propres à chaque composant et les propriétés globales ont pu être vérifiées sans difficulté sur une plate-forme B'composée d'un initiateur et d'une cible VCI. A partir de deux initiateurs B', le temps de vérification devient prohibitif.

## 1.7 Conclusion

Dans ce chapitre, nous avons présenté une méthode de conception incrémentale qui permet d'ajouter des fonctionalités à un composant, en réaction à une extension de l'ensemble des sollicitations auxquelles il est soumis. Les composants considérés sont décrits sous la forme de machines de Moore synchrones, ce qui correspond à une description précise au cycle près et au bit près du composant à réaliser. Les règles d'ajout que nous avons définies sont strictes (plus strictes que d'autres, proposées par [109] notamment), mais elles garantissent une forme de compatibilité entre fonctionalités que n'offrent pas des ajouts plus libres. Par construction, les fonctionalités ajoutées n'engendrent pas de régression quant aux fonctionalités initialement présentes. De surcroît, les règles d'incrément que nous avons définies peuvent être incorporées dans la spécification CTL du composant lors des différentes étapes de conception. Ceci permet de faire évoluer conjointement la spécification et le composant conformément à un incrément donné. En pratique, cette méthode de conception a été utilisée pour la conception et la vérification de composants de régulation de contrôle de flux.

Ces travaux renforcent le lien entre la spécification d'un composant et son implémentation, afin de promouvoir sa vérification par Model-Checking lors des différentes étapes de sa conception. Par contre, ils ne permettent pas de contourner le problème d'explosion combinatoire rencontré lors de la vérification de ces systèmes, même lorsque ceux-ci sont de taille modeste. Les propriétés locales à chaque composant sont vérifiées par construction, ou peuvent être vérifiées par Model-Checking, par contre les propriétés globales, qui doivent être vérifiées sur le système complet ne peuvent être vérifiées simplement. Le chapitre suivant présente une perspective de ce travail utilisant la spécification des composants pour améliorer la vérification par Model-Checking de ces systèmes.

# Chapitre 2

# Abstraction de composants à partir de leur spécification

La démarche précédente permet de propager les transformations subies par un composant dans sa spécification. Au fur et à mesure de la complexification du système, la spécification se complique (le nombre de formules augmente, la complexité de chaque formule augmente). La vérification du composant dans son environnement réel (représenté par d'autres composants, décrits au niveau CABA) est limitée par le phénomène d'explosion combinatoire du nombre d'états du système à analyser. En pratique, il est possible de vérifier la plate-forme décrivant un système VCI-PI pour deux maîtres et un esclave avec l'outil VIS. Pour un plus grand nombre de maître ou d'esclave, le model-checker ne peut aboutir.

Dans le contexte de vérification par model-checking, le problème d'explosion combinatoire a été contourné de différentes façons[1]

- L'abstraction de composant (puis intégration dans un produit standard).
- La vérification compositionnelle (ou raisonnement assume-guaranty).
- Le raffinement d'abstraction (ou CEGAR).

**L'abstraction de composant** (puis intégration dans un produit standard). Une première idée consiste à *retirer l'information inutile* à la vérification du composant concret. Considérons un composant concret et une propriété CTL à vérifier sur ce composant. On souhaite effectuer une abstraction de l'arbre d'exécution du composant concret préservant les propositions atomiques qui apparaissent dans la formule. L'arbre d'exécution abstrait contiendra moins d'informations, mais suffisamment pour décider de la validité de la formule. Ceci revient à construire l'automate quotient pour une relation d'équivalence particulière : la plus grande (auto)-bisimulation du composant concret pour l'ensemble des propositions atomiques de la formule CTL à vérifier [16]. Divers algorithmes de calcul d'autobisimulation ont été proposés dans la littérature. L'algorithme proposé par A. Bouajjani et al. [27] est implanté dans l'outil CAESAR/ALDEBARAN [30]. L'algorithme de Yannakakis et al. [127] considère en sus des informations temporelles. L'algorithme de Lin et al. [91] a la particularité d'être entièrement symbolique. Si cette approche fonctionne bien pour la réduction d'un unique composant, elle perd de son intérêt pour la vé-

[1] Je laisse volontairement de côté le *Bounded Model Checking* basé sur des techniques SAT, et le *property monitoring*, qui permettent de s'assurer *partiellement* de la validité d'une formule de la spécification (mais contournent de ce fait les problèmes d'explosion combinatoire).

rification de propriétés globales comprenant plusieurs composants, tout particulièrement dans le contexte des machines de Moore synchrones : afin de garantir un produit synchronisé exact, *tous les signaux d'interface de chaque composant doivent être rendus visibles*. Dans le cas des Machines de Moore, chaque signal de sortie est positionné dans chaque état de la Machine de Moore. De ce fait, la plus grande autobisimulation n'apporte pas (ou très peu) de réduction par rapport à la Machine de Moore initiale. Les essais expérimentaux que nous avons faits avec l'outil VIS, utilisant l'algorithme de réduction de Lin [91] l'ont mis en évidence. La réduction des composants puis le Model-Checking du système réduit était contre-performant par rapport au Model-Checking appliqué au système sans réduction (et utilisant une relation de transition partitionnée)[2].

**La vérification compositionnelle** (ou raisonnement assume-guaranty). Le raisonnement assume-guaranty permet de décomposer la vérification d'une propriété globale en une suite de sous-preuves locales. Idéalement, pour chaque sous-preuve, le nombre de composants impliqués est réduit, et la vérification de cette sous-preuve par model-checking évite le problème d'explosion combinatoire [80]. La difficulté consiste à trouver le "bon découpage", d'une preuve en sous-preuves (quelles sont les hypothèses et les buts de chaque sous-preuve), et des associations de composants à réaliser pour chaque sous-preuve. Cette approche permet effectivement de vérifier des systèmes modulaires, composés de nombreux modules de faible complexité. Par exemple, T. Henzinger et al. ont appliqué cette méthode avec succès pour vérifier un processeur de traitement du signal ([79]). Dans certains cas, une abstraction d'un composant peut être utilisée à la place du composant concret dans une sous-preuve (on se limite alors à des propriétés globales exprimées dans le fragment universel de la logique CTL).

**Le raffinement d'abstraction** (ou CEGAR). Le raffinement d'abstraction calcule l'abstraction la plus grossière permettant de décider si une propriété est vérifiée par un composant ou non. Cette méthode procède itérativement : étant donnée une abstraction du système, cette abstraction contient-elle suffisamment d'information pour que l'on puisse déterminer la valeur de vérité de la propriété ou non ? Si c'est bien le cas, on peut vérifier la propriété ; sinon, il faut raffiner l'abstraction, en lui injectant un surplus d'information (issu du comportement réel), puis renouveler ce traitement. Cette approche, essentiellement utilisée pour la vérification de programmes séquentiels ou concurrents (programmes C ou Java multithreads), est implantée dans les outils BLAST [78] et Verisoft [8]. Elle a longtemps été limitée à la vérification de propriétés ACTL. La définition d'une abstraction combinant une sur-approximation et une sous-approximation des comportements du programme a permis l'extension de cette démarche pour toute propriété CTL [74][75].

La combinaison de ces différentes techniques est actuellement en plein essor et semble un axe prometteur pour repousser les limites du model-checking. Dans ce contexte, nous proposons d'utiliser une partie de la spécification de chaque composant comme une première abstraction dudit composant, à intégrer dans un processus de verification CEGAR. Puis d'incorporer de nouvelles propriétés lors des étapes de raffinement.

L'utilisation d'une abstraction d'un composant, plutôt que du composant lui même permet

[2] L'algorithme de Lin ne permet pas de réduire chaque composant indépendemment des autres, ceci peut expliquer les contre-performances de cette approche.

de masquer certaines variables du système, qui n'ont pas d'influence pour une propriété globale donnée. De ce fait, la représentation symbolique du système abstrait est plus concise que la représentation du système concret. On peut alors utiliser l'outil de model-checking classique pour vérifier des propriétés sur le système abstrait, et en déduire la validité de propriétés sur le système concret.

## 2.1 Vue générale de la démarche proposée

Nous proposons d'utiliser un sous-ensemble des formules logiques de la spécification de chaque composant pour construire un modèle abstrait de chaque composant. Ce modèle plus concis *remplace* le modèle concret sur la plate-forme de vérification, lorsque l'on cherche à vérifier des propriétés globales. Ce modèle abstrait est une sur-approximation de l'ensemble des comportements du composant concret (au sens où, le modèle abstrait *simule* le modèle concret). De ce fait, il peut être utilisé pour la vérification de propriétés globales dans le fragment ACTL.

Soit $\psi$ une propriété globale, exprimée dans la logique ACTL. Deux cas sont à considérer :
- Si $\psi$ est vérifiée sur le produit des abstractions des composants du système, alors $\psi$ est vérifiée pour le produit des composants concrets.
- Si $\psi$ n'est pas vérifiée sur le produit des composants abstraits, cela peut provenir de deux causes distinctes : ou bien $\psi$ n'est pas vérifiée sur le produit des composants concrets ; ou bien $\psi$ est bien vérifiée sur le produit des composants concrets, mais l'abstraction d'un (ou plusieurs) composant(s) est trop imprécise, et contient des comportements invalidant la propriété, qui ne font pas partie du modèle concret.

En effet, le modèle abstrait que l'on construit à partir d'un sous-ensemble de sa spécification peut être trop imprécis. Cela survient si le sous-ensemble de formules choisi pour réaliser l'abstraction ne capture pas finement l'ensemble des comportements pertinents pour la vérification de la propriété globale. Dans ce cas, le résultat de vérification de la propriété globale est la valeur "faux", et l'examen du contre-exemple permet de raffiner l'abstraction, pour éliminer ce comportement non pertinent.

Le schéma de principe du processus de vérification est donné sur la figure 2.1. Nous nous plaçons toujours dans l'hypothèse des systèmes décrits sous la forme de machines de Moore synchrones. Nous excluons les signaux de mode `inout`, et les signaux multi-source. Notre approche utilise différentes briques de base de l'outil VIS. A terme, nous souhaitons l'intégrer totalement dans l'outil VIS.

Cette démarche est inspirée de celle proposée par Xie et Browne dans [125]. Ces auteurs ont proposé un algorithme d'abstraction de composants à partir de leur spécification décrite dans la logique LTL, et ont intégré une boucle de vérification par raffinement d'abstraction dans l'outil COSPAN [76]. Récemment, M. Schickel et H. Eveking dans [94] synthétisent des composants "abstraits" à partir de formules du sous-ensemble simple de PSL, afin d'accélérer la simulation des systèmes matériels et de détecter préventivement les éventuelles incompatibilités des composants à assembler. Par ailleurs, dans [108] Peng et Tahar proposent de modéliser *l'environnement* d'un composant donné par un composant abstrait construit à partir d'une formule ACTL. Ils utilisent ce composant abstrait pour effectuer des preuves de type assume-guaranty, avec l'outil VIS. Tout récemment, D. Kroening *et al.* [84] ont proposé un outil de vérification CEGAR pour la

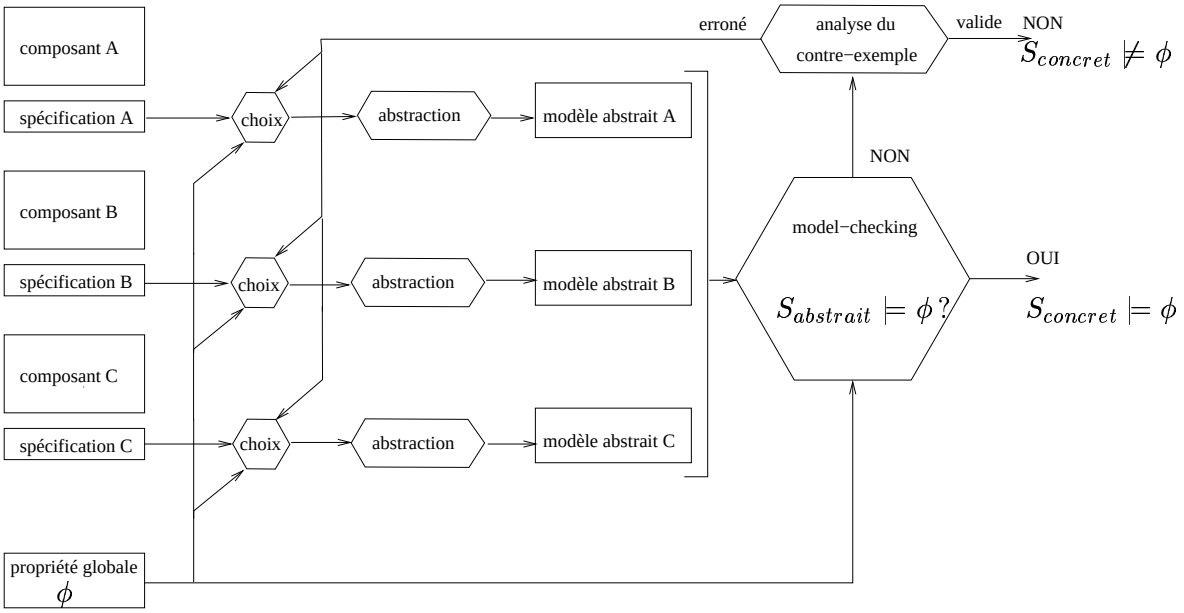


Fig. 2.1 – Vérification par raffinement d'abstraction à partir de la spécification de chaque composant.

vérification de systèmes décrits en Verilog. S'ils définissent des abstractions particulières pour représenter des mots de bits, ils ne prennent pas en compte l'aspect modulaire du programme Verilog, et ne considèrent pas la spécification de ses composants pour guider la vérification.

Enfin, de nombreux travaux concernant la synthèse de moniteurs ont été publiés récemment [11], [25], [26], [98], [28]. Ils visent essentiellement la synthèse de moniteurs à intégrer sur une plate-forme de simulation (et non pas dans un environnement de vérification), et s'intéressent au sous-ensemble simple de PSL ou SEREs (puisque c'est pour de la simulation)...

## 2.2 Algorithme d'abstraction

Nous cherchons à construire un composant qui satisfasse à un ensemble de propriétés de la spécification, tout en étant moins précis que le composant initial. Il s'agit d'un problème de *synthèse de composant*. Le composant synthétisé doit :

- être un modèle pour chaque propriété dont il est issu (et ne pas être un modèle pour la négation de chacune de ces propriétés).
- simuler tout composant "plus concret" qui satisfait cet ensemble de propriété.

Le modèle est synthétisé à des fins de vérification ; en particulier, l'imprécision peut se traduire sous forme d'indéterminisme[3].

La construction du *tableau* d'une propriété répond à notre problématique [93]. Un algorithme de construction est proposé dans [72]. La complexité de cet algorithme de construction est po-

[3]Une implantation physique du composant nécessite la résolution de cet indéterminisme, par l'introduction de variables d'entrées substituées aux variables indéterminées.

lynomiale pour ACTL et exponentielle pour CTL et pour LTL. Nous avons suivi la démarche alternative proposée par K. Schneider dans [113] pour la construction d'un automate de Büchi en un temps *linéaire avec la taille de la formule*, et produisant une représentation *symbolique* de l'automate de Büchi, également de taille linéaire. Nous proposons dans [36] et [31] un algorithme de synthèse linéaire en temps pour CTL\X (l'automate construit peut être de taille exponentielle, un codage symbolique –non encore défini– pourra réduire cette complexité).

## 2.3 Premiers résultats expérimentaux

Nous avons expérimenté cette approche sur le système composé des convertisseurs VCI-PI (cf. figure 1.6. Etant donné un système complet, composé de plusieurs éléments (convertisseurs maîtres et esclaves, initiateurs et cibles VCI, bus et arbitre de bus PI), et une propriété globale (englobant plusieurs composants), nous avons appliqué le processus de raffinement d'abstraction pour prouver la propriété globale sur le système abstrait.

Nous décrivons ci-dessous les trois propriétés globales étudiées (dans le cas de $n$ initiateurs et $m$ cibles) :

**P1 :** $\mathbf{AG}(\text{initiator[i]_state} = \text{TRANS} \Rightarrow \mathbf{AF}\ (\text{bus_arbiter_signal_gnt[i]} = 1))$

**P2 :** $\mathbf{AG}(\text{initiator[i]_state} = \text{TRANS} \Rightarrow \mathbf{AF}\ (\text{master_wrapper[i]_state} = \text{WRITE}))$

**P3 :** $\mathbf{AG}(\text{initiator[i]_state} = \text{TRANS} \Rightarrow \mathbf{AF}\ (\text{target[j]_signal_rsp} = 1))$

Bien que les propositions atomiques de chacune de ces propriétés ne soient relatives qu'à deux composants, les composants n'apparaissant pas explicitement dans la propriété interviennent implicitement.

L'analyse du contre-exemple et le choix des propriétés à ajouter lors des étapes de raffinement sont déterminés manuellement. La vérification de P1 a nécessité 3 étapes de raffinement, avec un total de 6 propriété de la spécification des composants initiateur[i], convertisseur maître[i] et arbitre de bus. La vérification de P2 a été réalisée en 2 itérations et a nécessité 4 formules. La vérification de P3 a éte effectuée en 4 itérations et a nécessité 10 formules.

Le tableau 2.1 compare les performances obtenues pour la vérification de ces trois propriétés sur le système concret et sur le système abstrait (à l'issue du processus de raffinement). La nomenclature de chaque plate-forme est X M - Y S (C/A). Elle désigne une plate-forme comprenant X maitres et Y, esclaves PI (donc X initiateurs et Y cibles VCI) ; la lettre C indique que les composants sont concrets et la lettre A que les composants sont abstraits. Le calcul de l'ensemble des états accessibles ainsi que la vérification des trois propriétés sont effectués plus rapidement sur la plate-forme abstraite que sur la plate-forme concrète. Le nombre de variables BDD n'est pas réduit, par contre de très nombreuses variables n'interviennent pas dans la vérification : ce ne sont pas des variables de décision pour ce calcul, aussi n'apparaissent-telles pas dans les BDD construits pour cette vérification. De plus, la profondeur de l'espace d'états à parcourir est bien plus réduite sur le système abstrait (colonne FSM depth).

| Plateform name | FSM depth | # BDD var | BDD size (peak) | # reach. states | reach. state space traversal time | property name | property checking time |
|---|---|---|---|---|---|---|---|
| | | | | | | P1 | 6.75s |
| 1 M - 1 S (C) | 475 | 289 | 34 578 | 8.56 E +06 | 50.61s | P2 | 6.70s |
| | | | | | | P3 | 6.88s |
| | 7 | 261 | 522 | 2.73 E +16 | 2.1s | P1 | 0.1s |
| 1 M - 1 S (A) | 8 | 292 | 796 | 1.42 E +21 | 5.3s | P2 | 0.2s |
| | 10 | 302 | 1002 | 1.69 E +22 | 8.0s | P3 | 0.25s |
| | | | | | | P1 | 46mn |
| 2 M - 1 S (C) | 604 | 436 | 161 846 | 3.10 E +10 | 43mn | P2 | 19mn |
| | | | | | | P3 | 48mn |
| | 10 | 395 | 1005 | 2.83 E +20 | 3.5s | P1 | 0.8s |
| 2 M - 1 S (A) | 11 | 442 | 1079 | 4.93 E +24 | 3.8s | P2 | 0.5s |
| | 14 | 501 | 1564 | 2.50 E +26 | 12.1s | P3 | 8.25s |

Tab. 2.1 – Performances pour la vérification des propriétés P1, P2 et P3. Les résultats ont été obtenus à l'aide du model-checker VIS avec l'option de réordonnancement dynamique sur un Pentium IV, cadencé à 3.20GHz avec 1Mo de cache et 1Go de RAM

## 2.4 Conclusion et Perpectives

Nous avons défini un algorithme d'abstraction d'un composant utilisant un sous-ensemble de rêgles ACTL, produisant un composant abstrait, qui, intégré dans le processus de vérification compositionnelle, peut allèger considérablement le processus de vérification. Les expérimentations menées sur la plate-forme intégrant des convertisseurs de protocoles ont montré l'intérêt de cette démarche. Nous souhaitons poursuivre cette approche en travaillant sur trois aspects complémentaires :

1. Intégration de l'algorithme d'abstraction dans un processus de vérification de type CEGAR, couplé au model-checker VIS. L'analyse automatique (ou partiellement automatique) du contre-exemple est un problème non résolu : il s'agit de déterminer si un contre-exemple (abstrait) est pertinent ou erroné, puis de traduire un contre-exemple erroné en un ensemble de propriétés de la spécification des composants (et sans doute de contraintes supplémentaires). La production d'un contre-exemple erroné peut découler d'une spécification incomplète des composants. Peut-on alors compléter la spécification des composants ?
2. Définition de critère de choix d'un sous-ensemble de règles logiques de la spécification pour converger rapidement vers une bonne abstraction.
3. Développement d'un environnement de vérification de type Assume-Guaranty, partiellement interactif, permettant à l'utilisateur de définir les abstractions à partir des spécifications, ainsi que les regroupements de composants nécessaires à ce mode de vérification.

# Deuxième partie

# Algorithmique pour la vérification

# Chapitre 3

# Vérification de systèmes décrits à un niveau abstrait

Ce chapitre définit une structure de données de type DAG, les DDD [58], pour représenter les ensembles d'états des systèmes dynamiques, ainsi que les opérations de manipulation de ces structures nécessaires pour construire un model-checker symbolique. Nous montrons également comment les DDD sont utilisés pour représenter des systèmes concurrents disposant d'informations structurées, décrits dans le langage Promela [81]. Cette représentation est utilisée pour vérifier des propriétés par model-checking. Les performances du model-checker utilisant les DDD pour représentation symbolique sont comparées aux techniques de model-checking classiques, de type énumérative ou basée sur les BDD. Nous montrons que les DDD hiérarchiques offrent une réelle plus-value pour l'analyse de systèmes structurés très fortement concurrents.

Les travaux décrits ici ont été réalisés dans le cadre d'une collaboration entre le LabRI, le LIP6 et la DGA entre 1999 et 2001 (projet CLOVIS). Ils ont été poursuivis entre 2002 et 2006 par les travaux de thèse de V. Beaudenon, concernant l'exploitation de la structuration du programme et des données inhérente aux programmes impératifs concurrents. Parallèlement, d'autres travaux ont été menés sur les DDD, notamment dans le cadre de la thèse de Y. Thierry-Mieg [123], encadrée par F. Kordon et J.-M. Couvreur, qui propose une extension hiérarchique des DDD. Nous avons bénéficié de ces résultats récents pour développer un model-checker performant, et l'appliquer à des systèmes matériels de moyenne complexité décrits en langage Promela.

## 3.1 Préliminaires

Nous nous intéressons ici à une représentation des systèmes matériels plus abstraite que celle décrite dans la partie précédente. Si le modèle des machines de Moore synchrones est adapté pour représenté des systèmes précis au cycle près et au bit près, des modèles de plus haut niveau permettent de représenter plus grossièrement des systèmes complets. Le langage de description de matériel SystemC intègre le niveau de description transactionnel (*TLM* pour *Transaction Level Modelling*), qui permet l'indentification des composants et des flux de communications entre eux

sans définir les caractéristiques précises de ces entités.

### 3.1.1 Description des systèmes : un sous-ensemble statique de Promela

Les systèmes que nous considérons ont les caractéristiques suivantes :
- ils sont constitués d'un ensemble statique d'entités concurrentes exécutant chacune une suite d'instructions (*processus* ou *tâches*)) ;
- l'évolution des entités est asynchrone ;
- les entités communiquent par envoi et réception de messages ou par variables partagées ;
- les données manipulées sont structurées : les types entiers, tableaux statiques, enregistrements à profondeur fixe permettent de décrire des messages complexes. Des files d'attentes (bornées, mais qui peuvent être de grande profondeur) mémorisent les messages en transit.

