UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UBO Université de Bretagne Occidentale

Brest MÉTROPOLE

THESE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Mourad DRIDI

Vers le support des systèmes à criticité mixte sur des architectures *NoC*

Thèse présentée et soutenue à Brest, le 18 Octobre 2019
Unité de recherche : Lab-STICC UMR CNRS 6285

Rapporteurs avant soutenance :

Abdoulaye GAMATIE	Directeur de Recherche, CNRS, LIRMM
Claire PAGETTI	Maître de Recherche, ONERA, DTIM

Composition du Jury :

Président	:	Eric GRESSIER-SOUDAN	Professeur des Universités, CNAM
Examinateurs :		António CASIMIRO	Maître de Conférences, Université de Lisbonne
		Abdoulaye GAMATIE	Directeur de Recherche, CNRS, LIRMM
		Claire PAGETTI	Maître de Recherche, ONERA, DTIM
Dir. de thèse	:	Frank SINGHOFF	Professeur des Universités, UBO, Lab-STICC
Encadrant	:	Jean-Philippe DIGUET	Directeur de Recherche, CNRS, Lab-STICC
Encadrant	:	Stéphane RUBINI	Maître de Conférences, UBO, Lab-STICC

Remerciements

Pour commencer ce manuscrit, je tiens à remercier toutes les personnes qui ont contribué à divers degrés au bon déroulement de ce travail de recherche.

Je remercie tout d'abord Frank Singhoff, Professeur à l'Université de Bretagne Occidentale (UBO)/Lab-STICC, pour ses conseils toujours pertinents en tant que directeur de thèse. Je le remercie du temps qu'il a pu me consacrer, ainsi que de la confiance et du soutien qu'il a su m'accorder pour mener à bien ce manuscrit.

Je remercie vivement Stéphane Rubini, Maitre de conférences à l'Université de Bretagne Occidentale (UBO)/Lab-STICC et Jean-Philippe Diguet Directeur de recherche CNRS/Lab-STICC pour leur encadrement, leur expertise et leur soutien pendant la préparation de cette thèse. C'était un plaisir de travailler avec eux.

Je remercie particulièrement Abdoulaye Gamatié Directeur de recherche CNRS/-LIRMM, et Claire Pagetti ingénieur de recherche ONERA/DTIM pour avoir été rapporteurs de cette thèse ; et également Antònio Casimiro Maître de conférences à l'Université de Lisbonne/Lasige d'être examinateur.

Je remercie particulièrement Eric Gressier-Soudan Professeur des Universités (CNAM)/Cédric d'être président du jury.

Mes remerciements vont vers Mounir Lallali pour son aide et ses conseils, et aussi Laurent Lemarchand et Valérie Marc pour nos discussions scientifiques.

Un remerciement particulier à Martha Johanna pour m'avoir accueilli dans son équipe durant mon séjour scientifique à l'Université Technique de Munich (TUM).

Je tiens de plus à saluer tous mes collègues (docteurs ou doctorants) au Lab-STICC avec qui j'ai partagé cette expérience de recherche : Rahma, Baccouri, Chabha, Antoine, Zerkane, Camélia, Steven, Valery, Nam, Arwa, Mayssa, Illham et Musaab.

Enfin, mes sentiments les plus chaleureux sont pour ma famille. En particulier, je remercie mes parents pour leur soutien et encouragement depuis toujours. Je voudrais également remercier mon frère Seif du soutien inconditionnel qu'il m'a apporté. Une pensée très chaleureuse à mes sœurs Chayma, Rim et Monia pour leur amour et encouragement. Un grand remerciement également à mon âme soeur Manel pour sa présence à mes côtés durant toutes ces années.

Bonne lecture !

Publications

Journaux internationaux

1. Mourad Dridi, Stéphane Rubini, Mounir Lallali, Martha Johanna Sepúlveda Flórez, Frank Singhoff and Jean-Philippe Diguet. 2019. Design and Multi-Abstraction-Level Evaluation of a NoC Router for Mixed-Criticality Real-Time Systems. ACM Journal on Emerging Technologies in Computing Systems. Article 2 : 1-37, February 2019. (**Rang Q2**)

2. Mourad Dridi, Stéphane Rubini, Frank Singhoff, Jean-Philippe Diguet. DTFM : a Flexible Model for Schedulability Analysis of Real-Time Applications on NoC-based Architectures. ACM SIGBED Review 14(4) : 53-59 (2017). Special issue of the 4^{th} International Workshop on Real-Time Computing and Distributed systems in Emerging Applications (REACTION).

3. José Rufino, António Casimiro, Antónia Lopes, Frank Singhoff, Stéphane Rubini, Valérie-Anne Nicolas, Mounir Lallali, Mourad Dridi, Jalil Boukhobza, Lyes Allache. NORTH : Non-intrusive Observation and RunTime verification of cyber-pHysical systems. Ada User Journal, 2018 vol 39, no 4

Conférences ou workshops internationaux

1. Mourad Dridi, Frank Singhoff, Stéphane Rubini, Jean-Philippe Diguet. ECTM : A New Communication Model to Network-On-Chip Schedulability Analysis. 24th International Conference on Reliable Software Technologies – Ada-Europe 2019, June 2019, Varsow, Poland. (**Rang B**)

2. Mourad Dridi, Stéphane Rubini, Mounir Lallali, Martha Johanna Sepulveda Florez, Frank Singhoff, Jean-Philippe Diguet. DAS : An Efficient NoC Router for Mixed-Criticality Real-Time Systems. 35th IEEE International Conference on Computer Design (ICCD) 2017, 229-232. Boston, MA, USA.

3. Mourad Dridi, Mounir Lallali, Stéphane Rubini, Jean-Philippe Diguet, Frank Singhoff. Modeling and Validation of a Mixed-Criticality NoC Router Using the IF Language. 10th International Workshop on Network-On-Chip Architectures (NoCArch), Boston, MA, USA.

Séminaires, exposés

1. Mourad Dridi, Stéphane Rubini, Frank Singhoff, Jean-Philippe Diguet. NoC and Mixed-criticality Systems. GDR SOC2, Emerging Interconnect Technologies in ManyCore architectures, Paris, cité internationale – Collège d'Espagne. November 2017.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Dans le contexte des systèmes temps réel, le respect des contraintes temporelles est aussi important que l'exactitude du résultat. Autrement dit, la validité d'un système temps réel ne dépend pas seulement des résultats logiques des traitements mais également de la date de livraison du résultat [1]. Aujourd'hui, les systèmes temps réel sont de plus en plus présents autour de nous. Ils sont présents dans de nombreux secteurs d'activités comme les transports (contrôleurs pour les avions, les voitures, . . .), les systèmes de détection (radars, sonars, . . .), et le multimédia (lecteurs vidéo, téléphones, . . .).

Suivant l'importance accordée aux contraintes temporelles, nous distinguons les systèmes temps réel strict ou dur et les systèmes temps réel mou ou souple [1]. Dans les systèmes temps réel dur, nous devons respecter les contraintes temporelles même dans la pire des situations d'exécution possibles. Un dépassement d'échéance peut conduire à des situations critiques, voire catastrophiques [2]. Le contrôle de commande d'un avion et les applications de surveillance sont des exemples d'applications temps réel dur. Contrairement aux systèmes temps réel dur, les systèmes temps réel mou peuvent tolérer certains dépassements d'échéance [2]. Les applications de visioconférence et les jeux en réseau sont des exemples de systèmes temps réel mou.

Ces dernières années, il y a eu un intérêt accru pour l'intégration de plusieurs applications temps réel sur la même plate-forme d'exécution dans le but de réduire le coût, le poids et la consommation d'énergie. Cette intégration a conduit à la conception des systèmes à criticité mixte. Dans la communauté temps réel, les systèmes à criticité mixte sont constitués d'applications avec des niveaux de criticité différents qui s'exécutent sur la même plateforme d'exécution [3, 4].

La conception d'architectures multiprocesseurs est une autre tendance qui a suscité une attention toute particulière ces dernières années. Les réseaux sur puce

(ou en anglais Network-on-Chip (NoC)) sont devenus un composant essentiel des architectures multiprocesseurs. Son adoption a été motivée par la nécessité de trouver une alternative aux communications par bus qui ne peuvent offrir le débit requis par un nombre croissant de processeurs, de mémoires et autres composants devant échanger concurremment des données. Un NoC est une méthode d'interconnexion qui simplifie l'intégration de composants complexes sur des systèmes sur puce (ou en anglais System-on-Chip (SoC)) [5] et permet de résoudre, dans une certaine mesure, le problème de saturation constaté avec les systèmes à base de bus. Les NoC présentent de nombreux avantages tels que l'augmentation des débits d'échanges, la réduction des temps de latence, l'extensibilité et la flexibilité [5].

Au regard de ces avantages, l'intégration des systèmes à criticité mixte sur des architectures NoC présente une solution prometteuse en termes de performance, coût, taille, poids et puissance [6].

Nous supposons dans ce travail que l'exécution des systèmes à criticité mixte sur des architectures NoC exige l'assurance des contraintes temporelles pour les applications critiques tout en minimisant l'impact du partage de ressources sur les applications non critiques.

Nous nous intéressons dans le cadre de ce travail au challenge consistant à déployer des systèmes à criticité mixte sur des architectures NoC. Dans la suite, nous détaillons les problématiques étudiées dans le cadre de cette thèse.

1.1 Motivations et objectifs

Nous pouvons rencontrer plusieurs obstacles lors du déploiement des systèmes à criticité mixte sur les architectures NoC. Ces obstacles sont principalement :

1. **L'incompatibilité des routeurs NoC existants avec les exigences des systèmes à criticité mixte**

   Le partage de ressources dans un NoC tel que les liens physiques et les routeurs introduit des interférences et des délais de communications supplémentaires, ce qui complique l'analyse d'ordonnancement du système.

   Ces dernières années, divers routeurs pour les systèmes temps réel ont été proposés pour les architectures NoC. La plupart de ces routeurs sont capables de réduire et borner le temps de communication et de satisfaire les exigences temporelles pour les applications temps réel dur et mou [5].

   Toutefois, pour déployer des systèmes à criticité mixte sur des architectures NoC, nous avons besoin d'un routeur NoC qui assure à la fois les contraintes

temporelles des communications critiques et limite la réservation des ressources pour les communications non critiques.

L'incompatibilité des routeurs NoC existants avec les exigences des systèmes à criticité mixte est une des raisons du non déploiement de ce genre du système sur les architectures NoC [7].

2. **Des modèles de tâches et de flux incomplets**

   Les modèles de tâches et les modèles de flux existants dans la littérature ne sont pas applicables immédiatement avec les architectures NoC. De nombreux modèles de tâches et de flux ont été proposés pour traiter les applications temps réel [8, 6] pour les NoC. Toutefois, les modèles de tâches proposés ne prennent pas en compte les communications à travers le NoC, l'affectation des tâches et les latences introduites par ce nouveau type de ressources partagées [6]. De même, les modèles de flux proposés pour les NoC ignorent l'ordonnancement des tâches [8]. Ainsi, ces modèles ne sont pas suffisants pour modéliser intégralement un système temps réel déployé sur un NoC.

3. **Absence d'analyse des communications dans les techniques d'analyses d'ordonnancement temps réel**

   Les architectures NoC introduisent des nouvelles interférences variables en fonction de la configuration du NoC, de l'affectation des tâches et de l'état du réseau [5, 9]. Les délais de communication produits suite à ces nouvelles interférences doivent être considérés dans l'analyse d'ordonnancement du système.

   Toutefois, les techniques d'analyse d'ordonnancement existantes ignorent ces interférences [10]. Ne pas tenir compte des interférences introduites par les architectures NoC fausse l'analyse d'ordonnancement du système.

1.2 Contributions

Pour résoudre les problèmes mentionnés précédemment, nous avons proposé plusieurs contributions sous la forme d'un routeur, de modèles de tâches, de flux et de communications pour les NoC.

Nous avons proposé un nouveau routeur NoC dans le but d'exécuter des systèmes à criticité mixte sur un NoC. Afin de prédire l'ordonnancement des systèmes à criticité mixte sur un NoC, nous avons proposé un modèle de tâches et de flux ainsi qu'un modèle de communication. Dans la suite, nous détaillons nos contributions.

- **DAS, un routeur pour les systèmes à criticité mixte**

  Afin de répondre à la première problématique, nous avons proposé un routeur NoC appelé *Double Arbiter and Switching Router* (DAS). DAS est un routeur conçu pour déployer un système temps réel à criticité mixte sur des architectures NoC. DAS assure les contraintes temporelles pour les communications critiques tout en limitant la réservation des ressources par les communications critiques. Afin de répondre aux exigences des systèmes à criticité mixte, DAS implante deux modes de fonctionnement, deux niveaux de préemption, deux techniques de contrôle de flux et deux étages d'arbitrage.

  Nous avons évalué le routeur proposé sur 3 niveaux d'abstraction différents (circuit, transaction et système).

- **Dual Task And Flow Model (DTFM)**

  Afin de répondre à la deuxième problématique, nous avons proposé DTFM. DTFM est une méthode conçue pour modéliser les systèmes temps réel déployés sur des architectures NoC. À partir du modèle de tâches, du modèle de NoC et du placement des tâches, DTFM calcule automatiquement le modèle de flux correspondant. DTFM prend en compte la technique de commutation adoptée par le NoC. Il supporte les techniques Wormhole [11] et Store And Forward (SAF) [12].

- **Exact Communication Time Model (ECTM)**

  Pour tenir compte des communications lors de l'analyse d'ordonnancement des tâches, nous avons proposé ECTM. ECTM est un modèle de communication pour les architectures NoC. Nous montrons qu'ECTM peut conduire à une analyse d'ordonnancement efficace. Il supporte les techniques Wormhole et Store and Forward. Il modélise les communications par des graphes de tâches tout en tenant compte du modèle de NoC utilisé.

  Afin de répondre à la troisième problématique, nous proposons de combiner DTFM et ECTM. D'abord, DTFM calcule le modèle de flux. Ensuite, ECTM applique ses transformations sur le modèle de flux calculé par DTFM afin de conduire une analyse d'ordonnancement avec des méthodes classiques de la littérature.

Nous avons utilisé DTFM lors de l'évaluation de DAS. Par contre, le modèle de communication ECTM n'est pas directement applicable au routeur proposé. En effet, DAS requiert un modèle de communication supportant conjointement SAF et Wormhole alors que nous avons proposé ECTM pour ces deux modes de commutation séparément. L'adaptation d'ECTM pour les caractéristiques de DAS est l'un de nos futurs travaux pour cette thèse.

1.3 Plan

Ce document est divisé en huit chapitres. Les deux premiers chapitres exposent les notions de base nécessaires à la compréhension de nos travaux ainsi que des travaux connexes aux nôtres. Le chapitre 2 présente les systèmes à criticité mixte. Le chapitre 3 présente les architectures NoC.

Nous discutons des motivations, des objectifs et des contributions de cette thèse dans le chapitre 4. Ensuite, les chapitres 5, 6 et 7 présentent les contributions de cette thèse. Dans le chapitre 5, nous détaillons l'architecture et le fonctionnement de DAS. Puis, dans le chapitre 6, nous discutons de l'évaluation de DAS sur plusieurs niveaux d'abstraction. Nous décrivons DTFM et le modèle de communication pour les architectures NoC proposées dans le chapitre 7. Finalement, nous terminons cette thèse par une conclusion.

Première partie

Contexte - État de l'art

2

Les systèmes temps réel à criticité mixte

Sommaire

Introduction

Exécuter des systèmes à criticité mixte sur des architectures multiprocesseurs est une solution prometteuse en terme de performance de calcul et de consommation énergétique [6]. L'ordonnancement des systèmes à criticité mixte sur des architectures multiprocesseurs a connu un regain d'intérêt ces dix dernières années ce qui s'est traduit par un grand nombre de propositions [6].

La première partie de ce chapitre présente les concepts de base de l'analyse d'ordonnancement des systèmes à criticité mixte. Nous présentons brièvement les systèmes temps réel. Ensuite, nous discutons de la définition des systèmes à criticité mixte. Nous expliquons aussi le rôle des algorithmes d'ordonnancement et les différentes techniques d'analyse d'ordonnancement.

Dans la deuxième partie de ce chapitre, nous proposons une taxonomie des algorithmes d'ordonnancement multiprocesseurs. Puis, nous présentons un état de l'art des algorithmes d'ordonnancement temps réel multiprocesseurs pour les tâches dépendantes. Ensuite, nous abordons les algorithmes multiprocesseurs qui s'intéressent aux systèmes à criticité mixte.

2.1 Systèmes temps réel

Définition 1. ***(Système temps réel)***

Un système temps réel est défini comme un système dont le comportement dépend de l'exactitude des traitements effectués et de la date où les résultats sont produits [1].

En d'autres termes, dans le contexte d'un système temps réel, nous devons satisfaire deux contraintes importantes :

- **Exactitude logique :** le système produit des résultats logiquement corrects.
- **Exactitude temporelle :** le système produit les résultats au bon moment (avant les échéances du système).

Les systèmes temps réel sont de plus en plus présents autour de nous [1]. Nous pouvons les trouver dans de nombreux domaines : aéronautique, militaire, robotique, télécommunication, etc.

Les systèmes temps réel sont généralement classés en plusieurs catégories suivant le niveau de sûreté temporelle demandé. Les deux catégories principales sont les systèmes temps réel dur (ou en anglais HARD REAL-TIME SYSTEM) et les systèmes temps réel mou (ou en anglais SOFT REAL-TIME SYSTEM) [13].

Dans la suite, nous définissons les systèmes temps réel dur et mou.

2.1.1 Systèmes temps réel dur

Définition 2. *(**Système temps réel dur**)*

Les systèmes dits temps réel dur sont ceux pour lesquels le système doit impérativement garantir le respect des échéances fixées pour l'exécution des tâches [13].

Le respect de toutes les contraintes temporelles est indispensable pour le bon fonctionnement d'un système temps réel dur. Dans le contexte d'un système temps réel dur, une échéance non respectée peut endommager le système et conduire à des situations catastrophiques. Un système temps réel dur doit impérativement respecter toutes les contraintes temporelles même dans la pire des situations d'exécution possibles [2]. Le pilote automatique d'un avion et le système de surveillance d'une centrale nucléaire sont des exemples des systèmes temps réel dur.

2.1.2 Systèmes temps réel mou

Définition 3. *(**Système temps réel mou**)*

Un système temps réel mou est un système où l'on tolére que certaines contraintes temporelles ne soient pas satisfaites. Ces violations peuvent dégrader l'expérience utilisateur du système sans compromettre le fonctionnement global [2].

Contrairement au système temps réel dur, le système temps réel mou peut tolérer un certain nombre de dépassements d'échéance sans que cela ait des conséquences catastrophiques. Les applications multimédia sont des exemples de systèmes temps réel mou [2].

Dans la suite nous présentons les concepts classiques des modèles de tâches pour un système temps réel.

2.1.3 Modèle de tâches

Définition 4. *(**Tâche**)*

Une tâche est un ensemble d'instructions destinées à être exécutées sur une unité de calcul [14].

La caractérisation d'une tâche peut varier d'un modèle de tâches à un autre.

Divers modèles de tâches ont été proposés [6]. Dans la suite, nous introduisons la notion de tâche périodique. Puis, nous détaillons les paramètres les plus utilisés dans les modèles de tâches classiques. Ensuite, nous discutons des dépendances.

2.1.3.1 Tâche périodique, apériodique et sporadique

Dans cette partie, nous définissons les tâches périodiques, les tâches sporadiques et les tâches apériodiques.

Définition 5. *(**Tâche périodique**)*

Une tâche périodique est une tâche dont le réveil est régulier et le délai entre deux réveils successifs est constant [14].

Définition 6. *(**Tâche sporadique**)*

Une tâche sporadique est une tâche caractérisée par un délai minimum entre deux réveils successifs [14].

Définition 7. *(**Tâche apériodique**)*

Les tâches apériodiques sont réveillées à des instants aléatoires. Elles sont généralement activées sur un événement extérieur [14].

Les tâches périodiques et les tâches sporadiques se répètent indéfiniment. Les instances de réveil d'une tâche périodique sont séparées par une période constante. Par contre, les tâches sporadiques se caractérisent par un délai minimum entre deux réveils. Contrairement aux tâches périodiques et sporadiques, les tâches apériodiques s'exécutent une seul fois.

2.1.3.2 Caractéristiques d'une tâche temps réel

Nous considérons un ensemble de n tâches $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Chaque tâche τ_i est caractérisée par :

- O_i (Offset) : date de premier réveil d'une tâche τ_i. Elle représente la date à laquelle la tâche τ_i peut commencer son exécution.
- C_i (Capacity) : capacité de la tâche τ_i. Elle représente la durée d'exécution d'une tâche τ_i. Dans la majorité des travaux d'ordonnancement temps réel, ce paramètre est considéré comme le pire temps d'exécution (ou en anglais Worst Case Execution Time (WCET)) [6].
- D_i (Deadline) : échéance de la tâche τ_i. Elle représente l'instant auquel l'exécution d'une tâche doit être terminée et dont le dépassement provoque une violation de la contrainte temporelle.
- P_i (Period) : période qui représente la durée séparant deux instants de réveils successifs.
- π_i (Priority) : priorité de la tâche.

Tous ces paramètres sont illustrés dans la figure 2.1.

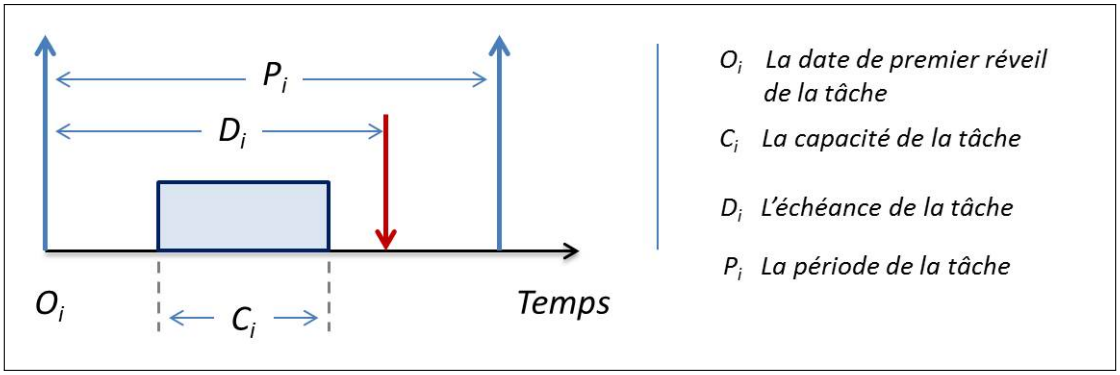


Figure 2.1 – Paramètres d'une tâche périodique.
Une tâche périodique τ_i est représentée par cinq paramètres : une date de premier réveil O_i, une capacité C_i, une échéance D_i, une période P_i et une priorité π_i.

Dans la suite, nous discutons des contraintes de dépendance et de précedence entre les tâches.

2.1.3.3 Dépendance

Définition 8. *(**Précédence**) Une précédence désigne un ordre partiel d'exécution des tâches sans qu'il y ait nécessairement un transfert de données entre ces tâches [15].*

Définition 9. *(**Dépendance**)*

Une dépendance désigne un ordre partiel d'exécution des tâches avec un transfert de données entre ces tâches [15].

En d'autres termes, la dépendance entre deux tâches impose une précédence entre la tâche qui produit des données et celle qui en consomme. Nous appelons la tâche qui produit les données la tâche source et la tâche qui consomme les données la tâche destination [15].

Plusieurs modèles de tâches ont été proposés pour les tâches dépendantes. Les dépendances entre les tâches peuvent être modélisées par des graphes acycliques orientés (ou en anglais Directed Acyclic Graph (DAG)) [16].

Définition 10. *(**Directed Acyclic Graph (DAG)**)*

Un DAG est un graphe $G = (\Gamma , \Psi)$, orienté et acyclique, où Γ représente les tâches, et Ψ les dépendances entre les tâches [16].

La figure 2.2 présente un exemple de DAG.

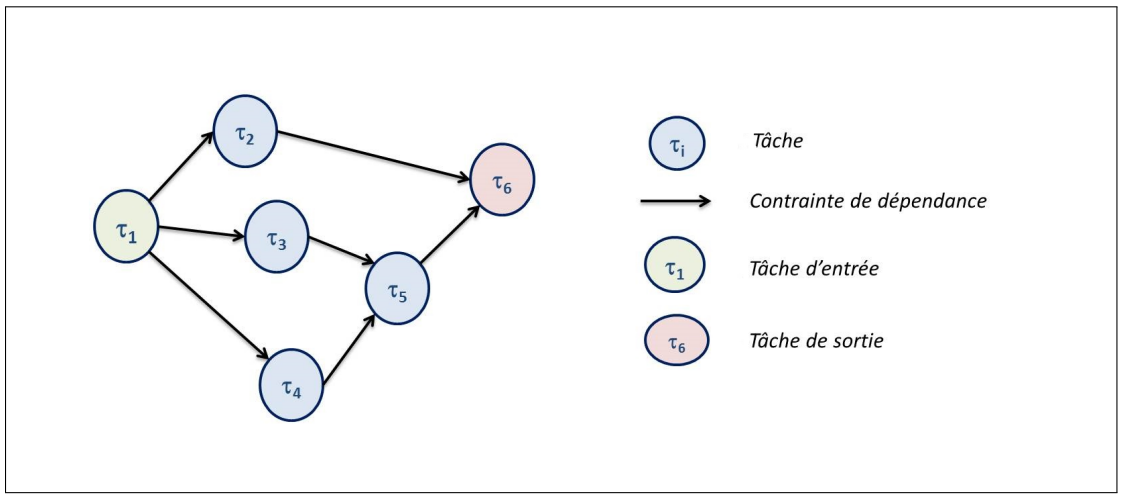


Figure 2.2 – Exemple d'un DAG.
Dans un DAG, les nœuds représentent les tâches et les arcs représentent les relations de dépendance entre les tâches. Une tâche d'entrée (respectivement de sortie) dans un DAG est une tâche n'ayant pas de prédécesseur (respectivement de successeur) dans le graphe.

