REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Ibn Khaldoun de Tiaret
Faculté des Mathématiques et d'Informatique
Département d'Informatique
Laboratoire d'Informatique et Mathématiques

Thèse

Présentée pour l'obtention du diplôme de Doctorat LMD
En : Informatique
Spécialité : Génie Informatique
Option : Systèmes Embarqués et Temps Réel

Par : DAOUD Hayat

Sujet

Conception **I**ncrémentale **P**rouvée pour p**ro**totypage

rapide de **NoC** tolérant aux fautes à base de technologie FPGA :
CIPRONoC

Soutenue publiquement, le 02/05/2019, devant le Jury composé de :

MM.

HADDOUCHE Kamel	Professeur	Université de Tiaret	Président
CHOUARFIA Abdallah	Professeur	Université d'Oran	Examinateur
GUEZOURI Mustapha	Professeur	Université d'Oran	Examinateur
BELARBI Mostefa	MCA	Université de Tiaret	Encadreur
SENOUCI Abdelkader	Professeur	Université de Tiaret	Co-encadreur

Dédicace

À toute ma famille, ma belle-famille, mon mari, mes amis … et ma fille ASSIA TASNIME.

À mes parents qui ont sacrifié pour moi, je les remercie de m'avoir rendu mes yeux …

Remerciements

Avant tout, je remercie ALLAH le tout puissant de m'avoir donné le courage et la santé pour finir ce travail de thèse.

Je tiens à remercier l'équipe du Laboratoire d'Informatique et de Mathématiques (LIM) et, plus particulièrement, Monsieur SENOUCI Abdelkader, Directeur du laboratoire, de nous avoir offert toutes les conditions de recherche et pour m'avoir fait l'honneur de codiriger ce travail. Je sais combien son temps est précieux, et je lui exprime toute ma gratitude.

Je remercie très chaleureusement mon encadreur de thèse Monsieur BELARBI Mostefa, Maître de Conférences « A », à l'Université Ibn Khaldoun de Tiaret pour tous ses conseils, ses recommandations, son soutien, ses encouragements et de m'avoir supporté tout au long de cette thèse. J'ai beaucoup apprécié et pris un grand plaisir à travailler avec lui.

Je remercie l'ensemble des Membres du Jury qui m'ont fait l'honneur d'accepter, de lire et d'évaluer cette thèse.

Je ne saurais oublier ceux qui ont tant contribué à ces travaux, Le Directeur du laboratoire LCOMS-Metz Monsieur Camel TANOUGAST ; il avait un rôle scientifique et humain très important pendant la préparation de ma thèse avec ses remarques pertinentes qui ont fait progresser ce travail de recherche. Aussi, je remercie Monsieur Dominique MERY, Directeur de l'école doctorale de la région Lorraine Nancy, de m'avoir donné une base de recherche sur le domaine de spécification formelle. Les deux laboratoires ont fait preuve d'utilité et de collaboration dans la recherche scientifique.

Effectuer une thèse de Doctorat est un travail qui comporte de bons côtés comme l'apprentissage à de nouveaux concepts, la découverte permanente mais aussi des moments plus difficiles. Afin de surmonter ces derniers, le soutien des personnes qui vous entourent est indispensable. J'ai eu la chance d'avoir bénéficié de cet appui et c'est avec gratitude que je m'adresse à mes amis et à tous les membres de ma famille pour leur soutien sans limite et leurs encouragements.

Un grand remerciement à notre université Ibn Khaldoun de Tiaret de nous avoir donné une chance de faire face à un travail scientifique et de nous faciliter les procédures d'inscription et de stages à l'étranger.

Résumé

Les approches pour la conception de tolérance aux fautes de réseau sur puce (NoC) pour une utilisation dans un System-on-Chip (SoC) à base de technologie reconfigurable FPGA sont complexes surtout pour les systèmes sur puce multiprocesseur (MPSoC). Pour ce but, l'utilisation de méthodes formelles rigoureuses, basées sur la conception incrémentale et la théorie de la preuve, est devenue une étape essentielle dans le processus de validation. La méthode formelle Event-B est une approche prometteuse pour être utilisée pour développer un modèle et de prouver la sûreté de fonctionnement des architectures MPSoCs. À cet effet, ce manuscrit propose une approche de la vérification formelle pour NoCs et les contraintes de sûreté de fonctionnement d'une architecture NoC comprenant le choix relatif au chemin de routage des paquets de données et la stratégie imposée pour les nœuds détectés comme défectueux. Le processus de formalisation est basé sur un développement correctement validé par construction de l'architecture NoC en utilisant le formalisme de Event-B. La partie statique du modèle est spécifiée dans le contexte et la partie dynamique dans la machine. Les résultats obtenus par raffinement démontrent l'efficacité de détecter au préalable des erreurs dans un système afin de proposer des solutions pour assurer un fonctionnement fiable.

Mots-Clés : Réseau sur puce, Switch, Routage adaptatif, Machine, Contexte, Modèle, Spécification, Raffinement, Preuve formelle, Correction par construction.

Abstract

The approaches of designing faults tolerance network on chip (NoC) that use in the FPGA based System-on-Chip (SoC) technology are difficult especially in the case of multiprocessor SoCs (MPSoC). For this purpose, the use of rigorous formal approaches, based on progressive design and proof theory, has become an essential step in the validation process. Formal Event-B method as a promising approach can be used to develop this model and prove the dependability of MPSoCs architectures. This manuscript provides an approach for the formal verification of NoCs and dependability constraints of a NoC architecture including the choice on the data packet routing path and strategy imposed for the faulty nodes detection. The formalization process is based on a development properly validated by construction of the NoC architecture using the Event-B formalism. The static part of model is specified in the context and dynamic part in the machine. The results, obtained by refinement, show the effectiveness to detect prior errors in a system in order to propose solutions to ensure reliable operational.

Keywords: Network on Chip, Switch, Adaptive-routing, Machine, Context, Model, Specification, Refinement, Formal proof, Correction by construction.

Table des matières

LISTE DES FIGURES
LISTE DES TABLEAUX
LISTE DES ACRONYMES
INTRODUCTION GENERALE 1
CHAPITRE I : RESEAUX SUR PUCE 4
INTRODUCTION 4
I.1. EMERGENCE DES RESEAUX SUR PUCE 4
I.1.1. Evolution des systèmes sur puce et l'exigence de nouvelles structures *4*
I.1.2. Structures actuelles de communication intégrée *5*
I.1.3. Paradigme des réseaux sur puce *6*
I.1.4. Couches réseaux dans les NoCs *7*
I.2. MÉTRIQUES DE PERFORMANCES 8
I.2.1. Le diamètre *8*
I.2.2. La bande passante utile *8*
I.2.3. La bande passante de bissection *8*
I.2.4. La latence *9*
I.2.5. La surface *9*
I.2.6. La diversité de chemin *9*
I.2.7. L'extensibilité *9*
I.2.8. La flexibilité *10*
I.3. ARCHITECTURE DU NOC 10
I.3.1. Composants d'un réseau sur puce (NoC) *10*
I.3.2. Caractéristiques des réseaux sur puce *14*
I.4. QUALITÉ DE SERVICE 26
I.5. QUELQUES NOCS ACADÉMIQUES 27
I.5.1. SPIN *27*
I.5.2. ANoC *28*
I.5.3. QNoC *28*
I.5.4. Hermès *30*
I.6. RÉSEAUX SUR PUCE RECONFIGURABLES À BASE DE FPGA 30
I.7. LIMITES DES RÉSEAUX SUR PUCE 36
I.7.1. Cohérence des caches *36*
I.7.2. Fiabilité *36*
I.7.3. Ordre des communications *37*
I.8. OBJECTIFS DE LA CONCEPTION DES NOC 37
I.9. SÛRETÉ DE FONCTIONNEMENT DES RÉSEAUX SUR PUCE 37
I.9.1. Définition *37*
I.9.2. SdF appliquée aux réseaux sur puce *38*
CONCLUSION 38
CHAPITRE II : TECHNIQUES DE VÉRIFICATION 40
INTRODUCTION 40
II.1. OUTILS DE CONCEPTION ET SIMULATION 40
II.1.1. Flot de conception classique sur FPGA *40*
II.1.2. Utilité d'un langage de description de matériel *41*
II.1.3. Flot de conception *41*
II.1.4. Concepts de base *42*
II.1.5. Quelques langages de description par programmation FPGA *44*
II.1.6. Discussions sur les outils de programmation FPGA *45*
II.2. MÉTHODES FORMELLES 46
II.2.1. Définition *46*
II.2.2. Intérêt d'une vérification par preuve formelle *47*

II.3. MÉTHODES FORMELLES ET CYCLE DE VIE DU LOGICIEL 48
II.4. NIVEAUX D'ABSTRACTION DES MÉTHODES FORMELLES 49
II.5. TECHNIQUES DE VÉRIFICATION FORMELLE 50
II.6. SYSTÈMES FORMELS, VÉRIFICATION, PREUVE, MODEL-CHECKING 52
II.7. APPROCHE FORMELLE 54
II.7.1. Réseaux de Pétri *54*
II.7.2. Diagramme états de transitions *55*
II.7.3. Méthode VDM *56*
II.7.4. Méthode Z *56*
II.7.5. Méthode B *57*
II.8. MÉTHODES FORMELLES ET LE MODÈLE CHECKING 58
II.9. CHOIX DE LA TECHNIQUE DE VÉRIFICATION 60
II.10. OUTILS DE VÉRIFICATION 61
II.10.1. Outils de vérification par preuve *61*
II.10.2. Outils de vérification par model checking *62*
CONCLUSION 63
CHAPITRE III : LA METHODE B 64
INTRODUCTION 64
III.1. LA METHODE B 64
III.1.1. Pourquoi choisir la méthode B *64*
III.1.2. Les fondements théoriques *65*
III.1.3. Principe de la méthode B *67*
III.2. EVOLUTION DU B VERS EVENT-B 70
III.3. STRUCTURE D'UN MODÈLE EVENT-B 71
III.3.1. Contexte *73*
III.3.2. La machine abstraite *73*
III.3.3. Les événements *74*
III.3.4. Notion du raffinement en Event B *74*
III.4. TLA+ PAR RAPPORT À EVENT B 75
III.5. PLATEFORME RODIN 76
III.5.1. Description de la plateforme *76*
III.5.2. Obligations de preuve *76*
III.6. ATELIER B 78
III.7. LE « PLUG-IN » PROB 79
CONCLUSION 79
CHAPITRE IV : APPLICATION ET VALIDATION 80
INTRODUCTION 80
IV.1. MODÈLE D'UN Q-SWITCH 80
IV.1.1. Rôle d'un Q-Switch *80*
IV.1.2. Les composants d'un Q-Switch *81*
IV.1.3. Algorithme de routage XY *84*
IV.1.4. L'algorithme de routage adaptatif *85*
IV.1.5. Spécification formelle du QNoC *85*
IV.2. MODÈLE RKT-SWITCH 94
IV.2.2. Définition *94*
IV.2.3. Spécification avec RODIN *96*
IV.3. STRUCTURE DE NOCS EN RÉSEAU 104
IV.3.1. Vue d'ensemble de Vertex algorithmes de coloration *105*
IV.3.2. Spécification et vérification formelle *106*
IV.3.3. Résultats et performances *110*
CONCLUSION 110
CONCLUSION GÉNÉRALE ET PERSPECTIVES 112

BIBLIOGRAPHIE.....114

Liste des figures

Chapitre I

Figure I. 1 *Structures de communication traditionnelle*....5
Figure I. 2 *Structure d'un réseau sur puce* [56]....10
Figure I. 3 *Illustration de la structure d'un routeur d'un réseau sur puce* [3] 11
Figure I. 4 *Exemple de structure d'une tuile d'un NoC* [57]....11
Figure I. 5 *Positionnements des files d'attentes dans un routeur : (A) en entrée (B) en sortie (C) en sortie virtuelle*....12
Figure I. 6 *Topologies souvent utilisées dans les NoCs : (A) Crossbar (B) Buttery (C) Clos (d) Benes (E) Anneau (ring) (F) Anneau à cordes (chordal ring) (G) Tore (torus) (H) Tore en quinconce (folded torus) (I) Maillage à deux dimensions (2D-Mesh) (J) Arbre élargi (fat tree)* 16
Figure I. 7 *Architecture du réseau sur puce SPIN* [15]....27
Figure I. 8 *La topologie pour ANOC*....28
Figure I. 9 *Réseau QNOC : (A) Topologie du réseau ; (B) Architecture du routeur.* 29
Figure I. 10 *Topologie maillée 2D du NoC Hermès*....30
Figure I. 11 *Illustration de changement de topologie de NoC dans l'approche* [96] *pour différentes applications* 34
Figure I. 12 *Illustration de deux couches distinctes (NoC et PE) dans les approches NoC reconfigurables* 35
Figure I. 13 *L'approche DyNoC* [104] *: Illustration de l'évolution d'un réseau dynamique reconfigurable dans le temps* 36

Chapitre II

Figure II. 1 *Les étapes de conception moderne*....41
Figure II. 2 *Architecture d'un schéma de spécification* Z [106] 57

Chapitre III

Figure III. 1 *Cycle de développement classique d'un logiciel*....65
Figure III. 2 *Cycle de développement d'un logiciel développé avec la méthode B* 67
Figure III. 3 *Forme générale d'une machine abstraite* 69
Figure III. 4 *Processus de développement en B* [107]....70
Figure III. 5 *Machine et contexte* 72
Figure III. 6 *Relation entre composants d'un modèle Event-B* 72
Figure III. 7 *Les clauses possédant un contexte* 73
Figure III. 8 *Les clauses possédant une machine* 73
Figure III. 9 *Outils du noyau de la plateforme Rodin* [123]....78

Chapitre IV

Figure IV. 1 *Architecture d'un Q-Switch* [132]....81
Figure IV. 2 *Architecture du port de sortie du routeur Q-Swtich* 82
Figure IV. 3 *Règles des priorités à droite* [132] 83
Figure IV. 4 *Application des règles prioritaires dans les directions possible (A) est, (B) nord, (C) ouest, (D) sud* 84
Figure IV. 5 *Modélisation étape-par-étape de l'architecture NoC*....86
Figure IV. 6 *Niveau abstrait*....86
Figure IV. 7 *Modèle raffiné par ajout d'un réseau dans le modèle M0*....87
Figure IV. 8 *Transfert de paquet (p) entre les switchs*....87

Figure IV. 9 *Introduction des canaux* 88
Figure IV. 10 *L'ajout des Output Ports* 88
Figure IV. 11 *L'ajout des Input Ports* 89
Figure IV. 12 *Un réseau maillé avec 2D-Mesh* 91
Figure IV. 13 *Switchs : structure et liaisons* 91
Figure IV. 14 *Exemple d'une zone active* 95
Figure IV. 15 *(A) Cas possible (B) Cas impossible* 95
Figure IV. 16 *Modélisation étape-par-étape de l'architecture NoC* 96
Figure IV. 17 *Niveau abstrait* 96
Figure IV. 18 *L'envoie des paquets autant que flits* 97
Figure IV. 19 *Routage des flits en gardant une copie locale* 97
Figure IV. 20 *Initialisation des événements* 102
Figure IV. 21 *Activation de l'évènement Send* 102
Figure IV. 22 *Transmission de paquet dans le réseau* 103
Figure IV. 23 *Réception du paquet et suppression de la copie locale* 104
Figure IV. 24 *Système auto-organisé multi-nœuds en réseau appliqué aux communications sans fil* 107
Figure IV. 25 *Coloration des nœuds du système auto-organisé en vert* 108
Figure IV. 26 *Coloration des nœuds défaillants du système auto-organisé en rouge* 109
Figure IV. 27 *Coloration du nœuds test du système auto-organisé en bleu* 110

Liste des tableaux

Tableau IV. 1 *Sommaire des obligations de preuve*.....93
Tableau IV. 2 *Sommaire des statistiques des obligations de preuve*.....110

Liste des acronymes

ACK : ACKnowledgement
AMN : Abstract Machine Notation
ANoC : Asynchronous NoC
ASIC : Application Specific Integrated Circuit
BE : Best Effort
BIST : Built-In Self Test
bps : bit par seconde
CAO : Conception Assité par Ordinateur
CLB : Configurable Logic Block
CoNoChi : Configurable Network-on-Chip
CPU : Central Processing Unit
CT : Cut-Through
DfT : Design for Testability
DSM : Deep Sub-Micron
DyNoC : Dynamic NoC
FIFO : First In First Out
FLITS : FLow control uinITS
FPGA : Field Programmable Gate Array
GS : Guaranteed Services
ICAP : Internal Configuration Access Port
IP : Intellectual Property
JHDL : Java Hardware Description Language
LUT : Look Up Table
MPSoC : MultiProcessor System on Chip
MSI : Maximal Set Independance
NACK : Non-ACKnowledgement
NI : Network Interface
NoC : Network on Chip
OSI : Open Systems Interconnection
PE : Process Element
PO : Proof Obligation
PVS : Prototyp Verification on System
QNoC : Quality of service Network on Chip
QoS : Quality of Service
ReNoC : Reconfigurable Network-on-Chip
RTL : Register Transfer Language
SAF : Store-And-Forward
SdF : Sûreté de Fonctionnement
SNS : Smart Network Stack
SoC : System-on-Chip
SPIN : Scalabele Programmable Integrated Network
SVC : Switched Virtual Circuit
TDMA : Time-Division Multiple Access
TLA : Temporal Logic of Actions
TPO : Théorie du Premier Ordre
VDM : Vienna Development Method
VHDL : Very high speed integrated circuit Hardware Description Language

WH : WormHole

INTRODUCTION GENERALE

Etant donné l'évolution croissante de la complexité des systèmes sur puce (SoCs) associée à l'augmentation du nombre d'éléments de calcul intégrés dans les systèmes multiprocesseurs (MPSoCs), il apparait que le support de communication devient un des éléments clés. En effet, les MPSoCs nécessitent un support de communication doté d'une grande bande passante, et d'une forte adaptabilité et flexibilité. La vérification de tels systèmes complexes au cours de leur conception est, généralement, réalisée par simulation. Cette dernière permet la détection des erreurs grossières dans une conception et présente l'avantage d'évaluer les performances temporelles des systèmes modélisés. Cependant, cette approche ne garantit pas l'absence d'erreurs dans la conception établie, car elle ne permet pas de considérer toutes les configurations et les entrées possibles du système. L'utilisation des méthodes formelles, comme l'Event-B et, en particulier, le paradigme de correction par construction [1] [2] présentent un grand intérêt pour spécifier des systèmes électroniques. Ce paradigme offre une approche alternative à prouver et à définir des systèmes et architectures correctes en utilisant des étapes de raffinement et des techniques méthodologiques validées [3] [4] [5] [6].

Dans la littérature de nombreuses études ont porté sur l'utilisation des méthodes formelles pour vérifier les systèmes de communication et leurs protocoles. Le model-checking est une technique automatisée qui permet de vérifier si certains modèles d'un système satisfont une certaine spécification [7] [8]. Le modèle est décrit dans un modèle de type machine à états vérifiés en se basant sur une logique temporelle. Beaucoup de travaux utilisent le model-checking ou une composition de celui-ci et de démonstration des théorèmes. Le travail de Clarke et al., publié dans la référence [9], montre la vérification des propriétés temporelles de réseaux paramétrés en anneau et en arbre binaire. Une première étape consiste à utiliser une grammaire de réseau sans contexte pour modéliser des systèmes de communication du réseau. Puis, les propriétés temporelles sont vérifiées en utilisant un model-checking. Une spécification était développée qui satisfait un protocole de diffusion dans un réseau en arbre binaire en utilisant le model-checking SPIN et l'outil de démonstration de la méthode Coq [10]. Dans la référence [11], Curzon développe un modèle structurel du commutateur ATM Fairsile et compare sa spécification comportementale en utilisant l'outil de démonstration HOL [12] qui procède par déduction à partir d'axiomes et de règles d'inférence dans une logique donnée.

L'absence d'une impasse dans le réseau Æthereal a été vérifiée par utilisation d'outil PVS [13]. La vérification des communications sur puce NoC est l'un des principaux défis dans la vérification des SoCs [14] [15] qui reposent sur des réseaux intégrés complexes où les méthodes formelles sont utiles à leur conception, en particulier, pour les NoCs dits dynamiquement

reconfigurables. En effet, les NoC adaptatifs à base de technologie FPGA dynamiquement reconfigurables sont complexes à vérifier compte tenu de la principale difficulté liée aux composants IP (Inelectual Property) associés pouvant être dynamiquement relocalisés et exécutés. Cette complexité est sans cesse plus grande étant donné que l'évolution croissante des MPSoCs où les contraintes de coût et de performance, liées à la complexité et le nombre croissant de modules ou IP interconnectés, doivent être résolues. Les réseaux actuels de communication sur puce mettent en œuvre les transmissions de données par paquets entre les nœuds aux routeurs interconnectés. Parfois, les communications dans ces réseaux sont complexes ; c'est la principale raison pour laquelle des algorithmes de routage basés XY et tolérant aux fautes ont été développés [16]. Ces algorithmes permettent aux routeurs d'un réseau de détecter et corriger des erreurs de routage. En particulier, de nouvelles techniques de routage adaptatif et tolérant aux fautes avec détection d'erreurs basées XY pouvant modifier localement et temporairement les schémas d'acheminement ont été initialement introduites dans le fonctionnement sans faute du réseau [17]. Habituellement, les conceptions de telles structures NoC sont vérifiées par des simulations logiques pour détecter les erreurs. Cependant, ces simulations seules ne sont pas suffisantes pour détecter exhaustivement les erreurs et donc améliorer leurs architectures [18]. L'exploration des NoCs ont principalement portés sur la performance [19] [20] [21], la latence [22], la bande passante [23], l'estimation de la consommation [24], la détection et la correction des erreurs [25] [26], et les surfaces logiques utilisées. D'autres proposent des méthodes de routage pour les problèmes deadlock [27] [28] [29] visant la caractérisation du trafic [30].

L'objectif du travail de la thèse est d'intégrer les mécanismes de vérification formelle dans le flot de conception des systèmes électroniques. À cet effet, nous utilisons la méthode Event-B pour spécifier, vérifier et prouver le comportement des architectures NoCs. Notre objectif est de réduire le temps de simulation dans le flot de conception selon une méthode de preuve formelle par raffinement correspondant aux différentes étapes de la conception conjointe (matérielle, logicielle).

Le manuscrit est organisé comme suit :

- Le chapitre I présente une étude bibliographique sur l'évolution des réseaux sur puce et leurs métriques de performance. Ensuite, il détaille les différentes topologies NoC en expliquant leurs caractéristiques ; les réseaux sur puce reconfigurables et leurs limites sont également présentés. Ce chapitre se termine par la définition de la sûreté de fonctionnement (SdF) et application aux architectures NoC.

- Le chapitre II présente les techniques de vérification ; plus précisément, il relate les outils de conception et simulation qui testent la majorité des cas de fonctionnement à survenir et les

méthodes formelles qui prouvent la sûreté de fonctionnement des systèmes architecturaux.

- Le chapitre III explique comment est procédée une vérification formelle avec la méthode B ainsi que les avantages qu'une telle méthode apporte par rapport au domaine de conception. On parle aussi de l'évolution de la méthode B vers Event-B, et l'adaptation à la modélisation des systèmes par raffinement (correction par construction) au travers d'une description avec l'outil RODIN associé à la méthode B.

- Le chapitre IV propose et valide une méthodologie pour prototypage rapide d'un NoC spécifique (QNoC) en commençant par la structure interne du réseau associé où sont spécifiés les composants d'un routeur NoC et la vérification des mécanismes de fonctionnement. Après la vérification de l'algorithme de routage XY appliqué dans l'acheminement des paquets d'un routeur source à un routeur destination, une spécification et une vérification d'un algorithme de contournement tolérant aux fautes utilisé dans le cas de l'architecture considérée sont présentées ; la preuve des performances du NoC utilisé dans un système auto-organisé est aussi détaillée dans ce chapitre. Les développements expérimentaux réalisés montrent l'apport et l'intérêt de l'intégration de la preuve formelle dans les plans de conception de tels NoCs.

- Enfin, une conclusion générale dresse un bilan des résultats et discute les avantages et les inconvénients des travaux de recherche développés. Les perspectives apportées par les concepts architecturaux proposés dans ces travaux sont également présentées.

CHAPITRE I :

RESEAUX SUR PUCE

INTRODUCTION

Le réseau sur puce est un concept récent d'interconnexions dans les systèmes sur puces. Comme toute nouvelle technologie, elle requiert des efforts en recherche, en particulier, pour l'accélération et la simplification des phases de conception. L'espace de conception d'une architecture de communication sur puce se définit par plusieurs paramètres tels que la topologie, l'algorithme de routage, et la politique d'arbitrage. Par conséquent, le choix des valeurs des différents paramètres pour réaliser un réseau de communication simple et efficace est difficile et complexe. Puisque, chaque décision a pour but d'optimiser la fonction de coût du réseau et de satisfaire les contraintes.

Le but de ce premier chapitre est d'introduire le paradigme des réseaux sur puce ainsi que les notions qui lui sont associées. À cet effet, nous allons décrire les points de caractérisation des réseaux sur puce ; à savoir : les métriques de performances, l'architecture, la qualité de service, la reconfiguration à base de FPGA, les limites, la conception, et la sûreté de fonctionnement.

I.1. EMERGENCE DES RESEAUX SUR PUCE

I.1.1. Evolution des systèmes sur puce et l'exigence de nouvelles structures

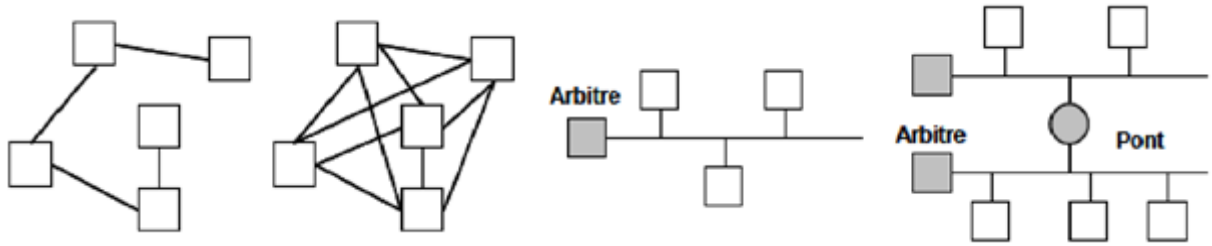
Durant les deux dernières décennies, l'évolution des technologies d'intégration sur silicium, respectant la loi de Moore [31], a permis d'intégrer sur une seule puce un système complet (processeurs, coprocesseurs, mémoires, etc.). Or, pour répondre aux besoins des applications émergentes qui regroupent différentes fonctionnalités au sein de la même architecture, les systèmes sur puce deviennent de plus en plus complexes. De plus, dans le but de réduire le temps de développement, les systèmes sont souvent construits en réutilisant des blocs préconçus et validés couramment désignés comme des blocs IP (Intellectual Property). C'est pourquoi, l'évolution du système tend vers une augmentation de la complexité et du nombre d'IP entre plusieurs dizaines à centaines d'unités de traitement [32].

Dès lors, un nouveau problème apparaît quant à l'interconnexion de ces nombreux éléments. En effet, les structures de communication classiques sont performantes lorsque le nombre d'unités à interconnecter est réduit. En outre, les fils d'interconnexion ne bénéficient pas des avantages liés aux réductions technologiques et la plus grande partie de la consommation énergétique et de la surface utilisée provient aujourd'hui des interconnexions [33]. Par conséquent, la conception des nouveaux systèmes sur puce se focalise sur la structure de communication qui, à l'instar des IPs, doit pouvoir être réutilisable et faire face aux évolutions technologiques [34] [35]. Et au nombre

croissant de blocs de calcul à interconnecter. Pour faire face à cette évolution actuelle, les nouvelles structures de communication doivent être flexibles. Ainsi, grâce à un interfaçage approprié, de nombreux IPs hétérogènes pourront être interconnectés. Ensuite, la structure doit avoir la capacité d'être étendue sans dégrader ses performances. D'autre part, de plus en plus de circuits auront des horloges différentes qui nécessiteront une synchronisation globale du système au travers d'une implantation Globalement Asynchrone Localement Synchrone (GALS) [36] [37] du système de communication. Enfin, dans les technologies en dessous de 90 nm, des phénomènes indésirables, dits effets submicroniques profonds (en anglais Deep Sub-Micron - DSM) [38], vont générer des bruits sur les interconnexions entrainant des erreurs de transmissions. C'est pourquoi, ces erreurs devront être évitées lors de la conception à l'aide d'architectures dont les propriétés électriques sont prévisibles et maîtrisées, ou bien par le biais de mécanismes de détection/correction d'erreurs ou de tolérance aux fautes.

I.1.2. Structures actuelles de communication intégrée

Traditionnellement, les structures de communication utilisées pour les systèmes sur puce sont globalement basées soit sur des connexions point-à-point entre les éléments communicants, soit sur des bus partagés [39].



(a) Point à Point (b) Crossbar (c) Bus partagé (d) Bus hiérarchique

FIGURE I. 1 ***STRUCTURES DE COMMUNICATION TRADITIONNELLE***

1. La connexion point à point (P2P)

Dans une connexion point à point (Figure I.1(a)), des liens dédiés sont établis entre chaque couple d'éléments communicants. Cette solution offre de nombreux avantages en termes de performances car les contraintes en latence et en débit peuvent être parfaitement respectées.

De plus, il ne peut y avoir des congestions et aucun arbitrage n'est nécessaire pour l'accès au médium de communication [40].

Cependant, la connexion point à point ne fournit aucune tolérance aux fautes/pannes pouvant survenir sur un lien. D'autre part, ce mode de connexion ne présente aucune flexibilité car elle est

adaptée qu'à une architecture donnée et, donc, difficilement réutilisable. Enfin, une grande partie de la bande passante est typiquement perdue car les liens ne sont que très peu utilisés [41].

2. La connexion complète (Crossbar)

Si tous les éléments communicants sont reliés entre eux par des connexions point-à-point, la connexion est dite complète [42] (Figure I.1(b)). Dans ce cas, la structure tolère mieux les pannes mais les liens sont encore davantage sous-exploités. En outre, le coût en termes d'interconnexions est le plus élevé, ce qui limite énormément l'extensibilité car il n'est pas concevable d'utiliser ce type d'architectures lorsque le nombre d'unités à interconnecter est très grand.

3. Le bus partagé

Le bus partagé répond aux inconvénients des bus P2P ou Crossbar ; c'est pourquoi, dans les systèmes sur puce complexes, le moyen de communication actuellement le plus répandu dans l'industrie est le bus partagé (Figure I.1(c)). En effet, ils sont beaucoup plus flexibles et mieux réutilisables comparés aux liaisons point à point.

En revanche, l'inconvénient principal des bus partagés provient du fait qu'ils n'autorisent qu'une seule connexion à la fois. La bande passante allouée à chaque élément diminue donc avec le nombre d'éléments communicants. De plus, les bus requièrent un mécanisme supplémentaire d'arbitrage permettant de traiter les demandes d'accès concurrentes. Du point de vue physique, les longueurs de fils dans les bus sont souvent plus grandes, ce qui aboutit à des consommations d'énergie plus élevées et des délais de transmission pouvant dépasser la période d'horloge du système [43].

4. Le bus hiérarchique

Dans le but d'augmenter l'extensibilité et les performances des bus partagés, il existe une structure appelée bus hiérarchique [4] (Figure I.1(d)), qui permet de connecter plusieurs bus partagés à l'aide de ponts (bridges). Ces derniers peuvent servir de convertisseur de protocoles si les bus n'utilisent pas le même protocole. Cependant, les connexions entre les bus ne sont optimales que lorsqu'il n'y a pratiquement pas de communications sur les deux bus. En outre, le problème d'extensibilité est réduit mais pas résolu.

I.1.3. Paradigme des réseaux sur puce

Même si les bus partagés ont servi pendant de nombreuses années dans les systèmes sur puce, leurs défauts face aux besoins des applications émergentes ont été montrés par l'évolution des technologies d'intégration ainsi que la convergence des applications. Ainsi, il est très difficile

d'ajouter de nouveaux éléments communicants à un bus car, d'une part, la bande passante allouée à chaque élément diminue lorsque le nombre d'éléments augmente, et, d'autre part, le contrôle du bus se complexifie limitant ainsi l'extensibilité de cette structure de communication. Ainsi, ces limitations rendent les bus incompatibles avec les futures générations de circuits intégrant un très grand nombre de ressources de traitement [44]. Or, ce genre de problèmes a déjà été étudié et des solutions ont été trouvées dans le contexte des réseaux locaux [45] et des réseaux d'interconnexion de machines parallèles [46]. En effet, des structures de communication distribuées, basées sur un découpage des communications en couches, ont permis de maîtriser la complexité des systèmes de communication et de faire évoluer les médiums de transmission et la nature des données transportées tout en gardant une compatibilité entre les systèmes. Partant de ce constat, l'idée d'intégrer un réseau de communication sur une puce est née au début des années 2000 [47] avec le paradigme des réseaux sur puce (NoCs) [32] [41] [48], dont le nom dérive de système sur puce (SoC). Le principe est d'adapter les solutions des réseaux non intégrés au contexte des SoCs en tenant compte de la latence, la surface ou la consommation d'énergie dont les contraintes sont spécifiques aux circuits intégrés.

I.1.4. Couches réseaux dans les NoCs

Nous ne pouvions pas introduire le concept des réseaux sur puce sans parler de la pile protocolaire. En effet, dans le domaine des réseaux, l'ensemble des règles et des méthodes pour l'échange des données entre les sources et les destinations représente le protocole de communication. Or, afin que sa mise en œuvre soit la moins complexe possible, le protocole a été divisé en plusieurs couches ayant chacune des fonctionnalités spécifiques. Il s'agit du modèle Open Systems Interconnection (OSI) [49] qui découpe le protocole de communication des réseaux en sept couches (physique, liaison de données, réseau, transport, session, présentation et application). Cependant, ce découpage n'est pas approprié pour les réseaux sur puce car, généralement, seules les quatre premières couches sont implantées [50] [51].

Pour mieux comprendre les différents points abordés dans cette partie, il est indispensable de définir quelques éléments de base caractérisant le format des données circulant à travers un NoC :

- L'information que l'on veut transmettre est appelée message et représente la totalité de l'information à transmettre. Le message peut contenir autant de données qu'il le souhaite.
- Un message est découpé en paquets (packet) pour permettre les communications en parallèle ou lorsque le message est trop grand.
- Un paquet peut être décomposé en FLIT (FLow control unIT) : il correspond à la plus petite unité pour transmettre des données.

I.2. MÉTRIQUES DE PERFORMANCES

Afin de caractériser et de mesurer les performances d'un réseau, plusieurs métriques ont été proposées [52]. Ces métriques sont : le diamètre, la bande passante utile, la bande passante de bissection, la latence, la surface, la diversité de chemin, l'extensibilité et la flexibilité.

I.2.1. Le diamètre

Le diamètre d'un réseau est défini comme étant la distance maximale entre deux nœuds du réseau. Cette première métrique sert aussi à classer les topologies de réseau. En effet, il existe trois types de diamètre [41] :

- À longueur de ligne fixe ; c'est le cas des topologies qui ont un diamètre évoluant en fonction du nombre de nœuds interconnectés dans le réseau. Un exemple typique est le Crossbar. Ce genre de topologie n'est pas facilement extensible et est donc plus approprié pour les réseaux avec un nombre limité de nœuds.

- Linéaire : l'exemple le plus courant est celui de la topologie maillée qui a l'avantage d'être facilement étendue.

- Logarithmique : nous pouvons citer le cas des réseaux hyper-cubiques, en arbre ou encore multi-étages. Les topologies à diamètre logarithmique conviennent mieux lorsque le nombre de nœuds à connecter devient important.

I.2.2. La bande passante utile

Une deuxième métrique caractérisant les performances d'un réseau est le débit de données effectif ou bande passante utile [4]. En effet, le débit de données simple est une caractéristique permettant de quantifier le volume de données transitant dans le réseau. Ainsi, chaque élément du réseau (liens, nœuds) peut alors être caractérisé par sa bande passante qui est couramment exprimée en bit par seconde (bps). Cependant, cette valeur ne reflète pas le fonctionnement du réseau. C'est pourquoi, la bande passante utile est une métrique plus pertinente. Elle correspond au débit de données en réception cumulé sur l'ensemble des ressources connectées au réseau.

I.2.3. La bande passante de bissection

Il existe un deuxième type de bande passante permettant de caractériser un réseau. Il s'agit de la bande passante de bissection qui est calculé à partir de la largeur de bissection du réseau. Cette largeur représente le nombre minimum d'arcs permettant de diviser le graphe du réseau en deux sous-ensembles de nœuds de même cardinalité (plus ou moins un nœud). Ainsi, la somme des

bandes passantes sur les fils associés aux arcs représente la bande passante de bissection. Cette-dernière est souvent utilisée pour mesurer la localité dans les communications ainsi que les goulots d'étranglement susceptibles de se produire.

I.2.4. La latence

Souvent associée à un besoin de qualité de service, l'estimation de la latence des transferts est un critère de performance important. Selon que la mesure soit faite au niveau du flit, du paquet ou du message, la latence peut avoir plusieurs définitions. Ainsi, lorsque la latence est associée à un flit, elle correspond au temps écoulé entre l'introduction du flit dans le réseau et sa réception par la ressource destinatrice. Dans le cas du paquet, il s'agit du temps entre l'envoi du premier flit et la réception du dernier du paquet. Enfin, de façon analogue, la latence d'un message est mesurée par le temps entre l'envoi du premier paquet et la réception du dernier contenant le message.

I.2.5. La surface

La surface est une des premières métriques qui est évaluée car elle contribue directement au coût du circuit à fabriquer. Cependant, à partir d'une certaine échelle, sa miniaturisation génère des effets de bord (diaphonie, délais de propagation, etc.) qui doivent être pris en compte (évités ou gérés) pour ne pas altérer le comportement du système.

I.2.6. La diversité de chemin

La diversité de chemin [53] est une métrique qui va caractériser la capacité du réseau à être tolérant aux fautes. En effet, un réseau offre de la diversité de chemin si pour tous (ou presque tous) les couples (source/destination), il existe plusieurs chemins entre eux. C'est pourquoi, lorsque la diversité de chemin est faible ou inexistante, cela peut entrainer des problèmes de contentions à travers tout le réseau.

I.2.7. L'extensibilité

Avec l'évolution des technologies, l'intégration des systèmes sur puce ne cesse d'augmenter et avec elle, le nombre de composants à interconnecter. En conséquence, il est important que le réseau sur puce soit capable d'être étendu. Cette extensibilité du réseau [44] représente son aptitude à augmenter efficacement les performances en fonction du nombre d'éléments communicants. En particulier, la bande passante utile doit augmenter proportionnellement lorsque de nouveaux éléments de traitement sont rajoutés au réseau.

I.2.8. La flexibilité

La dernière métrique de caractérisation de performance est la flexibilité [54] qui représente le degré d'adaptation du réseau à différents éléments communicants hétérogènes et à sa capacité de réutilisation dans de futurs systèmes. Cependant, il est assez difficile de concilier ces deux exigences avec des besoins en performances. En effet, typiquement un réseau conçu spécifiquement pour une application donnée afin d'offrir des performances optimales ne pourra pas être réutilisé pour d'autres applications. A l'inverse, un réseau plus générique sera moins performant mais plus flexible.

I.3. ARCHITECTURE DU NoC

Les NoCs sont principalement caractérisés par leurs architectures et leurs protocoles de communication. Alors que les premières définissent les relations structurelles entre les éléments du réseau, les seconds spécifient la manière selon laquelle le réseau doit se comporter en fonction de conditions variées [55].

Dans cette section sont décrits les éléments constitutifs du réseau ; à savoir : les adaptateurs réseaux, les nœuds ou routeurs et les liens. Les différentes topologies les plus couramment implantées dans les NoCs sont aussi présentées.

I.3.1. Composants d'un réseau sur puce (NoC)

Le réseau sur puce est composé de nœuds (aussi appelés routeurs ou switchs) qui servent à transmettre les données selon un protocole de communication choisi. Les données sont propagées entre les routeurs à travers les liens de communication point à point monodirectionnels ou bidirectionnels. Quant à l'interfaçage entre le réseau et les éléments de traitement, il est réalisé par des adaptateurs réseaux [45].

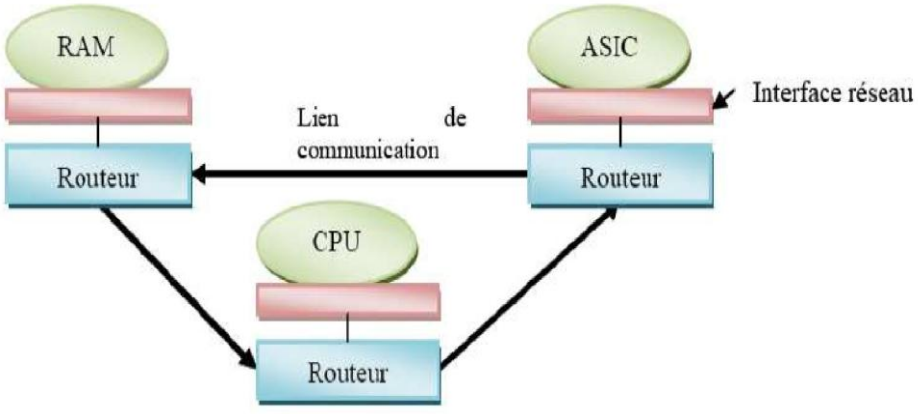


FIGURE I. 2 *STRUCTURE D'UN RESEAU SUR PUCE* [56]

1. Les nœuds ou routeurs

Le rôle principal d'un nœud de routage est d'acheminer les données d'une source à une destination. Il est constitué de :

- ***File d'attente*** pour stoker les paquets qui transitent dans le réseau, elle est généralement matérialisée par des mémoires FIFO (First In First Out) ;
- ***Un commutateur*** qui connecte les files d'entrées aux ports (ou files) de sorties ;
- ***Une unité de routage et d'arbitrage*** qui assure la fonction d'aiguillage et gère les situations de conflits.

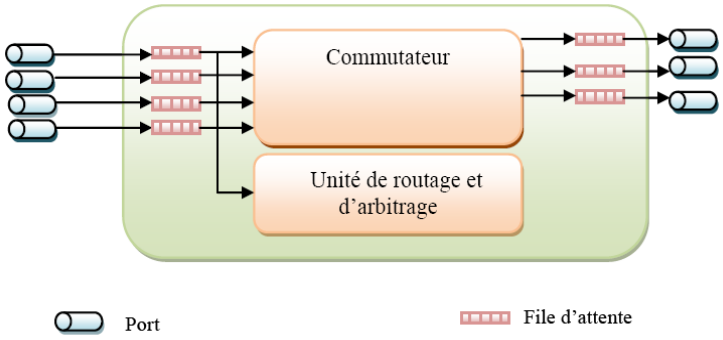


FIGURE I. 3 *ILLUSTRATION DE LA STRUCTURE D'UN ROUTEUR D'UN RESEAU SUR PUCE* **[3]**

Chaque routeur peut être connecté à une ressource de l'application (unités de calcules, mémoires, etc.) qui exécute une fonction particulière et peuvent être matérialisée par un ASIC, processeur ou FPGA.