Ce niveau de description est une abstraction du système réel. Les entités concurrentes sont décrites uniquement par la suite des traitements qu'elles réalisent, aucune information quant à l'implantation réelle de ces traitements n'est donnée. Certaînes entités pourront être implantées sous forme matérielle, d'autres sous forme logicielle. La durée de chaque traitement élémentaire n'est pas connue, et on ne suppose pas de synchronisme entre les entités. Ceci justifie la sémantique d'exécution asynchrone du système. La communication est abstraite en envoi/réception de messages. Les messages sont typés et ont une structure complexe, permettant d'identifier de façon commode l'entête du message et son contenu. Les variables partagées peuvent être utilisées ; l'accès à une variable partagée simple (i.e. non structurée) est atomique.

Le choix de ce sous-ensemble permet d'exploiter la structure des modèles, et les phénomènes de localité, qui sont complètement masqués par le modèle synchrone (ou le modèle de simulation *event-driven*), tout en représentant l'information significative des messages.

Les caractéristiques que nous retenons ici sont un *sur-ensemble* de celles utilisées dans le système de synthèse d'architecture DISYDENT [6], qui permet d'explorer plusieurs solutions architecturales pour implanter physiquement un système concurrent. Les systèmes synthétisables par l'outil Disydent sont décrits dans le langage DPN permettant la représentation d'une restriction finie du modèle des réseaux de Kahn [86]. Les restrictions de DPN par rapport à notre sous-ensemble sont les suivantes : DPN interdit l'usage des variables partagées ; les communications entre tâches DPN sont point à point : un canal de communication ne peut pas être partagé entre plusieurs émetteurs et/ou récepteurs. DPN ne dispose pas de type structuré : les données manipulées sont des vecteurs de bits (ou *mot* (de bits)) d'une taille (ou *largeur*) définie par l'utilisateur. Les transferts de messages spécifient le nombre de mots (et la largeur des mots) à écrire ou à lire dans le canal de communication. Les restrictions de DPN sont guidées par des propriétés de déterminisme et des contraintes d'automatisation de l'implémentation matérielle[1].

Notre objectif est de proposer un outil de vérification par model-checking de systèmes concurrents asynchrones englobant le modèle DPN (avec une visée d'implantation automatique), et pouvant être comparé aux techniques traditionnelles de model-checking, pour une classe plus large de systèmes. Nous nous intéressons aux exemples classiques de l'algorithmique distribuée : les philosophes, l'algorithme d'élection sur un anneau, l'algorithme de Peterson, l'algorithme Ba-

[1] Le déterminisme est garanti si l'instruction de sélection n'est pas employée, et si la capacité des canaux de communication est suffisamment grande.

kery) et aux protocoles de communication décrits à un niveau abstrait, par exemple ZCSP [21] ou VCI-SPIN [47].

### 3.1.2 Techniques de vérification par modèle – description et limites

La vérification de propriétés logico-temporelles pour de tels systèmes est réalisée classiquement par model-cheking (ou vérification par modèle). Un système de transitions est dérivé du programme représentant le système, puis l'ensemble des états accessibles du système de transitions est construit, et différents algorithmes de parcours de graphe sont utilisés pour visiter et analyser les états afin de déterminer si une propriété donnée est satisfaite ou non.

#### Le model-checker SPIN

Il est naturel d'envisager l'utilisation de l'outil SPIN [82] [81] pour vérifier des propriétés logico-temporelles sur les systèmes caractérisés précédement. SPIN est un outil de simulation et de vérification de systèmes statiques et dynamiques (de taille finie) décrits dans le langage Promela. Il a été utilisé avec succès pour vérifier des propriétés de sûreté et de vivacité (d'instances finies) d'algorithmes distribués et de protocoles de communication. Le mécanisme de vérification mis en œuvre est le suivant : Après avoir construit une représentation interne du programme sous forme de système de transitions, SPIN parcourt explicitement le graphe des états accessibles, à la recherche d'un état (ou d'une séquence d'états) invalidant la propriété. Cette recherche est effectuée par un parcours du graphe des états accessibles *en profondeur d'abord*, et à la volée (c'est à dire lors de la construction du graphe des états accessibles). Si un état (ou une séquence d'états) invalidant la propriété est trouvé, le processus de vérification est terminé, un diagnostic d'échec et une trace exhibant un contre-exemple sont fournis. Si l'intégralité du graphe des états accessibles est parcouru sans qu'un état (ou une séquence d'états) invalidant la propriété ait été rencontré, alors le processus de vérification est terminé et la propriété est vérifiée.

Cette approche est bien adaptée à la recherche de contre-exemple, dans les phases de mise au point des modèles, lors des premières étapes de conception des systèmes. Par contre, la preuve qu'une propriété est vérifiée nécessite le parcours de l'intégralité du graphe des états accessibles. Des améliorations notables ont été apportées à ce processus de base : les *réduction d'ordre partiel* permettent de ne considérer qu'un sous-ensemble des séquences d'exécution du système ; le *bit-state hashing* et le *state-space caching* optimisent la gestion de l'ensemble états précédemment visités.

En pratique, SPIN permet de vérifier des propriétés logico-temporelles linéaires (LTL) sur des systèmes présentant jusqu'à $10^6$ ou $10^7$ états. Il nous a été possible de vérifier des propriétés de vivacité sur le protocole ZCSP[2] que nous avons décrit dans le sous-ensemble statique de Promela [21], au prix d'un gros effort de simplification du modèle et de réduction des domaines

[2] le protocole ZCSP sécurise des communications sur des liens pouvant perdre des messages. Il a été implanté dans la machine parallèle MPC. Ce protocole nécessite le stockage de nombreuses informations qualifiant les messages émis non encore acquittés, ce qui rend sa vérification beaucoup plus complexe que celle du protocole *sliding-window* auquel il s'apparente

de définition des variables. Par ailleurs, nous avons modélisé le réseau VCI-SPIN [3] en Promela afin de vérifier l'absence de livelocks. Là encore, la vérification avec l'outil SPIN n'a pu être menée à bien qu'après de très fortes simplifications.

### Le model-checking à base de Diagrammes de Décision Binaire (BDD)

Une approche alternative consiste à parcourir le graphe des états accessibles en manipulant directement des ensembles d'états, sans énumérer les éléments des ensembles. Cette approche est bien adaptée à un parcours du graphe des états accessibles *en largeur d'abord*, et est particulièrement utilisée pour la vérification de propriétés logico-temporelles arborescentes (CTL). La vérification de propriétés CTL sur un système de transitions fini (appelée ici une structure de Kripke) peut être réalisée de façon *ensembliste*, si l'on dispose [115] :

1. d'une représentation symbolique d'ensembles d'états,
2. d'opérateurs ensemblistes efficaces pour cette représentation symbolique,
3. et des opérateurs *Post* et *Pre*, permettant de construire respectivement les ensembles des états successeurs et prédécesseurs d'un ensemble d'états donné[4].

Cette démarche a été transposée pour des représentations *symboliques* (disons *non énumératives*) d'ensembles d'états. La représentation symbolique d'ensemble d'états la plus communément répandue est basée sur les diagrammes de décision binaire [37]. À l'ensemble d'états $E$ est associé sa fonction caractéristique $\chi_E$, elle même représentée par un BDD `E`. La représentation des ensembles sous forme de BDD a les propriétés suivantes :

- Pour un ordre total des variables d'état, `E` est une représentation canonique de $E$.
- Le nombre de noeuds (et d'arcs) de `E` ne dépend pas de la cardinalité de $E$, et en général, sa représentation en mémoire est de beaucoup plus petite taille que l'énumération de $E$.
- Les opérations ensemblistes sur $E$ et $F$ sont réalisées par des parcours combinés de `E` et `F` de complexité quadratique (opérateur APPLY).
- Les opérations $Post(E,t)$ et $Pre(E,t)$ (pour $t$ un sous-ensemble de $T$ l'ensemble des transitions) sont réalisées par composition d'opérations élémentaires sur les BDD (une substitution de variables, une conjonction (APPLY), une quantification existentielle).

Cette représentation présente une limitation majeure : le problème de l'ordre des variables. Si la taille du BDD ne dépend pas de la cardinalité de l'ensemble qu'il représente, cette taille dépend grandement de l'ordre d'occurrence des variables dans le BDD. Il n'existe pas de stratégie garantissant un "bon ordre". Celui-ci dépend de l'ensemble calculé, un bon ordre pour une étape de calcul peut s'avérer mauvais pour une autre étape. Des stratégies de réordonnancement dynamique repoussent ce défaut mais ne l'élimine pas.

De nombreux outils de model-cheking académiques et commerciaux ont été proposés depuis l'avènement des BDD, tout particulièrement dans le domaine de la vérification des systèmes matériels synchrones : la chaîne d'outils SIAM-PRIAM [57] et le model-checker SMV [96] sont

[3] Il s'agit d'un réseau multi-étages intégrant 8 routeurs RSPIN $4 \times 4$, et muni de 16 convertisseurs de protocoles VCI-SPIN, développé au LIP6 [47].

[4] On peut étendre ces notions à la vérification de systèmes infinis, intégrant du temps représenté par des variables réelles, ou intégrant des types de données infinies (files non bornées, entiers naturels), en les munissant d'une représentation symbolique finie et sous certaines contraintes ; ceci sort du cadre des systèmes finis qui nous occupe ici.

incontestablement les pionniers, l'outil VIS [68] est le premier à analyser du code Verilog synchrone. Les outils VPN-VMC et VPN-PSM que nous avons développés R. Bawa et moi sont parmi les premiers à appliquer des techniques de model-checking sur des programmes VHDL respectant la sémantique de simulation *event-driven* du langage [65] [17] [18]. Plus récemment, ces méthodes d'analyse ont été proposées dans des outils de CAO de circuits commerciaux, tels Cadence-SMV [4], Rulebase [5] ou IFV [3]. Les outils réalisant l'équivalence de deux descriptions (combinatoire ou séquentielle) sont bien implantés dans le processus de conception de circuit industrielle ; en revanche, les outils vérifiant des propriétés logico-temporelles par model-cheking restent encore peu utilisés.

L'utilisation des BDD pour les systèmes asynchrones est possible mais elle reste plus marginale. Les travaux de Pastor et al. [106] sont les premiers à avoir proposé l'utilisation des BDD pour la construction d'ensembles d'états de réseaux de Petri bornés. Par la suite, ils ont été utilisés dans l'outil PETRIFY [56] comme représentation symbolique d'ensemble d'états de systèmes permettant la synthèse de circuits asynchrones [107], ou l'analyse de systèmes concurrents asynchrones [116].

## 3.2 DDD plats et hiérarchiques

Pour pallier les déficiences des BDD, de très nombreuses représentations d'ensemble d'états à base de DAG ont été proposées dans la littérature. Parmi ceux-ci, les MDD [51], puis les 2KMDD [53] permettent une analyse très efficace de systèmes asynchrones. La représentation symbolique est spécialisée pour le codage d'états de réseaux de Petri bornés. Les opérations de manipulation sont optimisées pour le traitement des actions locales.

La structure de données que nous proposons, les DDD [58], offre une plus grande souplesse : elle permet de représenter des systèmes dynamiques (des files d'attente non bornées à priori), des données structurées, et elle intègre la hiérarchie. Nous verrons que cette dernière caractéristique permet d'obtenir des gains en performances très significatifs pour certaines classes de systèmes.

### 3.2.1 Définition structurelle

Etant donné un ensemble de variables $V$, dont chaque élément $e$ est doté d'un domaine de définition $Dom(e)$, un Diagramme de Décision de Données, ou DDD, est un DAG utilisé pour représenter des ensemble finis de suites d'affectations de variables de $V$. Les noeuds du DAG représentent les variables de $V$ et les arcs sont étiquetés par les valeurs affectées à ces variables.

La feuille 1 est le critère d'appartenance : une séquence d'affectations $< (var_1 = val_1), (var_2 = val_2)..., (var_n = val_n) >$ est reconnue (fait partie de l'ensemble $E$) si le DDD `E` contient un chemin $n_1 \xrightarrow{v_1} ...n_n \xrightarrow{v_n} 1$. La feuille 0 est le critère de non appartenance : une séquence d'affectations représentée sur le DDD par un chemin menant à la feuille 0 ne fait pas partie de l'ensemble $E$. La feuille $\top$ est le critère d'approximation : une séquence d'affectations menant à la feuille $\top$ représente une séquence imparfaitement définie.

**Définition 5** *DDD [58] L'ensemble* $\mathbb{D}$ *des DDDs est défini tel que* $d \in \mathbb{D}$ *si :*
- $d \in \{0, 1, \top\}$ *ou*
- $d = (e, \alpha)$ *avec :*
  - $e \in E$
  - $\alpha : \mathcal{D}om(e) \to \mathbb{D}$, *tel que* $\{x \in \mathcal{D}om(e) \mid \alpha(x) \neq 0\}$ *est fini.*

*Nous représentons par* $e \xrightarrow{a} d$ *le DDD* $(e, \alpha)$ *tel que* $\alpha(a) = d$ *et pour tout* $x \neq a$, $\alpha(x) = 0$.

Un DDD est bien défini s'il ne contient pas de symbole $\top$.

Les DDD se distinguent des BDD ou des MDD sur plusieurs points :
- la définition des DDD n'impose pas d'ordre d'occurence des variables le long des séquences d'affectations, et une même variable peut apparaitre plusieurs fois le long d'une même séquence. Cette particularité permet de représenter aisément des structures de type liste sans que l'on ait à prédéfinir leur taille.
- Le fait que $\alpha$ soit une fonction garantit la finitude de la représentation (l'arité d'un noeud d'étiquette $e$ est au maximum le cardinal de $Dom(e)$), par contre, cela impose des restrictions lors du regroupement de séquences. Le regroupement de séquences ne respectant pas ces restrictions est désigné par la feuille $\top$.
- Les séquences n'appartenant pas à l'ensemble E ne sont pas représentées dans le DDD représentant E. Cette particularité est commune aux DDD et MDD, mais elle se distingue des BDD. Elle a plusieurs conséquences :
  - il n'est pas nécessaire de descendre à la feuille 1 pour évaluer un calcul si les données du prefixe sont suffisantes. Dans le cas du BDD il faut descendre jusqu'aux feuilles et c'est là seulement que l'on sait si la séquence (entière) est reconnue ou non ;
  - on ne peut pas exploiter la réduction de longueur de chemin comme dans les BDD, dans lesquels les noeuds correspondant à des variables qui ne sont pas décisives (pour un ensemble de sous-séquences données) ne sont pas représentés.
- La définition permet la construction de DDD hiérarchiques. En effet la définition ne fait aucune hypothèse sur le type des variables de $V$, qui peuvent être des variables représentant des ensembles, et non pas des valeurs. Dans [59], J.-M. Couvreur et Y. Thierry-Mieg ont introduit les SDD, qui sont une généralisation des DDD pour pouvoir prendre en compte la hiérarchie, et nous avons montré dans [19] qu'en fait, lorsque les variables ont des domaines finis, les SDD répondent bien à la définition des DDD.

Dans [58], nous montrons que si les séquences menant à la feuille 0 ne sont pas représentées, la structure DDD est canonique pour une interprétation des chemins en termes de séquences d'affectations.

### 3.2.2 Introduction d'un ordre sur les variables

Cette structure est trop expressive pour la représentation d'ensemble d'états de systèmes statiques. L'absence de définition d'un ordre d'occurrence des variables nécessite l'introduction des notions d'approximation de séquence et de bonne définition, qui ne sont pas réellement nécessaires dans le cadre des systèmes statiques. Dans le projet CLOVIS, les model-checkers prototypes développés pour la vérification de programmes VHDL et de différentes classes de réseaux de Petri n'utilisaient pas réellement cette liberté : implicitement, un ordre global restreignant l'ordre d'occurrence des variables était adopté, le symbole $\top$ ne devait plus survenir qu'en cas

d'erreur d'implémentation.

Dans [19] nous avons explicité cette notion d'ordre, en définissant un ordre des variables compatible avec une forme restreinte de répétition de variables. Les DDD ordonnés (c'est à dire dont l'ordre d'occurrence des variables le long de tous les chemins respecte cet ordre global) permettent de représenter des variables scalaires, tableaux, et files.

Les DDD ordonnés, sont alors toujours bien définis (le symbole $\top$ n'est plus nécessaire) et sont une représentation canonique pour un ensemble d'états. L'application d'opérateurs ensemblistes sur des DDD bien définis produit un DDD résultat canonique (respectant l'ordre global des variables) et bien défini. En pratique, les DDD sont implantés dans une table d'unicité, garantissant que chaque séquence et ensemble de séquences est représenté de façon unique.

### 3.2.3 Opérations sur les séquences et sur les ensembles de séquences

Les séquences d'affectations peuvent être concaténées. Cette concaténation est étendue aux ensembles de séquences : la concaténation de $E_1$ avec $E_2$ signifie que chaque séquence $w_1$ de $E_1$ est concaténée avec chaque séquence $w_2$ de $E_2$. Les opérations d'union, d'intersection et de différence ensembliste ont été définies dans [58]. Le mode opératoire est très semblable à l'algorithme APPLY des BDD, seuls les cas de base, et le nombre de successeurs à considérer diffèrent.

La complémentation n'est pas définissable, car on ne connait pas a priori le domaine des variables. Ces opérations ensemblistes préservent l'ordre sur les variables et la bonne définition des DDD.

### 3.2.4 Transformation d'ensembles : la notion d'homomorphisme

La notion d'homomorphisme est sans doute la plus originale. Elle formalise les opérations de transformations d'ensembles, représentés par des DDD, sous la forme de parcours partiaux des DDD. Le parcours du DDD à modifier, ainsi que la reconstruction du résultat sous forme canonique est réalisée par un homomorphisme. Cette notion formalise l'ensemble des transformations applicables sur chaque chemin d'un DDD.

Soient $\phi$ la transformation à appliquer à un DDD $d$, résultant en un nouveau DDD $d'$, noté $d' = \phi(d)$. L'ensemble des transformations admissibles est caractérisé par : $\phi$ est un homomorphisme au sens mathématique du terme. C'est à dire, soient deux DDD *ordonnés* $d_1$ et $d_2$, $d_1 + d_2$ est leur union, alors $\phi(d_1 + d_2) = \phi(d_1) + \phi(d_2)$ (cette propriété n'est pas vérifiée pour les DDD bien définis en général).

Les homomorphismes inductifs forment une sous-classe des homomorphismes. Ils ont la particularité d'être définis localement au noeud où ils s'appliquent. Cette définition locale permet de construire une procédure effective de transformation de DDD. Cette procédure parcours le DDD "en profondeur d'abord", applique la transformation sur chaque chemin du DDD (en arrêtant le parcours du chemin dès que possible), et produit un DDD résultat canonique.

**Définition 6 (Homomorphisme Inductif [58])** *Soit $c$ un DDD and $\phi(e,x)$, $(e \in E, x \in Dom(e))$ une famille d'homomorphismes (particularisés pour la variable $e$ et ses valeurs possibles).*

$$\Phi(d) = \begin{cases} 0 & if\ d = 0 \\ \top & if\ d = \top \\ c & if\ d = 1 \\ \sum_{x \in Dom(e)} \phi(e,x)(\alpha(x)) & if\ d = (e,\alpha) \end{cases}$$

*est un homomorphisme.*

**Exemple 1** L'affectation de la constante `cst` à la variable `var` peut être réalisée par un parcours partiel défini à partir de l'homomorphisme inductif $\langle \mathsf{setCst} \rangle$. Celui-ci prend en compte les deux paramètres *var* (représentant `var`) et *const* (représentant `cst`) ; son application au noeud du DDD d'étiquette $e$ et à l'arc sortant d'étiquette $x$ est définie comme suit :

$$\langle \mathsf{setCst}(var, cst) \rangle (e,x) = \begin{cases} \text{if } var = e & e \xrightarrow{cst} \langle \mathsf{id} \rangle \\ \text{else} & e \xrightarrow{x} \langle \mathsf{this} \rangle_{=\langle \mathsf{setCst}(var,cst) \rangle} \end{cases}$$

L'homomorphisme $\langle \mathsf{SetCst}(var, cst) \rangle$ se propage à partir de la racine du DDD. Lorsqu'il s'applique sur le couple $(e,x)$ (le noeud $e$ et l'arc sortant étiqueté par $x$), si $e$ représente la variable $var$, l'étiquette de l'arc $x$ est remplacée par $cst$ ; sinon, l'homomorphisme se propage vers le noeud successeur de $e$ suivant l'arc d'étiquette $x$ (désigné par $\alpha(x)$). La canonisation du résultat est assurée lors de la remontée de l'homomorphisme, par le biais de la table d'unicité des noeuds.

Les homomorphismes inductifs ne garantissent pas la préservation de l'ordre ni la bonne définition de leur résultat. Nous avons montré dans [19] que l'application de la composition d'homomorphismes inductifs de base aux DDD ordonnés préserve l'ordre et la bonne définition du résultat.

### 3.2.5 Mécanismes d'accès propres aux DDD hiérarchiques

Dans [19], nous avons particularisé l'homomorphisme inductif aux DDD hiérarchiques et avons montré que cette définition est cohérente avec les traitements ad-hoc proposés par J.-M. Couvreur et Y. Thierry-Mieg dans [59] et implantés dans la bibliothèque [1].

De plus, nous avons défini des primitives permettant l'accès (et la modification) de données au travers de différents niveaux de hiérarchie. Ces primitives permettent d'aiguiller les homomorphismes : ceux-ci sont propagés à partir d'un niveau de hiérarchie $k$, vers les noeuds successeurs du même niveau de hiérarchie $k$, ou bien vers des noeuds successeurs de niveau de hiérarchie $k+1$. Ces primitives sont les suivantes [19] :

- $\langle \mathsf{Split} \rangle$ : cette primitive énumère les éléments d'un ensemble situé à un niveau hiérarchique supérieur.
- $\langle \mathsf{Flat} \rangle$ : cette primitive "écrase" localement un niveau de hiérarchie. L'ensemble $S$, de niveau de hiérarchie $k+1$ et associé à l'arc $e_i$ de niveau de hiérarchie $k$ est substitué à cet arc.

– ⟨Rebuild⟩ : cette primitive "reconstitue" le niveau de hiérarchie préalablement écrasé. L'ensemble $S$, implanté dans le niveau de hiérarchie $k$ est déporté dans le niveau de hiérarchie $k+1$ et un arc $e_i$ lui est associé dans le niveau de hiérarchie $k$.

## 3.3 Application aux systèmes décrits dans le sous-ensemble statique de Promela

### 3.3.1 Représentation d'un état sous forme de DDD hiérarchique

Un état du programme est une configuration de toutes les variables composant de programme. La structure du programme et de ses types de données transparait dans la structure hiérarchique du DDD représentant l'état du programme :

Le DDD principal, de niveau hiérarchique 0, contient les variables permettant l'identification de chaque processus déclaré, et de chaque variable globale. Un DDD de niveau 1 représente l'état d'un processus ou d'une variable globale. L'état d'un processus est composé d'une variable désignant la valeur du compteur ordinal et d'éventuelles variables désignant les valeurs des éventuelles variables locales. Un DDD représentant l'état d'une variable (identifiée au niveau 0 si elle est globale ou au niveau 1 si elle est locale) est lui-même hiérarchisé en suivant la composition de son type.