2.1.4 Modèle de support d'exécution

Après avoir présenté les modèles de tâches temps réel, nous décrivons dans cette partie le support d'exécution pour ces systèmes. Nous pouvons distinguer deux types de supports d'exécution : les systèmes monoprocesseurs et les systèmes multiprocesseurs. Dans les systèmes multiprocesseurs, nous trouvons les réseaux sur puce.

Définition 11. *(**Système monoprocesseur**)*

Un système monoprocesseur fournit une seule unité de calcul pour exécuter les applications [5].

Définition 12. *(**Système multiprocesseur**)*

Un système multiprocesseurs fournit au même moment plusieurs unités de calcul pour exécuter les applications [5].

Contrairement aux systèmes monoprocesseurs, sur les supports multiprocesseurs, les applications disposent de plusieurs unités de calcul simultanément pour s'exécuter.

Définition 13. *(**Réseau sur puce**)*

Le réseau sur puce (en anglais Network-On-Chip (NoC)) est un paradigme de connexion entre les unités de calcul qui sont intégrées dans un système sur puce [17].

Dans un NoC, les messages sont acheminés de l'émetteur vers une ou plusieurs destinations via des routeurs [5]. Nous détaillerons l'architecture, le mode de fonctionnement et les caractéristiques des NoC dans le chapitre suivant.

Dans la suite, nous définissons les systèmes à criticité mixte.

2.2 Systèmes à criticité mixte

Aujourd'hui, les supports d'exécution sont de plus en plus performants et fournissent des capacités de calcul plus importantes qu'avant. En effet, les plateformes actuelles comportent plusieurs unités de calcul. Par la suite, les possibilités matérielles ont fortement augmenté et la consommation énergétique est devenue un enjeu primordial [6]. L'exécution d'une application par support d'exécution n'est plus rentable car l'unité de calcul est sous-utilisée.

Il est donc primordial de pouvoir tirer parti de ces capacités de calcul et de minimiser le gaspillage d'énergie. Afin de répondre aux besoins d'intégrer des applications avec des niveaux de criticité différents sur le même support d'exécution, les systèmes temps réel à criticité mixte ont été proposés.

Dans le domaine avionique, nous pouvons trouver des applications avec des niveaux de criticité différentes. Les normes de sécurité et les pratiques de l'industrie avionique ont une définition particulière des systèmes à criticité mixte [18, 19, 20]. Nous pouvons citer à titre d'exemple les normes CEI 61508, DO-178B et ISO 26262 [21]. Dans ces normes industrielles, une isolation spatiale et temporelle entre les fonctions de niveaux de criticité différents est requis.

Cependant, la définition de système à criticité mixte proposée pour la première fois par Vestal [3] et ultérieurement par Burns et Davis [4] précise que les plates-formes d'exécution doit fournir des modes de fonctionnement différents pour les applications avec des niveaux de criticité différents [3]. Cette définition fait que le défi du MCS est de garantir les contraintes temporeeles des applications critiques, tout en minimisant l'impact du partage de ressources sur les applications non critiques.

Définition 14. *(**Systèmes temps réel à criticité mixte**)*

Les systèmes temps réel à criticité mixte sont des systèmes temps réel ayant la particularité d'avoir plusieurs modes de criticité (au moins deux) et des tâches avec des niveaux de criticité différents (au moins deux niveaux) [3, 4, 22].

Dans les systèmes ayant deux niveaux de criticité, nous distinguons deux types de tâches : les tâches critiques et les tâches non critiques.

Définition 15. *(**Tâches critiques**)*

Les tâches critiques (ayant un niveau de criticité élevé) sont des applications temps réel dur. Les échéances de ces applications doivent être respectées [3].

Définition 16. *(**Tâches non critiques**)*

Les tâches non critiques (ayant un niveau de criticité faible) sont des applications temps réel mou. Elles peuvent tolérer certains dépassements de leurs échéances [3].

En outre, un système à criticité mixte doit fonctionner sous plusieurs modes de criticité (au minimum deux modes) selon l'état du système [22].

Définition 17. *(**Mode de criticité**)*

Dans le contexte des systèmes à criticité mixte, le mode de criticité définit les paramètres d'exécution du système tels que les capacités et les périodes des tâches [22].

Dans cette partie, nous présentons le modèle des systèmes à criticité mixte proposé par Vestal [3]. Puis, nous décrivons le mécanisme de changement de mode.

2.2.1 Modèle de tâches

D'après Vestal [3], un système est composé d'un ensemble de n tâches périodiques ($\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$) ayant k niveaux de criticité différents [3]. Nous notons ici que le système à criticité mixte fonctionne sous j modes de criticité avec $j \geq 2$ et $k \geq 2$.

Chaque tâche τ_i de l'ensemble Γ est caractérisée par un ensemble de paramètres : $\tau_i = \{ O_i, C_i(a), D_i\ P_i, Crit_i \}$

avec :

- O_i représente la date du premier réveil de la tâche τ_i.
- $C_i(a)$ représente le pire temps d'exécution de la tâche τ_i sous le mode de criticité a avec 1 ≤ a ≤ j. Nous notons ici que C_i est une fonction. La valeur de C_i est déterminée en fonction du niveau de criticité de la tâche et du mode de criticité actuel du système [3].

TABLE 2.1 – Capacité d'une tâche pour un système à criticité mixte.
La tâche τ_i possède plusieurs valeurs de capacité en fonction du mode de criticité. $C_i(1)$ est la capacité de τ_i dans le mode 1. $C_i(j)$ est la capacité de τ_i dans le mode j.

Mode de criticité	**Mode 1**	$\cdots$	**Mode a**	$\cdots$	**Mode j**
C_i	$C_i(1)$	$\cdots$	$C_i(a)$	$\cdots$	$C_i(j)$

- D_i représente l'échéance de la tâche τ_i.
- P_i représente la période de la tâche τ_i.
- $Crit_i$ représente le niveau de criticité de la tâche τ_i.

  Aucune violation d'échéances n'est autorisée pour les tâches ayant un niveau de criticité élevé. Cependant, nous pouvons tolérer certains dépassements d'échéances pour les tâches ayant un niveau de criticité plus faible.

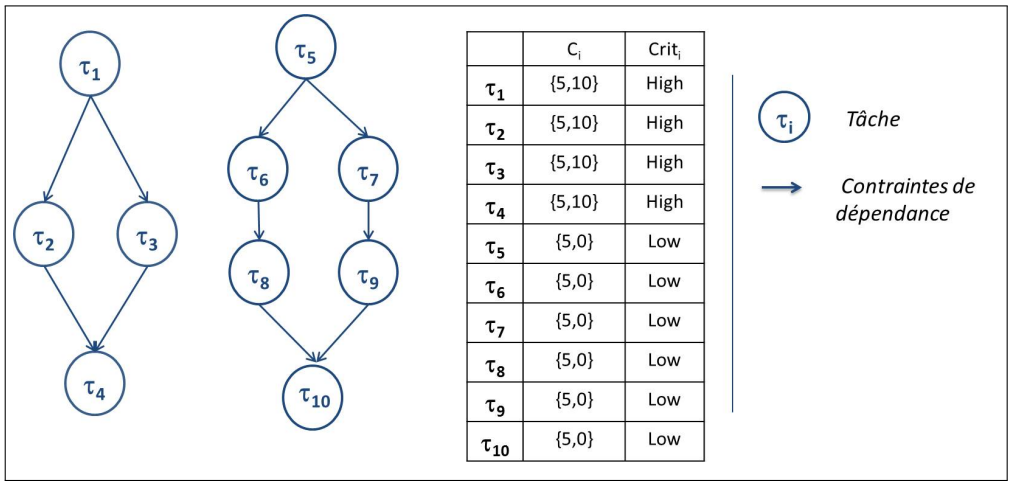
2.2.2 Changement de mode de criticité

La modélisation d'un système à criticité mixte proposée par Vestal [3] est caractérisée par plusieurs modes de criticité. Le système peut fonctionner sous différents modes de criticité. Pour chaque mode, la capacité de chaque tâche prend une valeur spécifique [22] (voir Table 2.1). Le système définit la capacité des tâches afin de représenter les différents niveaux d'exigences des autorités de certification [3, 22].

Généralement un système caractérisé par n niveaux de criticité peut fonctionner sous n modes de criticité. Le système est par défaut dans le mode de criticité le

plus faible. Lorsque les tâches de haute criticité risquent de ne pas respecter leur échéance, le système passe du mode de criticité courant vers un autre mode plus critique. Le système doit définir les seuils de changement de mode [3].

Lorsque le système se trouve dans un niveau de criticité donné, les protocoles de changement de mode font l'hypothèse que les échéances de toutes les tâches dont la criticité est plus faible que le niveau de criticité actuel ne sont plus garanties, et ce pour permettre de garantir les échéances des tâches ayant un niveau de criticité élevé.



	C_i	$Crit_i$
τ_1	{5,10}	High
τ_2	{5,10}	High
τ_3	{5,10}	High
τ_4	{5,10}	High
τ_5	{5,0}	Low
τ_6	{5,0}	Low
τ_7	{5,0}	Low
τ_8	{5,0}	Low
τ_9	{5,0}	Low
τ_{10}	{5,0}	Low

FIGURE 2.3 – Exemple d'un système à criticité mixte avec deux modes de criticité et deux niveaux de criticité.
Les tâches (τ_1, τ_2, τ_3, τ_4) sont des tâches ayant un niveau de criticité High. Les tâches (τ_5, τ_6, τ_7, τ_8, τ_9, τ_{10}) sont des tâches ayant un niveau de criticité Low. Chaque tâche possède deux valeurs de capacité. La première valeur est considérée sous le mode non critique. La deuxième valeur est considérée sous le mode critique.

La figure 2.3 présente un exemple de système à criticité mixte. Dans cet exemple, nous considérons deux niveaux de criticité : niveau de criticité élevé (noté High) et niveau de criticité faible (noté Low). En outre, nous considérons deux modes de criticité : mode critique et mode non critique. La figure 2.3 présente le graphe des tâches du système ainsi que la capacité et le niveau de criticité pour chaque tâche.

La figure 2.4 illustre le changement de mode pour le système considéré dans cet exemple. Dans le mode non critique, le système ordonnance les tâches High et Low. Cependant, dans le mode non critique, les tâches Low sont suspendues.

Dans la suite, nous discutons des différents méthodes d'analyse d'ordonnancement.

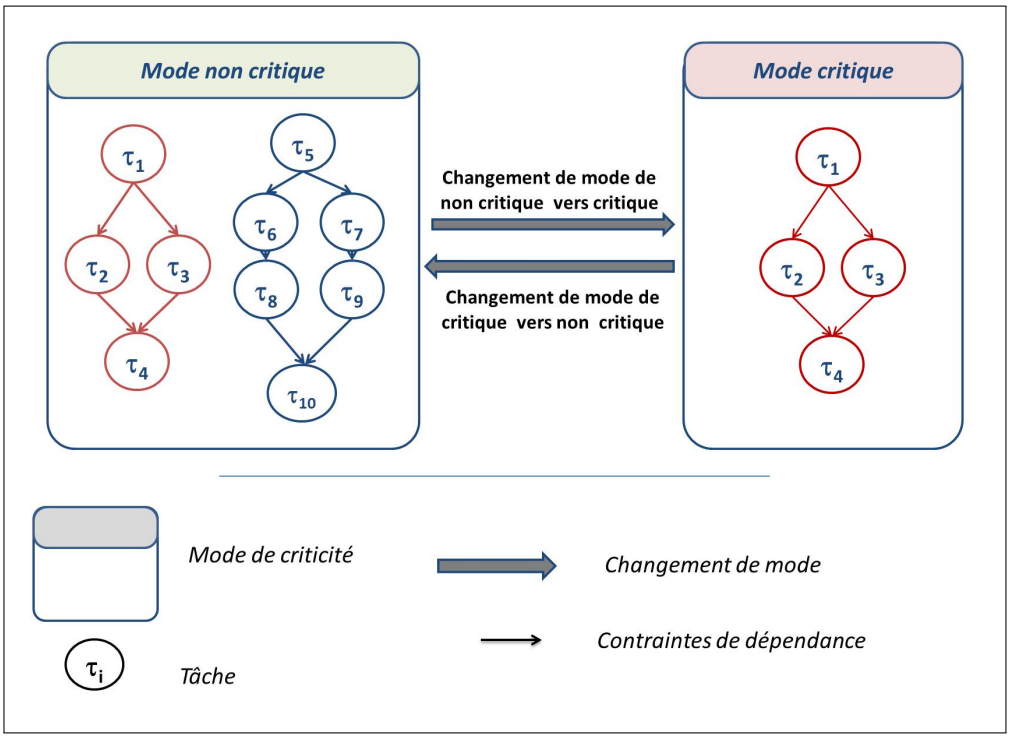


FIGURE 2.4 – Exemple d'un changement de mode de criticité.
Nous considérons dans cet exemple deux modes de criticité : un mode critique et un mode non critique. Le protocole de changement de mode définit les seuils de transition d'un mode vers un autre.

2.3 Introduction à l'analyse d'ordonnancement des systèmes temps réel

Les systèmes temps réel sont soumis à des contraintes temporelles. Ainsi, la problématique de l'analyse d'ordonnancement temps réel est principalement de prévoir avec le plus d'exactitude possible si ces contraintes temporelles peuvent être respectées.

Dans cette partie, nous définissons divers concepts associés à l'analyse d'ordonnancement temps réel. Ensuite, nous détaillons les différentes méthodes d'analyses pour les systèmes temps réel.

2.3.1 Algorithme d'ordonnancement

Définition 18. *(**Algorithme d'ordonnancement**) Un algorithme d'ordonnancement consiste à définir une allocation spatiale et temporelle des tâches sur les unités de calcul de sorte que les contraintes temporelles soient satisfaites [23].*

Le rôle principal d'un algorithme d'ordonnancement consiste à déterminer l'ordre total et les dates de démarrage d'exécutions des tâches sur une ou plusieurs unités de calcul.

Divers algorithmes ont été proposés selon le modèle de tâches, le modèle de support d'exécution et le type d'ordonnancement.

Dans la suite, nous définissons les différentes caractéristiques des algorithmes d'ordonnancement temps réel.

2.3.1.1 Ordonnancement avec ou sans préemption

Définition 19. *(**Ordonnancement préemptif**)*

Un ordonnanceur préemptif peut interrompre une tâche au profit d'une tâche plus prioritaire. Cette interruption s'appelle une préemption [23].

Définition 20. *(**Ordonnancement non préemptif**)*

Un ordonnanceur non préemptif n'a pas la capacité d'arrêter l'exécution de la tâche courante [23].

Un ordonnanceur préemptif a la capacité d'interrompre une tâche en cours d'exécution, d'en mémoriser l'état, et d'exécuter une autre tâche plus prioritaire. Dans le cadre d'un ordonnancement non préemptif, les préemptions des tâches ne sont pas autorisées.

2.3.1.2 Ordonnancement hors ligne ou en ligne

Définition 21. *(**Ordonnancement hors ligne**)*

Un algorithme d'ordonnancement hors ligne prend la totalité de ses décisions d'ordonnancement avant l'exécution du système [23].

Définition 22. *(**Ordonnancement en ligne**)*

Un algorithme d'ordonnancement en ligne prend les décisions d'ordonnancement lors de l'exécution et dispose par conséquent d'avantage d'informations sur les temps d'exécution des tâches du système [23].

L'avantage d'une approche hors ligne est un sur-coût minimal lors de l'ordonnancement. Cette approche exige la connaissance de tous les paramètres avant l'exécution [23].

L'avantage d'une approche en ligne est la capacité d'adapter l'ordonnancement aux changements de l'environnement.

2.3.1.3 Ordonnancement global ou par partitionnement

Dans le contexte d'algorithmes d'ordonnancement multiprocesseurs, nous distinguons deux grandes familles : les algorithmes globaux et les algorithmes partitionnés.

Définition 23. *(**Ordonnancement global**)*

Sur un système comprenant m unités de calcul, le rôle d'un algorithme d'ordonnancement global consiste à affecter en ligne les m tâches les plus prioritaires aux m unités de calcul [24].

Définition 24. *(**Ordonnancement par partitionnement**)*

Le principe d'un algorithme utilisant une stratégie par partitionnement est de placer chaque tâche sur une unité de calcul, et ensuite d'exécuter sur chaque unité de calcul un algorithme d'ordonnancement mono processeur [24].

Dans le contexte d'un ordonnancement par partitionnement, la migration des tâches n'est pas autorisée. Cependant, l'ordonnancement global autorise la migration des tâches d'une unité de calcul à une autre [24].

Dans la suite, nous discutons des différentes techniques d'analyse d'ordonnancement.

2.3.2 Analyse d'ordonnancement

L'analyse d'ordonnancement se concentre principalement sur deux problèmes : la faisabilité et l'ordonnançabilité [25].

Nous pouvons définir un système ordonnançable et un système faisable comme suit :

Définition 25. *(**Système faisable**)*

Pour un ensemble de tâches donné s'exécutant sur un ensemble de processeurs donné, l'ensemble de tâches est dit faisable si l'ensemble de tâche est ordonnançable par au moins un algorithme d'ordonnancement existant [26].

Définition 26. *(**Système ordonnançable**)*

Pour un ensemble de tâches donné s'exécutant sur un ensemble de processeurs donné et pour un algorithme d'ordonnancement donné, l'ensemble de tâches est dit ordonnançable si toutes les contraintes temps réel sont respectées selon l'algorithme d'ordonnancement [26].

Plusieurs méthodes ont été proposées pour analyser l'ordonnancement d'un système temps réel. Nous pouvons citer l'approche analytique et l'approche par simulation. Dans la suite, nous détaillons ces deux approches.

2.3.2.1 Approche analytique

L'analyse par approche analytique consiste à vérifier la validation d'une condition d'ordonnonçabilité ou/et de faisabilité. Ces conditions sont généralement basées sur le comportement du système sous son pire scénario d'exécution [27].

2.3.2.2 Approche par simulation de l'ordonnancement

Une autre approche d'analyse d'ordonnancement est la simulation. Cette méthode est couramment utilisée pour l'analyse des performances, le dimensionnement ou la mise au point des systèmes temps réel.

Cette approche comprend deux étapes. La première étape consiste à modéliser le comportement d'un système à l'aide d'un formalisme adéquat. La deuxième étape consiste à exécuter ce modèle avec un outil de simulation. Les scénarios de simulations sont définis par l'utilisateur. Le résultat de simulation peut être de simples traces de l'exécution du système ou des évaluations de certains comportements du système [25].

Dans le contexte d'ordonnancement temps réel, plusieurs outils de simulation ont été proposés. Nous citons à titre d'exemple Cheddar [28], MAST [29] et SimSo [30].

Dans ce contexte, l'analyse temporelle sur un intervalle de simulation fini appelé intervalle de faisabilité est suffisante. Par la suite, si toutes les tâches respectent leurs échéances sur cet intervalle alors le système de tâches est ordonnançable [31].

2.4 Ordonnancement temps réel multiprocesseur de tâches dépendantes

Contrairement à l'ordonnancement monoprocesseur, l'ordonnancement multiprocesseur est caractérisé par la présence de plusieurs unités de calcul sur lesquelles peuvent s'exécuter les tâches. Dans le contexte d'ordonnancement multiprocesseurs, nous rencontrons principalement les problématiques suivantes [32] :

- **Placement des tâches :** sur quelle(s) unité(s) de calcul une tâche doit-t-elle s'exécuter ?
- **Migration des tâches :** une tâche peut-elle migrer d'une unité de calcul vers une autre pour s'exécuter ?
- **Ordonnancement des tâches :** quand une tâche peut-elle commencer son exécution sur une unité de calcul donnée ?
- **Ordonnancement des communications :** quand, comment et combien de temps faut-il pour transporter les données d'une tâche source vers une tâche destination ?

L'ordonnancement d'un système temps réel ayant des contraintes de dépendance est un problème bien connu dans la littérature. Ce problème est un problème NP-complet [32].

Cependant, plusieurs algorithmes ont été proposés dans le cadre multiprocesseurs. Ces algorithmes sont basés sur des hypothèses diverses et leurs fonctionnalités sont également différentes. Ainsi, il est difficile de les grouper dans un contexte unifié. Dans cette section, nous proposons une taxonomie qui classe ces algorithmes en différentes catégories. Ensuite, nous présentons brièvement les algorithmes proposés par la communauté. Puis, nous discutons des algorithmes qui ordonnancent les systèmes à criticité mixte sur des architectures multiprocesseurs.

2.4.1 Une taxonomie des algorithmes d'ordonnancement multiprocesseurs

Afin de décrire les variantes d'algorithmes d'ordonnancement et de décrire la portée de notre étude, nous introduisons Figure 2.5 une taxonomie pour les algorithmes d'ordonnancement multiprocesseurs.

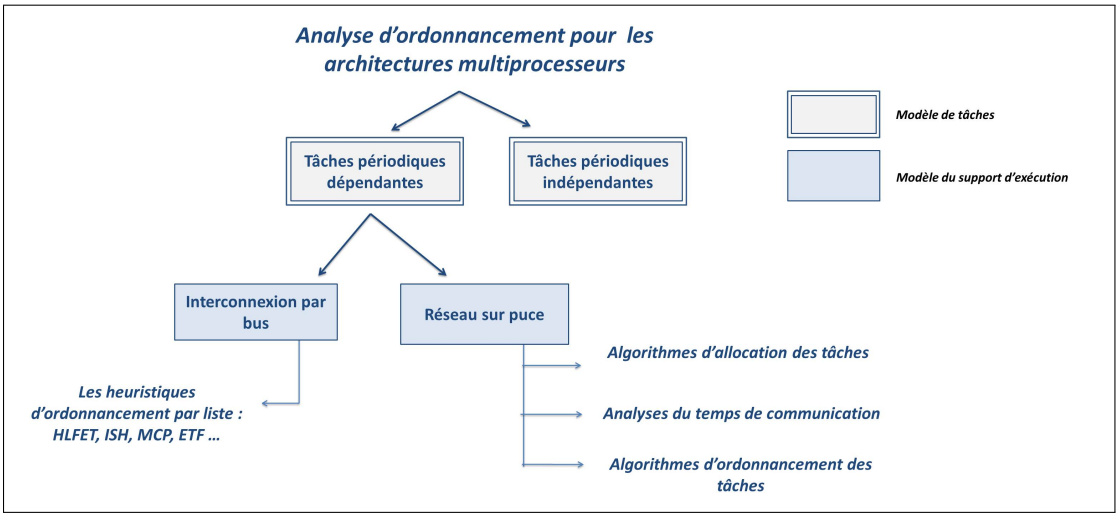


FIGURE 2.5 – Taxonomie des algorithmes d'ordonnancement multiprocesseur. Pour le modèle de tâche, nous considérons les tâches périodiques dépendantes. Concernant le modèle de support d'exécution, nous considérons les architectures multiprocesseurs et les architectures réseau sur puce. Pour les architectures multiprocesseurs il existe les heuristiques par liste. Dans le contexte de réseau sur puce, plusieurs travaux ont été proposés dans l'objectif d'ordonnancer ces systèmes temps réel : des analyses du temps de communication, des algorithmes d'ordonnancement des tâches et des algorithmes d'allocation des tâches.

Dans le contexte des architectures multiprocesseurs, plusieurs algorithmes qui adoptent ces hypothèses ont été proposées. La plupart des algorithmes sont des heuristiques basées sur l'approche d'ordonnancement par liste (ou anglais LIST SCHEDULING ALGORITHMS)[33].

Dans le contexte des architectures disposant d'un réseau sur puce, plusieurs protocoles de communication, analyses de communication et algorithmes d'affectation des tâches ont été proposés dans l'objectif d'ordonnancer les systèmes temps réel.

Dans la suite, nous discutons brièvement les heuristiques d'ordonnancement de liste. Ensuite, nous présentons les différentes solutions proposées pour ordonnancer des systèmes temps réel sur des réseaux sur puce.

2.4.2 Heuristiques d'ordonnancement par liste

L'ordonnancement par liste consiste à construire une liste dans laquelle les tâches prêtes sont classées par priorité décroissante. Dès qu'une unité de calcul est libre, la tâche de priorité la plus élevée lui est affectée. Une tâche est dite prête à un instant donné si elle est réveillée et satisfait toutes ses contraintes de dépendance.

Plusieurs algorithmes de liste ont été proposés pour ordonnancer des systèmes temps réel ayant un modèle de tâches périodiques dépendantes sur des architectures multiprocesseurs[33, 34]. Nous pouvons citer les algorithmes Highest Level

First with Estimated Times (HLFET), Insertion Scheduling Heuristic (ISH), Modified Critical Path (MCP), Earliest Time First (ETF), Dynamic Level Scheduling (DLS) et Localized Allocation of Static Tasks (LAST).