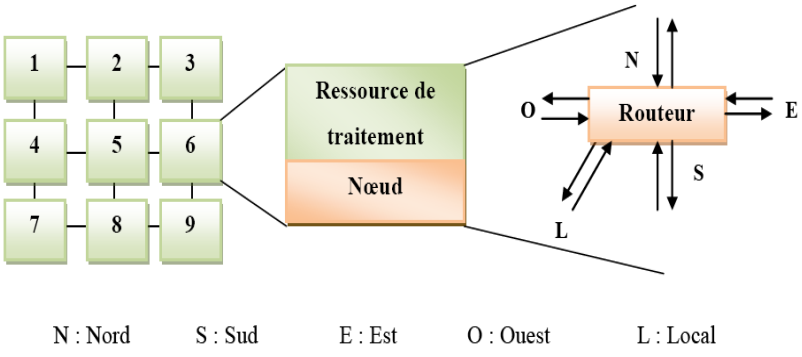


FIGURE I. 4 *EXEMPLE DE STRUCTURE D'UNE TUILE D'UN NOC* **[57]**

A- File d'attente

Le rôle de la temporisation des paquets est d'éviter la destruction des paquets en conflits. Ces derniers peuvent être temporisés dans des files d'attentes. Or, dans la majorité des NoCs, la surface consommée par un routeur provient de la place prise par ces files. Dès lors, un compromis doit être

trouvé lors de la conception du routeur pour limiter au maximum la surface utilisée par les files d'attente sans pour autant dégrader les performances requises. Les files d'attentes sont principalement caractérisées par leur taille et leur position au sein du routeur. La taille des files d'attente comprend la largeur des files en nombre de bits et leur profondeur définissant le nombre de mots qui peuvent être stockés. Le premier paramètre qui influe directement sur la taille des files d'attente est le mode de commutation choisi [44]. Concernant la largeur de la file, elle ne pourra pas être plus petite que celle d'une unité de contrôle de flux (FLow Control digit Unit). De plus, la taille doit être rigoureusement déterminée car elle peut affecter la fréquence d'horloge maximale et le coût en termes de surface et de consommation d'énergie des routeurs. Enfin, une taille inappropriée peut impliquer des congestions au sein du réseau avec une réduction de la bande passante utile. Les files d'attente peuvent être placées à différentes positions dans le routeur : en entrée, en sortie ou en sortie virtuelle [58].

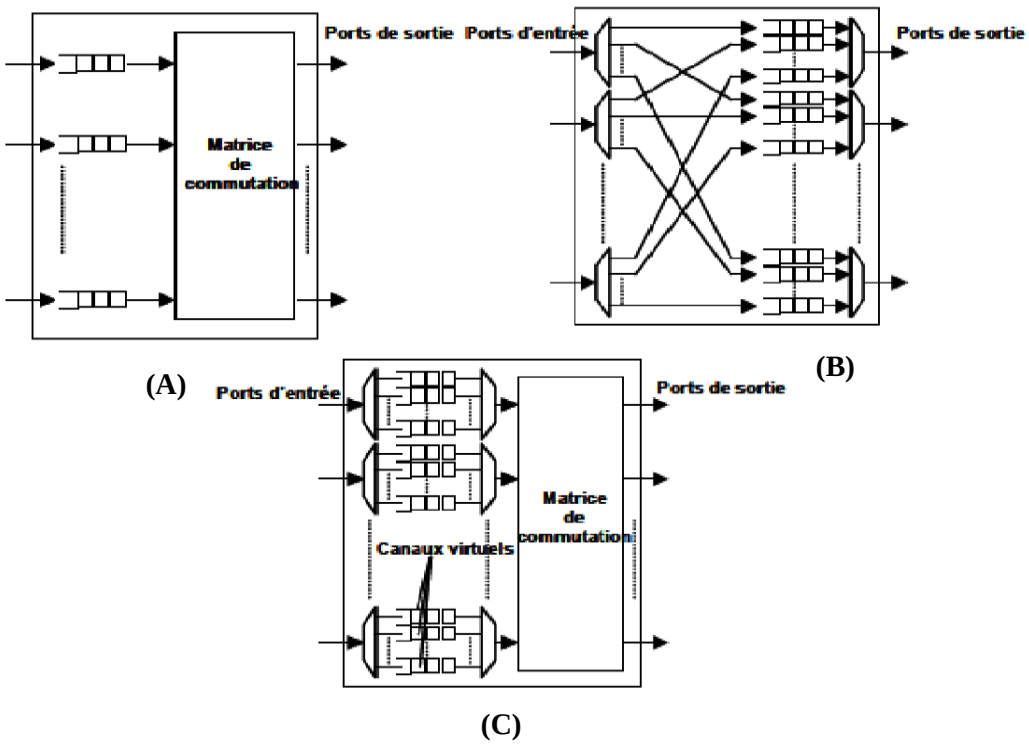


FIGURE I. 5 *POSITIONNEMENTS DES FILES D'ATTENTES DANS UN ROUTEUR : (A) EN ENTREE (B) EN SORTIE (C) EN SORTIE VIRTUELLE*

a) Files d'attente en entrée

Dans ce premier cas, chaque port d'entrée du routeur possède une file d'attente. Bien que cette technique soit la moins coûteuse en surface, elle peut induire une saturation à cause du blocage de tête de ligne (head-of-line) [59]. Cela arrive lorsqu'une donnée en tête de file ne peut accéder au port de sortie associé bloquant ainsi les autres paquets de la file même si leur sortie est libre.

b) Files d'attente en sortie

Lorsque les files d'attente sont positionnées en sortie du routeur, chaque port de sortie a autant de files d'attente que de ports d'entrée. Cette technique offre de meilleures performances que la précédente mais la surface est forcément plus importante.

c) Files d'attente en sortie virtuelle ou canaux virtuels

L'idée des files d'attente en sortie virtuelle est de combiner les avantages des deux techniques précédentes. Ainsi, chaque port d'entrée possède plusieurs files d'attente servant à répartir les paquets entrant en fonction de leur destination ou bien de leur priorité. On parle alors de canaux virtuels car pour un unique canal physique, il y aura autant de canaux virtuels que de files d'attente [60] [17]. Bien qu'ils impliquent une augmentation de la surface et de la latence à cause de la mémorisation et du contrôle nécessaires, les canaux virtuels possèdent plusieurs avantages :

- ils évitent les interblocages (les paquets bloqués sont stockés dans différents canaux virtuels pour laisser passer les autres paquets) [61] [62] ;
- ils optimisent l'utilisation des liens (liens rarement laissés dans un état oisif) ;
- ils fournissent des services séparés (à travers des trafics séparés et des niveaux de priorités différents).

B- Matrice de commutation

Les ports d'entrée et de sortie du routeur sont tous connectés les uns aux autres grâce à une matrice de commutation [55]. En effet, la matrice a pour rôle de multiplexer les données en entrées du routeur vers ses ports de sortie.

C- Unité de routage et d'arbitrage

Lorsqu'au sein d'un routeur au moins deux paquets souhaitent sortir par le même port et ce, en même temps, un arbitrage doit être fait afin de choisir celui qui sortira le premier. Bien qu'il existe différentes techniques pour arbitrer les paquets en conflit, la caractéristique la plus commune est l'impartialité. En effet, l'arbitre doit être capable de fournir un accès équitable aux ports de sorties si les paquets ont la même priorité. Les principales politiques d'arbitrage que l'on puisse trouver dans les NoCs sont l'accès multiple à répartition dans le temps (TDMA), la réservation de Time-slot, le tourniquet ou Round-robin, et la priorité fixe [13].

2. Interface Réseau

Une interface réseau traduit les messages envoyés par les modules en requêtes compréhensibles à l'interconnexion [4], ceci permet de séparer matériellement la fonction « calcul » de la fonction « communication » et des IPs réutilisables [18] . Elle est composée d'une partie *back-end* liée au routeur et d'une partie *front-end* connectée à la ressource de traitement. Le second bloc implémente des standards qui réalisent l'adaptation des protocoles de communication entre la ressource et le routeur. [16]

3. Liens de communication

Les liens sont des connexions logiques entre deux (ou plusieurs) éléments communicants [48] [59]. Ils peuvent être composés d'un ou plusieurs canaux physiques. Ces derniers servent à transporter l'information entre deux routeurs ou encore entre un routeur et une interface réseau. C'est pourquoi, bien qu'étant réduits, les problèmes de délais et de dispersion des signaux communément segmentés par des répéteurs, le plus souvent des buffers, qui permettent de restaurer le niveau de la tension des fils. Dès lors que les fils ont une longueur significative, les répéteurs peuvent incorporer des registres pour *pipeliner* la transmission des données. Ainsi, celles-ci arrivent à une cadence qui coïncide avec leur utilisation par les éléments de calculs connectés au réseau. Enfin, il existe trois types de liens qui sont définis par le sens des transmissions. Ils peuvent être unidirectionnels en entrée, unidirectionnels en sortie ou bidirectionnels (entrée/sortie).

I.3.2. Caractéristiques des réseaux sur puce

Un réseau sur puce est défini par une topologie, un mode de commutation, une stratégie de routage et une politique de contrôle de flux. Ces différentes caractéristiques sont présentées dans cette section.

A- Topologie

De nombreuses topologies des réseaux sur puce sont actuellement disponibles. Une topologie déni la manière dont les nœuds, les routeurs et les liaisons de données sont connectés les uns aux autres. Les topologies des réseaux sur puce peuvent être classées en 3 grands groupes [1] : les réseaux directs, indirects et les réseaux irréguliers.

Dans les réseaux directs, chaque nœud a des liaisons point-à-point avec un ensemble de nœuds voisins [1]. Les nœuds contiennent les blocs de calcul et/ou de mémorisation et des modules d'interfaçage avec le routeur (les blocs NI - Network Interface) [16]. Dans ces réseaux directs, un routeur est connecté à d'autres routeurs voisins à travers les canaux de communication.

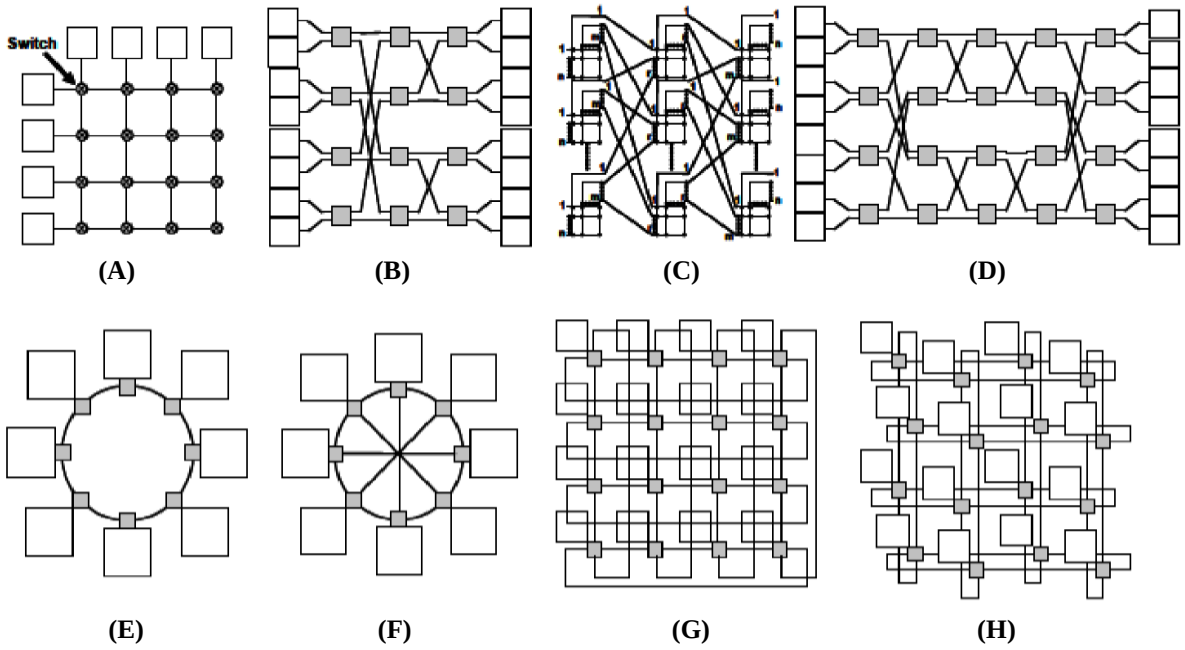
- **Les réseaux sur puce de type direct :** la plupart des topologies de type direct ont une

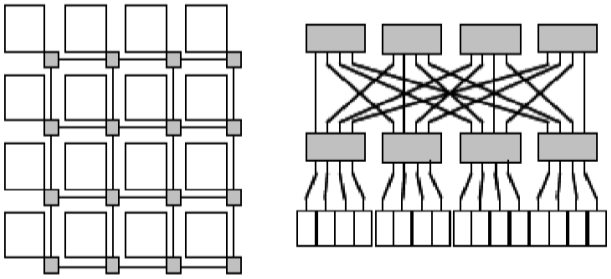
structure orthogonale où les nœuds sont disposés de telle manière que chaque canal produit un déplacement dans une seule direction. Le routage peut être, matériellement, facilement implanté. Les réseaux sur puce directs les plus souvent utilisés sont les réseaux maillés : 2D-mesh, torus, folded torus et octagon [5] [63].

- **Les réseaux sur puce de type indirect :** chaque nœud est connecté à un routeur qui est connecté à d'autres routeurs par des liaisons point-à-point. Le réseau sur puce de type indirect le plus simple est le réseau Crossbar où chaque élément de calcul PE (Process Element) est connecté à d'autres PE via un seul routeur [64]. Cependant, le principal inconvénient de ce type de réseau indirect Crossbar est son manque d'extensibilité et son coût relativement important en termes de connectivité proportionnel au nombre de PE. C'est la raison pour laquelle, ce réseau est rarement utilisé.

- **Les réseaux irréguliers :** Ce sont des structures d'interconnexion qui représentent généralement une combinaison de bus partagé ou hiérarchique avec des topologies de type direct et indirect. Ce genre de topologie est souvent associé à des applications spécifiques [18].

Il existe une grande variété de topologies de réseaux [55], qu'elles soient directes ou indirectes, régulières ou irrégulières. Evidemment, la liste des topologies qui suit n'est pas exhaustive mais elle représente celles qui sont le plus souvent utilisées dans les NoCs.





(I) (J)

FIGURE I. 6 ***TOPOLOGIES SOUVENT UTILISEES DANS LES NOCS : (A) CROSSBAR (B) BUTTERY (C) CLOS (D) BENES (E) ANNEAU (RING) (F) ANNEAU A CORDES (CHORDAL RING) (G) TORE (TORUS) (H) TORE EN QUINCONCE (FOLDED TORUS) (I) MAILLAGE A DEUX DIMENSIONS (2D-MESH) (J) ARBRE ELARGI (FAT TREE)***

La figure (Figure I.6 (A)) présente une topologie Crossbar, aussi appelé commutateur de croisement qui permet les connections directes de toutes les entrées et sorties sans étage intermédiaire. En effet, ce commutateur consiste généralement en une matrice de multiplexeurs [53]. Ils ont l'avantage d'être non bloquants et d'effectuer, en l'absence de conflits, la commutation des entrées vers les sorties en un cycle d'horloge.

Les Crossbars consomment généralement moins d'énergie et offrent des débits et un parallélisme supérieur à ceux des bus traditionnels. Les inconvénients majeurs résident dans leur coût élevé et leur extensibilité très limitée.

Afin de diminuer la complexité des Crossbars, il est possible de les réaliser en cascadant plusieurs étages de petits Crossbars (2 entrée/sorties). La structure équivalente ainsi obtenue est un exemple de réalisation d'un réseau de type multi-étages [55]. Dans cette catégorie de réseaux, un compromis est fait entre la complexité en termes de surface et de latence. En effet, le diamètre du réseau est augmenté. Un autre type de réseau cubique (k-ary n-cubes) [53] sont constitués de k^n nœuds répartis dans un cube à n dimensions avec k nœuds dans chacune d'elles. L'exemple de réseau cubique le plus simple est l'anneau simple (ring) (voir Figure I.6 (E)) ou l'anneau à cordes (chordal ring) (voir Figure I.6 (F)) qui est un k-ary 1-cube. Les coûts en surface et puissance dissipée d'un anneau croissent linéairement avec le nombre d'éléments connectés au réseau. Quant aux performances, elles diminuent avec la taille du réseau car la bande passante de bissection est très limitée. Un deuxième réseau cubique très connu est le tore (torus) qui est un k-ary 2-cube (voir Figure I.6 (G)). Il peut être vu comme un réseau à maille à deux dimensions avec des liens rajoutés afin de rendre régulier. Dans le but de diminuer la longueur de ces liens, les routeurs peuvent être disposés en quinconce (folded torus) (voir Figure I.6 (H)). La surface du réseau en tore est à peu près identique à celle du réseau maillé mais l'énergie consommée est plus petite et les performances sont meilleures car la distance moyenne est inférieure.

Les réseaux maillés (n-dimensional k-ary mesh) sont très proches des réseaux cubiques puisqu'ils

ont pratiquement la même structure. La seule différence réside dans le fait que les liens connectant les routeurs aux extrémités d'une même colonne (ou ligne) de routeurs sont supprimés.

La topologie qui est le plus couramment utilisée dans les NoCs est la maille à deux dimensions, ou k-ary 2-D mesh (Figure I.6 (I)). Son implantation au niveau layout est plus simple car les routeurs peuvent être pré-placés et tous les liens ont la même longueur. Quant à la surface, elle grossit linéairement avec la taille du réseau [55]. Cependant, ce type de réseau possède une distance moyenne relativement grande et une bande passante de bissection limitée, ce qui réduit les performances lorsque le réseau est très chargé. Un autre problème qui peut survenir dans cette topologie est l'accumulation du trafic au centre du réseau, ce qui produit le phénomène de hot-spot.

Dans les topologies en arbre (d-dimensional k-ary (fat) trees), la bande passante de bissection est faible car le trafic est concentré au niveau de la racine de l'arbre. C'est pourquoi, une solution possible au problème est de dupliquer la racine. La topologie ainsi obtenue est appelée arbre élargi [65] (Figure I.6 (J)). Elle comprend des niveaux de routeurs dans lesquels chaque routeur possède k liens vers des routeurs fils. Les interfaces réseaux sont attachées seulement aux feuilles de l'arbre [47]. Bien que la bande passante de bissection soit grandement élargie, le coût en surface est plus élevé. De plus, le layout des arbres élargis est plus difficile à implanter lorsque le nombre de nœuds est très grand en comparaison avec le layout des topologies à maille ou des tores.

Au final, le choix de la topologie est complexe. Il est fortement lié aux contraintes de l'application et au trafic sur le réseau. Il résulte d'un compromis entre les performances intrinsèques de la topologie et le coût que représente son implantation.

B- Modes de commutation (switching)

Il existe deux techniques qui sont principalement utilisées : la commutation de circuit et la commutation de paquet [20].

a) La commutation de circuit

Avec la commutation de circuit [42] [4], une connexion est établie entre la source de données et sa destination et est maintenue pour toute la durée de la communication. Ainsi, des éléments du réseau (liens, nœuds) sont alloués pour le transfert de données et ne peuvent pas être utilisés pour d'autres communications entre d'autres couples d'émetteur/récepteur tant que la connexion reste active. Dans ce mode, les messages sont transmis entièrement en suivant le même chemin de la source à la destination. Lorsque la connexion est terminée, le chemin est alors « libéré ». Les informations de contrôle (pour le début et la fin de connexion par exemple) sont séparées des données. Ce

mécanisme est bien adapté lorsque l'on veut exécuter des transferts de données importants entre une même source et destination, car il rentabilise le temps passé à négocier la connexion, et garantit une qualité de service. De plus, il n'y a pas de blocages possibles.

b) La commutation de paquet

Lors d'une transmission de données par commutation de paquet [42] [4], les données à échanger sont découpées en paquets et sont émis et acheminés indépendamment sur le réseau. Cela permet d'accéder au réseau plus vite car il n'y a pas besoin d'établir une connexion au préalable entre les routeurs du chemin de paquets. Les paquets ainsi émis peuvent suivre des acheminements différents dans le réseau et sont réassemblés à l'arrivée. C'est pourquoi, l'ensemble des paquets du message doivent contenir des informations sur la destination. Il peut aussi arriver que des paquets se perdent, auquel cas ils devront être retransmis. Dans ce mode, les délais de transmission sont non-déterministes à cause des contentions possibles ou de perte de paquets nécessite de renvois de paquets. Ainsi, cette technique de commutation requiert des mécanismes de contrôle de congestion et de flux.

Enfin, la structure des routeurs est plus complexe que pour un réseau implantant la commutation de circuit.

La commutation de paquets présente principalement quatre politiques de mémorisation des paquets au sein du réseau :

- Différé ou Store-And-Forward (SAF).
- Passage à travers ou Cut-Through (CT).
- Trou de ver ou WormHole (WH).
- Circuit virtuel commuté ou Switched Virtual Circuit (SVC).

- **Différé ou Store-And-Forward (SAF) :** La politique SAF correspond à un protocole de commutation de paquets dans lequel les routeurs vont stocker totalement les paquets avant de les renvoyer. Ainsi, il est possible que le paquet doive attendre si le routeur suivant n'est pas disponible ou ne dispose pas suffisamment de place pour le stocker [66].

Bien que le SAF permet d'utiliser des algorithmes de routage plus élaborés, il introduit une latence à chaque routeur traversé. En outre, le SAF requiert une quantité d'espace de stockage importante car les routeurs ont besoin de mémoriser plusieurs paquets complets en même temps.

- **Passage à travers ou Cut-Through (CT) :** Cette technique de commutation CT [44] a été introduite dans le but de réduire la latence du modèle SAF. En effet, dans le cas du CT, la granularité des éléments commutés est plus petite qu'un paquet. Ce dernier est fractionné en flits. Avec ce mode, un nœud peut commencer à envoyer un paquet si le nœud suivant lui garantit qu'il peut stocker le paquet dans sa totalité. Dans le cas contraire, le nœud doit pouvoir garder le paquet. La capacité de mémorisation du nœud est donc la même que pour le mode SAF mais la latence est diminuée en absence de contention, puisqu'il n'est plus nécessaire d'attendre la réception complète du paquet pour l'acheminer d'un nœud à un autre.
- **Trou-de-ver ou WormHole (WH) :** La politique WH est basée sur le principe que les flits d'un paquet peuvent être transmis d'un routeur à l'autre dès qu'un flit peut être stocké et non plus nécessairement dès qu'un paquet complet peut être stocké [42] [4] [44]. Ainsi, la taille des files d'attente en entrée des routeurs est plus petite, de l'ordre de quelques flits. Cette technique de commutation offre une faible latence pour la traversée des routeurs car ces derniers n'ont plus à stocker entièrement les paquets. En outre, le coût en surface est réduit car les files d'attente sont beaucoup plus petites. Cependant, il peut arriver qu'un paquet occupe plusieurs routeurs en même temps, et dans ce cas, il est possible qu'il bloque la transmission d'autres paquets, aboutissant à une contention en cascade, voire même à une situation de blocage cyclique appelée interblocage [61].
- **Circuit virtuel commuté ou Switched Virtual Circuit (SVC) :** La technique SVC [44] consiste à définir des circuits virtuels entre une source et une destination, et ce, au sein d'un réseau à commutation de paquets. Dans ce modèle, un paquet spécial est d'abord envoyé dans le réseau afin de réserver les ressources nécessaires pour le chemin entre la source et la destination correspondant au circuit virtuel entre les deux. Lorsque le chemin a bien été établi, l'émission des données peut commencer. Cette technique n'est efficace que si le temps de transmission des données est beaucoup plus long que le temps d'établissement du circuit virtuel.

C- L'adressage et les techniques de routage

a) L'adressage

Dans le but de router les paquets dans un réseau sur puce, une identification ou adresse unique [55] doit être affectée à chaque élément de destination susceptible d'être atteint. C'est pourquoi, il est assez courant d'utiliser un adressage logique qui est ensuite traduit en adresses et sous-adresses physiques. En effet, certaines destinations pouvant être liées hiérarchiquement, il faut alors pourvoir identifier les différents niveaux d'adresses. L'espace des adresses physiques peut être assigné en fonction du nombre d'éléments interconnectés ou alors en fonction de la position relative de ces éléments dans le réseau.

b) Les techniques de routage

Le mécanisme de routage au sein du NoC est responsable de la transmission correcte et efficace des paquets à travers le réseau. L'algorithme de routage s'occupe de résoudre le problème du choix de routage à prendre au niveau de chaque routeur. Les algorithmes de routage déterminent le chemin dans le réseau qui sera pris par un paquet entre un nœud source et un nœud destinataire. Le choix d'un algorithme de routage dépend de plusieurs critères tels que la basse consommation de routage, la complexité logique de routage (plus ou moins simple), les meilleures performances de transmission, la tolérance aux fautes du réseau, la meilleure distribution de trafic, etc. Les algorithmes de routage peuvent être classés en plusieurs catégories [67] telles que le routage statique ou déterministe, dynamique ou adaptatif, distribué ou source, minimal ou non-minimal, tolérant aux fautes ou non, etc. [68].

Dans une stratégie de routage déterministe, le chemin des paquets est déterminé à partir des adresses de sa source et de sa destination [62]. A l'inverse, dans une stratégie de routage adaptatif, le chemin de routage est décidé dans les routeurs avant chaque saut des paquets, et implique des mécanismes dynamiques d'arbitrage (par exemple basé sur l'encombrement local de lien) [69]. Ainsi, un routage adaptatif offre de meilleurs avantages qu'un routage déterministe (par exemple l'équilibrage dynamique du trafic en tenant compte des zones de congestion) mais il est plus complexe dans sa mise en œuvre [32]. En outre, l'utilisation d'un routage adaptatif peut produire un interblocage statique ou dynamique. Les approches déterministes fournissent toujours le même chemin entre une paire source/destination, et elles constituent le meilleur choix pour des profils de trafic uniformes ou réguliers.

Concernant les algorithmes de routage dans les NoCs, Il existe de nombreux algorithmes de routage avec différentes propriétés. Ils sont principalement associés à la topologie du réseau.

Les algorithmes les plus souvent utilisés sont les suivants :

1. **Destination-tag** : Dans la technique destination-tag [53], l'adresse de destination est utilisée directement pour router les paquets. En effet, dans un réseau multi-étages de type k-ary n-y, comme le Buttery, l'adresse de destination est un nombre de n-digit dans la base k. Ainsi, chaque digit de l'adresse sert à sélectionner le bon port de sortie du routeur à chaque étage du réseau. L'adresse représente donc l'information de routage et se situe dans l'entête du paquet.
2. **Ordonné selon les dimensions ou dimension ordered (X-Y ou plus)** : Le routage ordonné selon les dimensions [42] [53] peut être vu comme un routage de type destination-tag

appliqué aux réseaux à maille ou cubiques. Ici, plutôt que de sélectionner le port de sortie d'un routeur d'un étage donné, chaque digit est utilisé pour choisir un nœud selon une dimension donnée. Les valeurs des digits peuvent être exprimées soit relativement aux coordonnées de l'émetteur, soit de manière absolue, ce qui correspond aux coordonnées de la destination. Dans le cas du routage relatif, les routeurs décrémentent à chaque saut la valeur correspondant à la dimension courante. Par conséquent, lorsque les paquets arrivent à leur destination, les valeurs des digits sont alors nulles.

3. **Street-sign :** Le routage street-sign [42] est assez ressemblant au routage destination-tag car il utilise un codage des informations de routage en indiquant pour chaque routeur à traverser le numéro du port de sortie à emprunter. Ces sorties peuvent aussi être identifiées par les points cardinaux du routeur Nord, Sud, Est, Ouest. Lorsque le routeur possède quatre ports d'entrée/sortie, les informations de routage concernant les routeurs précédents n'étant pas utiles et elles sont supprimées au fur et à mesure de la propagation des paquets dans le réseau. Cette technique de routage peut être appliquée à n'importe quel type de topologie.

4. **Par déviation ou de "la patate chaude" :** La méthode la plus simple de routage dynamique est la technique de routage par déviation [66] [70], dite de la patate chaude (hot potato). Dans ce modèle, lorsqu'un paquet entre dans un routeur, il est dévié vers un autre port de sortie si le port qu'il est sensé emprunté est déjà occupé. En revanche, des interblocages dynamiques peuvent apparaître. Une solution simple pour régler ce problème réside dans l'utilisation de règles de priorités.

5. **Selon le plus court chemin :** Un algorithme de routage est de plus court chemin [53] si chaque décision de routage est faite pour atteindre la cible en passant par les acheminements les plus courts. Ainsi, avant chaque saut d'un paquet, une fonction de routage va calculer, à partir des identifiants du routeur local et de la destination, quelle(s) est (ou sont) la (ou les) sortie(s) du routeur permettant au paquet d'arriver à sa destination en suivant le (ou les) chemin(s) le (ou les) plus court(s). Ces algorithmes ont l'avantage d'éviter les problèmes d'interblocages en offrant une latence réduite. Mais ils sont moins flexibles lorsqu'un routeur est congestionné.

Si on considère la classification des algorithmes de routage statique ou dynamique, on caractérise les algorithmes de routage statiques (*static routing*) comme les algorithmes définissant des chemins fixes et identiques pour transférer les paquets de données d’un nœud source vers un nœud destinataire au sein d’un réseau. Ce type de routage ne prend pas en compte les conditions du réseau

telles que sa charge actuelle (*network load*), l'occupation des canaux de transmission, la distribution du trafic, etc. Ce type de routage est simple à mettre en œuvre et nécessite de faibles ressources logiques. Parmi les exemples des algorithmes de routage statique, nous trouvons les algorithmes de routage *XY*, *turn model* (*west-first, north-last, negative-first* [71], etc.). Par contre, dans les algorithmes de routage dynamiques (*dynamic routing*), toutes les décisions d'acheminement des paquets de données sont prises en considérant les conditions en cours de la transmission dans le réseau. Dans ce type de routage, le chemin entre un nœud source et un nœud destinataire évolue au cours du temps en fonction des conditions du réseau. Généralement, le cheminement des paquets entre deux nœuds du réseau est rarement le même. Ces techniques de routage nécessitent davantage de ressources matérielles (logique plus complexe) pour leur réalisation en comparaison à la réalisation de technique de routage statique. Parmi les exemples d'algorithmes de routage dynamique, nous pouvons citer les algorithmes *fully-adaptive* [72], *odd-even* [73], *congestion look-ahead* [74], etc. Les algorithmes de routage statique et dynamique peuvent être également classés selon la façon dont les informations de routage sont stockées et l'endroit où les décisions de routage sont prises (au niveau routeur local ou routeur source). Dans l'algorithme de routage distribué, les décisions de routage s'effectuent au niveau de chaque routeur. Lorsqu'un routeur reçoit un paquet, il consulte soit ses tables de routage, soit sa fonction de routage pour calculer la destination du paquet. Ceci implique que pour chaque routeur du réseau, les fonctions ou tables de routage doivent être implantées. Dans le cas d'un algorithme de routage *source*, avant l'envoi d'un paquet à un nœud destinataire présumé, le nœud source intègre la « feuille de route » du paquet à transmettre. Cette dernière permet à tous les routeurs dans lesquels le paquet transite de l'acheminer selon son plan de route indiqué. L'inconvénient de ce type d'algorithme de routage est la nécessité d'intégrer des informations additionnelles au sein des paquets de données augmentant ainsi leurs tailles.

Une autre façon de distinguer les algorithmes de routage est leur classification selon la distance parcourue par un paquet à travers le réseau entre son nœud source et nœud destinataire. On distingue ainsi les algorithmes de routage minimaux et non-minimaux. Dans le cas d'un algorithme minimal, le chemin de routage d'un paquet entre deux nœuds est le chemin le plus court. L'algorithme de routage *XY* est un exemple de routage minimal. Par contre, les algorithmes de routage non-minimaux n'ont pas de contraintes de distance pour acheminer des paquets au sein d'un réseau. Ainsi, le nombre de chemins alternatifs entre deux nœuds est plus élevé. L'avantage de tels algorithmes est qu'ils permettent d'éviter des situations d'embouteillages tout en distribuant de façon plus homogène le trafic dans le réseau. Cependant, ce type de routage nécessite généralement des ressources additionnelles pour son implantation comparé à des algorithmes minimaux.

Enfin, les algorithmes de routage peuvent être considérés comme tolérants aux fautes ou non. Un algorithme de routage est tolérant aux fautes, si malgré la défaillance d'un des routeurs du réseau, l'acheminement des paquets s'effectue néanmoins au sein du réseau entre deux nœuds. La majorité des algorithmes de routage déjà mentionnés ne sont pas tolérants aux fautes.

Généralement, les défaillances éventuelles d'un des routeurs du réseau, changent la structure et la topologie du réseau et les rendent irrégulières, compliquant ainsi et rendant impossible le routage de paquets. Plusieurs exemples d'algorithmes de routage tolérants aux fautes peuvent être trouvés dans la littérature [75] [76] [77] [78].

Etant donné qu'aucun contrôle global n'est fait sur l'acheminement des paquets au sein du NoC, il peut se produire des situations de blocages mutuels entre paquets. Ces blocages, aussi appelés interblocages sont le plus souvent dus à l'algorithme de routage utilisé. Il en va de même pour les techniques servant à les empêcher [4]. Il existe deux types d'interblocages, les interblocages statiques (ou *deadlock*) et les interblocages dynamiques (ou *livelock*).

- **Les interblocages statiques**

Lorsqu'une dépendance existe entre plusieurs paquets, ils peuvent se bloquer mutuellement et produire un interblocage statique. En effet, cela se produit lorsqu'un routeur est en attente d'envoyer un paquet vers un autre routeur qui est lui aussi en attente et ainsi de suite jusqu'au routeur initial. De toutes les commutations de paquets, c'est la technique WormHole qui est la plus encline à ce genre d'interblocages [61]. Ces derniers peuvent être évités en choisissant un algorithme de routage approprié, comme par exemple le fait d'interdire certains virages dans une topologie maillée [79], ou encore grâce aux canaux virtuels. En effet, si un paquet se trouve être bloqué dans un canal virtuel, il pourra être doublé par un autre paquet utilisant le même canal physique mais stocké dans un autre canal virtuel.

- **Les interblocages dynamiques**

Les interblocages dynamiques se produisent lorsqu'un paquet reste indéfiniment dans le réseau sans jamais atteindre sa destination [42] [4]. La cause peut provenir d'une congestion du réseau qui ferait dérouter systématiquement le paquet ou bien encore d'une mauvaise gestion des priorités, si le lien de sortie est toujours occupé par des paquets considérés comme plus prioritaires. Dans ce cas, on parle de phénomène de famine (ou *starvation*).

Qu'ils soient statiques ou dynamiques, les interblocages peuvent aussi être résolus à l'aide de l'anticipation de paquet. Il s'agit de re-router ou de supprimer les paquets impliqués dans un

potentiel blocage. Les paquets détruits devront être retransmis par la source. Cette technique s'applique le plus souvent dans des architectures de réseaux indirects.

On parle de conflit lorsqu'au moins deux paquets ou flits au sein d'un routeur désirent emprunter simultanément le même port de sortie. Ensuite, le conflit peut se propager, conduisant au phénomène de congestion du réseau, et, par conséquent, à une réduction de ses performances. Parallèlement, un problème relatif peut arriver lorsque deux éléments communicants (un émetteur et un récepteur) ne sont pas synchrones au niveau de la cadence d'injection des données et de la cadence de leur consommation. Dans ce cas, les techniques employées pour résoudre le problème sont nommées techniques de contrôle de flux [55].

- **Le contrôle de congestion**

La conséquence principale de la congestion est la diminution de l'efficacité du réseau. D'une part, les paquets en attente dans le réseau auront une latence augmentée, et, d'autre part, la bande passante utile sera diminuée. Enfin, cela pourra avoir un effet sur la charge du réseau [55]. C'est pourquoi, dans le but de limiter ou d'éliminer la congestion, différentes techniques de contrôle de congestion ont été proposées. La plus simple résout le problème en supprimant les paquets qui sont à la cause de la congestion mais elle introduit une augmentation de la latence. Une autre solution réside dans les routages dynamiques qui feront en sorte d'éviter les zones congestionnées du réseau ou encore dans la diminution des taux d'injection des paquets dans le réseau en vue de décongestionner les zones encombrées. En revanche, lorsque des contraintes sont imposées en termes de bande passante et de latence, la congestion peut être contrôlée grâce à la réservation de ressources. Pour cela, l'émetteur doit spécifier ses besoins au réseau qui, en fonction des ressources disponibles, réservera ou pas des ressources pour le flux associé. En effet, cela empêche d'invalider la qualité de service des flux déjà présents dans le réseau. Puis, lorsque le flux est terminé, le réseau doit en être notifié pour libérer les ressources réservées et les rendre disponibles pour de nouveaux flux.

- **Le contrôle de flux**

Grâce au contrôle de flux, l'émetteur n'enverra pas plus de données que ce que l'interface réseau du récepteur ne pourra supporter. Ce contrôle peut être implanté entre routeurs adjacents et entre émetteurs/récepteurs. Là encore, plusieurs méthodes existent ; elles consistent à détruire des paquets qui ne peuvent pas être mémorisés dans les interfaces du récepteur, dévier les paquets dans le réseau tant qu'ils ne pourront pas être acceptés par l'interface réseau du récepteur, ou encore réguler l'injection des paquets en retardant leur émission dans le réseau pour contrôler et limiter le débit de

l'émetteur.

Les deux méthodes les plus courantes qui utilisent une réservation de ressources sont le contrôle de flux basé sur la poignée de main (*handshaking*) et celui basé sur le crédit d'émission. Dans le premier, le routeur en amont envoie les flits (ou paquets) au prochain routeur et attend une confirmation de sa part pour déterminer le fait qu'un paquet non accepté doive être retransmis [65]. La méthode basée sur le crédit d'émission [47] consiste à permettre aux données de rentrer dans le réseau (ou dans le routeur suivant) seulement si elles ont acquis un crédit de contrôle de flux. Ce crédit est libéré quand la donnée arrive à destination. Enfin, il n'est pas toujours nécessaire d'avoir recours à un contrôle de flux notamment lorsque le flux est garanti par l'application.

La plupart des techniques de contrôle de flux peuvent gérer la congestion de lien, mais toutes les techniques ne peuvent pas (par eux-mêmes) réaffecter toutes les ressources nécessaires pour la retransmission en cas d'erreur, soit la correction d'erreur ou d'une technique à gérer les transferts fiables doivent être mises en œuvre à une couche supérieure.

- STALL / GO : est une technique simple tête qui nécessite que deux fils de commande, un pour la signalisation de disponibilité des données, l'autre va vers l'arrière et de signalisation soit une condition de buffers remplis (STALL) ou de buffers gratuit (GO) il peut être mis en œuvre avec mise en mémoire buffer distribuée (pipelining) le long de lien. Il a une bonne performance et une récupération rapide de la congestion n'a aucune disposition pour la gestion des erreurs protocoles chargés de traiter flit interruption de niveau supérieur.

- T-Error : est une technique qui peut détecter les défauts, en faisant usage d'un second signal d'horloge retardé, à chaque étage de mémoire de buffer. L'entrée de l'horloge retardé détecte les nouveaux échantillons de données d'entrée pour les incohérences, en émettant un signal de commande VALIDE, l'étape de resynchronisation ajouté entre la fin de lien et le routeur de réception, pour gérer le décalage entre les horloges différés et originales. Le cycle d'horloge peut être utilisé pour fournir une plus grande fiabilité en configurant des liens avec l'espacement et la fréquence appropriée. Ce système ne fournit pas un mécanisme de gestion des défauts en profondeur.

- ACK/NACK : lorsque les flits sont envoyées sur une liaison, une copie locale est conservée dans une mémoire de buffer. Quand un ACK est reçu par l'expéditeur, il supprime la copie de flit de son buffer. Lorsqu'un NACK est reçu, l'expéditeur lit sa file d'attente de sortie et commence à renvoyer les flits, à partir de celle qui est corrompue. La mise en œuvre peut être soit de bout en bout soit inter-switch. L'expéditeur doit avoir une mémoire buffer de taille 2N

+ k. N est le nombre de buffers rencontrés entre la source et la destination k dépend de la latence de la logique à l'expéditeur et le récepteur.

Nous distinguons trois protocoles sur la gestion des paquets et des acquittements ACK/NACK.

- **Protocole send-and-wait**

Dans ce protocole, un seul paquet est émis à la fois par l'émetteur, puis se met en attente d'un ACK ou d'un NACK. Le récepteur émet un ACK si la réception a été bonne, sinon un NACK. A la réception par l'émetteur d'un ACK, il transmet le paquet suivant, sinon (réception d'un NACK) il retransmet le premier paquet. Pour ce protocole, il suffit d'ajouter des messages de commande (émis uniquement par le récepteur) du type ACK ou NACK. L'émetteur doit garder dans un tampon le paquet émis jusqu'à réception d'un acquittement positif.

- **Protocole Go-Back N (retour arrière de N paquets)**

Dans ce protocole, l'émetteur peut envoyer ses paquets tant qu'il en a transmettre, sans attendre l'ACK. A la réception d'un ACK, l'émetteur peut identifier le dernier paquet envoyé et bien reçu par le récepteur, et qui n'est pas forcément le dernier paquet envoyé (envoi par anticipation). En revanche, à la réception d'un NACK, le dernier paquet envoyé est non acquitté et tous ceux qui suivent sont retransmis. Ainsi, certains paquets peuvent être envoyés plusieurs fois sans être sujet à une erreur de transfert. De plus, l'émetteur doit garder en mémoire, dans des tampons, les paquets envoyés et non encore acquittés positivement. A la réception d'un ACK, l'émetteur peut supprimer les paquets associés et libérer de l'espace dans ses tampons.

- **Protocole avec répétition sélective**

Ce protocole, contrairement au deux précédents, doit utiliser des ACK/NACK avec un mécanisme de numérotation des paquets. Les messages ACK/NACK contiennent le numéro du paquet concerné par l'acquittement. Lors de la réception d'un NACK du paquet *n*, seul le paquet *n* acquitté négativement est renvoyé. Le récepteur doit faire un réordonnancement des paquets reçus avant de les transmettre lui-même dans l'ordre. De même, l'émetteur doit garder en mémoire l'ensemble des paquets envoyés et non encore acquittés positivement.