A un identificateur de variable déclarée au niveau $k$ correspond une représentation énumérative de ses éléments au niveau $k+1$. Les variables du programme peuvent être de trois types :
- scalaire : il s'agit des variables représentant le compteur ordinal de chaque processus, et des variables déclarées dont le type est scalaire (`bool, bit, int, ...`).
- vectorielle : il s'agit des variables de type composé de taille fixe (type `array` ou `record`). Les champs des variables de taille fixe sont désignées par un ensemble figé de variables distinctes, sur un DDD de niveau $k+1$.
- conteneur : il s'agit des variables de type composé dont la taille évolue dynamiquement au cours de l'exécution du programme (type `channel`). Une telle variable définie au niveau $k$ est expansée en une liste de ses éléments au niveau $k+1$ ; la variable désignant un élément de la liste est répétée pour chaque élément présent, et la fin de la répétition est délimitée par un symbole particulier ♯ qui n'appartient au domaine de définition d'aucune variable du programme.

`int x ;` est représenté par $x \xrightarrow{I \xrightarrow{0} 1} 1$

`int a[2] ;` est représenté par $a \xrightarrow{a_0 \xrightarrow{I \xrightarrow{0} 1} a_1 \xrightarrow{I \xrightarrow{0} 1} 1} 1$

`chan c[2] of bit ;` est représenté par $c \xrightarrow{c \xrightarrow{2} c \xrightarrow{\sharp} 1} 1$ lorsque le canal est vide, et par $c \xrightarrow{c \xrightarrow{2} c \xrightarrow{I \xrightarrow{1} 1} c \xrightarrow{\sharp} 1} 1$ lorsqu'il contient un unique élément de valeur 1.

L'ordre global associé au programme est l'ordre lexicographique induit par cette structure hiérarchique.

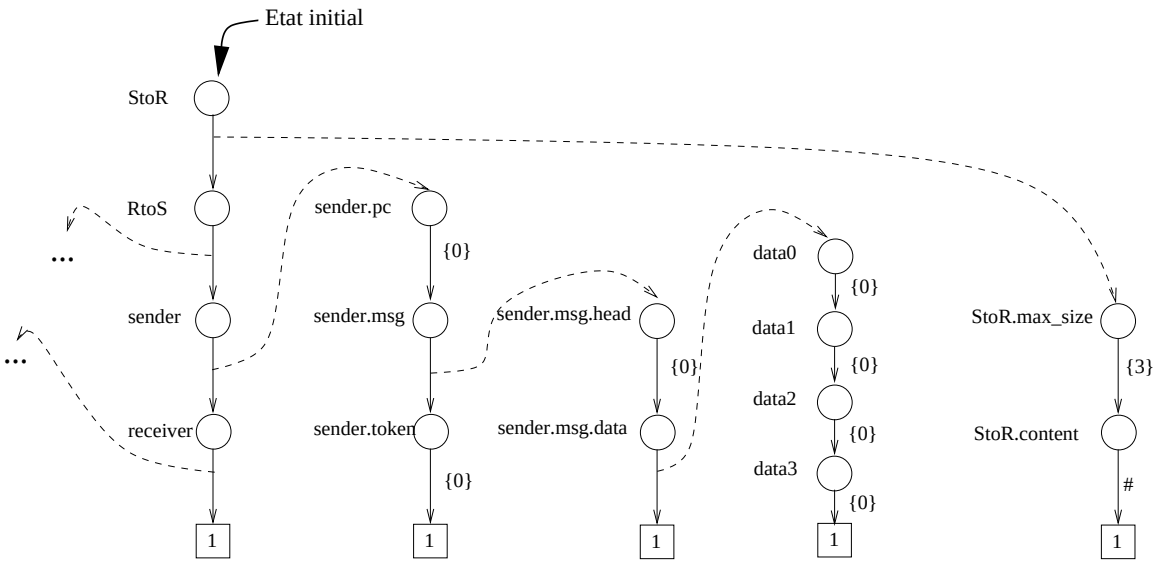


Fig. 3.1 – Le DDD hiérarchique représentant l'état initial du programme Promela donné en exemple.

**Exemple 2** Considérons la structure de programme ProMeLa suivante.

```
typedef message{
   header short;
   int data[4];
}

typedef acknowledgment{
...
}

active proctype sender(){
message msg;
bit token;
/* instructions de sender */
}

active proctype receiver(){
/* déclarations et instructions de receiver */
}

chan StoR chan[3] of message;
chan RtoS chan[1] of acknowledgment;
```

Le DDD hiérarchique représentant l'état initial du programme est donné sur la figure 3.1. Pour simplifier la figure, le niveau de hiérarchie représentant les ensembles d'entiers n'est pas représenté explicitement. Les ensembles d'entiers sont représentés directement sur les arcs : le symbole $\{0\}$ représente le DDD $I \xrightarrow{0} 1$.

### 3.3.2 Homomorphismes associés aux instructions du langage Promela

La sémantique de chaque instruction promela est traduite sous forme d'homomorphismes qui, appliqués au DDD représentant l'ensemble des états initiaux, permettent de construire l'ensemble des états successeurs en une ou plusieurs instructions. Les homomorphismes élémentaires essentiels permettant de construire les homomorphismes représentant le traitement des instructions du langage ProMeLa sont les suivants :

- La sélection d'un sous-ensemble d'états satisfaisant une expression, évaluée à la volée. Ce traitement est effectué par l'homomorphisme $\langle \mathsf{SelectExpr}(expr) \rangle$, qui combine les homomorphismes élémentaires $\langle \mathsf{SetCst} \rangle$, $\langle \mathsf{Up} \rangle$ et $\langle \mathsf{Down} \rangle$.
- L'affectation d'une variable (de type scalaire), à la valeur résultant de l'évaluation à la volée d'une expression, réalisée par l'homomorphisme $\langle \mathsf{SetExpr}(var, val) \rangle$. L'affectation d'un élément d'une variable de type vectoriel ou l'écriture dans un conteneur sont des variantes de $\langle \mathsf{SetExpr}(var, val) \rangle$. La preuve que l'homomorphisme $\langle \mathsf{SetExpr} \rangle$ respecte l'ordre global des variables est donnée dans [19].

Les homomorphismes représentant les instructions du langage sont obtenus par composition (au sens de composition de fonctions) d'homomorphismes élémentaires. Par exemple, considérons l'instruction d'affectation `var = expr ;` placée dans le processus `p_i` sur la ligne $l$. L'homomorphisme traduisant l'ensemble des traitements effectués lors de l'exécution de cette instruction est bâti à partir de :

- l'homomorphisme $\langle \mathsf{SelectCst}(pc_i, l) \rangle$ qui sélectionne le sous-ensemble des états pour lesquels le compteur ordinal du processus `p_i` est en position $l$,
- l'homomorphisme $\langle \mathsf{SetExpr}(var, expr) \rangle$ qui, pour chaque chemin du DDD opérande, modifie la valeur de `var` en lui affectant la valeur de l'expression `expr` évaluée le long du chemin courant) ;
- l'homomorphisme $\langle \mathsf{SetCst}(pc_i, l+1) \rangle$ qui met à jour la valeur du compteur ordinal en lui affectant la position suivante (la ligne $l+1$).

L'homomorphisme associé à l'instruction est alors le suivant :
$\langle \mathsf{setCst}(pc_i, l+1) \rangle \circ \langle \mathsf{setExpr}(var, expr) \rangle \circ \langle \mathsf{SelCst}(pc_i = l) \rangle$

Les instructions de lecture et écriture dans les variables ou les canaux de communication ont la même structure. L'homomorphisme $\langle \mathsf{setExpr}() \rangle$ est remplacé par d'autres homomorphismes ad-hoc. La gestion de séquences atomiques (et de possibles ruptures d'atomicité), ainsi que des clauses de réveil `timeout` est réalisée par des enchaînements des homomorphismes représentant les instructions à séquencer.

### 3.3.3 HaDDock, un model-checker symbolique

Le model-checker HaDDock, développé par V. Beaudenon dans sa thèse, met en pratique les résultats théoriques et les techniques présentées ci-dessus. Ce model-checker analyse un programme décrit dans le sous-ensemble statique de promela : il en extrait la représentation de l'état initial sous forme de DDD hiérarchique, et construit un ensemble d'homomorphismes représentant les traitements induits par l'exécution de chaque instruction du programme. HaDDock calcule alors l'ensemble des états accessibles du programme (il s'agit d'un parcours en avant et en largeur d'abord), puis il détermine le sous-ensemble des états accessibles satisfaisant une certaine propriété exprimée dans la logique CTL (il s'agit là de parcours en arrière, leur réalisation est

décrite dans [20], [22], [19]). HaDDock peut construire une représentation symbolique basée sur les BDD, s'appuyant sur la bibliothèque BUDDY [2], ou bien sur les DDD ou SDD, s'appuyant sur la bibliothèque DDD [1]. Le model-checker s'appuyant sur la bibliothèque BUDDY a été réalisé par S. Taktak pendant son stage de Master [119].

Plusieurs stratégies de parcours de l'espace d'état ont été investiquées. Les stratégies à base de saturation se sont avérées les plus efficaces : dans [52] G. Ciardo et al. ont montré que la saturation est adaptée aux structures de DAG modifiées "localement", et les DDD hiérarchiques ou plats appartiennent à cette catégorie. Dans [19], nous proposons deux stratégies de saturation, l'une privilégiant le temps de calcul au détriment de l'occupation mémoire, et une autre favorisant une occupation mémoire réduite, au prix d'un temps de calcul plus long.

### 3.3.4 Résultats expérimentaux

Le model-checker HaDDock a été comparé à SPIN, ainsi qu'à une implantation sur BDD avec ordonnancement dynamique sur des exemples académiques, mais également sur des exemples issus d'applications réelles.

#### Comparaison avec les BDD et SPIN [22]

Les exemples utilisés à titre de comparaison sont des benchmarks classiques de l'algorithmique répartie. Les composants sont tous identiques (excepté leur identification) ; le nombre de composants interagissant est paramétrable (sauf pour le protocole sliding window, pour lequel la largeur de la fenêtre est paramétrable).

Les exemples analysés sont :
- Un système très fortement concurentiel, composé de nombreux composants très simples connectés en anneau : le problème des philosophes.
- Deux systèmes présentant moins de composants, mais ceux-ci exécutant un algorithme plus complexe : l'algorithme d'élection sur un anneau, et le protocole sliding-window.
- Deux systèmes communicant par variables partagées, fortement synchronisés et manipulant des expressions arithmétiques plus complexes. L'architecture de ces systèmes ne permet pas de dégager un "ordre naturel" des variables : l'algorithme de Peterson et l'algorithme Bakery.

Les résultats présentés donnent des éléments de comparaison des model-checkers SPIN et HaDDock sur différents systèmes académiques, pour différentes représentations de léspace des états (énumérative, BDD, DDD plats, DDD hiérarchiques).

Les résultats présentés sur la Table 3.1 montrent les performances, en temps et mémoire, pour le calcul de l'ensemble des états accessibles et la vérification de propriétés de vivacité, de systèmes concurrents paramétrés (le paramètre est $N$), pour SPIN (colonnes SPIN), et pour HaDDock muni des BDD avec réordonnancement dynamique (colonnes BDD), des DDD plats sans chercher à ordonner au mieux les variables (colonnes DDD no order) et les DDD plats avec un ordre statique coincidant avec la propagation du flux d'information lorsque cela est possible (colonne DDD order, pas de bons ordres pour Sliding Window, Peterson et Bakery).

Tab. 3.1 – Comparaison des performances des outils SPIN et HaDDock (muni des BDD ou des DDD plats). Extrait de [22]

| | | SPIN | | BDD | | | DDD | | | DDD-O | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N | $\|S\|$ | reach (sec.) | mem MB | reach (sec.) | check (sec.) | mem. MB | reach (sec.) | check (sec.) | mem MB | reach (sec.) | check (sec.) | mem MB |
| Les philosophes $\mathbf{AG}(\mathbf{AF}(eating\ philosopher))$ | | | | | | | | | | | | |
| 5 | 244 | $< 1$ | 2.3 | $< 1$ | $< 1$ | 93 | $< 1$ | $< 1$ | 4.3 | $< 1$ | $< 1$ | 5.5 |
| 10 | 5.9e4 | 2.3 | 36.7 | 4 | 6 | 94 | 31 | 170 | 17.6 | 9 | 2 | 11 |
| 11 | 1.7e5 | 7.6 | 41.6 | 9 | 15 | 94 | 82 | 935 | 30 | 14 | 2 | 12.8 |
| 12 | 5.3e5 | 29.1 | 90.1 | 23 | 42 | 94 | 277 | 6652 | 57 | 21 | 3 | 16.6 |
| 13 | 1.6e6 | 193 | 114.8 | 55 | 117 | 94 | 1021 | 3.7e4 | 105 | 30 | 3 | 19.6 |
| 14 | 4.8e6 | 660 | 737 | 137 | 337 | 95 | 4534 | $> 24h$ | 203 | 41 | 4 | 24.4 |
| 15 | 1.4e7 | 2251 | 1067 | 380 | 1231 | 96 | 18375 | $> 24h$ | 397 | 54 | 5 | 29.6 |
| 20 | 3.5e9 | * | $> 1GB$ | * | * | * | * | * | $> 1GB$ | 182 | 9 | 64.1 |
| 50 | 7.2e23 | * | $> 1GB$ | * | * | * | * | * | $> 1GB$ | 8006 | 67 | 994 |
| Election sur un anneau $\mathbf{AF}(best\ candidate\ elected)$ | | | | | | | | | | | | |
| 5 | 5.4e3 | 10 | 23 | 510 | 191 | 94 | 8 | 1 | 6.3 | 11 | 1 | 6.2 |
| 6 | 3.2e4 | 262 | 144 | 5056 | 5394 | 144 | 34 | 5 | 11 | 37 | 3 | 11 |
| 7 | 1.8e5 | * | $> 1GB$ | * | * | * | 127 | 22 | 28 | 95 | 13 | 19 |
| 10 | 3.3e7 | * | $> 1GB$ | * | * | * | 4533 | * | 663 | 716 | * | 77 |
| Sliding Window $\mathbf{AG}(\mathbf{AF}(new\ message\ sent))$ | | | | | | | | | | | | |
| 2 | 4.0e5 | $< 1$ | 8.1 | 552 | 221 | 102 | 486 | 142 | 17.8 | 662 | 206 | 17 |
| 3 | 3.5e7 | 29 | 450 | 65429 | 63216 | 332 | 4.8e5 | | 565 | 4.8e5 | | 475 |
| 4 | * | * | $> 1GB$ | * | * | * | * | * | $> 1GB$ | * | * | $> 1GB$ |
| Algorithme d'exclusion mutuelle de Peterson $\mathbf{AF}(query_i\ satisfied)$ and $\mathbf{AG}(query_i \Rightarrow \mathbf{AF}(critical_i))$ | | | | | | | | | | | | |
| 2 | 208 | $< 1$ | 5.5 | $< 1$ | 1 | 94 | $< 1$ | $< 1$ | 3.6 | | | |
| 3 | 2.5e4 | $< 1$ | 5.8 | 17 | 39 | 93 | 7 | 4 | 5.9 | | | |
| 4 | 6.4e6 | 20 | 402 | 1674 | 1093 | 400 | 1278 | 216 | 113 | | | |
| 5 | * | * | $> 1GB$ | 4.7e5 | $> 24h$ | 900 | * | * | $> 1GB$ | | | |
| Algorithme d'exclusion mutuelle "Bakery" $\mathbf{AF}(client_i\ satisfied)$ | | | | | | | | | | | | |
| 4 | 1.8e5 | $< 1$ | 5.2 | 50 | $< 1$ | 94 | 17 | $< 1$ | 8.9 | | | |
| 5 | 1.6e7 | 1385 | 6.4 | 1231 | $< 1$ | 93 | 379 | 5 | 46 | | | |
| 6 | 2.2e9 | $> 24h$ | * | * | * | * | 61901 | 47 | 455 | | | |

Pour chaque outil, la colonne *reach* indique le temps nécessaire pour construire l'ensemble des états accessibles, la colonne *check* indique le temps additionnel nécessaire pour vérifier la propriété CTL, la colonne *mem* indique la quantité de mémoire nécessaire pour construire l'ensemble des états accessibles (une fois cet ensemble construit, la vérification de propriété CTL ne nécessite pas de plus de mémoire).

Les expériences ont toutes été menées sur un Pentium IV cadencé à 3.2 GHz, les calculs dépassant plus de 24h ou plus d'1 Go de mémoire sont représenté par le symbole *.

Les résultats de la table 3.1 montrent que pour ces exemples, pour les systèmes fortement concurrents, le model-checking symbolique (BDD ou DDD plats) est plus efficace que le model-cheking énumératif de SPIN : pour un même système, l'ensemble des états accessibles est calculé plus rapidement par les méthodes symboliques, et en général le model-checker symbolique permet d'analyser des systèmes comportant plus de composants élémentaires .

Les performances obtenues avec les BDD, utilisant une stratégie de réordonnancement dyna-

Tab. 3.2 – Performances de vérification avec DDD hiérarchiques. Extrait de [19]

| N | $\|S\|$ | no order | | | | | | natural order | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | BFS | | cyclic sat | | local sat | | BFS | | cyclic sat | | local sat | |
| | | reach | mem | reach | mem | reach | mem | reach | mem | reach | mem | reach | mem |
| Les philosophes | | | | | | | | | | | | | |
| 10 | 5.9e4 | 19 | 25 | 13 | 10 | 10 | 56 | 4 | 5.7 | 0 | 1.8 | 0 | 2.2 |
| 15 | 1.4e7 | 991 | 889 | 588 | 276 | * | * | 25 | 16 | 0 | 1.9 | 0 | 2.8 |
| 50 | 7.2e23 | * | * | * | * | * | * | 3308 | 532 | 5 | 4.7 | 1 | 8.3 |
| 100 | 5.2e47 | * | * | * | * | * | * | * | * | 22 | 13 | 3 | 22 |
| 1000 | 1.3e477 | * | * | * | * | * | * | * | * | 9166 | 1078 | 258 | 1343 |
| Election sur un anneau | | | | | | | | | | | | | |
| 15 | 1.9e11 | * | * | 8501 | 1821 | * | * | 2529 | 278 | 12 | 5.5 | 7 | 35 |
| 100 | 1.5e75 | * | * | * | * | * | * | * | * | 671 | 32 | 443 | 1866 |
| 1000 | 5.3e751 | * | * | * | * | * | * | * | * | 73156 | 336 | * | * |
| Sliding Window | | | | | | | | | | | | | |
| 2 | 4e5 | 3447 | 27.5 | 1.6e5 | 284 | 2244 | 774 | 2741 | 29.2 | 2.2e5 | 267 | 2569 | 657 |
| 3 | 3.5e7 | * | * | * | * | * | * | * | * | * | * | * | * |
| Peterson | | | | | | | | | | | | | |
| 3 | 2.5e4 | 53 | 5.4 | 25 | 6 | 5 | 7.5 | | | | | | |
| 4 | 6.4e6 | 31042 | 214 | 27160 | 201 | 4454 | 396 | | | | | | |
| Bakery | | | | | | | | | | | | | |
| 4 | 1.8e5 | 702 | 13 | 806 | 18 | 167 | 42 | | | | | | |
| 5 | 1.6e7e6 | 148017 | 142 | 110241 | 163 | 14767 | 491 | | | | | | |

mique, sont comparables à celles obtenues avec les DDD (non hiérarchiques) sans ordre particulier. Si un "bon ordre", reflétant la topologie du système et surtout le flux d'information entre les composants est trouvé, alors les performances obtenues pour les DDD munis de cet ordre statique sont les meilleures (jusqu'à 50 philosophes et 10 électeurs). Cet ordre statique, issu de l'analyse macroscopique du fonctionnement du système n'est en général pas un bon ordre pour l'implantation en BDD.

L'utilisation des méthodes symboliques est d'autant plus intéressante que le système est fortement concurrent. Pour les systèmes séquentiels (sliding-window), SPIN calcule plus rapidement l'espace d'états que les model-checkers symboliques pour une fenêtre de taille 3, mais aucun model-checker ne peut construire l'espace d'états pour une fenêtre de taille 4.

L'introduction de la hiérarchie dans les DDD, ainsi que des méthodes de saturation, a accentué cette tendance (voir Table 3.2, à comparer avec les résultats de la Table 3.1). Pour les systèmes très fortement concurrents, les DDD hiérarchiques sont bénéfiques : HaDDock peut calculer l'ensemble des états accessibles pour un système composé de 1000 philosophes ou de 1000 électeurs lorsque l'ordre des variables sur le DDD principal suit le flux d'information de l'application ; pour les systèmes plus séquentiels (Peterson, Bakery et Sliding Window), les DDD hiérarchiques sont contre-productifs (les performances de HaDDock avec DDD hiérarchiques sont moins bonnes que les performances de SPIN, ou des DDD plats).

**Applications concrètes[19]**

Le model-checker HaDDock a également été utilisé pour vérifier des propriétés de sûreté et de vivacité sur des systèmes issus d'applications réelles. Nous avons tenté d'analyser des programmes Promela modélisant fidèlement les protocoles ZCSP et ANI, ainsi que le réseau d'interconnexion SPIN à interface VCI.

Les deux premiers n'ont pu être analysés par model-checking symbolique (ou alors dans un temps supérieur à celui de SPIN). Ceci s'explique notamment parce qu'ils comportent peu de processus, ces derniers sont assez complexes, et les synchronisations entre processus sont très fortes : le système présente peu de concurrence.

Le réseau SPIN à interface VCI a des caractéristiques tout autres : il comprend de nombreux processus et est très fortement concurrent. C'est le candidat idéal pour les méthodes symboliques. En effet, HaDDock sur SDD a pu construire l'ensemble des états accessibles d'un système composé de 4 routeurs $2 \times 2$, connectés à 2 initiateurs et 2 cibles VCI, émettant des messages comprenant jusqu'à 7 cellules. Cette analyse a permis de montrer que l'architecture du réseau VCI-SPIN séparant les chemins d'acheminement des requêtes des chemins acheminant les réponses était exempt d'interblocage, alors que l'architecture ne distinguant pas ces chemins pouvait mener à des interblocages.

HaDDock muni des DDD hiérarchiques est le seul outil à avoir pu construire l'espace d'états de ce système : HaDDock sur BDD ou DDD, ou encore SPIN n'ont pu construire que des sous-ensembles très restreints de cet espace d'états, en considérant des systèmes très simplifiés et artificiellement synchronisés afin de réduire la concurrence.

Les expériences menées confirment les tendances dégagées sur les systèmes académiques : les systèmes comportant peu de composants, qui sont complexes et fortement synchronisés ne sont pas bien analysables par les méthodes symboliques. Par contre, pour les systèmes très concurrents, composés de nombreux processus assez simples, le model-checker HaDDock offre des possibilités d'analyse qui n'existaient pas auparavant.

## 3.4 Prolongements possibles

Les travaux décrits ici ont montré que l'exploitation de la structure des systèmes à analyser permettait de repousser les limites des model-checkers symboliques, pour certaines classes d'application. Les DDD hiérarchiques sont un support permettant de représenter la structure du système en processus, aussi bien que la structure des types de données manipulés. Les limites du model-checking symbolique sont repoussées, et non pas abolies. Ces méthodes de model-checking, très générales, permettent l'analyse de systèmes de moyenne complexité ; elles ne sont toujours pas adaptées pour la vérification de systèmes très complexes, décrits avec une très grande précision.