[33] présente une étude comparative entre les différentes heuristiques de liste. Nous remarquons que l'heuristique qui fournit les meilleurs résultats est l'heuristique HLFET. La complexité de cette heuristique est d'ordre O(n^2). [33] montre que l'heuristique HLFET retourne des solutions qui sont à moins de 5% de la solution optimale dans 90% des cas.

Dans le paragraphe suivant, nous détaillons le principe de cet algorithme en précisant ses avantages et ses inconvénients.

HLFET

L'algorithme HLFET fonctionne de la façon suivante :

1. HLFET utilise un DAG.

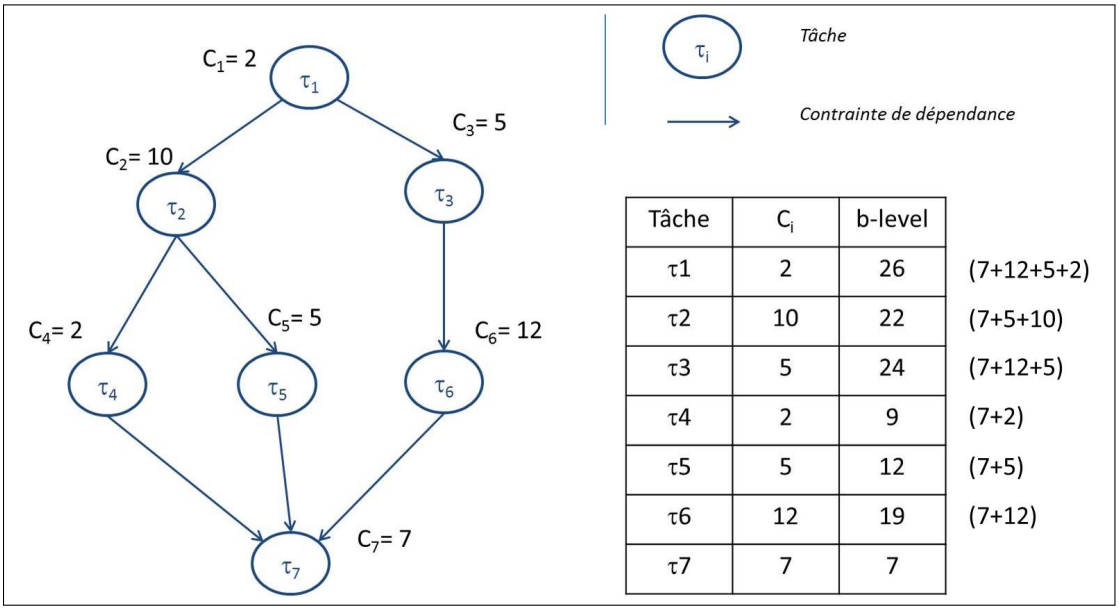
2. On calcule un b-level pour chaque tâche. Le b-level (Bottom level of a node) est la longueur du plus long chemin entre un nœud (une tâche) donné et le nœud de sortie (la tâche de sortie).

3. On construit une liste des tâches prêtes dans un ordre décroissant selon la valeur de b-level. Au départ, cette liste ne contient que les tâches d'entrée.

4. On élit la première tâche dans la liste pour l'unité de calcul disponible (ou l'unité de calcul affectée) et qui permet au plus tôt l'exécution.

Afin d'illustrer l'algorithme HLFET, nous prenons l'exemple décrit dans les figures 2.6 et 2.7. Comme le montre la figure 2.6, nous considérons 7 tâches périodiques dépendantes organisées par un DAG. La figure 2.7 montre l'ordonnancement généré par HLFET pour le jeu de tâches considéré sur une architecture multiprocesseur de 3 unités de calcul.

Tout d'abord, nous calculons le b-level pour chaque tâche. Nous rappelons ici que le b-level d'une tâche donnée est la longueur du plus long chemin entre la tâche considérée et la tâche τ_7. Par exemple, le b-level de la tâche τ_1 égale à la somme des capacités des tâches τ_7, τ_6, τ_3 et τ_1. Ensuite, nous construisons une liste des tâche prêtes dans un ordre décroissant selon la valeur de b-level. Ainsi, la première tâche dans cette liste sera affecté au premier unité de calcul disponible.



Tâche	C_i	b-level	
τ1	2	26	(7+12+5+2)
τ2	10	22	(7+5+10)
τ3	5	24	(7+12+5)
τ4	2	9	(7+2)
τ5	5	12	(7+5)
τ6	12	19	(7+12)
τ7	7	7	

FIGURE 2.6 – Exemple HLFET :
nous considérons 7 tâches périodiques dépendantes (τ_1, τ_2, τ_3, τ_4, τ_5, τ_6, τ_7). La tâche τ_1 est la tâche d'entrée. La tâche τ_7 est la tâche de sortie. Dans la figure 2.7, nous calculons la valeur de b-level pour chaque tâche.

Étape	Liste des tâches prêtes	PE0	PE1	PE2
0	τ_1	τ_1	---	---
1	τ_3, τ_2	τ_3	τ_2	---
2	τ_6	τ_6	τ_2	---
3	τ_5 , τ_4	τ_6	τ_5	τ_4
4	τ_7	τ_7	---	--

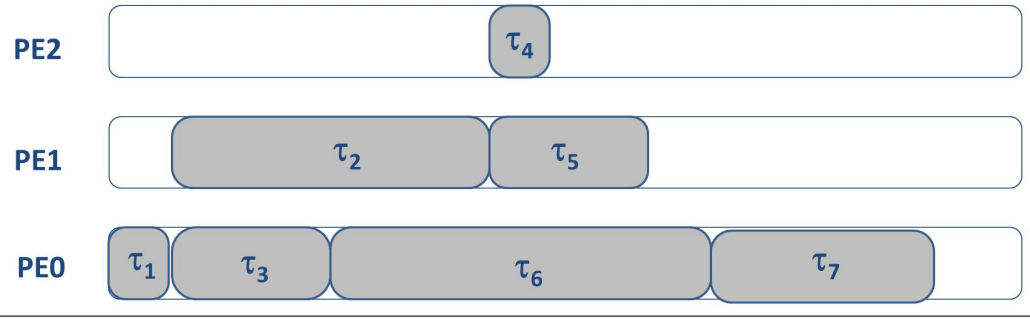


FIGURE 2.7 – Exemple HLFET :
en se basant sur la valeur de b-level pour chaque tâche, HLFET construit la liste des tâches prêtes dans un ordre décroissant. Ensuite, il ordonnance la première tâche sur une unité de calcul disponible.

2.4.3 Réseau sur puce

Dans le contexte d'un réseau sur puce, le partage de ressources comme les routeurs et les liens physiques résulte parfois en de nouvelles interférences et en des délais de communications non prédictibles, ce qui complique par la suite l'analyse d'ordonnancement de ces systèmes.

Sur ces problématiques, plusieurs solutions ont été proposées ces dernières années [35]. Nous pouvons les classer selon 3 objectifs :

2.4.3.1 Algorithmes d'allocation des tâches

Plusieurs algorithmes ont été proposés pour calculer la meilleur affectation des tâches selon un critère donné. Minimiser à la fois la moyenne de temps de communication et la consommation énergétique sont les principales fonctions objectives traitées dans la littérature [36, 37].

2.4.3.2 Analyse du temps de communication

Les communications au sein d'un NoC ont récemment attiré une attention particulière en raison de leur potentiel pour l'amélioration des performances et de l'efficacité énergétique [5].

Plusieurs architectures NoC ont été conçues dans l'objectif de minimiser le temps de communication en évitant les collisions entre les communications [5]. En outre, des analyses de pire temps de communication au sein du NoC ont été élaborées [8, 38, 39].

2.4.3.3 Algorithmes d'ordonnancement des tâches

Il y a peu d'approches qui traitent de l'ordonnancement des tâches dans un réseau sur puce et encore moins qui considèrent l'ordonnancement des communications. Nous pouvons grouper ces solutions en deux classes : les solutions qui ignorent les communications dans le NoC et les solutions qui prennent en compte les communications uniquement [10].

G. Varatkar et al [40] ont développé un algorithme d'ordonnancement en deux étapes ayant l'objectif de minimiser la consommation d'énergie du NoC. Puisque les temps de communications ne sont pas considérés, cet algorithme n'est pas compatible avec les exigences des systèmes temps réel dur.

Lei et al [41] ont proposé un algorithme d'ordonnancement et d'allocation en deux étapes. Pour l'allocation, ils utilisent un algorithme génétique. Pour le temps de communication, ils considèrent la moyenne de pire temps de communication. De

ce fait, cet algorithme n'est pas compatible avec les exigences des systèmes temps réel dur.

2.5 Conclusion

Dans la première partie de ce chapitre, nous avons présenté les concepts de base de l'analyse d'ordonnancement des systèmes temps réel et des systèmes à criticité mixte. Nous avons présenté les différentes catégories de système temps réel. Ensuite, nous avons introduit les concepts autour des systèmes à criticité mixte en exposant les modèles de tâches considérées dans la littérature. Ainsi, nous avons abordé les différentes approches utilisées pour l'analyse d'ordonnancement.

Dans la deuxième partie, nous avons présenté les algorithmes d'ordonnancement multiprocesseur qui considèrent des tâches périodiques dépendantes. Nous avons détaillé les algorithmes par liste et leurs performances. Ensuite, nous avons introduit des méthodes proposées pour ordonnancer des systèmes temps réel sur des architectures NoC.

Plusieurs protocoles et architectures de NoC ont été proposés dans l'objectif d'ordonnancer des systèmes temps réel sur ces architectures. Nous présentons l'état de l'art de ces solutions dans le chapitre suivant.

3

Les réseaux sur puce

Sommaire

Introduction

Le réseau sur puce est un paradigme d'interconnexion qui a été introduit en 2000 puis utilisé industriellement plusieurs années plus tard. Il améliore les performances de communication des architectures d'interconnexion classiques sur puce. Il assure également les performances de communication requises afin d'accompagner l'évolution des systèmes sur puce [5].

Cependant, l'intégration des systèmes temps réel, en général, et des systèmes à criticité mixte, en particulier, sur des architectures réseau sur puce exige que les communications transportées via le réseau et les tâches exécutées sur les unités de calcul respectent leurs contraintes temporelles [32]. Pour cela, une analyse de l'ordonnancement pour les tâches et les communications est obligatoire afin de valider le bon fonctionnement du système [8].

Dans ce chapitre, nous expliquons en quoi les moyens de communication classiques ne répondent plus aux exigences des systèmes sur puce et ce que le réseau sur puce apporte comme solution. Puis, nous détaillons l'architecture d'un routeur NoC. Ensuite, nous évoquons les notions fondamentales relatives aux réseaux sur puce. Nous présentons les communications temps réel et les analyses de pire temps de communication. Finalement, nous discutons des architectures de réseau sur puce et des protocoles existants qui supportent les systèmes à criticité mixte.

3.1 Interconnexions dans les systèmes sur puce

3.1.1 Techniques classiques d'interconnexion

Définition 27. *(**Système sur puce**)*

Un système sur puce (ou en anglais System-on-Chip (SoC)) est un ensemble de composants matériels et de logiciels conçus et intégrés dans une seule puce électronique pour réaliser les fonctionnalités demandées [5].

Nous pouvons trouver les systèmes sur puce dans de nombreux secteurs d'activités. Ils peuvent être utilisés dans les téléphones mobiles, les consoles portables ou encore les tablettes [42].

Un SoC est composé d'unités de calcul, de mémoires et de périphériques d'entrée et de sortie. Ces éléments communiquent entre eux via une structure d'interconnexion. Les techniques classiques d'interconnexion utilisées dans les systèmes sur puce sont les bus partagés et les bus hiérarchiques [42].

L'interconnexion point à point ne connecte que deux éléments. Le bus partagé et le bus hiérarchique sont utilisés dans les SoC dont le nombre d'éléments à interconnecter peut aller jusqu'à la dizaine [42].

Les architectures d'interconnexion sur puce ont connu une évolution remarquable depuis les années 1990. En suivant cette évolution, le nombre de cœurs qui y sont intégrés est en augmentation. Cette augmentation rend l'utilisation des bus partagés et les autres techniques d'interconnexion inadéquates pour ce type d'architectures [5]. Afin de répondre aux exigences des systèmes sur puce, un nouveau paradigme d'interconnexion est apparu vers le milieu des années 2000 : le réseau sur puce.

3.1.2 Réseau sur puce

Définition 28. *(**Réseau sur puce**)*

Le réseau sur puce (ou en anglais Network-On-Chip (NoC)) est un paradigme de connexion entre les unités de calcul intégrées dans un système sur puce. Il achemine des données de l'émetteur vers un ou plusieurs récepteurs via des routeurs [5].

Un NoC est composé principalement de routeurs, de nœuds, d'interfaces réseau (ou en anglais *Network Interfaces (NI)*) et de liens physiques unidirectionnels. Un exemple d'architecture NoC en topologie grille (ou MESH) 2D 3×3 est illustré par la figure 3.1.

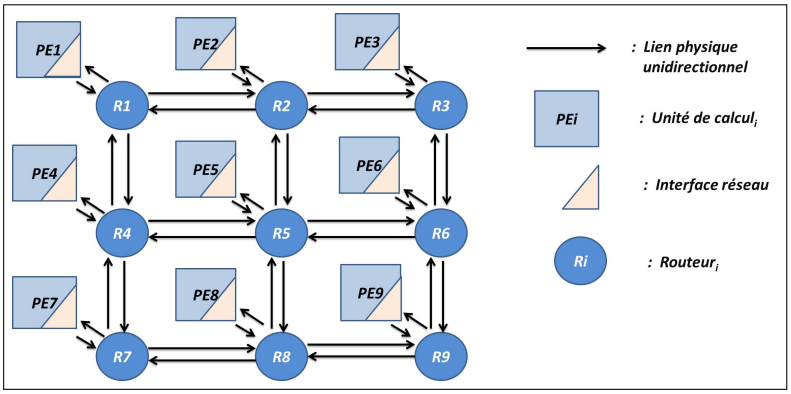


FIGURE 3.1 – Exemple d'un NoC 3x3 en topologie grille
Le réseau sur puce est composé des routeurs R_i, des unités de calcul PE_i, des interfaces réseau et des liens physiques e_{RxRy}, e_{RxPEy} ou e_{PExRy}.
e_{RxRy} représente le lien qui transporte des données du routeur R_x vers le routeur R_y. e_{PExRy} représente le lien qui transporte des données de l'unité de calcul PE_x vers le routeur R_y.
e_{RxPEy} représente le lien qui transporte des données du routeur R_x vers l'unité de calcul PE_y.

Définition 29. *(**Routeur**)*

Le routeur est l'entité responsable de l'acheminement des données entre les nœuds intégrés dans un réseau sur puce à travers les différents liens d'interconnexions du réseau [5].

Définition 30. *(**Nœud**)*

Les nœuds peuvent être des processeurs, des mémoires, des unités de calcul (ou en anglais Prcoessing Element (PE)) ou encore une combinaison de plusieurs de ces éléments. Chaque nœud est connecté à son propre routeur à travers une interface réseau [5].

Définition 31. *(**Interface réseau**)*

L'interface réseau relie un nœud à un routeur. Par conséquent, elle permet au nœud d'envoyer des données au routeur et de recevoir des données envoyées par le routeur [5].

Définition 32. *(**Lien physique**)*

Les liens physiques sont des connexions logiques entre deux éléments communicants. Ils permettent les connexions de type point à point entre les nœuds et les routeurs et entre les routeurs [5].

Pour résumer, dans un réseau sur puce, les unités de calcul s'échangent des données via le routeur. L'interface réseau constitue l'interface entre l'unité de calcul et le routeur. Le rôle du lien physique consiste à fournir une bande passante et à transporter des données entre les éléments communicants.

Le réseau sur puce possède plusieurs paramètres à déterminer comme la topologie du réseau, la technique de commutation, la stratégie de routage et la politique d'arbitrage.

Dans la suite, nous présentons la composition du routeur NoC. Ensuite, nous détaillons certaines notions fondamentales relatives aux NoC.

3.2 Routeur du réseau sur puce

Dans cette partie, nous présentons l'architecture globale d'un routeur et de ses composants.

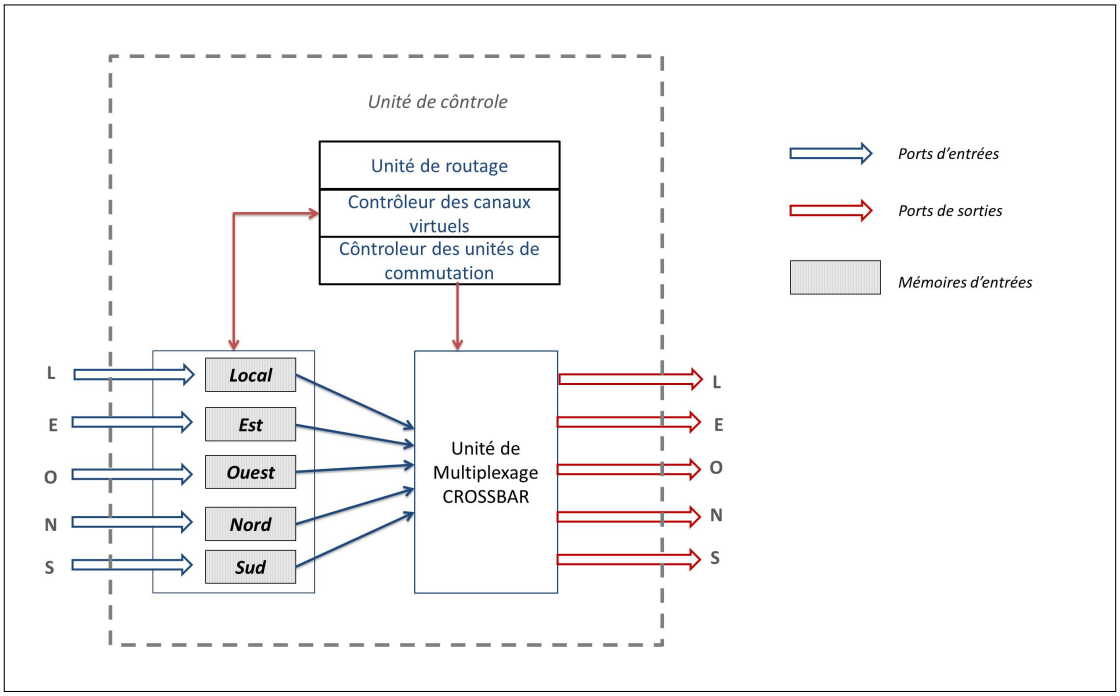


FIGURE 3.2 – Composition d'un routeur NoC

Durant ces dernières années, plusieurs architectures de routeur NoC ont été proposées dans le but d'améliorer les performances. Toutes ces architectures conservent la structure globale du routeur classique [5].

Un routeur est composé de ports d'entrées/sorties, de mémoires tampons, d'une unité de contrôle et d'une unité de multiplexage dite crossbar [5]. Nous présentons l'architecture globale du routeur figure 3.2.

Définition 33. *(**Ports d'entrées/sorties**)*

Le rôle des ports d'entrées consiste à introduire des données dans le routeur tandis que les ports de sorties sont responsables de la sortie des données hors du routeur [5].

Dans un réseau sur puce, chaque routeur dispose de plusieurs ports d'entrées/sorties. Le nombre de ports pour un routeur dépend de la topologie du réseau et de la position du routeur au niveau de cette topologie [5].

Définition 34. *(**Mémoires tampons**)*

Les mémoires tampons sont des mémoires FIFO. Ces mémoires sont responsables du stockage des données dans le routeur [5].

Afin de transmettre les données vers la destination, un routeur de réseau sur puce contient des mémoires tampons. Ces mémoires peuvent êtres utilisées en entrée ou/et en sortie pour pouvoir gérer plusieurs données en parallèle [5].

Définition 35. *(**Unité de contrôle**)*

L'unité de contrôle est l'unité responsable de la gestion de contrôle de données avec les routeurs voisins [5].

Autrement dit, cette unité est responsable de l'envoi et de la réception des requêtes et des acquittements échangés avec les routeurs voisins. Elle gère également le stockage des données en cas de chemins bloqués [5].

Dans l'état de l'art, plusieurs structures de cette unité ont été proposées. Elle peut être également composée d'un contrôleur de crossbar, d'un contrôleur de mémoires tampons ou d'un contrôleur de canaux virtuels et d'une unité de routage [5].

Définition 36. *(**Crossbar**)*

Le crossbar définit un réseau de connexions au sein du routeur. Via le crossbar, chaque port d'entrée doit pouvoir accéder à l'ensemble des ports de sorties [5].

Le rôle du crossbar consiste à acheminer les données du port d'entrée vers le port de sortie.

Dans la suite, nous détaillons les notions fondamentales relatives aux NoC.

3.3 Concepts relatifs aux réseaux sur puce

Dans l'état de l'art, plusieurs architectures de réseau sur puce ont été proposées. Chacune de ces architectures est caractérisée par ses propres paramètres qui la rend différente des autres. Dans cette partie, nous présentons certaines de ces caractéristiques.

3.3.1 Topologie

Définition 37. *(**Topologie**)*

La topologie désigne le graphe des liens entre les différents éléments du réseau [43].

La topologie définit comment les routeurs sont interconnectés en utilisant les liens du réseau. Elle est souvent modélisée par un graphe qui spécifie l'organisation physique du réseau.

Pour un réseau sur puce, nous pouvons avoir plusieurs topologies : linéaire, grille, arbre, tore, multigrille, hypercube, etc.

La topologie en grille (ou en anglais Mesh) est la topologie commune pour les réseaux sur puce. Elle permet d'utiliser des règles de routage simples et peu coûteuses [43].

3.3.2 Paquet

Dans un NoC, les nœuds communiquent en s'échangeant des messages à travers le réseau. Afin d'assurer une utilisation équitable et efficace des ressources du réseau, ces messages sont souvent divisés en paquets avant d'être transmis.

Définition 38. *(**Paquet**)*

Un paquet est la plus petite unité de communication qui contient des informations de routage. Il est composé de plusieurs flits [5].

Chaque message est divisé en paquets. Chaque paquet est composé de plusieurs flits. Chaque flit est composé de plusieurs phits.

Définition 39. *(**Flit**)*

Un flit (ou en anglais FLOW CONTROL UNIT*) correspond à la plus petite unité de communication pour laquelle nous pouvons définir un contrôle de flux. Il est composé de plusieurs phits [5].*

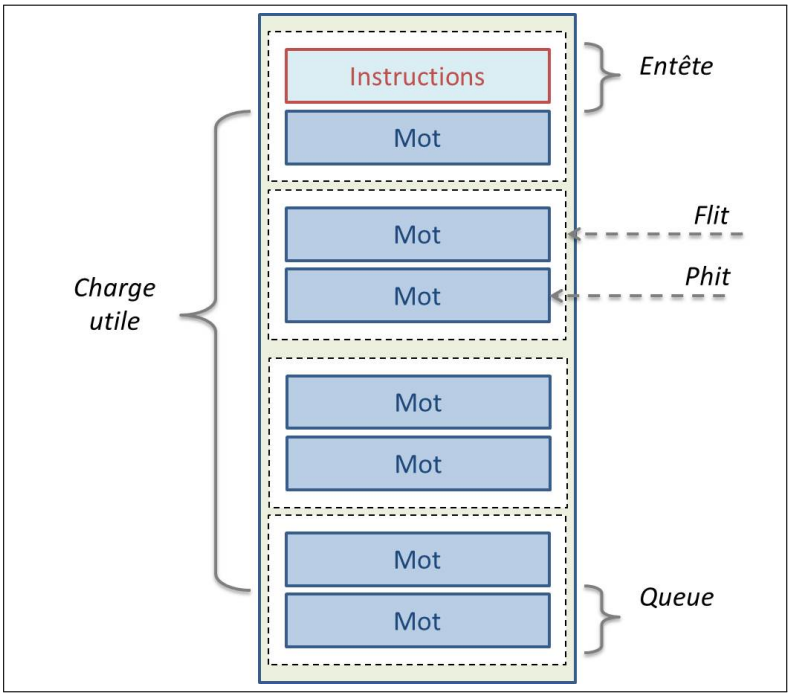


FIGURE 3.3 – Structure d'un paquet.
La structure d'un paquet commence par un entête, puis le corps du paquet et il se termine par une queue.

Définition 40. ***(Phit)***

Un phit (ou en anglais PHYSICAL UNIT*) correspond à la quantité de bits qui peut être transportée en une seule fois sur un lien [5].*

La figure 3.3 présente la structure d'un paquet. L'entête transporte notamment des informations relatives au routage et qui sont indispensables pour l'acheminement de ce paquet vers sa destination. Le corps transporte la charge utile du paquet, qui est le message à transmettre. La queue indique la fin du paquet.

3.3.3 Algorithme de routage

Définition 41. ***(Algorithme de routage)***

Un algorithme de routage détermine le chemin emprunté par un paquet entre la source et la destination [43].

L'algorithme de routage joue un rôle important dans les architectures de communication. Il est soigneusement élaboré par les concepteurs du fait de son impact sur les performances du réseau. Le choix de l'algorithme de routage est basé sur

un compromis entre une utilisation optimale des liens de communication et un algorithme simple qui peut être facilement mis en œuvre sur silicium [44].

Dans le contexte des réseaux sur puce, il existe plusieurs algorithmes de routage avec des propriétés différentes. Le routage XY, le routage YX, le routage par la source et le routage plus court chemin (ou en anglais *Shortest Path Routing*) sont des exemples d'algorithmes de routage utilisés dans les NoC [44].

Pour les topologies de grille 2D, l'algorithme de routage XY est l'algorithme le plus utilisé.