I.4. QUALITÉ DE SERVICE

A l'instar des réseaux d'ordinateurs, les applications au sein d'un NoC peuvent avoir besoin d'une certaine qualité de service (QoS) pour assurer des contraintes sévères en débits et/ou en latence. Les autres services qui sont considérés comme essentiels sont l'intégrité des données,

l'absence de perte des données et l'ordonnancement des données [80]. Afin de fournir tous ces services, des adaptations doivent être apportées au niveau du protocole de communication. Ces adaptations vont principalement concerner le routage et l'arbitrage des paquets (ou des flits). Dans les NoCs, il existe essentiellement deux classes de QoS qui sont directement liées aux trafics associés. Il s'agit des trafics de type Best Effort (BE) ou dit "Au mieux" et des trafics de type Guaranteed Services (GS) ou "Service Garanti". Dans le cas du BE**,** aucune garantie de performance ne peut être donnée. Cependant, il offre un service garantissant que les toutes les données seront transmises et ce, correctement. L'avantage principal de cette classe de QoS est d'optimiser l'utilisation moyenne des ressources du réseau. Quant au **GS,** il est implanté grâce à des mécanismes permettant de réserver des ressources de communication pour garantir un débit et/ou une latence fixée, quel que soit la charge du réseau. Ainsi, cela implique qu'une connexion doit être établie entre l'émetteur et le récepteur.

I.5. QUELQUES NoCS ACADÉMIQUES

Dans ce paragraphe, quatre types d'architectures de réseaux sur puce sont présentées.

I.5.1. SPIN

Le réseau SPIN (Scalable Programmable Integrated Network) [14] est considéré comme l'une des premières propositions concrètes de NoC à commutation de paquets. Il a été développé par le LIP6 (Laboratoire d'Informatique de Paris 6) de l'Université Pierre et Marie Curie en 2000. Ce réseau s'appuie sur une topologie en arbre quaternaire élargi (Figure I.7) offrant ainsi une latence limitée.

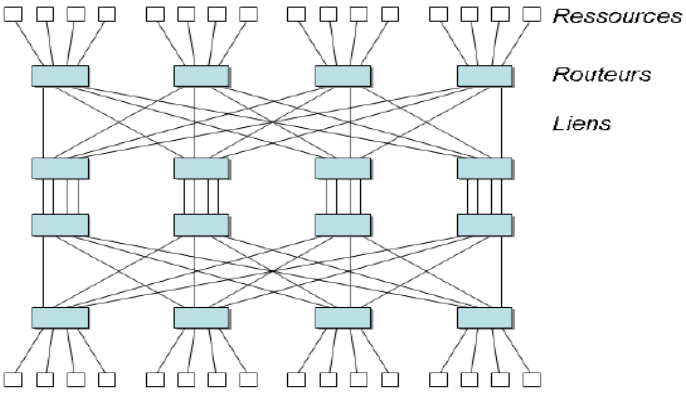
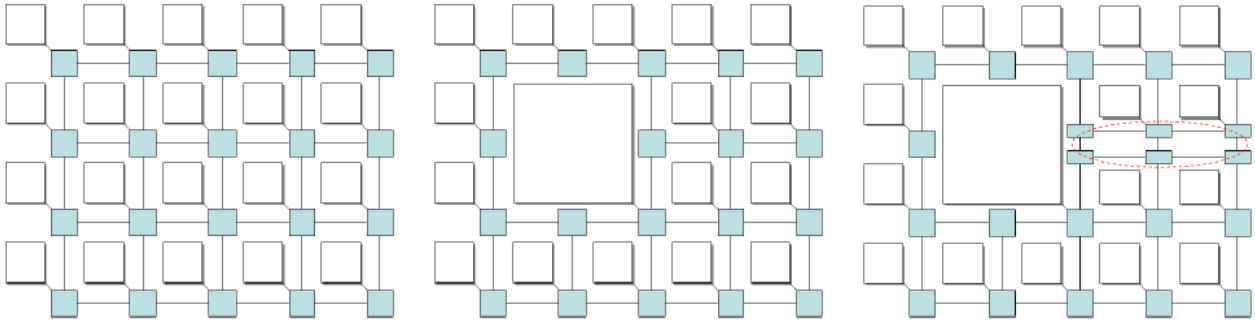


FIGURE I. 7 *ARCHITECTURE DU RESEAU SUR PUCE SPIN* [15]

Cette structure est extensible et réduit le nombre de routeurs utilisés ce qui permet d'économiser de l'énergie. Le contrôle de flux est basé sur un mécanisme de crédits sur des liens bidirectionnels.

I.5.2. ANoC

Le réseau ANoC (Asynchronous NoC) [19] est une architecture proposée par le LETI (Laboratoire d'Electronique et de Technologie de l'Information de Grenoble) dans le cadre du projet FAUST (Flexible Architecture of Unified System for Telecom). Cette structure de communication est flexible et performante en termes de latence et de débit. Ce réseau a été utilisé pour construire un système sur puce qui a la possibilité de gérer les besoins des applications de télécommunication (B3G Beyond 3G, 4G) [18]. Il est disposé en grille 2D avec commutation de paquets de type WormHole.



(a) Topologie régulière *(b) Topologie adaptée aux ressources* *(c) Topologie adaptée au d´débit*

FIGURE I. 8 *LA TOPOLOGIE POUR ANOC*

Dans cette topologie (voir Figure I.8 (a)), chaque routeur est relié à ses quatre voisins et à la ressource de traitement la plus proche. Cette structure nous permet de d´développer facilement des stratégies de routage en termes de mode de routage et d'algorithme de routage, passe à l'échelle avec la taille système, et s'implémente aisément sur silicium. De plus, avec cette structure les routeurs peuvent être génériques car tous les routeurs sont identiques. Le réseau étant utilisé dans un système hétérogène (les tailles des ressources sont variables), la topologie du réseau doit être capable de s'adapter à la taille des ressources. Dans ce cas-là, la topologie peut être remplacée par une topologie irrégulière avec certains liens supprimés (voir Figure I.8 (b)). Cependant, la suppression des liens affecte le d´débit global du réseau et peut créer localement des phénomènes de congestion. Pour y remédier, il est possible d'augmenter la bande passante locale en doublant les routeurs et en adaptant le routage (voir Figure I.8 (c)).

I.5.3. QNoC

L'architecture QNoC (Quality-of-service NoC) [6] est un réseau en topologie 2D qui peut être retaillée selon l'application, et peut donc posséder des irrégularités. Il supporte des échanges synchrones ou asynchrones et assure une commutation de paquets de types WormHole.

Le but principal d'un réseau d'interconnexion sur puce est de répondre à toutes les demandes de communication de données des modules hétérogènes au sein de la puce. Un QNoC devrait remplacer les bus non seulement communs mais aussi d'autres types de besoins de communication et définir les niveaux de service appropriés pour les supporter : signalisation, temps réel, lecture/écriture et bloc de transfert. Deux classes de services ont déjà été proposées dans ce contexte [81] : meilleur effort et débit garanti.

- **Signalisation** couvre les messages urgents et les paquets qui sont très courts en tenant compte de la plus haute priorité dans le réseau pour assurer le plus court temps de latence. Ce niveau de service est adapté pour les signaux de commande et réduit le besoin de dédicace spéciale, un fils à usage unique pour eux.

- **Service en temps réel** correspondant au niveau des garanties de bande passante et de latence pour les applications en temps réel, telles que les flux audio et vidéo de traitement. Ce service (comme tous les autres) est à commutation de paquets (et n'utilise pas des circuits virtuels). ; un certain niveau maximal de bande passante peut être attribué à chaque lie en temps réel et il ne devrait pas être dépassé. Ce résultat est obtenu soit par la conception de chaque module ou par des circuits d'exécution dans le réseau.

- **Lecture/écriture (RD/WR)**, le niveau de service fournit une sémantique de bus et est donc destiné à soutenir la mémoire courte et le registre d'accès.

- **Niveau de service de bloc-transfert** est utilisé pour les transferts de messages longs et les grands blocs de données.

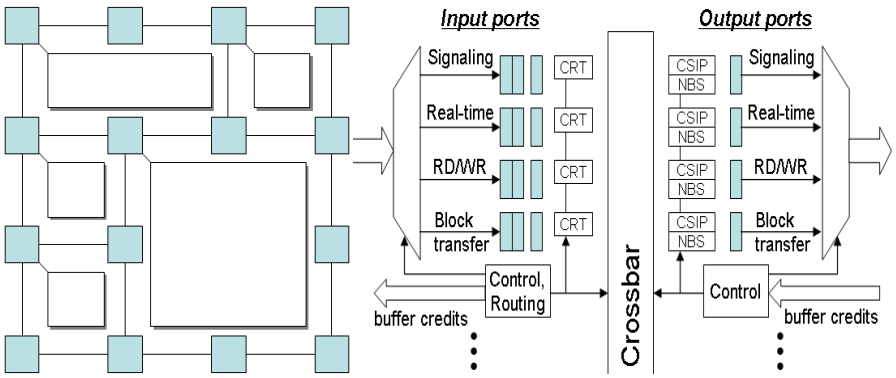


FIGURE I. 9 ***RESEAU QNOC : (A) TOPOLOGIE DU RESEAU ; (B) ARCHITECTURE DU ROUTEUR.***

Un ordre de priorité est établi entre les quatre niveaux de service ou la signalisation à la plus haute priorité et le bloc de transfert le plus bas. Ainsi, les niveaux de service sont simplement mis en

œuvre au moyen d'un mécanisme de priorité. Les niveaux de service supplémentaires peuvent être définis si on le souhaite, aussi longtemps comme une priorité le classement est respecté. Par exemple, la RD/WR niveau de service qui peut être divisée en normale et urgent RD WR/ sous-niveaux.

Un réseau *QNoC* présente un gain élevé en termes de performances (débit de transmission et bande passante réseau). Néanmoins, ce *NoC* reconfigurable présente un surcoût important en termes de ressources logiques d'implantation. Dans ce contexte de conception de réseau sur puce reconfigurable adapté à la mise en œuvre de système *MPSoC*, un compromis doit être recherché en termes de vitesse de performance et de ressources d'implantation matérielle optimisée.

I.5.4. Hermès

Le réseau Hermès, développé par l'équipe de F. Moraes [20] chaque routeur possède un réseau sur puce à topologie maillée 2D avec commutation de paquets (WormHole). Un buffer sur les ports d'entrée, un arbitrage de type « Round-Robin » et un algorithme de routage gérant les conflits permettent d'éviter les situations d'interblocage dans ce réseau.

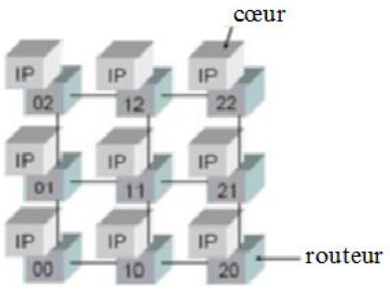


FIGURE I. 10 *TOPOLOGIE MAILLEE 2D DU NOC HERMES*

I.6. RÉSEAUX SUR PUCE RECONFIGURABLES À BASE DE FPGA

Une classification des *NoC* en fonction des changements possibles de leurs paramètres au cours de fonctionnement peut être établie soit comme des *NoC statiques* ou des *NoC dynamiques*.

Les réseaux sur puce dont les topologies, le placement d'éléments de calcul *PE*, les fonctions de routage et d'aiguillage et les routeurs sont fixes et inchangeables dans le temps sont considérés comme *statiques*. Dans cette classe de *NoC*, nous trouvons les architectures *NoC* mentionnées dans la section précédente [82] [15] [83] [84] [85] [86].

Dans le cas où, les réseaux sur puce dont les paramètres mentionnés ci-dessus peuvent changer au cours du fonctionnement sont considérés comme *réseaux dynamiques*. Les *NoC* dynamiques sont souvent confondus et associés avec les *NoC reconfigurables*. Par la suite, nous donnons un résumé

des travaux existants dans la littérature sur les *NoC reconfigurables,* et nous proposons une définition précise de la notion de « *reconfigurable* » associée aux *NoC*.

Une méthodologie de développement des processus dynamiques reconfigurables dans le cadre des *NoC* a été proposée dans la littérature [87]. Ces travaux définissent la notion de *reconfiguration* au sein d'une structure *NoC* comme le changement de sa fonction de routage au cours du fonctionnement tout en conservant le réseau fonctionnel et opérationnel pendant le processus de reconfiguration. De même, au cours du fonctionnement d'un *NoC*, ses spécificités de conception initiales telles que la topologie du réseau ou l'algorithme de routage, peuvent être modifiées au cours du fonctionnement du *NoC* de façon, soit involontairement suite à des défaillances des parties du réseau, soit volontairement à cause de suppressions ou d'ajouts de nouveaux modules au sein du réseau. En pratique, ces situations de changement mènent souvent vers des blocages des messages ou paquets de données dans le réseau. En général, pour y remédier, l'algorithme de routage utilisé doit être modifiable dans le but de rétablir les connexions entre tous les nœuds ou routeurs du réseau associés à des modules de calcul PE.

Une approche théorique dans ce sens, permettant à un *NoC*, de continuer de fonctionner sans interruptions tout en assurant les contraintes définies initialement lors de sa conception et une *qualité de service* (*QoS*[1]), a été également proposée dans [87]. L'approche proposée considère que la topologie du réseau reste inchangée au cours de ce processus de reconfiguration. Une autre approche, basée également sur l'hypothèse d'une conservation de la topologie du réseau au cours d'un processus de reconfiguration est détaillée dans [88]. Dans cette approche, la reconfigurabilité du *NoC* est assurée par des modules appelés *maître de configuration* (*configuration master*), correspondant à des processeurs ou CPU typiques. Ces modules peuvent accéder, au cours du fonctionnement et à travers l'infrastructure de communication existante du réseau, aux interfaces de réseau *NI* (*Network Interface*) dans le but de modifier ou de mettre en œuvre la « (re)configuration » du réseau. Une autre approche reposant sur un outil de modélisation, de simulation et d'implantation de *NoC* reconfigurable a été également proposée dans [89]. Plus précisément, à partir d'une description haut niveau du réseau et d'un environnement de simulation variable permettant d'appliquer des stimulis correspondant à diverses applications, une configuration optimale du réseau peut être déterminée et transcrite en langage *VHDL* pour implantation. Cette approche permet la conception de *NoC* (re)configurables et statiques, car les critères et paramètres

[1] ***QoS*** - Aptitude d'un service à répondre adéquatement à des exigences, exprimées ou implicites, qui visent à satisfaire ses usagers [92].

pour une application donnée ne sont pas imposés par l'environnement de simulation durant l'exécution, mais dans la phase de modélisation du réseau *NoC*.

Une architecture *NoC* reconfigurable dynamiquement et destinée pour les *MPSoC*[2] reconfigurables est présentée dans [90]. Cette architecture permet la reconfiguration dynamique au cours du temps d'exécution du réseau, de certains paramètres du réseau tels que l'algorithme de routage, la technique d'aiguillage, la taille des messages à transmettre. Dans cette approche, toutes les ressources du réseau utilisées doivent être allouées statiquement dans la phase de conception. Les *PE* sont également reconfigurables dynamiquement. L'idée maîtresse de cette approche est que différents *PE* connectés à un *NoC* ont des différents besoins en communication et que donc à chaque type de *PE*, il faut adapter et (re)configurer le réseau *NoC* associé. Ce type de *NoC* assure la reconfigurabilité de ses nœuds grâce à un noyau (*kernel*) sous forme d'un modèle de référence *OSI* modifié et nommé *SNS* (*Smart Network Stack*). En fonction des données à transférer, le *SNS* décide quel type d'algorithme de routage, de technique d'aiguillage et des tailles de données à choisir. Ces informations sont placées dans l'entête des paquets considérés, qui seront analysés par les routeurs du réseau. Ainsi, par exemple, dans le cas où des besoins importants en communication sont sollicités, une taille de paquet plus grande sera choisie, tandis qu'une technique d'aiguillage initiale par défaut de type *packet switchning*, sera remplacée par la technique de type *circuit switching*.

Une approche similaire, destinée également pour la conception de *MPSoC* est présentée dans [91]. La principale différence est le nombre de paramètres du réseau (*NoC* de type Æthereal [92]) pouvant être reconfigurés au cours de son temps d'exécution et la possibilité de reconfiguration des *PE* (*Silicon Hives processing cores* [93]) associés au réseau.

Une approche, nommé *ReNoC* pour *Reconfigurable NoC*, permettant la reconfigurabilité de la topologie d'un réseau *NoC* par combinaison des techniques d'aiguillage *circuit* et *packet switching* est proposée dans [94]. Dans ce type de réseau ReNoC, deux types de couche de topologie sont distingués : une couche *physique* correspondant à la couche physique réelle du réseau, et une couche *logique* correspondant au réseau perçu par l'application et qu'elle utilise pour ses besoins en communication. La couche *logique* est configurée dans la phase d'initialisation en fonction des demandes en communication. Pour des besoins en communication élevés et pour une consommation plus faible, la technique d'aiguillage *circuit switching* est privilégiée par le *ReNoC*

[2] ***Multiprocessor System-on-Chip*** - Système sur puce multiprocesseur. Circuit intégré sur une seule puce, qui comporte plus d'un processeur dans sa structure. Certains auteurs emploient parfois les termes *système multicœurs* et *architecture multicœurs* pour décrire ce type de composant électronique [92].

par rapport à la technique d'aiguillage *packet switching.*

Dans les systèmes reconfigurables dynamiquement à base de *FPGA*, les contraintes telles que le placement de *PE* et les performances nécessaires pour une application donnée sont variables au cours du fonctionnement de ces systèmes. C'est pourquoi, les systèmes reconfigurables qui reposent sur une structure de communication *NoC* nécessitent des réseaux dont la topologie est modifiable. C'est dans ce cadre que l'architecture *CoNoChi*[3] a été proposée [95]. Cette architecture *NoC* permet un changement et une adaptation dynamique de la structure du réseau en fonction du nombre de *PE* associés au réseau à un instant donné. La solution architecturale proposée consiste à adapter le nombre de routeurs au nombre de *PE* placés statiquement ou dynamiquement sur un circuit *FPGA* en utilisant le procédé de reconfiguration dynamique partielle des circuits *FPGA* utilisés (technologie *Virtex* de *Xilinx*). En adaptant le réseau de communication pour un nombre de *PE* donné, le nombre de routeurs nécessaires reste toujours minimal. Ceci réduit considérablement la latence moyenne du réseau et les ressources nécessaires pour son implantation. L'approche *CoNoChi* a été validée expérimentalement sur une plateforme *Xilinx Virtex 2 Pro* [95].

Une approche similaire aux structures *CoNoChi* et *ReNoC* est présentée dans [96]. L'approche repose sur un circuit *FPGA* divisé en deux parties. Une première partie statique (non-reconfigurable), contenant des *PE* (cœurs logiciels ou matériels de processeurs *soft-core* - *MicroBlaze* ou / et *hard-core* - *PowerPC*), des mémoires et des interfaces réseaux permettant de connecter les *PE* entre eux via un réseau *NoC*. Une seconde partie reconfigurable, contenant un réseau *NoC* et des éventuelles connexions de type *point-à-point*. Cette deuxième partie peut également contenir des *PE*. Le routage au sein du réseau est géré par des tables de routage, stockées dans les *BRAM* (mémoire sur puce du *FPGA*), dont le contenu peut être changé dynamiquement au cours du fonctionnement. Cette reconfiguration dynamique est assurée par un contrôleur de reconfiguration connecté à une mémoire externe contenant les fichiers de configuration (*bitstream*[4]). Ce contrôleur peut, en cours d'exécution, reconfigurer les parties reconfigurables du *FPGA* en chargeant les bits de configuration stockés dans la mémoire externe via l'interface *ICAP*[5] vers les zones reconfigurables. Ainsi, grâce à une reconfiguration partielle du circuit, le réseau peut être adapté aux besoins d'une application donnée (voir Figure I.8 [95]). Les résultats d'implantation et expérimentaux de cette approche montrent que des gains considérables en termes de

[3] ***CoNoChi*** - Configurable Network-on-Chip. Le réseau sur puce configurable.

[4] ***Bitstream*** - Séquence de données constituée de données strictement numériques [92].

[5] ***ICAP*** - *Internal Configuration Access Port.*

consommation et de temps de latence des transmissions de paquets de données peuvent être obtenus. Cependant, cette approche nécessite des durées des phases de reconfiguration des zones reconfigurables relativement élevées de l'ordre de dizaines, voir centaines de *milliseconde* (*ms)* pour les zones reconfigurables de taille de plusieurs *frames* [96].

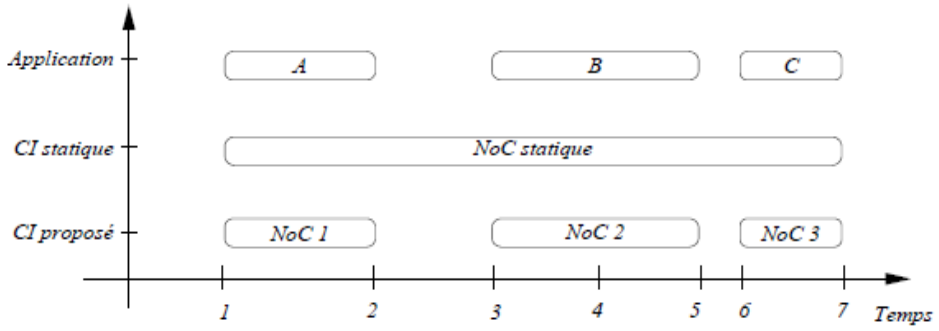


FIGURE I. 11 ***ILLUSTRATION DE CHANGEMENT DE TOPOLOGIE DE NOC DANS L'APPROCHE*** **[96]** ***POUR DIFFERENTES APPLICATIONS***

L'approche *DyNoC* (*Dynamic NoC*) permet un placement dynamique de modules (*PE*) au cours du temps d'exécution d'un réseau *NoC* [97] [98]. Dans cette approche, un réseau *NoC* classique est initialement créé, où à chaque routeur du réseau un *PE* est associé occupant une surface (*tile*). Au cours du fonctionnement, de nouveaux *PE* peuvent être placés ou intégrés au sein du réseau. Ces nouveaux *PE* peuvent être de taille supérieure que la taille initiale destinée pour un *PE* (*tile*). Dans ce cas, le nouveau *PE* dynamiquement placé dans le réseau couvre une surface du réseau correspondant à certains *PE* et routeurs placés initialement à cet endroit du réseau. Les routeurs qui sont couverts par un *PE* placé dynamiquement seront désactivés et ne pourront plus être utilisés pour le routage de messages au sein du réseau. Par contre, ces routeurs peuvent être utilisés comme de la logique supplémentaire pour le nouveau *PE*. Les routeurs désactivés seront réactivés de nouveau, une fois le *PE* placé dynamiquement a accompli sa tâche et le système revient à la configuration initiale. Ce processus d'activation et de désactivation de routeurs rend le réseau dynamique. En plaçant un module dynamiquement dans le réseau, ce dernier devient d'une structure hétérogène et nécessite un algorithme de routage adapté pour acheminement des messages vers toutes les destinations du réseau sans blocages et / ou difficultés. Le réseau *DyNoC* repose sur un algorithme de routage *XY* modifié et adapté aux placements dynamiques de *PE* au sein du réseau (Algorithme *S-XY - Surrounding XY* – le *XY* « contournant »). Parmi les principaux inconvénients de *DyNoC*, nous citons les ressources importantes nécessaires pour son implantation et un taux *PE* / NE[6] assez élevé.

[6] PE / NE - Le taux *Processing Element* / *Network Element*

La plupart des approches *NoC* reconfigurables présentées dans la littérature définissent la reconfigurabilité des *NoC* comme un moyen de modifier leurs configurations au cours de leur fonctionnement. Ces configurations de réseau correspondent des changements de paramètres de réseau tels que l'algorithme de routage, les techniques d'aiguillage, la *Qualité de Service - QoS*, adaptés à une application donnée et pouvant être changés au cours du fonctionnement du réseau. Cependant, dans le cadre de la conception de système reconfigurable à base de technologie *FPGA*, nous considérons une architecture reconfigurable comme une architecture permettant une modification de sa structure architecturale ou de traitement de données à tout moment. Dans ce contexte, nous considérons uniquement les travaux de la littérature relatifs aux approches de conception de réseau de type ou similaires aux structures *ReNoC* et *CoNo-Chi* [95] [94] [96] comme des *NoC reconfigurables*. En effet, hormis ces approches, les réseaux présentés dans la littérature possèdent des structures topologiques fixes au cours de leur fonctionnement. Nous les classifions donc par la suite comme des *NoC paramétrables* aux côtés de *NoC reconfigurables*. Nous notons également, que ces *NoC reconfigurables* distinguent le réseau utilisé pour la communication des *PE*, possédant une structure homogène et composée uniquement de routeurs, au reste du système (voir la Figure I.9). Dans ces approches, pour une application donnée, soit les *PE* sont reconfigurés en d'autres *PE*, soit on reconfigure le réseau *NoC* pour l'adapter à l'ensemble de *PE* utilisés pour l'application considérée.

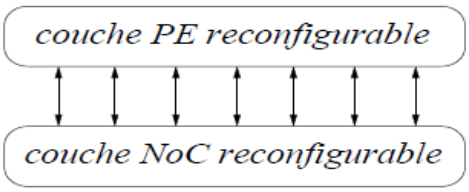


FIGURE I. 12 ***ILLUSTRATION DE DEUX COUCHES DISTINCTES (NOC ET PE) DANS LES APPROCHES NOC RECONFIGURABLES***

Dans le cadre des systèmes reconfigurables à base de *FPGA*, où des *PE* peuvent apparaître, disparaître ou être délocalisés au sein du réseau au cours du temps, il apparaît plus « naturel », d'un point de vue reconfiguration, que le réseau de communication utilisé soit « omniprésent » et que les *PE* apparaissent et disparaissent dans le réseau de manière dynamique tout en adaptant sa topologie et sa façon d'acheminer les messages dans le réseau. L'approche *DyNoC* en est un exemple (voir la Figure I.13).

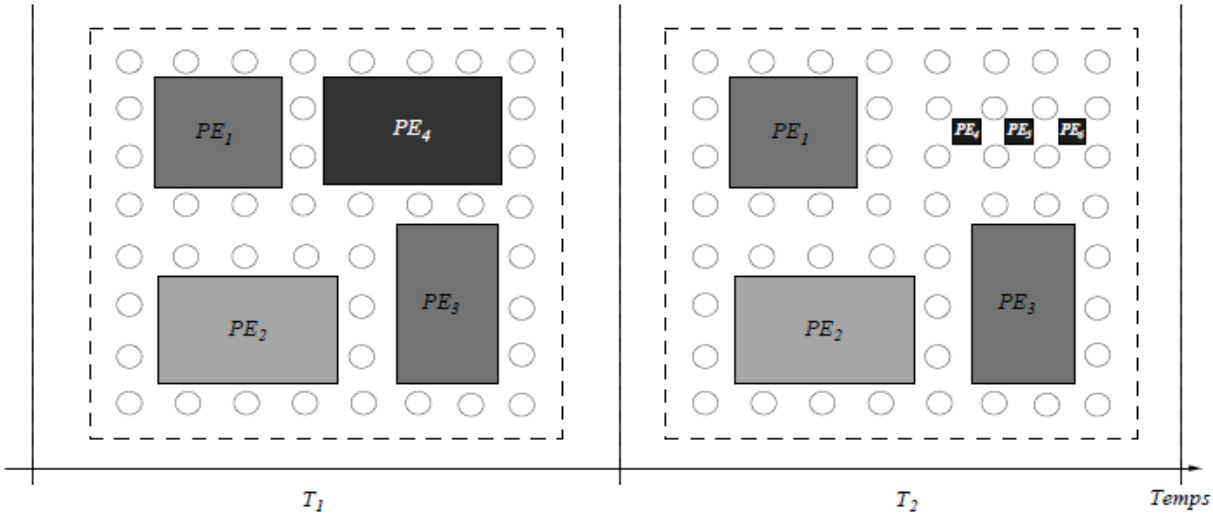


FIGURE I. 13 ***L'APPROCHE DYNOC*** [104] ***: ILLUSTRATION DE L'EVOLUTION D'UN RESEAU DYNAMIQUE RECONFIGURABLE DANS LE TEMPS***

I.7. LIMITES DES RÉSEAUX SUR PUCE

Malgré les avantages apportés par les réseaux sur puce, ces structures présentent quelques limites que nous allons évoquer ci-dessous.

I.7.1. Cohérence des caches

Lorsqu'une donnée est partagée, plusieurs copies sont stockées dans les mémoires caches des IPs. Chaque ressource (IP) effectue des opérations sur sa copie ce qui engendre un problème de cohérence des caches.

Parmi les approches permettant de résoudre ce problème, citons :

- ✓ La solution par espionnage [21] résout le problème, mais augmente la consommation d'énergie lorsque le nombre d'IPs connectées est grand.
- ✓ Une seconde alternative consiste à ne pas placer les données partagées dans les caches. Ceci pallie au problème, mais l'énergie consommée reste importante lorsque la quantité d'informations partagées est grande.
- ✓ Enfin, la solution la plus utilisée repose sur l'utilisation d'un répertoire centralisé qui permet de mémoriser l'état actuel des caches [22]. Elle réduit la consommation de l'énergie et le temps d'exécution du système [23].

I.7.2. Fiabilité

L'intégration des réseaux sur silicium entraine une vulnérabilité aux défauts physiques et aux

effets transitoires (tel que le Crosstalk) [24]. Les NoCs doivent garantir une bonne transmission et une intégrité des données transférées. Des techniques de détection et de correction des erreurs sont mises en place pour résoudre ce problème.

I.7.3. Ordre des communications

Les réseaux à communication de paquets peuvent ne pas conserver l'ordre temporel de l'envoi des paquets. Il faut mettre en place des mécanismes qui exécutent un ré-ordonnancement lors de la réception. Cette opération est d'autant plus importante lorsqu'il s'agit de requêtes de lecture/écriture qui doivent être exécutées selon leur ordre d'envoi [56].

I.8. OBJECTIFS DE LA CONCEPTION DES NoC

Dans la phase de conception du NoC, il y a des objectifs à respecter comme la minimisation de consommation d'énergie, la réduction de la surface de la puce, maximisation du temps de performance. Dans tous ces objectifs, le plus important est celui de la réduction de la consommation d'´énergie dans le cas des systèmes embarqués mobiles, car leur fonctionnement dépend de la durée de vie de la batterie. En deuxièmes, la réduction de la surface de la puce, car 80% de la surface (pour la plupart des NOC) est occupée par les routeurs et les mémoires. Pour résoudre ce problème, on doit optimiser les algorithmes de routage au niveau des routeurs (switches) et augmenter l'utilisation des mémoires.

En réalité ces objectifs sont contradictoires, car en réduisant l'énergie consommée, on augmente le temps d'exécution des composants (CPU) qui fonctionnent en mode économie. Par conséquent, est contraint de trouver un compromis entre ces différents objectifs contradictoires.

I.9. SÛRETÉ DE FONCTIONNEMENT DES RÉSEAUX SUR PUCE

I.9.1. Définition

La sûreté de fonctionnement (SdF) est l'ensemble des dispositifs mis en place pour déterminer la fiabilité et la sécurité d'un système. En sens large, la sûreté de fonctionnement est définie comme étant la *science des défaillances* incluant leur connaissance, leur évaluation, leur prévision, leur mesure et leur maîtrise [99] [100]. Un système possède 5 critères pour évaluer son niveau de SdF :

- **Disponibilité :** évalue la qualité du système à fonctionner lorsqu'un service est demandé.
- **Fiabilité :** continuité de fonctionnement du système.
- **Sûreté :** impacte d'une défaillance sur l'environnement et les utilisateurs.
- **Intégrité :** absence d'altération inappropriée du système.

- **Maintenabilité :** définit si un système est facilement modifiable et s'il est facile à remettre en œuvre après une défaillance.

Si l'on souhaite également quantifier la sécurité du système, un sixième critère correspondant à la confidentialité peut être rajouté aux attributs d'un système. Cet attribue correspond à l'absence de divulgation d'informations non-autorisées.

I.9.2. SdF appliquée aux réseaux sur puce

Concernant le système microélectronique embarqué du réseau géré, le SoC devient plus sensible aux phénomènes produisant des défauts, et le comportement de son support de communication centrale peut être affecté par différents facteurs, pouvant conduire à un échec du système sur puce entier.

Il existe deux familles de techniques qui sont généralement employées dans les NoCs : les technique en ligne et hors ligne).

- **La technique en ligne de type *BIST :* (Built-In-Self-Test)**

Les techniques de BIST comprennent des circuits de test à l'intérieur du NoC et exiger plus de ressources matérielles pour leur mise en œuvre mais sur profit d'un temps de test moindre

- **La technique hors ligne de type *DfT : (Design-for-Testabilty)***

Les techniques DfT nécessitent un module externe de test qui injectent des vecteurs au sein du réseau. Le principal avantage de cette technique est qu'elle nécessite pas de ressources matérielles supplémentaires au réseau, mais nécessite un temps plus grand pour les tests comparé aux techniques en ligne. Pour ce qui est de la capacité de localisation des erreurs, les techniques hors ligne comportent souvent de meilleurs résultats, proches de 99% pour la détection d'erreur [101] [102].

CONCLUSION

Il existe plusieurs réseaux sur puce académiques. Cependant, peu de solutions ont été commercialisées malgré qu'un réseau sur puce soit un nouveau paradigme d'interconnexion inspiré des réseaux informatiques et composé de routeurs communicants entre eux via des liens physiques. Ceci est dû à la complexité du processus de conception et surtout au coût qu'il présente.

Nous avons présenté les éléments constitutifs d'une architecture NoC, puis nous avons détaillé les protocoles de communication nécessaires à la transmission de l'information au sein des NoCs. Nous avons vu aussi que différents mécanismes de gestion de flux de communications

peuvent être mis en place. Ainsi, grâce à leur flexibilité et extensibilité, les réseaux sur puce permettent de découpler les communications et les traitements au sein d'un SoC. Cependant, il faut bien noter que la conception d'un système doit être faite en fonction du trafic généré par l'application ciblée. En effet, afin d'obtenir des performances optimales (en termes de surface, débit, latence, etc.), l'analyse des communications permet de définir l'architecture d'un NoC la mieux adaptée à une application.

Ce chapitre a permis d'étudier un espace de conception qui se structure autour de deux axes fondamentaux. Le premier vise à définir une architecture, en se basant sur les contraintes des communications de l'application. La conception de cette architecture peut être divisée en trois problèmes distincts :

- Le choix de la topologie est très important car la capacité du réseau à diffuser efficacement l'information dépend essentiellement de la géométrie de sa structure.
- Le dimensionnement de la largeur des canaux du réseau n'a pas été adressé à ce jour, bien que la détermination de la meilleure largeur des canaux pour une application donnée, est nécessaire pour optimiser la surface, les flux au sein du réseau, ainsi que le choix du calibrage et l'espacement des fils qui déterminent la fréquence de fonctionnement du canal.
- La taille de la mémoire tampon de chaque entrée d'un routeur ou d'une interface réseau a des répercussions lourdes sur la surface et la consommation d'énergie. Ainsi, l'utilisation globale des ressources en mémoire tampon doit être réduite au maximum.

Il en résulte que la majorité des réseaux classiques proposés dans la littérature ne sont pas prouvés et adaptés pour la conception de système auto-organisé reconfigurable à base de technologie FPGA. Plus précisément, afin d'exploiter efficacement ces technologies reconfigurables, les approches NoC classiques doivent être adaptées dans le but d'intégrer dans ces réseaux les aspects de versatilité. C'est pourquoi, dans le chapitre suivant, nous allons préciser les techniques de vérification pour voir la performance du NoC avant toute intégration et permettant de répondre aux attentes d'une conception d'un système reconfigurable à une structure NoC adaptative.

CHAPITRE II :

TECHNIQUES DE VERIFICATION

INTRODUCTION

Ce chapitre présente dans sa globalité l'état de l'art des principes qui définissent les méthodes de vérification du point de vue conception matérielle avec les outils de développement et de simulation associés, plus particulièrement, les outils dédiés à l'implantation matérielle d'applications pour systèmes embarqués, à la vérification semi-formelle et à la vérification par des méthodes formelles.

II.1. OUTILS DE CONCEPTION ET SIMULATION

Les FPGA sont des composants numériques reconfigurables dont on définit le fonctionnement interne par le biais d'un langage de description matérielle de type HDL (Hardware Description Language) comme par exemple VHDL (Very High Speed Intergrated Circuit HDL). On définit ainsi une architecture de traitement dans laquelle on peut utiliser toutes les ressources internes du composant, ainsi que les entrées/sorties, les communications avec d'autres composants, etc. L'évolution rapide des techniques de fabrication et des technologies ont permis de voir la naissance de FPGA de plus en plus performants et de plus en plus complexes. On trouve ainsi dans les FPGAs de dernière génération des processeurs embarqués, des liens de communications ultrarapides, et bien d'autres fonctionnalités. Ils sont aussi de plus en plus denses (jusqu'à 1 milliard de portes logiques équivalentes) et de plus en plus rapides (jusqu'à 500 MHz). Ce qui permet la réalisation de systèmes très complexes et très performants. De plus, le potentiel de reconfigurabilité statique et dynamique permet de tenir compte de l'évolution des standards et ainsi de modifier l'architecture interne alors que la carte et le reste des composants restent figés. Les FPGAs sont souvent utilisés soit pour le prototypage des systèmes complexes comme les SoC, soit comme des systèmes dédiés au traitement du signal et d'image.

II.1.1. Flot de conception classique sur FPGA

Le flot de conception d'une application sur FPGA est habituellement réalisé en plusieurs étapes. Tout d'abord, les spécifications algorithmiques permettent de définir l'architecture par une démarche d'Adéquation Algorithme Architecture. On décrit ensuite cette architecture avec un langage HDL. On peut ensuite simuler le système, et le modifier si nécessaire. Viennent ensuite les phases de synthèse et de placement routage, qui consistent à déterminer quels éléments vont effectivement être utilisés dans le FPGA, comment ils vont être connectés entre eux, et quel endroit du composant ils seront placés, etc. Chaque phase nécessite une vérification du bon fonctionnement

global et des normes de temps d'exécution, et si nécessaire des modifications.

II.1.2. Utilité d'un langage de description de matériel

L'utilisation d'un langage de description matérielle permet plusieurs niveaux de conception (plus au moins abstrait correspondant à des descriptions comportementales ou structurelles. Ainsi, un langage HDL permet :

- Un raisonnement plus "formel" sur les montages ;
- Des simulations avant leur réalisation ;
- Une entrée (standardisée) vers de nombreux outils de synthèse automatique ;
- Une disponibilité de bibliothèques de descriptions de sous-ensembles ;
- Une portabilité où une description VHDL est souvent exigée dans la livraison des dispositifs électroniques numériques (armée, aérospatial, télécom, etc.).

II.1.3. Flot de conception

La figure qui suit montre les étapes de conception moderne.

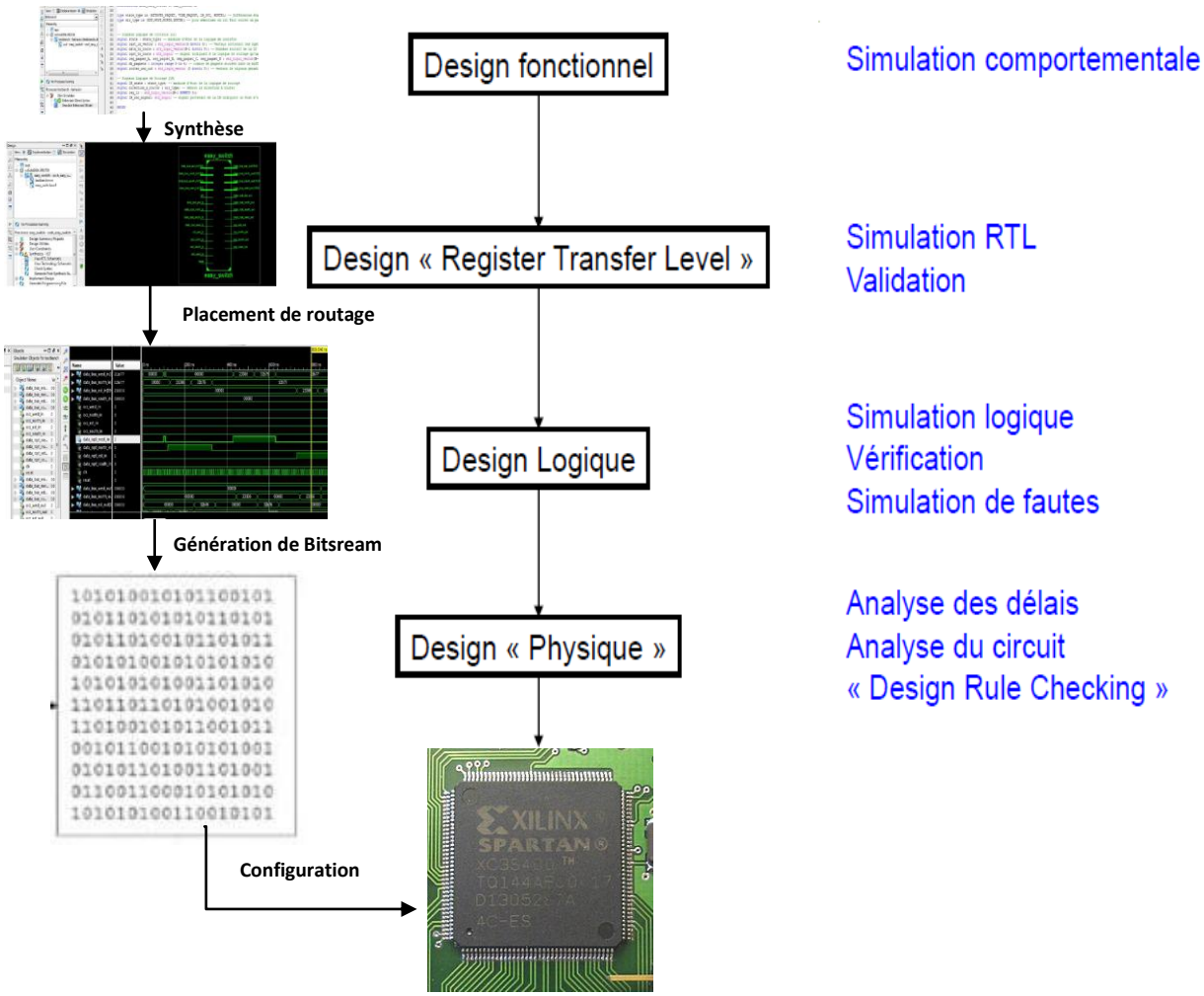


FIGURE II. 1 *LES ETAPES DE CONCEPTION MODERNE*

Les étapes de conception moderne après une description HDL d'un système commence par des phases de simulations comportementales jusqu'à une analyse temporelle des réponses du système (où des délais) et une analyse du circuit ou des résultats de synthèse retraçait ses formes d'une configuration de circuit.