En l'état actuel, les DDD hiérarchiques présentent encore des limitations : ils sont difficiles à manipuler. L'écriture d'homomorphismes est laborieuse et source d'erreur. Le développement

d'un cadre théorique permettant l'écriture simple des traitements à effectuer lors d'un parcours de DDD, sans avoir à gérer explicitement les changements de niveaux de hiérarchie est nécessaire. L'explicitation de ces mécanismes, par le biais des homomorphismes $\langle \mathsf{Split} \rangle$, $\langle \mathsf{Flat} \rangle$ et $\langle \mathsf{Rebuild} \rangle$ était une étape nécessaire ; il faut maintenant poursuivre cette tâche afin d'offrir au programmeur des lignes directrices, ou des gabarits de conception limitant les risques d'erreur de définition des homomorphismes.

Les méthodes et techniques proposées dans ce chapitre, dans le cadre de systèmes décrits en Promela peuvent être transposés pour d'autres langages de description, tels VHDL ou SystemC. Un prototype de model-checker (à base de DDD plats) avait été réalisé dans le cadre de la collaboration LaBRI-LIP6-DGA [67] : aux homomorphismes élémentaires réalisant les traitements d'affectation dans les variables ou signaux, avait été ajouté un ordonnanceur émulant le cycle de simulation (d'un sous-ensemble) du langage VHDL.

Les idées exposées dans ce chapitre peuvent être reprises pour appliquer les méthodes de model-checking symbolique lors de la conception de systèmes matériels décrits en SystemC ou VHDL. Cette analyse peut intervenir à différentes étapes du processus de conception d'un système : au niveau transactionnel, dans lequel seules les communications (asynchrones) sont décrites, ou à un niveau plus détaillé (collection de machines à états finis synchrones, avec signaux de Moore ou de Mealy). Dans ce dernier cas, les contraintes d'ordonnancement liées à la représentation du cycle de simulation peuvent limiter le gain apporté par les méthodes de saturation. La notion de hiérarchie exploitée ici pourrait être poussée plus avant en considérant la description de composants statiques (qui fait défaut à Promela), et qui est présente dans les descriptions structurelles VHDL ou SystemC.

# Chapitre 4

# Détection automatique de deadlocks dans les NoCs

Les réseaux intégrés sur puce (ou Networks on Chip, NoCs) sont un composant crucial des systèmes intégrés sur puce. Du fait de la complexité croissante de ces systèmes et notamment du nombre important de composants à interconnecter, les besoins en débit et bande passante ont énormément crû. Les traditionnels bus (bus système et bus périphérique) se sont vus remplacer par des architectures de réseau plus complexes, telles les hiérarchies de bus (AMBA), les réseaux étagés (SPIN, [73]), les topologies de grille (DSPIN [103]) ou d'autres encore (OCTAGON [55], EATHERAL [71], reconfigurable à base de FPGA ...). Ces architectures de réseaux étaient déjà bien connues dans le domaine des communications entre machines distantes. Elles sont maintenant transposées sur une unique puce de silicium, ce qui induit des contraintes particulières : 1) leur topologie doit être simplement implantable en deux dimensions ; 2) en cas de défaillance d'un composant interne, celui-ci ne peut pas être remplacé.

La conception d'un NoC et des protocoles d'acheminement qu'il pourra fournir est une tâche bien identifiée. Elle se heurte à des problèmes d'apparition de deadlocks (situation d'interblocage de la progression des messages dans les canaux du réseau). Les techniques de model-checking sont assez inefficaces pour déterminer des propriétés d'absence de deadlocks dans les NoCs. On a vu au chapitre précédent que l'outil SPIN ne pouvait pas traiter correctement ce problème pour un modèle très simplifié du réseau VCI-SPIN ; les techniques symboliques basées sur les DDD hiérarchiques permettent de répondre partiellement au problème, mais ne permettent pas de prouver qu'une architecture complète est exempte de deadlock.

Dans ce chapitre, nous abordons le problème spécifique de la preuve d'absence de blocage sous un angle différent : plutôt que d'utiliser des outils de vérification très généraux, basés sur l'énumération –même symbolique– de l'espace d'états, nous nous concentrons sur la preuve d'une propriété uniquement – l'absence de deadlock– et analysons le graphe de dépendances entre messages circulant dans le réseau. Cette méthode évite de parcourir l'espace d'états du système.

Dans ce cadre, nos contributions sont les suivantes :

– une extension des résultats théoriques de Duato [64], permettant de modéliser des pro-

tocoles disposant de plusieurs types de messages avec des dépendances entre eux. Cette distinction des messages en plusieurs types est nécessaire pour analyser les protocole manipulant des messages de type "requête" et de type "réponse", par exemple le protocole VCI.
- l'établissement d'une condition suffisante (et non pas nécessaire et suffisante), garantissant l'absence de deadlock, et la proposition d'un algorithme original de détection de deadlock pour cette condition. Cet algorithme est comparé à l'approche standard décrite dans [114] sur un ensemble de réseaux réels. La comparaison porte sur deux critères :
  - l'acuité des résultats : notre condition étant seulement suffisante, notre méthode peut rejeter des réseaux considérés comme susceptibles de contenir des deadlocks, alors qu'ils n'en contiennent pas ;
  - les performances des algorithmes en temps et mémoire.

Ces travaux ont été réalisés dans le cadre de la thèse de Sami Taktak, débutée fin 2004. Ils ont mené à la réalisation d'un outil prototype *ODI-The deadlockfinder*, utilisé lors de la conception d'une architecture de réseau et de la fonction de routage associée, afin de s'assurer que la solution proposée est exempte d'interblocage. Plusieurs problèmes théoriques restent à explorer.

## 4.1 Détection de deadlock dans les NoCs - état de l'art

### 4.1.1 Hypothèses et résultats classiques

Nous considérons des réseaux d'interconnexion composés d'un ensemble de routeurs interconnectés par des canaux de communications. Un routeur peut émettre un nouveau message (s'il est connecté à un noeud source), il transmet un message d'un de ses canaux d'entrée vers un de ses canaux de sortie, il doit consommer le message (s'il est raccordé au noeud destinataire du message). La fonction de routage, associant à un message provenant sur un canal d'entrée un canal de sortie, est de type *wormhole* (le message est scindé en unités d'échange élémentaire (*flit*) ; tous les flits d'un même message suivent le même parcours ; à un instant donné, Les flits d'un même message peuvent être en transit sur plusieurs routeurs) ; elle peut être déterministe (pour un message vers un destinataire donné et une configuration d'environnement donnée, elle propose un unique canal de sortie par routeur) ou adaptative. Ces hypothèses sont classiquement utilisées dans la littérature.

Dans ce contexte, un réseau contient un deadlock s'il présente une configuration de messages dans ces canaux de communication qui ne peut plus progresser. Un réseau est dit *deadlock-free* (exempt de deadlock) si à partir d'un réseau vide, une configuration de deadlock ne peut jamais se produire.

Les résultats assurant la construction de fonctions de routage exemptes de blocage sont les suivants :
- Pour les fonctions de routage déterministe, Dally et Seitz ont établi qu'un réseau est deadlock-free si et seulement si le graphe des dépendances induit par la fonction de routage est sans cycle [60].
- Pour les fonctions de routage adaptatif, cette condition est assouplie. Le graphe de dé-

pendances[1] induit par la fonction de routage peut contenir des cycles *s'il existe une sous-fonction de routage permettant de véhiculer tout message, qui elle est sans cycle* [64].

Cela signifie que pour des fonctions de routage non adaptatives, les configuration d'interblocage *doivent être évitées* car il n'y a pas de possibilité de résorbtion du deadlock. Pour les fonctions de routage adaptatives, des configurations d'interblocage *peuvent apparaître*, mais dans ces configurations, il doit toujours être possible de résorber l'interblocage en empruntant des *canaux d'échappement*. Duato propose une approche constructive : à partir d'une fonction de routage deadlock-free, il augmente le degré d'adaptativité des routeurs en ajoutant des canaux. Par construction, la fonction de routage obtenue est deadlock-free (la fonction deadlock-free initiale est la sous-fonction de routage par les canaux d'échappement. Ces résultats ont été étendus par Fraigniaud et Fleury dans [69] à diverses classes de fonctions de routage. Schwiebert et Jayasimha [114] ont proposé une autre condition nécessaire et suffisante garantissant l' absence de deadlock pour des fonctions de routage dépendant, pour le routeur courant, du canal d'entrée et de la destination. Une synthèse des résultats est proposée dans [85], elle étend notamment les échanges à des trafics multicast (une source, plusieurs destinations). A notre connaissance, il n'existe pas de résultats étendant ces conditions d'absence de blocage lorsque les réseaux véhiculent plusieurs types de messages interdépendants.

### 4.1.2 Limitations

Les hypothèses énoncées précédemment ne permettent pas de rendre compte de dépendances entre messages. Dans de nombreux protocoles, la réception sur un noeud d'un message de type $X$ induit l'émission, par le même noeud, d'un message de type $Y$. Les capacités de stockage des noeuds n'étant pas infinies, il peut arriver qu'un message de type $X$ arrivant à destination ne puisse être consommé, parce qu'il attend que le message de type $Y$ progresse dans le réseau ; ce dernier peut lui même être bloqué dans sa progression par un message de type $X$. On est alors confronté à une situation d'interblocage. L'hypothèse "le noeud destination consomme toujours le message qui lui est destiné" n'est pas respectée, et les résultats théoriques établis par Duato puis étendus par Fraignaud puis Schwiebert ne sont plus applicables directement.

### 4.1.3 Une solution pratique

Une solution communément utilisée par les architectes de réseaux consiste à construire un réseau d'interconnexion (deadlock-free) *par type de message* [48] [70] [103]. De ce fait, les messages de type $X$ ne peuvent pas être bloqués par des messages de type $Y$, lesquels pourront toujours être acheminés vers leur destination et consommés.

Cette solution évite le partage des ressources entre messages de types différents. Elle a l'avantage de la simplicité, mais l'inconvénient de répliquer le matériel. Nous montrons que le partage de ressources qu'autorise le résultat de Duato peut être mis à profit dans le cas de protocoles à plusieurs types de messages.

[1] Les dépendances impliquées sont plus complexes que dans le cas de fonction de routage déterministe, cf. section 4.2.2

## 4.2 Prise en compte des dépendances liant plusieurs types de messages

### 4.2.1 Hypothèses supplémentaires

En sus des hypothèses communément admises, rappelées dans la section 4.1.1, nous introduisons trois hypothèses pour prendre en compte les dépendances entre messages de types différents.

1. Les messages sont typés. L'ensemble des types de messages est ordonné suivant un ordre partiel $<$. Soient $t$ et $t'$ deux types distincts de messages, si $t < t'$, alors le message de type $t$ est dit *plus prioritaire* que le message de type $t'$.
2. Lorsqu'un message arrive à son noeud destination, il est consommé *sous condition*. Seuls les messages de niveau de priorité le plus élevé sont consommés sans condition.
3. Le chemin suivi par un message, du noeud source au noeud destination dépend de sa destination, *de son type*, et de l'état des canaux de sortie du routeur courant. Sur chaque routeur, une fonction de routage adaptative définit l'ensemble des canaux de sortie empruntable par un message. Une fonction de sélection détermine un unique canal libre parmi l'ensemble des canaux retournés par la fonction de routage. S'il n'y a aucun canal disponible, le message en transit est placé en attente jusqu'à ce qu'un canal se libère.

Le premier item introduit la notion de type et d'ordre partiel entre les types de message. Intuitivement, lors de l'apparition d'une configuration de deadlock, les messages les plus prioritaires seront évacués en premier par des canaux d'échappement qui leur sont dédiés, puis les messages de priorité inférieure seront évacués par des canaux d'échappement empruntables uniquement par des messages de priorité supérieure ou égale à la leur (les messages plus prioritaires étant évacuables par des canaux qui leur sont propres, il n'y a pas de cause de deadlock entre messages de types différents).

Le second item impose que les messages évacués en premier soient consommés par leur noeud destination. Cela évite les dépendances cycliques entre types de messages.

Le troisième item précise la nature de la fonction de routage. Outre les éléments classiquement considérés (routeur courant, noeud destination), celle-ci dépend maintenant du *type* de message à véhiculer.

### 4.2.2 Théorème de Duato

Le théorème de Duato établit une condition nécessaire et suffisante garantissant l'absence de deadlock dans les réseaux à routage wormhole adaptatifs. Pour se faire, il définit un graphe de dépendances étendues, entre les canaux du réseau d'interconnexion, relativement à la fonction de routage $R$ et à une de ses sous-fonction $R_1$. Il définit quatre types de dépendances entre canaux.

- dépendance directe : il y a une dépendance directe de $c_i$ vers $c_j$ s'il existe un message transitant par $c_i$ qui peut être transmis à $c_j$. $c_i$ et $c_j$ sont fournis par $R$.
- dépendance indirecte : il y a une dépendance indirecte de $c_i$ vers $c_j$ s'il existe un message transitant par $c_i$ qui peut être transmis à $c_j$ *au travers de canaux non fournis par* $R_1$. Pour ce message, $c_i$ et $c_j$ sont fournis par $R_1$.

- dépendance croisée : il y a une dépendance croisée de $c_i$ vers $c_j$ s'il existe un message transitant par $c_i$ qui peut être transmis à $c_j$. $c_i$ n'est pas fourni par $R_1$, mais $c_j$ l'est.
- dépendance indirecte croisée : il y a une dépendance indirecte de $c_i$ vers $c_j$ s'il existe un message transitant par $c_i$ qui peut être transmis à $c_j$ *au travers de canaux non fournis par* $R_1.c_i$ n'est pas fourni par $R_1$, mais $c_j$ l'est.

Le Graphe de dépendances étendues associé à une sous-fonction de routage $R_1$ est noté $EDG(R, R_1)$. C'est un graphe orienté. Ses noeuds représentent les canaux du réseau, et les arcs représentent les dépendances directes, indirectes, croisées, et indirectes croisées induites par la sous-fonction de routage $R_1$ relativement à $R$.

Le théorème de Duato établit le résultat suivant.

**Théorème 3 (Théorème de Duato [64])** *Une fonction de routage cohérente, connexe et adaptative $R$ est deadlock-free ssi il existe une sous-fonction de routage connexe $R_1$ telle que $EDG(R, R_1)$ est sans cycle.*

### 4.2.3 Extension à plusieurs types de messages

Nous avons un ordre sur les types de messages, et une fonction de routage $R$. Nous cherchons à construire une fonction de routage $R_t$ par type de message $t$. $R_t$ est définie de telle sorte qu'elle ne peut partager des canaux d'échappement qu'avec des messages de type qui lui sont strictement plus prioritaire.

**Définition 7** *$R_t$ est une sous-fonction de routage de $R$ définie telle que :*

$$\forall\, x, y \in N,\ \forall\, t, t' \in T,\ t < t'$$
$$R_t(x, y) \subseteq R(x, y, t)$$

*telle que*

$$\forall c \in R_t(x, y), c \in R(x, y, t)\ \textit{et}\ c \notin R(x, y, t')$$

*i.e. les canaux fournis par $R_t$ ne peuvent véhiculer que des messages de type $t$ ou d'un type de priorité supérieure.*

Muni de cette fonction de routage particulière, nous avons établi le théorème suivant [120].

**Théorème 4** *Soit $R$ une fonction de routage connexe et adaptative, et $T$ l'ensemble ordonné des types de message. Si $\forall\, t \in T$, $R_t$ est connectée et deadlock-free, alors $R$ est deadlock-free.*

L'idée de la preuve est la suivante : $R_t$ fournit un chemin d'échappement pour tout message de type $t$. Soit $t_0$ le type de messages le plus prioritaire. Si $R_{t_0}$ est deadlock-free, alors tous les messages de type $t_0$ peuvent être évacués du réseau en utilisant les canaux fournis par $R_{t_0}$. Alors, tout message de type $t_n$ tel que $t_0 < t_n$ et $\nexists\, t_k | t_0 < t_k < t_n$, peut être évacué aux travers de canaux fournis par $R_{t_n}$. En effet, les canaux fournis par $R_{t_n}$ ne peuvent contenir que des messages de type $t_0$ ou $t_n$. Puisque tous les messages de type $t_0$ peuvent être évacués par les

canaux fournis par $R_{t_0}$, nous pouvons les retirer. Alors si $R_{t_n}$ est deadlock-free, les canaux fournis par $R_{t_n}$ permettent d'évacuer les messages de type $t_n$. De proche en proche, tous les messages de tous les types peuvent être évacués, et le réseau est sans blocage.

### 4.2.4 Algorithme de détection de deadlock

Un algorithme de détection de deadlock applicable aux réseaux véhiculants des messages de types interdépendants se déduit directement de ce résultat : étant donné une fonction de routage $R$ et un ensemble ordonné de type de messages $T$, nous construisons une fonction de routage $R_t$ pour chaque type de message $t$, telle que $R_t$ satisfasse à la définition 4.5. Si pour chaque $R_t$, $R_t$ contient une sous-fonction de routage $R_{1_t}$ tel que $EDG(R_t, R_{1_t})$ est sans cycle, alors le réseau est exempt d'interblocage.

---

input : R,T
output : Is R deadlock-free ?

1- split $R$ into $R_t$
2- check for each $t$ : $R_t$ is connected and satisfies def. 4.5, if not reject $R$.
3- for each $R_t$ : find $R_{1_t} \subseteq R_t$ such that its EDG contains no cycle.
4- if step 3 succeded, $R$ is deadlock-free else $R$ may induce an unsolvable deadlock.

---

**Remarque :** En théorie, on pourrait avoir le choix pour construire $R_t$ à partir de $R$. Pour chaque type de message $t_i$, on pourrait choisir un sous-ensemble suffisant garantissant l'évacuation des messages de type $t_i$, et restreindre $R$ pour que les messages de $t_i$ n'empruntent plus les canaux surnuméraires. Cette approche revient à modifier $R$. De plus, le choix d'un sous-ensemble des canaux d'évacuation pour un type de message $t_i$ pourrait être remis en cause lors de la construction du sous-ensemble des canaux de type $t_j$ pour $i < j$. En pratique, nous découpons $R$ en $R_t$ en prenant l'ensemble maximal des canaux répondant à la définition 4.5.

## 4.3 Notre algorithme de preuve d'absence de deadlock

Nous nous sommes intéressés à la résolution de l'étape 3 de l'algorithme précédent. Nous cherchons à extraire de $R_t$ une sous-fonction $R_{1_t}$ permettant l'échappement des messages de type $t$. Initialement, la fonction $R_{1_t}$ est équivalente à $R_t$. On va chercher à restreindre $R_{1_t}$. Partant de l'EDG de $R_t$ relativement à $R_{1_t}$, nous souhaitons supprimer des arcs de l'EDG pour briser tous les cycles de l'EDG. La suppression d'arc dans l'EDG correspond à la *restriction* de $R_{1_t}$. Si l'on obtient une restriction de $R_{1_t}$ connexe associée à un EDG sans cycle, alors $R_t$ n'induira pas de blocage. On ne peut évidemment pas retirer n'importe quel arc, sous peine de rompre la connexité de la fonction de routage. Par ailleurs, la restriction de $R_{1_t}$ peut induire de nouvelles dépendances indirectes, croisées et indirectes croisées.

L'algorithme standard, décrit par [114] énumère tous les cycles de l'EDG et supprime les arcs sans stratégie particulière. Le choix de suppression d'un arc à une étape donnée *peut être remis en cause ultérieurement*, ce qui oblige à reprendre le calcul des étapes précédentes. La complexité de cette solution est exponentielle.

Nous avons établi une condition *suffisante* garantissant l'absence de deadlock, permettant de retirer simplement des arcs de l'EDG. L'idée est la suivante : lorsque l'EDG contient un cycle (que l'on cherche à briser, en retirant un arc), nous recherchons, dans le cycle $\mathcal{C}$, un canal qui puisse être vidé indépendemment de la configuration des messages présents dans les canaux formant le cycle. Partant d'un canal $c_0$ du cycle $\mathcal{C}$, soit $c_k$ le canal de $\mathcal{C}$ le plus lointain de $c_0$ pouvant recevoir un message provenant de $c_0$. Si pour tous les messages véhiculés de $c_0$ à $c_k$, au travers de canaux $c_i$ (pour $0 \leq i \leq k$) appartenant à $\mathcal{C}$, ces messages peuvent être véhiculés par d'autres canaux (situés hors de $\mathcal{C}$), alors les arcs entre $c_0$ et les $c_i$ peuvent être supprimés. Cette condition, exprimée pour un cycle, peut être étendue à toute une composante fortement connexe.

La formulation exacte de cette condition, ainsi qu'un schéma de preuve garantissant l'absence de deadlock sont donnés dans [120].

En pratique, la restriction de $R_t$ s'effectue en manipulant les composantes fortement connexes de l'EDG plutôt que chaque cycle de l'EDG (la composante fortement connexe regroupe tous les cycles ayant des canaux communs). L'algorithme complet, ainsi qu'un exemple détaillé sont donnés dans [120].

Notre solution se distingue de l'algorithme standard sur les points suivants :
- Bien qu'exponentielle dans le pire cas, notre solution tire partie du partage des cycles en travaillant directement sur les composantes fortement connexes plutôt qu'en énumérant les cycles (l'énumération des cycles est exponentielle), la construction des composantes fortement connexes est polynomiale (cf. algorithme de Tarjan, [121]).
- Elle supprime les arcs suivant un critère raisonnable ; le retrait d'un arc n'entraine jamais de perte de la connexité (la reconstruction de l'EDG et le test de bonne connexité doivent être réeffectués pour chaque retrait d'arc avec la solution standard, et pas pour notre algorithme).
- Notre critère de suppression d'un arc est basé sur une condition suffisante garantissant l'absence de deadlock. Il se peut qu'avec notre solution, un cycle ne puisse pas être brisé car il ne satisfait pas à notre condition, mais qu'en réalité il n'induise pas de deadlock. Notre algorithme concluera que la fonction de routage peut produire un deadlock, alors qu'en réalité il n'en est rien.

## 4.4 Résultats expérimentaux

Cette section présente les résultats expérimentaux obtenus pour l'analyse automatique de deadlock. Différentes topologies et fonctions de routage réelles ont été utilisées. L'algorithme standard et l'algorithme que nous proposons ont été comparés. Une partie de ces résultats ainsi qu'une analyse détaillée de l'évolution des performances des algorithmes est décrite dans la version étendue de [120].

### 4.4.1 Élements de comparaison

Les résultats attendus de l'expérience portent sur plusieurs points :
- l'acuité de notre méthode de choix de l'arc à retirer, basé sur la condition suffisante exposée en sec. 4.3. Pour des topologies réalistes, notre méthode statuera-t-elle qu'un réseau exempt de blocage peut en contenir ?
- la comparaison, en temps de calcul et en occupation mémoire, de l'algorithme standard avec celui que nous proposons.

### 4.4.2 Ensemble de test

Nous avons considéré deux topologies combinées avec différentes fonctions de routage, certaines étant deadlock-free et d'autres non.

**La grille 2D**, utilisée dans DSPIN [103], est une grille $n \times n$ connectant chaque paire de voisins par deux liens unidirectionnels. Cette grille véhicule des messages de même type. Les fonctions de routage suivantes ont été évaluées :
- X-first. Les messages sont systématiquement routés le long de l'axe X puis de l'axe Y. C'est une fonction de routage non adaptative exempte de blocage pour un seul type de message.
- shortest path. Les messages suivent un des plus courts chemins reliant la source à la destination. C'est une fonction de routage adaptative pouvant conduire à des deadlocks.