Définition 42. *(**Algorithme de routage XY**)*

L'algorithme de routage XY consiste à transférer le paquet horizontalement puis verticalement en utilisant les coordonnées du routeur actuel et celles du routeur de destination [44].

3.3.4 Technique de commutation

Définition 43. *(**Technique de commutation**)*

La technique de commutation définit la méthode d'acheminement des données au sein du réseau [43].

Nous pouvons trouver principalement deux techniques de commutation : la commutation de circuit et la commutation de paquet.

Définition 44. *(**Commutation de circuit**)*

En commutation de circuit, les communications se font en deux étapes. Le chemin physique de la source jusqu'à la destination est d'abord réservé, puis, le message est transmis entièrement [43].

Définition 45. *(**Commutation de paquet**)*

En commutation de paquet, les paquets sont envoyés à travers le réseau sans réservation préalable d'un chemin de la source à la destination. Ceux-ci définissent leur chemin au fur et à mesure de leur propagation dans le réseau jusqu'à leur arrivée à destination [43].

En adoptant la commutaion de paquet, pour chaque paquet reçu, le routeur calcule le routage, puis envoie le paquet vers le port de sortie correspondant à sa destination.

La commutation de paquet réduit la durée de non disponibilité des ressources en cours d'utilisation par rapport à la commutation de circuit. En revanche, la commutation de paquet nécessite une gestion d'interférences au niveau des routeurs.

Dans le contexte des réseaux sur puce, il existe trois types de commutation de paquet : *Store-And-Forword (SAF)*, *Virtual-Cut-Through (VCT)* et *Wormhole*. Ces trois techniques de commutation sont illustrées par la figure 3.4.

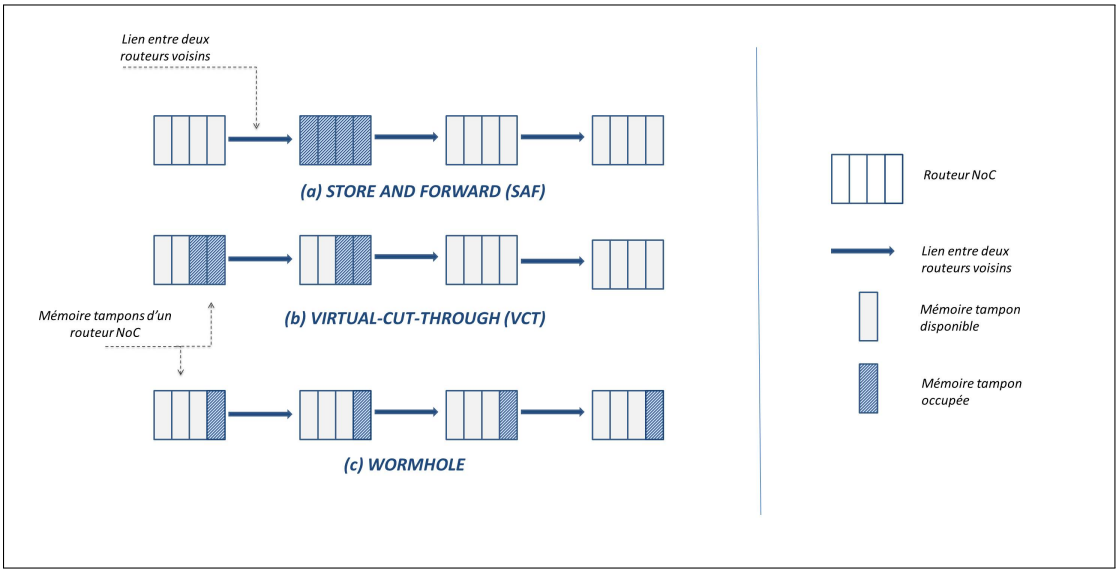


FIGURE 3.4 – Techniques de commutation.
La commutation SAF exige la réception de la totalité du paquet pour la transmission au routeur suivant. La technique VCT réduit le temps de communication par rapport à SAF. Wormhole réduit la taille du tampon du routeur par rapport à VCT. Le paquet considéré dans cet exemple est de taille 4 flits.

3.3.4.1 *Store-and-Forward (SAF)*

Définition 46. *(**Store-and-Forward (SAF)**)*

Chaque routeur attend la réception de la totalité des flits constituant un paquet, le stocke entièrement avant de le transmettre au routeur suivant [12].

La commutation SAF implique un temps de communication élevé par rapport aux autres techniques de commutation. C'est pourquoi cette technique n'est pas privilégiée dans le domaine des NoC [12].

3.3.4.2 *Virtual-Cut-Through (VCT)*

Définition 47. *(**Virtual Cut Through (VCT)**)*

La technique VCT a été introduite pour réduire la latence du SAF. Un routeur fait suivre les flits d'un paquet au plus vite, mais il peut stocker tout le message s'il y a une interférence avec d'autres messages [45].

Cette technique améliore le temps de communication par rapport à SAF. Le routeur courant n'est plus obligé de stocker la totalité du paquet avant de le transmettre. Le paquet peut être envoyé dès que le prochain routeur possède suffisamment de place pour l'accueillir. Cette technique nécessite de prévoir un tampon au moins égal à la taille du paquet dans chaque routeur [5].

3.3.4.3 *Wormhole*

Définition 48. ***(Wormwhole)***

La technique Wormwhole consiste à acheminer le paquet sous forme de flits, sans possibilité de stocker la totalité du paquet. Ainsi on réduit la taille des tampons mémoires avec le risque d'interblocage en cas d'interférence de l'un des flits [12].

L'unité de transmission de paquet dans la technique Wormhole est le flit. Par conséquence, cette technique réduit les besoins en espace mémoire pour les stockages intermédiaires [12].

La technique Wormhole offre de meilleures performances en termes de temps de communication par rapport à SAF et VCT, mais elle introduit également des risques d'interférence dans le réseau [5].

3.3.5 Politique de mémorisation et canaux virtuels

Nous avons montré précédemment que la commutation de paquets nécessite des mémoires tampons. Dans ce paragraphe, nous présentons les différentes politiques de mémorisation utilisées dans le domaine des NoC.

Définition 49. ***(Politique de mémorisation)***

La politique de mémorisation définit le positionnement des mémoires tampons par rapport aux ports d'entrées et de sorties du routeur [46].

Dans l'état de l'art, il existe principalement trois stratégies de stockage : file d'attente en entrée, file d'attente en sortie et file d'attente en entrée avec canaux virtuels [46].

Pour la stratégie file d'attente en entrée, les files d'attente sont placées aux entrées du routeur. Avec la stratégie file d'attente en sortie, les files d'attente sont placées aux sorties du routeur. En adoptant la stratégie file d'attente en entrée avec canaux virtuels, chaque port d'entrée dispose de plusieurs canaux virtuels.

Définition 50. ***(Canal virtuel)***

Les canaux virtuels (VC) sont des canaux logiques qui partagent un lien physique en utilisant des tampons de mémorisation distincts [47].

L'utilisation des canaux virtuels conduit à une augmentation de la surface et de la latence de communication. Cependant, cette stratégie présente de nombreux avantages. Premièrement, les canaux virtuels permettent d'éviter les situations d'interblocages. Deuxièmement, ils permettent d'optimiser l'utilisation des liens et du réseau. Enfin, ils permettent également de séparer les communications de différentes classes [47].

3.3.6 Politique d'arbitrage

Lorsque deux ou plusieurs ports d'entrées demandent l'acheminement du paquet vers le même port de sortie, au même moment, nous avons besoin d'un arbitre pour sélectionner un seul port d'entrée selon une politique d'arbitrage donnée.

Définition 51. *(**Politique d'arbitrage**)*

L'arbitrage résout le problème de contention qui se présente lors d'interférence entre différentes données qui veulent accéder à la même ressource [47].

La figure 3.5(a) présente un exemple d'interférence due au partage de ressources dans un réseau sur puce. Dans cet exemple, nous considérons deux flux ρ_1 et ρ_2. Ces deux flux passent par le même routeur R_3 et demandent en même temps le même lien e_{R3R4}. Mais le routeur R_3 ne peut router qu'un seul flux par port de sortie. D'où le rôle de l'arbitre dans la sélection des flux qui partagent les mêmes liens.

Dans l'état de l'art, il existe plusieurs techniques d'arbitrage utilisées dans les architectures NoC. Les principales techniques d'arbitrages sont : l'arbitrage à priorité tournante (ou en anglais *Round Robin*), l'arbitrage à priorité calculée et l'arbitrage Time Division Multiple Access [5].

Définition 52. *(**L'arbitrage à priorité tournante**)*

L'arbitrage à priorité tournante donne l'accès à la communication qui détient un jeton, et à chaque cycle d'horloge, le jeton change de propriétaire [47].

Définition 53. *(**L'arbitrage à priorité calculée**)*

L'arbitrage à priorité calculée permet d'affecter à chaque communication un niveau de priorité prédéterminée. Cette priorité est stockée dans l'entête du paquet [47].

L'arbitrage à priorité calculée est l'équivalent de l'arbitrage à priorité fixe dans la communauté temps réel.

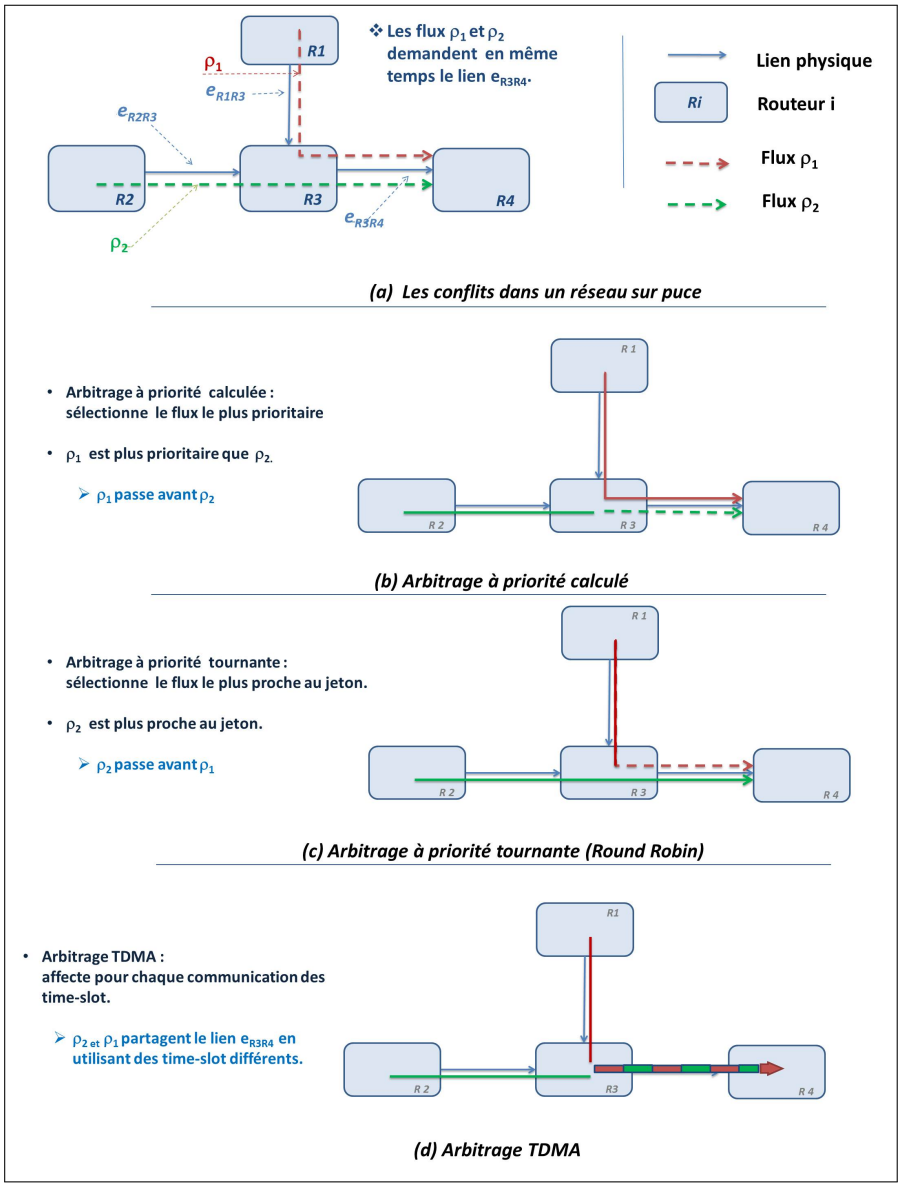


FIGURE 3.5 – Politiques d'arbitrage
(a) présente un exemple d'interférence due au partage de ressources dans un réseau sur puce. Nous présentons dans (b) le comportement de l'arbitre à priorité tournante. (c) décrit le comportement de l'arbitre à priorité calculée. Le comportement de l'arbitre TDMA est illustré dans (d).

Définition 54. ***(L'arbitrage TDMA)***

L'arbitrage TDMA (ou en anglais Time Division Multiple Access*) est un mode de multiplexage consistant à diviser le temps en périodes courtes appelées time-slots. Chaque communication est attribuée à un time-slot. Durant toute la durée du time-slot, les ressources partagées seront entièrement réservées à la communication concernée [5].*

3.3.7 Préemption

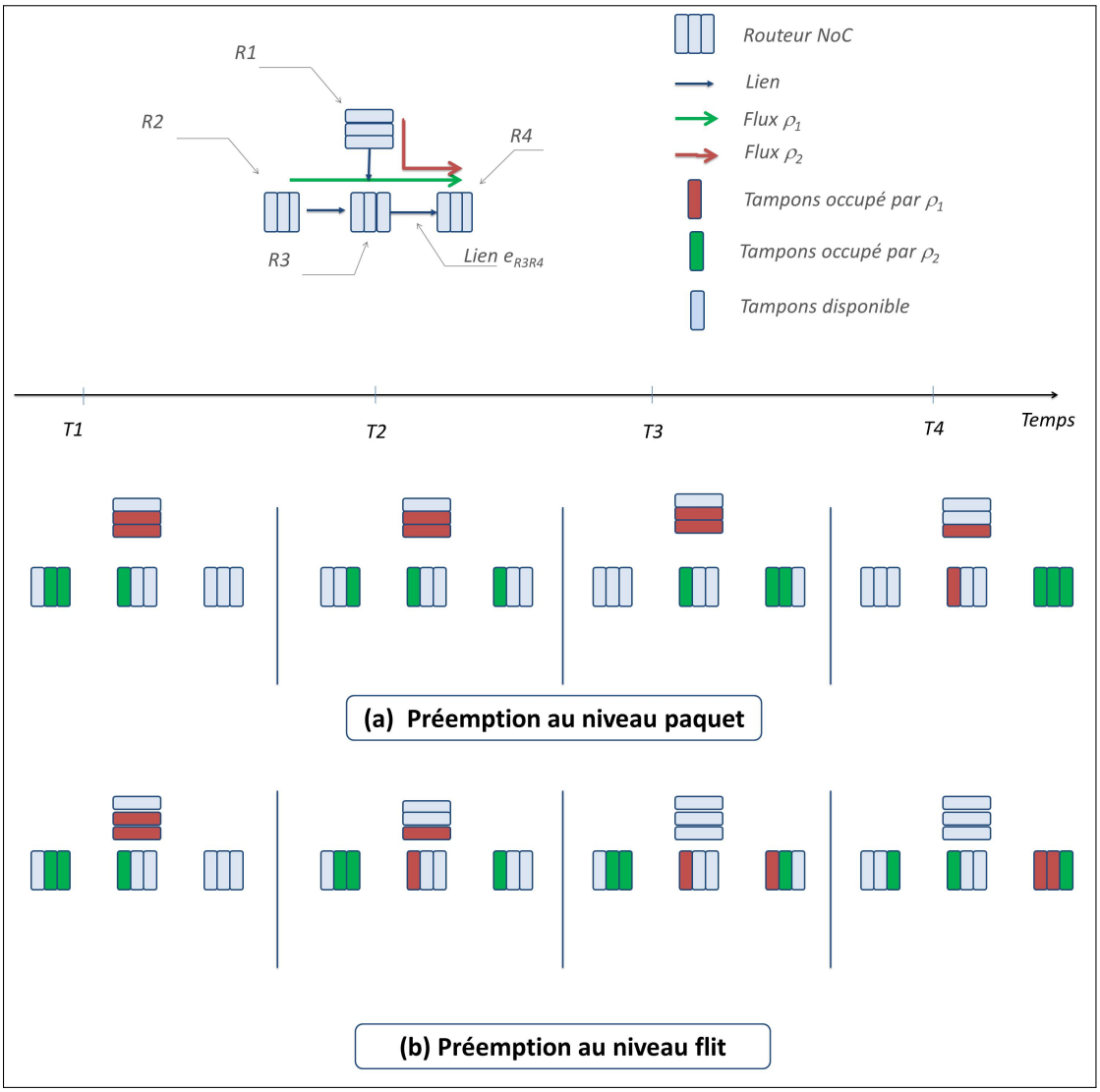


Figure 3.6 – Préemption dans le réseau sur puce.
Les flux ρ_1 et ρ_2 partagent le lien e_{R3R4}. Dans (a), ρ_1 attend le passage de $\rho 2$. Dans (b), ρ_1 préempte $\rho 2$ au niveau flit.

Définition 55. ***(Préemption)***

Dans le contexte des réseaux sur puce, la préemption est la capacité du routeur à interrompre un flux en cours de transmission en faveur d'un flux de priorité supérieure.

Il existe deux niveaux de préemption : le niveau paquet et le niveau flit.

Nous présentons figure 3.6 les deux exemples de préemption impliquant les flux ρ_1 et ρ_2. Les deux flux partagent le même lien physique e_{R3R4}. Le flux ρ_2 arrive au routeur R_3 avant ρ_1. Cependant, ρ_1 est plus prioritaire que le flux ρ_2. La transmission des flux ρ_1 et ρ_2 est présentée en 4 étapes différentes (T1, T2, T3 et T4) dans figure 3.6.

Dans le premier exemple, présenté figure 3.6 (a), le routeur NoC adopte une préemption au niveau paquet. Dans cet exemple, ρ_1 est bloqué dans le routeur R_1 jusqu'au passage de tout le paquet ρ_2.

Dans le deuxième exemple, présenté dans la figure 3.6 (b), le routeur NoC adopte une préemption au niveau flit. Dans cet exemple, ρ_1 préempte la transmission du flux ρ_2 au niveau flit. Le reste du paquet de ρ_2 est bloqué dans le routeur R_2 jusqu'au passage du flux ρ_1. La préemption au niveau flit est utilisée généralement en présence de plusieurs canaux virtuels.

3.3.8 Exemples de routeurs

Dans ce paragraphe, nous présentons trois exemples de routeurs NoC :

- *Without Virtual Channel Router (WVC-Router)*
- *Virtual Channel Router (VC-Router)*
- *Wormhole NoC Router (WNoC-Router).*

WVC-Router est le routeur NoC le plus simple. Il utilise la technique de commutation Wormhole. L'arbitrage adopté par ce routeur est l'arbitrage à priorité tournante [5].

VC-Router et *WNoC-Router* utilisent des canaux virtuels. *VC-Router* applique une préemption au niveau paquet avec un arbitrage à priorité tournante tandis que *WNoC-Router* applique une préemption au niveau flit avec un arbitrage à priorité calculée [5].

La figure 3.7 présente l'architecture de base pour un routeur NoC qui utilise des canaux virtuels. Le tableau 3.1 présente les différentes caractéristiques pour ces trois routeurs.

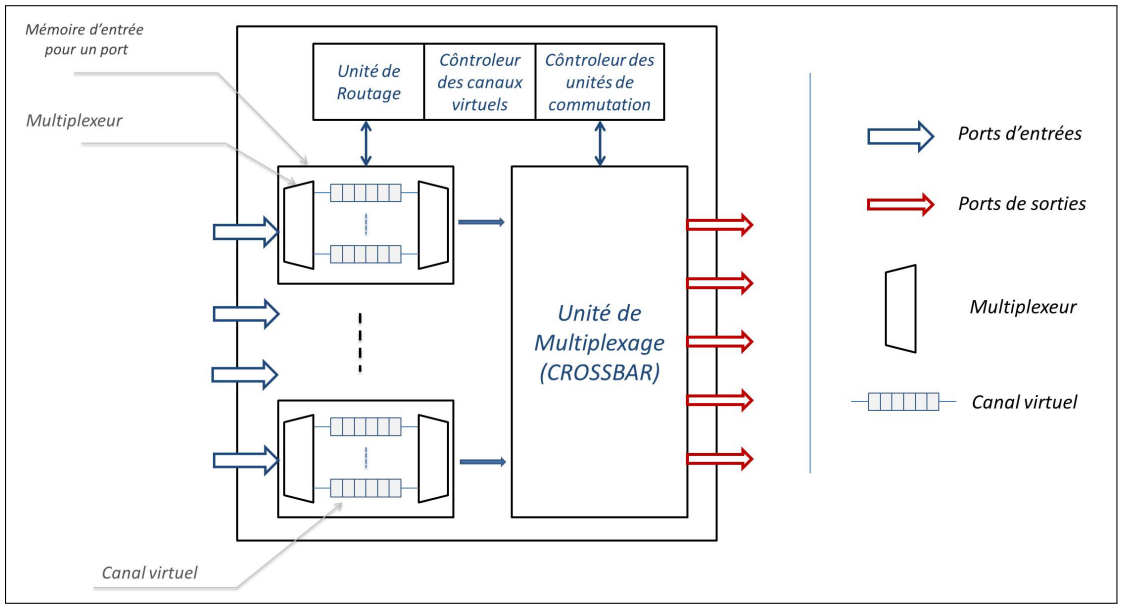


FIGURE 3.7 – Composition du routeur NoC avec canaux virtuels

TABLE 3.1 – Exemples des routeurs NoC

Routeur	WVC-ROUTER	VC-ROUTER	WNOC-ROUTER
Technique de commutation	Wormhole	Wormhole	Wormhole
algorithme de routage	XY	XY	XY
Politique de mémorisation	Fils d'attente en entrée	Avec canaux virtuels	Avec canaux virtuels
Politique d'arbitrage	Priorité tournante	Priorité tournante	Priorité calculé
Niveau de préemption	Paquet	Paquet	Flit

3.4 Communications temps réel et qualité de service

Selon la configuration adoptée, un réseau sur puce peut supporter plusieurs types de communication. Dans cette partie, nous définissons les communications temps réel. Ensuite, nous décrivons la qualité de service fournie par un NoC.

3.4.1 Communications temps réel

Un réseau sur puce permet d'exécuter plusieurs tâches simultanément. Ces tâches échangent des informations entre elles via l'infrastructure de communication du NoC. Certaines communications sont caractérisées par des contraintes temporelles. Ces communications sont appelées communications temps réel dans la suite de ce mémoire.

Définition 56. *(**Communications temps réel**)*

L'exactitude des communications temps réel repose non seulement sur l'exactitude de l'information échangée mais aussi de sa date d'arrivée [8].

Dans le cadre des communications temps réel, un paquet de données reçu trop tard par une destination pourrait être inutile ou même avoir une conséquence grave [8].

Dans l'état de l'art, il existe deux classes de communication temps réel : communication temps réel dur (ou en anglais *Hard Real-Time Communication*) et communication temps réel mou (ou en anglais *Soft Real-Time Communication*) [8].

3.4.1.1 Communications temps réel dur

Définition 57. *(**Communications temps réel dur**)*

Les communications temps réel dur sont les communications temps réel pour lesquelles nous devons obligatoirement respecter les échéances même sous les pires scénarios du fonctionnement du système [8].

Les communications temps réel dur sont utilisées dans les applications critiques où une échéance non respectée pour l'une de ces communications peut conduire à des résultats catastrophiques. Garantir le respect des contraintes temporelles pour ce type de communication est strictement obligatoire pour le bon fonctionnement du système.

3.4.1.2 Communications temps réel mou

Définition 58. *(**Communications temps réel mou**)*

Les communications temps réel mou sont les communications temps réel où nous pouvons tolérer un dépassement occasionnel d'une échéance sans causer des dégâts graves [8].

Dans le contexte des communications temps réel mou, respecter toutes les échéances n'est souhaitable que pour des raisons de performance. Un exemple classique de communication temps réel mou est celui des systèmes multimédias.

3.4.2 Qualité de service

Afin de répondre aux exigences des applications temps réel, le NoC doit fournir une qualité de service donnée pour chaque communication.

Définition 59. *(**Qualité de service (QoS)**)*

La qualité de service (QoS) indique la capacité du NoC à respecter les exigences de communication spécifiées par l'application [47].

Les services attendus de la part d'un NoC sont que les communications soient correctes, complètes et qu'elles respectent les contraintes de performance. Le temps de communication et la bande passante fournis par le réseau sont des exemples de contraintes de performance.

Dans ce contexte, les architectures NoC sont classées en deux principales catégories : services meilleur effort (ou en anglais *Best Effort (BE)*) et services garantis (ou en anglais *Guaranteed Services (GS)*).

3.4.2.1 Meilleur effort

Définition 60. *(**Meilleur effort**)*

Un NoC meilleur effort garantit seulement que les données échangées seront correctement et complètement transmises. Il n'offre pas de garantie sur les performances [5].

Les NoC TILEPro64 [48], UT TRIPS [49] et Intel TeraFLOPS [50] sont des exemples de NoC meilleur effort.

3.4.2.2 Services garantis

Définition 61. *(**Services garantis**)*

Un NoC services garantis offre non seulement une communication correcte et complète mais également avec des performances garanties [5].