II.1.4. Concepts de base

Le flot de conception d'un système sur puce regroupe plusieurs niveaux d'abstraction. Dans chaque niveau, le concepteur s'intéresse à la résolution d'un problème. Les outils de CAO[7] sont utilisés intensivement et assurent la transition entre les différents niveaux d'abstraction. Nous pouvons traiter un système complexe de deux manières qui sont :

- L'approche dite « descendante » (ou « **Top-down** » en anglais).
- L'approche « ascendante » (ou « **Bottom-up** » en anglais).

Remarque : il y a une grande similitude de conception pour les FPGA, les CPLD et les ASIC.

Le développement d'une application sur FPGA par des outils CAO suit l'enchainement des étapes suivantes :

1) Spécification de la conception

Dans cette étape, il s'agit de définir :

- ❖ Le nombre de broches d'entrée-sortie et leur localisation dans la puce FPGA.
- ❖ La spécification de la fréquence d'horloge du système.
- ❖ La spécification de la mémoire requise pour l'application.

2) Développement de la conception

Dans cette étape, il s'agit :

- ❖ De définir une spécification de méthodologie de conception (outil de développement utilisé).
- ❖ De la saisie du circuit Codage RTL (VHDL, Verilog, etc.) selon une approche :

[7] **CAO** **C**onception **A**ssistée par **O**rdinateur : technique de conception aujourd'hui très éprouvées et largement employées afin de concevoir des circuits électroniques

- Graphique (Machine à états).
- Syntaxique de type HDL (**H**ardware **D**escription **L**anguage)

❖ De vérification par simulations (Pré et Post synthèse).

3) Synthèse logique

La synthèse est le processus qui convertit la représentation de la conception à partir d'un code HDL ou d'une description graphique fourni pour produire une représentation au niveau porte logique. Elle s'occupe de déterminer quelles sont les structures susceptibles de répondre au cahier des charges étudié, et de produire un code de programmation unique sous forme d'un fichier de configuration.

4) Placement et routage

À partir des fichiers de synthèse, l'outil de conception procède au placement et au routage. Un algorithme de routage permet de configurer des liaisons physiques du FPGA afin de faire l'aiguillage des données qu'il reçoit vers leurs destinations par action sur des cellules de routage et permettant de définir les chemins qui relient l'ensemble des cellules logiques du FPGA (**CLB**[8]**)** configurées selon des fonctions logiques désirées. Ces algorithmes de routage sont différents d'une conception à une autre. Plusieurs étapes sont donc nécessaires pour obtenir un fichier de configuration :

- **Etape de partitionnement :** les équations logiques sont partitionnées en un autre ensemble équivalent d'équations. Chaque équation de ce nouvel ensemble peut être implantée dans un seul bloc logique du composant cible ***FPGA.***
- **Etape de placement :** Des blocs logiques sont sélectionnés dans la matrice FPGA et sont affectés au calcul des booléennes issues de l'étape précédente.
- **Etape de routage :** les ressources d'interconnexion sont affectées pour permettre les interconnexions des différents blocs logiques matérialisant les fonctions logiques à mettre en œuvre.
- **Etape de génération des données numériques de configuration :** les informations abstraites de routage, de placement et les équations implantées dans les blocs sont

[8] **CLB : Configurable Logic Block**

transformées en un ensemble de valeurs numériques correspondant aux données de configuration qui seront chargées dans le composant ***FPGA***.

5) Intégration et implémentation

L'implémentation est la réalisation proprement dite qui consiste à mettre en œuvre l'algorithme sur l'architecture du circuit configurable cible. Cela revient à compiler, à charger, puis à lancer l'exécution sur un ordinateur ou un calculateur. C'est une étape de programmation physique et de tests électriques qui clôture la réalisation du circuit.

II.1.5. Quelques langages de description par programmation FPGA

VHDL :

Les langages de description matérielle comme VHDL ou Verilog, utilisés habituellement pour le développement de FPGA, sont de nature concurrente. La programmation en VHDL implique une bonne connaissance non seulement de l'algorithme mais aussi de l'architecture du FPGA et du synthétiseur utilisé. Pour mieux exploiter le parallélisme intrinsèque de l'algorithme, il faut exécuter les tâches de traitement en parallèle (non séquentiel) pour satisfaire aux contraintes temporelles. Pour la plupart des spécialistes du contrôle ou du traitement du signal qui sont souvent des chercheurs et des ingénieurs de développement en logiciel, ces langages matériels ne sont pas familiers et sont d'une utilisation parfois délicate. C'est probablement une des principales raisons qui freine la démocratisation de la technologie FPGA. Pour tenter d'apporter une réponse appropriée à ce problème, de nouveaux langages de conception de circuits, basés sur le langage C, ont été proposés. On peut citer entre autres le langage System-C [27] [28] de Synopsys ou le Handel-C [29] de Celoxica.

Handel-C :

Commercialisé par Celoxica, Handel-C [29] ajoute quelques extensions, comme le parallélisme ou la gestion d'une horloge au langage ANSI/ISO-C. Selon ses concepteurs, il permet à des ingénieurs en logiciel de réaliser des circuits numériques très rapidement. Le style de programmation est en effet identique à celui du C standard et il est inutile d'étudier la philosophie des langages exclusivement dédiés au matériel. En particulier, Handel-C permet d'exprimer très facilement le parallélisme matériel et les synchronisations. En voici les principales caractéristiques :

- Une affectation de variable prend exactement un cycle d'horloge,
- Les tailles des variables (y compris celles liées aux entrées/sorties) peuvent être définies au bit près, ceci afin d'occuper le moins de ressources possibles,

- Les constructions parallèles sont autorisées ; les branches les plus rapides attendent la fin de l'opération de la branche la plus lente avant de continuer,
- Il existe des canaux de communication entre processus,
- Des primitives *ram* et *rom* permettent de gérer directement les blocs de mémoire enfouis dans les FPGA ou bien externes.

System-C :

L'un des objectifs de System-C est de fournir un outil permettant la description et la simulation de systèmes incluant des parties logicielles et matérielles (conception co-design). Une fois les tests achevés, un compilateur System-C génère une *netlist* exploitable par les outils de placement-routage. Soutenu par des compagnies telles Xilinx, Synopsys, Innoveda ou Cadence, System-C concurrence VHDL dans le cas de conception à partitionnement logiciel-matériel.

JHDL :

Les chercheurs provenant des milieux académiques ou industriels proposent aussi d'autres alternatives à VHDL et Verilog pour faciliter la conception de circuits numériques comme par exemple JHDL [30]. Lors du développement d'un système comportant des composants logiciels et des modules fonctionnant sur un FPGA, la méthode de travail usuelle consiste à décrire et tester séparément logiciel et matériel. JHDL [30] permet la description, la simulation et l'exécution de l'application au moyen d'un unique langage issu de *Java*. JHDL propose un ensemble de classes modélisant les divers composants d'un circuit numérique comme des composants logiques, des bascules ou des additionneurs.

Chaque famille de FPGA dispose de plus d'une bibliothèque définissant ses particularités. Il est ainsi possible d'instancier une LUT d'un Virtex. Ce langage implique de bonnes connaissances du matériel utilisé.

II.1.6. Discussions sur les outils de programmation FPGA

La plupart des outils proposent des environnements de développement pour faciliter la conception des circuits sur puces, mais souvent, des connaissances spécifiques matérielles sont encore nécessaires implicitement à plusieurs niveaux. D'où la nécessité d'ajouter un outil de vérification en parallèle avec l'outil de conception comme les méthodes formelle, leur caractéristique en font un outil fondamental, qui amène une certaine rigueur dans le processus de conception de développement. C'est cette rigueur, jointe à des concepts de production, qui fait leur succès dans l'industrie. Il est à préciser que l'utilisation de ces méthodes est imposée par les

pouvoirs publics afin de vérifier l'élaboration de systèmes numériques ou informatiques en relation avec des êtres humains, en particulier pour la conception des architectures complexes dont un dysfonctionnement entraînerait des conséquences inacceptables.

II.2. MÉTHODES FORMELLES

II.2.1. Définition

Une méthode est par définition une démarche permettant d'atteindre un objectif donné. Cette démarche (dans le cas particulier d'une méthode informatique) doit fournir des outils de modélisation et de raisonnement. Par « formelle », il faut comprendre l'établissement de règles strictes afin de structurer la « forme », le tout au sens mathématique. En effet, la rigueur de cette discipline correspond bien à cette idée de règles. Une utilisation de ces concepts est permise grâce à un langage formel, offrant une syntaxe ou des notations (ensemblistes, logiques, etc.) à un énoncé. Le langage d'une méthode formelle forme sa sémantique. Chaque énoncé et/ou expression possède ainsi une signification mathématique précise, qui n'a d'autre sens que celui décrit par la syntaxe du texte (toute interprétation personnelle est bannie). En l'occurrence, le but d'une méthode formelle est de proposer un processus de production, ainsi que des outils permettant d'offrir une certaine « abstraction » (fournie par la rigueur mathématique intrinsèque à chaque méthodologie) au logiciel à développer. L'abstraction permettra de représenter un système suivant des notions « génériques » (au sens de haut niveau : produit cartésien de deux ensembles finis pour « simuler » une table, ensembles mathématiques pour représenter les types des valeurs, etc.). Les méthodes formelles décrivent [9] :

- Les hypothèses sur l'environnement dans lequel le système va évoluer.
- Les contraintes que le système doit établir.
- Une implémentation respectant ces conditions.

Ainsi, elles peuvent être utilisées lors des différentes étapes du processus de développement d'un système. Une méthode formelle comporte [10] :

- Un langage formel dans lequel le système et ses propriétés pourront être représentés, c'est-à-dire formalisés ou modélisés formellement.

- Un ensemble d'outils pour raisonner sur des éléments de ce langage.

Dans cette représentation, le concepteur se focalise sur l'essence même des propriétés de son système (le sens logique des propriétés données au système). Il modélise de manière « mathématique » son système, puis étudie le bien fondé de ses spécifications. En aucun cas, le concepteur ne travaille sur la description « bas niveau » de son système implémenté dans un langage de programmation quelconque.

II.2.2. Intérêt d'une vérification par preuve formelle

Afin de dégager les bienfaits des méthodes formelles, il est nécessaire au préalable d'analyser certaines approches possibles pour décrire un système. L'objectif est d'en dégager les points essentiels. Le langage naturel, est la première étape de description d'un système. Cependant, ce type de description ne convient pas, principalement à cause de certains critères :

- Manque de précision dans le discours.
- Spécifications trop ambiguës et/ou incomplètes correspondent à un défaut pouvant conduire à une perte de temps, d'argent, etc.
- Impossibilité de prouver quelque énoncé « oral » que ce soit.

De même, la description de spécifications dans un langage de programmation quelconque pourrait sembler convenir, puisque c'est la transcription des fonctions du système. Après réflexion, il semble que ce mode ne corresponde pas à l'utilisation attendue pour les raisons suivantes :

- Manque d'abstraction flagrant (un code avec des noms de variables quelconques, des traitements tout autant « inconnus » ne veulent rien dire ; sorti de son contexte, à la limite, personne ne sait ce que fait le système, excepté peut être le concepteur, etc.)
- Particularité de l'étude dans un langage de programmation. Une solution correspondant à un cas particulier des solutions possibles de représentation « concrète » du système. Solution où des choix ont été faits et où la « généralité » (abstraction) recherchée est perdue. Dès lors, toute preuve de certaines propriétés du système est impossible.

Ce constat ayant été fait, il semble nécessaire de se tourner vers une autre représentation, plus « abstraite ». L'intérêt des solutions abstraites provient évidemment de leur « originalité » dans le processus de production industriel mais surtout de leurs propriétés (architecture, rigueur, sémantique stricte, interprétation personnelle impossible, etc.).

Comme précédemment, il est possible de présenter brièvement les apports de cette représentation :

- Apporte une sémantique claire et surtout non ambiguë (par leurs principes mathématiques).
- Conduit à de précises et concises descriptions des logiciels.

- Propose une abstraction de haut niveau.
- Les démonstrations de propriétés sont possibles car la syntaxe mathématique est offerte « à l'origine », c'est l'état de base du projet.

Devant ces avantages certains, la spécificité du mode de développement suivant des spécifications formelles suppose quelques inconvénients :

- Les spécifications formelles sont difficiles à comprendre pour les non-initiés, leur « côté mathématique » peut rebuter certains informaticiens proches de l'aspect codage.
- Pour les mêmes raisons que précédemment, leur écriture n'est pas facile, car l'abstraction est initiée par le concepteur. Si ce dernier écrit ses spécifications comme s'il les écrivait dans un code source quelconque, le résultat sera douteux, etc.

II.3. MÉTHODES FORMELLES ET CYCLE DE VIE DU LOGICIEL

Dans le cycle de vie d'un logiciel, les méthodes formelles apparaissent à différentes étapes et de plusieurs manières. Bien entendu, la rédaction des spécifications de l'application doit être réalisée dans un langage rigoureux ne donnant pas prise aux interprétations. Lorsque cette phase est terminée, plusieurs actions sont possibles :

- La validation des spécifications : par exemple, la vérification par évaluation sur un modèle (model checking), utilisée pour les protocoles, permet de valider un protocole en testant toutes les exécutions possibles et en y recherchant si telle propriété attendue est vérifiée ;
- L'étude de la cohérence des spécifications : par exemple, dans une description axiomatique, il est important de savoir si les axiomes définis par l'utilisateur admettent un modèle non vide. Le système de vérification PVS, par exemple [11], possède des outils mécanisés qui vérifient certaines conditions de cohérence ;
- Le prototypage des spécifications : si l'on considère PROLOG ou les langages fonctionnels de type LISP ou ML sous leurs formes pures comme des langages permettant d'écrire des spécifications exécutables, ils servent alors d'outils de prototypage des spécifications. Des systèmes tels que VDM ou B possèdent des interpréteurs permettant d'exécuter les spécifications. Lors du développement proprement dit de l'application, l'utilisation de méthodes formelles présente des vantages :
- Les méthodes de type Hoare ou Dijkstra permettent de démontrer manuellement que le code vérifie localement ses pré- et post-conditions. Celles-ci sont complétées par le calcul du raffinage de Morgan appliqué aux algorithmes [13] ou aux données [103].

- Il existe différentes techniques de raffinage permettant de transformer par étapes la spécification, en général très abstraite, en un programme exécutable. Les étapes de ce raffinage sont prouvées en garantissent la sûreté.

Une technique de raffinage est en général très liée au langage de spécification. Lorsque le développement est terminé, l'utilisation d'une méthode formelle, Est-elle ou VDM par exemple, permet la génération automatique ou semi-automatique de jeux de tests. Cela permet de valider a posteriori, sans la garantir totalement, la mise en œuvre qui a été faite des spécifications. On distingue les tests générés directement à partir des spécifications des tests obtenus à partir de comportements simulés. Les apports énoncés ici se retrouvent également dans les évolutions futures de l'application spécifiée. Toute modification ou tout complément des spécifications pourra être validée, prototypée, raffinée, testée selon les mêmes méthodes. C'est donc une garantie de plus quant à l'évolutivité de l'application. Il est bien évident que le coût de ces méthodes est élevé. La durée de vérification est allongée par les étapes formelles qui ne sont pas directement liées au codage de l'application, par exemple les démonstrations de cohérence. Ces méthodes nécessitent, pour être applicables, des ateliers informatisés en règle générale très lourds. Actuellement, il n'existe pas encore de méthodes formelles permettant de sécuriser, de manière réaliste, le cycle dans son entier. Si nous prenons la technologie B de J-R. Abrial pour laquelle il existe de bons outils informatiques, et dont le développement est parmi les plus avancés, le développement d'une application demande la démonstration d'une quantité de théorèmes que l'on finit toujours par être tenté de demander au système d'en accepter certains comme axiomes.

II.4. NIVEAUX D'ABSTRACTION DES MÉTHODES FORMELLES

On peut distinguer plusieurs niveaux d'implication des méthodes formelles dans le processus de développement d'une application. Il existe un niveau adapté à chaque cas de figure, prenant en compte les exigences de délais, de coûts, de qualité, de sécurité et l'état de la technologie. La classification suivante est utilisée comme suit [104] :

- **Niveau 0**

Aucune utilisation d'une méthode formelle. Les documents sont exclusivement en langue naturelle utilisant éventuellement du pseudo-code et des diagrammes explicatifs. Leur utilisation est purement manuelle et la validation du logiciel obtenu est faite par des procédures extensives de tests. Il serait cependant hâtif de conclure qu'un tel développement est purement non formel : la programmation est bel et bien une formalisation. Mais les langages de programmation couramment utilisés se prêtent mal au raisonnement et à l'analyse mathématique.

- **Niveau 1**

Utilisation de notations et de concepts formalisés, par exemple tables ou organigrammes. Le langage de spécification est plus rigoureux mais permet difficilement d'approcher les détails. Les preuves, quand il y en a, restent informelles. Ce niveau complète habilement une méthode classique de développement.

- **Niveau 2**

Utilisation de notations formelles et d'outils semi-automatisés. Les spécifications sont rédigées dans un langage mathématique : théorie des ensembles, théories logiques, théorie des types, etc. Des outils informatisés permettent certaines facilités : aide à l'édition, vérification de consistance, gestion des dépendances, etc. Ce niveau est parfois atteint par de petits projets de développement dans un domaine très qualifié.

- **Niveau 3**

Utilisation de notations formelles et d'un environnement d'outils détaillés et complets. Comme au niveau 2, les spécifications sont mathématiques et rigoureuses. L'utilisation d'outils plus ou moins automatiques d'assistance à la preuve sécurise l'ensemble du processus de développement du logiciel. Le logiciel est certifié conforme aux spécifications initiales. Tant en raison des limitations techniques actuelles de ces outils que pour des raisons évidentes de coût de développement, ce niveau de rigueur est réservé à de petites applications – éventuellement complexes – dans des domaines très spécifiques.

II.5. TECHNIQUES DE VÉRIFICATION FORMELLE

La simulation et les tests sont les deux en effet, techniques qui contribuent depuis longtemps à valider et tester l'exactitude d'un système critique. Le système à vérifier est testé pour révéler d'éventuelles erreurs sur un ensemble de situations choisies. Faute de temps, il est souvent impossible de tester ce système pour toutes les situations possibles. L'ensemble des cas à tester doit être choisi avec une attention particulière pour couvrir le maximum de scénarios distincts possibles. Néanmoins, il est impossible de garantir que le produit sera exemple d'erreur [105]. Cependant, ces techniques sont tout à fait insuffisantes et permettent d'explorer qu'une partie des comportements possibles. Alors que les techniques de vérification formelle, garantissent qu'une propriété est vérifiée par la totalité des exécutions possibles du système.

Les deux principales techniques de la vérification formelle sont :

- **La vérification déductive (Theorem proving)**

Cette approche a pour principe de poser un modèle du système à vérifier qui est vu comme un ensemble d'axiomes, souvent donnés par le concepteur. Ensuite, il s'agit de prouver un ensemble d'assertions déterminant ainsi la conformité du système. Cette preuve est faite plus ou moins manuellement. La vérification est faite suivant la structure syntaxique. Ce type de vérification est utilisé essentiellement dans le domaine des spécifications algébriques et logiques.
Exemples : triplets ou logique de Hoare {P} S {Q} (Hoare 1969).

L'inconvénient de l'approche *Theorem proving* est la nécessité d'un utilisateur expert et beaucoup de main-d'œuvre pour prouver une quantité raisonnable du code. C'est probablement la raison pour laquelle très peu de réussites industrielles ont été obtenues avec cette méthode.

- **Vérification sémantique (model checking)**

Cette approche est basée sur les modèles, et permet une vérification simple et efficace. La vérification basée sur les modèles ou Model-checking est surtout applicable pour les systèmes ayant un espace d'états fini. Les algorithmes de vérification dans cette méthode utilisent l'ensemble des états que le système peut atteindre pour prouver la satisfaction ou la non-satisfaction des propriétés. Elle regroupe aussi plusieurs techniques entièrement automatiques dans lesquelles la propriété à vérifier est testée de façon exhaustive sur l'ensemble des exécutions possibles du système.

Les différentes techniques du model checking diffèrent par leur façon de représenter ces ensembles (infinis) d'exécutions. Citons le model checking dit *symbolique* [105] qui travaille de façon ensembliste sur les configurations que peut prendre le système, et le model checking par approche automate, dans lequel l'ensemble des exécutions du système ainsi que la propriété sont représentés par des automates.

Le domaine d'application de cette méthode est très vaste tel que la conception de systèmes matériels à contrôle intensif (control-intensive systems hardware), les protocoles de communication, pilotes, etc.

Les principaux inconvénients du Model Checking sont :

- Que la preuve d'un système n'aide pas à le comprendre.
- De la difficulté d'écriture d'une spécification.
- De l'explosion possible des états d'un système pouvant conduire à un problème majeur qui limite la taille des systèmes vérifiables.

En termes de travaux académiques, l'une des dernières techniques de model checking pour la vérification des systèmes complexes a été proposée [7]. Cette technique s'appuie sur le graphe quotient des états d'un système. Bien que récente, cette théorie est implémentée dans des outils qui se sont montrés capables d'analyser des spécifications des systèmes réels [7]. En 2007, J. Sifakis [105] présente un ensemble de perspectives pour pallier aux problèmes liés à l'utilisation des méthodes de Model Checking. En particulier, la nécessité :

- De garantir qu'un modèle représente fidèlement le système. Le terme "fidèlement" signifie ici que la relation entre le modèle et le système puisse être définie au travers d'une sémantique formelle vérifiable.
- D'améliorer le passage à l'échelle. Bien que Les algorithmes de vérification actuels sont assez efficaces, ils souffrent de limitations liées à la complexité inhérente aux systèmes composés d'un nombre élevé de composants. Deux approches alternatives sont alors envisagées.

La première consiste à calculer des propriétés sur le système à partir de propriétés calculées à partir de ses composants et de règles d'assemblages. La deuxième consiste à élaborer des preuves de propriétés autour d'architectures formelles concernant une classe de systèmes [105].

Il est à noter que les techniques de preuve ont l'avantage de pouvoir traiter des systèmes à nombre d'états infini contrairement à la technique de model checking qui repose sur des systèmes finis et décidables. Néanmoins, l'une des limitations de l'utilisation de la preuve en industrie est le fait que les prouveurs de théorèmes ont besoin d'être assistés par des ingénieurs expérimentés. En effet, quel que soit leur niveau d'automatisation, tous les outils de preuve nécessitent l'intervention d'un spécialiste, car leur usage requiert une très grande connaissance aussi bien au niveau du modèle à vérifier que des techniques de preuves utilisées.

II.6. SYSTÈMES FORMELS, VÉRIFICATION, PREUVE, MODEL-CHECKING

L'approche formelle fait appel à un système formel, en tant que support de modélisation. Tout système formel comprend trois éléments de base (L ; S ; fs), où L est un langage, défini en donnant sa syntaxe et utilisé pour construire des modèles formels de spécification. S est un domaine sémantique, une structure mathématique constituée d'ensembles, fonctions, relations, opérateurs, etc. La fonction fs, dite fonction sémantique, établit une correspondance entre une entité syntaxique et une entité sémantique. En général, on définit fs de façon inductive, à partir de la définition des règles syntaxiques de L. Aux trois éléments de base, on ajoute une troisième composante, T, qui est

une théorie du premier ordre (TPO), définie sur les entités du domaine sémantique, éventuellement augmenté d'autres domaines. Une TPO est une structure de la logique formelle, qui comprend un ensemble de prédicats qui s'appliquent aux entités du domaine sémantique et un langage d'expressions logiques, défini à partir des prédicats, à l'aide d'opérateurs logiques classiques et/ou modaux. La TPO peut inclure également un système de déduction, constitué d'un ensemble d'axiomes et d'un ensemble de règles de déduction. Le rôle de la TPO sera d'exprimer des propriétés relatives aux entités sémantiques. La présence d'un système de déduction introduit la possibilité de prouver les dites propriétés, relativement à des entités sémantiques. Il est à signaler que pour certains systèmes formels, le domaine sémantique et la TPO se confondent. C'est le cas des formalismes basés sur la logique du premier ordre et la théorie des ensembles.

La TPO T associée à un système formel (L ; S ; fs) permet de formuler des propriétés relatives aux entités sémantiques. Une formule logique P peut être évaluée à partir d'une entité sémantique $\sum$. Cette dernière peut fournir des valeurs aux variables libres de P. La façon dont $\sum$ fournit des valeurs permettant d'évaluer la formule P dépend de la nature de l'entité sémantique $\sum$. Un cas fréquent est le suivant : l'entité sémantique $\sum$ est un système de transition qui produit potentiellement un ensemble dit comportement de $\sum$, noté $C(\sum)$. Les éléments de $C(\sum)$ sont des chaînes, c'est à dire des suites infinies d'états du système $\sum$. La propriété P peut être évaluée relativement à chacune de ces chaînes. Dans ce cas, la TPO associée appartient sûrement au domaine des logiques temporelles, qui sont les plus aptes à exprimer des propriétés relatives à des chaînes.

Si $\sigma \in C(\sum)$, nous notons σ [P] l'évaluation de la propriété P par σ, et nous notons $\sigma \models P$ si σ [P] = vrai. Dans ces conditions, $\sum$ une entité sémantique de comportement $C(\sum)$, et P une propriété. Nous dirons que P est valide relativement à $\sum$ si, quelle que soit $\sigma \in C(\sum)$, $\sigma \models P$. Dans ce cas on note : $\sum \models P$.

La vérification de la validité est une forme de test qui s'applique à $\sum$ et à P, et qui répond par oui ou par non à la question : P est-elle valide relativement à $\sum$? Suivant la mise en œuvre de l'algorithme de vérification, on parlera de preuve (theorem proving) ou de model-checking.

La preuve repose sur la présence d'un système de déduction associé à la TPO du système formel. L'entité sémantique $\sum$ est représentée par un ensemble de formules $H(\sum)$ qui jouent le rôle d'hypothèses. Dans ce cas, on montre $\sum \models P$ en vérifiant, par une preuve de P sous les hypothèses $H(\sum)$ (ce qui est noté $H(\sum) \vdash P$).

Le model-checking est basé sur les procédures de décision de la validité ou de la satisfiabilité. C'est-à-dire, la recherche de modèles, au sens logique du terme, par exploration de l'espace d'états.

Si nous considérons, par exemple, le cas des systèmes de transition, les modèles de P sont les chaînes σ telles que σ |= P. En général, les algorithmes de décision les plus efficaces sont ceux qui décident de la satisfiabilité (appelés SAT solvers). Dans ce cas, on vérifiera que ∑ |= P en cherchant à satisfaire ¬ P. Si l'on trouve une chaîne σ ∈ C(∑) telle que σ |= ¬ P, alors σ est un contre-exemple de la propriété étudiée (P). Si aucun contre-exemple n'est trouvé, on conclut que ∑ |= P.

Pour finir cette section, quelques observations relatives au type de propriétés soumises à vérification. Les deux principales classes sont les propriétés de sûreté et de vivacité. Elles s'expriment facilement à l'aide de deux opérateurs de la logique temporelle linéaire : ◊ (éventuellement) pour la vivacité et □ (nécessairement), pour la sûreté. Si P est une formule logique d'état (c'est à dire sans opérateur temporel) exprimant une propriété de sûreté, alors il s'agira d'établir la validité de la formule □ P, ce qui veut dire "*P est satisfaite par tout état du système visité lors de toute évolution possible*". En d'autres termes, P est un invariant du système. Les méthodes de vérification basées sur le model-checking sont bien adaptées à la décision de la satisfiabilité et de la validité des formules de la logique temporelle, dont l'invariance. Les méthodes basées sur la preuve le sont moins en général, sauf justement pour les propriétés d'invariance qui sont en général vérifiables par des techniques de preuve inductive.

Nous allons présenter quelques langages formels qui s'inscrivent dans le cadre conceptuel que nous venons de présenter et qui, par conséquent, sont adaptés à une démarche de vérification par la preuve ou le model checking.

II.7. APPROCHE FORMELLE

Les spécifications formelles utilisent des notations mathématiques pour décrire de façon précise les propriétés qu'un système d'information doit posséder. Des méthodes ensemblistes comme VDM ont été développées à partir des années 60, et cette approche continue avec l'apparition plus récente de la notation Z et de la méthode B.

Comme il y a d'autres méthodes de modélisation et visualisation en parallèle citons :

II.7.1. Réseaux de Pétri

C'est un outil de modélisation utilisé généralement en phase préliminaire de conception de système pour leur spécification fonctionnelle, modélisation et évaluation.
Les principaux utilisateurs de ces réseaux sont les informaticiens et les automaticiens. Cependant c'est un outil assez général pour modéliser des phénomènes très variés. Il permet notamment :

- la modélisation des systèmes informatiques,
- l'évaluation des performances des systèmes discrets, des interfaces homme-machine,
- la commande des ateliers de fabrication,
- la conception de systèmes temps réel,
- la modélisation des protocoles de communication,
- la modélisation des chaines de production (de fabrication),
- ...

En fait, tout système dans lequel circule objets et information :

- ils permettent de d´écrire de manière précise mais non formelle la structure d'un système,
- ils offrent un support graphique de conception,
- ils permettent de décrire un système étape par étape, en décomposant en ´éléments plus simples les éléments constitutifs initiaux du système,
- ils permettent de décrire à l'aide d'un même support de base, à la fois la structure et la dynamique d'un système,
- ils permettent de passer d'une description graphique d'un système à une description formelle permettant l'analyse mathématique du système (cohérence).

II.7.2. Diagramme états de transitions

Un diagramme états-transitions est un schéma utilisé en génie logiciel pour représenter des automates déterministes. Il fait partie du modèle UML et s'inspire principalement du formalisme des *statecharts* et rappelle les grafcets des automates. S'ils ne permettent pas de comprendre globalement le fonctionnement du système, ils sont directement transposables en algorithme. Tous les automates d'un système s'exécutent parallèlement et peuvent donc changer d'état de façon indépendante.

- **Eléments**

Transitions : En plus des états de départ (au moins un) et d'arrivée (nombre quelconque), une transition peut comporter les éléments facultatifs suivants :

- Un évènement
- Une condition de garde
- Une liste d'actions

Quand l'événement se produit alors que les états de départ sont actifs et que la condition de garde est vraie alors les actions seront déclenchées.

Etats : Cette exécution est enrichie lorsque les états définissent une action d'entrée et une action de sortie : l'action de sortie de l'état de départ est exécutée d'abord, puis l'action de la transition, puis l'action de l'état d'arrivée.

Autres éléments : Les points de jonction ne sont qu'un élément graphique permettant de regrouper plusieurs segments de transition de façon à rendre le schéma plus lisible. Les points de décision permettent de simuler un choix : si-alors-sinon, qui entraîne deux états différents.

II.7.3. Méthode VDM

VDM (Vienna Development Method) est plus fondée sur une vision dénotationnelle et opératoire – transformant ainsi les programmes en fonctions mathématiques – que sur une méthode basée sur la théorie axiomatique des ensembles, comme le sont Z ou B. La méthode VDM utilise un langage appelé à l'origine Meta-IV et maintenant VDM-SL (VDM Specification Language). Elle donne la dynamique des systèmes informatiques spécifiés au moyen de prédicats de type pré- ou post-conditions. VDM-SL permet une approche modulaire. Les données et les opérations peuvent être présentées comme des modules (passage des spécifications au programme) en utilisant des règles logiques sur les obligations de preuve qui sont systématiquement engendrées à partir de la théorie de VDM [8]. Comme en Z, on distingue les variables avant et après une opération par une décoration. Le système logique employé par VDM comporte trois valeurs (vrai, faux, indéfini). La valeur indéfinie est utilisée lorsqu'une fonction partielle apparaissant dans une expression logique et est appliquée en dehors de son domaine de définition. Malgré son nom, VDM est plus une notation qu'une méthode, et elle ne possède pas d'outils d'aide à l'écriture de spécification ni de mécanisme de raffinage.

II.7.4. Méthode Z

Une spécification en Z est constituée d'ensembles que l'on utilise pour spécifier un système informatique et de schémas. Une spécification sera constituée d'un morcellement de schémas. Les schémas sont utilisés pour décrire à la fois les aspects statiques et les aspects dynamiques d'un système [38]. On entend par aspects statiques les états et les relations d'invariants entre les états du système et par aspects dynamiques les opérations qui sont possibles, les relations entre les entrées et les sorties et les changements d'états qui peuvent arriver. Le nom du schéma est inscrit dans la première ligne du cadre qui l'entoure. Une ligne horizontale sépare la partie déclarative et la partie réservée aux prédicats. S'il y a plusieurs prédicats, on considère leur conjonction :

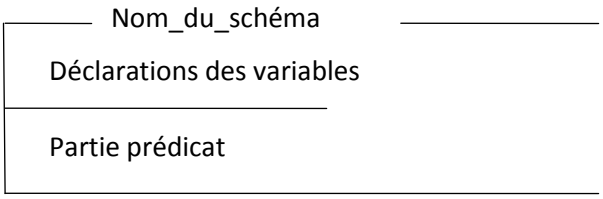


FIGURE II. 2 *ARCHITECTURE D'UN SCHEMA DE SPECIFICATION* Z [106]

On peut également noter un schéma sous forme textuelle, telle que :

<Nom_du_schéma>==[Declarations_des_variables|Partiepredicat]

On y reconnaît tout simplement la définition d'un ensemble par compréhension. Les données d'un système sont munies d'un type. Ces types de données ne sont pas interprétés par une représentation informatique, mais par des ensembles construits au moyen d'ensembles prédéfinis (comme celui des entiers relatifs) et des opérateurs ensemblistes usuels (union, produit cartésien, etc.). Les opérations sont spécifiées par un schéma avec les variables avant l'opération et les variables après l'opération. On distingue d'ailleurs les variables après l'opération par une apostrophe. Une spécification Z utilise deux catégories de symboles :

— les symboles de la logique des prédicats¬, ∧, ∨, ⇒, ⇔, ∀, etc.

— les symboles de la théorie des ensembles, ceux de base (∪, n

Mais aussi le cardinal avec #, l'ensemble des parties finies d'un ensemble avec des points (…) ou les symboles fonctionnels et relationnels (↔, →, des distinctions sont faites entre les fonctions injectives, surjectives, partielles ou totales) [106]. Plusieurs outils d'aide à la spécification et à sa vérification existent en Z. On distingue plus particulièrement celui développé à l'initiative de M. Spivey. Cet ensemble d'outils permet une aide à l'écriture d'une spécification et une vérification de types.

II.7.5. Méthode B

La méthode B est issue de travaux menés par J-R. Abrial dans les années 80, après ses travaux sur Z. Elle regroupe :

— un langage de spécification AMN (Abstract Machine Notation) assorti d'un système de preuves ;

— une technique de raffinage et de codage et les systèmes de preuves correspondants. Comme Z, B repose sur la théorie des ensembles et la logique des prédicats du premier ordre. Contrairement à Z,

B a une coloration développement dans la façon de spécifier les opérations : en effet, on ne les spécifie pas en termes de pré- et post-conditions, mais au moyen de substitutions généralisées. Ce mécanisme peut s'interpréter comme une extension de l'affectation telle qu'elle existe dans les langages impératifs, et que nous allons exposer dans la suite. Enfin, B offre une notation homogène pour la spécification et le développement.

II.8. MÉTHODES FORMELLES ET LE MODÈLE CHECKING

Les concepteurs de logiciel savent par expériences que le code du logiciel n'est pas susceptible de fonctionner correctement après une, deux ou trois compilations. Parfois, cela prend un certain temps pour découvrir pourquoi un programme « correct » échoue. Pour répondre à ce besoin, de nombreuses méthodes utilisant des techniques de model checking ont été développées.

- ***Techniques basées sur la simulation***

En pratique, l'une des techniques de vérification la plus connue et utilisée est la simulation (Monin, 2000) (Baier et Katoen, 2008). Le simulateur permet à l'utilisateur d'étudier le comportement du système. Ceci passe par la détermination, sur la base du modèle du système, des réactions que doit avoir le système vis-à-vis de certains scénarios spécifiques. Ces scénarios sont fournis par l'utilisateur ou générés par des outils tels que les générateurs aléatoires de scénarios.

La simulation est généralement utile pour une première évaluation rapide de la qualité du prototype (l'étape de conception). Elle est cependant moins adaptée à détecter les erreurs subtiles, car pour le simulateur, il est impossible de générer tous les scénarios possibles du système, et encore moins de tous les simuler. Dans la pratique, seul un petit sous-ensemble de tous les scénarios possibles est effectivement examiné. Par conséquent, il existe un risque réaliste que les défauts subtils restent cachés. Les scénarios inexplorés pourraient révéler l'erreur fatale. D'ailleurs, il est difficile de quantifier le degré d'exactitude du système lors de l'examen d'un nombre restreint de scénarios. De même, les mesures quantitatives du nombre d'erreurs laissées dans le système sont difficiles à obtenir ; tout comme des indications sur la probabilité que telles erreurs soient découvertes lorsque le système est en fonctionnement. Dans la pratique, cela signifie souvent que le critère pour arrêter la simulation est simplement au moment où le projet est à court de capitaux.

- ***Techniques basées sur le test***

Tandis que la simulation et le model checking sont basés sur une description du modèle où tous les états possibles du système peuvent être générés, la technique de vérification basée sur le test (Myers, 1979) (Ammann et Offutt, 2008) (Mathur, 2008) est applicable dans le cas où il est

difficile, voire impossible, d'obtenir un modèle du système à vérifier. Avec le test, des séquences d'actions, représentant différents scénarios d'exécutions possibles, permettent de vérifier une réaction spécifique. Un paramètre important du test est la mesure à laquelle l'accès à l'état interne du système testé peut être obtenu. Deux types de tests peuvent être effectués sur le système. Le test à boite blanche peut accéder totalement à la structure interne d'une implémentation, tandis que dans le test à boite noire, la structure interne est complètement cachée. En pratique, des scénarios intermédiaires sont souvent rencontrés, appelés test à boite grise. L'avantage principal du test est sa large applicabilité, en particulier pour les produits finaux, et il n'est pas restreint seulement aux modèles. Le désavantage est comparable à la simulation, car le test complet est pratiquement impossible. Comme la simulation, le test peut montrer la présence d'erreurs, et non leur absence.

Les tests sont plutôt improvisés et pas très systématiques. Comme résultat, le test est une activité très élaborée, prédisposant aux erreurs et difficilement gérable. Similaire au model checking, le point de départ de la méthode du test sur des modèles est la spécification précise du système. Avec les méthodes du test traditionnelles, une telle base est souvent absente. Basés sur cette spécification formelle, les algorithmes de la génération du test génèrent des tests probablement valides, et testent ce qu'il faut tester et pas plus. Les outils du test implémentant ces algorithmes, fournissent une génération de test automatique, plus rapide, et moins source d'erreurs.

Le test de régression implique la vérification du comportement correct d'une version modifiée d'un système existant. Cela implique typiquement l'adaptation, la sélection et la répétition des tests existants. Dans la méthode de test basée sur le modèle, une petite modification du système conduit seulement une adaptation de son modèle, et ensuite, un nouveau test peut être automatiquement généré.

En pratique, le modèle basé sur le test a été implémenté dans plusieurs outils logiciels et a démontré son efficacité dans diverses études (par exemple un logiciel d'application pour le e-business, les compilateurs et les systèmes d'exploitation). Pour plusieurs systèmes, comme les systèmes embarqués contrôlant l'échange d'informations, des erreurs ont été découvertes alors qu'elles n'ont pas été identifiées avec les techniques de tests conventionnelles.

- ***Techniques basées sur la preuve de théorème***

La preuve de théorème, (Cook, 1971, Dingel et Filkorn, 1995) nécessite que le système soit spécifié sous forme d'une théorie mathématique, ou doit être transformée en une telle forme. En utilisant un ensemble d'axiomes (le théorème de base), un démonstrateur (le logiciel) tente de construire soit une preuve de théorème en générant les étapes de preuves intermédiaires ; soit de réfuter les

axiomes énoncés. Les axiomes sont intégrés ou fournis par l'utilisateur. Les démonstrateurs sont également appelés assistants de preuves. La demande générale de prouver des théorèmes d'un type assez général et l'utilisation de logiques indécidables exigent certaines interactions avec l'utilisateur. Il existe différentes variantes : hautement automatisée avec les programmes d'assistance de preuves, et interactifs avec des capacités spéciales.

Les vérificateurs de preuves sont des assistants de preuves automatisées qui nécessitent une interaction limitée avec l'utilisateur. Principalement, le vérificateur vérifie si un utilisateur fournit une suggestion de preuve valide ou non. La capacité du vérificateur de preuves à produire les mesures de preuves intermédiaires de façon automatique est plutôt limitée. Afin de réduire le temps de recherche à démontrer le théorème, l'interaction avec l'utilisateur prend place. L'utilisateur peut aider à trouver la meilleure stratégie à mener sur une preuve. Habituellement, les assistants de preuves interactifs aident à donner une preuve en gardant la trace des choses restantes à faire et en fournissant des conseils sur la façon dont ces théorèmes peuvent être prouvés. Le degré d'interactions avec l'utilisateur est généralement assez élevé. Cela couvre non seulement le contenu du théorème, mais aussi la façon dont il est utilisé. En outre, l'utilisation de démonstrateur ou de vérificateur de preuves exige une plus grande habitude des utilisateurs. En général, on évite des petites parties de preuves (triviales ou analogues à), alors que l'assistant de preuves exige explicitement ces étapes. Le principal avantage de la preuve de théorèmes est qu'il peut agir avec des espaces d'états infinis et peut vérifier la validité des propriétés pour des valeurs de paramètres arbitraires. Par contre, l'inconvénient principal de la preuve de théorèmes est la lenteur du processus de vérifications, le risque d'erreurs et le temps utilisé pour guider la preuve. Aussi, la logique mathématique utilisée par l'assistant de preuves exige un degré assez élevé d'expertise des utilisateurs.

II.9. CHOIX DE LA TECHNIQUE DE VÉRIFICATION

Nous savons que les méthodes basées sur des modèles de systèmes sont plus pertinentes et plus précises pour la détection d'erreurs que les méthodes se lançant directement à la recherche d'erreur sur les systèmes. La simulation est l'une des techniques les plus utilisées actuellement, mais son grand désavantage est que le simulateur ne peut pas générer tous les scénarios possibles. Par conséquent, des erreurs fatales peuvent ne jamais être détectées. La preuve de théorème demande une grande interaction avec l'utilisateur ce qui rend la technique moins automatique. Le test est une technique très puissante et les tests traditionnels sont moins adaptés que les tests sur les modèles de systèmes. Le test reste toujours limité par la découverte des erreurs et non de leurs

absences. La technique du model checking comme méthode formelle est une méthode très performante et très utilisée dans le domaine des protocoles de communication dans les réseaux informatiques.