La topologie 2D a été étendue, afin d'ajouter des chemins d'échappement. Chaque paire de voisins est connectée par deux liens unidirectionnels dans chaque direction. Pour chaque paire de lien liant deux voisins dans une direction donnée, l'un des liens est sélectionné suivant la stratégie shortest path, et l'autre selon la stratégie X first. Cet ensemble de seconds liens forme la sous-fonction de routage deadlock-free. Cette fonction de routage est nommée "Shortest path with escape path". Elle est adaptative et deadlock-free.

La topologie **Fat Tree** (utilisée dans le réseau SPIN [73] à interface VCI) connecte chaque noeud de niveau $i$ avec ses successeurs de niveau $i+1$ selon deux liens unidirectionnels. Le réseau véhicule deux types de messages : les *requêtes* et les *réponses*. Les messages de type *réponse* sont les plus prioritaires. Deux fonctions de routage ont été étudiées :
- Sans séparation : tous les canaux sont utilisables par les requêtes et les réponses. La fonction de routage est adaptative lorsque les messages transitent vers les niveaux inférieurs (vers les racines de l'arbre élargi) et déterministe lorsque les messages transitent vers les feuilles de l'arbre. Le fait de véhiculer plusieurs types de messages interdépendants peut conduire à des deadlocks.
- Avec séparation : l'ensemble des canaux est divisé en deux parties disjointes : l'une regroupant les canaux véhiculant les requêtes et l'autre les réponses. La fonction de routage est adaptative lorsque les messages transitent vers la racine, et déterministe lorsque les messages transitent vers les feuilles. Elle est deadlock-free.

### 4.4.3 Résultats et discussion

Les expériences ont été menées sur un Pentium4 cadencé à 3 GHz muni de 512 Mo de RAM.

La table 4.1 synthétise les performances obtenues par l'algorithme standard et l'algorithme que nous proposons pour les cinq exemples.

Tab. 4.1 – détection de deadlock selon les algorithmes SCC ou standard

| | | | our algorithm | | standard algorithm | | |
|---|---|---|---|---|---|---|---|
| messages | $I$ | $R$ | # of nodes | time | # of nodes | time | deadlock-free |
| 1 type | mesh | XY | 4900 | 1624.80s | 3600 | 1314.80s | yes |
| | | SP | 3025 | 1342.52s | 100 | 6.03s | no |
| | | SPEP | 256 | 19.68s | 100 | 21.24s | yes |
| 2 types | fat tree | NSEP | 256 | 21.83s | 256 | 43.40s | no |
| | | SEP | 256 | 11.29s | 256 | 21.94s | yes |

Les points suivants sont notables :

- La condition suffisante est assez fine pour ne pas rejeter abusivement de réseaux : toutes les topologies exemptes de deadlocks ont été identifiées comme telles, et les topologies pouvant mener à des deadlocks ont bien été reconnues.
- Dans tous les cas, l'algorithme que nous proposons est plus performant que l'algorithme standard. Qu'il soit appliqué avec une fonction de routage déterministe ou adaptative, dans une configuration à un seul type de messages ou à plusieurs, il permet de traiter des topologies contenant plus de routeurs que l'algorithme standard.
- La taille du réseau (en nombre de routeur) n'est pas le facteur déterminant de la complexité de l'algorithme. C'est plutôt la complexité de la fonction de routage qui importe (cf. test SPEP). Même dans ces cas, l'algorithme que nous avons proposé permet de traiter des réseaux réalistes de taille significative.

## 4.5 Prolongements possibles

A partir de ces travaux, un outil de vérification prototype a été développé. Il a permis de vérifier l'absence de deadlock pour certains réseaux classiques de taille significative, lorsque ceux-ci véhiculent plusieurs types de messages.

Plusieurs points théoriques restent à approfondir. Citons notamment :

- le partitionnement de $R$ en $R_t$ respectant la définition 4.5. Comme mentionné dans la remarque, actuellement nous construisons les $R_t$ à partir de $R$ sans modifier $R$. En fait, $R$ peut être "trop large" au sens ou elle offrirait trop de partage pour des messages de types différents. Dans ce cas, le découpage de $R$ en $R_t$ peut ne pas respecter la définition ; la fonction de routage est rejetée (on considère qu'elle peut contenir un deadlock). Il pourrait être intéressant de proposer une fonction de routage $R'$, restreignant $R$, pour laquelle il existe un découpage en $R'_t$ exempt de deadlock. L'objectif est alors modifié : on ne vérifie

plus seulement qu'une fonction de routage $R$ est exempte de blocage, on synthétise une fonction de routage $R'$ qui est sans blocage à partir de $R$ qui peut en contenir.
- la qualification précise de la condition exposée en sec. 4.3 : est-elle seulement suffisante ? Pour statuer sur ce cas, il nous faut trouver une configuration de messages qui ne satisfasse pas cette condition et qui ne provoque pas d'interblocage. Par la suite, nous souhaitons rechercher des critères plus fins pour la condition d'apparition de deadlocks.
- l'extension de la méthode à d'autres classes de fonctions de routage. Les fonctions de routages considérées dépendent du noeud courant, du destinataire et du type de message. Les extensions du théorème de Duato proposées par Fraignaud ou Schwiebert peuvent également être étendues à différents types de messages. La prise en compte du canal d'entrée dans la fonction de routage est un élément important pour l'élaboration de fonctions de routage pour des grilles 2D à trous.

D'un point de vue pratique, la mise à disposition pour les concepteurs de NoCs d'un outil de vérification automatique de deadlock ouvre un champ d'investigation élargi pour la conception d'algorithmes de routage. Cet outil offre la possibilité de tester différentes architectures et fonctions de routage dès les premières étapes de conception. Jusqu'alors, la définition de fonctions de routage répondait à un objectif de simplicité et de sécurité (routage déterministe, duplication du matériel pour éviter le partage de ressources entre messages de types différents). Ces solutions ne peuvent plus être appliquées directement : les topologies régulières souffrent d'irrégularités dues aux défauts de fabrication et le routage classique *X-first* ne peut plus s'appliquer ; dans le cas de protocoles impliquant plusieurs types de messages, la duplication systématique de matériel peut s'avérer trop coûteuse. Dans ce contexte, disposer d'un outil permettant de s'assurer rapidement de la validité de nouveaux choix architecturaux combinés à des fonctions de routage intensifiant le partage est une aide précieuse.

L'outil *ODI-The deadlock finder* sera utilisé dans les mois à venir dans le cadre de deux travaux menés au LIP6 :
- L'élaboration d'un algorithme de routage déterministe paramétré pour grille 2D à trou (dans le cas de transfert d'un seul type de message).
- La définition d'un protocole de sécurisation des transferts de messages sur le réseau DSPIN (routage Xfirst avec 4 types de messages. but : limiter la réplication du matériel)

Dans les deux cas, l'outil permettra de déterminer si une solution originale, combinant une topologie particulière avec des contraintes architecturales et une fonction de routage ad-hoc, est bien exempte d'interblocage. A l'avenir, cet outil pourrait être une brique de base d'un environnement de conception de haut niveau de réseaux sur puce.

# Troisième partie

# Vérification des contraintes temporelles de composants

# Chapitre 5

# Vérification des contraintes temporelles de composants mémoires embarqués

Les travaux décrits dans cette partie ont été réalisés dans le cadre du projet MEDEA+ *Blueberries* (2004-2006). La tâche visait le développement de méthodes formelles pour la vérification des caractéristiques temporelles de composants mémoires embarqués. Les partenaires du projet pour cette tâche étaient STMicroelectronics (fourniture et analyse d'études de cas), TNI-VALYOSIS (fourniture d'un outil d'abstraction fonctionnelle) et le LSV (méthodes formelles).

Actuellement, les mémoires dans les circuits électroniques prennent une importance prépondérante : elles représentent actuellement jusqu'à 70% de la surface des circuits numériques embarqués (SOC) ; pour certaines applications complexes, jusqu'à 400 bloc-mémoires peuvent être intégrés sur un même circuit ; la performance globale d'un circuit dépend essentiellement du temps d'accès à ces mémoires. Par ailleurs, la complexité de ces mémoires provient non seulement de l'accroissement du nombre de point-mémoires contenus, mais surtout des nouvelles fonctionnalités offertes répondant aux exigences accrues des nouvelles applications (modes veille, changement de tension d'alimentation, mémoires associatives haute densité,...).

Pour répondre à ces exigences de performance, *les mémoires embarquées sont directement conçues au niveau transistor*. La dénomination de tels circuits est *full-custom*, par opposition à l'utilisation de bibliothèques de portes logiques standard, ayant des caractéristiques fonctionnelles et temporelles prédéterminées (ces derniers sont dénommés *standard-cell)*. En pratique, ces mémoires sont validées dans un processus de vérification basé essentiellement sur de la simulation électrique (e.g., voir [HSIM]). Ces simulations sont très coûteuses en temps, et ne permettent qu'une vérification parcellaire. A l'issue de cette simulation, peuvent subsister des dysfonctionnements qui ne seront détectés qu'après la fabrication. De telles erreurs nécessitent des mois d'efforts humains et matériels pour être localisées, puis corrigées. Ce retard compromet alors gravement le succès commercial du produit.

Pour ces raisons, d'autres voies que la simulation méritent d'être explorées afin d'accroitre la qualité du processus de validation, et de détecter les erreurs de conception au plus tôt. Nous proposons d'utiliser le modèle des automates temporisés paramétrés pour définir des conditions garantissant le fonctionnement correct de composants. Ces conditions très fines permettent de

comprendre les liens entre les délais des différents composants, et peuvent être utilisées à des fins d'optimisation sûres de certains délais. Nous proposons une méthode de résolution polynomiale permettant de garantir des plages de bon fonctionnement de circuits.

La section 5.1 présente différentes approches pour déterminer les caractéritiques temporelles d'un système embarqué, compare leur complexité et le degré de finesse du résultat d'analyse fourni. La section 5.2 décrit notre principale contribution dans ce contexte. La section 5.3 présente les vérifications temporelles qui ont pu être menées à bien sur la mémoire embarquée SPSMALL, conçue et commercialisée par STMicroelectronics. La section 5.4 brosse les principales perspectives et collaborations liées à ce travail.

## 5.1 Détermination des caractéristiques temporelles d'un composant embarqué

Les principales méthodes de détermination des caractéristiques temporelles d'un composant embarqué décrit sous la forme d'un ensemble de transistors ou d'un réseau de portes logiques sont présentées ci-après.

**La simulation électrique.** Elle s'applique pour un circuit décrit au niveau transistor (modèle SPICE [112]). Les modèles des transitors et leurs interconnexions (résistances et capacités) relient les grandeurs électriques (tension et courant en chaque point du circuit) par des équations intégro-différentielles. Le simulateur électrique résoud numériquement ces systèmes d'équations, pour déterminer l'évolution du courant et de la tension en différents points du circuit. La détermination est très précise : le simulateur ne fournit pas uniquement une valeur de tension, mais il détermine la courbe d'évolution de la tension au cours du temps (et donc la pente de ces courbes lors des changements de niveau). Cette évaluation permet d'analyser fonctionnellement le circuit, mais elle est surtout utilisée pour déterminer les temps de propagation d'un front d'un point à un autre du circuit, ainsi que la raideur des fronts propagés, ou l'apparition éventuelle de glitchs.

Cette analyse est incontournable pour caractériser les circuits *full-custom*. Elle est également très utilisée pour déterminer les temps de cycles minimaux pour les circuits *standard-cells*.

Le modèle d'un transitor CMOS est défini par une dizaine de paramètres (électriques, géométriques, technologiques). Ainsi, même pour des petites portions de circuit, les systèmes d'équations à résoudre sont de très grande taille. En pratique, les concepteurs sont amenés à élaguer une partie du circuit pour n'en soumettre qu'une portion au simulateur électrique. Le choix de la portion à analyser repose entièrement sur l'expertise du concepteur. Le choix des scénarii de tests est également crucial. Le délai de propagation d'un front dans une porte (un assemblage de transitors) dépend de l'environnement de cette porte : les valeurs des autres signaux d'entrée, d'éventuels effets parasites liés à la présence rapprochée d'autres lignes sur lesquelles se propagent d'autres fronts. Là encore, le choix des "vecteurs de test" appliqués repose entièrement sur l'expertise du concepteur.

**L'analyse statique** est une alternative à la simulation, et évite le problème de détermination

de jeux de tests pertinents. De plus, elle se positionne à un niveau plus abstrait que le modèle en transistors, et vise essentiellement la détermination des chemins critiques. Cette approche procède en deux étapes : l'abstraction fonctionnelle puis l'analyse de timing. La première identifie des regroupements de transistors comme des unités fonctionnelles, et détermine des minorants et majorants du délai de traversée d'un front d'une des entrées vers la sortie pour chaque unité fonctionnelle. L'analyse de timing établit alors un graphe de dépendances causales entre signaux de sortie des unités fonctionnelles, puis détermine la longueur de chaque chemin dans ce graphe. C'est l'analyse de timing *topologique*, dont la complexité est linéaire avec la taille du graphe[1]. Différentes procédures d'analyse hiérarchique ont été proposées ([87], [63]). Elle a été appliquée avec succès pour la réalisation de circuits complexes (plusieurs millions de transistors).

La non prise en compte de la *fonctionalité* des portes peut aboutir à une approximation trop grossière des délais globaux de propagation (notamment dus à la présence de *faux chemins*). C'est fréquemment le cas pour les systèmes soumis à des contraintes de performances très fortes, nécessitant une conception *full-custom*. Diverses techniques ont été proposées pour améliorer cette analyse statique topologique, visant à éliminer (partiellement) les faux chemins que l'analyse produit (i.e. *Path sensitization* de [88], analyse par *mode de fonctionnements* [126]). Ces améliorations consistent à réintroduire – au moins partiellement – la fonctionalité des portes logiques traversées.

**La simulation Min-Max** est utilisée pour déterminer les temps de propagation de fronts des signaux d'un réseau de portes logiques. Le temps de traversée de chaque porte est déterminé dans un intervalle. Il s'agit d'une simulation *logique* (et non plus électrique), prenant en compte la fonctionalité de chaque porte. Le réseau est supposé sans cycle (les éventuelles boucles combinatoires sont coupées). La détermination exacte des bornes de propagation des signaux pour ce modèle est un problème exponentiel (dans le nombre de délais et pour chaque vecteur d'entrée) dans le pire cas. Des approximations conservatives, de complexité polynomiale, ont été proposées (cf. [124], [46]) ; les bornes qu'elles estiment sont parfois trop pessimistes. En outre, ces méthodes permettent d'analyser et d'éliminer les faux glitches.

**Le problème "Time Separation of Events" (ou TSE).** La détermination du temps séparant deux événements dans un circuit composé de portes logiques est une instanciation du problème plus général de *Time Separation of Events*, très étudié dans la décade 1990. Le système temporisé à analyser est modélisé sous la forme d'un graphe de contraintes temporelles (*timing constraints graph*), dans lequel chaque noeud représente un événement, et les arcs représentent les dépendances causales entre ces événements[2]. Les arcs sont étiquetés par des délais de propagation prenant leur valeur dans des intervalles. Les noeuds sont affublés d'opérateurs *min* ou *max*. L'instant d'occurrence de l'événement $q$, $t_q$ est fonction de l'instant d'occurrence des événements sur les noeuds directement précédesseurs $p \in pred(q)$, des délais de propagations de $p$ à $q$, et de l'opérateur affublant le noeud $q$. $t_q = op_{p \in pred(p)}(t_p + \delta_{p,q})$ avec $op \in \{\min, \max\}$. Le problème consiste à trouver les bornes des temps de séparation des instants d'occurrence d'événements $i$ et $j$ du graphe. K. McMillan et D. Dill ont montré qu'il s'agissait d'un problème NP-complet en produisant une réduction au problème 3-SAT [97].

Diverses approximations polynomiales ont été proposées ([24],[41], [42]). Les temps de traver-

[1] pour des graphes sans cycle.

[2] Il s'agit ici des dépendances causales *fonctionnelles* et non pas topologiques, comme dans le cas de l'analyse statique.

sée de circuits asynchrones complexes (le "solveur" d'équations différentielles DIFFEQ [45] et le décodeur d'instructions asynchrone RAPPID [110]) ont pu être validés par ces approches ; similairement, différents problèmes temporisés d'allocation de ressources ont pu être traités [44]. De façon marginale, ce problème a été utilisé pour déterminer des conditions suffisantes garantissant le bon fonctionnement de circuits asynchrones [45].

Une remarque importante doit être formulée : *le graphe de contraintes temporelles n'est pas conditionnel.* On ne peut pas y exprimer des comportements conditionnels, du type "Si $e_1$ arrive avant $e_2$ au noeud $n$, alors générer l'événement $e_3$ sinon générer l'événement $e_4$". Cette conditionnelle peut être représentée par une disjonction de cas, sur deux graphes distincts (l'un représentant l'ordonnancement $e_1$ précédant $e_2$ et produisant l'événement $e_3$, et l'autre représentant l'ordonnancement $e_2$ précédant $e_1$ et produisant l'événement $e_4$)[3]. Le graphe de contraintes temporelles est suffisant pour représenter des circuits combinatoires sans boucle, ou pour des modèles concurrents de type *free-choice Petri Nets* ([83]). Par contre, l'introduction de latchs conduit à des modèles conditionnels. Nous verrons en section 5.3 comment nous avons traité ce point.

**L'accessibilité d'un modèle d'automates temporisés.** Les automates temporisés ont été proposés en 1990 par R. Alur et D. Dill [13]. C'est une extension des automates permettant de représenter d'une part les traitements à réaliser, et d'autre part la durée de ces traitements. Les délais des traitements sont comptabilisés par des variables particulières, *les horloges.* De très nombreuses études ont été réalisées sur ce modèle et ses extensions. J'en retiens un résultat important : l'accessibilité temporisée est décidable ; elle est basée sur la construction du graphe des régions, laquelle est de complexité exponentielle avec le nombre d'horloges. [118] définit le model-checking de ces systèmes. KRONOS [117] ou UPPAAL [90] sont des outils d'édition et d'analyse de systèmes décrits sous la forme d'automates temporisés concurrents. Leur particularité réside dans une représentation et une manipulation efficaces des graphes des états accessibles temporisés (plus communément appelés graphes des zones), grâce à la structure de données *Difference-Bounded Matrices, DBM.* Un tour d'horizon des principaux résultats concertant le modèle des automates temporisés et le model-checking associé est présenté dans [29]. L'outil HYTECH [77] quant à lui, permet la représentation et l'analyse de systèmes hybrides (dont les systèmes temporisés sont une sous-classe). En nous focalisant sur notre problématique, il permet l'analyse de systèmes temporisés paramétrés (dont les délais ne sont pas instanciés). L'accessibilité pour des automates temporisés paramétrés sans cycles est décidable ; les zones (éventuellement paramétrées) sont représentées par des unions de polyèdres convexes.

Dès 1995, O. Maler et A. Pnueli ont proposé l'utilisation des automates temporisés pour modéliser des circuits asynchrones décrits au niveau porte, et analyser leurs performances temporelles [95]. Ils adoptent le modèle *bi-bounded inertial delay* : les délais de traversée des portes sont inertiels et compris dans un intervalle entier. Avec l'outil KRONOS [117], ils déterminent les chemins critiques de circuits asynchrones comprenant entre 20 et 100 portes (ces résultats sont obtenus au prix d'approximations conservatives éliminant certains faux chemins [111]). Des résultats comparables peuvent être obtenus avec UPPAAL.

L'analyse du graphe d'accessibilité fournit des bornes exactes, et la complexité de la génération du graphe est comparable à la complexité de la simulation logique temporisée (exhaustive)

[3]Evidemment, le nombre de graphes à envisager croît exponentiellement avec le nombre de conditionnelles indépendantes.

ou au TSE. Pour être plus précis, la construction (ou le parcours à la volée) du graphe des états accessibles permet de construire (ou de parcourir à la volée) des graphes de contraintes temporisées *de systèmes conditionnels*. Ce peut être vu comme la construction simultanée de plusieurs graphes de contraintes temporisées de TSE avec résolution des choix. De façon générale, l'analyse de systèmes temps-réels décrits sous la forme d'automates temporisés n'est pas limitée à la détermination de chemins critiques. Elle permet la vérification de propriétés temporelles plus riches, exprimées dans une logique temporelle temporisée (TCTL par exemple).

De façon similaire, l'analyse du graphe d'accessibilité d'un réseau de Petri temporisé (ou d'autres modèles opérationnels) peut être utilisée à des fins de vérification ou de synthèse de circuits asynchrones [122] [56]. Dans ce contexte, plusieurs abstractions ont été proposées ([99], [128], [107]). Elles permettent de vérifier automatiquement l'absence de dysfonctionnement de circuits temporisés asynchrones. L'étude comparative présentée dans [99] montre que des circuits d'une vingtaine de portes sont analysables.

**Vers l'analyse de systèmes paramétrés.** L'analyse paramétrée de tels systèmes peut fournir une information beaucoup riche que l'analyse de modèles à délais entiers ou réels. Elle permet notamment d'exhiber des relations liant les délais et garantissant un fonctionnement correct du système. T. Amon, G. Borriello *et al.* ont proposé d'analyser des graphes de contraintes temporisés pour lesquels les délais sont donnés sous forme paramétrée. Une première méthode basée sur la programmation par contraintes (CLP) a été proposée dans [14]. Elle pose les bases de la vérification paramétrée et met en avant l'intérêt des résultats fournis par cette approche : l'explicitation du lien entre les délais des composants et les contraintes garantissant le bon fonctionnement peut être utilisée à des fins de vérification, d'optimisation ou de synthèse. Une seconde approche, basée sur l'utilisation de la logique de Presburger est décrite dans [15]. Les formes particulières de formules de Presburger induites par la nature du problème TSE paramétré limitent la complexité des procédures d'analyse. Dans ce dernier cas, l'application à des problèmes concrets n'a pas été abordée de façon convaincante.

R. Clariso et J. Cortadella proposent dans [54] une analyse paramétrée des temps de traversée de circuits combinatoires (avec boucles combinatoires pour les éléments mémorisants) utilisant des techniques d'accessibilité paramétrée munies de primitives de l'interprétation abstraite pour garantir la terminaison des calculs (widening notamment). De fait, cette approche permet d'extraire des ensembles de contraintes linéaires entre les délais, garantissant le fonctionnement correct du circuit. Elle a été appliquée sur un pipeline asynchrone ainsi que sur des exemples tests de la littérature comprenant une dizaine de portes et une quinzaine de signaux.

Nous avons appliqué une démarche similaire dans [50] (sans les techniques d'interprétation abstraite, inutiles dans notre cas) et avons pu extraire un ensemble de conditions suffisantes garantissant le fonctionnement correct de la mémoire embarquée SPSMALL, pour deux implémentations différentes de cette architecture, une optimisée pour un temps de réponse bref et l'autre pour une faible consommation. L'analyse repose sur la construction de l'ensemble des états accessibles paramétrés du système, décrit sous la forme d'une collection d'automates temporisés. L'introduction de contraintes liant les délais rend inaccessibles les zones de "mauvais fonctionnement". L'introduction de ces contraintes n'est pas automatique : elle nécessite une bonne connaissance du fonctionnement du circuit, ainsi qu'un découpage en plusieurs sous-parties qui sont raccordées manuellement. L'ensemble des contraintes obtenues nous a permis d'optimiser les temps de *setup* des signaux d'entrée du circuit. Les résultats obtenus ont été validés a posteriori

par simulation électrique.