Nexus [51], QNoC [52], TTNoC [53] et Aelite [54] sont des exemples de NoC à services garantis. Ces réseaux garantissent les contraintes sur la bande passante offerte et les contraintes sur le temps de communication.

3.5 Ordonnancement des communications temps réel

L'analyse du temps de communication est obligatoire pour étudier l'ordonnancement des communications temps réel. Cette analyse doit prendre en compte les spécifications du réseau sur puce. Autrement dit, elle doit considérer tous les types d'interférences possibles dues aux ressources partagées dans un NoC.

Dans cette partie, nous introduisons un modèle de flux. Ensuite, nous discutons de l'analyse du temps de communication en fonction de la technique de commutation adoptée. Nous expliquons le temps de trajet et sa dépendance aux techniques de commutation. Puis nous décrivons les interférences possibles dans un NoC. Finalement, nous présentons brièvement quelques analyses de pire temps de communication.

3.5.1 Modèle de flux

Définition 62. *(**Flux**)*

Dans le contexte d'un NoC, un flux est un ensemble de messages échangés entre une tâche émettrice et une tâche réceptrice à travers l'infrastructure de communication du NoC [8].

Plusieurs modèles de flux ont été proposés pour modéliser les communications au sein d'un NoC. Ces modèles dépendent de la configuration du NoC considéré [32, 11]. Dans la suite, nous présentons le modèle proposé par A. Burns dans [11].

Le système est constitué d'un ensemble de m flux $\psi = \{ \rho_1, ..., \rho_m \}$

Chaque flux ρ_i est caractérisé par un ensemble d'attributs $\rho_i = (O_i, C_i, D_i, P_i, Crit_i)$

- O_i représente la date du premier réveil du flux ρ_i.

- C_i représente le pire temps de communication de tous les messages du flux ρ_i.
- D_i représente l'échéance du flux ρ_i. C'est-à-dire l'instant auquel la transmission d'un message doit être terminée et dont le dépassement provoque une violation de la contrainte temporelle.
- P_i désigne la période du flux ρ_i. Elle représente la durée séparant deux instants de réveils successifs.
- $Crit_i$ désigne le niveau de critcité du flux ρ_i.

Ce modèle a été considéré dans de nombreux travaux qui traitent des communications temps réel au sein de NoC [38], [55].

Le calcul de C_i dépend de la configuration du NoC. Dans la suite, nous expliquons les analyses du temps de communication pour différentes configurations de NoC.

3.5.2 Temps de trajet

Définition 63. *(**Temps de trajet**)*

Le temps de trajet (noté PD) (ou en anglais Path Delay (PD)) est la durée de communication d'un message sans interférence entre les flux [8].

Autrement dit, PD est la durée nécessaire pour la transmission d'un paquet de la source vers la destination tout en supposant qu'il n'y a pas d'interférences dans le réseau. PD dépend de la distance entre la source et la destination, de la taille du paquet, de la bande passante et de la technique de commutation utilisée. Dans la suite nous présentons l'expression de PD pour les différentes techniques de commutation [32] :

PD pour SAF

$$PD_{SAF} = (size_{max}/B_{link} + R_{hop}) \cdot Hops \tag{3.1}$$

PD pour Wormhole

$$PD_{wormhole} = \lceil size_{max}/Flit_{size} \rceil \cdot Flit_{size}/B_{link} + (R_{hop} \cdot Hops) \tag{3.2}$$

où

- R_{hop} représente le temps de routage pour chaque routeur.
- B_{link} représente la bande passante du lien entre deux routeurs voisins.

- $size_{max}$ est la taille maximale du paquet.
- $Hops$ est le nombre de sauts entre la source et la destination.
- $Flit_{size}$ est la taille d'un flit.

En adoptant la technique SAF, un routeur doit attendre la réception de la totalité du paquet pour pouvoir le transmettre au routeur suivant. Cependant, en utilisant la technique Wormhole, un routeur peut envoyer les flits reçus au routeur suivant sans attendre la réception de la totalité de message.

3.5.3 Pire temps de communication

Définition 64. *(**Pire temps de communication**)*

Le pire temps de communication se produit lorsque le paquet provenant du flux observé se confronte à tous les paquets des autres flux qui partagent les mêmes ressources avec leurs tailles maximales [8].

Le pire temps de communication dépend principalement de la configuration du NoC considéré et des types d'interférences entre les flux. Dans un premier temps, nous détaillons les situations d'interférences possibles dans un NoC. Dans un second temps, nous listons les analyses de pire temps de communication en fonction de la configuration du NoC.

3.5.3.1 Interférences

Les interférences entre les flux dans un réseau sur puce sont principalement les interférences directes et les interférences indirectes.

Définition 65. *(**Interférence directe**)*

Une situation d'interférence directe se produit lorsque 2 flux demandent au même moment le même lien physique [32].

Définition 66. *(**Interférence indirecte**)*

Une situation d'interférence indirecte se produit lorsque deux flux ne partagent aucun lien physique, mais interfèrent tous les deux directement avec le même flux [32].

Afin d'expliquer les situations d'interférence directe et indirecte, nous prenons l'exemple présenté sur la figure 3.8. Dans cet exemple, nous considérons trois flux ρ_1, ρ_2 et ρ_3.

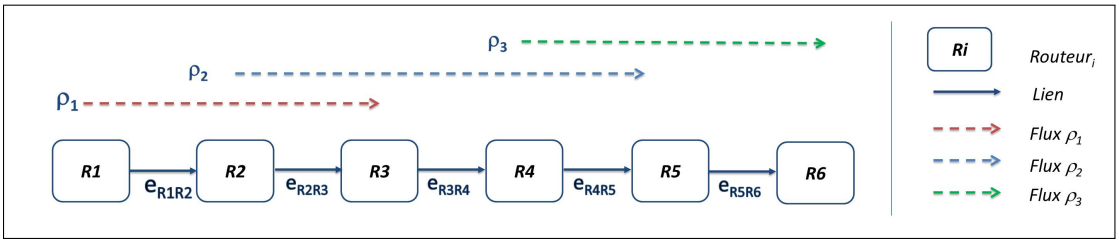


FIGURE 3.8 – Interférences des flux dans un NoC.
Le flux ρ_1 utilise les liens e_{R1R2} et e_{R2R3} ; le flux ρ_2 utilise les liens e_{R2R3}, e_{R3R4} et e_{R4R5} ; le flux ρ_3 utilise les liens e_{R4R5} et e_{R5R6}.

ρ_1 et ρ_2 demandent au même moment le lien physique e_{R2R3}. Nous sommes donc devant une situation d'interférence directe entre les flux ρ_1 et ρ_2.

Il y a également un interférence directe entre les flux ρ_2 et ρ_3. Ces deux flux partagent au même moment le lien e_{R4R5} .

ρ_1 et ρ_3 ne partagent aucun lien physique mais ρ_1 est bloqué par ρ_2 qui est lui-même bloqué par ρ_3. Les flux ρ_1 et ρ_3 ont des interférences directes avec le même flux ρ_2 Nous sommes donc devant une situation d'interférence indirecte entre les flux ρ_1 et ρ_3.

3.5.3.2 Analyse du pire temps de communication

L'analyse du pire temps de communication dépend essentiellement de la configuration du NoC. Les situations d'interférences (les interférences directes et indirectes) sont les mêmes. Les temps de communication suite à ces situations dépendent de la configuration du NoC. Le niveau de préemption, le type d'arbitrage et la technique de commutation sont des paramètres influant sur le calcul du pire temps de communication [5].

La figure 3.9 présente un exemple de deux temps de communication différents pour le même type d'interférence et pour deux configurations de NoC différentes. Dans cet exemple, nous considérons deux flux ρ_1 et ρ_2. Ces deux derniers partagent le même lien physique e_{R2R3} et par la suite, nous sommes devant une situation d'interférence directe entre ρ_1 et ρ_2. Dans la figure 3.9 (a), nous présentons le comportement de la première configuration du NoC (NoC_1) face à cette interférence. Dans la figure 3.9 (b), nous présentons le comportement de la deuxième configuration (NoC_2).

NoC_1 utilise un arbitrage *Round Robin* avec une préemption au niveau paquet. Donc, le flux ρ_1 doit attendre le passage du ρ_2. Ce temps d'attente qui est produit suite à cette situation d'interférence égale au temps de passage du flux ρ_2.

NoC_2 implante un arbitrage à priorité calculée avec une préemption au niveau flit. Autrement dit, le flux ρ_1 préempte le flux ρ_2 au niveau flit et donc le temps

d'attente produit dans cette situation d'interférence revient au temps de passage d'un seul flit.

En conclusion, malgré le fait que les deux exemples présentés dans la figure 3.9 traitent la même situation d'interférence, les temps de communication produits suite à cette interférence ne sont pas les mêmes.

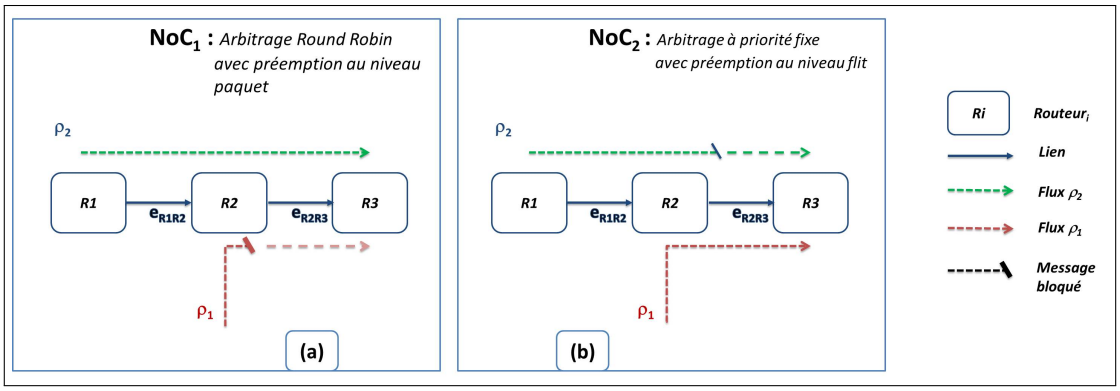


FIGURE 3.9 – Temps de communication pour une situation d'interférence directe

Ces dernières années, plusieurs analyses de pire temps de communication ont été proposées selon différentes configurations de NoC.

Ainsi, Qian et al. utilisent *network-calculus* afin de calculer le pire temps de communication pour un *Wormhole* NoC [56].

Shi et al. calculent le pire temps de communication pour un *Wormhole* NoC qui implante une préemption au niveau flit et des canaux virtuels [57]. Cette analyse a été améliorée dans [58].

Enfin, [59] et [38] proposent une analyse de pire temps de communication pour un Wormhole NoC avec une politique de partage de canaux virtuels.

3.6 Réseaux sur puce et systèmes à criticité mixte

Dans le cadre des tâches dépendantes déployées sur des architectures NoC, l'analyse d'ordonnancement des tâches seulement n'est plus suffisante pour décider de l'ordonnancement du système. Nous devons également étudier l'ordonnancement des communications afin de pouvoir conclure sur l'ordonnancement du système [6].

Plusieurs travaux ont traité ce sujet ces derniers années en proposant des routeurs NoC et des protocoles qui répondent aux exigences des systèmes à criticité mixte. La conception des NoC qui supportent plusieurs classes de communication représente les premiers efforts pour cette intégration [6, 5]. Dans la suite, nous

TABLE 3.2 – Tableau récapitulatif pour les NoC qui supportent les communications GS et BE

Ref	NoC	Topologie	Technique de commutation	Algorithme de routage	QoS
[60]	Æthereal	2D maillée	Circuit et paquet (*Wormhole*)	source	GT BE
[61]	SoCBUS	2D maillée	Commutation de circuit et SAF	Packet connected circuit	GT BE
[62]	Nostrum	2D maillée	Paquet	Hot-Potato	GT, BE
[69] [70]	μSpider		*Wormhole*	Routage XY street-sign	GT BE
[64]	Mango	2D maillée	Circuit et paquet	Source	GT, BE
[65]	DSPIN	2D maillée	*Wormhole*	Routage XY	BE, GT
[68]	SuperGT	2D maillée	*Wormhole*	Routage XY	BE, GT

présentons, brièvement, les NoC qui supportent différentes classes de communication. Ensuite, nous décrivons les protocoles qui ont été conçus pour supporter les systèmes à criticité mixte.

3.6.1 Architectures du réseau sur puce

Parmi les NoC qui ont été proposés ces dernières années, nous trouvons des architectures meilleur effort, des architectures à services garantis et des architectures hybrides. Dans la suite, nous présentons brièvement chacune de ces approches. Ensuite, nous détaillons les NoC qui supportent différentes classes de communication.

Les architectures NoC meilleur effort et NoC à services garantis sont présentés respectivement dans les paragraphes (3.4.2.1 et 3.4.2.2)

Réseau sur puce hybride

Ces dernières années, plusieurs efforts ont été faits pour déployer des systèmes à criticité mixte sur un réseau sur puce. Les premiers efforts étaient la conception de NoC qui supportent à la fois des communications temps réel dur et des communications temps réel mou [6].

Æthereal [60], SoCBUS [61], Nostrum [62],[63], Mango [64], DSPIN [65], artNoC [66] ALPIN [67] et SuperGT [68] sont des architectures NoC qui supportent les deux classes de communications.

Dans le tableau 7.3, nous détaillons les caractéristiques de ces NoC. Pour chaque NoC, nous présentons les principales caractéristiques : topologie, technique de commutation, algorithme de routage et la qualité de service fournie.

Les NoC DSPIN [65], μSpider [68] , SuperGT [68] et Æthereal [60] combinent les canaux virtuels avec la technique TDMA afin d'assurer une isolation entre les différentes classes de communication.

Dans ces architectures, la technique TDMA est utilisée pour assurer les contraintes temporelles des communications temps réel dur. Les tables TDMA sont utilisées dans les interfaces réseau pour autoriser l'entrée des paquets dans le réseau selon un séquencement pré-ordonnancé afin d'éviter tout type d'interférence.

μSpider [69] et SuperGT [68] utilisent la technique de commutation Wormhole. Æthereal, SoCBUS et Mango combinent les techniques de commutation de circuit et de paquet.

Le routage XY est utilisé dans DSPIN, SuperGT et μSpider. Æthereal et Mango utilisent le routage par la source.

SuperGT [68] peut être considéré comme une évolution d'Æthereal [60]. Dans SuperGT, les communications temps réel mou utilisent les time-slots libres du TDMA.

μSpider utilise deux techniques d'arbitrages. Les communications temps réel mou peuvent utiliser l'arbitrage à priorité fixe et l'arbitrage à priorité tournante.

Mango est un NoC asynchrone qui fournit également des services BE et GT. Afin de garantir les communications temps réel dur, il exige deux conditions. Premièrement, le délai maximum de transmission d'un flit est connu. Deuxièmement, l'intervalle inter-flits est borné. Il exploite également plusieurs canaux virtuels.

Dans ces architectures, le routeur gère à la fois les deux classes de communications temps réel (dur et mou). Il y a plusieurs techniques qui peuvent être utilisées dans le routeur NoC pour assurer le partage de ressources entre ces deux classes de communications.

La technique TDMA fournit une isolation entre les différents classes de communication. Toutefois, elle sous-utilise les ressources. Nostrum, Æthereal et DSPIN utilisent cette technique.

Une autre politique possible consiste à utiliser une priorité calculée dans le but d'assurer les contraintes temporelles pour les communications temps réel dur. Dans ce contexte, certains NoC utilisent des canaux virtuels avec des priorités différentes. Les communications temps réel dur sont plus prioritaires par rapport aux communications temps réel mou.

3.6.2 Protocoles pour isolation ou criticité mixte

Plusieurs protocoles ont été proposés pour déployer des systèmes à criticité mixte sur des architectures NoC. Dans la suite, nous décrivons certains de ces protocoles.

3.6.2.1 *Wormhole NoC Protocol for Mixed-Criticality Systems (WPMC)*

Burns et al. ont proposé un protocole basé sur la technique Wormhole WPMC [11]. En se basant sur les travaux [55], [38] et [59], WPMC ajoute le concept de criticité dans l'analyse des communications. Avec ce protocole, le routeur fonctionne sous deux modes de criticité : mode critique (HI) et mode non critique (LO). Ce protocole a été amélioré dans [71].

3.6.2.2 *Time-Triggered Extension Layer (TTEL)*

Ahmadian et al. ont proposé un composant appelé *Time-Triggered Extension Layer (TTEL)* [72]. Ce composant doit être integré dans les interfaces réseau afin de supporter les systèmes à criticité mixte. Avec ce composant, le NoC peut supporter plusieurs classes de communication tels que les communications temps réel dur et les communications temps réel mou.

Au niveau de la couche transport, TTEL impose les périodes et les phases pour les communications temps réel dur afin d'éviter les interférences entre les communications de différentes classes. Les communications temps réel mou utilisent le reste de la bande passante qui n'est pas réservée.

Au niveau de la couche réseau, TTEL gère l'attribution des priorités aux messages.

Un routage par la source est requis pour utiliser TTEL. En utilisant cet algorithme de routage, le nœud source prend toutes les décisions sur le chemin du paquet.

3.6.2.3 Run-Time Configurable NoC

Tobuschat et al. ont proposé un NoC configurable à l'exécution qui supporte plusieurs classes de communication [73]. Contrairement à tous les autres protocoles, les communications temps réel mou sont plus prioritaires que les communications temps réel dur.

L'attribution de priorité dans cette architecture est dynamique. L'entête de flit est étendu avec un champ supplémentaire qui retient la priorité de chaque flit.

Dans cette approche, les auteurs ne prennent pas en compte le mode de criticité. De plus, pour les longs paquets, l'ajout des informations supplémentaires à chaque flit peut entraîner des latences supplémentaires.

Enfin, l'utilisation de la stratégie Wormhole avec une priorité partagée et une préemption au niveau flit complique la prédiction de temps de communication en augmentant l'écart entre le temps de trajet et le pire temps de communication.

3.6.2.4 NoCDepend

[74] propose également une architecture du NoC qui supporte les communications temps réel dur et les communications temps réel mou. Cette architecture assure qu'il n'y aura pas d'interférences entres les deux classes de communication. Elle distingue deux régions de communications pour assurer l'isolation entre les deux classes de communication.

3.6.2.5 QoSinNoC

[75] propose une architecture du NoC qui permet de supporter les communications temps réel dur et les communications temps réel mou. Contrairement à [74], cette architectures n'isole pas les deux classes de communication. Les communications temps réel mou peuvent utiliser les régions critiques sans avoir d'interférence.

3.6.3 ARTEMIS

[76] propose un outil de simulation dénommé Another Real-Time Engine for Message Integration Simulation (ARTEMIS). Cet outil est utilisé pour l'analyse des temps de communication dans des réseaux temps-réel et pour l'analyse de scénarios d'ordonnancement à criticité mixte. Cet outil intègre deux types de protocoles. Le premier est un protocole centralisé. Il est organisé autour de la désignation d'un noeud central dans le réseau, responsable de la synchronisation des niveaux de criticité de chaque noeud via un mécanisme d'émission multiple. Le second protocole est basé sur une approche distribuée. Il propose une gestion locale à chaque noeud de la criticité.

3.6.4 Bilan : NoC et système à criticité mixte

Les architectures basées sur TDMA introduisent une sur-réservation statique due au calendrier temporel, réduisant la performance en termes d'utilisation de ressources dans la plupart des cas.

La plupart des approches avec QoS dynamique favorisent les communications temps réel dur sans chercher à améliorer les communications temps réel mou.

3.7 Conclusion

Le réseau sur puce présente une solution prometteuse pour les architectures SoC qui surmontent les limites des techniques d'interconnexion classiques. Il permet de construire des architectures flexibles et extensibles. Augmenter les performances de calcul et réduire le coût en surface représentent les principaux avantages des NoC. Par ailleurs, ils peuvent supporter différentes classes de communication.

Actuellement, l'intégration des systèmes à criticité mixte sur des architectures NoC présente un défi pour les concepteurs. Plusieurs NoC et protocoles de NoC ont été proposés dans cet objectif.

Dans ce chapitre, nous avons identifié les enjeux actuels liés à la maîtrise des communications sur puce. Ensuite, nous avons détaillé les caractéristiques des NoC. Puis, nous avons présenté l'architecture des routeurs NoC et leurs fonctionnalités au sein du réseau sur puce. Nous avons présenté également les différentes classes de communications.

Dans le chapitre suivant, nous allons expliquer et détailler les problématiques étudiées et les contributions proposées dans le cadre de cette thèse, puis les hypothèses prises dans ce travail.

4

Motivations et hypothèses

Sommaire

4.1 Introduction

Après avoir introduit les notions fondamentales des systèmes à criticité mixte et des réseaux sur puce dans les chapitres précédents, nous discutons dans ce chapitre des motivations, des objectifs et des contributions de cette thèse.

Tout d'abord, nous détaillons les problématiques étudiées. Nous montrons l'incompatibilité des routeurs NoC existants avec les exigences des systèmes à criticité mixte. Nous présentons, ainsi, l'inadéquation des techniques d'analyses d'ordonnancement classiques avec les systèmes temps réel déployés sur des architectures NoC. Ensuite, nous décrivons le modèle du système considéré ainsi que les notations utilisées. Puis, nous expliquons les hypothèses posées pour ce travail. Finalement, nous décrivons brièvement les contributions de cette thèse.

4.2 Problématiques étudiées

Dans cette partie, nous discutons des motivations de ce travail. Nous présentons les problèmes qui empêchent l'intégration des systèmes à criticité mixte sur des architectures NoC aujourd'hui.

4.2.1 Incompatibilité des routeurs NoC avec les systèmes à criticité mixte

Pour déployer des systèmes à criticité mixte sur des architectures NoC, les routeurs doivent assurer les contraintes temporelles des application critiques tout en minimisant l'impact du partage de ressources sur les applications non critiques [6].

Les routeurs NoC existants ne répondent pas aux besoins des systèmes à criticité mixte. Les routeurs NoC qui supportent des systèmes temps réel peuvent être classés en trois catégories : les routeurs avec un arbitrage TDMA, les routeurs qui implantent un arbitrage à priorité calculée et les routeurs qui combinent ces deux derniers [77]. Dans la suite, nous montrons l'inadéquation de ces classes de routeurs avec les exigences des systèmes à criticité mixte.

4.2.1.1 Routeurs TDMA

Afin d'éviter les interférences dans le réseau sur puce, certains routeurs NoC implantent l'arbitrage TDMA. Cette politique d'arbitrage divise le temps en périodes courtes appelées *time-slot*. Chaque *time-slot* est attribué à une communication. Autrement dit, les ressources partagées seront entièrement réservées à la communication concernée durant toute la durée du *time-slot*. Cette politique

réduit les besoins en termes de tampons mémoires et assure une isolation entre les différentes classes de communications [5].

La figure 4.1 présente un exemple d'utilisation d'un tampons mémoire d'un routeur TDMA. Dans cet exemple, nous considérons 3 flux : 1 flux critique et 2 flux non critiques. Les 3 flux partagent le même tampon. Le routeur attribue à chaque flux des *time-slots* différents.

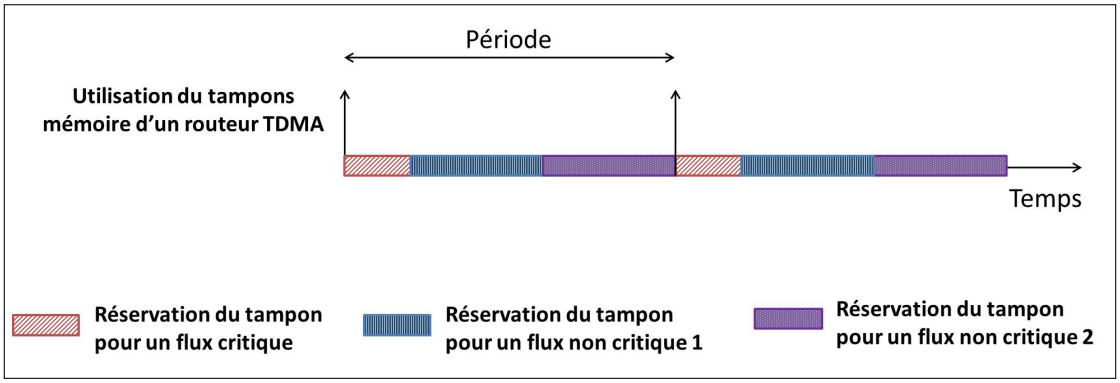


FIGURE 4.1 – Réservation des ressources dans un routeur TDMA

L'isolation entre les différentes classes de communication affecte significativement les flux non critiques. En effet, afin de garantir les contraintes temporelles pour les flux critiques, le calcul de largeur des *time-slots* est basé généralement sur les pires scénarios. En conséquence, le réseau sur-réserve les ressources pour les flux critiques [6]. Les routeurs TDMA ne permettent donc pas aux applications non critiques d'exploiter efficacement les ressources non utilisées par les applications critiques.

4.2.1.2 Routeurs avec un arbitrage à priorité calculée

Les routeurs de cette classe implantent un arbitrage à priorité calculée. En cas d'interférence dans le réseau, le flux le plus prioritaire passe avant les autres flux. Cette politique d'arbitrage peut être implantée au niveau paquet (non préemptive) comme au niveau flit (préemptive). Autrement dit, les paquets des flux prioritaires peuvent préempter ou non les autres flux selon le niveau de préemption adopté [5].

Wormhole est la technique de commutation la plus utilisée dans cette classe du NoC. Elle fournit un taux d'utilisation du réseau très élevé par rapport aux autres techniques de commutation. En plus, elle requiert des tampons mémoires de petites tailles [5].