Cette technique correspond au mieux à notre domaine, car elle pourra révéler des erreurs non détectées par les autres méthodes formelles comme le test et la simulation. La technique du model checking sera déployée pour la vérification de la pertinence des graphes sémantiques.

II.10. OUTILS DE VÉRIFICATION

La conception moderne des systèmes embarqués électroniques fait maintenant appel à des outils de vérification formelle basés sur des techniques très efficaces pour manipuler leurs fonctions et leurs comportements. Ces outils permettent de s'assurer qu'un système possède bien les fonctionnalités qu'on lui suppose. Plusieurs assistants (ou prouveurs) ont été développés afin d'automatiser le processus de preuve, c'est-à-dire finaliser une preuve suggérée par l'utilisateur grâce aux différentes méthodes. On peut citer dans ces domaines les travaux et outils suivants : On trouve : PVS (1996), Coq (2002), HOL (1993), Isabelle (1994) et autres, dont les plus courants sont présentés dans le paragraphe suivant.

II.10.1. Outils de vérification par preuve

Ces langages sont généralement considérés comme de bons outils pour la conception et le prototypage des programmes sûrs et corrects.

- **Coq**

C'est un assistant de preuve développé au sein de l'INRIA. Il permet de proposer une spécification d'un système sous la forme de théorèmes et d'assertions. Le cadre de développement de Coq facilite la preuve mécanique de ces théorèmes. Le pouvoir expressif de Coq est tel que l'on peut envisagé des preuves sur des notions mathématiques très avancées, par exemple le théorème des 4 couleurs ou des programmes de complexité importante (comme un compilateur pour un noyau significatif du langage C). Coq est basé sur le lambda calcul (formalisme de représentation des fonctions mathématiques avec des principes de logiques qui permet d'exprimer toutes les fonctions calculables) et la logique d'ordre supérieur (les quantificateurs portent sur les prédicats et les fonctions sont considérées comme des variables et non des fonctions) [107].

- **Isabelle**

C'est un prouveur de théorème développé entre l'université de Cambridge et l'Université de Munich. Il est considéré comme un environnement interactif de développement pour mettre en œuvre la théorie des formalismes logiques. Il a été instancié pour soutenir un raisonnement du premier ordre logique, d'ordre supérieur, de la théorie des ensembles et des systèmes logiques modales. Ses principaux domaines d'utilisation sont la vérification des protocoles des compilateurs, la sécurité et la cryptographie, les logiciels, le matériel et la formalisation des langages de programmation et des mathématiques. Comparé à Coq, Isabelle est un outil très utilisé. Il propose une plus grande automatisation de la phase de preuve [108].

- **PVS (Prototype Verification System)**

Le prouveur PVS est un système pour spécifier et vérifier des propriétés sur des systèmes. Il repose sur la logique d'ordre supérieur classique et fournit un ensemble de procédures primitives qui sont appliquées de manière interactive sous la conduite de l'utilisateur. Les conclusions incluent des règles propositionnelles, des quantificateurs, l'induction, la réécriture, la simplification des procédures de décision à l'aide de l'arithmétique, les données et l'abstraction prédicat, et la vérification de modèle symbolique. PVS inclut une procédure de décision basée sur BDD relationnelle et fournit ainsi une intégration expérimentale entre théorèmes et le model checking de CTL [107].

II.10.2. Outils de vérification par model checking

Il existe plusieurs outils de vérification pour model checking. Parmi eux, on trouve :

- **TLC**

C'est un outil utilisé pour vérifier des spécifications écrites en TLA+ à différents niveaux de raffinements.

Il comprend des composants pour l'analyse syntaxique, le formatage, la simulation et l'analyse exhaustive [21]. TLC met en œuvre les techniques du model checking et permet de valider les spécifications écrites. On peut utiliser TLC pour réaliser des tâches liées à la validation des spécifications temporelles TLA+ [109].

- **SMV**

L'outil de vérification SMV a été développé par Kenneth McMillan [107], et permet de vérifier des propriétés temporelles des systèmes de grande taille, grâce à la combinaison d'un vérificateur sur

modèle symbolique et d'un prouveur spécialisé. SMV est très efficace pour vérifier automatiquement les propriétés de la logique combinatoire en interaction des machines à états finis. Parfois, lors de la vérification des propriétés de la logique de contrôle complexes, le vérificateur devra produire un contre-exemple. Il s'agit d'une trace de comportement qui viole la propriété spécifiée. Cela rend SMV un outil de débogage très efficace, et est donc un système de vérification formelle.

- **SPIN**

L'outil Spin vise la vérification efficace des logiciels et n'est pas adapté à la vérification du matériel. L'outil prend en charge un langage de haut niveau pour spécifier des descriptions des systèmes, appelés PROMELA (Un métalangage de processus). Il peut également être utilisé comme un moyen efficace pour vérifier des propriétés de sécurité et de vivacité [107].

CONCLUSION

Les méthodes formelles sont apparues depuis longtemps en milieu académique. L'idée même de vérifier la correction des programmes date des années 60 et remonte même à l'apparition des ordinateurs. Quelques expériences en milieu industriel ont eu lieu durant les années 70, mais sans lendemain faute notamment d'outils de support. En effet, les manipulations formelles requises sont fastidieuses et sujettes à des erreurs lorsqu'elles sont manuelles. Actuellement, il y a un tournant avec l'arrivée à maturité de plusieurs méthodes, d'outils munis d'interfaces qui se modernisent, et de logiciels d'assistance à la preuve de plus en plus efficaces. En particulier, nous pouvons mentionner le cas du langage B qui peut être exploité pour accélérer le prototypage de système et donc réduire le temps de vérification dans les domaines de la conception de systèmes électroniques et d'informatique industriel. Ceci nous pousse actuellement à travailler davantage sur la méthode formelle B pour l'aide à la vérification des systèmes électroniques.

CHAPITRE III :

LA METHODE B

INTRODUCTION

Parmi le nombre croissant de méthodes formelles, les méthodes orientées modèle telles que VDM, Z ou B semblent avoir prouvé leur applicabilité et leur efficacité. Ces méthodes sont basées sur la description du modèle. Elles consistent à définir un modèle par attributs variables qui caractérisent l'état du système décrit, les invariants et d'autres propriétés qui doivent être remplies, et les différentes opérations qui modifient ces variables.

Dans ce chapitre, nous allons argumenter le choix de la méthode B, ses fondements théoriques et son principe. Aussi, l'évolution du B vers l'Event-B sera relatée ainsi que la structure du modèle Event-B. Enfin, nous allons présenter les outils tels que la plateforme Rodin, l'atelier B, et le plug-in ProB.

III.1. LA METHODE B

La méthode B a été inventée par J. R. Abrial, et est décrit dans la référence auteur du livre The B Book [110] qui présente les fondements de cette méthode. Nous proposons ici une présentation de la méthode B qui a pour but d'introduire les concepts et les termes liés à la méthode B. Nous décrivons également l'outil utilisé lors des études, l'Atelier B développé par la société ClearSY.

III.1.1. Pourquoi choisir la méthode B

Le choix de la méthode B n'est pas dû au hasard. Il répond à un certain nombre de critères, notamment concernant l'applicabilité d'une telle méthode en milieu industriel. Comme évoqué dans l'introduction, la méthode B n'est qu'une méthode formelle comme une autre, une technique proposée par J. R. Abrial, influencé par ses travaux sur la méthode Z.

La méthode B a pour avantage d'avoir déjà été utilisée en milieu industriel réel, notamment dans le projet Météor qui a servi de projet pilote pour cette technologie [111]. De plus, elle couvre le cycle complet de développement et repose sur des fondements assez simples comme la logique des prédicats et la théorie des ensembles. Le fait de couvrir le cycle complet de développement permet en outre de mieux l'intégrer à un cycle industriel déjà existant. Cela rassure les industriels, leur montrant que les méthodes formelles ne sont pas uniquement réservées à un usage académique.

Le formalisme des ASMs est très proche de celui proposé dans B. Les travaux de Boerger [112] montrent de façon claire et précise leur bonne utilisation sur la modélisation des problèmes liés au langage Java. Cependant, les ASMs manquent d'un outil industriel de développement semblable à

l'Atelier B. Les considérations et les objectifs de cette thèse ayant de fortes composantes industrielles, nous avons préféré choisir la méthode B aux ASMs.

Ces deux avantages, l'expérience industrielle et les fondements théoriques de la méthode B, en font un candidat préférentiel pour mener nos travaux. Notre objectif à terme n'est pas de laisser l'utilisation des méthodes formelles à une activité de recherche ou d'experts dans le domaine. L'objectif est clairement d'industrialiser le processus de développement formel afin que les produits, et plus seulement les prototypes, bénéficient des apports de la modélisation et de la formalisation. Pour se faire, nous devons nous reposons sur des outils ou des techniques disparates et hétérogènes. Nous devons proposer une méthodologie cohérente, reposant sur des fondements simples et compréhensibles, abordables par nombre d'ingénieurs et développeurs en informatique. Bien sûr, l'intégration des technologies formelles dans les processus d'industrialisation nécessite de temps. Mais la méthode B nous apparaît comme le candidat idéal pour commencer cette approche et cette lente migration.

III.1.2. Les fondements théoriques

La méthode B est une méthode formelle destinée au développement de logiciels. Cette méthode englobe le processus complet de développement depuis la spécification jusqu'à l'implémentation, et permet de prouver que l'implémentation du logiciel est conforme à sa spécification. La preuve fait partie intégrante du processus, et chaque étape de la spécification à l'implémentation peut et doit être prouvée.

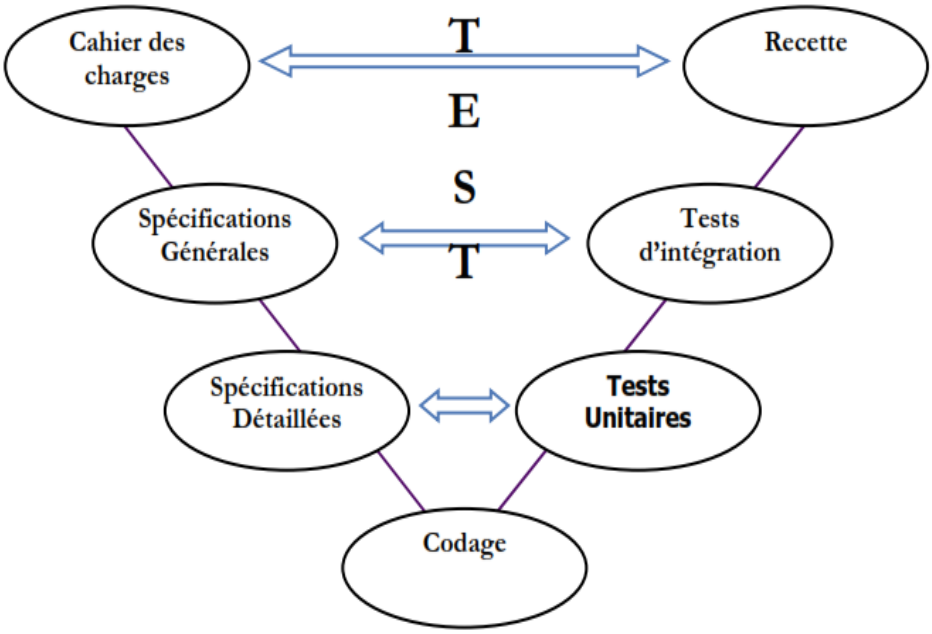


FIGURE III. 1 *CYCLE DE DEVELOPPEMENT CLASSIQUE D'UN LOGICIEL*

La Figure III.1 décrit le cycle de vie standard de développement d'un logiciel, aussi appelé cycle en V. À chaque étape du développement correspond une phase de test. Avec la méthode B, dont le cycle de développement est décrit Figure II.2, il ne reste que les tests de conformité entre le code obtenu et le cahier des charges, les autres tests étant assurés par la preuve.

La notion de base de la méthode B est la *machine abstraite*. Celle-ci encapsule des données et des opérations sur ces données. Une *machine abstraite* peut être vue comme un ensemble de données et de services permettant d'accéder à ces données. Notons qu'une spécification B peut être constituée de plusieurs machines abstraites. Les principaux éléments composant une machine abstraite sont les variables, l'invariant et les opérations :

- Les variables représentent les données encapsulées par la machine. Celles-ci ne peuvent être modifiées que par l'intermédiaire des opérations définies dans la machine ;
- L'invariant consiste en une série de prédicats définissant le type des variables, ainsi que les différentes propriétés devant être garanties par la machine ;
- Les opérations définissent les services proposés par la machine. Il peut leur être associé des pré-conditions, définissant les conditions devant être remplies pour pouvoir appeler l'opération.

Différents concepts sont associés à la spécification de ces composants, les principaux étant les suivants :

- La théorie des ensembles. La théorie des ensembles est principalement utilisée pour définir les données manipulées par une machine ;
- La logique des prédicats. Celle-ci est utilisée dans l'invariant pour décrire les propriétés devant être conservées par la machine, ainsi que pour préciser les pré-conditions nécessaires à l'appel d'une opération ;
- Les substitutions généralisées. C'est le formalisme utilisé pour la définition des opérations fournies par la machine.

La méthode B a pour avantage de reposer sur des concepts simples aisément représentables. Cela permet une compréhension plus rapide des modèles. Ainsi, le langage B, par sa définition non ambiguë, permet de se mettre d'accord sur des spécifications, puis de les utiliser pour obtenir du code. Cependant, l'obtention du code n'est pas aussi immédiate. En effet, la Figure II.2 montre qu'après une phase de spécification formelle, nous pouvons dériver une implémentation formelle. Cette dernière est obtenue grâce à une succession de raffinements. En effet, les machines abstraites peuvent être raffinées afin d'ajouter, à chaque niveau de raffinement, de nouveaux détails

permettant d'expliciter l'algorithme choisi et la manière d'opérer. Afin d'assurer la cohérence de ces raffinements avec la machine abstraite d'origine, des lemmes sont générés à chaque étape du raffinement. Ces lemmes représentent les propriétés qui doivent être préservées après chaque raffinement et sont dépendantes du raffinement précédent. Pour valider le nouveau raffinement, il faut apporter la preuve de chacun de ses lemmes. Notons que ces lemmes sont également appelés *Proof Obligation (PO).* Une fois tous les détails ajoutés, le dernier stade du développement B est atteint. Il s'agit de l'implémentation formelle. Cette implémentation correspond au dernier raffinement. Il est écrit en B0, un sous-ensemble de B proche d'un langage de programmation classique comme C. Le code correspondant à cette implémentation peut alors être généré automatiquement à l'aide d'un traducteur.

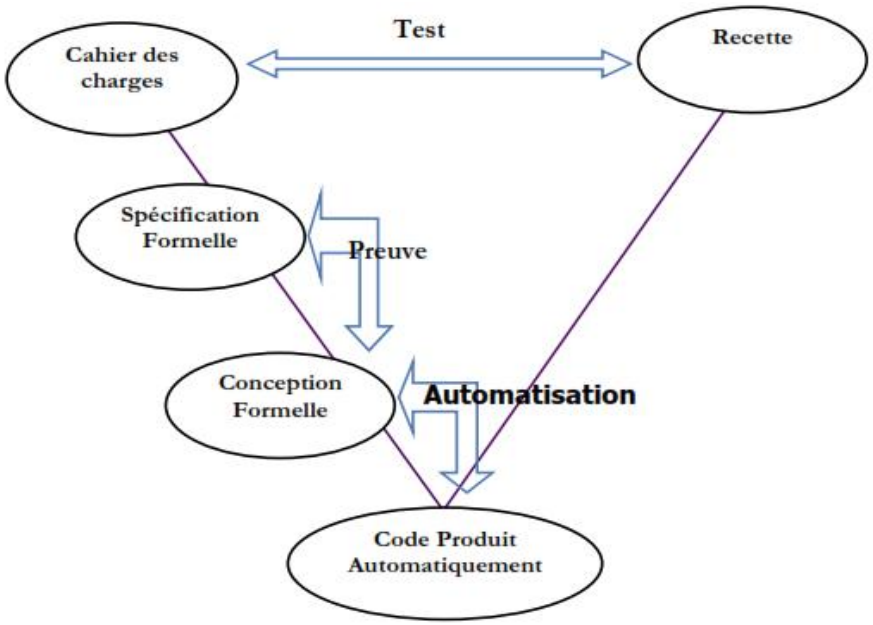


FIGURE III. 2 ***CYCLE DE DEVELOPPEMENT D'UN LOGICIEL DEVELOPPE AVEC LA METHODE B***

III.1.3. Principe de la méthode B

La méthode B est une méthode formelle couvrant toutes les phases d'un cycle de développement. En effet, les modèles B expriment différents niveaux d'abstraction allant de la phase de conception préliminaire jusqu'à la phase de codage.

Le passage d'une phase à une autre dans un processus de développement en B correspond à un incrément de spécifications en cours de développement. L'aspect caractéristique de la méthode B est qu'elle est basée sur des modèles mathématiques théories et notations (théorie des ensembles de Zermelo-Fränkel avec l'axiome du choix, le concept de substitution généralisée) [113] [114] et donc peut produire la preuve formelle que la mise en œuvre d'un programme répond aux exigences. Les

exigences sont identifiées et intégrés dans le processus de spécification. Le développement d'un projet selon la méthode B comporte deux activités étroitement liées : l'écriture de textes formels et la preuve de ces mêmes textes.

L'activité d'écriture consiste à rédiger les spécifications formelles de machines abstraites à l'aide d'un formalisme mathématique de haut niveau. Ainsi, une spécification B comporte des données (qui peuvent être exprimées entre autres par des entiers, des booléens, des ensembles, des relations, des fonctions ou des suites), des propriétés invariantes portant sur ces données (exprimées à l'aide de la logique des prédicats du premier ordre), et enfin des services permettant d'initialiser puis de faire évoluer ces données (les transformations de ces données sont exprimées à l'aide de substitutions). L'activité de preuve d'une spécification B consiste alors à réaliser un certain nombre de démonstrations afin de prouver l'établissement et la conservation des propriétés invariantes en question. La génération des assertions à démontrer est complètement systématique. Elle s'appuie notamment sur la transformation de prédicats par des substitutions.

Le développement d'une machine abstraite se poursuit par une extension de l'activité d'écriture lors d'étapes successives de raffinement. Raffiner une spécification consiste à la reformuler en une expression de plus en plus concrète, mais aussi à l'enrichir.

L'activité de preuve concernant les raffinements consiste également à réaliser un certain nombre de vérifications statiques et à prouver que le raffinement constitue bien une reformulation valide de la spécification. Le dernier niveau de raffinement d'une machine abstraite se nomme l'implantation [115].

L'environnement de la méthode B englobe des outils utilisables et très utiles pour élaborer des spécifications des systèmes. La méthode B permet de développer des services dans un processus par étapes et à combiner des services. B est un cadre flexible qui est extensible pour gérer les contraintes d'équité, mais des travaux complémentaires sont nécessaires [116].

1. Notion de la machine abstraite

La machine abstraite est un composant B qui décrit le premier niveau d'un développement. Elle peut être vue comme une structure mathématique qui a un certain état ainsi que des propriétés associées. Sa forme est donnée dans la Figure III.3

```
MACHINE
    M(X, u)
CONSTRAINTS
    C                          /* spécification des paramètres */
SETS
    S ;                        /* ensembles donnés */
    T = {a, b}                 /* ensembles énumérés */
CONSTANTS
    c                          /* liste de constantes (concrètes) */
PROPERTIES
    R                          /* spécification des constantes */
VARIABLES
    x                          /* liste de variables (abstraites) */
INVARIANT
    I                          /* spécification des variables */
ASSERTIONS
    J_1; ...; J_n              /* liste de prédicats d'assertions */
INITIALISATION
    U                          /* substitution d'initialisation */
OPERATIONS
    r ⟵ nom_op(p) = PRE P THEN K END ;
    ...
END
```

FIGURE III. 3 *FORME GENERALE D'UNE MACHINE ABSTRAITE*

2. La spécification B

Une spécification B [114] est structurée en un ensemble de machines. La partie statique d'une machine B est la définition d'espace des états qui apparaît dans les clauses VARIABLES et INVARIANTS. La première clause énumère les composants d'états et la seconde définit des restrictions sur les valeurs possibles que ces composants peuvent prendre. L'état de la machine ne peut être modifié que par des opérations. Le langage permettant d'exprimer cette partie dynamique est un langage de substitutions généralisées. Une opération peut posséder également des paramètres d'entrée et de sortie. Les opérations B peuvent être appelées par des composants extérieurs.

3. Le raffinement en B

Le mécanisme de raffinement en B classique consiste à reformuler successivement les variables et les opérations de la machine abstraite, afin d'arriver finalement à un module qui constitue le programme à implémenter. Les étapes intermédiaires de reformulation sont appelées les raffinements et le dernier niveau de raffinement est appelé l'implémentation. Lors des phases initiales de spécification, les instructions des opérations utilisent des préconditions et de l'indéterminisme. Le raffinement fait que ces préconditions deviennent de plus en plus larges (faibles) et les instructions de plus en plus déterministes. Les machines B sont étiquetées selon leur niveau d'abstraction, du niveau le plus abstrait jusqu'au niveau le plus concret. Ce mécanisme de raffinement permet de préserver des propriétés du système déjà prouvées dans des modèles de plus haut niveau. La structure d'un raffinement B est donnée dans la Figure III.4.

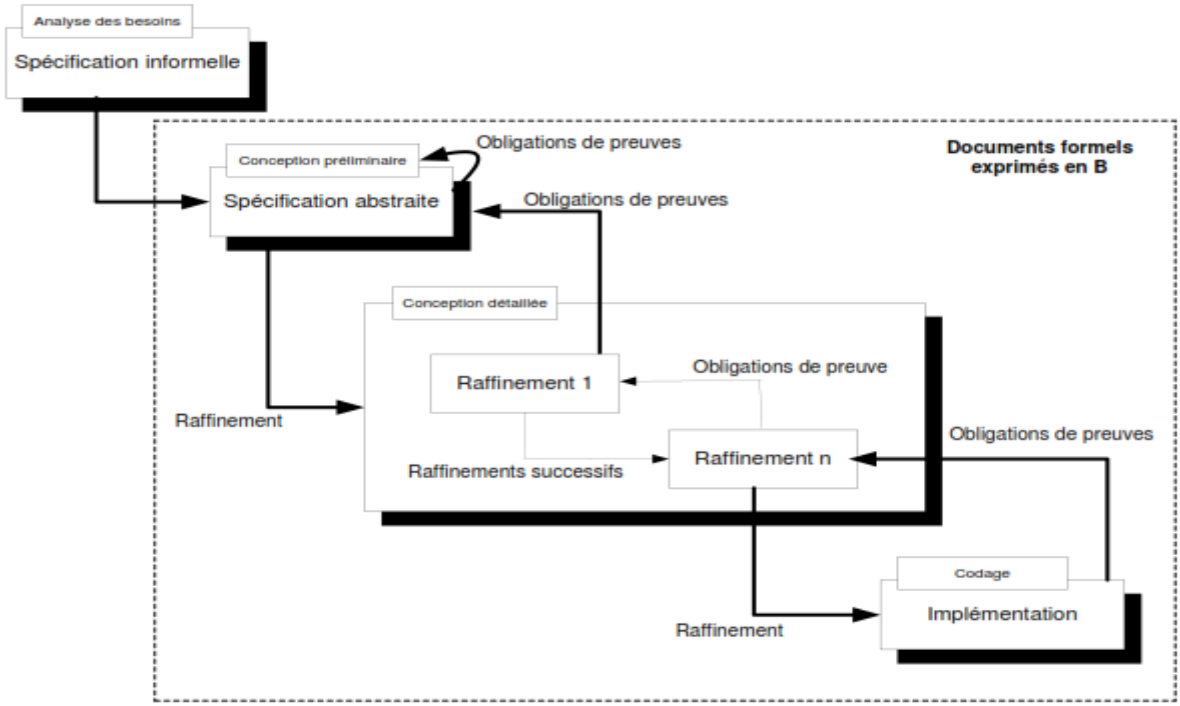


FIGURE III. 4 *PROCESSUS DE DEVELOPPEMENT EN B* [107]

4. L'Atelier B

L'Atelier B est un outil capable de prouver la consistance de chaque composant de B, ainsi que les relations entre eux. Après une spécification formelle du système sous forme d'une machine abstraite, et après chaque raffinement, l'outil Atelier B dispose aussi d'un outil de preuve interactif, appelé le prouveur B. Il génère et prouve les obligations de preuves afin que le système développé soit efficace et juste. Après cette étape, on se trouvera dans l'un des deux situations ci-dessous

- Soit la preuve demandée sera exécutée avec succès par l'Atelier B et donc la fin de preuve.
- Soit la tentative de preuve échoue et donc il faut résoudre les problèmes manuellement à l'aide des hypothèses proposée par l'assistant prouveur.

Une fois les obligations de preuves résolues, il ne reste plus qu'appliquer le raffinement des machines et recommencer la preuve, le nombre de raffinements n'est pas limité. Le processus sera donc répété jusqu'à ce que l'on arrive à une implémentation, qui gère par la suite une application exécutable avec un maximum de confiance [117].

III.2. EVOLUTION DU B VERS EVENT-B

La méthode Event-B [114] est une évolution de la méthode B, dont l'objectif est de modéliser des systèmes fermés. Un système fermé est un système modélisé avec l'ensemble de toutes les

interactions avec son environnement. Il n'y a donc plus besoin de modéliser les entrées ou sorties pour communiquer avec l'environnement. Pour cela, les opérations B sont remplacées par des événements en Event-B. Contrairement aux opérations B qui sont appelées par des composants, les événements Event-B se déclenchent spontanément si une condition (appelée garde) devient vraie. Contrairement au B classique, Event-B offre également la possibilité d'exprimer certaines contraintes dynamiques telles que des contraintes de vivacité (liveness). Ces points font que le B classique est mieux approprié pour le développement de logiciels et Event- B pour le développement de systèmes [117]. Les principales différences entre Event-B [114] et B classique sont les suivantes :

- la notion d'opération est remplacée par la notion d'événement. Un événement a une garde (sans précondition), et peut être déclenché si la garde est vraie.
- le remplacement des notions de structuration de machines (USES, SEES, INCLUDES, IMPORTS, etc.) par la seule notion de contexte, qui regroupe les constantes et les ensembles de bases, ainsi que des axiomes associés.
- la disparition de certains types de données et leurs opérateurs associés, notamment les séquences (seq), les structures (struct) et les arbres (btree).
- la disparition de certaines substitutions. Chaque événement en Event-B a en fait une forme très simple. La forme la plus compliquée peut-être exprimée par un seul ANY contenant des assignations (déterministes et non déterministes) parallèles. Les variables du ANY sont les « paramètres » de l'événement. Il existe deux formes simplifiées, une pour des événements sans paramètre et une pour des événements sans paramètre et sans garde.
- une notion adaptée du raffinement, permettant l'introduction de nouveaux événements pour séparer un événement en plusieurs événements dans la machine raffinée, et inversement, de fusionner plusieurs événements.

III.3. STRUCTURE D'UN MODÈLE EVENT-B

Un modèle Event-B est décomposé en deux parties : le contexte qui contient la partie statique du modèle et la machine qui contient la partie dynamique du modèle. Cette séparation permet d'indiquer à une machine donnée les contextes qu'elle « voit ». Un modèle Event-B peut contenir des contextes seulement, des machines seulement ou les deux. Dans le premier cas, le modèle représente une structure mathématique pure. Le deuxième cas représente quant à elle un modèle non paramétré. Le dernier cas représente un modèle paramétré par les contextes. La figure suivante présente les deux composants avec leurs différentes clauses telles qu'elles sont déclarées dans la

plateforme RODIN [118].

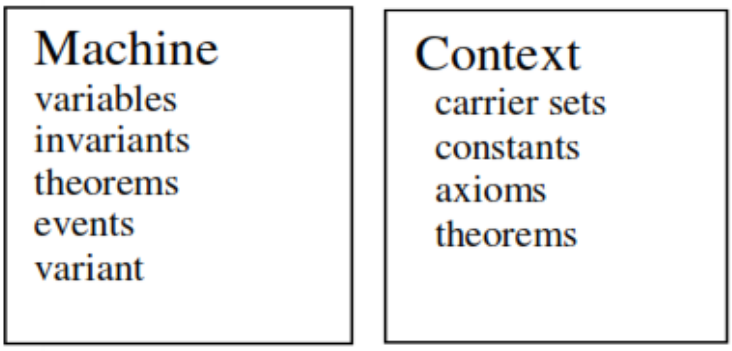


FIGURE III. 5 *MACHINE ET CONTEXTE*

Les modèles peuvent **voir** des contextes (clause SEES), pour accéder à la théorie sur laquelle un modèle est construit. Un modèle peut être **raffiné** par un autre modèle. Une différence importante vient du fait qu'un événement peut être raffiné par plusieurs événements (*découpage des événements*), et inversement, un événement peut en raffiner plusieurs (*fusions des événements*). Une fusion de deux événements $E1$ et $E2$ est équivalente au choix borné des deux événements $E1|\ E2$. Un contexte peut être **étendu** par un autre contexte. Le raffinement de contexte est simplement une *extension* des déclarations du premier contexte. Un modèle raffiné $M2$ peut voir le raffinement $C2$ d'un contexte $C1$ vu par l'abstraction $M1$ du modèle raffiné (voir figure III.5). Les relations de raffinement entre composants sont transitives (et réflexives). Les ensembles et les constantes déclarées dans un contexte sont, des **paramètres** du modèle. Il est prévu d'avoir une opération **d'instanciation** des modèles, c'est-à-dire remplacer un contexte abstrait par des ensembles et des constantes d'une théorie plus spécifique.

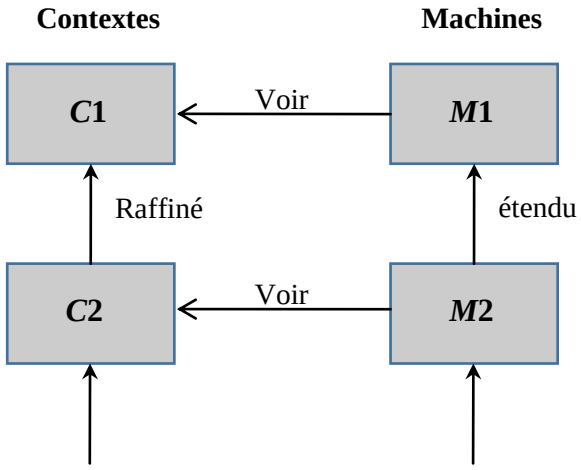


FIGURE III. 6 *RELATION ENTRE COMPOSANTS D'UN MODELE EVENT-B*

III.3.1. Contexte

On peut considérer le contexte comme la partie statique du modèle contenant plusieurs clauses.

```
CONTEXT ctxtN
   EXTENDS ctxt1, ctxt2, …
   SETS S1, S2, …
   CONSTANTS C1, C2, …
   AXIOMS A1, A2, …
   PROPERTIES P1, P2, …
END
```

FIGURE III. 7 *LES CLAUSES POSSEDANT UN CONTEXTE*

La Figure III.7 décrit les clauses du contexte ; à cet effet nous distinguons :

- La clause CONTEXT représente le nom du composant qui devrait être unique dans un modèle.
- La clause EXTENDS déclare la liste des contextes qu'étend le contexte décrit (*ctxt1, ctxt2*...).

Un contexte peut étendre un autre contexte en rajoutant de nouvelles constantes et de nouvelles propriétés.

- La clause SETS définit les ensembles porteurs du modèle (*s1, s2…*). Ces ensembles non vides servent à typer le reste des entités du modèle.
- La clause CONSTANTS contient la liste des constantes utilisées par le modèle (*c1, c2…*).
- La clause AXIOMS définit les propriétés liées aux constantes et notamment leurs types (*A1, A2…*).
- La clause THEOREMS exprime des propriétés qui peuvent être déduites à partir des propriétés présentées dans la clause AXIOMS (*P1, P2…*)

III.3.2. La machine abstraite

```
MACHINE machN
   REFINES machM
   SEES ctxt1, ctxt2,…
   VARIABLES v1, v2, …
   INVARIANTS I1, I2, …
   THEOREMS T1, T2, …
   EVENT E1, E2, …
END
```

FIGURE III. 8 *LES CLAUSES POSSEDANT UNE MACHINE*

La machine est la partie dynamique du modèle et elle est constituée de plusieurs clauses :

- La clause MACHINE représente le nom du composant (mahN) qui devrait être unique dans un modèle.
- La clause REFINES déclare le nom de la machine raffinée (machM) par la machine décrite.
- La clause SEES spécifie la liste des contextes « vus » par la machine (ctxt1, ctxt…). Dans ce cas, la machine peut utiliser les constantes et les propriétés figurant dans les contextes.
- La clause VARIABLES contient la liste des variables du modèle (v1, v2…).
- La clause INVARIANTS définit les propriétés d'invariance du modèle telles que des informations sur les types des variables et des propriétés de sûreté (*I*1, *I*2…).
- La clause THEOREMS exprime des propriétés qui peuvent être déduites des propriétés d'invariance de la machine et des propriétés présentes dans les clauses AXIOMS et THEOREMS du contexte vu (*T*1, *T*2…). En outre, cette clause peut contenir des propriétés que l'on souhaite prouver afin de les employer dans la preuve des invariants du modèle.
- La clause VARIANT définit l'expression du variant du modèle.
- La clause EVENTS contient la liste des événements qui opèrent une ou plusieurs substitutions sur la valeur des variables (*E*1, *E*2…). Parmi ces événements, l'événement INITIALISATION donne une valeur initiale aux variables.

III.3.3. Les événements

Un événement Event-B correspond à un changement d'état dénotant une transition dans le système modélisé. Un événement est essentiellement composé d'une garde (la clause WHEN) qui définit les conditions nécessaires au déclenchement de l'évènement et d'une action (la clause THEN) qui définit l'évolution des variables d'états. Notons que plusieurs gardes d'événements peuvent être vraies en même temps. Néanmoins, un seul événement peut se déclencher et le choix de cet événement est non déterministe. Un évènement peut posséder des paramètres (des variables locales) définis dans la clause ANY [117].

III.3.4. Notion du raffinement en Event B

Le raffinement en Event-B consiste à développer le système de manière incrémentale en partant d'un modèle abstrait qui constitue une spécification du système. A chaque étape de raffinement, des détails du système sont rajoutés graduellement dans un modèle concret qui doit préserver la fonctionnalité et les propriétés des modèles plus abstraits. Ces détails rajoutés apparaissent dans l'état du système en ajoutant des variables, et dans le comportement en détaillant les événements de l'abstraction ou en ajoutant de nouveaux événements. Notons que les nouveaux

événements raffinent un événement particulier de l'abstraction qui est l'évènement vide (appelé skip). Des obligations de preuve sont générées à chaque étape de raffinement afin d'assurer la correction du raffinement [117].

III.4. TLA+ PAR RAPPORT À EVENT B

La logique temporelle des actions (Temporal Logic of Actions) TLA a été introduite par Leslie Lamport en 1989 [119]. TLA permet de spécifier et de raisonner au sujet des systèmes concurrents. Un système et ses propriétés sont représentés dans la même logique. Leslie Lamport a développé progressivement un langage de spécifications, appelé TLA+ étendant TLA par des notations de la théorie des ensembles et une structuration sous forme de modules [120].

Un module représente la plus petite unité de structure complète dans TLA+. Cette structure est constituée des parties suivantes :

- Définitions ;
- Hypothèses et théorèmes ;
- Import et inclusion ;
- Export ;

Les majeures différences entre des spécifications en Event B et en TLA+ sont les suivantes :

- Une spécification TLA+ est une formule temporelle, alors que B ne permet pas d'exprimer des formules temporelles.
- A la différence de B, TLA+ n'est pas typé. La correction de type des variables d'une spécification TLA+ Spec est une propriété d'invariance affirmant que tout état atteint pendant toute exécution satisfaisant Spec, une variable d'état est un élément d'un ensemble approprié (son type). Nous trouvons le type erreurs en vérifiant cette propriété d'invariance. Le fait qu'il est non typé, TLA+ est plus expressif que B.
- TLA+ et B manipulent les ensembles et les fonctions mais ils utilisent différents opérateurs pour les décrire. Les relations sont des constructions B qu'on ne retrouve pas dans TLA+. Une spécification B utilise des fonctions qui sont des relations particulières. On peut distinguer les fonctions, les injections, les surjections et les bijections (qui peuvent être totales ou partielles) tandis qu'une spécification TLA+ ne dispose que des enregistrements et des fonctions totales.
- TLA+ peut être utilisé pour spécifier des propriétés de sûreté et de vivacité. B ne spécifie que des propriétés de sûreté [107].

III.5. PLATEFORME RODIN

III.5.1. Description de la plateforme

Un grand atout d'Event-B est la plate-forme RODIN [121], basée sur Eclipse. RODIN stocke les modèles dans une base de données et fournit des nouveaux prouveurs puissants, qui peuvent être manipulés à l'aide d'une interface graphique. Un autre aspect intéressant est la possibilité d'étendre RODIN à l'aide de «plug-ins» permettant notamment de :

- Prouver les modèles en utilisant les prouveurs de l'Atelier B. il dispose généralement de son propre prouveur, appelé NewPP.
- Vérifier les modèles par les techniques du model checking en utilisant ProB.
- Animer les modèles en utilisant Anim B.

L'intérêt de RODIN est qu'elle est une plateforme extensible. D'ailleurs, elle est toujours en cours d'amélioration et d'extension par d'autres «plug-ins» [121]**.**

III.5.2. Obligations de preuve

Afin de garantir la correction de notre modèle, il est indispensable de le prouver. Pour y parvenir, le générateur d'obligations de preuve génère automatiquement des obligations de preuve. Une obligation de preuve définit ce que doit être prouvé pour un modèle. Les obligations de preuve sont de la forme H => G. Le but G est à démontrer en partant de l'ensemble H des hypothèses. Les règles des obligations de preuve sont au nombre de onze, et pour chacune le générateur d'obligations de preuve génère une forme spécifique [122]**.** Ainsi, nous avons un événement **evt**, un axiome **axm**, un théorème **thm**, un invariant **inv**, une garde **grd**, une action **act**, un variant ou une witness**x**. Les règles d'obligation de preuve pouvant être générées pour ces éléments sont les suivantes :

1) ***INV*** : règle de préservation de l'invariant qui assure que chaque invariant dans une machine donnée est préservé par tous les événements. Elle est de la forme «*evt/inv/INV* ».
2) ***FIS :*** assure qu'une action non déterministe est faisable. La forme est « *evt/act/FIS* ».
3) ***GRD :*** renforcement des gardes abstraits ; les gardes des événements concrets sont plus fortes que celles des abstractions. Elle est de la forme « *evt/grd/GRD* ».
4) ***MRG :*** la garde d'un événement concret qui fusionne deux événements abstraits est plus forte que la disjonction (ou logique) des gardes de ces deux événements abstraits. Le nom de cette règle est « *evt/MRG* ».

5) ***SIM :*** chaque action dans un événement abstrait est correctement simulée dans le raffinement correspondant. Autrement dit, l'exécution d'un événement concret n'est pas en contradiction avec son abstraction. Le nom est « *evt/act/SIM* ».
6) ***NAT :*** sous une condition que les gardes d'un événement convergent ou anticipé sont vérifiées, le variant numérique proposé est un *entier naturel*. Son nom est « *evt/NAT* ».
7) ***FIN :*** dans une condition où les gardes d'un événement convergent ou anticipé sont vérifiées, l'ensemble variant proposé est un ensemble *fini*. Son nom est « *evt/FIN* ».
8) ***VAR*** : chaque événement convergeant diminue le variant numérique proposé ou l'ensemble variant proposé. De plus, chaque événement anticipé n'augmente pas le variant numérique proposé ou l'ensemble variant proposé. Son nom est « evt/VAR ».
9) ***WFIS*** : chaque témoin (witness) proposé dans la clause WITH (si elle existe) d'un événement concret existe vraiment. Son nom est « evt/x/WFIS ».
10) ***THM*** : un théorème écrit dans une machine ou dans un contexte est vraiment prouvable. Le nom de telle règle est « thm/THM ».
11) ***WD*** : règle de bonne définition des axiomes, théorèmes, invariants, gardes, actions, variant et witness. Selon la nature de l'élément, les noms pour cette règle sont respectivement « axm/WD », « thm/WD », « inv/WD », « grd/WD », « act/WD», «V/WD» ou «evt/x/WWD».

1. Génération d'obligations de preuve au sein de la plateforme Rodin

Il est extrêmement important de savoir comment Rodin construit et essaye de résoudre les obligations de preuve associées à des machines abstraites et leurs raffinements. Cela nous permet de :

- Concevoir des machines et des invariants correctes.
- Détecter les erreurs de conception
- En cas d'une preuve non-automatique, il permet d'interagir avec l'outil.

Les obligations de Preuve sont générées automatiquement par un outil de la plate-forme Rodin appelé le générateur d'obligations de preuve. Plus précisément :

- Cet outil accomplit une vérification statique y compris une analyse lexicale, une analyse syntaxique et une vérification de type des contextes et des machines.
- Il décide alors de ce qui doit être prouvé.
- Les résultats sont des séquents différents, qui sont transmises aux prouveurs effectuant des preuves automatiques ou interactives.

Ce processus est illustré dans la figure 7.