Ces premiers résultats encourageants nous ont poussés à développer une méthode automatique d'extraction de ces contraintes. Cette méthode s'explique simplement comme la résolution d'un problème *inverse* du problème TSE. C'est l'objet de la section suivante. La démarche complète d'analyse de la mémoire SPSMALL est exposée ensuite.

## 5.2 Analyse du problème TSE inverse paramétré

### 5.2.1 Problème TSE inverse

On suppose donné un graphe de contraintes temporisées $G(N, E)$. $N$ est l'ensemble des noeuds qui peuvent être de trois types : *delay*, *min* ou *max*. $E$ est l'ensemble des arcs orientés. Un arc pointant d'un noeud $p$ vers un noeud de type *delay* $q$ est affublé d'un délai $\delta_{p,q}$, compris dans un intervalle entier. On note $\overline{\delta}$ le vecteur des délais $\delta_{p,q}$ pour tous $p$ et $q$. $G$ contient un unique noeud source $s$. On note $t_q$ l'instant d'occurrence de l'événement $q$. $t_q$ est défini tel que :

- si $q$ est un noeud *delay* autre que la source, alors il a un unique prédécesseur $p$ et $t_q = t_p + \delta_{p,q}$
- si $q$ est un noeud *min*, alors $t_q = \min_{p \in pred(q)}(t_p)$
- si $q$ est un noeud *max*, alors $t_q = \max_{p \in pred(q)}(t_p)$

Le problème TSE *direct* consiste à déterminer pour chaque paire de noeuds $(i, j)$ du graphe les bornes de l'intervalle défini par $\mid t_i - t_j \mid$. La détermination du temps de réponse maximal est donnée par $\max_j(\mid t_j - t_s \mid)$. Ce temps de séparation correspond à une somme de délais d'arêtes consécutives du graphe, reliant le noeud source $s$ au noeud $j$.

Le problème TSE *inverse* suppose donné le graphe G *et un ordre total $\mathcal{O}$ sur les événements du graphe*. Cet ordre correspond à une exécution possible du système. On cherche alors à déterminer le domaine $\Delta$ des intervalles de valeurs des délais garantissant que le système fonctionnera en respectant l'ordre $\mathcal{O}$.

En fait, un ordre total caractérise un ensemble de traces d'exécutions temporelles équivalentes. Un mode de fonctionnement (associé à un chemin critique) est déterminé par un ordre partiel des événements du graphe. Une fois les bornes des intervalles garantissant l'exécution selon un ordre total obtenues, on pourra *relâcher* certaines de ces bornes, tout en préservant le mode de fonctionnement du système.

A notre connaissance, peu de travaux ont été publiés sur ce problème inverse. T. Burks et K. Sakhalla, dans [38], cherchent à minimiser le temps de cycle de circuits synchrones en modélisant le système sous la forme de problèmes d'optimisation de programmation linéaire avec contraintes *min* et *max*. La résolution s'appuye sur des techniques standard d'optimisation combinatoire (branch and bound), avec diverses améliorations (simplex). Du fait de la complexité de la procédure (exponentielle), seuls des systèmes de taille réduite sont analysables. S. Chakraborty *et al.* utilisent leur méthode de résolution approchée d'analyse de délais pour donner des garanties de *bon fonctionnement* du circuit. Les garanties portent sur des valeurs extrêmes de temps de

*setup* des signaux d'entrée. Le bon fonctionnement du circuit est ici entendu comme un circuit ne présentant pas de glitch [43]. Ce résultat est présenté comme un produit dérivé de leur méthode d'analyse basée sur la simulation min-max, et n'est pas défini comme un problème de TSE.

### 5.2.2 La méthode

On suppose donnés $G$ un graphe de contraintes temporelles et $\mathcal{O}$ un ordre total sur les événements de $G$. La méthode de résolution du problème TSE inverse se décompose en trois étapes :

1. **Construire le système d'inéquations** $\mathcal{I}$ : pour chaque inéquation de $\mathcal{O}$ (de type $t_i < t_j$), construire $\Sigma_{\pi_i} < \Sigma_{\pi_j}$ (de type $\delta_{s,k} + \cdots + \delta_{l,i} < \delta_{s,m} + \cdots + \delta_{n,j}$) et simplifier (supprimer les termes communs des côtés gauche et droit). La construction de $\mathcal{I}$ est polynomiale avec la taille du graphe.
2. **En dériver le système d'inéquations** $\mathcal{J}$ : pour chaque inéquation de $\mathcal{I}$ (de type $\delta_{s,k} + \cdots + \delta_{l,i} < \delta_{s,m} + \cdots + \delta_{n,j}$), remplacer chaque terme $\delta_{i,j}$ de la partie gauche par une variable $y_{i,j}$ et chaque terme $\delta_{k,l}$ de la partie droite par une variable $x_{i,j}$. La construction de $\mathcal{J}$ est linéaire avec le nombre d'inéquations (lui même linéaire avec le nombre de sommets de $G$).
3. **Résoudre** $\mathcal{J}$ : les solutions $\overline{l}$ (en $y$) et $\overline{u}$ (en $x$) délimitent une zone $\Delta$ pour laquelle l'ordre $\mathcal{O}$ est respecté. On cherche à construire la zone $\Delta$ la plus vaste possible, garantissant l'ordre $\mathcal{O}$. C'est un problème d'optimisation combinatoire en nombres entiers, de complexité polynomiale avec le nombre d'inconnues de $\mathcal{J}$ (lui même polynomial avec le nombre de sommets de G).

### 5.2.3 Optimisation

Il peut être intéressant d'élargir autant que possible le domaine $\Delta$. En fait, élargir $\Delta$ revient à considérer d'autres ordres d'événements (garantissant toujours un même mode de fonctionnement). Plusieurs méthodes d'élargissement ont été envisagées :

- Prendre un nouvel ordre $\mathcal{O}'$, appliquer la méthode pour obtenir $\Delta'$. Réaliser l'union des domaines. Cette union peut ne pas être convexe. Prendre l'enveloppe convexe de l'union peut introduire des points ne garantissant ni $\mathcal{O}$ ni $\mathcal{O}'$ ; on peut éliminer ces derniers en restreignant $I \cup I'$, i.e. en introduisant des contraintes délimitant l'enveloppe convexe des points à exclure.
- Supprimer une ou plusieurs inégalités de $\mathcal{I}$ en s' assurant que cela n' induit pas de nouveau chemin critique. Si de nouveaux chemins critiques sont ajoutés, on peut exclure les points induisant les nouveaux chemins critiques en introduisant de nouvelles contraintes. On ajoute des inéquations délimitant l'enveloppe convexe des zones de points à exclure. C'est l'approche que nous avons utilisée dans [50].

Ces méthodes d'élargissement ont un coût exponentiel. Néanmoins, sur les exemples testés, elles se sont avérées utiles et efficaces.

### 5.2.4 Implémentation

Cette méthode inverse a été implantée dans le formalisme des automates temporisés paramétrés, avec l'outil HYTECH. Le graphe de contraintes du TSE est transformé de la façon suivante : chaque noeud de $p$ de $G$ correspond à un automate temporisé $A_p$, (implantant un délai, une fonction *min* ou une fonction *max*). Chaque arête reliant deux noeuds $i$ et $j$ correspond à une transition synchronisée entre les automates $A_i$ et $A_j$. Une horloge globale permet d'enregistrer l'instant d'occurrence de chaque événement. On peut alors inférer l'ordre $\mathcal{O}$ pour une valeur particulière de chaque délai du TSE, puis en déduire le système d'inéquations $\mathcal{I}$. La dérivation de $\mathcal{J}$ et l'obtention d'une solution optimale pour $\mathcal{J}$ (i.e. $\Delta$) est faite manuellement (mais est automatisable). La relaxation de $\Delta$ est également réalisée avec HYTECH, par une analyse du graphe des états accessibles paramétrés.

## 5.3 Application à la mémoire embarquée SPSMALL

Cette section présente l'analyse temporelle de la mémoire embarquée SPSMALL que nous avons étudiée dans le cadre du projet MEDEA+ BLUEBERRIES. La mémoire SPSMALL est un circuit construit à partir d'un générateur paramétré (en nombre et taille des mots stockés) et commercialisé par STMicroelectronics. Plusieurs capacités de stockage sont disponibles : la capacité varie de 3 mots de 2 bits à 512 mots de 128 bits ; pour chacune d'entre elles, deux versions sont proposées au public : l'une optimisant la vitesse et l'autre la consommation.

La mémoire peut effectuer deux opérations : *écrire* une donnée dans une case mémoire désignée par une adresse, ou bien *lire* la donnée préalablement stockée dans une case mémoire spécifiée par une adresse. La mémoire est *write-through* : lorsqu'une donnée est écrite dans une case mémoire, elle est également produite sur le port de sortie. L'interface du circuit désigne ses ports d'entrée et de sortie. Cette interface contient, pour une mémoire de $m$ mots de $n$ bits (entre autres) :

**ports d'entrée :** le signal d'horloge `CK`, l'adresse sur $p = log_2 m$ bits `A[0..p-1]`, la donnée sur $n$ bits `D[0..n-1]`, l'ordre de lecture ou d'écriture `WEN`.
**ports de sortie :** la donnée produite sur $n$ bits `Q[0..n-1]`.

La mémoire a un fonctionnement asynchrone, mais elle est plongée dans un environnement synchrone. Le front montant du signal d'horloge `CK` est le point de référence temporelle. La fréquence d'horloge $t_{cycle} = t_{HI} + t_{LO}$, les temps de *setup* (et hold) de chaque signal d'entrée $t_{setup_D}$, $t_{setup_A}$, $t_{setup_W}$, et le temps de réponse pour chaque opération $t_{max}^{write}$ et $t_{max}^{read}$ sont données par le constructeur dans la *spécification du composant* (ou datasheet). Ces données définissent le fonctionnement nominal de la mémoire : si la fréquence d'horloge et les temps de setup et hold sont respectés, le constructeur garantit que la mémoire fournira une réponse dans un temps borné par le temps de réponse spécifié. Concrètement, cette spécification est établie par simulation électrique d'une portion du modèle en transistor du circuit.

**Notre objectif :** On cherche à minimiser les valeurs des temps de *setup* de cette architecture, tout en garantissant sa bonne fonctionnalité. Pour se faire, nous allons déterminer les relations liant les paramètres temporels externes de la mémoire aux délais de traversée des composants

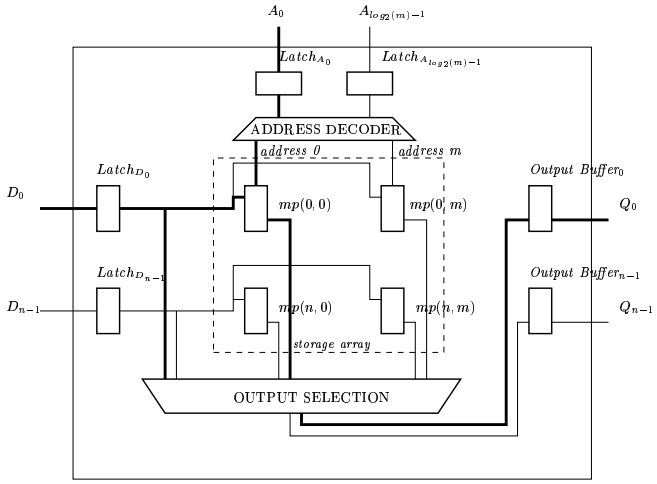


FIG. 5.1 – Architecture de la mémoire $m \times n$ et réduction à une mémoire $1 \times 1$

internes, puis résoudre un problème inverse de TSE.

### Etape 1- Modélisation de l'architecture au moyen d'un AFTG

La première étape consiste à élaborer d'un modèle formel combinant la fonctionalité des composants internes, et les délais de propagation des fronts de signaux au travers de ces composants. Cette modélisation comprend trois points :

- La réduction de la mémoire de $m$ mots de $n$ bits à une mémoire de 1 mot de 1 bit. On attribue au point mémoire conservé les caractéristiques suivantes : il induit la chaine de décodage et le temps d'accès les plus longs. La figure 5.1 montre l'architecture de la mémoire et sa réduction[4].
- L'abstraction fonctionnelle du modèle en transitors permettant la construction d'un *Graphe Fonctionnel Abstrait* (AFG en anglais). Les noeuds du graphe sont de trois types : les portes logiques, les latchs (ou bascules transparentes), les wires (ou équipotentielles). L'abstraction est réalisée automatiquement au moyen de l'outil TLL [7] de TNI-VALYOSIS.
- La détermination des délais de propagation des fronts au travers des éléments constitutifs de l'AFG, et leur report sur le graphe qui devient un *Graphe Fonctionnel et Temporel abstrait*, ou AFTG. Le modèle temporel retenu pour les wires et les latchs est le modèle *inertiel* et *bi-borné*, distinguant la propagation d'un front descendant de celle d'un front montant. Le délai de traversée d'une porte pour le changement d'un signal d'entrée est reporté sur le wire entrant. Ainsi, chaque wire ou latch $k$ de l'AFG se voit affublé de deux délais $\delta_k^{\uparrow} \in [l_k^{\uparrow}, u_k^{\uparrow}]$ et $\delta_k^{\downarrow} \in [l_k^{\downarrow}, u_k^{\downarrow}]$. Les valeurs de $l_k^{\uparrow}, u_k^{\uparrow}, l_k^{\downarrow}$ et $u_k^{\downarrow}$ ont été déterminées par simulation électrique[5] au moyen de l'outil HSIM [9].

L'AFTG pour la version "high-speed" de SPSMALL est donné sur la figure 5.2. Les valeurs numériques des délais internes et de la spécification correspondent à une mémoire de 3 mots de 2 bits [49].
$\delta_0^{\uparrow} \in (94,95)$, $\delta_0^{\downarrow} \in (65,66)$, $\delta_1^{\uparrow} \in (13,14)$, $\delta_1^{\downarrow} \in (17,18)$, $\delta_2^{\uparrow} \in (23,24)$, $\delta_2^{\downarrow} \in (29,30)$, $\delta_3^{\uparrow} \in (5,5)$, $\delta_3^{\downarrow} \in (2,2)$,

[4]La réduction a été faite manuellement après extraction fonctionnelle. Cette étape est automatisable.
[5]Cette étape est automatisable, c'est une partie du projet ANR *VALMEM*.

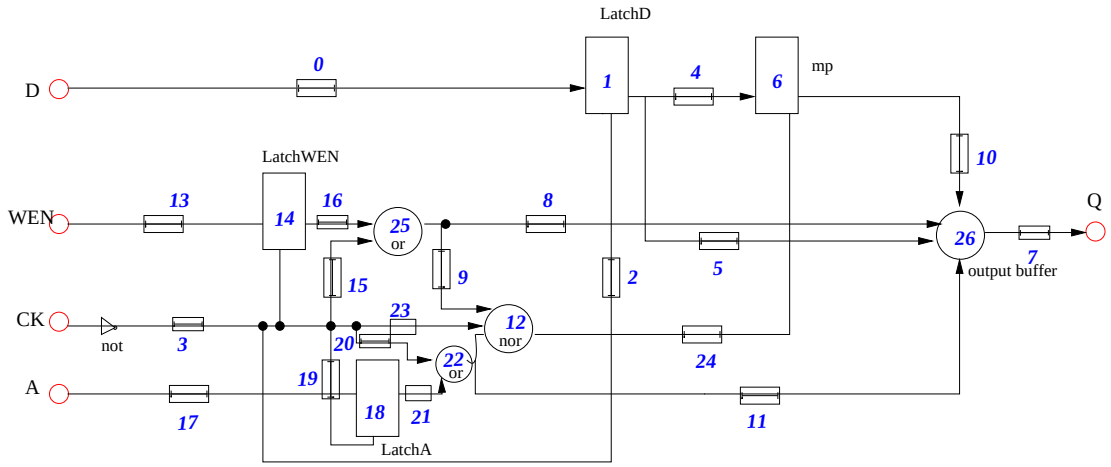


FIG. 5.2 – AFTG pour la version "high speed" de SPSMALL

$\delta_5^{\uparrow} \in (21,22)$, $\delta_5^{\downarrow} \in (44,45)$, $\delta_7^{\uparrow} \in (20,21)$, $\delta_7^{\downarrow} \in (19,20)$, $\delta_8^{\uparrow} \in (0,0)$, $\delta_8^{\downarrow} \in (21,22)$, $\delta_{13}^{\uparrow} \in (10,11)$, $\delta_{13}^{\downarrow} \in (8,8)$, $\delta_{14}^{\uparrow} \in (21,22)$, $\delta_{14}^{\downarrow} \in (21,22)$, $\delta_{15}^{\uparrow} \in (13,14)$, $\delta_{15}^{\downarrow} \in (10,11)$, $\delta_{16}^{\uparrow} \in (23,24)$, $\delta_{16}^{\downarrow} \in (0,0)$, $d_{HI} = 36$, $d_{LO} = 74$, $\delta_{setup_D} \in (108,?)$, $\delta_{setup_W} \in (48,?)$.[6] De plus : $t_{max}^{write} = 56$.

## Etape 2 - Caractérisation de l'expérience

La mémoire présente deux modes de fonctionnement : la *lecture* et l'*écriture*. Une expérience est la donnée d'un scénario d'application des signaux d'entrée produisant une sortie, sur un nombre borné de cycles.

Le mode de fonctionnement d'écriture combine deux mécanismes : l'écriture de la donnée dans le bon point mémoire et la recopie de la donnée d'entrée sur la sortie (*write-through*). L'étude du mécanisme write-through nécessite deux expériences : l'une consistant à imposer l'écriture d'un 1 logique, qui doit se retrouver sur le port de sortie après un délai maximal $t_{max}^{write}$ donné par la spécification (expérience W1), et l'autre consistant à écrire un 0 (expérience W0). La figure 16 montre le chronogramme associé à la première expérience. Celle-ci met en jeu une portion de l'AFTG (les buffers d'entrée et le mécanisme *write-through*), représenté sur la figure 5.4. Le chemin critique pour ce mode de fonctionnement est représenté en rouge.

Le temps de réponse pour cette expérience, $t_{CK \to Q\uparrow}$ correspond à la somme des délais des composants traversés par le chemin critique (noté $\Pi$ par la suite).

$$t_{CK \to Q\uparrow} = \delta_3^{\downarrow} + \delta_{15}^{\downarrow} + \delta_{25}^{\downarrow} + \delta_8^{\downarrow} + \delta_{26}^{\uparrow} + \delta_7^{\uparrow}$$

La valeur maximale du temps de réponse est obtenue pour la valeur maximale de chaque délai intervenant dans le chemin critique. On nomme $d_0$ le point correspondant à l'affectation

[6] Les valeurs maximales de $\delta_{setup_D}$ et $\delta_{setup_W}$ ne sont pas spécifiées (implicitement, elles doivent être inférieures au temps de cycle).

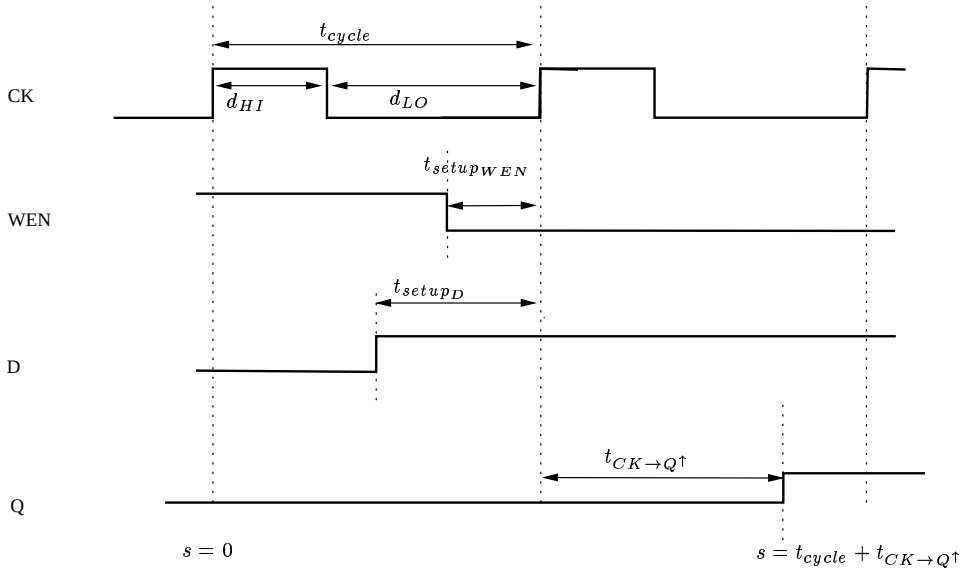


FIG. 5.3 – Chronogramme représentant les signaux externes pour l'expérience W1

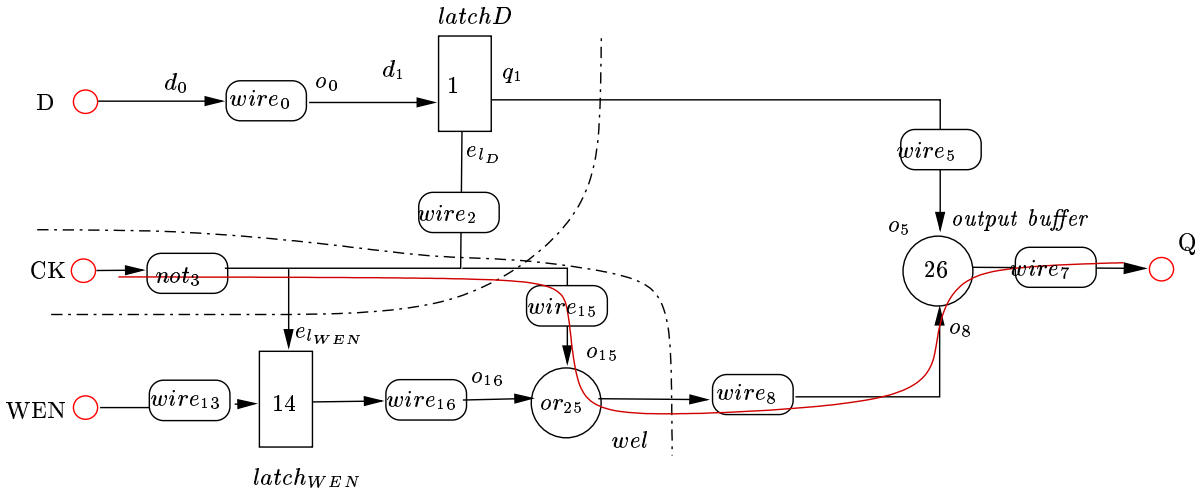


FIG. 5.4 – Portion de l'AFTG de la mémoire SPSMALL correspondant à l'expérience W1

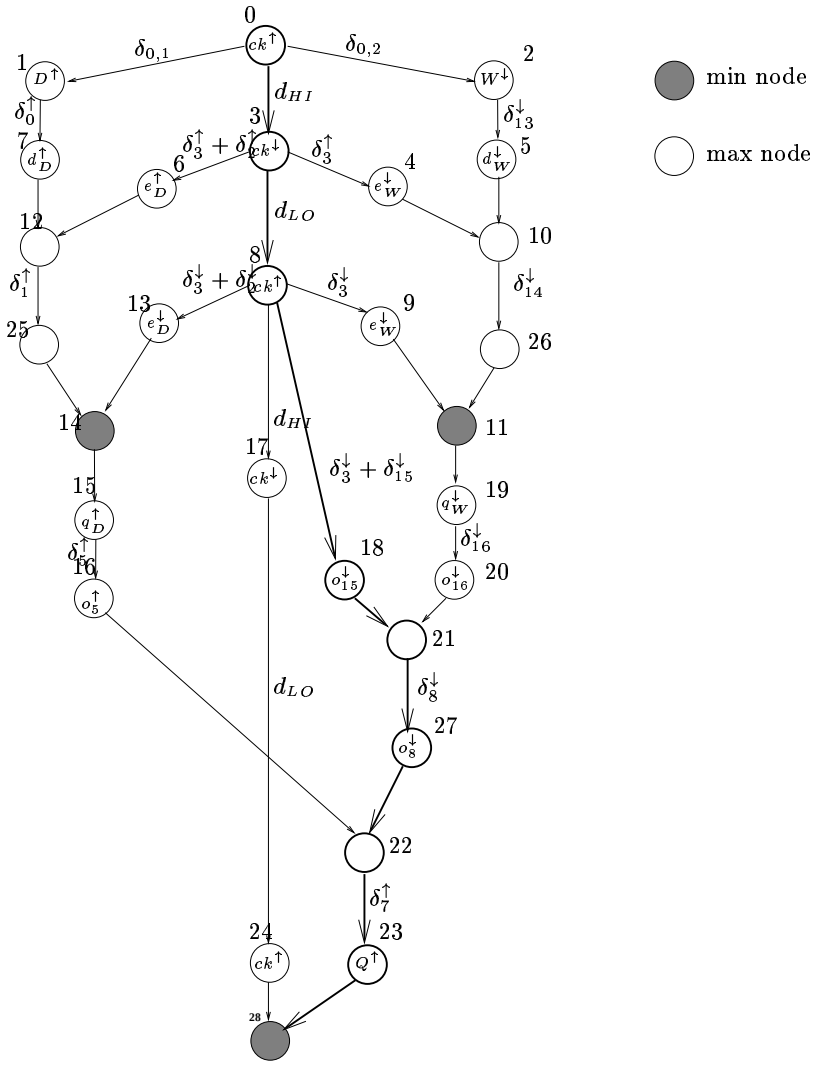


FIG. 5.5 – Graphe de contraintes temporelles de SPSMALL pour l'expérience W1

pour chaque délai de sa valeur maximale (i.e. , $d_0 = \{\delta_k^{\downarrow} = u_k^{\downarrow}$ et $\delta_k^{\uparrow} = u_k^{\uparrow}$ pour tout composant $k\}$). On doit avoir $t_{CK\rightarrow Q^{\uparrow}} \leq t_{max}^{write}$.