Cette classe de routeur ne répond pas aux exigences des systèmes à criticité mixte : soit elle est utilisée pour assurer uniquement les communications critiques sans offrir un taux d'utilisation important pour les communications non

critiques, soit, elle conduit à un taux d'utilisation très important pour les communications non critiques mais sans assurer les contraintes temporelles pour les communications critiques.

4.2.1.3 Routeurs hybrides

Afin de supporter différentes classes de communication, plusieurs NoC combinent l'arbitrage basé sur la priorité calculée avec l'arbitrage TDMA. Nous pouvons citer à titre d'exemple les NoC DSPIN [65] et Mango [64].

La technique TDMA est utilisée afin d'assurer une isolation entre les communications de différentes classes. L'arbitrage basé sur la priorité calculée permet d'améliorer la qualité des communications non critiques.

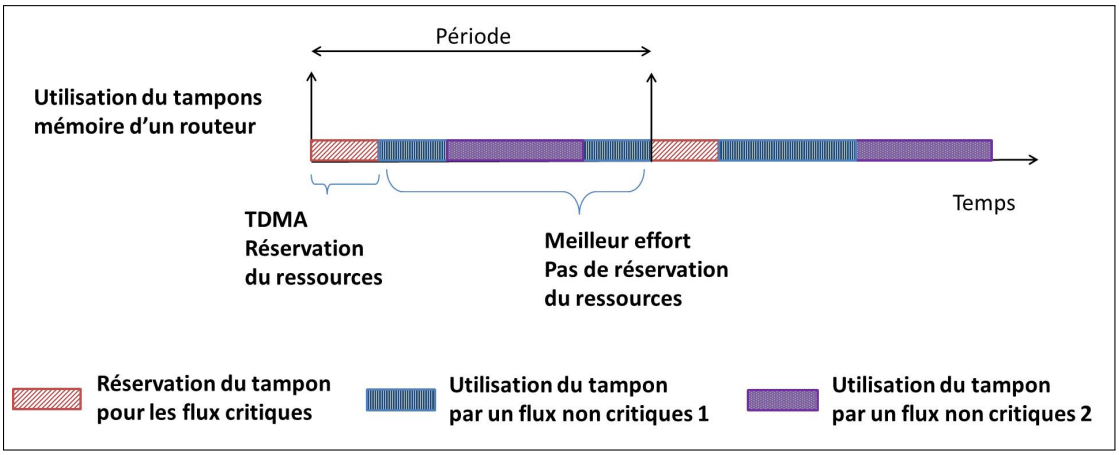


FIGURE 4.2 – Utilisation des ressources dans un routeur hybride

La combinaison entre l'arbitrage TDMA et l'arbitrage basé sur la priorité calculée représente un pas important vers l'intégration des systèmes à criticité mixte. Grâce à cette combinaison, certains NoC ont réussi à supporter plusieurs classes de communications.

La figure 4.2 présente un exemple d'utilisation d'un tampon mémoire d'un routeur hybride. Dans cet exemple, nous considérons 3 flux : 1 flux critique et 2 flux non critiques. Les 3 flux partagent le même tampon. Le routeur réserve un *time-slot* pour le flux critique et tout le reste pour les flux non critiques. L'utilisation du TDMA entre les flux ayant des niveaux de criticité différents assure l'isolation entre ces flux. Les flux non critiques partagent le reste de temps en adoptant la politique meilleur effort. Nous pouvons avoir des interférences et des préemptions entre les flux non critiques.

Les architectures ou les protocoles qui sont basés sur l'arbitrage TDMA surréservent les ressources pour les communications critiques ce qui affecte les com-

munications non critiques [5]. La sur-réservation de ressources pour les communications critiques est le principal inconvénient de cette combinaison.

4.2.2 Modèles de tâches incomplets

Les modèles de tâches existants négligent les communications et les différents interférences possibles dans le réseau et leur impact sur le fonctionnement du système [6].

Ces dernières années, plusieurs modèles de flux ont été proposés dans l'objectif d'analyser les communications au sein d'un réseau sur puce. Ces modèles prennent en compte les spécifications du réseau sur puce, mais ils négligent les tâches et leurs impacts sur les dates d'émissions et les échéances des flux [8].

Les modèles de tâches et les modèles de flux existants sont donc incomplets pour modéliser un système temps réel déployé sur un réseau sur puce dans son intégralité.

4.2.3 Techniques d'analyse d'ordonnancement incomplètes

Afin d'analyser l'ordonnancement des systèmes temps réel déployés sur des architectures NoC, nous devons analyser à la fois l'exécution des tâches et les communications au sein du réseau. Dans un NoC, les communications partagent plusieurs ressources comme les liens physiques et les routeurs. Des nouveaux types d'interférences ont été introduits par les NoC suite au partage de ressources entres les différentes communications.

Les modèles des communications considérés dans les techniques d'analyse d'ordonnancement actuelles ignorent les spécificités des NoC, les interférences et leurs impacts sur l'ordonnancement du système. Ces modèles ne prennent pas en compte la technique de commutation, le niveau de préemption et la technique d'arbitrage utilisés par le routeur.

Un modèle de communication qui considère les différents types de conflits possibles dans un réseau sur puce est indispensable pour pouvoir analyser l'ordonnancement d'un système temps réel déployé sur un NoC.

4.3 Contexte du travail

Dans cette partie, nous présentons, le modèle du système de criticité mixte utilisé dans le cadre de ce travail. Ensuite, nous décrivons le modèle du NoC considéré.

4.3.1 Modèle de système à criticité mixte

Nous considérons le modèle de Vestal [78] pour les systèmes à criticité mixte.

Selon le modèle de Vestal, un système à criticité mixte est constitué par un ensemble de tâches ayant des niveaux de criticité différents et qui fonctionnent sous des modes de criticité différents.

Hypothèse 1. *Dans ce travail, nous considérons deux niveaux de criticité et deux modes de criticité.*

Nous notons ici que les résultats obtenus et les contributions proposées dans le cadre de ce travail ne sont pas généralisables à plus de deux niveaux de criticité et deux modes de criticité.

Dans la suite, nous détaillons les deux niveaux et les deux modes de criticité considérés.

4.3.1.1 Deux niveaux de criticité

Le système est composé par un ensemble de n tâches périodiques $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Chaque tâche τ_i est caractérisée par un ensemble de paramètres : $\tau_i = \{ O_i, C_i, D_i, P_i, \Pi_i, Crit_i \}$

$Crit_i$ représente le niveau de criticité. Nous considérons deux niveaux de criticité pour les tâches :

- Tâches HIGH (critiques)

  Nous appelons "tâches HIGH (ou critiques)" les tâches ayant un niveau de criticité élevé ($Crit_i$ = HIGH). Les tâches critiques sont des tâches temps réel dur. Aucun dépassement d'échéance n'est autorisé.

- Tâches LOW (non critiques)

  Nous appelons "tâches LOW (ou non critiques)" les tâches ayant un niveau de criticité faible ($Crit_i$ = LOW). Les tâches non critiques sont des tâches temps réel mou. Certains dépassements d'échéance peuvent être tolérés.

Les tâches considérées dans ce modèle sont des tâches périodiques dépendantes. Autrement dit, les tâches se communiquent entre eux en échangeant des messages. Par la suite la dépendance entre les tâches du modèle considéré génère un modèle de flux.

Le système déployé sur un NoC est également composé par un ensemble de m flux $\psi = \{\rho_1, \rho_2, \ldots, \rho_m\}$. Chaque flux ρ_i est caractérisé par un ensemble de paramètres : $\rho_i = \{ O_i, C_i, D_i, P_i, Crit_i \}$

$Crit_i$ représente la criticité de flux ρ_i. Nous considérons également, pour les flux, deux niveaux de criticité :

- Flux HIGH (critiques)

  Nous appelons les flux ayant un niveau de criticité élevé ($Crit_i$ = HIGH) des "flux HIGH (ou critiques)". Les flux critiques sont des communications temps réel dur. Le respect des contraintes temporelles pour ce type de communication est obligatoire.

- Flux LOW (non critiques)

  Nous appelons les flux ayant un niveau de criticité faible ($Crit_i$ = LOW) des "flux LOW (ou non critiques)". Ces flux sont des communications temps réel mou. Un certain dépassement d'échéance peut être toléré pour les flux non critiques.

4.3.1.2 Deux modes de criticité

Dans le cadre des systèmes à criticité mixte, le mode de criticité définit les paramètres d'exécution du système.

Nous considérons dans ce travail deux modes de criticité :

- NORMAL

  Dans le mode NORMAL, le système garantit l'ordonnancement de toutes les tâches et tous les flux.

- DEGRADED

  Dans le mode DEGRADED, le système ne garantit que les tâches critiques et les flux critiques. Dans ce mode, les flux non critiques et les tâches non critiques peuvent être suspendus et leurs échéances ne sont pas garanties.

Les paramètres D_i et C_i pour les tâches et les flux dépendent du mode de criticité. Dans les chapitres 5 et 6, nous fournissons plus de détails concernant les modes de criticité et le protocole de changement de mode adopté.

Dans la suite, nous exposons les hypothèses posées dans le modèle de système à criticité mixte.

Hypothèse 2. *Nous supposons que chaque message hérite du niveau de criticité de sa tâche émettrice.*

Autrement dit, nous supposons que les messages émis par des tâches critiques sont également des messages critiques. Les messages émis par des tâches non critiques sont des messages non critiques.

Dans le cadre de ce travail, nous avons proposé un modèle de tache flux appelé DTFM. Ce modèle permet de calculer le niveau de criticité des messages en se basant sur le niveau de criticité des tâches émettrices.

Hypothèse 3. *Nous supposons que la communication entre deux tâches différentes se fait via uniquement un seul message.*

Hypothèse 4. *Nous supposons qu'un message est composé d'un seul paquet.*

Hypothèse 5. *Nous supposons que la taille des messages critiques ne peut pas dépasser 3 flits de 32 bits.*

Nous avons fondé cette hypothèse sur les caractéristiques des applications critiques et les supports d'éxecution les plus utilisées pour les systèmes temps réel dur.

Les applications critiques sont généralement des systèmes de contrôle/commande qui sont caractérisés par l'échange de messages de petite taille avec des contraintes temporelles fortes. Par exemple, dans le domaine de l'automobile, la taille de la charge utile du contrôleur réseau local (CAN) ne dépasse pas 8 octets [79]. Dans le domaine de l'aéronautique, la taille de la charge utile dans le protocole ARINC429 est limitée par 24 bits [80]. Dans l'application Rosace [81], la taille des messages échangés entre les tâches ne dépasse pas 3 flits de 32 bits.

Hypothèse 6. *Nous supposons dans ce travail que la communication entre les tâches se fait selon le protocole* AADL IMMEDIATE CONNEXION DATA*, qui est défini dans [82].*

Ce protocole fonctionne comme suit :

- Les tâches sources envoient leurs messages à la fin de leurs exécutions.
- Les messages doivent être lus au début de l'exécution des tâches destinations.

4.3.2 Modèle de NoC

Tout au long de ce travail, nous considérons la topologie grille. Les paramètres de la configuration du routeur utilisé tels que la technique de commutation, la politique de routage ou la politique d'arbitrage sont définies dans le chapitre 5.

Par ailleurs, le placement des tâches dans un NoC n'est pas traité dans le présent document :

Hypothèse 7. *Nous supposons que le placement des tâches dans le NoC (appelé en anglais* TASK MAPPING*) est calculé et fixé.*

4.4 Les contributions

Dans l'objectif d'ordonnancer des systèmes à criticité mixte sur des architectures NoC, nous avons proposé plusieurs contributions sous la forme d'un routeur, d'un modèle de tâches et de flux et d'un modèle de communication pour les NoC.

Dans cette section nous présentons brièvement les principales contributions de cette thèse.

4.4.1 Double Arbiter and Switching Router (DAS)

Afin de répondre à la première problématique (section 4.2.1), nous avons proposé DAS (*Double Arbiter and Switching router*). DAS est un routeur NoC qui a été conçu pour répondre aux exigences des systèmes à criticité mixte. En considérant le modèle de système à criticité mixte proposé par Vestal [3], DAS supporte deux niveaux de criticité et fonctionne selon deux modes de criticité.

Le routeur proposé utilise deux techniques de commutation, deux étages d'arbitrage et une politique de préemption à deux niveaux différents. DAS achemine les messages soit en Wormhole soit en SAF selon leurs niveaux de criticité. Grâce à ses deux étages d'arbitrage dans les ports d'entrée et sortie, les flux HIGH préemptent les flux LOW au niveau flit.

La figure 4.3 présente un exemple d'utilisation de DAS. Nous considérons dans cet exemple deux flux : 1 flux critique et 1 flux non critique. Les deux flux partagent le même lien physique. Dans le mode NORMAL, les deux flux partagent le lien sans interférence. Dans le mode DEGRADED, le flux critique préempte le flux non critique et utilise le lien. Le changement de mode de criticité permet aux flux non critiques d'utiliser les ressources non utilisées par les flux critiques.

Les routeurs avec un arbitrage à priorité calculé offrent un taux d'utilisation important pour les flux non critiques mais sans assurer les contraintes temporels pour les flux critiques ce qui s'oppose aux exigences des systèmes à criticité mixte.

Pour assurer les contraintes temporelles des flux critiques, les routeurs TDMA utilisent l'isolation entre tous les flux (i.e. entre les flux critiques, entre les flux non critiques et entre les flux critiques et non critiques). L'inconvénient de l'isolation est le faible taux d'utilisation de ressources pour les flux non critiques.

Les routeurs hybrides combinent l'isolation avec l'arbitrage basé sur la priorité calculé dans l'objectif d'améliorer le taux d'utilisation des ressources pour les flux non critiques et d'amoindrir l'impact de l'isolation sur les flux non critiques. Ces routeurs gardent l'isolation entre les flux critiques et les flux non critiques afin d'assurer les contraintes temporelles pour les flux critiques. Ils utilisent également un arbitrage basé sur la priorité calculée entre les flux non critiques afin

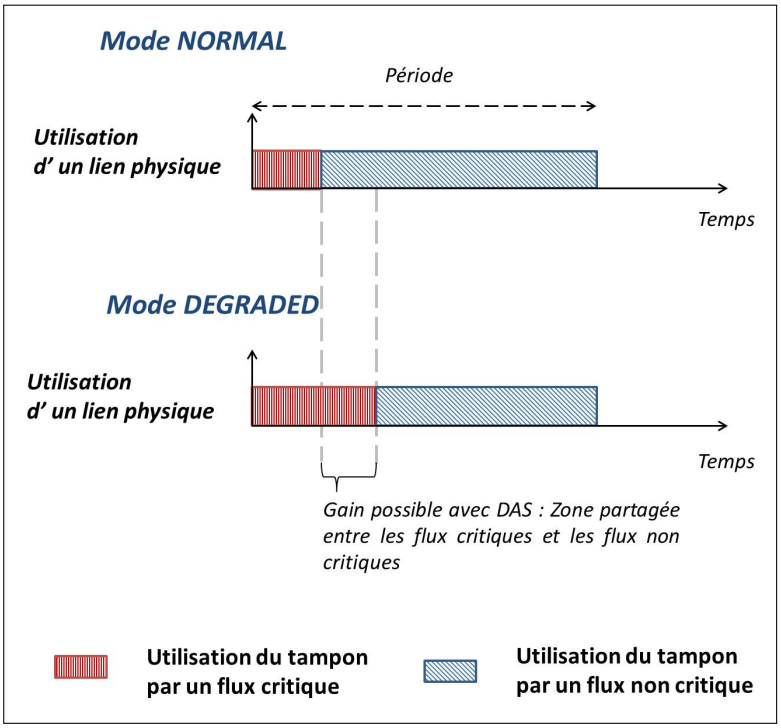


FIGURE 4.3 – Utilisation des ressources dans DAS

d'améliorer le taux d'utilisation des ressources pour les flux non critiques. Cependant, à cause de cette isolation, les flux non critiques ne peuvent pas bénéficier des ressources non utilisées par les flux critiques, ce qui présente le point faible de ces routeurs.

Donc, afin d'assurer les contraintes temporelles pour les flux critiques et d'offrir un taux d'utilisation important pour les flux non critiques, DAS propose un partage de ressources entre les flux ayant des niveaux de criticité différents sous deux modes de criticité. L'utilisation de deux modes de criticité nous permet à la fois d'assurer les contraintes temporelles pour les flux critiques et d'offrir aux flux non critiques la capacité d'utiliser les ressources non utilisés par les flux critiques.

Nous avons évalué DAS sur trois niveaux d'abstraction (circuit, transaction et système). Une évaluation du coût en surface et en performance de communication est faite pour le routeur proposé. Nous avons vérifié formellement le comportement de DAS. Une comparaison avec les routeurs concurrents montre l'efficacité du routeur proposé pour les hypothèses posées.

4.4.2 Dual Task and Flow Model (DTFM)

Nous avons proposé également une méthode appelée Dual Task and Flow Model (DTFM) pour modéliser les systèmes temps réel déployés sur un NoC. Cette

méthode est conçue pour surmonter les limitations des modèles de tâches et flux existants.

À partir du modèle de tâches, du modèle de NoC et de l'allocation des tâches, DTFM calcule le modèle de flux correspondant. DTFM prend en compte la technique de commutation adoptée par le NoC. Il supporte les techniques Wormhole et Store and Forward.

Cette contribution a été proposée pour répondre à la deuxième problématique (section 4.2.2)

4.4.3 Exact Communication Time Model (ECTM)

Dans l'objectif d'analyser l'ordonnancement des systèmes à criticité mixte sur des NoC, nous avons proposé un modèle de communication pour les architectures réseau sur puce appelé Exact Communication Time Model (ECTM).

ECTM abstrait les messages échangés via le réseau par un graphe de tâches. Avec ECTM, nous sommes capables d'analyser l'ordonnancement des tâches et des communications. Il supporte les techniques Wormhole et Store and Forward.

Cette contribution a été proposée pour répondre à la troisième problématique (section 4.2.3). La combinaison de DTFM et ECTM permet d'analyser l'ordonnancement des systèmes à criticité mixte déployés sur un NoC Wormhole ou SAF.

Par contre, ECTM n'est pas applicable directement à DAS. L'adaptation d'ECTM avec l'architecture de DAS est l'un de nos futurs travaux. Pour cette adaptation, ECTM doit supporter à la fois deux techniques de commutation, deux niveaux de préemption et deux politiques d'arbitrage.

4.5 Conclusion

Aujourd'hui, l'intégration des systèmes à criticité mixte sur des architectures NoC présente une solution prometteuse en termes de performance, de consommation d'énergie et de coût en surface et poids. Cependant, les interférences de communication au sein du NoC compliquent l'ordonnancement des systèmes à criticité mixte sur des réseaux sur puce.

Dans ce chapitre, nous avons d'abord identifié les défis abordés dans cette thèse. Ensuite nous avons exposé les hypothèses de ce travail suivi de nos contributions. Nous détaillons dans les prochains chapitres ces contributions.

Deuxième partie

Contributions

5

Routeur DAS (*Double Arbiter and Switching Router*)

Sommaire

5.1 Introduction

Dans ce chapitre, nous présentons DAS (*Double Arbiter and Switching Router*) : un routeur NoC conçu pour répondre aux exigences des systèmes à criticité mixte. DAS nous permet d'assurer les contraintes temporelles pour les applications critiques tout en minimisant l'impact du partage de ressources sur les applications non critiques. En considèrant le modèle de système à criticité mixte proposé par Vestal [3], DAS supporte deux niveaux de criticité et deux modes de criticité. Dans le but de répondre aux exigences de ce modèle, DAS implante deux techniques de contrôle de flux, deux étages d'arbitrage et une préemption à deux niveaux différents.

Ce chapitre est organisé comme suit : tout d'abord, nous décrivons l'architecture du routeur proposé. Ensuite, nous présentons le mécanisme du changement de mode utilisé dans DAS. Une analyse du pire temps de communication est présentée. Finalement, nous expliquons les raisons pour lesquelles nous avons choisi la technique SAF.

5.2 DAS (*Double Arbiter and Switching Router*)

L'architecture de DAS est représentée figure 5.1. DAS est composé de canaux virtuels, d'unités d'arbitrage d'entrée, d'unités d'arbitrage de sortie, d'une unité de contrôle et d'un crossbar.

Le routeur proposé combine 2 techniques de commutation : chaque port peut utiliser soit la technique WORMHOLE soit la technique SAF selon le niveau de criticité des messages traités. Il implante également l'algorithme de routage XY.

Dans la suite, nous détaillons le fonctionnement de DAS et nous motivons nos choix concernant les techniques de commutations utilisées. Ensuite, nous présentons le rôle de chacun des étages d'arbitrage utilisés dans DAS.

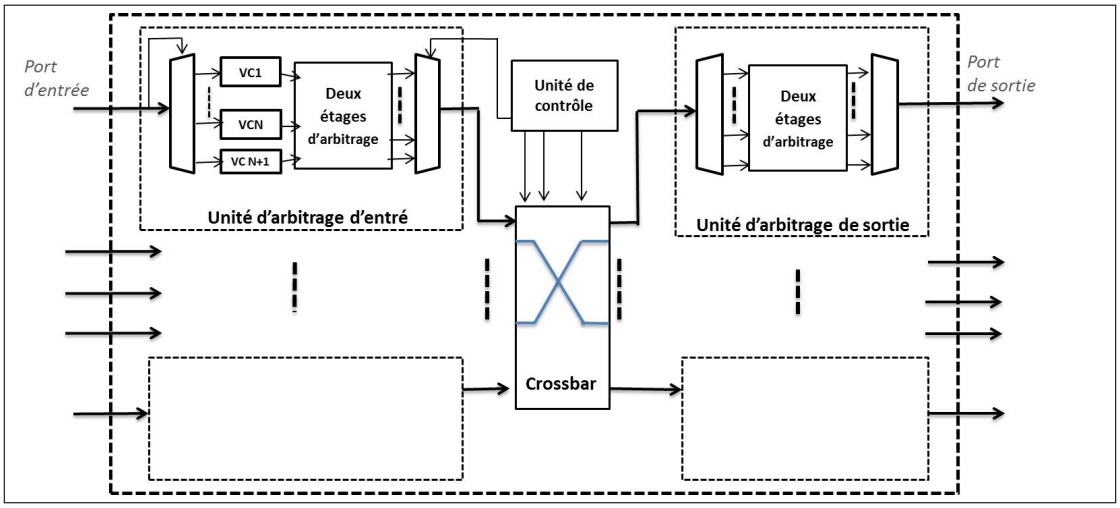


FIGURE 5.1 – Architecture du routeur DAS

5.2.1 Organisation des canaux virtuels

DAS utilise N+1 canaux virtuels. Les N premiers canaux virtuels sont dédiés aux flux HIGH, tandis que le dernier canal virtuel est dédié aux flux LOW. Les flux HIGH sont transmis selon la technique SAF, tandis que les flux LOW sont transmis selon la technique WORMHOLE.

Notons que tous les flux LOW qui partagent le même lien physique, partagent aussi le dernier canal virtuel. Contrairement aux flux LOW, chaque canal virtuel parmi les N premiers est dédié à un seul flux HIGH.

5.2.2 Technique SAF pour les flux HIGH

Les flux HIGH sont transmis selon la technique SAF. La préemption des flux HIGH est gérée au niveau du paquet. Autrement dit, un paquet HIGH ne peut pas être préempté par un autre flux.

Le principal inconvénient de cette configuration est la taille du tampon d'entrée qui doit être suffisamment importante pour stocker la totalité du paquet.

Dans notre contexte, nous considérons que cette taille est limitée, puisque nous supposons que les flux HIGH sont caractérisés par des messages de petite taille (hypothèse 5).

L'intérêt d'allouer un canal virtuel par flux est de pouvoir adapter l'analyse du pire temps de communication proposé dans [57]. Dans la partie 5.4, nous présentons l'analyse du pire temps de communication pour les flux HIGH en considérant cette configuration de DAS.

5.2.3 Technique WORMHOLE pour les flux LOW

Les flux LOW sont transmis avec la technique WORMHOLE. Cette technique est largement adoptée dans les NoC car elle réduit le temps de communication et elle requiert des tampons de petite taille [12].

Dans le but d'assurer un temps de communication prévisible et de minimiser le retard pour les flux HIGH, nous devons préempter les flux LOW dès qu'un flux HIGH est confronté à un conflit. Ainsi, pour le dernier canal virtuel (N+1), la préemption est implantée au niveau flit. Autrement dit, les flux HIGH, qui utilisent les N premiers canaux virtuels, peuvent préempter tous les flux LOW au niveau flit en cas de conflit.

5.2.4 Nombre de canaux virtuels

Le nombre N de canaux virtuels alloués aux flux HIGH ne dépend pas seulement des exigences des communications, mais aussi de l'allocation des tâches.

Si l'allocation des tâches est déjà calculée, nous pouvons choisir N comme le nombre maximal des flux HIGH qui partageant le même lien physique. Signalons ici que plus la valeur de N est élevée, plus le coût en surface pour la mise en œuvre du NoC est important.

Soit, pour un nombre N fixé, nous devons choisir une allocation des tâches qui nous permette d'avoir au plus N flux HIGH partageant le même lien physique.