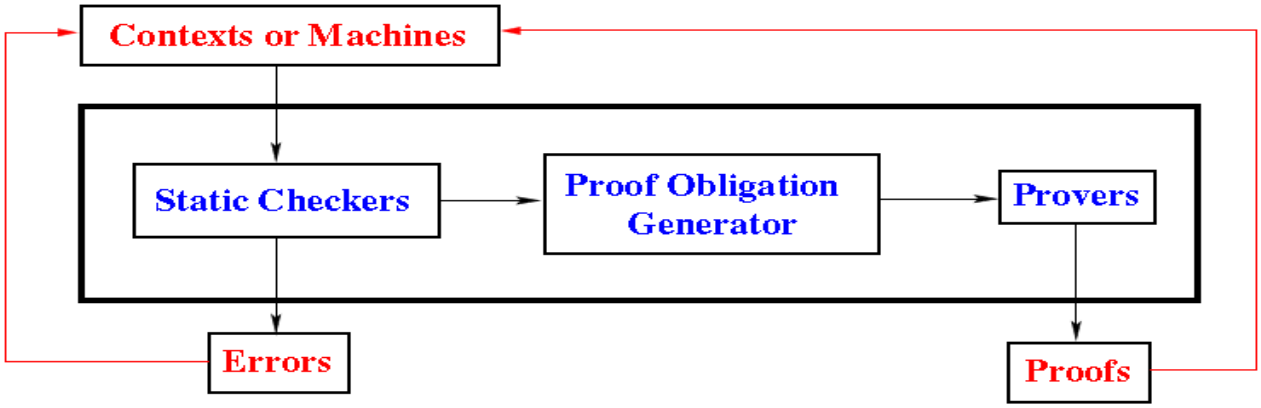


FIGURE III. 9 *OUTILS DU NOYAU DE LA PLATEFORME RODIN* [123]

Concernant la correction de preuve en Event-B un séquent est de la forme H ⊢ G où H est un ensemble fini de prédicats appelés hypothèses, et G est un prédicat unique appelé objectif (but). Un séquent signifie essentiellement que l'objectif devrait être une conséquence logique des hypothèses. Les preuves de séquents sont effectuées en utilisant des règles d'inférence. Une règle d'inférence signifie que si nous pouvons prouver tous les séquents dans A (antécédent), alors C (conséquent) a également été prouvé. La preuve d'un séquent peut donc être considérée comme un arbre fini. Les enfants d'un nœud sont les séquents dans l'antécédent de sa règle. Pour générer effectivement une obligation de preuve, nous avons besoin de trouver un arbre de preuve dont le nœud racine est étiqueté avec l'obligation de preuve [124]**.**

III.6. ATELIER B

L'Atelier B est un outil capable de prouver la consistance de chaque composant de B, ainsi que les relations entre eux. Après une spécification formelle du système sous forme d'une machine abstraite et après chaque raffinement, l'outil Atelier B dispose aussi d'un outil de preuve interactif, appelé le prouveur B. Il génère et prouve les obligations de preuve afin que le système développé soit efficace et juste. Après cette étape, on se trouvera dans l'une des deux situations :

- Soit la preuve demandée sera exécutée avec succès par l'Atelier B et donc la fin de preuve.
- Soit la tentative de preuve échoue et donc il faut résoudre les problèmes manuellement à l'aide des hypothèses proposée par l'assistant prouveur.

III.7. LE « PLUG-IN » PROB

L'outil ProB appelé aussi prouver permet l'animation et le *model-checking* des spécifications B [125], Event B, Z et même TLA+. Le ProB est entièrement automatique qui permet de visualiser le comportement dynamique d'une machine abstraite et on peut systématiquement explorer tous les états accessibles d'une machine Event-B pour vérifier des propriétés temporelles. ProB contient deux sous-systèmes importants :

- Un noyau qui traite les types de données de B (comme les ensembles et les relations), et qui implémente les opérations sur ces données (par exemple, le calcul du domaine d'une relation).
- Un interpréteur pour les substitutions, prédicats et expressions d'une machine B.

CONCLUSION

Dans ce chapitre, nous avons exposé dans un premier temps l'avantage d'utiliser les méthodes formelles afin de vérifier les systèmes. La vérification de modèle ou le "model checking" [126] est une technique de vérification formelle qui est développée afin de permettre une vérification automatique et exhaustive de systèmes. Elle s'applique à une large classe de systèmes : protocoles de communication, réseaux téléphoniques, industrie manufacturière, systèmes embarqués, etc.
Cette technique est basée sur une description du système sous forme d'un automate. La génération consiste à générer des états possibles, d'examiner tous ces états, et d'identifier ceux qui sont en contradiction avec la propriété vérifiée. Ce qui va être appliqué dans notre cas pour une vérification d'une conception d'un réseau sur puce dynamique ou adaptatif avec langage event-B à l'aide d'outil de vérification formelle RODIN.
Nous avons présenté la méthode B et son évolution. Nous avons introduit le fonctionnement de la méthode Event-B ainsi que la plateforme Rodin et son prouveur robuste ProB et tous les avantages que l'on peut tirer d'une vérification des systèmes électroniques par la méthode B. Nous avons consacré la dernière section pour détailler le mécanisme interne de génération des obligations de preuve au sein de la plateforme Rodin qui garantit la consistance du modèle, la correction du raffinement, la préservation de l'invariant, la décrémentation du variant, et la vérification du respect de la sémantique Event-B.

CHAPITRE IV :

APPLICATION ET VALIDATION

INTRODUCTION

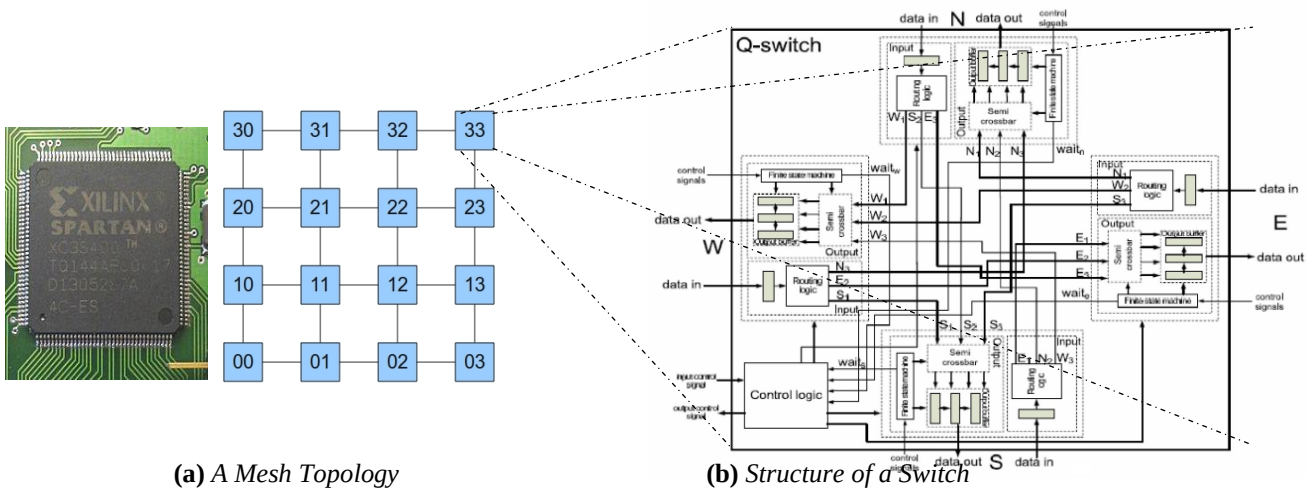
La topologie d'un réseau définit comment les routeurs sont interconnectés entre eux en utilisant les liens de réseau. Elle spécifie l'organisation physique du réseau, et elle est donc souvent modélisée par un graphe. Comme pour les réseaux informatiques, de nombreuses topologies sont envisageables pour construire un NoC. Dans notre cas d'étude, nous nous focalisons sur un routeur et réseau spécifique NoC permettant une sûreté de fonctionnement dans un contexte de fonctionnement adaptatif.

IV.1. MODÈLE D'UN Q-SWITCH

IV.1.1. Rôle d'un Q-Switch

Nous basons nos travaux sur une architecture de réseau Q-Switch basée sur une topologie de réseau Mesh à transmission de routage par paquets [127], [128], [81]. Le routage par paquets [127], [129] réduit les besoins en temps de latence et de tampon dans les routeurs. Le circuit de communication [81] est évité dans cette architecture en raison du coût élevé de l'établissement et la gestion des connexions du circuit. De même, les techniques de stockage et de routage avant sont également évitées, car elles peuvent entraîner les exigences de tampons élevées et par conséquent de lourdes peines dans la zone de silicium du routeur [81]. Le réseau est sans perte, et les liens sont supposés fiables de sorte qu'aucune retransmission ne soit requise. Les paquets transitent sur le réseau selon le plus court chemin, ce qui réduit la dissipation de puissance et maximise l'utilisation des ressources réseau. L'architecture a été intégré au sein d'une plate-forme à base de technologie FPGA selon au développement de processus de conception, et les mesures des performances du reseau ont été évaluées à partir d'une modélisation et simulation à l'aide du langage *VHDL* [130] et logiciel *Xilinx* [131].

Notre réseau comprend des routeurs interconnectés par liaisons *point à point* et la topologie peut varier en fonction des besoins du système, la taille du module et du placement. L'architecture de ce réseau, réalisée au sein de l'équipe ASEC du laboratoire LCOMS-Metz, a pour rôle de faire passer les messages entre un maximum de 4 éléments de traitements (couples routeurs-IP). Elle a une topologie maillée (voir Figure IV.1).



(a) *A Mesh Topology* **(b)** *Structure of a Switch*

FIGURE IV. 1 *ARCHITECTURE D'UN Q-SWITCH* [132]

IV.1.2. Les composants d'un Q-Switch

La structure d'un Switch (voir la figure IV.1) est la suivante :

- Un registre d'entrée par direction : Chaque paquet entrant est stocké dans un registre d'entrée. Un composant spécifique, appelé logique de routage, calcule la prochaine direction du paquet (N, E, S ou W; voir Fig.IV.1). Un maximum de trois paquets est autorisé par la direction. Les paquets sont transmis à la logique de sortie. Une politique d'arbitrage peut être adoptée pour définir les priorités entre paquets stockés dans les registres d'entrée d'un routeur, selon la prochaine direction des paquets. Cette politique est fondée sur les règles du droit de priorité (voir Figure IV.2).

- Une logique d'interconnexion qui est composée d'un demi-crossbar, un buffer de sortie et une machine d'états finis. L'interconnexion semi-crossbar se compose de trois entrées et quatre sorties. Les paquets entrants sont stockés dans les entrées en fonction des priorités. Si les voisins d'un commutateur ne sont pas occupés, la première sortie du semi-crossbar est l'un des routeurs adjacents.

- Une logique de sortie est qui constituée de registres. Ces registres stockent les paquets dans le cas où plusieurs paquets ont la même direction de sortie. Cette logique sortie est également utilisé lorsque la sortie sélectionnée (direction) est occupé (signal d'occ). Un maximum de trois messages peuvent être stockés dans une mémoire tampon de sortie. La machine d'état finie (FSM) gère les signaux de commande et son rôle est aussi d'éviter les collisions des paquets. En outre, la machine d'état finie (FSM) fournit une logique centrale avec des informations sur les états de routeur adjacents (de la situation d'attente, sur le signal, etc.).

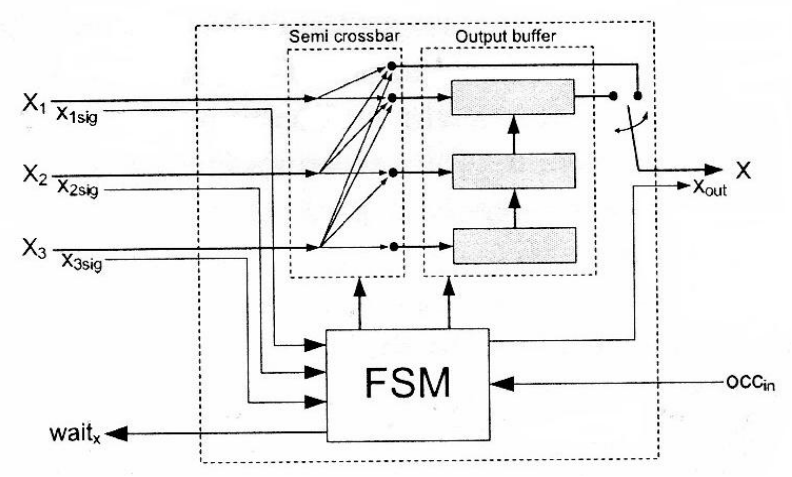


FIGURE IV. 2 *ARCHITECTURE DU PORT DE SORTIE DU ROUTEUR Q-SWTICH*

- Une logique de contrôle qui gère les connexions entre les ports d'entrée et de sortie d'un routeur. La logique de contrôle gère également le stockage de paquets qui ne peuvent pas être transférées à des directions suivantes, en raison de signaux d'occupation de routeurs voisins. En outre, si le routeur ne peut pas stocker plus de paquets entrants, la logique de contrôle informe les voisins (qui ont envoyé les paquets au routeur considéré) que le routeur ne peut pas accepter temporairement d'autres paquets.

Processus de routage

- L'algorithme de routage XY définit les règles d'acheminement des paquets, ainsi la source (s) et de destination (d) d'un paquet (p) être définies par des coordonnées 2D au sein du réseau (xs, ys) pour la source (s), et (xd, yd) à la destination (d).

- Le paquet (p) se déplace d'abord selon la dimension x, jusqu'à ce que xs = xd. Puis, le paquet (p) se déplace selon la dimension y, jusqu'à ce ys = m.

- Si le paquet (p) rencontre des éléments incapables de transmettre des données en dimension x, les routeurs prennent une décision temporaire de routage à dimension y.

- Il convient de noter que le réseau peut évoluer (suppression de certains liens, isolement de certains routeurs, etc.), et la transmission de données peut être perturbée. Toutefois, un mécanisme de reconfiguration assure que chaque paquet en transit trouve toujours un chemin menant à la destination du paquet. Si le paquet est stocké dans un routeur incapable de transmettre des données, la liaison entre ce routeur et la destination du paquet finira par être restaurée.

Remarque : Il peut arriver que plusieurs paquets prennent la même direction de sortie. Dans ce cas-là on prend jusqu'à 3 paquets au maximum.

Une politique d'arbitrage doit être adoptée pour la logique de routage qui gère la priorité d'envois des paquets. Cette politique est fondée sur les règles des priorités à droite. Elle est construite et intégré individuellement pour chaque port d'un routeur Q-Switch.

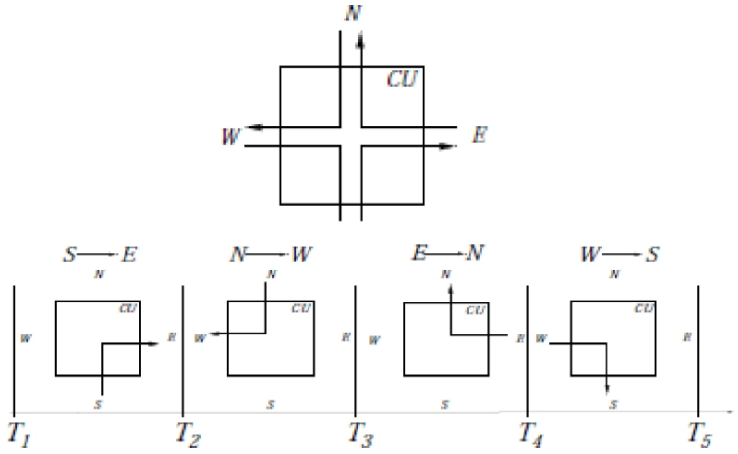
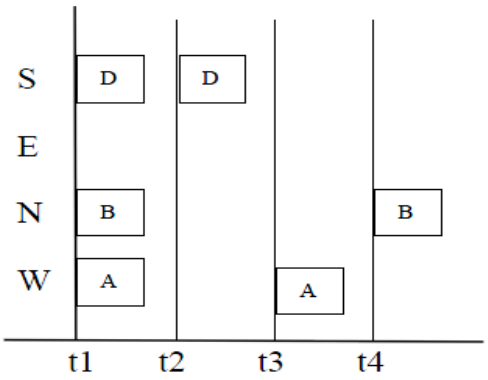
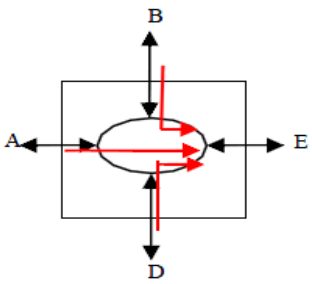


FIGURE IV. 3 *REGLES DES PRIORITES A DROITE* [132]

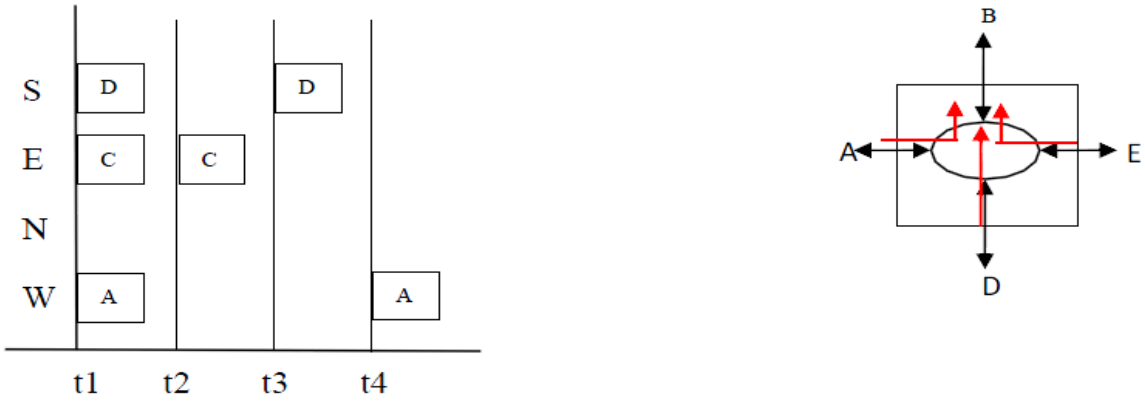
Par exemple, un port rencontre des paquets arrivants des directions nord, ouest et sud. Si nous appliquons les règles de priorité à l'emprise de ces sens, par conséquent, nous n'avons que les paquets de la direction du sud qui auront la plus haute priorité, puis ceux de l'ouest et enfin les paquets issus de la direction du nord. La même procédure est appliquée à tous les autres ports.

Des cas possibles :

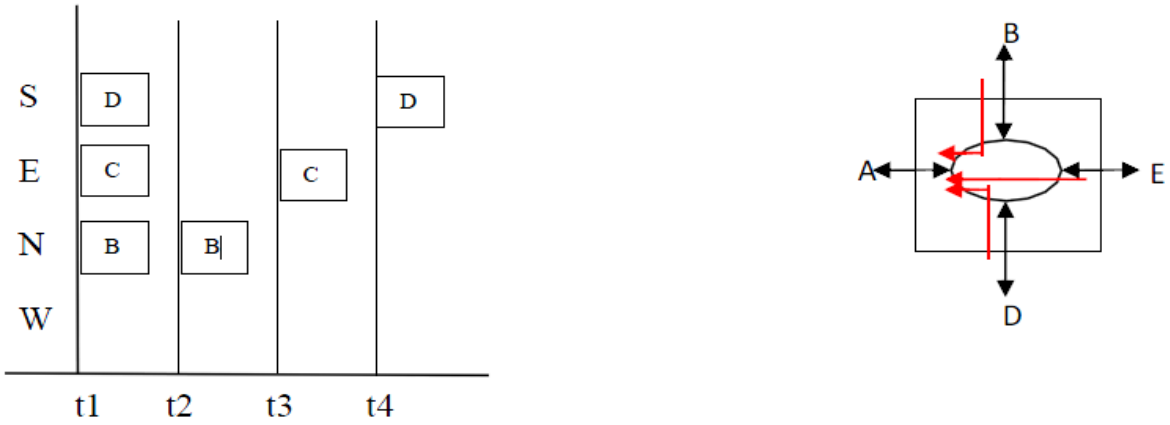




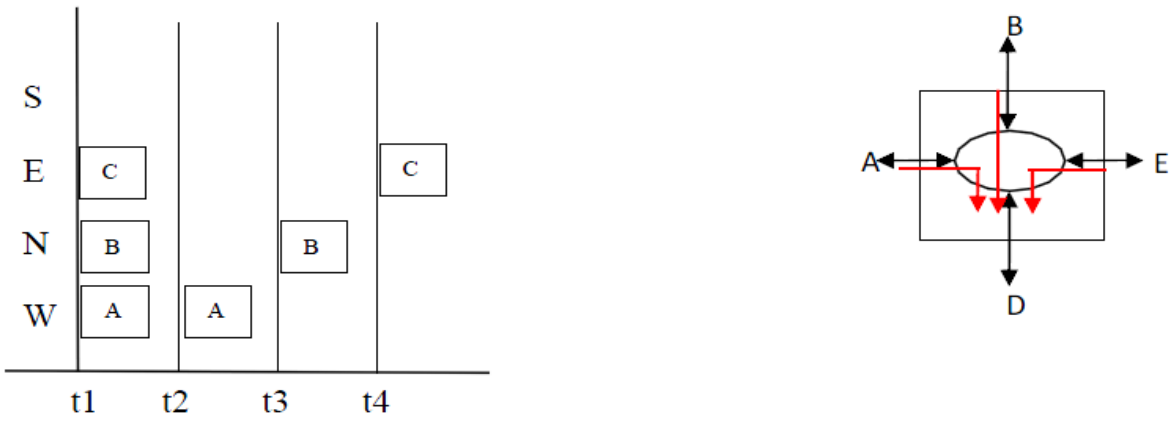
(A)



(B)



(C)



(D)

FIGURE IV. 4 ***APPLICATION DES REGLES PRIORITAIRES DANS LES DIRECTIONS POSSIBLE (A) EST, (B) NORD, (C) OUEST, (D) SUD***

IV.1.3. Algorithme de routage XY

L'algorithme de routage X-Y signifie que les paquets de données sont dirigés vers le destinateur à travers des routeurs selon l'axe X puis selon son axe Y avec la condition que le paquet ne prend pas la direction d'arrivée. Dans notre Q-Switch, si en cours de routage X un paquet

rencontre un élément de traitement, une procédure de routage adaptatif est mis en place temporairement, notamment en modifiant le routage Y.

IV.1.4. L'algorithme de routage adaptatif

Le Switch utilise un algorithme de routage basé sur l'algorithme XY [133] classique qui peut être utilisé initialement dans le réseau adaptatif car ce dernier n'est pas adapté à des situations irrégulières qui peuvent se produire lorsque les paquets sont acheminés selon l'axe X ensuite vers l'axe Y de direction du réseau. Si au cours des routages, les paquets rencontrent des blocages (défauts, modules, etc.) classiques, alors l'algorithme de routage utilisé permet leur contournement à l'aide de la logique de contrôle de chaque routeur qui distingue le type d'entité (routeur ou module de calcul) connecté à un routeur. Cet algorithme permet donc d'éviter les situations d'interblocage [133] pouvant arriver dans le QNoC et résout également l'inconvénient de l'ordre d'arrivée des paquets à un nœud du réseau. En effet, si au cours du fonctionnement du réseau, il n'y a pas de placements dynamiques entre deux modules de calcul, les chemins empruntés par les paquets envoyés à partir d'un module vers un autre module destinataire, seront identiques et de même longueur (même situation que l'algorithme XY), conservant alors l'ordre d'émission et de réception des paquets transmis. Par contre, les paquets d'un message envoyés par un module de calcul, peuvent être imbriqués à la destination avec les paquets d'autres messages envoyés par d'autres modules de calcul du réseau.

IV.1.5. Spécification formelle du QNoC

La présentation des modèles sera par niveau et le passage d'un niveau à un autre par un raffinement. La formalisation est réalisée de façon à commencer par le niveau le plus abstrait qui définit le rôle du service NoC en arrivant au niveau le plus bas, là où se trouve toute une spécification formelle qui définit le comportement détaillé de l'architecture **[149]**. Ainsi la modélisation définie de la manière suivante :

- Définir le rôle du service : faire passer un paquet dans un réseau (sending/receiving);
- Un paquet (data) passe dans un graphe d'un routeur à un autre, jusqu'à arriver à la destination ;
- Chaque Switch envoie un paquet à son voisin (nord, sud, est, west) selon la destination ;
- L'algorithme XY est appliqué pour envoyer les paquets, ainsi un paquet passe par l'axe des X, puis l'axe des Y pour arriver à la destination.

La figure IV.4 illustre les différentes étapes d'une modélisation pour aboutir un développement formel et détaillé. Il est à noter que le raffinement permet de décomposer la complexité de l'architecture NoC en œuvre de formalisation à différents niveaux d'abstraction (modélisation étape-par-étape voir Figure IV.5).

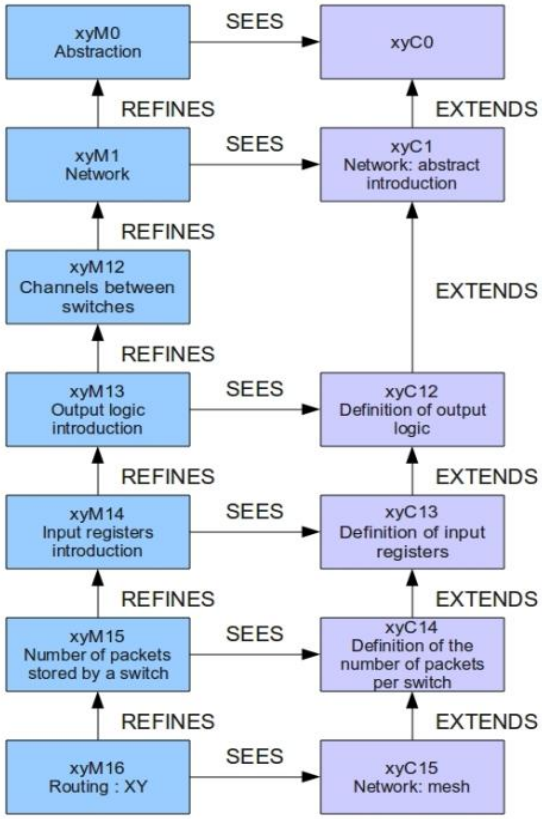


FIGURE IV. 5 ***MODELISATION ETAPE-PAR-ETAPE DE L'ARCHITECTURE NOC***

Spécification abstraite : xyM0

Le premier modèle xyM0 est une description abstraite du service offert par l'architecture NoC (voir figure IV.6). Plus précisément, l'envoi d'un paquet (p) par une source de commutation et la réception de (p) en une destination de routeur.

FIGURE IV. 6 ***NIVEAU ABSTRAIT***

Pour ce faire, un ensemble de routeurs (nœuds), un ensemble de paquets (MSG), une fonction d'acheminement à partir d'une source (src), associant les paquets et de leurs sources, une fonction d'acheminement à une destination (DST), paquets de couplage et de leurs destinations. Sont définis dans le contexte xyC0. La machine xyM0 (voit) le contenu du contexte xyC0, et avec ceux-ci,

décrit une vue abstraite de la prestation fournie par l'architecture NoC :

• Un événement SEND présente le cas de l'envoi d'un paquet (m) par sa source (s) à un routeur de destination (d).

• Un événement RECEIVE illustre la réception d'un paquet émis (m) par sa destination (d). En outre, le modèle xyM0 nous permet d'exprimer certaines propriétés et invariants selon l'expression suivante :

ran(received) ⊆ ran(sent)

Cet invariant exprime que chaque paquet reçu par un routeur de destination a été envoyé par une source de commutation.

Premier raffinement M0 (xyM1) : introduction du réseau

La machine xyM1 raffine la machine xyM0 et présente un réseau (un graphe) entre les sources et les destinations de paquets. Certaines propriétés sur le graphe sont définies dans le contexte xyC1 : graphe est non-vide, non transitive et symétrique.

FIGURE IV. 7 *MODELE RAFFINE PAR AJOUT D'UN RESEAU DANS LE MODELE M0*

Les événements de xyM0 sont raffinés par les événements suivants :

• L'événement SEND : Lorsqu'une source envoie un paquet, le paquet est placé dans le réseau.

• L'événement RECEIVE : Un paquet est reçu par sa destination, si le paquet a atteint la destination.

Les nouveaux événements sont également introduits par xyM0 :

• L'événement FORWARD (voir Figure IV.8) : dans le réseau, un paquet (p) transite d'un nœud (x) vers un autre nœud (y), jusqu'à ce que la destination (d) de paquet (p) soit atteinte.

FIGURE IV. 8 *TRANSFERT DE PAQUET (P) ENTRE LES SWITCHS*

• L'événement DISABLE : Un nœud est désactivé. Le nœud est interdit de communiquer avec ses voisins (échec, etc.). Lors de la désactivation de certains nœuds, nous nous assurons que les paquets qui transitent sur le réseau finiront par atteindre leur destination (soit après une reconfiguration du

réseau ou en laissant toujours un chemin vers des destinations disponibles).

• L'événement RELINK : ce modèle d'événements de la reconfiguration du réseau. Les nœuds mobile sont réactivés, les liens entre eux et leurs voisins sont restaurés, permettant ainsi la communication et les paquets transferts. La configuration du réseau contribue à démontrer la sécurité de la transmission de données entre une source et un commutateur de destination de l'interrupteur.

Le xyM1 machine présente également des propriétés du système : ran(received) ∩ ran(store) = ∅ Ceci démontre l'invariante un paquet (P) envoyé par une source se déplace, soit dans le réseau (store) ou est reçu par une destination.

Deuxième Raffinement (xyM12) : Introduction des Canaux

Ce deuxième raffinement décompose le cas FORWARD de la machine xyM1 en deux événements. La figure IV.9 illustre ce cas :

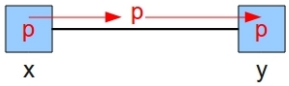


FIGURE IV. 9 ***INTRODUCTION DES CANAUX***

• Un raffinement de l'événement FORWARD représente l'adoption d'un paquet (P) à partir d'un commutateur (x) à un canal (CH), conduisant à un voisin (y).

• Un événement FROM_CHANNEL_TO_NODE est intégré et exprime le transfert d'un paquet (P) à partir d'un canal (CH) connecté à un Switch (n).

La machine xyM12 définit aussi certaines propriétés :

ran(c) ∩ ran(switch) = ∅

L'invariant exprime que chaque paquet envoyé est soit dans un canal, soit dans un Switch. Un paquet ne peut pas être envoyé simultanément dans un canal et à un Switch.

Troisième Raffinement (xyM13) : Introduction de la logique de sortie (Output Logic).

Ce raffinement nous permet d'introduire la structure d'un Switch progressivement. Nous exprimons, dans la machine xyM13, que les Switch disposent de ports de sortie (voir Figure IV 10). L'événement FOWARD abstrait est décomposé de la manière suivante :

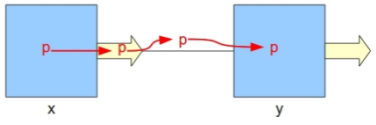


FIGURE IV. 10 ***L'AJOUT DES OUTPUT PORTS***

• Le raffinement de l'événement FORWARD ajoute le fait qu'un paquet (p), qui quitte un Switch (x) et en direction d'un voisin (y), entre initialement dans la logique de sortie (OP) de Switch (x) conduisant à (y).

• Un événement d'un nouveau modèle de transition, OUTPUT_BUFFER_TO_CHANNEL permet d'exprimer la transition du paquet (P) depuis un port de sortie (OP) à un canal (CH) menant à un Switch de destination (n). En outre, de nouvelles propriétés et les invariants sont définis dans la machine xyM13 selon les expressions suivantes :

inv1 : ran(chan) ⊆ ran(sent)

inv2 : ran(outputbuffer) ⊆ ran(sent)

inv3 : ran(outputbuffer) ∩ ran(chan) = ∅

L'inv1 invariant exprime que chaque paquet transitant dans un canal (CH) a été envoyé par une source (s) ; INV2 démontre que chaque paquet transitant dans un port de sortie (CH) a été envoyé par une source (s); INV3 présente du fait qu'un paquet est soit dans un port de sortie ou dans un canal. Le paquet ne peut pas être à la fois dans un port de sortie et un canal entre les deux commutateurs à la fois.

Quatrième Raffinement (xyM14) : Introduction de registre d'entrée

Ce raffinement (xyM14) ajoute des ports d'entrée de la structure d'un routeur (voir figure IV.11).il intègre les événements raffinés suivants :

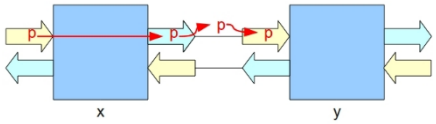


FIGURE IV. 11 ***L'AJOUT DES INPUT PORTS***

• L'événement SEND est raffiné : lorsqu'une source (s) de Switch envoie un paquet (p), le paquet (p) est mis dans un port d'entrée (IP) de l'interrupteur (s).

• Les actions décrites par l'événement abstrait FORWARD sont décomposés :
- L'événement SWITCH_CONTROL est un raffinement FORWARD, est modélise le passage d'un paquet (P), à partir d'un port d'entrée (IP) d'un Switch (x), à un orifice de sortie (OP) menant à un Switch (y).

- L'événement OUTPUT_BUFFER_TO_CHANNEL de transition présente le cas d'un paquet (P), d'un port de sortie (OP), à un canal (CH) menant à un Switch de destination (n).

- L'événement FROM_CHANNEL_TO_INPUT_BUFFER montre la transition d'un paquet (P) à partir d'un canal (CH) à un port d'entrée (IP) d'un Switch de destination (n).

La machine xyM14 machine présente également des propriétés et les invariants suivants :

inv1 : ran(inputbuffer) ⊆ ran(sent)
inv2 : ran(outputbuffer) ∩ ran(inputbuffer) = ∅
inv3 : ran(inputbuffer) ∩ ran(chan) = ∅

L'invariant exprime que chaque paquet transitant dans un port d'entrée (IP) a été envoyé par une source (s); INV2 démontre que chaque paquet transite soit dans un port de sortie (OP), soit à un port d'entrée (IP); INV3 présente le fait qu'un paquet est soit à un port d'entrée, soit dans un canal. Le paquet peut ne pas être à la fois dans un port d'entrée et un canal entre deux Switches.

Cinquième Raffinement (xyM15) : nombre de messages par Switch

Ce raffinement introduit le stockage des paquets dans un routeur. Chaque port de sortie d'un routeur peut stocker un nombre de paquets jusqu'à une limite de trois messages. Les paquets peuvent être bloqués dans un routeur, en raison de signaux "d'occupation" ou "attente" des routeurs voisins. Le SWITCH_CONTROL d'événement est raffiné, et ajoute le fait que la suite du passage d'un paquet à partir d'un port d'un Switch (x) d'entrée, à un port de sortie. Si le Switch (x) n'est pas plus occupé, il envoie un signal de libération au routeur précédent lié au port d'entrée. Un nouvel événement RECEIVE_BUFFER_CREDIT modélisé la réception d'un signal de sortie par un routeur (n).

Sixième Raffinement (xyM16) : Algorithme XY

Le dernier modèle xyM16 décrit l'architecture du réseau (graphique) selon un graphe à topologie maillée (voir Figure IV.12). Une limite numérique (nsize) est introduit pour délimiter le nombre de routeurs dans les dimensions x et y de la topologie du réseau. Le réseau est régulier de type 2D-Mesh, avec une taille (nsize _ nsize). Chaque routeur est couplé avec des coordonnées uniques (X; Y). avec 2 x [0 :: nsize 1] et y 2 [0 :: nsize 1].

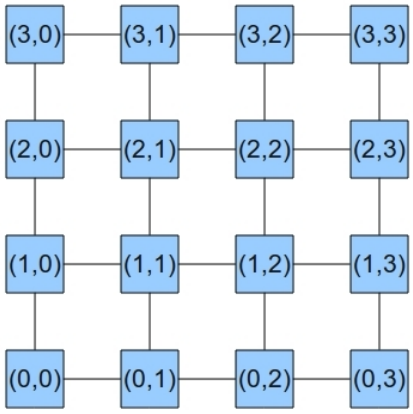


FIGURE IV. 12 *UN RESEAU MAILLE AVEC 2D-MESH*

Ce système de coordonnées permet d'être plus précis sur les voisins de chaque routeur, comme présenté dans la figure IV 12. Ce modèle donne également une description des nœuds de la structure d'un routeur (voir Figure IV.13) comme décrit ci-dessous :

- Un Switch a généralement quatre ports de sortie et quatre ports d'entrée (généralement étiqueté N, S, E et O), utilisés pour la communication avec les voisins.

- Cependant, deux autres cas sont distingués :

- les limites des Switch est dans le coin on peut avoir que deux ports de sortie et deux ports d'entrée (NE, NO, SE, SO).

- Autres limites de Switch sont identifiées lorsque l'on a trois ports de sortie et trois ports d'entrée possible pour l'acheminement (NSE, NSW).

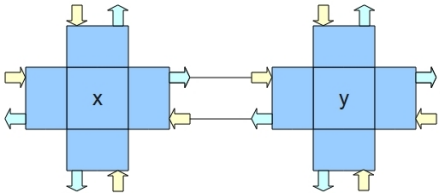


FIGURE IV. 13 *SWITCHS : STRUCTURE ET LIAISONS*

En outre, ce modèle concret introduit également l'algorithme de calcul d'itinéraire XY selon le pseudo-code suivant :

```
D: destination. Coordinates (Dx, Dy)
C: current node. Coordinates (Cx, Cy)
if (Cx > Dx) :
return W; (Case 1)
```

```
if (Cx < Dx) :
return E; (Case 2)
        if ((Cx = Dx) ∨  ((Cx > Dx) ^ W is blocked) ∨
         ((Cx < Dx) ∧ E is blocked)) :
if (Cy < Dy) :
return N; (Case 3)
if (Cy > Dy) :
return S; (Case 4)
```

Les cas de l'algorithme de routage XY sont jumelés avec des raffinements de l'événement SWITCH_CONTROL. Ainsi ; nous distinguons :

- Les modèles de SWITCH_CONTROL_LEFT de **cas 1** : un paquet (P) est transmis, à partir d'un port d'entrée d'un routeur (x), à un port de sortie, conduisant à un voisin (y) situé à W. Cet événement est déclenché si la coordonnée x de la destination (D) (du paquet (P)) est inférieure à la coordonnée x du nœud courant (X).

- Le modèle de SWITCH_CONTROL_RIGHT de **cas 2** : un paquet (P) est transmis, à partir d'un port d'entrée d'un routeur (x), à un port de sortie, conduisant à un voisin (y) situé à E. Cet événement est déclenché si la coordonnée x de la destination (D) (du paquet (P)) est supérieure à la coordonnée x du nœud courant (X).

- Le modèle de SWITCH_CONTROL_UP de **cas 3** : un paquet (P) est transmis, à partir d'un port d'entrée d'un routeur (x), à un port de sortie, conduisant à un voisin (y), situé à N. Cet événement est déclenché, soit si la coordonnées y de la destination (D) (du paquet (P)) est supérieure à la coordonnée y du nœud courant (x), soit, si la coordonnée x de la destination (d) est égale à la coordonnée x du nœud courant (X), soit si le paquet (P) ne peut pas transiter le long de l'axe des x.
- Le modèle de SWITCH_CONTROL_DOWN de **cas 4**: un paquet (P) est transmis, à partir d'un port d'entrée d'un routeur (x), à un port de sortie, conduisant à un voisin (y), situé à S. Cet événement est déclenché, soit si la coordonnée y de la destination (D) (du paquet (P)) est inférieure à la coordonnée y du nœud courant (x), soit si la coordonnée x de la destination (d) est égale à la coordonnée x du nœud courant (X), soit si le paquet (P) ne peut pas transiter le long de l'axe des x.

Résultats et discussion

Nous avons présenté un développement progressif d'une architecture de réseau sur puce, en utilisant le formalisme de l'événement B. La formalisation de l'architecture est présentée à partir

d'un niveau abstrait jusqu'à un niveau plus concret et cela de manière hiérarchique. La complexité de l'évolution est mesurée par le nombre d'obligations de preuve qui sont automatiquement/manuellement déchargées (voir tableau I).

Le prouveur ne sait pas démontrer cette obligation est fausse, selon les règles et l'ensemble de démonstrateur de connaissances mathématiques. Les commandes de preuve commandes contrôlent le contrôle automatique interactive.

On remarque dans le tableau 1 que pour le contexte et la machine xyC15 xyM14, il y a des preuves plus interactives que les automatiques. Ceci s'explique par le fait que la majorité de ces preuves interactives sont quasi-automatique. Plus précisément les preuves n'avaient pas besoin d'efforts difficiles (ni importation d'hypothèses ou en simplifiant les buts, etc.), la simple utilisation/fonctionnement de démonstrateurs (fourni par la plateforme RODIN), nous a permis de s'acquitter de ces obligations. Contrairement à la vérification par simulation seulement, notre travail fournit un cadre pour le développement de réseau sur puce sur l'architecture et l'algorithme de routage XY utilisant les propriétés essentielles de sécurité avec une preuve formelle qui affirme son exactitude.

Model	**Total**	**Auto**		**Interactive**	
xyC0	3	3	100%	0	0%
xyC1	6	6	100%	0	0%
xyC12	0	0	100%	0	0%
xyC13	0	0	100%	0	0%
xyC14	1	1	100%	0	0%
xyC15	5	0	0%	5	100%
xyM0	26	25	96.15%	1	3.85%
xyM1	38	28	73.68%	10	26.32%
xyM12	72	45	62.5%	27	37.5%
xyM13	74	37	50%	37	50%
xyM14	67	23	34.33%	44	65.67%
xyM15	24	14	58.33%	10	41.67%
xyM16	26	18	69.23%	8	30.77%
Total	342	200	58.48%	142	41.52%

TABLEAU IV. 1 *SOMMAIRE DES OBLIGATIONS DE PREUVE*

Après avoir atteint ce niveau de raffinement, plusieurs obstacles apparaissent. On a par exemple :

- Comment se comporter avec les zones défaillantes du réseau, et remarquer aussi que lorsque le paquet change de direction, c'est forcément pour prendre le plus court chemin. Il y a donc ici un risque de perte de temps, c'est ce qui est négatif pour la fiabilité du système.

- A un certain niveau de raffinement, peut-on générer du code VHDL d'une façon automatique et sur quelle base (obtenir au moins un skelton).

- Est-ce que le temps d'envoie des paquets est limité ? c'est-à-dire, si le temps actuel dépasse significativement le temps estimé, cela signifierait qu'il y a une possible situation de blocage.

IV.2. MODÈLE RKT-SWITCH

Cette deuxième architecture se base sur les mêmes principes que la première, excepté que la stratégie de routage change du Wormhole au ACK/NACK. Pour ce faire, plusieurs définitions et règles sont considérées.

IV.2.2. Définition

La zone désactivée d'un réseau est le reste du réseau n'appartenant pas à la zone activée.

Si un réseau n'a pas une zone activée (pas de nœuds défaillants ni de régions), il est entièrement désactivé. Tous les nœuds appartenant à la zone désactivée sont désactivés. S'il y a dans un réseau une zone activée formée autour d'un nœud défaillant, seuls les nœuds de routage enveloppant le nœud de routage défaillant changent de statut et deviennent activés. Les nœuds appartenant au reste du réseau ne changent pas leur mode et restent désactivés.

Un nœud désactivé achemine un paquet de données selon l'algorithme XY. Premièrement il achemine le paquet selon l'axe X, puis selon l'axe Y, jusqu'à ce que le paquet de donnée soit livré à la destination. Si le paquet arrive à la zone activée avant d'atteindre sa destination finale, les nouvelles règles de routage sont alors appliquées.