**Remarque :** La connaissance *a priori* du chemin critique peut paraitre paradoxale, en fait elle s'explique. Le chemin critique correspond à la propagation d'un signal de contrôle qui "fait progresser" le flot des données et des adresses en différents points du circuit, lorsqu'elles sont disponibles. L'architecture de SPSMALL a été pensée autour de ce signal de contrôle, dont le cheminement *doit* représenter le chemin critique. Son délai de propagation a été déterminé par simulation électrique, et confirmé par model-checking temporisé.

Le graphe de contraintes temporelles associé, correspondant à un bon fonctionnement de la mémoire, est donné sur la figure 5.5. Il correspond à une *famille de chronogrammes* décrivant le mécanisme *write-through* du mode de fonctionnement d'écriture du circuit.

Notre but est de déterminer un ensemble de relations liant les délais internes $\delta$ et les délais

externes $t_{HI}, t_{LO}, t_{setup_D}, t_{setup_W}$, garantissant que le chemin critique est $\Pi$.

### Etape 3 - Résolution du problème inverse de TSE

Il est important de comprendre que le graphe de contraintes TSE *est parcouru sans être explicitement construit*. Ceci est réalisé par une analyse d'accessibilité du système décrit sous la forme d'automates temporisés et paramétrés. En effet, le graphe des contraintes temporelles est implicitement parcouru lors de la construction du graphe des états accessibles du système complètement paramétré. En fait, la construction du graphe des états accessibles effectue la résolution de tous les conflits effectifs du modèle, et revient à parcourir plusieurs graphes de contraintes temporelles (chacun étant non conditionnel).

La portion de l'AFTG pertinente pour l'expérience est modélisée sous forme d'automates temporisés. Un automate temporisé particulier a été défini pour chaque type de noeud de l'AFTG (wire, porte logique, latch). Les automates représentant un wire ou un latch sont munis d'une horloge locale. Pour chaque élément de l'AFTG, un automate temporisé de type correspondant est instancié. Les arcs de l'AFTG correspondent à des transitions (d'automates distincts) synchronisées sur une même étiquette.

**1 -** Le calcul de $Post^*(Init)$ sur le modèle pour lequel $\overline{\delta}$ est instancié par $d_0$, permet d'obtenir $\mathcal{O}$, l'ordre total d'instant d'occurrence des événements induit par $d_0$.

$\mathcal{O} = t_0 \leq t_1 \leq t_3 \leq t_4 \leq t_2 \leq t_6 \leq t_5 \leq t_{19} \leq t_{20} \leq t_7 \leq t_8 \leq t_{15} \leq t_9 \leq t_{18} \leq t_{16} \leq t_{13} \leq t_{25} \leq t_{17} \leq t_{23} \leq t_{24}$.

**2 -** Soit $C_{\mathcal{O}}$ la contrainte représentant l'ordre $\mathcal{O}$. Le calcul de $Pre^*(Final \cap C_{\mathcal{O}}) \cap Init$ permet d'obtenir le système d'inéquations $\mathcal{I}$ garantissant que le système produira la séquence d'événements $\mathcal{O}$.

| | |
|---|---|
| 1. $0 \leq d_{HI} + d_{LO} - \delta_{setup_D}$ | (t0 ≤ t1) |
| 2. $d_{LO} - \delta_{setup_D} \leq 0$ | (t1 ≤ t3 after elim. of $d_{HI}$) |
| 3. $0 \leq \delta_3^{\uparrow}$ | (t3 ≤ t4 after elim. of $d_{HI}$) |
| 4. $\delta_3^{\uparrow} \leq d_{LO} - \delta_{setup_W}$ | (t4 ≤ t2 after elim. of $d_{HI}$) |
| 5. $d_{LO} - \delta_{setup_W} \leq \delta_3^{\uparrow} + \delta_2^{\uparrow}$ | (t2 ≤ t6 after elim. of $d_{HI}$) |
| 6. $\delta_3^{\uparrow} + \delta_2^{\uparrow} \leq d_{LO} - \delta_{setup_W} + \delta_{13}^{\downarrow}$ | (t6 ≤ t5 after elim. of $d_{HI}$) |
| 7. $0 \leq \delta_{14}^{\downarrow}$ | (t5 ≤ t19 after elim. of $d_{HI} + d_{LO} - \delta_{setup_W}$) |
| 8. $0 \leq \delta_{16}^{\downarrow}$ | (t19 ≤ t20 after elim. of $d_{HI} + d_{LO} - \delta_{setup_W} + \delta_{13}^{\downarrow} + \delta_{14}^{\downarrow}$) |
| 9. $-\delta_{setup_W} + \delta_{13}^{\downarrow} + \delta_{14}^{\downarrow} + \delta_{16}^{\downarrow} \leq -\delta_{setup_D} + \delta_0^{\uparrow}$ | (t20≤t7 after elim. $d_{HI} + d_{LO}$) |
| 10. $-\delta_{setup_D} + \delta_0^{\uparrow} \leq 0$ | (t7 ≤ t8 after elim. of $d_{HI} + d_{LO}$) |
| 11. $0 \leq -\delta_{setup_D} + \delta_0^{\uparrow} + \delta_1^{\uparrow}$ | (t8 ≤ t15 after elim. of $d_{HI} + d_{LO}$) |
| 12. $[-\delta_{setup_D} + \delta_0^{\uparrow} + \delta_1^{\uparrow} \leq \delta_3^{\downarrow}]$ | (t15 ≤ t9 after elim. of $d_{HI} + d_{LO}$) |
| 13. $0 \leq \delta_{15}^{\downarrow}$ | (t9 ≤ t18 after elim. of $d_{HI} + d_{LO} + \delta_3^{\downarrow}$) |
| 14. $\delta_3^{\downarrow} + \delta_{15}^{\downarrow} \leq -\delta_{setup_D} + \delta_0^{\uparrow} + \delta_1^{\uparrow} + \delta_5^{\uparrow}$ | (t18 ≤ t16 after elim. of $d_{HI} + d_{LO}$) |
| 15. $-\delta_{setup_D} + \delta_0^{\uparrow} + \delta_1^{\uparrow} + \delta_5^{\uparrow} \leq \delta_3^{\downarrow} + \delta_2^{\downarrow}$ | (t16 ≤ t13 after elim. of $d_{HI} + d_{LO}$) |
| 16. $\delta_2^{\downarrow} \leq \delta_{15}^{\downarrow} + \delta_8^{\downarrow}$ | (t13 ≤ t25 after elim. of $d_{HI} + d_{LO} + \delta_3^{\downarrow}$) |
| 17. $\delta_3^{\downarrow} + \delta_{15}^{\downarrow} + \delta_8^{\downarrow} \leq d_{HI}$ | (t25 ≤ t17 after elim. of $d_{HI} + d_{LO}$) |
| 18. $d_{HI} \leq \delta_3^{\downarrow} + \delta_{15}^{\downarrow} + \delta_8^{\downarrow} + \delta_7^{\uparrow}$ | (t17 ≤ t23 after elim. of $d_{HI} + d_{LO}$) |
| 19. $\delta_3^{\downarrow} + \delta_{15}^{\downarrow} + \delta_8^{\downarrow} + \delta_7^{\uparrow} \leq d_{HI} + d_{LO}$ | (t23 ≤ t24 after elim. of $d_{HI} + d_{LO}$) |

**3 -** Le relâchement de $\mathcal{I}$ par suppression d'inéquations produit un système $\mathcal{I}'$. Pour chaque relâchement, on s'assure que les états finaux de $Post*(Init \cap \mathcal{I}')$ ne contiennent pas d'états pour lesquels $t_{CK \to Q\uparrow} > \Pi$

Dans $\mathcal{I}$, les inéquations 3,7,8 et 13 n'apportent pas d'information. En outre, l'inéquation 12 (entre crochets) peut être retirée sans modifier le chemin critique.

**4 -** La substitution dans le système d'inéquations $\mathcal{I}'$, des $\delta_k$ par des variables $y_k$ dans les termes à gauche et $x_k$ dans les termes à droite produit le système $\mathcal{J}$, qui représente un ensemble de conditions suffisantes garantissant que le chemin critique est $t_{CK \to Q} \leq \delta_3^{\downarrow} + \delta_{15}^{\downarrow} + \delta_8^{\downarrow} + \delta_7^{\uparrow}$.

A partir de $\mathcal{J}$, en considérant $y_k = u_k$ et $x_k = l_k$ pour tout $k$, on peut optimiser les bornes des temps de *setup* des signaux d'entrée D et WEN. On a en effet :

$$\max \left\{ \begin{array}{c} d_{LO} \\ u_0^{\uparrow} \\ u_0^{\uparrow} + u_1^{\uparrow} + u_5^{\uparrow} - l_3^{\downarrow} - l_2^{\downarrow} \end{array} \right\} < t_{setup_D} < \min \left\{ \begin{array}{c} d_{HI} + d_{LO} \\ t_{setup_W} + l_0^{\uparrow} - u_{13}^{\downarrow} - u_{14}^{\downarrow} - u_{16}^{\downarrow} \\ l_0^{\uparrow} + l_1^{\uparrow} \\ l_0^{\uparrow} + l_1^{\uparrow} + l_5^{\uparrow} - u_3^{\downarrow} - u_{15}^{\downarrow} \end{array} \right\}$$

et

$$\max \left\{ \begin{array}{c} d_{LO} - l_3^{\uparrow} - l_2^{\uparrow} \\ u_{13}^{\downarrow} + u_{14}^{\downarrow} + u_{16}^{\downarrow} + t_{setup_D} - l_0^{\uparrow} \end{array} \right\} < t_{setup_W} < \min \left\{ \begin{array}{c} d_{LO} - u_3^{\downarrow} \\ d_{LO} + l_{13}^{\downarrow} - u_3^{\uparrow} - u_2^{\uparrow} \end{array} \right\}$$

Les valeurs numériques obtenues sont alors :

$$\Pi = 56 \qquad 100 < t_{setup_D} < 118 \qquad 46 < t_{setup_W} < 72$$

Elles sont compatibles avec les données de la datasheet (dans laquelle $\Pi \leq t_{max}^{write} = 56$, $t_{setup_D} = 107$ and $t_{setup_W} = 48$), et permettent une optimisation sûre des temps de *setup*.

Cette démarche a été appliquée pour déterminer les temps de *setup* des signaux D et WEN pour une autre implémentation de la mémoire SPSMALL, présentant des délais différents [66]. Elle a également été appliquée pour résoudre des exemples classiques de problèmes TSE (inverse) : l'allocation optimale de tâches concurrentes sur un système multiprocesseurs, la détermination des temps d'occupation de ressources et des temps de latence d'un buffer utilisé dans un contexte multi-émetteurs et multi-récepteurs [45].

## 5.4 Conclusion et prolongements envisagés

La vérification de propriétés temporelles de systèmes embarqués est un problème difficile. Du fait de la complexité exponentielle des méthodes produisant des solutions exactes, il faut trouver, pour chaque application, le bon compromis entre la *précision* des bornes des temps de traversée obtenues et la *taille* du système analysé.

Dans ce contexte, nous avons proposé une méthode automatique permettant d'extraire un ensemble de contraintes suffisantes, liant les paramètres temporels du système, et garantissant des plages de fonctionnement nominal. Le coeur de la méthode repose sur le parcours symbolique des "familles de chronogrammes" (ou graphe de contraintes temporelles) correspondant à un mode de fonctionnement correct du système. Le traitement est polynomial avec le nombre d'événements induits par le mode de fonctionnement. Les optimisations proposées restent de complexité exponentielle. Cette méthode a été appliquée pour optimiser les temps de setup de signaux d'entrées de deux instances différentes la mémoire embarquée SPSMALL, conçue et commercialisée par STMicroelectronics.

De mon point de vue, il subsiste deux écueils importants pour pouvoir appliquer cette méthode à des systèmes plus complexes :

- La construction automatique d'une abstraction fonctionnelle et temporelle "de taille raisonnable". Dans le cas de composants mémoire, nous avons montré que cette étape articulait deux types d'abstraction : l'abstraction architecturale limitant la portion de circuit à analyser et l'abstraction fonctionnelle à partir d'un réseau de transitors. En outre, ces abstractions influent sur la détermination des délais de propagation au travers des composants : les délais associés au point mémoire abstrait peuvent représenter des délais issus de différents points mémoire concrets.
- La donnée du chemin critique ou la caractérisation des bons points initiaux. Notre méthode est polynomiale mais elle repose sur la connaissance d'éléments cruciaux, pas toujours simples à obtenir. Si le système à analyser présente plusieurs modes de fonctionnement bien identifiés, avec pour chacun un "point de fonctionnement" spécifié, alors notre méthode apporte une réelle plus-value en déterminant les plages de fonctionnement du système, pour chacun des modes de fonctionnement. Si par contre nous ne disposons pas de la connaissance du chemin critique, la pertinence de l'analyse dépendra entièrement du choix du point initial.

Les recherches futures concernent l'étude de ces points d'achoppement.

Le projet ANR *VALMEM*, débuté en janvier 2007, fait suite au projet MEDEA+ BLUEBERRIES. Aux partenaires historiques de cette étude (STMicroelectronics et LSV) s'ajoute le LIP6[7], adjoignant son expertise concernant les problèmes d'extraction fonctionnelle et la caractérisation des délais au niveau transistor. Jusqu'à présent, ces deux aspects restaient assez découplés, et nous avons vu qu'il fallait maintenant les combiner. La prise en compte de composants réalisés en logique dynamique est également un objectif important : la logique dynamique permet de réduire le délai de propagation de certains fronts. Elle est très utilisée pour les circuits soumis à des impératifs de haute performance. Sa prise en considération pose des problèmes nouveaux, tant au niveau de l'abstraction que de l'exploitation de l'architecture globale du circuit. Dans cette architecture, des signaux de contrôle déterminent la phase de précharge et la phase d'évaluation de chaque signal. Ces particularités architecturales peuvent nous guider dans la pré-détermination du chemin critique. La détermination de "bons points initiaux" pour la méthode inverse reste à investiguer.

Un objectif complémentaire concerne l'application de cette méthode à des architectures de commandes de systèmes automatisés distribués et tolérants aux pannes. Ces systèmes complexes ont un fonctionnement multi-modes et peuvent basculer dynamiquement d'un mode à l'autre. Ce phénomène de changement de modes est intéressant à analyser. C'est l'objet du projet SIMOP

[7]Equipe CIAN : conception des systèmes analogiques et numériques

(collaboration LSV-LURPA[8] dans le cadre de l'Institut Farman de l'ENS-Cachan), débuté au printemps 2007.

Enfin, contrairement aux circuits mémoires, pour lesquels les délais sont déterminés précisément pour une technologie donnée (en faisant abstraction des phénomènes parasites), les délais intervenant dans les architectures de commandes distribuées sont déterminés de manière statistique ou probabiliste. L'extension de notre méthode aux systèmes pourvus de délais suivant des lois probabilistes est une perspective à plus long terme.

[8]Laboratoire Universitaire de Recherche en Production Automatisée

# Quatrième partie

# Conclusion et Perspectives

La conception de systèmes sur puce, ou plus généralement de systèmes embarqués reste une tâche difficile. Les cycles de production toujours plus rapides ont modifié les pratiques de développement ces dix dernières années. Les outils de modélisation et d'analyse de systèmes décrits au niveau transactionnel doivent contribuer à simplifier ce processus de développement. Les méthodes prenant en compte la réutilisation de composants existants, déjà vérifiés, doivent être conçues pour faciliter la conception basée sur l'asssemblage de composants.

Avec l'introduction des BDD, les méthodes de vérification par model-checking symbolique avaient apporté un réel espoir pour une solution complémentaire à la simulation. Ces méthodes ont trouvé un terrain d'application : les composants matériels décrits au niveau RTL, de faible ou moyenne complexité (jusqu'à 10 000 portes logiques). Elles ont montré leurs limites pour la vérification de systèmes complexes, ou décrits à un niveau plus abstrait.

J'ai montré dans ce manuscrit les principales contributions des recherches auxquelles j' ai participé, concernant la conception et la vérification de parties de systèmes embarqués. Ces contributions sont d'ordre méthodologique et algorithmique.

- *Méthodologique.* Nous avons proposé une méthode simplifiant la conception et la vérification de composants assurant le contrôle de flux. Nous avons ébauché une méthode de vérification compositionnelle, utilisant la spécification de composants déjà vérifiés pour faciliter la vérification de propriétés globales. Nous avons proposé une méthode permettant d'extraire des garanties de bon fonctionnement pour des systèmes temporisés.
- *Algorithmique.* Nous avons proposé différents algorithmes et structures de données permettant de repousser les limites des outils de vérification actuels.

Pour chacune de ces contributions, nous nous sommes attachés à mettre en pratique les méthodes et algorithmes proposés sur des systèmes concrets, de moyenne complexité. Ces expérimentations sont nécessaires pour évaluer la pertinence des modèles et méthodes proposés ; elles permettent en outre de cerner les points d'achoppement, et d'orienter la suite des travaux de recherche, en exploitant une particularité du système ou de la propriété à vérifier.

La recherche visant l'amélioration de ces techniques est toujours d'actualité, et le champ d'investigation est vaste.

L'amélioration des techniques de model-checking proprement dit (exhaustif ou partiel) est un premier axe de recherche. Si cette technique ne peut pas s'appliquer telle quelle sur un système complexe, elle trouve toujours son utilité pour vérifier des composants pris isolément. Le développement de méthodes compositionnelles, utilisant l'information structurelle du système ou le fait que certains composants ont déjà été vérifiés, permet de repousser les limites du model-cheking.

Une autre voie investiguée concerne le développement de méthodes et outils de conception garantissant un développement plus sûr, en définissant des cadres de conception contraints et spécifiques pour le développement d'un type de composant ou de système. Cette thématique recouvre différents aspects. On peut définir un cadre méthodologique de conception (comme la méthode B, ou plus modestement comme la méthode incrémentale) ; on peut également s'intéresser à la synthèse de composants à partir d'une spécification (à des fins d'implémentation réelle, et non pas à des fins de vérification comme nous l'avons proposé dans le chapitre 2).

L'association de la vérification et la simulation afin de générer de "bonnes" séquences de

stimuli est également un thème de recherche important ; puisque la simulation reste la technique d'analyse la plus utilisée, il faut pouvoir qualifier l'ensemble des tests effectués, en terme de taux de couverture, et pouvoir définir des gabarits de tests pertinents.

Tous ces domaines sont très actifs, de nombreux résultats sont proposés mais leur application reste encore bien en deçà des besoins. L'introduction du temps complique encore l'analyse, et les questions abordées dans le paragraphe précédent sont également pertinentes pour la conception et l'analyse temporelle des systèmes matériels.

Dans ce champ très vaste, les perspectives à moyen terme que j'envisage se déclinent en trois points :

**Amélioration du model-checking compositionnel.** Il s'agit de la suite des travaux décrits et mis en perspective dans le chapitre 2 concernant le développement d'un processus de vérification par raffinement d'abstractions pour des composants synchrones. Il s'intègre parfaitement dans un processus de conception par assemblage de composants pré-existants. L'algorithme d'abstraction a été défini, mais de nombreux points restent à investiguer. Ils concernent notamment la boucle de raffinement de ce processus de vérification. L'analyse du contre-exemple comprend plusieurs étapes : déterminer si le contre-exemple est valide ou non ; ceci peut se faire en simulant ce contre-exemple sur le système concret. Si le contre-exemple est valide, il faut en extraire une information permettant d'affiner l'abstraction : on peut construire un automate représentant la négation de ce contre-exemple et l'adjoindre à la description abstraite du système à vérifier, mais on peut aussi chercher à affiner l'abstraction en adjoignant de nouvelles propriétés de la spécification des différents composants. La correspondance entre un contre-exemple valide, et un ensemble de propriétés à choisir pour réaliser ce raffinement est un sujet d'étude. Ceci peut nous amener à proposer une méthode interactive (et non plus entièrement automatique), sollicitant l'expertise du concepteur, pour guider le choix des bonnes propriétés. Dans ce contexte, je souhaite développer une collaboration avec l'équipe du professeur H. Eveking de l'Université de Darmstadt, qui travaille également sur l'abstraction de composants pour accélérer la simulation de gros systèmes. J'entreprends actuellement des démarches pour que Cécile Braunstein puisse effectuer un séjour post-doctoral de 1 an dans l'équipe de H. Eveking en 2008, afin d'amorcer cette collaboration. D'autre part, les liens informels que j'entretiens avec D. Borrione et K. Allory-Morin du laboratoire TIMA (concernant la synthèse de moniteurs de propriétés PSL) pourraient trouver ici un cadre plus formel.