Ce type de problème est similaire à un problème d'affectation quadratique (en anglais *Quadratic Assignment Problem* (QAP)). QAP est un problème d'optimisation combinatoire NP-difficile [83]. Cependant, plusieurs heuristiques ont été proposées pour résoudre ce genre de problème. Elles peuvent être utilisées pour déterminer l'allocation des tâches adéquate. [84, 83] présentent des heuristiques multicritères ayant l'objectif d'optimiser le volume total des communications et le nombre des canaux virtuels requis. Ce problème d'optimisation n'est pas abordé dans cette thèse.

5.2.5 Unités d'arbitrage

Au sein d'un routeur NoC, nous avons besoin d'un arbitre pour implanter les règles d'arbitrage pour la sélection des flux prioritaires pour les ports d'entrée et les ports de sortie du routeur.

Dans le but de préempter au niveau flit les flux LOW, DAS implante deux types d'unités d'arbitrage : une unité d'arbitrage d'entrée et une unité d'arbitrage de sortie. Comme le montre les figures 5.2 5.3, chaque unité d'arbitrage utilise deux

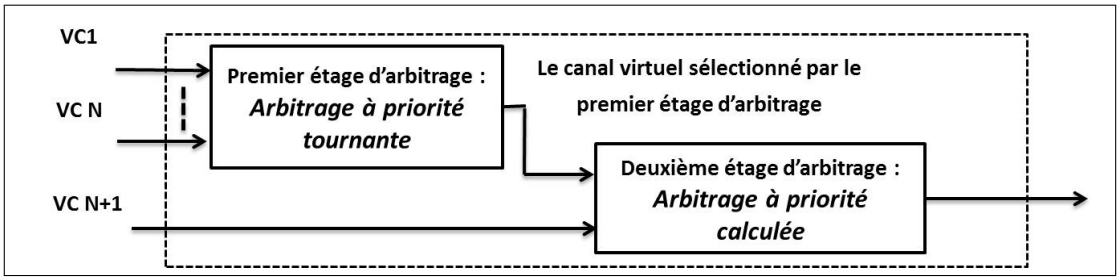


FIGURE 5.2 – Unité d'arbitrage d'entrée implanté dans DAS

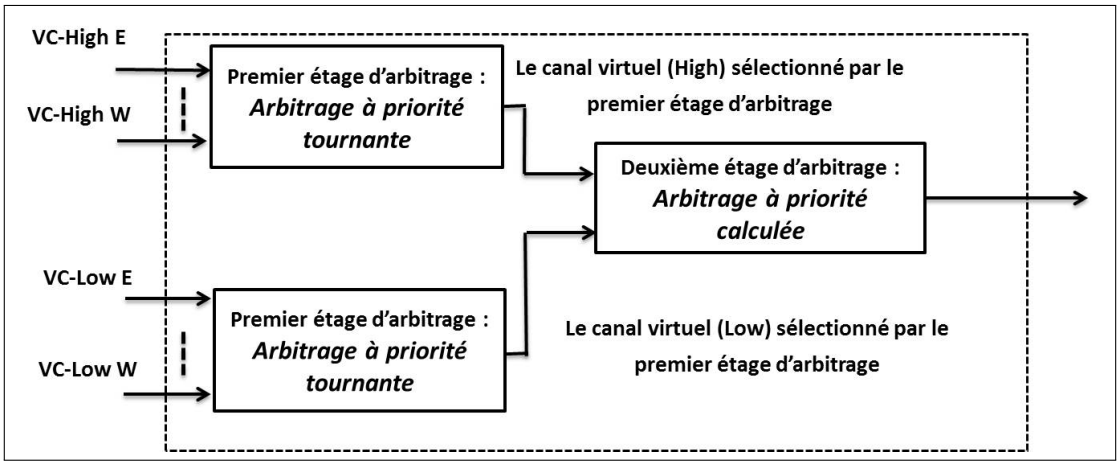


FIGURE 5.3 – Unité d'arbitrage de sortie implanté dans DAS

politiques d'arbitrages différentes : l'arbitrage à priorité calculée et l'arbitrage à priorité tournante. Dans la suite, nous décrivons ces unités.

5.2.5.1 L'unité d'arbitrage d'entrée

Pour un même port d'entrée, plusieurs canaux virtuels peuvent demander le routage de données vers des ports de sortie. Les ports de sortie demandés par ces canaux peuvent être différents ou identiques, tandis que le routeur ne peut sélectionner qu'un seul canal virtuel pour chaque port d'entrée. La fonction principale de l'unité d'arbitrage d'entrée est de sélectionner un seul canal virtuel pour chaque port d'entrée.

Prenons l'exemple présenté figure 5.4 afin d'expliquer le rôle de l'unité d'arbitrage d'entrée. Le port d'entrée E du routeur a reçu 3 paquets sur 3 canaux virtuels différents nommés respectivement VC1, VC2 et VC3. Ces trois paquets demandent le routage vers leur destination au même moment alors que le routeur ne peut envoyer à cet instant qu'un seul paquet par port d'entrée. Le paquet reçu sur le canal virtuel VC2 est plus prioritaire que les autres paquets. Nous supposons dans cet exemple, que la politique d'arbitrage utilisée est à priorité

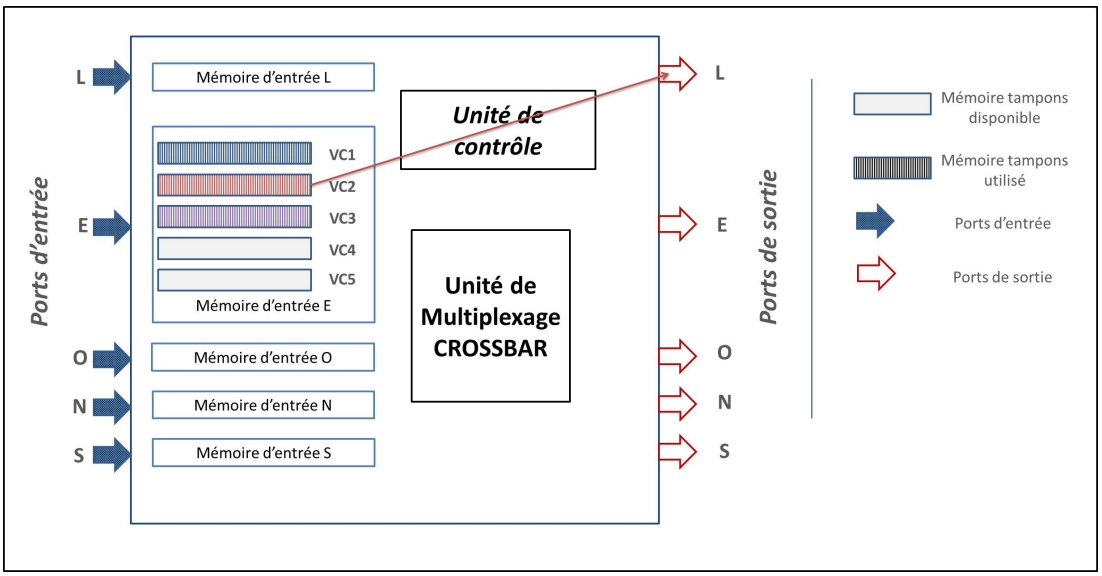


FIGURE 5.4 – Rôle de l'unité d'arbitrage d'entrée

calculée. En conséquence, l'unité d'arbitrage d'entrée sélectionne le canal virtuel VC2.

Le rôle de l'unité d'arbitrage d'entrée est de sélectionner un seul canal parmi ces trois canaux en se basant sur la politique d'arbitrage adoptée.

5.2.5.2 L'unité d'arbitrage de sortie

Plusieurs canaux virtuels de différents ports d'entrée peuvent demander le même port de sortie alors qu'un seul canal virtuel peut être sélectionné. La sélection du canal virtuel est la fonction principale de l'unité d'arbitrage de sortie.

Afin d'expliquer le rôle de l'unité d'arbitrage de sortie, nous prenons l'exemple présenté figure 5.5. 3 paquets (P1, P2 et P3) viennent de 3 ports d'entrée différents (E, O et N). Ces 3 paquets demandent, au même moment, le routage vers un même port de sortie (E). Mais le routeur ne peut router à cet instant qu'un seul paquet par port de sortie. Supposons dans cet exemple que le paquet P1 est plus prioritaire que les autres paquets (P2 et P3). Nous supposons également que la politique d'arbitrage utilisée est à priorité calculée. En conséquence, l'unité d'arbitrage de sortie sélectionne le paquet P1.

Le rôle de l'unité d'arbitrage de sortie est de sélectionner un canal parmi ces trois canaux en appliquant la politique d'arbitrage choisie.

5.2.5.3 Deux étages d'arbitrage

Les unités d'arbitrage d'entrée et de sortie sont basées sur deux étages d'arbitrage.

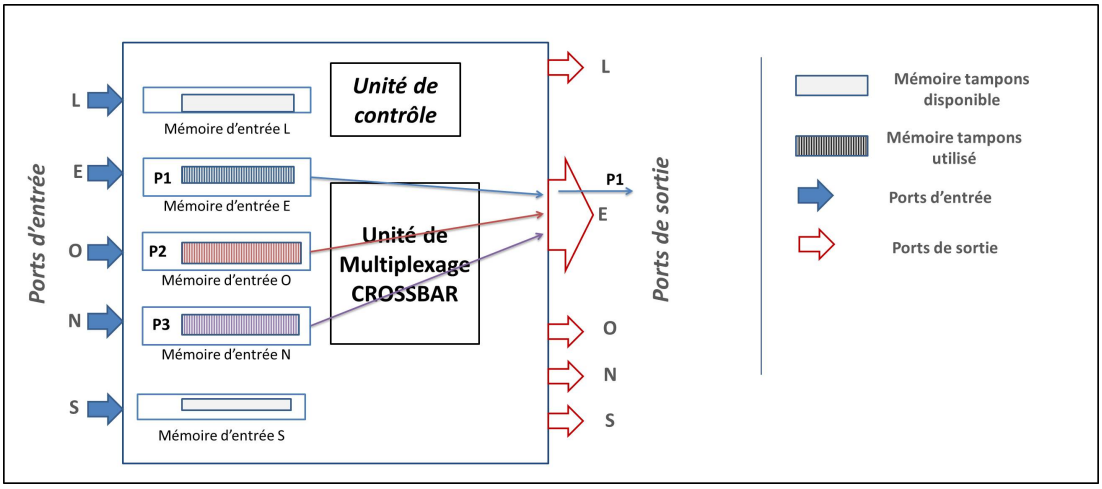


FIGURE 5.5 – Rôle de l'unité d'arbitrage de sortie

La figure 5.2 présente les deux étages d'arbitrage utilisés dans les unités d'arbitrage d'entrée de DAS. Le premier étage est un arbitrage à priorité tournante entre les N premiers canaux virtuels. Le deuxième étage est un arbitrage à priorité calculée entre le canal virtuel sélectionné par le premier étage et le dernier canal virtuel (VC N + 1).

La figure 5.3 présente les deux étages d'arbitrage utilisés dans les unités d'arbitrage de sortie de DAS. Le premier étage est un arbitrage à priorité tournante entre les canaux virtuels high sélectionnés par chaque port d'entrée. Le premier étage présente également un arbitrage à priorité tournante entre les canaux virtuels low sélectionnés par chaque port (les VC N+1 pour chaque port d'entrée) . Le deuxième étage est un arbitrage à priorité calculée entre le canal virtuel high sélectionné par le premier étage et le canal virtuel low sélectionné par le premier étage.

Dans les unités d'arbitrages d'entrée et de sortie, le deuxième étage d'arbitrage permet à DAS d'implanter la préemption au niveau flit. Le canal virtuel sélectionné par le premier étage d'arbitrage préempte le dernier canal virtuel au niveau flit en se basant sur la décision du deuxième étage d'arbitrage.

Dans la partie suivante, nous présentons les différents modes de fonctionnement de DAS et nous expliquons les règles du changement de mode de criticité adoptées par DAS.

5.3 Protocole de changement de mode de criticité

5.3.1 Modes de criticité

Comme nous l'avons vu au début de ce chapitre, le modèle de systèmes à criticité mixte introduit par Vestal et Baruah, et que nous considérons dans cette thèse, est caractérisé par les modes de criticité. Dans le contexte du routeur proposé, nous considérons deux modes de criticité : NORMAL et DEGRADED.

Dans le mode NORMAL, les échéances de tous les flux sont garanties, tandis que dans le mode DEGRADED, les échéances des flux LOW ne sont plus garanties, et ce pour permettre de garantir les échéances des flux HIGH.

Nous attribuons un mode de criticité à chaque port d'entrée/sortie du routeur. Les ports d'entrée/sortie sont par défaut dans le mode de criticité NORMAL. Lorsqu'un flux HIGH est confronté à un conflit avec les flux LOW, le système bascule vers le mode DEGRADED.

Dans la suite, nous expliquons comment et pourquoi le système passe d'un mode de criticité à un autre.

5.3.2 Règles du changement de mode de criticité

En mode NORMAL, un port d'entrée/sortie peut être utilisé par tous les flux (HIGH et LOW) sans interférence. En mode DEGRADED, seuls les flux HIGH peuvent utiliser le port.

La figure 5.6 résume les changements de mode possibles pour un port d'entrée/sortie par un automate. Le mode de criticité pour chaque port dépend essentiellement du résultat de ses unités d'arbitrage. Autrement dit, les deux étages d'arbitrage gèrent le changement de mode pour chaque port d'entrée/sortie.

Le mécanisme du changement de mode est destiné à respecter les quatre propriétés indiquées ci-dessous. Dans le chapitre 6, nous vérifions ces propriétés avec la technique du model-checking en utilisant le langage IF et ses outils [85].

Propriété 1 : En mode NORMAL, les flux HIGH et LOW utilisent le routeur sans interférence.

Autrement dit, dans le mode NORMAL, les flux HIGH et LOW partagent le même port d'entrée/sortie sur des plages temporelles disjointes.

Propriété 2 : En mode DEGRADED, les flux LOW sont suspendus.

En d'autres termes, dans le mode DEGRADED, les flux LOW ne sont pas autorisés à utiliser le port d'entrée/sortie.

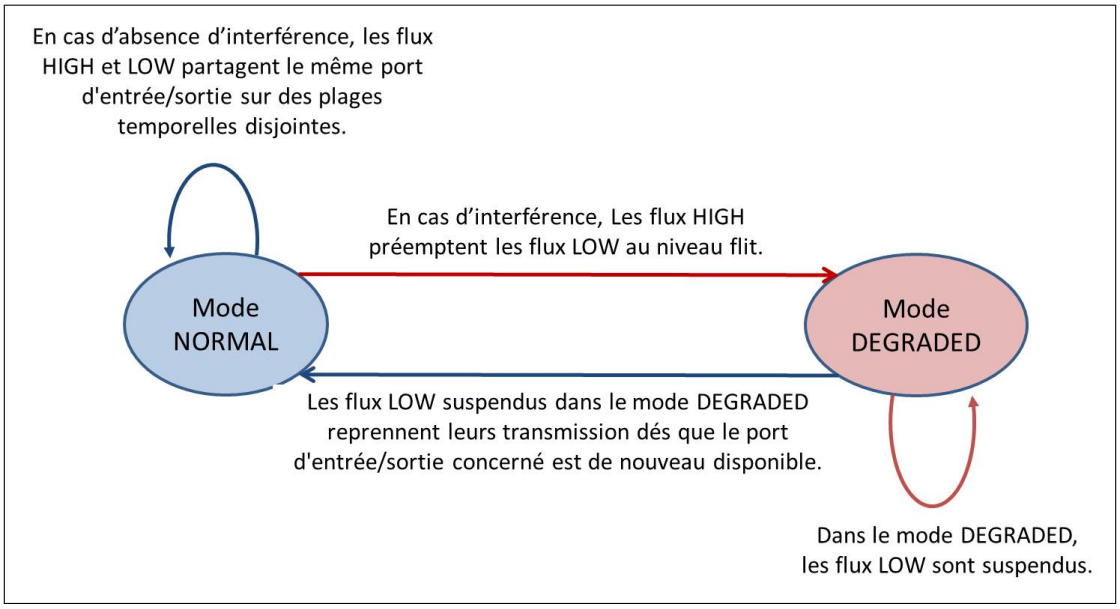


FIGURE 5.6 – Changements de mode de criticité pour chaque port d'E/S

Propriété 3 : Les flux HIGH préemptent les flux LOW au niveau flit.

Lorsque un flux HIGH demande un port d'entrée/sortie déja utilisé par un flux LOW, le flux HIGH préempte le flux LOW au niveau flit.

Propriété 4 : Les flux LOW préemptés reprennent leurs transmissions après la fin de la transmission des flux HIGH.

En d'autres termes, les flux LOW suspendus dans le mode DEGRADED reprennent leurs transmissions dès que le port d'entrée/sortie concerné est de nouveau disponible.

L'adoption de ces modes conduit à une analyse du pire temps de communication précise et conforme aux exigences des systèmes à criticité mixte. Dans la partie suivante, nous présentons l'analyse du pire temps de communication pour les différents modes de criticité en tenant compte des spécifications de DAS.

5.4 Analyse du pire temps de communication

Pour exécuter des applications temps réel sur un réseau sur puce, nous devons non seulement assurer le respect des contraintes temporelles pour les tâches, mais aussi assurer que les communications entre ces tâches respectent leurs échéances. Dans ce contexte, nous proposons une analyse du pire temps de communication pour les flux HIGH et LOW en prenant en compte les spécifications de DAS.

5.4.1 Les flux HIGH

Soit M un vecteur spécifiant les modes de criticité pour tous les ports d'entrée /sortie utilisés par un flux HIGH ρ_i ($M = \{S_0, ..., S_{n-1}\}$). Le pire temps de communication $C_i(M)$ pour ρ_i dépend des ports d'entrée et des ports de sortie concernés. Dans notre analyse, nous supposons que $C_i(M)$ est égal à la somme des pires temps de communication unitaire pour tous les routeurs appartenant à son chemin de transmission :

$$C_i(M) = C_i(S_0, ..., S_{n-1}) = \sum_{j=0}^{n-1} cu_{i_j}(S_j) \tag{5.1}$$

Avec

- $C_i(M)$ représente le pire temps de communication pour un flux ρ_i avec n sauts.
- n représente le nombre de sauts entre la source et la destination.
- S_j représente la combinaison du mode du port d'entrée et du mode du port de sortie pour le $j^{\text{ème}}$ saut pour le flux ρ_i (voir le tableau 5.1).
- $cu_{i_j}(S_j)$ est le pire temps de communication unitaire pour le $j^{\text{ème}}$ saut du flux ρ_i, i.e., la durée maximale de temps de transmission pour un saut donné. Il dépend du mode de criticité S_j du port d'entrée et du port de sortie.

  $cu_{i_j}(S_j)$ est calculé à partir du PATH DELAY (PD), du DIRECT INTERFERENCE DELAY (DID) et du PREEMPTIOIN DELAY (PTD). Nous avons introduit ces facteurs au chapitre 3.

Nous présentons dans le tableau 5.1 la valeur du pire temps de communication unitaire pour les flux HIGH en fonction du mode de criticité. Comme indiqué dans ce tableau, le pire temps de communication comporte le délai de préemption PTD lorsqu'au moins un des deux ports concernés passe en mode DEGRADED.

Rappelons que dans le contexte des systèmes à criticité mixte, l'analyse d'ordonnancement doit être effectuée pour chaque mode. Autrement dit, un système à criticité mixte est ordonnançable si, dans chaque mode, les tâches et les communications de ce mode sont ordonnançables [78].

Dans le mode NORMAL, les flux HIGH et LOW partageant les mêmes ressources sans interférence, nous ne prenons pas en compte le temps de préemption dans l'équation 5.1 pour les flux HIGH.

Dans le mode DEGRADED, des interférences entre les flux HIGH et LOW ont été détectées, mais seuls les flux HIGH doivent respecter leurs échéances dans ce mode. Le temps de préemption est pris en compte dans l'équation 5.1 pour les flux HIGH.

TABLE 5.1 – Analyse du pire temps de communication pour un flux HIGH ρ_i

S_j	**Le mode du port d'entrée**	**Le mode du port de sortie**	**Pire temps de communication unitaire** cu_{i_j}
NORMAL	NORMAL	NORMAL	$PD_{ij} + DID_{ij}$
DEGRADED	DEGRADED	NORMAL	$PD_{ij} + DID_{ij} + PTD_{ij}$
DEGRADED	NORMAL	DEGRADED	$PD_{ij} + DID_{ij} + PTD_{ij}$
DEGRADED	DEGRADED	DEGRADED	$PD_{ij} + DID_{ij} + 2$ x PTD_{ij}

5.4.2 Les flux LOW

Pour les flux LOW, nous utilisons l'analyse proposée par Shi et al. introduite dans [59]. Cette analyse s'applique pour la technique WORMHOLE avec une politique de partage des canaux virtuels.

Dans le but d'analyser le temps des communications, Shi et al considèrent deux types de priorités : NATURAL PRIORITY et SYSTEM PRIORITY.

Pour chaque flux, le calcul de ces deux priorités se fait hors ligne. L'objectif de SYSTEM PRIORITY est d'identifier le canal virtuel demandé. Autrement dit, les flux ayant la même valeur du SYSTEM PRIORITY utilisent le même canal virtuel. L'objectif de NATURAL PRIORITY est de gérer la sélection entre les flux qui partagent le même canal virtuel.

5.4.3 Exemple d'analyse du pire temps de communication pour les flux HIGH

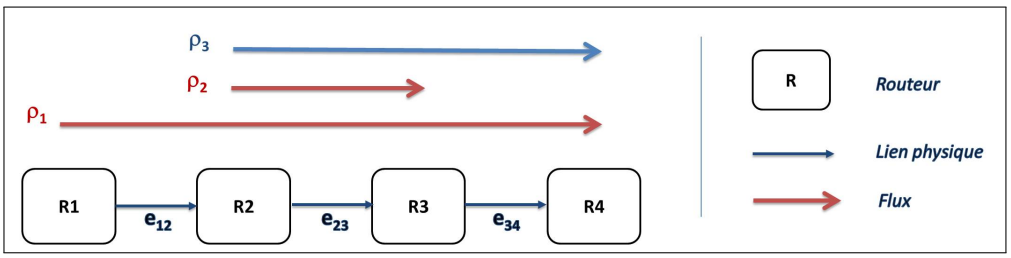


FIGURE 5.7 – Analyse du pire temps de communication

TABLE 5.2 – Modèle de flux

Flux	**Mode de criticité**	**Taille**	**Période (Unité de temps)**	**Échéance (Unité de temps)**
ρ_1	HIGH	2 flits	10	10
ρ_2	HIGH	2 flits	10	10
ρ_3	LOW	8 flits	10	10

Dans le but d'expliquer l'analyse du pire temps de communication, nous prenons l'exemple présenté figure 5.7.

Nous considérons 3 flux (ρ_1, ρ_2 et ρ_3).Ces flux utilisent 4 routeurs (R1, R2, R3 et R4) et 3 liens physiques (e_{12}, e_{23} et e_{34}). Le tableau 5.2 présente le modèle de flux : ρ_1 et ρ_2 sont des flux HIGH alors que ρ_3 est un flux LOW.

Le flux ρ_1 utilise les routeurs (R1, R2, R3, R4). Le flux ρ_2 utilise les routeurs (R2, R3). Le flux ρ_3 utilise les routeurs (R2, R3, R4).

Pour le calcul du PD, nous avons utilisé l'équation 3.1 présentée dans le chapitre 3.

Nous supposons également que $B_{link} = 1$, $PTD = 1$ et $size_{max} = 2$. Rappelons que

- B_{link} représente la bande passante du lien entre deux routeurs voisins.
- PTD représente le temps de préemption.
- $size_{max}$ est la taille maximale du paquet.

Dans la suite nous détaillons le calcul du pire temps de communication pour chaque flux HIGH.

Pire temps de communication de ρ_1

En appliquant l'équation 5.1, nous avons

$$C_1(M) = C_1(S_0, ..., S_2) = \sum_{j=0}^{2} cu_{1_j}(S_j) = cu_{1_0}(S_0) + cu_{1_1}(S_1) + cu_{1_2}(S_2)$$

ou encore :

$$cu_{1_0}(NORMAL) = PD_{1_0} + DID_{1_0} = 2/1 + 0 = 2$$
$$cu_{1_0}(DEGRADED) = PD_{1_0} + DID_{1_0} + PTD_{1_0} = 2/1 + 0 + 0 = 2$$

Notons ici que $DID = 0$ et $PTD = 0$ puisque le lien physique e_{12} n'est utilisé que par ρ_1.

$$cu_{1_1}(NORMAL) = PD_{1_1} + DID_{1_1} = 2/1 + 1 \cdot 2 = 4$$
$$cu_{1_1}(DEGRADED) = PD_{1_1} + DID_{1_1} + PTD_{1_1} = 2/1 + 1 \cdot 2 + 1 = 5$$

Notons ici que $DID \neq 0$ puisque ρ_1 partage le lien e_{23} avec le flux ρ_2. $PTD \neq 0$ puisque ρ_1 partage le lien e_{23} avec le flux ρ_3.

$$cu_{1_2}(NORMAL) = PD_{1_2} + DID_{1_2} = 2/1 + 0 = 2$$
$$cu_{1_2}(DEGRADED) = PD_{1_2} + DID_{1_2} + PTD_{1_2} = 2/1 + 0 + 1 = 3$$

$DID = 0$ puisque ρ_1 est le seul flux HIGH qui demande l'utilisation du lien physique e_{34}.