Les nœuds de routage activés n'obéissent pas aux mêmes règles de routage que les nœuds de routage désactivés. Ces règles sont décrites de la manière suivante [134] :

Règle 1 : Un nœud pair et activé ne peut pas acheminer des paquets venant de la direction Nord (North) vers la direction Est (East) et vice versa.

Règle 2 : Un nœud impair et activé ne peut pas acheminer des paquets venant de la direction Sud (South) vers la direction Ouest (West) et vice versa.

Règle 3 : Tous les nœuds activés, par défaut, ne peuvent pas acheminer des paquets venant respectivement des directions Nord et Est vers les directions Sud et Ouest.

Règle 4 : Tous les nœuds activés ne peuvent pas par défaut acheminer des paquets venant respectivement des directions Sud et Ouest vers les directions Nord et Est.

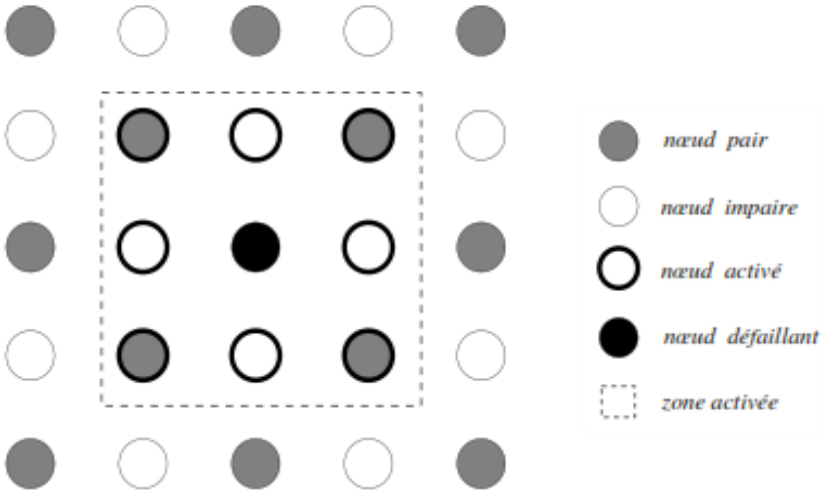


FIGURE IV. 14 ***EXEMPLE D'UNE ZONE ACTIVE***

Les cas possibles sont filtrés à partir de ces règles :

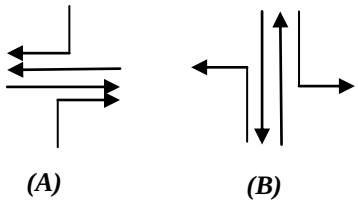


FIGURE IV. 15 ***(A) CAS POSSIBLE (B) CAS IMPOSSIBLE***

Puisqu'un message est découpé en paquets, eux même décomposés en flits, alors on peut réécrire les règles comme suit :

- Un flit d'un paquet qui se trouve dans la zone active peut circuler dans l'axe des x et pas dans l'axe des Y ;
- Un flit d'un paquet qui se trouve dans la zone active peut passer du nord vers le ouest mais pas vers l'est ;
- Il peut être acheminé en arrivant du Sud vers le l'Est non vers l'ouest.

Cette partie développe formellement la stratégie du réseau NoC. Cependant, nous avons donné une esquisse de la modélisation formelle [150]. Il est à noter que le raffinement nous permet de briser la complexité du fonctionnement en développant notre spécification à différents niveaux d'abstraction (raffinements).

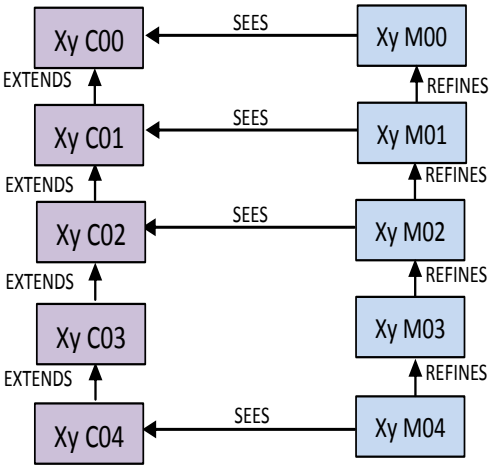


FIGURE IV. 16 ***MODELISATION ETAPE-PAR-ETAPE DE L'ARCHITECTURE NOC***

IV.2.3. Spécification avec RODIN

Le niveau abstrait : c'est le niveau abstrait qui permet de définir le rôle du réseau d'envoyer un nombre infini de message découpé en paquets d'une source à une destination.

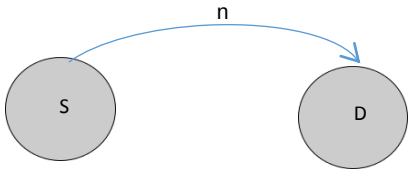


FIGURE IV. 17 ***NIVEAU ABSTRAIT***

xyC00 est le contexte abstrait qui spécifie les paquets, les nœuds, les sources, les destinations de chaque paquet. Les axiomes suivants décrient comment la source et la destination ont été définies pour un paquet. Chaque paquet a une unique source et unique destination qui sont différentes.

axm3 : src ∈ PACKETS → NODES

axm4 : dst ∈ PACKETS → NODES

La machine **xyM00** spécifie l'envoie et la réception de paquets de manière abstraite ;

L'envoie est défini avec l'action : sent ≔ sent ∪ {s ↦ p}

La réception avec l'action : rcvd ≔ rcvd ∪ {d ↦ p}

xyC01 : s'occupe de la division des paquets en flits ;

xyM01 :

axm1 : flits ∈ PACKETS → ℙ1(FLITS)

axm2 : ∀ p1, p2 · p1 ∈ PACKETS ∧ p2 ∈ PACKETS ∧ p1 ≠ p2 ⇒

L'événement SEND_FLIT est défini à ce niveau par l'action : f_sent ≔ f_sent ∪ {s ↦ f}

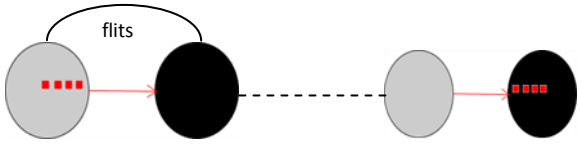


FIGURE IV. 18 *L'ENVOIE DES PAQUETS AUTANT QUE FLITS*

Second raffinement : l'ajout de la variable LocCopy

Le contexte **xyC02 :** la copie locale de paquet se trouve à l'origine dans les sources, et unique dans les sources de ces paquets. Le théorème (axm10) précise que la copie locale se trouve à l'origine dans un seul endroit du réseau.

axm7 : InitLocCopy ∈ NODES ↔ PACKETS

axm8 : ∀ p · p ∈ PACKETS ⇒ src(p) ↦ p ∈ InitLocCopy

axm9 : ∀ n, p · n ∈ NODES ∧ p ∈ PACKETS ∧ n ↦ p ∈ InitLocCopy ⇒ n = src(p)

axm10 : ∀ n1, n2, p · n1 ∈ NODES ∧ n2 ∈ NODES ∧ p ∈ PACKETS ∧ n1 ↦ p ∈ InitLocCopy ∧ n2 ↦ p ∈ InitLocCopy ⇒ n1 = n2

L'événement SEND_FLIT est défini à ce niveau par l'action : f_sent ≔ f_sent ∪ {s ↦ f}

Xy MO2 :

Dans cette machine, le raffinement ajoute l'événement RECEIVE_FLIT pour spécifier les conditions de réception d'un paquet :

act1 : f_rcvd ≔ f_rcvd ∪ {d ↦ f}

Troisième raffinement :

Xy M03 est un raffinement du comportement d'un nœud dans le cas où il sera défaillant, et quand il redevient normal. Il est exprimé dans les événements DISABLE et ENABLE. Ce niveau de raffinement nous permet aussi de créer la variable locCopy (voir figure IV.7. (02)) pour assurer l'envoie des flits d'un paquet sans attente.

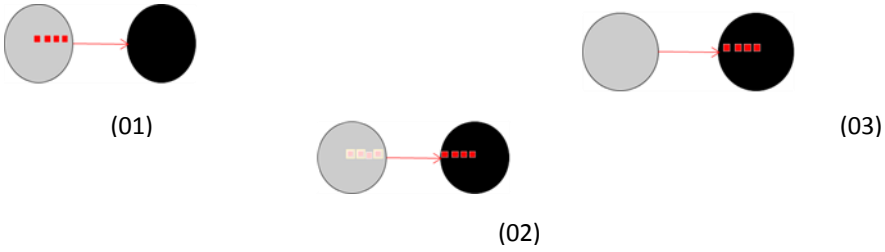


FIGURE IV. 19 *ROUTAGE DES FLITS EN GARDANT UNE COPIE LOCALE*

Les invariants suivants permettent de montrer comment ce modèle a été spécifié :

inv1 : gr ⊆ NODES × NODES

inv2 : str ∈ NODES ↔ FLITS

inv3 : ran(str) ⊆ ran(f_sent)

inv4 : ∀f· f ∉ ran (f_rcvd) ∧ f ∈ ran(f_sent) ⇒ f ∈ ran(str)

inv5 : locCopy ∈ NODES ↔ PACKETS

Disable : le nouveau graphe new-gr sera le graphe courant sans les liens bidirectionnels entre le nœud n et ses voisins

WHERE

grd1 : n ∈ NODES

grd2 : n ∈ dom(gr)

grd3 : new_gr ⊆ NODES × NODES

grd4 : new_gr = gr \ (({n} × gr[{n}]) ∪ (gr[{n}] × {n}))

Enable : quand n ne fait plus partie du graphe courant, on remet n dans le graphe courant avec les liens bidirectionnels avec ses anciens voisins.

WHERE

grd1 : n ∈ NODES

grd2 : n ∉ dom(gr)

grd3 : n ∈ dom(g)

Quatrième raffinement :

Xy C03 : Le contexte nous permet de briser le contexte 04 pour simplifier la preuve avec Rodin.

- Extension de la définition du graphe initial. Ajout de la notion de graphe carré. g est le graphe initial, c'est un graphe dans lequel tous les nœuds qui peuvent être reliés entre eux à cause de leurs coordonnées, sont voisins. C'est un graphe qui fait partie des graphes admissibles.

(∀ n1, n2, c1, c2 ·n1 ↦ c1 ∈ posnd ∧ n2 ↦ c2 ∈ posnd ∧ n1 ≠ n2 ∧(

(prj1(c1) = prj1(c2)−1 ∧ prj2(c1) = prj2(c2)) ∨

(prj2(c1) = prj2(c2)−1 ∧ prj1(c1) = prj1(c2)) ∨

(prj1(c1) = prj1(c2)+1 ∧ prj2(c1) = prj2(c2)) ∨

- Si on a un nœud x avec des coordonnées cx et qu'il existe un nœud d avec des coordonnées cd, tel que la coordonnée en abscisse d est inférieure à celle en abscisse de x, alors on peut encore se déplacer vers la gauche à partir de x.

$$\forall x, cx \cdot x \in NODES \land cx = posnd(x) \land (\exists d, cd \cdot d \in NODES \land cd = posnd(d) \land prj1(cx) > prj1(cd)) \Rightarrow$$

- Si on a un nœud x avec des coordonnées cx et qu'il existe un nœud d avec des coordonnées cd, tel que la coordonnée en abscisse d est supérieure à celle en abscisse x, alors on peut encore se déplacer vers la droite à partir de x.

$$\forall x, cx \cdot x \in NODES \land cx = posnd(x) \land (\exists d, cd \cdot d \in NODES \land cd = posnd(d) \land prj1(cx) < prj1(cd)) \Rightarrow (prj1(cx)+1 \mapsto prj2(cx)) \in ran(posnd)$$

- g est le graphe initial. C'est un graphe dans lequel tous les nœuds reliés entre eux deux à deux ont des coordonnées adjacentes en x et en y. C'est un graphe qui fait partie des graphes admissibles.

$$\begin{aligned}&(\forall n1, n2, c1, c2 \cdot n1 \mapsto c1 \in posnd \land n2 \mapsto c2 \in posnd \land \\ &n1 \neq n2 \land n1 \mapsto n2 \in g \Rightarrow \quad (\\ &\qquad (prj1(c1) = prj1(c2)-1 \land prj2(c1) = prj2(c2)) \lor \\ &\qquad (prj2(c1) = prj2(c2)-1 \land prj1(c1) = prj1(c2)) \lor \\ &\qquad (prj1(c1) = prj1(c2)+1 \land prj2(c1) = prj2(c2)) \lor\end{aligned}$$

- On exprime les "voisins" en diagonale "à distance 1" d'un nœud selon la condition suivante :

$$ind\_nbg \in NODES \rightarrow \mathbb{P}(NODES)$$

- Si un nœud nd fait partie de la zone défaillante de new_f_node et qu'un nœud f_node défaillant de l'ensemble set_of_f_nodes est un voisin direct ou en diagonale de nd, alors il fait partie de la zone défaillante autour de new_f_node.

$$\forall n, r, nod, nd \cdot n \mapsto r \in dom(zn) \land nod \in r \land nd \in zn(n \mapsto r) \land (nod \mapsto nd \in g \lor nd \in ind\_nbg(nod)) \Rightarrow nod \in zn(n \mapsto r)$$

- Théorème : si un nœud f_node défaillant de l'ensemble de nœuds set_of_f_nodes est un voisin direct ou en diagonale de new_f_node, alors il fait partie de la zone défaillante de new_f_node.

$$\forall n, r, nod \cdot n \mapsto r \in dom(zn) \land nod \in r \land (n \mapsto nod \in g \lor nod \in ind\_nbg(n)) \Rightarrow nod \in zn(n \mapsto r)$$

Xy C04 : Ce contexte introduit les opérateurs permettant de calculer la zone active entourant la zone défaillante proche d'un nœud. Le rectangle donné par za(a) et englobant a contient les nœuds nd dont les coordonnées (x,y) sont définies ainsi :

LimXmin(a)$\leq x \leq$ *LimXmax(a) et LimYmin(a)* $\leq y \leq$ *LimYmax(a)*

axm24 : $\forall a \cdot a \subseteq NODES \wedge a \neq \emptyset \Rightarrow za(a) = \{nd \mid (prj1(posnd(nd)) \geq LimXmin(a) \wedge prj1(posnd(nd)) \leq LimXmax(a) \wedge prj2(posnd(nd)) \geq LimYmin(a) \wedge$

Plusieurs cas se présentent :

Cas 1 : quand on désactive un nœud, on le groupe avec d'autres nœuds défaillants dans une zone défaillante. Par exemple on a le graphe g suivant :

n00	**n10**	**n20**	**n30**	**n40**
n01	**n11**	**n21**	**n31**	**n41**
n02	**n12**	**n22**	**n32**	**n42**
n03	**n13**	**n23**	**n33**	**n43**
n04	**n14**	**n24**	**n34**	**n44**

Cas 2 : Imaginons que n31 soit défaillant ainsi que n11, c'est-à-dire défaillant = {n11,n31}.

n00	**n10**	**n20**	**n30**	**n40**
n01	**n11**	**n21**	**n31**	**n41**
n02	**n12**	**n22**	**n32**	**n42**
n03	**n13**	**n23**	**n33**	**n43**
n04	**n14**	**n24**	**n34**	**n44**

Puisque ces nœuds ne sont pas voisins ou ne sont pas en diagonale (distance diagonale 1) l'un de l'autre, nous avons :

Groupe nœuds défaillants (n11) = {n11} ;

Groupe nœuds défaillants (n31) = {n31} ;

Cas 3 : Si n21 devient à son tour défaillant, on a une mise à jour de ces groupes de nœuds défaillants selon le graphe suivant :

n00	**n10**	**n20**	**n30**	**n40**
n01	**n11**	**n21**	**n31**	**n41**
n02	**n12**	**n22**	**n32**	**n42**
n03	**n13**	**n23**	**n33**	**n43**
n04	**n14**	**n24**	**n34**	**n44**

- Groupe nœuds défaillants (n11) = {n11,n21,n31} ;
- Groupe nœuds défaillants (n31) = {n11,n21,n31} ;
- Ainsi qu'un nouveau nœud défaillant n21 et un groupe associé à n21 ;
- Groupe nœuds défaillants (n21) = {n11,n21,n31}

Cas 4 : si n02 devient défaillant, on a une mise à jour de ces groupes de nœuds défaillants selon le graphe suivant :

n00	**n10**	**n20**	**n30**	**n40**
n01	**n11**	**n21**	**n31**	**n41**
n02	**n12**	**n22**	**n32**	**n42**
n03	**n13**	**n23**	**n33**	**n43**
n04	**n14**	**n24**	**n34**	**n44**

- Groupe nœuds défaillants (n11) = {n02,n11,n21,n31} ;
- Groupe nœuds défaillants (n21) = {n02,n11,n21,n31} ;
- Groupe nœuds défaillants (n31) = {n02,n11,n21,n31} ;
- Ainsi qu'un nouveau nœud défaillant n02 et un groupe associé à n02 groupe nœuds défaillants (n02) = {n02,n11,n21,n31} ;

Cas 5 : C'est le groupe de nœuds défaillants qui aide à définir la zone active, puisqu'il suffit de l'entourer d'un rectangle. Cela pourra se faire plus ou moins facilement puisqu'on dispose des coordonnées des nœuds défaillants.

Cas 6 : On complète les groupes de nœuds défaillants pour avoir un rectangle à l'aide des valeurs maximums et minimums des coordonnées en x et y du groupe. Comme illustré dans le graphe suivant :

n00	**n10**	**n20**	**n30**	**n40**
n01	**n11**	**n21**	**n31**	**n41**
n02	**n12**	**n22**	**n32**	**n42**
n03	**n13**	**n23**	**n33**	**n43**
n04	**n14**	**n24**	**n34**	**n44**

Ici, on complète le groupe défaillant {n02,n11,n21,n31} avec {n01,n12,n22,n32}, et il ne reste plus alors qu'à entourer la zone formée par les nœuds rouges et bleu pour obtenir la zone active.

Dans notre cas d'étude, les événements ont été prouvés et aussi animés par PRoB. L'outil nous a permis d'animer et de déceler une série de problèmes comme par exemple, les interblocages (Deadlocks) ou d'autre comportements inattendus d'un modèle. Le déroulement de l'envoie et l'avancement des filts d'un paquet, et la réception est montré dans les figures comme suit :

1- Au début, on initialise que rien n'est encore envoyé et que rien n'est encore reçu.

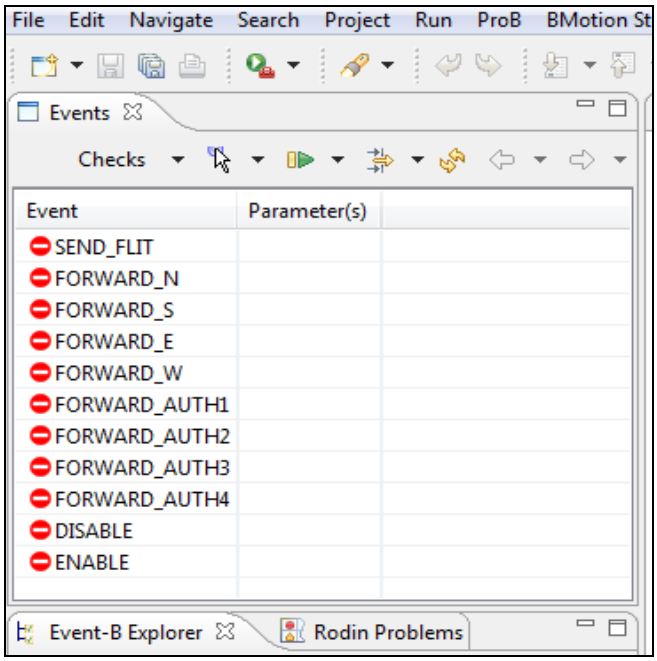


FIGURE IV. 20 *INITIALISATION DES EVENEMENTS*

2- On envoie un paquet, et on remarque qu'il a une copie locale qui est créé au niveau du nœud source comme mesure de sécurité (pour ne pas perdre l'information comme décrit dans le fonctionnement du réseau).

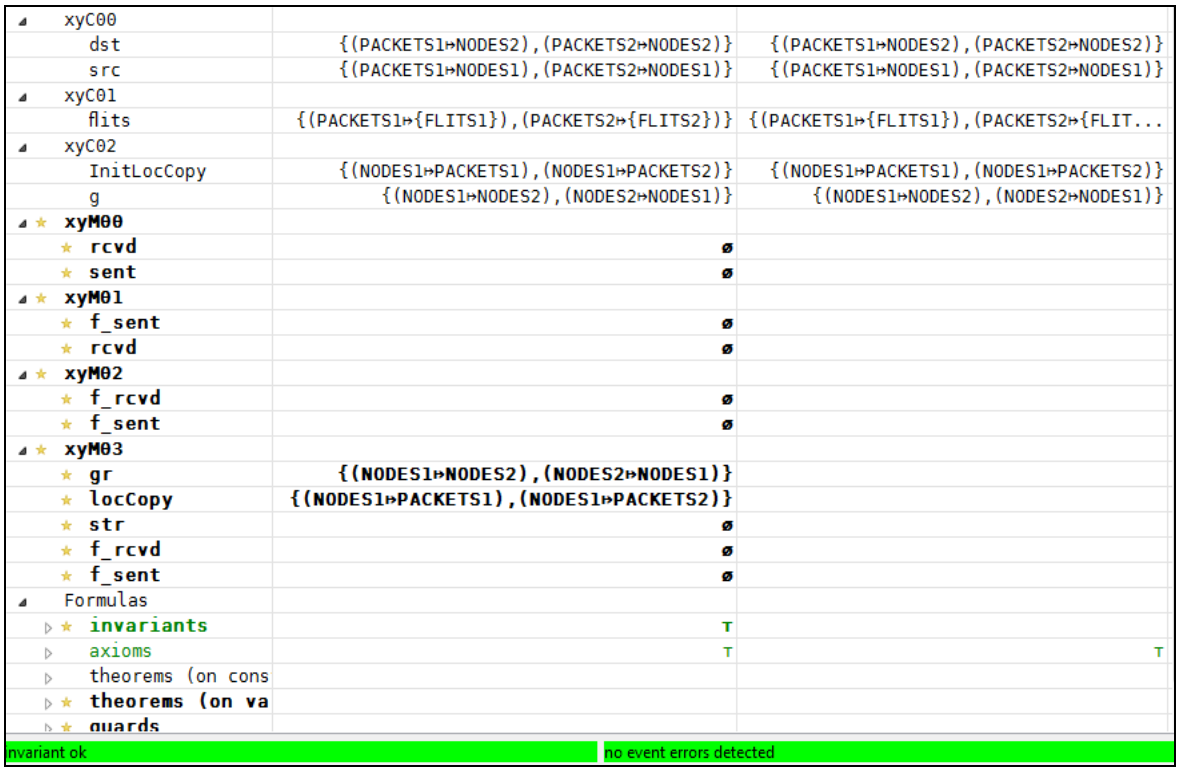
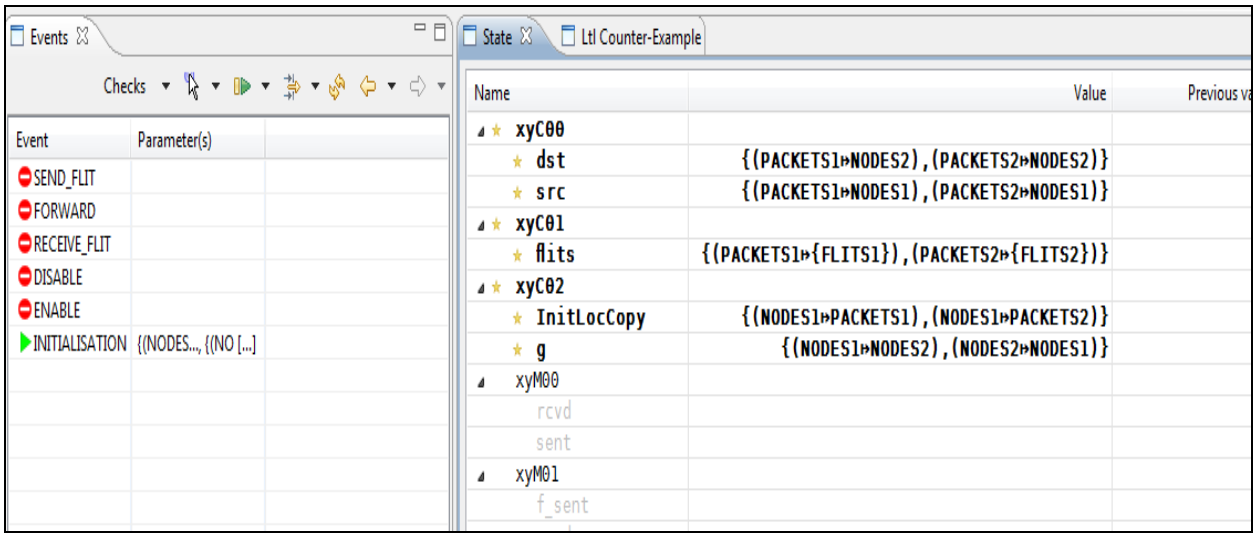


FIGURE IV. 21 *ACTIVATION DE L'EVENEMENT SEND*

3- Le paquet avance dans le réseau pour aller à la destination en déplaçant ces flits



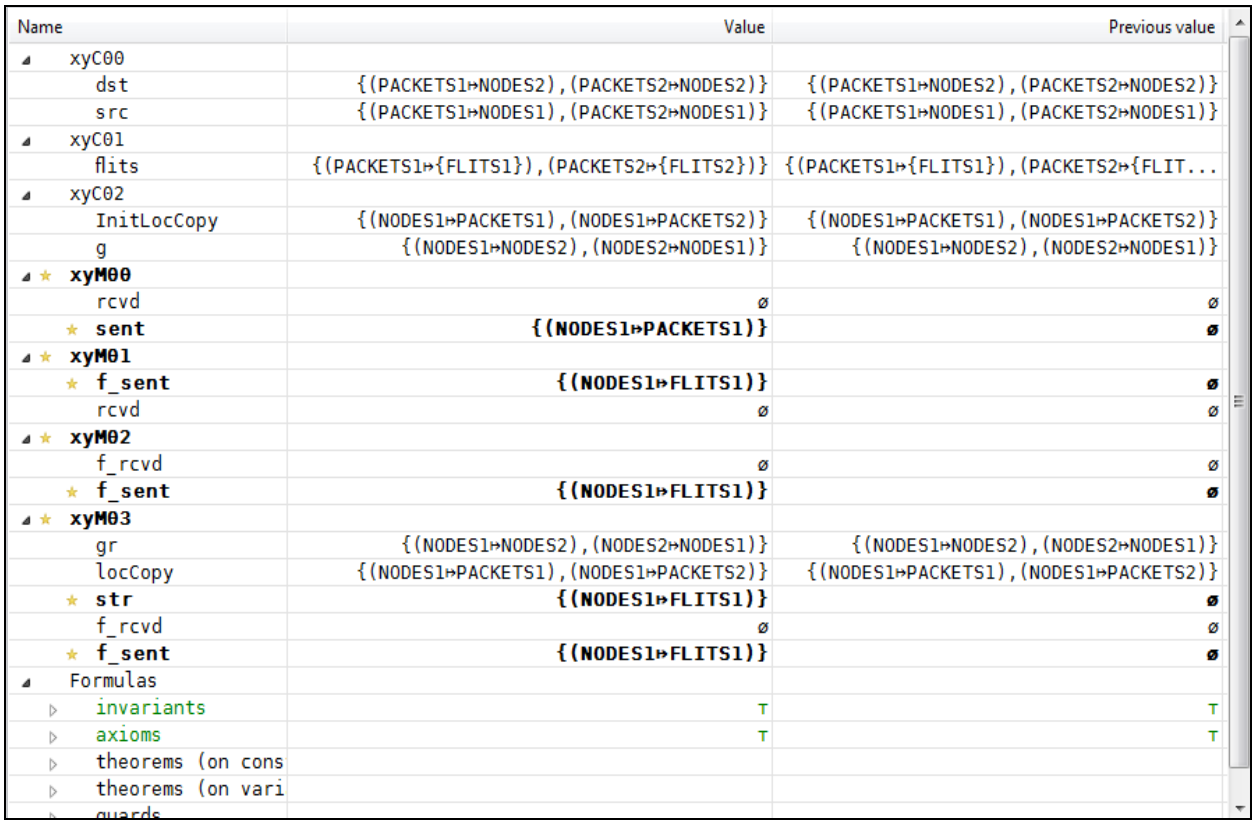


FIGURE IV. 22 *TRANSMISSION DE PAQUET DANS LE RESEAU*

4- On remarque qu'une fois le paquet reçu, la copie locale sera supprimée

Name	Value	Previous value
xyC00		
dst	{(PACKETS1↦NODES2),(PACKETS2↦NODES2)}	{(PACKETS1↦NODES2),(PACKETS2↦NODES2)}
src	{(PACKETS1↦NODES1),(PACKETS2↦NODES1)}	{(PACKETS1↦NODES1),(PACKETS2↦NODES1)}
xyC01		
flits	{(PACKETS1↦{FLITS1}),(PACKETS2↦{FLITS2})}	{(PACKETS1↦{FLITS1}),(PACKETS2↦{FLIT...
xyC02		
InitLocCopy	{(NODES1↦PACKETS1),(NODES1↦PACKETS2)}	{(NODES1↦PACKETS1),(NODES1↦PACKETS2)}
g	{(NODES1↦NODES2),(NODES2↦NODES1)}	{(NODES1↦NODES2),(NODES2↦NODES1)}
xyM00		
rcvd	∅	∅
sent	{(NODES1↦PACKETS1)}	{(NODES1↦PACKETS1)}
xyM01		
f_sent	{(NODES1↦FLITS1)}	{(NODES1↦FLITS1)}
rcvd	∅	∅
xyM02		
f_rcvd	∅	∅
f_sent	{(NODES1↦FLITS1)}	{(NODES1↦FLITS1)}
xyM03		
gr	{(NODES1↦NODES2),(NODES2↦NODES1)}	{(NODES1↦NODES2),(NODES2↦NODES1)}
locCopy	**{(NODES1↦PACKETS2),(NODES2↦PACKETS1)}**	**{(NODES1↦PACKETS1),(NODES1↦PACKE...**
str	**{(NODES2↦FLITS1)}**	**{(NODES1↦FLITS1)}**
f_rcvd	∅	∅
f_sent	{(NODES1↦FLITS1)}	{(NODES1↦FLITS1)}
Formulas		
invariants	⊤	⊤
axioms	⊤	⊤
theorems (on cons		
theorems (on vari		

FIGURE IV. 23 *RECEPTION DU PAQUET ET SUPPRESSION DE LA COPIE LOCALE*

Ce cas d'étude montré bien que l'animation et le model-checking sont des outils complémentaires à la preuve, indispensables pour la validation des modèles Event-B. Notre expérience montre que plusieurs modèles prouvés contiennent encore des fautes de comportement, et peuvent être décelées avec un animateur ou un model-checker. De plus, bien que des obligations de preuve pour l'absence d'interblocage soient prévues dans Event-B, elles ne sont pas encore implémentées dans RODIN. La raison est que cette obligation de preuve prend la forme d'une grande disjonction (la disjonction des gardes de tous les événements), qui est souvent très difficile à prouver. L'utilisation de PROB ne permet pas de prouver l'absence d'interblocages, mais elle permet de déceler automatiquement l'existence d'interblocages. En plus, l'obligation de preuve pour l'existence d'une solution pour les axiomes d'un contexte Event-B n'est pas encore implémentée dans RODIN. Là aussi, notre plug-in permet de valider l'existence d'une solution. S'il n'y a pas moyen de trouver des valeurs pour les constantes, l'outil va avertir l'utilisateur.

IV.3. STRUCTURE DE NoCS EN RÉSEAU

Une proposition d'un nouveau mécanisme auto-organisé pour la tolérance aux fautes des structures NoC adaptatives considérées comme les parties dynamiques ou reconfigurables d'un système en réseau multi-nœuds reconfigurables. Ce système était proposé dans [135]. Plus précisément, l'approche proposée repose sur l'auto-détection par tests et corrections d'erreurs au sein des structures NoC des nœuds communicants d'un système en réseau. Dans l'approche

considérée, on dissocie les tests et la fiabilité des parties statiques et dynamiques des nœuds reconfigurables.

IV.3.1. Vue d'ensemble de Vertex algorithmes de coloration

La brisure de symétrie a toujours été un problème central dans les systèmes distribués. Plusieurs techniques ont été mises en œuvre afin d'y parvenir, comme les algorithmes de Maximal Set Independance (MSI), des algorithmes de coloration de graphe, etc.

Un algorithme de coloration des sommets est un procédé de marquage graphique [119] [136] [137] [138] [139] [140] [141] [142]. Son objectif est d'attribuer les étiquettes vers les sommets du graphe. Les étiquettes sont souvent assimilées à des couleurs d'où son appelation *d'algorithme de graphe coloré*. La coloration / étiquetage est réalisée de telle manière qu'il n'existe pas deux sommets adjacents du graphe qui partage le même label/couleur. Une coloration propre d'un graphe G = {V, E} (V avec l'ensemble des sommets de G et E l'ensemble de ses arcs), en utilisant un ensemble de couleurs (couleurs ⊂ N | COULEURS = {1..N}), est une fonction f telle que (F: V → COULEURS|f (i) := f(j) si i ↔ j∈E). La valeur N minimale pour laquelle f est satisfaisant est appelée nombre chromatique de G et est généralement désigné par χ ou χ (G).

Ces règles sont généralement appliquées sur les graphes simples (connecté, irréflexive, non orienté, non pondéré). Dans notre cas, nous n'avons pas de sommets, mais on conserve le principe pour l'appliquer sur notre réseau.

a. Aperçu d'algorithmes de coloration Vertex

Plusieurs algorithmes ont été développés afin de colorer un graphe. Comme décrit dans l'ouvrage de Duffy, O'Connell et Sapozhnikov [136], les algorithmes de coloration de sommets peuvent être classés en deux catégories :

• Les algorithmes utilisant des techniques centralisées [138] [143]. Le terme « centralisé » implique qu'il existe au moins un « administrateur » qui décide pour la coloration de graphe. Il peut s'agir d'un sommet du graphe ou d'une entité qui n'est pas une partie du graphe, avec la connaissance complète de celle-ci (de la structure, des bords, nombre de sommets...). Quand '"L'administrateur" découvre une coloration correcte, il envoie un message à tous les sommets que l'algorithme est complété et leur donne leurs couleurs.

• Les algorithmes utilisant des techniques distribuées [136] [139] [140] [141] [144] [119]. Les algorithmes de coloration de graphes centralisés ont été développés en premier. Mais certains problèmes, qui ne peuvent être résolus à l'aide de ces solutions sont apparus (attribution des fréquences dans la norme IEEE 802.11, dans les télécommunications [136] [144] [145]. Par conséquent, de nouveaux algorithmes ont été développés. Ils sont encore étudiés de nos jours, dans

le but de les améliorer ou à en découvrir de nouveaux. Ces nouveaux algorithmes impliquent tous les sommets du graphe qui est coloré, et les sommets ont leur propre « intelligence ». En général, ils choisissent leurs propres couleurs en utilisant des probabilités, et connaitre quand ils ont choisi la même couleur qu'un voisin. Lorsqu'ils ont une bonne couleur, ils se retirent de la courbe non colorée [136] [119] [139] [140] [141] [144].

Dans ce travail, nous nous concentrons sur le développement d'algorithmes utilisant des techniques distribuées. En fait, il n'y a pas ou peu de vérification de l'exactitude des algorithmes précédents [136] [139] [140] [141] [144] [119] qui envisagent certains nombres aléatoires pour définir le processus de coloration sécurisé. La contribution principale est encore l'analyse de la complexité, et peut être intégré ultérieurement dans nos modèles.

b. Domaines d'application des algorithmes de graphes colorés

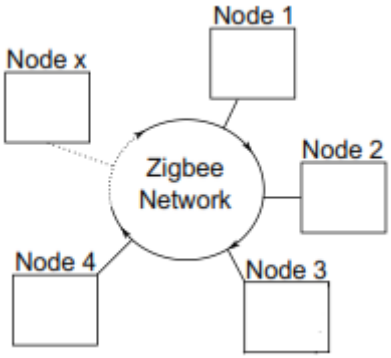
Il existe de nombreuses applications pratiques des algorithmes de graphes colorés qui comprennent :

- Une planification [146]. Les algorithmes de coloration de graphes peuvent être utilisés pour commander un ensemble des nœuds. Deux nœuds sont considérés comme adjacents quand ils peuvent avoir lieu en même temps. Le but est d'éviter que les tâches adjacentes se produisent en même temps. Mais dans notre cas il peut y avoir deux nœuds qui ont le même emploi, sauf par les nœuds tests qui ne peuvent pas corriger en même temps le même nœud défaillant.
- Chaque nœud correct doit être coloré en vert. Un nœud correct est un nœud qui peut envoyer et recevoir des paquets.
- Chaque nœud défaillant doit être coloré en rouge. Un nœud défaillant est un nœud qui ne peut ni envoyer et/ ou ni recevoir des paquets.
- Chaque nœud test doit être coloré en bleu. Un nœud test est un nœud choisi pour corriger le nœud défaillant.

IV.3.2. Spécification et vérification formelle

- **Niveau abstrait :**

Nous commençons par une spécification abstraite du problème en définissant le rôle du réseau qui envoie et reçoit des paquets. Ainsi, deux ensembles seront définis dans ce niveau ; les nœuds existant (NODES) et les paquets (PACKETS) envoyés par une source unique (src), et reçu par une unique destination (dst). Les sources sont différentes des destinations. Les axiomes suivants sont décrits dans le contexte du niveau abstrait comme suit [151] :



axm1 : NODES ≠ ∅

axm2 : PACKETS ≠ ∅

axm3 : src ∈ PACKETS → NODES

axm4 : dst ∈ PACKETS → NODES

axm5 : ∀ p· p∈ PACKETS ⇒ src(p) ≠ dst(p)

FIGURE IV. 24 *SYSTEME AUTO-ORGANISE MULTI-NŒUDS EN RESEAU APPLIQUE AUX COMMUNICATIONS SANS FIL*

On définit les variables sent et rcvd qui nous permettent d'effectuer les actions SEND et RECEIVE. Les invariants suivants sont décrits dans la machine du niveau abstrait comme suit :

inv1 : sent ∈ NODES ↔ PACKETS

inv2 : rcvd ∈ NODES ↔ PACKETS

inv3 : ran(rcvd) ⊆ ran(sent)

inv4 : ∀ s,p· s∈NODES ∧ p∈ PACKETS ∧ s↦p ∈ sent ⇒ s = src(p)

inv5 : ∀ d,p· d∈NODES ∧ p∈ PACKETS ∧ d↦p ∈ rcvd ⇒ d = dst(p)

inv6 : ∀ s1, s2, p· s1∈NODES ∧s2∈NODES ∧ p∈PACKETS ∧ s1↦p∈ sent ∧ s2↦p∈sent⇒ s1=s2

inv7 : ∀ d1, d2, p· d1∈NODES ∧d2∈NODES ∧ p∈PACKETS ∧ d1↦p∈rcvd ∧ d2↦p∈rcvd⇒ d1=d2

Les valeurs initiales des variables sont vides :

INITIALISATION ≙
act1 : sent ≔ ∅
act2 : rcvd ≔ ∅

L'événement SEND fait l'action : act1 : sent ≔ sent∪{s↦p}

L'événement RECEIVE fait l'action: act2 : rcvd := rcvd ∪{d↦p}

Premier raffinement :

Nous supposons que le graphe est donné sur un ensemble de nœuds. Ensuite, nous définissons un ensemble de couleurs (Red_Color, Green_Color, Blue_Color), dont les composantes seront les couleurs choisies par les nœuds lors de l'exécution de l'un des algorithmes de coloration de graphes.

CONSTANTS
GRAPH
Green_Color
Red_Color
Blue_Color

Nous spécifions certaines propriétés sur ces constantes de la manière :

axm1 :GRAPH ∈ NODES ↔ NODES

axm2 :GRAPH ≠ ∅

axm3 :COLOR ≠ ∅

axm4 :∀c·c∈COLOR⇒ c=Red_Color ∨ c=Green_Color ∨ c=Blue_Color

axm5 :∀ n·n∈ NODES ⇒ n∈dom(GRAPH)

axm6 :NODES◁id ∩GRAPH =∅

axm7 :GRAPH = GRAPH~

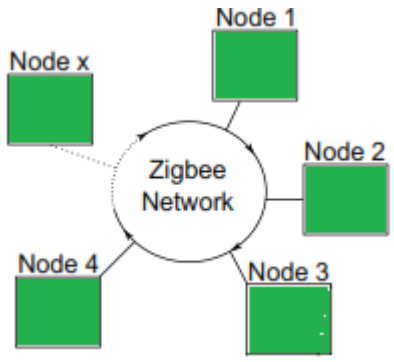
axm8 :∀ s·s⊆ NODES ∧ s ≠ ∅ ∧ GRAPH[s]⊆s⇒ NODES ⊆s

- L'axiome axm5 exprime que tous les sommets appartiennent au graphe, et qu'ils ne sont pas isolés,
- L'axiome axm6 exprime que le graphe est irréflexive : le voisin d'un sommet dans GRAPH doit être un autre sommet ; elle-même pas,
- L'axiome axm7 exprime que le graphe est symétrique ;
- L'axiome axm8 exprime que le graphe est connexe.