**Analyse temporelle de systèmes embarqués.** La méthode inverse du problème "Time Separation of events" exposée au chapitre 5 permet d'obtenir des plages de bon fonctionnement de systèmes temporisés. J'ai montré au chapitre 5 la plus-value qu'apportait ce type d'analyse, ainsi que sa complémentarité avec les méthodes de calcul de chemin critique. Ce thème de recherche va être poursuivi au travers des projets VALMEM (projet ANR de l'appel "Architecture du futur", de jan 2007 à dec 2009, en collaboration avec R. Chevallier de STMicroelectronics, L. Fribourg du LSV, P. Bazargan-Sabet et P. Renault du LIP6) et SIMOP (projet de l'institut FARMAN de l'ENS de Cachan, de jan 2007 à dec 2009, en collaboration avec L. Fribourg du LSV, O. de Smet et B. Denis du LURPA). Les thèmes de recherche abordés dans ces projets concernent l'abstraction temporelle et fonctionnelle d'une description au niveau transistor en une représentation sous forme d'automates temporisés, la prise en compte de la logique dynamique, la mise en oeuvre de la méthode inverse pour l'analyse de ces systèmes, la détermination de bons points de

fonctionnement initiaux, l'application de la méthode inverse sur plusieurs cas d'étude (différentes mémoires embarquées et architectures de commandes distribuées), l'analyse de la commutation de modes de fonctionnements. L'extension vers des modèles probabilistes est envisagée à plus long terme.

**Conception de fonctions de routage deadlock-free.** Dans la suite des travaux de thèse de S. Taktak, je souhaite poursuivre l'étude et le développement d'un environnement de conception de fonctions de routage pour réseaux intégrés sur puce. Les implications des choix faits dans la thèse de Sami doivent être étudiées au regard d'applications concrètes, notamment dans le contexte des réseaux en grille 2D présentant des défauts de connexions (liens ou routeurs). Ces topologies vont devenir prédominantes dans les années à venir. Dans ce contexte, les fonctions de routage simples (tant du point de vue conceptuel que de l'implantation) peuvent ne plus être adaptées ; il faut alors définir des fonctions de routage ad-hoc, permettant d'atteindre tous les noeuds connexes du réseau, et ceci en introduisant un très faible surcoût matériel.

# Bibliographie


[1] Bibliothèque DDD : http ://ddd.lip6.fr.

[2] Bibliothèque buddy : http ://sourceforge.net/projects/buddy.

[3] Cadence incisive formal verifyer IFV : www.cadence.com/cdnlive/na/files/2005/1198_paper.pdf.

[4] Cadence SMV : www-cad.eecs.berkeley.edu/~kenmcmil/smv/.

[5] Châine de vérification RULEBASE : www.rulebase.com.

[6] DIgital SYstem Design ENvironmenT, environnement de co-développement de systèmes embarqués : http ://www-asim.lip6.fr/recherche/disydent/.

[7] Outil d'abstraction d'un réseau de transistors TLL. In `http ://www.transeda.com/products/`.

[8] Projet verisoft : http ://cm.bell-labs.com/who/god/verisoft/.

[9] Simulateur électrique HSIM. In `http ://www.synopsys.com/products/`.

[10] *AMBA Specification*, 2.0 edition, 1999. www.arm.com/products/solutions/AMBA_Spec.html.

[11] Y. Abarbanel, I. Beer, L. Glushovsky, S. Keidar, and Y. Wolfsthal. FoCs : Automatic Generation of Simulation Checkers from Formal Specifications. In *Int. Conf. on Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 538 – 542. Springer-Verlag, 2000.

[12] B.T. Adler, L. de Alfaro, L. Dias da Silva, M. Faella, A. Legay, V. Raman, and P. Roy. Ticc : A tool for interface compatibility and composition. In *Int. Conf. on Computer Aided Verification (CAV)*, volume 4144 of *LNCS*, pages 59–62. Springer Verlag, 2006.

[13] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science 126*, pages 183–235, 1994.

[14] T. Amon and G. Borriello. An approach to symbolic timing verification. In *Proc. of* $29^{th}$ *ACM/IEEE Int. Conf. on Design Automation (DAC)*, pages 410–413. IEEE Computer Society Press, 1992.

[15] T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic timing verification of timing diagrams using Presburger formulas. In *Proc. of ACM/IEEE Int. Conf. on Design Automation (DAC)*, pages 410–413. IEEE Computer Society Press, 1997.

[16] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. Masson, 1992.

[17] R. Bawa. *Validation des systèmes décrits en VHDL*. PhD thesis, Université Pierre et Marie Curie (Paris 6), 1996.

[18] R. Bawa and E. Encrenaz. A tool for translation of VHDL descriptions into a formal model and its application to formal verification and synthesis. volume 1135, pages 171–174, 1996.

[19] V. Beaudenon. *Diagrammes de décision de données pour la vérification de systèmes matériels*. PhD thesis, Université Pierre et Marie Curie (UPMC, Paris 6) - LIP6, 2006.

[20] V. Beaudenon and E. Encrenaz. Utilisation des diagrammes de décision de données pour la vérification fonctionnelle de systèmes matériels. In *Colloque CNRS Jeunes chercheurs, MAJECSTIC, France*, oct. 2004.

[21] V. Beaudenon, E. Encrenaz, and J.L. Desbarbieux. Design validation of ZCSP with SPIN. In $3^{rd}$ *Int. Conf. on Application of Concurrency to System Design (ACSD)*, pages 102–110. IEEE Computer Society, 2003.

[22] V. Beaudenon, E. Encrenaz, and S. Taktak. Data decision-diagrams for promela systems analysis. Technical report, Laboratoire d'Informatique de Paris 6, accepted in STTT, jun. 2005.

[23] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. Météor : A successful application of B in a large project. In *World Congress on Formal Methods*, pages 369–387, 1999.

[24] W. Belluomini and C. Myers. Timed state space exploration using posets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 19(5), 2000.

[25] R. Bloem, B. Jobstmann, and A. Pnueli. Property-based logic synthesis for rapid design prototyping. Technical report, Prosyd Deliverable D2.2/1, 2005.

[26] D. Borrione, M. Liu, P. Ostier, and L. Fesquet. *Applications of Specification and Design Languages for SoCs*, chapter PSL-based online monitoring of digital systems. Springer, 2006.

[27] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, and P. Raymond. Minimal state graph generation. *Science of Computer programming*, 18(3), 1994.

[28] M. Boulé and Z. Zilic. Efficient Automata-Based Assertion-Checker Synthesis of PSL Properties. In $11^{th}$ *IEEE Int. Work. on High-Level Design Verification and Test (HLDVT)*, pages 69–76. IEEE Computer Society Press, nov 2006.

[29] P. Bouyer and F. Laroussinie. Vérification par automates temporisés. In *Systèmes temps-réel 1 : techniques de description et de vérification*, pages 121–150. Hermès, 2006.

[30] M. Bozga, J-C. Fernandez, A. Kerbrat, and L. Mounier. Protocol Verification with the ALDÉBARAN Toolset. *Int. Jour. on Software Tools and Technology Transfert (STTT)*, 1(1-2) :166–184, 1997.

[31] C. Braunstein. *Conception incrémentale et vérification de composants matériels et techniques d'abstraction de composants*. PhD thesis, Université Paris VI, 2007. to appear.

[32] C. Braunstein and E. Encrenaz. CTL-property transformations along an incremental design process. In *Int. Work. Advances on Critical Systems (AVoCS)*, volume Electronic Notes on Computer Sciences, 2005. Elsevier.

[33] C. Braunstein and E. Encrenaz. Formalizing the incremental design and verification process of a pipelined protocol converter. In *IEEE Int. Work. on Rapid Systems Prototyping (RSP)*, pages 103–109. IEEE Computer Society Press, jun 2006.

[34] C. Braunstein and E. Encrenaz. A further step in the incremental design process : Incorporation of the increment specification. In *Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, 2006. short paper.

[35] C. Braunstein and E. Encrenaz. CTL-property transformations along an incremental design process. *Int. Journal on Software Tools and Technology Transfert (STTT)*, 9(1) :77–88, feb 2007.

[36] C. Braunstein and E. Encrenaz. Using CTL Formulae as Component Abstraction in a Design and Verification Flow. In *Int. Conf. on Application of Concurrency in System Design (ACSD)*. IEEE Computer Society Press, 2007.

[37] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8) :677–691, 1986.

[38] T. Burks and K. Sakallah. Min-max linear programming and the timing analysis of digital circuits. In *Proc. of ACM/IEEE Int. Conf. on Computer-Aided Design (ICCAD)*, pages 152–155. IEEE Computer Society Press, 1993.

[39] D. Cansell and D. Méry. Abstraction and Refinement of Features. In S. Gilmore and M. Ryan, editors, *Language Constructs for Designing Features*, pages 65–84. Springer-Verlag, 2001.

[40] F. Cassez, M. Ryan, and P-Y. Schobbens. Proving Feature Non-Interaction with Alternating-Time Temporal Logic. In S. Gilmore and M. Ryan, editors, *Language Constructs for Describing Features*, pages 85–104. Springer Verlag, 2001.

[41] S. Chakraborty and D. Dill. Approximate algorithms for time separation of events. In *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 190–194. IEEE Computer Society, 1997.

[42] S. Chakraborty, D. Dill, and K. Yun. Efficient algorithms for approximate time separation of events. *Sadhana : Academy Proceedings in Engineering Sciences*, 27(2) :129–162, 2002.

[43] S. Chakraborty, D. Dill, K. Yun, and K. Chang. Practical timing analysis of asynchronous circuits using time separation of events. In *Proc. of IEEE Custom Integrated Circuits Conference*. IEEE Computer Society Press, 1998.

[44] S. Chakraborty, P. Subrahmanyam, and D. Dill. Approximate time separation of events in practice. In *Proc. of 5th ACM/IEEE Int. Workshop TAU*. IEEE Computer Society Press, 1997.

[45] S. Chakraborty, K. Yun, and D. Dill. Timing analysis for extended burst-mode circuits. In *Proc. of* $3^{rd}$ *IEEE Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 101–111. IEEE Computer Society Press, 1998.

[46] S. Chakraborty, K. Yun, and D. Dill. Timing analysis of asynchronous systems using time separation of events. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(8) :1061–1076, 1999.

[47] H. Charlery. *Intégration d'un micro-réseau à commutation de paquets dans un système multiprocesseur à mémoire partagée intégré sur puce.* PhD thesis, Université Pierre et Marie Curie (Paris 6), 2005.

[48] Hervé Charlery, Alain Greiner, Emmanuelle Encrenaz, Laurent Mortiez, and Adrijean Andriahantenaina. Using VCI in a on-chip system around SPIN network. In $11^{th}$ *IEEE Int. Conf. on Mixed Design of Integrated Circuits and Systems*, pages 571–576, 2004.

[49] R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Timing analysis of an embedded memory : SPSMALL. In *10th WSEAS Int. Conf. on Circuits and Systems, WSEAS Trans. on Circuits and Systems, vol 5(7)*, pages 973–978, 07 2006.

[50] R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Verification of the generic architecture of a memory circuit using parametric timed automata. In *Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *LNCS*, pages 113–127. Springer, 2006.

[51] G. Ciardo, G. Luettgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. Technical report, 1999.

[52] G. Ciardo, G. Lüttgen, and R. Siminiceanu. Saturation : An efficient iteration strategy for symbolic state-space generation.

[53] G. Ciardo and A. Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In D. Borrione and W. Paul, editors, *CHARME*, volume 3725 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2005.

[54] R. Clariso and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. ASP-DAC*, pages 628–633, 2004.

[55] R. Maruccia G. Pieralisi L. Scandurra A. Coppola, M. Locatelli. Spidergon : a novel on-chip communication network. *Int. Symp. on System on Chip*, 2004.

[56] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. PETRIFY : a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3) :315–325, 1997.

[57] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Int. work. on automatic verification methods for finite state systems*, pages 365–373, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[58] J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.A. Wacrenier. Data decision diagrams for petri net analysis. In $23^{rd}$ *Int. Conf. on Applications and Theory of Petri Nets (ATPN)*, pages 101–120. Springer-Verlag, 2002.

[59] J.M. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'05)*, 2005.

[60] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5) :547–553, 1987.

[61] L. de Alfaro, L. da Silva, M Faella, A. Legay, P. Roy, and M. Sorea. Sociable interfaces. In *Int. Work. on Frontiers of Combined Systems (FroCos)*, volume 3717 of *LNAI*, pages 81–105. Springer Verlag, 2005.

[62] L. de Alfaro and T.A. Henzinger. Interface theories for component-based design. In *Int. Work. on Embedded Software (EMSOFT)*, volume 2211 of *LNCS*, pages 148–165. Springer Verlag, 2001.

[63] K Dioury, A Lester, A Debreil, G Avot, A. Greiner, and M.-M. Louërat. Hierarchical static timing analysis at bull with hitas. In *Int. conf. on Design Automation and Test in Europe (DATE)*, pages 406–410. IEEE Computer Society Press, 2000.

[64] José Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. Parallel Distrib. Syst.*, 6(10) :1055–1067, 1995.

[65] E. Encrenaz. *Une méthode de vérification de programmes VHDL basée sur des modèles formels de Réseaux de Petri.* PhD thesis, Université Pierre et Marie Curie (UPMC, Paris 6) - LIP6, 1995.

[66] E. Encrenaz and L. Fribourg. Time Separation of Events : an Inverse Method. Technical Report LSV-07-08, Laboratory Specification and Verification, 02 2007.

[67] E. Encrenaz, D. Poitrenaud, and E. Paviot-Adet. Vérification de programmes VHDL à l'aide de DDD. Technical Report LIP6 - Rapport final du marché 99-34-024.

[68] R. Brayton et al. VIS : a system for verification and synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, $8^{th}$ *Int. Conf. on Computer Aided Verification CAV*, volume 1102, pages 428–432, New Brunswick, NJ, USA, / 1996. Springer Verlag.

[69] E. Fleury and P. Fraigniaud. A General Theory for Deadlock Avoidance in Wormhole-Routed Networks. *IEEE Trans. Parallel Distrib. Syst.*, 9(7) :626–638, 1998.

[70] B Gebremichael, M. Vaandrager, F. Zhang, K. Goossens, E. Rijpkema, and A. Radulescu. Deadlock prevention in the aetheral protocol. In *IFIP WG 10.5 Correct Hardware Design and Verification Conference (CHARME)*, volume 3725 of *LNCS*, pages 345–348, 2005.

[71] K. Goossens, J. Dielissen, and A. Radulescu. Æthereal network on chip : Concepts, architecture and implementations. *IEEE Design & Test of Computers*, 22(5) :414–421, 2005.

[72] O. Grumberg and D.E. Long. Model Checking and Modular Verification. In *Int. Conf. on Concurrency Theory*, volume 527 of *LNCS*, pages 250–263. Springer Verlag, 1991.

[73] P. Guerrier and A. Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. In *Int. conf. on Design Automation and Test in Europe (DATE)*, pages 250–256. IEEE Computer Society Press, 2000.

[74] A. Gurfinkel and M. Chechik. Why waste a perfectly good abstraction ?. In *Int. conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 212–226, 2006.

[75] A. Gurfinkel, O. Wei, and M. Chechik. Yasm : A software model-checker for verification and refutation. In *Int. Conf. on Computer-Aided Verification (CAV)*, volume 4144 of *LNCS*, 2006.

[76] R. Hardin, Z. Har'El, and R. Kurshan. Cospan. In *Proceeding of Int. Conf. on Computer-Aided Design (CAV)*, volume 1102, pages 423–427. Springer-Verlag, 1996.

[77] T. Henzinger, P. Ho, and H. Wong-Toi. A User Guide to HYTECH. In *Int. conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[78] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with BLAST. In *Int. Work. on Model-Checking of software (SPIN)*, volume 2648 of *LNCS*, pages 235–239. Spinger Verlag.

[79] T. A. Henzinger, X. Liu, S. Qadeer, and S. K. Rajamani. Formal specification and verification of a dataflow processor array. In *IEEE/ACM Int. conf. on Computer-aided Design (ICCAD)*, pages 494–499. IEEE Computer Society, 1999.

[80] T.A. Henzinger, S. Qadeer, and S.K. Rajamani. You Assume, We Guarantee : Methodology and Case Studies. In *Int. conf. on Computer Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 440–451, London, UK, 1998. Springer-Verlag.

[81] G. Holzmann. *The SPIN Model Checker. Primer and Reference Manual.* Addison-Wesley Publishers, 2004.

[82] G.J. Holzmann. The Model Checker Spin. *IEEE Trans. on Software Engineering*, 23(5) :279–295, May 1997.

[83] H. Hulgaard and S. Burns. Bounded delay timing analysis of a class of CSP programs. *Formal Methods in System Design*, 11 :265–294, 1997.

[84] H. Jain, D. Kroening, N. Sharygina, and E. Clarke. VCEGAR : Verilog CounterExample Guided Abstraction Refinement. In *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2007. Accepted paper.

[85] D. N. Jayasimha, L. Schwiebert, D. Manivannan, and J. May. A foundation for designing deadlock-free routing algorithms in wormhole networks. *J. ACM*, 50(2) :250–275, 2003.

[86] G. Kahn. The semantics of a simple language for parallel programming. In Jack L. Rosenfeld, editor, *Information Processing 74 : Proceedings of the IFIP Congress 74*, pages 471–475. IFIP, North-Holland Publishing Co., August 1974.

[87] Y. Kukimoto and R. K. Brayton. Hierarchical functional timing analysis. In *Design Automation Conference (DAC)*, pages 580–585. ACM/IEEE Computer Society Press, 1998.

[88] Y. Kukimoto and R.K. Brayton. Exact required time analysis via false path detection. In *Design Automation Conference (DAC)*, pages 220–225. ACM/IEEE Computer Society Press.

[89] K. Lano. *The B Language and Method, A guide to Practical Formal Development.* Springer-Verlag, Secaucus, NJ, USA, 1996.

[90] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1 :134–152, 1997.

[91] B. Lin and A.R. Newton. Implicit manipulation of equivalence classes using binary decision diagrams. In *ICCD '91 : Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, pages 81–85, Washington, DC, USA, 1991. IEEE Computer Society.

[92] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property Preserving Abstractions for the Verification of Concurrent Systems. *Formal Methods in System Design*, 6(1) :11–44, 1995.

[93] D. E. Long. *Model Checking, Abstraction, and Compositional Verification.* PhD thesis, Carnegie Mellon University, 1993.

[94] M. Braun H. Eveking M. Schickel, V. Nimbler. On Consistency and Completeness of Property-Sets : Exploiting the Property-based Design Process. In *Proc. of Int. ECSI Forum on Design Languages.* ECSI, 2006.

[95] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *Int. conf. on Correct Hardware Design and Verification Methods (CHARME)*, volume 987, pages 189–205, 1995.

[96] K. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[97] K. McMillan and D. Dill. Algorithms for interface timing specification. In *Proc. of the IEEE Int. Conf. on Computer Design (ICCD)*, pages 48–51. ACM/IEEE Computer Society, 1992.

[98] K. Morin-Allory and D. Borrione. On-line monitoring of properties built on regular expressions sequences. In *Forum on specification & Design Languages (FDL'06)*, Sep 2006.

[99] C. Nelson, C. Myers, and T. Yoneda. Efficient verification of hazard-freedom in gate-level timed asynchronous circuits. In *Int. conf. on Computer-Aided Design (ICCAD)*, pages 424–432. IEEE Computer Society / ACM, 2003.

[100] On-Chip Bus Development Working Group. *Virtual Component Interface Standard (VCI).* VSI Alliance, 2000.

[101] Open Microprocessors System Initiatives. *OMI324 : PI-Bus Standard Specification.* Siemens, Munich, Germany, 1994.

[102] J. Ossima-Kheba. Conception incrementale de wrappers vci-pi et vci-amba, 2003. Master Thesis of University Paris VI.

[103] I. Panades, A. Greiner, and A. Sheibanyrad. A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach. *NanoNet*, 2006.

[104] R. Passerone, L. de Alfaro, T. Henzinger, and A. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis : Two faces of the same coin. In *International Conference on Computer Aided Design (ICCAD)*, pages 132–139. IEEE Computer Society, 2002.

[105] R. Passerone, J. Rowson, and A. Sangiovanni-Vincentelli. Automatic synthesis of interfaces between incompatible protocols. In *Proceedings of the Design Automation Conference (DAC)*, pages 8–13. IEEE Computer Society, 1998.

[106] E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri net analysis using boolean manipulation. In Robert Valette, editor, *Application and Theory of Petri Nets (ATPN)*, volume 815 of *Lecture Notes in Computer Science*, pages 416–435. Springer, 1994.

[107] M. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. In *Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 2–11. IEEE Computer Society Press, 2000.

[108] H. Peng, S. Tahar, and F. Khendek. Comparison of SPIN and VIS for protocol verification. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(2) :234–245, February 2003.

[109] M. Plath and M. Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 41(1) :53–84, 2001.

[110] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. RAPPID : An asynchronous instruction length decoder. In *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 60–70. IEEE Computer Society Press, 1999.

[111] R. Ben Salah, M. Bozga, and O. Maler. On timing analysis of combinational circuits. In *Int. conf. on Formal Modelling and Analysis of timed systems (FORMATS)*, volume 2791, pages 204–219, 2003.

[112] S. Sandler and C. Hymowitz. *SPICE Circuit Handbook*. Mc Graw Hill, 2006.

[113] K. Schneider. *Verification of Reactive Systems*. Springer verlag, 2004.

[114] L. Schwiebert and D. N. Jayasimha. A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing. *Journal of Parallel and Distributed Computing*, 32(1) :103–117, 1996.

[115] J. Sifakis. A unified approach for studying the properties of transition systems. *Theoretical Computer Science*, 18(3) :227–258, June 1982.

[116] M. Solé and E. Pastor. Evaluating symbolic traversal algorithms applied to asynchronous concurrent systems. In *ACSD*, pages 207–216. IEEE Computer Society, 2004.

[117] S.Yovine. KRONOS : A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1 :123–133, 1997.

[118] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[119] S. Taktak. Model-checking symbolique de programmes promela. comparaison des représentations BDD et DDD, 2004. Master Thesis of University Paris VI.

[120] S. Taktak, E. Encrenaz, and J.-L. Desbarbieux. A tool for automatic detection of deadlock in whormhole networks on chip. In $11^{th}$ *IEEE international Workshop on High-Level Design Verification and Test (HLDVT)*, pages 203–210. IEEE Computer Society Press, 2006.

[121] Tarjan. Depth-First Search and Linear Graph Algorithms. pages 146–160, 1972.

[122] R. Thacker, W. Belluomini, and C. Myers. Timed circuit synthesis using implicit methods. In *Proc. of IEEE International Conference on VLSI Design*, pages 181–188. IEEE Computer Society Press, 1999.

[123] Y. Thierry-Mieg. *Techniques pour le model-checking de spécification de Haut Niveau.* PhD thesis, Université Pierre et Marie Curie (UPMC, Paris 6) - LIP6, 2004.

[124] E. Ulrich, K.P. Lentz, S. Demba, and R. Razdan. Concurrent min-max simulation. In *European Conference on Design Automation (Euro-DAC)*, pages 554–557. IEEE Computer Society Press, 1991.

[125] F. Xie and J.C. Browne. Verified systems by composition from verified components. In $9^{th}$ *European Software Engineering Conference*, pages 277–286, New York, NY, USA, 2003. ACM Press.

[126] H Yalcin, M Mortazavi, R Palermo, C Bamji, K.A. Sakallah, and J.P. Hayes. An advanced timing characterization method using mode dependency. In *Design Automation Conference (DAC)*, pages 657–660. IEEE Computer Society Press, 2001.

[127] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. *Int. Conf. on Computer Aided Verification (CAV)*, 697 :210–224, 1993.

[128] H. Zheng, E. Mercer, and C. Myers. Modular verification of timed circuits using automatic abstraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9) :1138–1153, 2003.