Dans la machine de ce niveau abstrait raffiné, on va considérer que tous les nœuds peuvent envoyer et recevoir des paquets. Cela nous permet donc de les colorés en vert (voir la figure.). La variable CCorrecteNode est définie avec la propriété suivante :

inv1 : CCorrectNode ∈ NODES ⇸ COLORS
inv2 : CorrectNode ∈ NODES

Cette propriété permet de définir un nœud fait partie de l'ensemble des nœuds colorés. Ensuite, le graphe peut être coloré en vert



green_color ≙

WHERE
grd1 : p ∈ PACKETS
grd2 : node∈ NODES
grd3 : node ∈ dom(GRAPH)
grd4 : node = CorrectNode
grd5 : Green_Color ∉ CCorrectNode[GRAPH[{node}]]
grd6 : CorrectNode↦p ∈ sent
grd7 : CorrectNode↦p ∈ rcvd
THEN
act1 : CCorrectNode(node) ≔ Green_Color
END

FIGURE IV. 25 *COLORATION DES NŒUDS DU SYSTEME AUTO-ORGANISE EN VERT*

Deuxième raffinement :

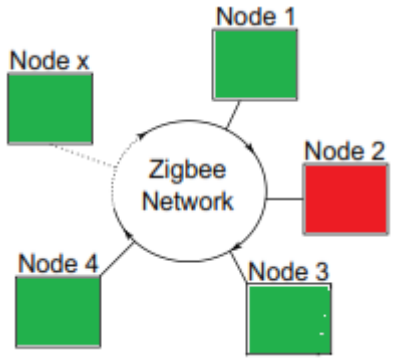
Maintenant, la question est de calculer la fonction de coloration avec la couleur rouge. Nous devons trouver une propriété inductive par simuler le calcul de la fonction. Deux variables seront ajoutés à ce niveau; FaultyNode et CFaultyNode qui sont définis avec les propriétés suivantes :

INVARIANTS

inv1 : FaultyNode ∈ NODES

inv2 : CFaultyNode ∈ NODES ⇸ COLOR

FaultyNode est un nœud défaillant donc il être coloré avec la couleur rouge. L'action sera donc :



red_color ≙

WHERE

grd1 : p∈ PACKETS
grd2 : node∈ NODES
grd3 : node ∈ dom(GRAPH)
grd4 : node = FaultyNode
grd5 : Red_Color ∉ CFaultyNode[GRAPH[{node}]]
grd6 : FaultyNode ↦p ∉sent
grd7 : FaultyNode ↦p ∉rcvd

THEN

act1 : CFaultyNode(node) ≔Red_Color

END

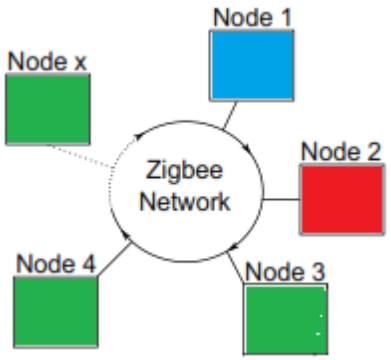
FIGURE IV. 26 *COLORATION DES NŒUDS DEFAILLANTS DU SYSTEME AUTO-ORGANISE EN ROUGE*

Troisième raffinement :

Dans ce niveau le calcul de la fonction de choix du nœud test est spécifié d'une façon simple afin de briser la complexité du rôle de ce nœud. Il suffit alors de respecter les règles du choix du nœud test et le reste est considéré comme procédure à appliquer après la sélection du nœud test selon les règles suivantes :

- La règle 1 : ne pas être un nœud défectueux. Ce qui est définit dans la grd9.
- La règle 2 : disposer des ressources matérielles nécessaires pour mettre en œuvre l'*IP* testeur.
- La règle 3 : disposer du *bitsream* de configuration de l'*IP* testeur. Les grd10 et grd11 respect cette règle parce que si le nœud ne dispose pas du fichier de reconfiguration, il peut néanmoins le recevoir à partir d'un autre nœud.
- La règle 4 : ne pas être occupé par une tâche prioritaire. Les grd7 et grd8 veulent dire que ce nœud n'est pas occupé avec l'envoie ou la réception de paquets.

Après avoir respecté les règles, le nœud peut être choisi comme nœud test (voir figure.) et l'action de coloration du nœud test en bleu est effectuée.



blue_color ≙

WHERE

grd1 : p∈ PACKETS
grd2 : node∈ NODES
grd3 : node ∈ dom(GRAPH)
grd4 : node = TestNode
grd5 : Blue_Color ∉ CTestNode[GRAPH[{node}]]
grd6 : TestNode ↦p ∉sent
grd7 : TestNode ↦p ∉rcvd
grd8 : TestNode ≠ FaultyNode
grd9 : src(BitstreamPacket)= TestNode
grd10 : dst(BitstreamPacket)= TestNode

THEN

act1 : CTestNode(node) ≔Blue_Color

END

FIGURE IV. 27 *COLORATION DU NŒUDS TEST DU SYSTEME AUTO-ORGANISE EN BLEU*

IV.3.3. Résultats et performances

La méthode Event-B génère des obligations de preuve. Le résumé de ces obligations de preuve libéré automatiquement ou de manière interactive est une mesure de la complexité du développement lui-même :

Element Name	Total	Auto	Manual	Reviewed	Undischarged
Dft_Test_NoC	**29**	**26**	**3**	**0**	**0**
Test C00	1	1	0	0	0
Test C01	0	0	0	0	0
Test M00	16	14	2	0	0
Test M01	3	3	0	0	0
Test M02	3	3	0	0	0
Test M03	6	5	1	0	0

TABLEAU IV. 2 *SOMMAIRE DES STATISTIQUES DES OBLIGATIONS DE PREUVE*

On remarque que la plupart des preuves sont faites automatiques ce qui nous permet de dire qu'on n'a pas forcé la preuve. Le développement progressif actuel se concentre sur des algorithmes de coloration de Vertex. Ils ont l'intégration d'erreurs possibles lors du choix de couleurs par des nœuds et c'est le point ou on peut manipuler.

CONCLUSION

Ce chapitre présente les premiers éléments d'une méthodologie globale pour gérer l'approche correcte par construction à base de raffinement appliqué à un réseau NoC. D'autres travaux sont

nécessaires pour adapter la technique et les outils. C'est aussi une première étape vers notre objectif d'obtenir un cadre de développement intégrant des vérifications formelles pour un mécanisme auto-organisé pour la tolérance aux fautes d'un réseau multi-nœuds reconfigurables. L'approche développée permet de tester efficacement les NoCs de type MPSoC. Notre système multi-nœuds s'appuie sur des modules de communication sans fil ZigBee Xbee permettant une grande robustesse de transmission. Notre travail se concentre ici sur l'analyse et la vérification formelle des performances proposées afin d'améliorer et de proposer des techniques prévues par le cadre de l'événement B. Ces travaux sont menés en parallèle avec les travaux proposés et simulés avec des outils de modélisation bas niveau, ce qui permet de minimiser le temps de réalisation et de vérification de ces mécanismes intégrés complexes.

CONCLUSION GENERALE ET PERSPECTIVES

L'évolution actuelle des architectures des SoC tendant vers des MPSoC reconfigurables dynamiquement et sûres de fonctionnement à base de technologie FPGA. En effet, la flexibilité et la bande passante sont nécessaires à une mise en œuvre de SoC contenant des IP. Le support de communication associé joue un rôle important dans les MPSoC qui nécessite d'introduire des mécanismes de sûreté de fonctionnement (SdF) afin de prévenir la défaillance d'un système.

Les méthodes formelles sont apparues depuis plusieurs décennies en milieu académique. L'idée même de vérifier la correction des programmes date des années 60 et remonte même à l'apparition des ordinateurs. Quelques expériences en milieu industriel ont eu lieu durant les années 70, mais sans lendemain faute notamment d'outils de support. En effet, les manipulations formelles requises sont fastidieuses et sujettes à des erreurs lorsqu'elles sont manuelles. Actuellement, il y a un tournant avec l'arrivée à maturité de plusieurs méthodes, d'outils munis d'interfaces qui se modernisent, et de logiciels d'assistance à la preuve de plus en plus efficace.

L'analyse de l'état de l'art des deux premiers chapitres du manuscrit, nous a permis d'établir le besoin de réduire le temps de vérification et donc la sûreté de fonctionnement d'un système requis pour les NoCs. Ces réseaux nécessitent une protection contre les erreurs survenant au démarrage ou au cours de fonctionnement. Les fautes peuvent causer des défaillances partielles et générales du système. Dans ce contexte, les approches de SdF sont proposées pour trouver des solutions en ligne permettant la localisation et l'isolation de régions fautives du NoC afin de protéger sa fonctionnalité globale en particulier pour les réseaux adaptatifs. Une solution architecturale d'un réseau Q-Switch sûre de fonctionnement a été proposée et intègre les solutions de détection d'erreurs de routage pour les algorithmes adaptatifs, et de détection d'erreurs de paquets de données.

Dans ces travaux, on vise à proposer des mécanismes de vérification des performances d'un réseau sur puce pour les MPSoC en termes de fonctionnalités à la fois au niveau des blocs de tests et des algorithmes utilisés et appliqués au sein du NoC pour la détection et l'isolation d'erreurs. Une proposition de prototypage rapide par preuve formelle, nous a permis de réduire le temps de vérification et de prouver formellement la sûreté de fonctionnement de l'architecture QNoC. Notamment, en ce qui concerne les périodes de modifications des acheminements des données dans le cas de détection d'erreurs localisées au sein de ce même réseau.

La principale originalité de l'algorithme de routage proposé et intégré au sein du réseau QNoC est qu'il repose sur la base de règles de contournement. Le routage est alors effectué dans une zone activée dépendant du nombre de nœuds ou de régions défectueuses ou pour le cas d'une zone

rectangulaire (ou un groupe de nœuds formant un rectangle) ; les nœuds activés forment un anneau autour de la région défaillante considérée et dans lequel une zone de routage est activée pour permettre des itinéraires de contournement. Nous avons formalisé notre projet de preuve qui permet la validation d'un algorithme de routage de paquets dans les nœuds voisins des régions défectueuses du réseau. Notre approche repose sur un développement progressif des activités de routage de l'architecture de réseau sur puce, en utilisant le formalisme événementiel B. La formalisation de l'architecture est présentée d'une manière hiérarchique à partir d'un niveau abstrait jusqu'à un niveau plus concret. Nous avons amélioré et prouvé l'algorithme de routage tolérant aux fautes du Q-Switch à l'aide de l'outil RODIN où les obligations de preuve sur les mesures de la complexité du développement ont été automatiquement/manuellement déchargées. Cette approche permettra d'élaborer le processus global de la modélisation B événementielle qui rend la notion de raffinement intuitive. La principale utilisation/fonctionnement des expérimentateurs (fourni par la plateforme RODIN) nous a permis d'acquitter ces obligations. Contrairement à seule vérification par simulation, notre travail se concentre ici sur l'analyse d'une vérification formelle proposée pour améliorer des vérifications fonctionnelles selon la méthode Event-B à l'aide de ses plug-in spécifiques et un cadre pour le développement des architectures réseau sur puce et l'algorithme de routage XY en utilisant les propriétés essentielles ainsi qu'une preuve de correction formelle qui affirme sa sécurité.

Dans le cadre de nos futurs travaux, nous considérons une traduction la plus concrète (détaillée et proche de la forme algorithmique) dans un modèle de langage intermédiaire à partir duquel une description matérielle (par exemple en VHDL) peut être extraite. En outre, nous notons que les premiers niveaux de la conception d'une architecture NoC avec la spécification Event-B sont des cas globaux de routage méthodologique exprimé au domaine des structures intéressantes et génériques réutilisables à base de raffinement. Nous prévoyons d'étudier davantage le domaine de la généricité et la réutilisabilité des modèles déjà validés.

BIBLIOGRAPHIE

[1] S.JOVANOVIC, C.TANOUGAST et S.WEBER, «A new high-performance scalable dynamic interconnectio, for FPGA-based reconfigurable systms. Application-Specific Systems,» *Application-Specific Systems, Architectures and Processors,* pp. 61-66, 2-4 July 2008.

[2] S.JOVANIVIC, C.TANOUGAST, C.BOBDA et S.WEBER, «A Dynamic Schalable Structure for Dynamically Reconfigurable FPGAs,» *Microprocessors and Microsystemss, Elsevier,* vol. 33, n° %11, pp. 24-36, Febrary 2009.

[3] J.DELORME, «Méthodologie et modélisation et exploitation d'architecture de réseaux sur puce appliquée aux télécommunications,» Institut national des sciences appliquées de Rennes, 2007.

[4] R.LEMAIRE, «Conception et modélisation d'un système de contrôle d'applications de télécommunication avec une architecture de réseau sur puce,» Institut National Polytechnique de Grenoble (CEA-LETI), 2006.

[5] T.MARESCAUX, A.BARTIC, D.VERKEST, S.VERNALDE et R.LAUWEREINS, «Interconnection Networks Enable Fine-Gram Dynamic Multi-tasking on FPGA,» *the series Lecture of Notes in Computer Science,* vol. 2438, pp. 795-805, 2002.

[6] S.Pasricha et N.Dutt, «On-Chip Communication Architecture,» *The Morgan Kaufmann Series in Systems on Silicon Elsevier,* pp. 439-466, 2008.

[7] L.DURET, Y.THIERRY-MIEG, S.BAARIR A et F.KORDON, «Nouvelles techniqques de model cheking pour la vérification de systèmes complexes,» rapport technique, HAL archives Génie Logiciel, 2004.

[8] D.Bjorner et C-B.Jones, «The vienna development method: the meta-language,» *Lecture Notes in Computer Science,* vol. 61, 1978.

[9] E. Clarke, O. Grumberg et S. Jha, «Verifying parameterized networks.,» *ACM Transactions on Programming Languages and Systems (TOPLAS),* p. 19(5):726 – 750, September 1997.

[10] A. F. a. F. S. R. Bharadwaj, «Formalizing Inductive Proofs of Network Algorithms,» In Proceedings of 1995 Asian Computing Science, 1995.

[11] P. Curzon., «Experiences formally verifying a network component,» *in proceedings of IEEE Conference on Computer Assurance, IEEE Press,* 15-18 june 1994.

[12] M. G. a. T. Melham, «Introduction to Hol: A Theorem Proving Environnement for Higher Order Logic,» Cambridge, University Press, 1993.

[13] F. V. M. Z. K. G. E. R. A. R. B. Gebremichael, «Deadlock Prevention in the Æthereal Protocol,» In Proceedings of CHARME'05, October 2005.

[14] P.GUERRIER, «Un Réseau d'interconnexion pour Systèmes Intégrés,» Thèse de Doctorat, Université PARIS 6 R Pierre et Marie Curie, 2000.

[15] A.ANDRIAHANTENAINA et A.GREINER, «Micro-réseau pour systèmes intégrés: Réalisation d'un réseau SPIN à 32 ports,» *Design Automation and Test in Europe Conference, Munchen, Germany,* pp. 1128-1129, March 2003.

[16] [En ligne]. Available: http://ocpip.org. [Accès le 12 avril 2014].

[17] A.MELLO, L.TADESCO, N.CALAZANS, V.LAERT et F-G.MORAES, «Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC,» *18th SBCCI, Florianolpolis,* pp. 178-183, 4-7 September 2005.

[18] X-T.TRAN, «Méthode de Test et Conception en Vue du Test pour les Réseaux sur Puce Asynchrones: Application au Réseau ANoC,» Thèse de doctorat, Institut National Polytechnique de Grenoble (CEA-LETI), 2008.

[19] [En ligne]. Available: http://www-leti.cea.fr/fr. [Accès le 12 avril 2014].

[20] F.MORAES, A.MELLO, L.MôLLER, L.OST et N.CALAZANS, «A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping,» *IFIP Very Large Scale Integration (VLSI-SoC), Darmstadt, Germany,* pp. 318-323, 1-3 December 2003.

[21] L.Mirko et P.Massimo, «Exploring Energy/Performance Tradeoffs in Shared Memory MPSoCs: Snoop-Based Cache Coherence vs. Software Solutions,» *EDAA-European design and Automation*

Association. Munich, Germany, pp. 208-513, Mar 2005.

[22] N.EISLEY, L-S.PEH et L.SHANG, «In-Network Cache Coherence,» *39th annual IEEE/ACM International Symposium on Microarchitecture, MICRO-39, Orlando, FL,USA,* pp. 321-332, 9-13 Dec 2006.

[23] H.CHTIOUI, R.BEN ATITALLAH, S.NIAR, M.ABID et J6L.DEKEYSER, «Gestion dela cohérence des caches dans les architectures MOEPC utilisant des NoC complexes,» *Conférence Française sur les Systèmes d'Exploitation (CFSE), Fribourg, Suisse,* 12-16 Février 2008.

[24] L.ANGHEL et M.NICOLAIDIS, «Cost Reduction and Evaluation of a Temporary Faults Detecting Technique,» *Design, Automation and Test in Europe (DATE 2000), Paris,* pp. 591-598, 27-30 March 2000.

[25] I.CIDON et I.KEIDER, «Zooming in on Network-on-Chip Architectures,» *Technical Report CCIT 565, Technion Department of Electrical Engineering,* vol. 5869 of the series Lecture Notes in Computer Science, December 2005.

[26] C.NICOPOULOS, V.NARAYANAN et C-R.DAS, «Network-on-chip architectures: a hlostic design exploration,» *Springer,* 2010.

[27] P.R. Panda, «SystemC-a modeling platform supporting mupltiple design avbstractions,» *The 14th International Symposium on System Synthesis Proceeding, Motreal, Duebec, Canada,* pp. 75-80, September 30-October3, 2001.

[28] [En ligne]. Available: http://www.synopsys.com. [Accès le 21 novembre 2015].

[29] Handel-C Language Reference Manual Version 2.1, Celoxica, clerixica Limited, 2001.

[30] [En ligne]. Available: http://www.jhdl.org/documentation/guide.html. [Accès le 21 novembre 2015].

[31] G.E.Moore, «Cramming more components onto integrated crcuits,» vol. 86, n° %11, pp. 82-85, january 1998.

[32] L.Benini et G. De Micheli, «Networks on chip: a new paradigm for systems on chip design,» *in Proc-Design, Automation and Test in Europe Conference and Exhibition, Paris, France,* pp. 418-419, 4-8 March 2002.

[33] W.Dally, «Interconnect-Limited VLSI architecture,» *in Prod.IEEE International Inter-connect Technology Conference, San Francisco, CA, USA,* pp. 15-17, 24-26 May 1999.

[34] K.Keutzer, A.Newton, J.Rabaey et A.Sangiovanni-Vincentelli, «System-level-design: orthogonalization of concerns and platform-based design,» *IEEE Transactions on Computer-Aided Design of Inetegrated Circuits and Systems* , vol. 19 Issue 12, pp. 1523-1543, Dec 2000.

[35] M.Sgroi, M.Sheets, A.Mihal, K.Keutzer, S.Malik, J.Rabaey et A.Sangiovanni-Vincentelli, «Addressing the system-on-a- chip interconnect woes through communication based design,» *in Proc. Design Automation Conference, Yokoharna, Japan,* January 30-February 2, 2001.

[36] D.M.Chapiro, «Globally-asynchronous systms,» Ph.D. dissertation, Stanford Univ, CA, 1984.

[37] T.Meincke, A.hemani, S.Kumar, P.Ellervee, J.Oberg, T.Olsson, P.Nilsson, D.Lindqvist et H.Tenhunen, «Globally asynchronous locally synchronous architecture for large hign-performance ASICs,» *in Proc. IEEE International Symposium on Circuits and Systems ISCAS 99 Orlando, FL,USA,* pp. 512-515, 30 May-2 June 1999.

[38] D.Blaauw et K.Gala, «Deep-submicron issues in high-perfomance design,» *in Proc. International Workshop on power and Timing Modeling, Optimization and Simulation, Yverdon-lesbains Switzerland,* pp. 2-13, September 26-28 2001.

[39] K.Lahiri, A.Raghumathan et S.Dey, «Evaluation of the trafic-performance characteristics of system-on-chip communication architectures,» *in Proc. Fourteenth International Conference on VLSI Design, Bangalore, India,* pp. 29-35, 3-7 Jan 2001.

[40] C.Zeferino, M.Kreutz, L.Carro et A.Susin, «A study on communication issues for systems-on chip,» *in Proc. 15th Symposium Integrated Cicuits and Systems Design, Porto Alegre, Brazil,* pp. 121-126, September 9-14 2002.

[41] W.Dally et B.Towles, «Route packets, not wires: on-chip interconnection netwoks,» *in Proc.*

Design Automation Conference, Yokohama, Japan, pp. 684-689, January 30- February 2, 2001.

[42] S.Evain, «uSpider Environnement de Conception de Réseau sur Puce,» Ph.D. dissertation, IETR-INSA Rennes, Lab-STICC{UBS Lorient}, 2006.

[43] C.Grecu, P-P.Pande, A.lvanov et R.Saleh, «Structured interconnect architecture: a solution for the non-scalability of bus-based SoCs,» *in GLSVLSI 04: Proceedings of the 14th ACM Great Lakes symposium on VLSI. ACM, New York, NY, USA,* pp. 192-195, Apr.2004.

[44] A.Leroy, «Optimizing the on-chip communication architecture of low power systems-on-chip in deep submicron technology,» Ph.D. dissertation, Université Libre de Bruxelles, 2006.

[45] A.S.Tanenbaum, Computer networks (3rd ed). Prentice-Hall, 1996.

[46] D.Culler, A.Gupta et J.P.Sign, «Parallel Computer Architectute: A hard ware/software Approach,» Morgan Kaufmann Publishers Inc, 1999.

[47] P.Guerrier et A.Greiner, «A generic architecture for on-chip packet-switched interconnections,» *in Proc. Design Automatiion and Test in Europe Conference and Exhibition, Paris, France,* pp. 250-256, 27-30 March 2000.

[48] A.Jantsch et H.Tenhunen, Networkd on chip. Kluwer Academis Publishers, Hingham, MA, USA, 2003.

[49] H.Zimmermann, «OSI Reference Model: he ISO Model of Architecture for Open Systems Interconnection,» *IEEE Transactions on Communications,* vol. 28 Issue4, pp. 425-432, 1980.

[50] M.Millberg, E.Nilsson, R.Thid, S.Kumar et A.Jantsch, «The Nostrum backbone-a communication protocol stack for Networks on Chip,» *in Proc. 17th International Conference on VLSI Design, Mumbai, India,* pp. 693-696, 5-9 January 2004.

[51] T.Bjerregaard et S.Mahadevan, «A survey of research and practices of Network-on-chip,» *ACM Comput.Surv,* vol. 38, n° %11, pp. 1-51, 2006.

[52] J.Kim, D.Nicopoulos, N.Vijaykrishnan et C.R. Das, «Design and analysys of an NoC architecture from performance, reliability and energy perspective,» *Proc. Symposium on Architecture for Networking and Communication Systems ANCS'05, Princeton, NJ, USA,* pp. 173-182, October26-28,2005.

[53] W.Dally et B.Towles, «Principles and practices of Interconnection Networks,» *Morgan Kaufmann Publishers,* 2003.

[54] J.Quartana, «Conception de réseaux de communication sur puce asynchrones: application aux architecture GALS,» Ph.D dissertation, INRIA, 2006.

[55] G.D. Micheli et L.Benini, «Networks on chips: Technology and Tools (Systems on Silicon),» *Morgan Kaufmann Publishers,* 2006.

[56] A.SCHERRER, «Analyses statistiques des communications sur puce,» Thèse Doctorat, Ecole Normale Supérieure de Lyon, 2006.

[57] J.DELORME et D.HOUZET, «Technique de mapping pour les réseaux sur puce 2D,» *PATMOS 2007, Suéde GÖteborg,* pp. 1-6, Septembre 2007.

[58] E.Rijpkema, K.Goossens, A.Radulescu, J.Dielissen, J. van Meerbergen, P.WIelage et E.Waterlander, «Trade offs in the design of a router with both guarateed and beste effort services for networks on chip,» *on Proc. Design, Automation and Test in Europe Conference and Exhibition, Munich, Germany,* pp. 350-355, March 3-7, 2003.

[59] M.Karol, M.Hluchyj et S.Morgan, «Input Versus Output Queueing on a Space-Division Packet Switch,» *IEEE Transactions on Communications,* vol. 35, n° %112, pp. 1347-1356, 1987.

[60] W.Dally, «Virtual-channel flow control,» *in Proc. 17th Annual International Symposium on Computer Architecture, New York, NY, USA,* pp. 60-68, 28-31 May 1990.

[61] L.Ni et P.Mckinley, «A survey of wormhole routing techniques in direct networks,» *IEEE Computer,* vol. 26 Issue 2, pp. 62-76, 1993.

[62] W.Dally et H.Aoki, «Deadlock-free adaptive routing in multicomputer networks using virtual channels,» *IEEE Trans. Parallel and Distributed Systems,* vol. 4, n° %14, pp. 466-475, 1993.

[63] W-J.DALLY, «Future Directions for On-Chip Interconnection Networks,» *OCIN Workshop, Stanford, California, USA,* pp. 1-20, 6-7 December 2006.

[64] F.KARIM, A.NGUYEN et S.DEY, «An interconnect architecture for networking systems on chips,» *Micro, IEEE,* vol. 22 Issue 5, pp. 36-45, September-October 2002.

[65] C.E. Leiserson, «Fat-trees: universal networks for hardware-efficient supercomputing,» *IEEE Trans.Computer,* vol. 34, n° %110, pp. 892-901, 1985.

[66] S.Kumer, A.Jantsch, J-P. Soininen, M.Forsell, M.Millberg, J.Oberg, K.Tiensyrja et A. Hemani, «A network on chip architecture and design methodology,» *in Proc. IEEE Computer Society Annual Symposium on VLSI, Pittsburgh, PA,USA,* pp. 105-112, 25-26 April 2002.

[67] J. Duato, S.Yalamanchili et L. Ni, «Interconnection Networks: An Engineering Approach,» *Morgan Kaufmann Publishers,* 2002.

[68] J.DUATO, «A new theory of deadlock-free adaptive multicast routing in wormhole networks,» *Proceedings of the Fifth IEEE Syposium on Parallel and Distributed Processing, Dallas, USA,* pp. 64-71, 1-4 Dec 1993.

[69] P. Gaughan et S.Yalamanchili, «Adaptive routing protocols for hypercube interconnection networks computer,» vol. 26, n° %15, pp. 12-23, May 1993.

[70] M. Steenstrup, Routing in communications networks, Prentice Hall International (UK), 1995.

[71] M. Palesi, R.Holsmark, S. Kumar et V. Catania, «Application Specific Routing Algorithms for Networks on Chip,» *IEEE Transactions on Parallel and Distributed Systems,* vol. 20 Issue 3, pp. 316-330, Apr 2009.

[72] W-J. Dally, «Performance analisis of K-ary n-cube interconnection networks,» *IEEE Transactions,* vol. 39 Issue 6, pp. 775-785, 1990.

[73] J. HU et R. MARCULESCU, «DyAD- smart routing for networks-on-chip,» *Proceedings of the 41st Design Automation Conference San Diego, CA,USA,* pp. 260-263, 7-11 july 2004.

[74] J.KIM, D. PARK, T. THEOCHARIDES, N.VIJAYKRISHNAN et C.R. DAS, «A low latency router supporting adaptivity for on-chip interconnects,» *Proceeding of the 42nd annual Design Automation Conference San Diego, CA, USA,* pp. 559-564, June 13-17, 2005.

[75] C.-J. GLASS et L-M Ni, «Fault-tolerant wormhole routing in meshes,» *The Twenty- Third International Symposium on Fault-Tolerant Computing, FTCS-23. Digest of Papers, Toulouse, France ,* pp. 240-249, 22-24 June 1993.

[76] Jie WU, «A fault-tolerant and deadlock-free routing protocol in 2d meshes based on odd-even turn model,» *IEEE Transactions on computers,* vol. 52 Issue 9, pp. 1154-1169, Sept 2003.

[77] R.V. BOPPANA et S.CHALASANI, «Fault-tolerant wormhole routing algorithms for mesh networks,» *IEEE Transactions on Computers ,* vol. 44 Issue 7, pp. 848-864, 1995.

[78] K-H. CHEN et G-M. CHIU, «Fault-tolerant routing algorithm for meshes without using virtual channels,» *Journal of Information science and Engineering,* vol. 14 Issue 4, pp. 765-783, 1998.

[79] C.Glass et L. Ni, «The Turn Model for Adaptive Routing,» *in Proc. 19th Annual International Symposium on Computer Architecture, Queensland, Australia,* pp. 278-287, May 19-21, 1992.

[80] K. Goossens, J;Dielissen, J. van Meerbergen, P. Poplavko, A. Rdulescu, E. Rijpkema, E. Waterlander et P. Wielage , «Guaranteeing the quality of services in networks on chip,» *Networks on chip, Kluwer Academic Publishers,* pp. 61-82, 2003.

[81] E. Rijpkema, K. Goossens et P. Wielage, «A router Architecture for Networks on Silicon,» *Proceedings of Progress, 2nd worshop on embedded systems,* 2001.

[82] I. Augé, R-K Bawa, P. Guerrier, A. Greiner, L. Jacomme et F. Pétrot, «User Guided High Level Synthesis,» *International Conference on Very Large Scale Integration (VLSI'97), Gramado Brasil,* pp. 464-475, 1997.

[83] A.-S. Kumar, M-P Kumar, S. Murali , V. Kamakoti, L. Benini et G-D. Micheli, «A Buffer-Sizing Algorithm for Network- on-Chips wth Multiple Voltage-Frequency Islands,» *journal of electrical end computer engineering-special Issue of Networks-on Chip: Architectures, Design Methotodologies, and case Studies, Article ID 537286,* vol. 2012, p. 12, 2012.

[84] T. Marescaux, V. Nollet, J-Y. Mignolet, A. Bartic, W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde et R. Lauwereins, «Run-time support for heterogenous multitasking on reconfigurable SoCs,» *the VLSI Journal-Special Issue: Networks on chip and reconfigurable fabrics Integration,* vol. 38 Issue 1, pp. 107-130, October 2004.

[85] F. Karim, H. Konuk, Kim Keesup et S. Dey, «Validation and test of network processors and ASICs,» *Proceedings 20th IEEE VLSI Test Symposium(VTS 2002). Monterey, CA, USA,* pp. 407-407, 28 April- 2 May 2002.

[86] P.P. PANDE, C. GRECU, A. IVANOV et R. SALEH, «Design of a switch for network on chip applications,» *in Proceeding of the the 2003 International Symposium on Circuits ans Systems, ISCAS'03, Bangkok, Thailand,* vol. 5, pp. 217-220, 25-28 May 2003.

[87] O.LYSNE, T.M. PINKSTON et J. DUATO, «A methodology for developing dynamic network reconfiguration processes,» *in Proceedings, International Conference on Parallel Processing,* pp. 77-86, 9 October 2003.

[88] A.HANSSON et K. GOOSSENS, «Trade-offs in the configuration of a network on chip for multiple use-cases,» *Proc. International Symposium on Networks on Chip (NoCs), Princeton, New Jersey, USA,* pp. 233-242, 7-9 May 2007.

[89] D. CHING, P. SCHAUMONT et I. VERBAUWHEDE, «Integrated modelling and geration of a reconfigurable network-on-chip,» *International -Journal of embdded Systems,* vol. 1 Issue 3, pp. 218-227, 2005.

[90] B, AHMAD, A. ERDOGAN, S et KHAWAM, «Architecture of a dynamically reconfigurable NoC for adaptive reconfigurable MPSoC,» *First NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2006, Istanbul, Turkey,* pp. 405-411, 15-18 June 2006.

[91] A. Kumar, A; HANSSON, J.HUISKEN et H.CORPORAAL, «An FPGA design flow for reconfigurable network-based multi-processor systems on chip,» *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07, Nice, France,* pp. 1-6, 16-20 April 2007.

[92] K. Goossens, J. Dielissen et A. Radulescu, «The Æthereal network on chip: Concepts, architectures, and implementations,» *IEEE Design and Test of Computers* , vol. 22, n° %15, pp. 414-421, sept-oct 2005.

[93] T.-R. Halfhill, «Silicon Hive Breaks Out,» Microprocessor Report, Dec1, 2003.

[94] M.B. STENSGAARD et J.SPARSO, «Renoc: A network_on_chip architecture with reconfigurable topology,» *NoCs 2008. Second ACM/IEEE International Symposium on, Newcastle, UK,* pp. 55-64, 7-10 April 2008.

[95] T. PIONTECK, R. KOCH et C. ALBRECHT, «Applying partial reconfiguration to networks-on-chips,» *International Conference on Field Programmable Logic ans Applications, 2006. FPL'06, Madrid,Spain,* pp. 1-6, 28-30 August 2006.

[96] V. RANA, D. ATIENZA, M.D. SANTAMBROGIO, D. SCIUTO et G. DE MICHELI, «A Reconfigurable Network-on-Chip Architecture for Optimal Multi-Processor SoC Communication,» *16th IFPIP/IEEE International Conference on Very Large Scale Integration, Venue, Greece,* pp. 321-326, October 13-15, 2008.

[97] C. BOBDA, A. AHMADINIA, M. MAJER, J. TEICH, S. FEKETE et J. van der VEEN, «Dynoc: A dynamic infrastructure for communication in dynamically reconfigurable devices,» *in International Conference on Field Programmable Logic and Applications, Tampere, Finland,* pp. 153-158, August 24-26, 2005.

[98] M. MAJER, C. BOBDA, A. AHMADINIA et J. TEICH, «Packet Routing in Dynamically Changing Networks on chip,» *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado,* p. 154.b, April 4-8, 2005.

[99] Z. Gilles, «Sûreté de fonctionnement des systèmes industriels complexes Principaux concepts,» *Techniques de l'ingénieur Systèmes d'information et de communication,* Vols. %1 sur %2base documentaire : TIB397DUO, S8 250, 2009/06/10.

[100] V. A., Sûreté de fonctionnement des systèmes industriels, Collection : Direction des études et recherches d'Electricité de France (EDF), 01/03/1997 .

[101] M. Hosseinabady, A. Banaiyan, M.N. Bojnordi et Z. Navabi, «A concurrent testing method for noc Switches,» *in Proceeding of the conference on Design, automation and test in Europe: Proceeding, ser. DATE "06. European Design and Automation Association"*, pp. 1171-1176, 2006.

[102] S. Lin, W. Shen, C. Hsu, C. Chao et A. Wu, «Faukt-tolerant router with built-in self-test/self-diagnosis and fault-isolation circuits for 2d-mesh based chip multiprocessor systems,» *International Journal Of Electrical Engineering,* vol. 16, pp. 213-222, 2009.

[103] C.Morgan et P.Gardiner, «Data refinement of predicate transformers,» *In Theorical Computer Science,* vol. 87, n° %11, pp. 143-162, 16 september 1991.

[104] J.Rushby, «Formal Methods and the Certification of Critical Systems,» *Computer Science Laboratory, SRI International, Menlo Park CA 94025, USA, Technical Report CSL-93-7,* pp. 15-21, December 1993.

[105] R.Alur, «TECHNIQUE FOR AUTOMATIC VERIFICATION OF REAL-TIME SYSTEMS,» Thése de doctorat Université de Stanforf, August 1991.

[106] J. SPIVEY, The Z notation: A reference Manual, Second edition Prentice Hall International, 1992.

[107] O. MOSBAHI, «Développement formel des systèmes automatisés,» Thèse de doctorat, Université Tunis-El Manar, 2008.

[108] [En ligne]. Available: http://isabelle.in.tum.de. [Accès le 15 mars 2014].

[109] W. Bo, «GUI for model chekers,» Thesis Master of Science, Department of Computer Science Faculty of EEMCS, Delft University of Technology, Pays-Bas, June 2006.

[110] J.-R. Abrial, The B book : assigning programs to meanings, Cambridge University Press, 1996a.

[111] P. Behm, P. Benoit, A. Faivre et J-M. Meynadier, «Météor: A Successful Application of B in a Large Project,» *Proceedings of FM'99-Formal Methods, Toulouse, France,* Vols. %1 sur %21, LNCS1708, pp. 369-387, 20-24 Septembre 1999.

[112] E. BÖRGER, «The Origines and the Development of the ASM Method for High Level System Design and Analysis,» *In Journal of Universal Computer Science ,* vol. 8, n° %11, January2002.

[113] D. Canssel et D. Mery, «Foundations of the B Method,» *Computing and Informatics,* vol. 22, pp. 1-31, 2003.

[114] J.-R. Abrial, «Modeling in Event-B: System and Software Engineering,» *Cambridge University Press,* 2010.

[115] Clearsy, «Proof Obligations MANUEL DE PREFERENCE,» *France,* 2007.

[116] B. Mermet et D. Mery, «Incremental Specification of Telecommunication Services,» *Proceedings of First IEEE International Conference on Formal Engineering Methods Hiroshima, Japan,* pp. 60-69, 12-14 Nov 1997.

[117] A. Matoussi, «Construction de spécification formelles abstraites dirigée par les buts Ordinateur et société,» Thèse de doctorat. Université Paris-Est, 2011.

[118] M.Jastram, Rodin User's Handbook, 2012.

[119] Y. Métivier, J.M. Robson, N. Saheb-Djahromi et A. Zemmari, «An analysis of an optimal bit complexity randomised distributed vertex xolouring algorithm (extended abstract),» *OPODIS,* pp. 359-364, 2009.

[120] L. Lamport, Specifying Systems The TLA+ Language and Tools For Hardware and Software Engineers, Version of 18 June 2002.

[121] M. Jastram, Rodin User's Handbook, 2012.

[122] I.Sayar et M-T. Bhiri, «From an abstract specification in event-B toward an UML/OCL model,» *Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering, FormaliSE 2014, Hyderabad, INDIA,* pp. 17-23, June 3, 2014.

[123] J.-R. Abrial, «Summary of Event-B Proof Obligations,» *Department of Computer Science Swiss Federal Institute of Technology Zürich (ETH Zürich), Bucharest DEPLOY 2-day Course ,* 14-16

July 2010.

[124] Olivier Ligot, Jens Bendisposto et Michael Leuschel, «Debugging Event-B Models using the ProB Disprover Plug-in,» EU funded research projects: IST 511599 RODIN, technical repport for deploy project, AFADL 2007, pp 1-4.

[125] J. Bendisposto, M. Leuschel, O. Liegot et S. Mireille, «La validation de modèles Event-B avec le plug-in ProB pour RODIN,» Deploy technical repoort " RSTI-TSI-27/2008.AFADL 2007", 2008, pp 1065-1084.

[126] P. Schnoebelen, «Vérification de Logiciels: Techniques et outils du model,» *ouvrage collectif, coordination P. Schnoebelen, Vuibert, ISBN 2-, Paris,* 1999.

[127] J.-D. Owens, W-J. Dally, R. Ho, D-N-J. Jayasimba, S-W. Keckler et L-S. Peh, «Research challenges for on-chip interconnection networks,» *IEEE micro,* vol. 5, pp. 96-108, 2007.

[128] P. Guerrier et A; Greiner, «A Scalable Architecture for System-on-Chip Interconnections,» *in: Sophia Antipolis Forum on MicroElectronics (SAME99), Sophia Antipolis France,* pp. 90-93, 1999.

[129] C.-J. Glass et L-M Ni, «The turn model for adaptive routing,» *Journal of the ACM (JACM),* vol. 41 Issue5, pp. 874-902, 1994.

[130] T. E. (. o. S. The Designer's Guide to VHDL, The Designer's Guide to VHDL, Third Edition (Systems on Silicon), May 29, 2008.

[131] P. P. Chu, FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version, 14 mars 2008 .

[132] C. Grecu, A.Ivanov, R. Saleh, E.r Sogomonyan et P. Pande, «On-line Fault Detection and Location for NooC Interconnects,» *in Proceedings of the International On-Line Testing Symposium, IOLTS'06 Lake of Como, Italy,* pp. 1-8, July 10-12, 2006.

[133] W.-J. Dally et C-L. Seitz, «The torus routing chip,» *Distributed computing,* vol. 1 Issue 4, pp. 187-196, 1986.

[134] C. Killian, C. Tanougast, F. Monterio et A. Dandache, «A New Efficient and Reliable Dynamically Reconfigurable Network-on-Chip,» *Journal of Electrical and Computer Engineering, special issue Design and Automation for Integrated Circuits ans Systems,,* vol. 2012 Article ID 843239, p. 16.

[135] M. Heil , C. Tanougast, K. Cheng et A. Dandache, «Wireless network for self-reconfigurable hardware nodes,» 2013.

[136] K. Duffy, N. O'Connell et A. Sapozhnikov, «Complexity analysis of a decentralised graph colouring algorithm,» *inf, Process, Lett,* pp. 107:60-63, July 2008.

[137] J. Mycielski, «Sur le coloriage des graphes,» *Colloq. Math.3,* pp. 161-162, 1955.

[138] B. R. Nickerson, «Graph colouring register allocation for processors with multi-register operands,» *in Proceeding of the ACM SIGPLAN 1990 conference on Programming language design and implementation, PLDI'90, New York, NY, USA,* pp. 40-52, 1990.ACM.

[139] Johannes Schneider et Roger Wattenhofer, «Anew technique for distributed symmetry breaking,» *in Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, PODC'10, New York, NY, USA,* pp. 257-266, 2010. ACM.

[140] Jennie C. Hansen, Marek Kubale et Lukasz Kuszner*Adam Nadolski, «Distributed largest-first algorithm for graph coloring,» *in Euro-Par,* pp. 804-811, 2004.

[141] Marek Kubale et Lukasz Kuszner, «A better practical algorithm for distributed graph coloring,» *in PARELEC,* pp. 72-75, 2002.

[142] Jérémie Chalipin, Emmanuel Godard, Yves Métivier et Akka Zemmari, «Autour des algorithmes distribués,» *LaBri-Laboratoire Bordelais de recherche en informatique,* 2011.

[143] Noga Alon et Nabil Kahale, «A spectral technique for coloring random 3-colorable graphs,» *SIAM J.Comput,* pp. 26:1733-1748, December 1997.

[144] L. Hervé, «Téléphone cellulaire-coloriages pour allocation de fréquences,» *La Recherche-N.390,* pp. 98-99, 2005.

[145] E-G. Villegas, E. Lopez-Aguilera, R. Vidal et Paradells J, «Effect of adjacent-channel interference in IEEE802.11 WLANs,» *Cognitive Radio Oriented Wireless Network and Cmmunications, 2007 CrownCom2007. 2nd International Conference,* pp. 118-125, August 2007.

[146] Mohammad Malkawi, Mohammad Al-Haj Hassan et Osama Al-Haj Hassan, «Anew exam scheduling algorithm using graph coloring,» *Int. Arab J; Inf. Technol,* pp. 5(1):80-86, 2008.

[147] A. Mello, L. Tedosco, N. Calazans et F. Moraes, «Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC,» *in Proc. the Symposium on Integrated Cricuits and Systems Design, Florianoôlis, brazil,* pp. 178-183, 4-7 Sep 2005.

[148] C. Killian, C. Tanougast, F. Monterio et A. Dandache, «Online routing fault detection for reconfigurable NoC,» *in International Conference on Field Programmable Logic and Applications, Milano, Itally,* pp. 183-186, Aug 31- Sept2, 2010.

[149] M.Andriamiarina, **H. Daoud** ,M. Belarbi, D.Méry.and C,Tanougast «Formal Verification of Fault Tolerant NoC-based Architecture,» *First International Workshop on Mathematics and Computer Science (IWMCS2012),* n° hal-00763092, 2012.

[150] **Hayat Daoud**, Camel Tanougast, Mostefa Belarbi, Mikael Heil*: Formal specification and verification of wireless networked self-organized Systems on Chip.* CoDIT 2014: 730-735

[151] **Hayat Daoud**, Camel Tanougast, Mostefa Belarbi, Mikael Heil, Camille Diou*: Formal Proof of the Dependable Bypassing Routing Algorithm Suitable for Adaptive Networks on Chip QnoC Architecture.* Systems 5(1): 17 (2017).