Pire temps de communication de ρ_2

$$C_2(M) = C_2(S_0) = cu_{2_0}(S_0)$$

$$cu_{2_0}(NORMAL) = PD_{2_0} + DID_{2_0} = 2/1 + 1 \cdot 2 = 4$$
$$cu_{2_0}(DEGRADED) = PD_{2_0} + DID_{2_0} + PTD_{2_0} = 2/1 + 1 \cdot 2 + 1 = 5$$

$DID \neq 0$ puisque ρ_2 partage le lien e_{23} avec le flux ρ_1. $PTD \neq 0$ puisque ρ_2 partage le lien e_{23} avec le flux ρ_3.

5.5 Efficacité de SAF pour les systèmes à criticité mixte

Dans cette partie, nous expliquons le choix de la technique SAF dans ce chapitre. En effet, il y a deux raisons qui motivent ce choix. Premièrement, la technique SAF, en considérant les hypothèses prises dans ce travail, réduit le pire temps de communication par rapport aux autres techniques de commutation. Deuxièmement, la technique SAF avec la configuration proposée dans DAS, réduit le degré de pessimisme de l'analyse du pire temps de communication par rapport à la technique WORMHOLE.

La technique WORMHOLE est la solution la plus populaire pour son faible temps de communication et son faible coût en mémoire par rapport aux autres techniques [12]. Cependant en considérant les hypothèses de cette thèse (les hypothèses 5 et 6), la technique WORMHOLE perd de son efficacité devant SAF.

5.5.1 Réduction du pire temps de communication

La combinaison de SAF avec les canaux virtuels nous permet d'éviter les interférences indirectes. En adoptant la configuration du SAF proposée dans DAS, le pire temps de communication C_{SAF} est composé seulement du PATH DELAY, du DIRECT INTERFERENCE DELAY et du PREEMPTION DELAY :

$$C_{SAF} = PD_{SAF} + DID_{SAF} + PTD \tag{5.2}$$

Cependant, en utilisant la technique WORMHOLE, l'analyse du pire temps de communication comporte toujours des interférences indirectes. L'équation du pire temps de communication en adoptant la technique WORMHOLE est alors :

$$C_{wormhole} = PD_{wormhole} + DID_{wormhole} + IID_{wormhole} + PTD \tag{5.3}$$

La combinaison entre les canaux virtuels et la technique WORMHOLE n'évite pas les interférences indirectes.

Dans la suite, nous comparons les pires temps de communication fournis par ces deux techniques de commutations (les équations 5.2 et 5.3).

- PATH DELAY

  En utilisant la technique WORMHOLE, les flux sont acheminés en pipeline sur plusieurs routeurs. En absence de conflit, cette solution offre le meilleur PD, par rapport à toutes les autres solutions [12]. La différence entre $PD_{wormhole}$ et PD_{SAF} dépend essentiellement de la taille des paquets et de la distance entre la source et la destination.

  Plus la distance entre la source et la destination est importante, plus la différence entre $PD_{wormhole}$ et PD_{SAF} est importante.

  En outre, plus la taille du paquet est importante, plus la différence entre $PD_{wormhole}$ et PD_{SAF} est importante.

  Notons ici que dans tous les cas, $PD_{wormhole}$ est plus petit que PD_{SAF} mais, en considèrant les hypothèses de cette thèse (les hypothèses 5 et 6), la différence entre $PD_{wormhole}$ et PD_{SAF} est réduite.

- PREEMPTION DELAY

  Les deux techniques de commutation SAF et WORMHOLE offrent la même latence pour PREEMPTION DELAY.

- DIRECT INTERFERENCE DELAY

  La valeur de DIRECT INTERFERENCE DELAY est variable et dépend de plusieurs facteurs comme la taille des messages et l'état du réseau. Elle

dépend également de la date d'arrivée d'un message bloqué par rapport à la date d'arrivée du message bloquant.

Notons ici que la borne maximale de DIRECT INTERFERENCE DELAY pour la technique SAF est égale à celle du WORMHOLE ($DID_{SAF} = DID_{Wormhole}$). Par contre, la technique WORMHOLE augmente la probabilité d'avoir des situations d'interférences directes par rapport à la technique SAF.

En conclusion, en considérant les hypothèses de ce travail et un taux d'utilisation du réseau important, et vu l'absence des interférences indirectes en utilisant l'architecture matérielle proposée, la technique WORMHOLE perd une grande partie de son efficacité par rapport à la technique SAF.

5.5.2 Réduction du degré de pessimisme

Nous supposons dans cette thèse que l'objectif d'un système à criticité mixte est de garantir les contraintes temporelles pour les applications critiques tout en minimisant l'impact du partage de ressources sur les applications non critiques [6]. Dans le contexte des architectures NoC, notre objectif est de garantir les contraintes temporelles pour les flux HIGH et minimiser l'impact du partage de ressources sur les flux LOW. Autrement dit, l'objectif de notre architecture NoC est de limiter la réservation des ressources pour les flux HIGH tout en garantissant le respect de ses contraintes temporelles.

La réservation des ressources pour les flux HIGH est basée sur l'analyse du pire temps de communication. Plus l'analyse est pessimiste, plus la réservation conduit à un gaspillage de ressources. Nous devons donc trouver non seulement une solution qui minimise le temps de communication mais aussi une solution qui offre une analyse du pire temps de communication la moins pessimiste. Dans cette partie, nous comparons le degré de pessimisme de la technique WORMHOLE avec SAF tout en considérant l'architecture proposée dans ce chapitre.

Rappelons que le pire temps de communication se produit lorsque le paquet provenant du flux observé se confronte à tous les paquets des autres flux qui partagent les mêmes ressources avec leurs tailles maximales [8]. Il dépend principalement de la configuration du NoC considéré et des types d'interférences entre les flux.

Définition 67. ***(Degré de pessimisme d'une analyse de temps de communication)***

Nous définissons le degré de pessimisme d'une analyse de temps de communication (noté Δ) comme la différence entre le pire temps de communication (C) et le temps de trajet (PD).

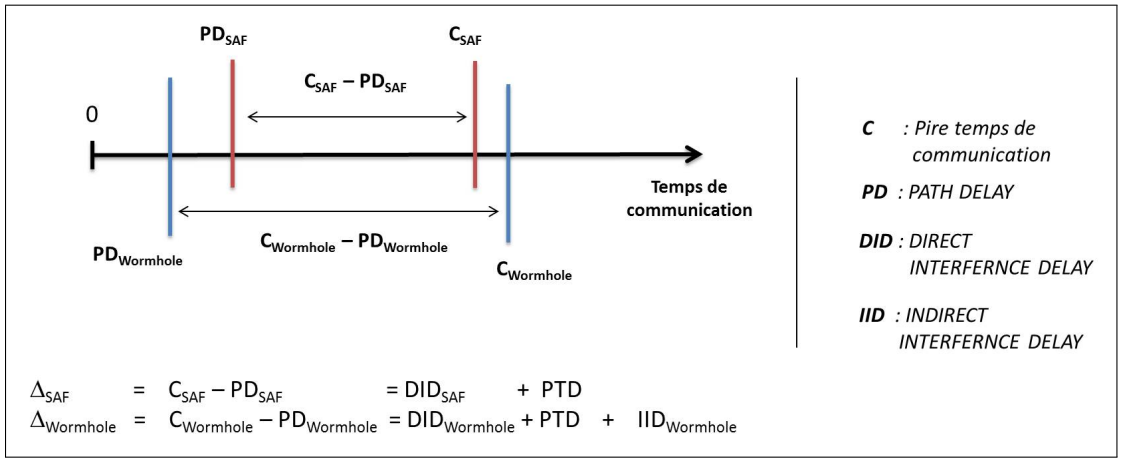


FIGURE 5.8 – Temps de communication pour les techniques SAF et Wormhole

Le degré de pessimisme dépend de plusieurs facteurs comme la taille de message, la distance entre la destination et la source et le taux d'utilisation du réseau.

La figure 5.8 présente une comparaison du degré de pessimisme entre les techniques SAF et WORMHOLE en considérant ces hypothèses. L'instant 0 dans cette figure présente le début d'émission du messages. $PD_{Wormhole}$ (respectivement PD_{SAF}) présente le temps de communication du message en l'absence de conflits et en utilisant la technique WORMHOLE (respectivement en utilisant la technique SAF). $C_{Wormhole}$ (respectivement C_{SAF}) présente le pire temps de communication du message en utilisant la technique WORMHOLE (respectivement en utilisant la technique SAF). Dans cette figure, nous exposons le degrée de pessimisme pour chaque mode de commutation. Pour la technique WORMHOLE, le temps de communication comporte des interférences indirectes, ce qui n'est pas le cas avec pour la technique SAF.

WORMHOLE est la technique la plus retenue dans le domaine des NoC. Plusieurs NoC industriels utilisent la technique WORMHOLE comme Raw, TILE64 et TRIPS [5] car elle réduit le temps de communication et réduit également la taille des tampons. Cependant, le partage des ressources comme les liens physiques et les routeurs peut affaiblir l'efficacité de la technique WORMHOLE. Dans le contexte des systèmes à criticité mixte, la présence des interférences indirectes dans la technique WORMHOLE impose un degré de pessimisme non négligeable.

$$\Delta_{wormhole} = C_{wormhole} - PD_{wormhole} = DID_{wormhole} + IID_{wormhole} + PTD \quad (5.4)$$

L'équation 5.4 présente une expression du degré de pessimisme pour la technique Wormhole.

La solution SAF présente une analyse du pire temps de communication plus

précise qui limite la réservation des ressources aux flux HIGH et qui minimise par la suite l'impact du partage de ressources sur les flux LOW.

$$\Delta_{SAF} = C_{SAF} - PD_{SAF} = DID_{SAF} + PTD \tag{5.5}$$

L'équation 5.5 présente une expression du degré de pessimisme pour la technique SAF avec la configuration proposée.

Pour des messages de petites tailles et un taux d'utilisation de réseau important, la technique SAF fournit la solution la moins pessimiste.

5.6 Conclusion

Dans ce chapitre, nous avons présenté l'architecture et le fonctionnement de DAS. Nous avons également détaillé l'analyse du pire temps de communication pour DAS. Puis, nous avons discuté de nos choix par rapport aux autres configurations possibles.

DAS est conçu pour exécuter des systèmes à criticité mixte sur des architectures NoC. Il supporte deux niveaux de criticité et deux modes de criticité. Dans le but de répondre aux exigences des systèmes à criticité mixte, DAS combine deux techniques de commutation. Selon le niveau de criticité du flux, le routeur choisit la technique WORMHOLE ou SAF. Afin de gérer les conflits entre les flux, il implante également deux étages d'arbitrage. Le premier étage gère les conflits entre les flux HIGH, le deuxième étage entre les flux HIGH et les flux LOW.

Il existe plusieurs métriques qui permettent d'évaluer les performances du routeur NoC tels que le coût en surface et le temps de communication. Dans le chapitre suivant, nous présentons une évaluation de DAS selon ces métriques.

6

Évaluation de DAS sur plusieurs niveaux d'abstraction

Sommaire

6.1 Introduction

Dans le cadre de cette thèse, nous avons proposé DAS, un routeur NoC qui supporte les systèmes à criticité mixte. Dans l'objectif d'étudier la faisabilité du routeur proposé, nous avons examiné DAS selon plusieurs critères comme le coût en surface, le temps de communication et le respect des exigences du système à criticité mixte. Pour cette évaluation, nous avons modélisé DAS sur plusieurs niveaux d'abstraction.

Dans ce chapitre, nous présentons brièvement les niveaux d'abstraction utilisés dans l'évaluation de DAS. Ensuite, une évaluation du coût en surface est présentée. Puis, nous évaluons le temps de communication. Une étude de cas est donnée à titre d'illustration dans cette partie. Enfin, nous vérifions le respect des propriétés du routeur proposé en utilisant le langage IF et ses outils. Finalement, nous terminons ce chapitre par une brève discussion sur les points forts et les points faibles de DAS en fournissant une étude comparative avec les routeurs existants.

6.2 Plusieurs niveaux d'abstraction

Dans ce chapitre, nous avons implanté et évalué DAS sur trois niveaux d'abstraction différents :

- **Niveau circuit/transistor**

  Nous avons modélisé DAS en Verilog HDL [86]. Cette évaluation nous a permis de mesurer et comparer son coût en surface avec WVC-Router et VC-Router.

- **Niveau transaction (ou en anglais *Transaction level modeling (TLM)*)**

  Pour le niveau TLM, nous avons modélisé DAS en SystemC puis nous l'avons intégré dans un simulateur de NoC appelé SHoC [87]. L'évaluation TLM, nous a permis de mesurer le temps de communication des flux avec DAS et de les comparer avec celui de WNoC-Router et VC-Router.

- **Niveau système**

  IF est un langage à base d'automates temporisés communicants. IF et ses outils sont utilisés pour l'élaboration de spécifications formelles [88]. Ils proposent également des approches de validation.

  Nous avons modélisé DAS avec IF puis nous avons validé son fonctionnement. L'évaluation nous a permis de vérifier formellement le respect des propriétés du système à criticité mixte par DAS.

Dans la suite, nous détaillons l'évaluation du coût en surface.

6.3 Évaluation niveau circuit : évaluation du coût en surface

Le coût matériel est l'un des critères primordiaux qui détermine la faisabilité d'une nouvelle architecture. Nous nous intéressons dans cette partie à l'évaluation du coût en surface pour le routeur proposé. Pour cela, nous avons modélisé DAS en Verilog HDL [86], puis nous l'avons synthétisé. Nous avons alors comparé le coût en surface de DAS avec celui des routeurs WVC-Router et VC-Router.

Dans cette partie, nous décrivons les outils utilisés dans cette évaluation. Ensuite, nous présentons les résultats de l'évaluation.

6.3.1 Verilog HDL

Verilog HDL est parmi les langages de description matériel les plus populaires. Verilog HDL a été développé par la société Gateweay Design Automation en 1984. Il est devenu un standard public en 1995 (IEEE 1364-1995) et sa dernière version a été publiée en 2006 (IEEE 1364-2005) [89] [86].

6.3.2 Synthèse et résultats

La synthèse est un processus automatisé qui permet d'aboutir à une représentation structurelle de bas niveau d'un circuit électrique [90]. Les outils de synthèse actuels comme FPGA Express [91], Leonardo Spectrum [91] et Design Compiler [92] acceptent Verilog HDL comme langage d'entrée.

Afin d'évaluer le coût en surface de DAS, nous avons implanté et synthétisé 3 architectures de routeurs : DAS, WVC-router et VC-router. Tous ces routeurs possèdent les mêmes paramètres : 5 ports d'E/S, la même largeur de phits égale à 32 bits et une taille identique des tampons d'entrée à 16 flits. Les architectures des routeurs WVC-routeur et VC-Routeur ont été détaillées dans le chapitre 3.

Dans le but de synthétiser ces trois routeurs, nous avons utilisé le logiciel Design Compiler de Synopsys [92]. Ce dernier permet la synthèse logique grâce aux outils Design Vision (en mode graphique) et DC Shell (en mode ligne). Il se présente comme une référence au niveau de l'utilisation industrielle. Il génère également des rapports décrivant la surface d'implémentation [92]. Dans cette synthèse, nous avons utilisé la technologie 28 nm ST SOI.

Nous présentons dans le tableau 6.1 le coût total en surface pour chaque routeur.

TABLE 6.1 – Les résultats de synthèse avec Synopsys DC/28 nm ST SOI

	WVC-Router	**VC-Router**	**DAS**
Nombre de ports	5	5	5
Nombre de canaux virtuels	1	3	3
Largeur de phits	32 bits	32 bits	32 bits
Taille des tampons par port	16 flits	16 flits	16 flits
Surface totale du puce	16046.31	18369.47	18831.32

Le surcoût en surface pour DAS est d'environ 17% par rapport à WVC-Router. L'augmentation du coût en surface dans ce cas, est expliquée par l'absence des canaux virtuels dans WVC-Router.

Le surcoût en surface pour DAS est d'environ 2.5% par rapport à VC-Router. L'augmentation du coût en surface dans ce cas, est expliquée par l'ajout des deux étages d'arbitrage.

6.4 Évaluation niveau transaction : évaluation du temps de communication

Dans le but d'évaluer les temps de communication produits par DAS, nous avons développé ce dernier en SystemC [90]. Le développement de DAS en SystemC se

fait au niveau TLM. À ce niveau d'abstraction, le routeur est modélisé comme une plateforme composée de plusieurs modules. La communication entre les modules est gérée via un modèle d'interconnexion constitué des canaux de communication.

Nous avons intégré DAS dans un simulateur de NoC appelé SHoC [87]. SHoC est un SystemC-TLM simulateur cycle accurate qui fournit tous les composants nécessaires pour les simulations de multi-coeurs [87]. Il supporte plusieurs générateurs de trafic et de consommateurs. Il nous permet également d'observer le trafic transmis au sein du NoC.

Nous avons développé 3 NoC :

- 2D 4×4 NoC avec le routeur DAS
- 2D 4×4 NoC avec WNoC-Router
- 2D 4×4 NoC avec VC-Router.

Dans cette évaluation, nous n'avons pas utilisé WVC-Router puisque ce dernier ne supporte pas des flux avec des niveaux de criticité différentes.

Nous avons également utilisé deux types de trafic :

- **Trafic uniforme**

  Pour ce type de trafic, les nœuds source et destination de chaque flux sont sélectionnés aléatoirement selon la méthode Uunifast [93].

- **Trafic All-To-One**

  Pour ce type de trafic, le nœud source de chaque flux est sélectionné aléatoirement en utilisant Uunifast [93]. Le nœud destination est fixé par l'utilisateur. Tous les flux partagent le même nœud destination [90].

Dans la suite, nous évaluons les temps de communication produits par DAS pour les flux HIGH. Puis, nous évaluons l'impact du partage de ressources sur les flux LOW. Finalement, une étude de cas est proposée dans le but d'étudier l'impact du temps de communication sur les tâches et le fonctionnement global de l'application.

6.4.1 Temps de communication pour les flux HIGH

Dans cette section, nous évaluons les temps de communication fourni par DAS pour les flux HIGH. Nous comparons DAS avec WNoC-Router et VC-Router en considérant les deux types de trafic : All-To-One et uniforme.

6.4.1.1 Évaluation avec un trafic uniforme

Dans cette évaluation, nous générons un seul flux HIGH. La génération de la source et de la destination de ce flux est aléatoire selon la méthode UuniFast [93]. La taille des messages de ce flux est fixée à 3 flits.

Avec ce flux HIGH, nous générons un ensemble de flux LOW selon un traffic uniforme. Les flux LOW sont générés de façon à ce qu'ils partagent les mêmes liens physiques avec le flux HIGH. La taille des messages de flux LOW est 8 flits.

Le choix des tailles de flux HIGH et LOW est basé sur les hypothèses 4 et 5.

Les périodes et les dates d'émission de chaque flux sont générées aléatoirement selon la méthode UuniFast [93].

Durant cette évaluation, nous augmentons le taux d'utilisation du réseau en ajoutant des flux LOW et en minimisant les périodes des flux LOW existants. Nous avons effectué 100 simulations.

Les figures 6.1 et 6.2 présentent le temps de communication pour le flux HIGH en fonction du taux d'utilisation du réseau. Les distances entre la source et la destination pour le flux HIGH sont respectivement de 3 et 4 liens physiques.

Nous présentons dans ces figures la valeur moyenne et la borne maximum des temps de communication pour chaque routeur. VC-Router (MAX) représente la courbe des valeurs maximales pour le temps de communication en utilisant VC-Router. VC-Router (MOY) représente la courbe des valeurs moyennes pour le temps de communication en utilisant VC-Router. DAS (MAX) représente la courbe des valeurs maximales pour le temps de communication en utilisant DAS. La courbe DAS (MOY) représente la courbe des valeurs moyennes pour le temps de communication en utilisant DAS.

Ces figures montrent que DAS est capable de réduire significativement le temps de communication des flux HIGH par rapport au routeur VC-Router. Pour 15% du taux d'utilisation du réseau et pour une distance de 3 liens physiques, DAS réduit le temps de communication avec un pourcentage de 80% par rapport à VC-Router.

6.4.1.2 Évaluation avec un trafic All-To-One

Nous commençons cette évaluation par la génération d'un seul flux HIGH. Nous appelons ce flux FIRSTHIGH. Ce flux est généré aléatoirement selon Uunifast [93].

Pour chaque simulation, nous générons deux ensembles de flux :

- Un ensemble de 10 flux HIGH qui partagent les mêmes liens physiques que le flux FIRSTHIGH dans le but de créer des contentions entre les flux HIGH.

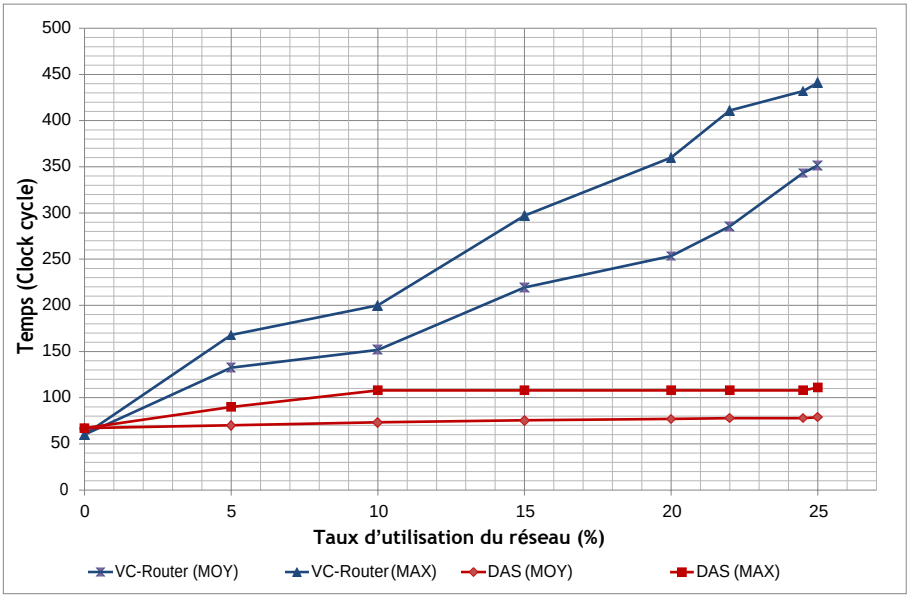


FIGURE 6.1 – Temps de communication pour un flux HIGH - 3 liens physiques

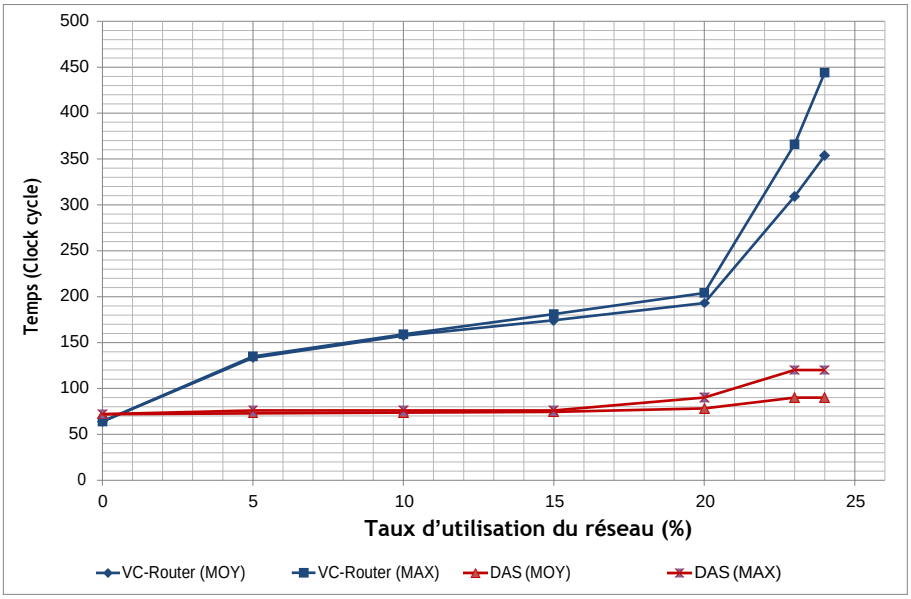


FIGURE 6.2 – Temps de communication pour un flux HIGH - 4 liens physiques

- Un ensemble de 40 flux LOW qui partagent également les mêmes liens physiques dans le but de créer des contentions entre les flux HIGH et LOW. Les flux LOW sont générés selon le trafic All-To-One.

Nous faisons varier la taille des flux HIGH de 2 à 6 flits par pas de 2 flits.

Notons de plus que les périodes et les dates d'émissions sont générées aléatoirement selon Uunifast [93].

La figure 6.3 présente le temps de communication du flux FIRSTHIGH en fonction du taux d'utilisation du réseau et pour des tailles différentes de flux. Elle présente les résultats de DAS et de WNoC-Router pour des distances de 3 et 4 liens physiques. Dans cette figure, nous présentons la valeur moyenne des temps de communication.

Les courbes de la figure 6.3(a) présentent les résultats pour un flux de 2 flits. Elles montrent que DAS est capable de réduire significativement le temps de communication de FIRSTHIGH par rapport à WNoC-Router. Pour 14% du taux d'utilisation du réseau, DAS réduit de 64% le temps de communication par rapport à WNoC-Router pour une distance de 4 liens physiques.

La combinaison de SAF avec les canaux virtuels empêche les interférences indirectes. L'absence des interférences indirectes dans le réseau de DAS peut expliquer ces résultats.

Les courbes de la figure 6.3(b) présentent les résultats pour un flux de 4 flits. Les résultats montrent que les temps de communication fourni par DAS et WNoC-Router sont similaires avec un léger avantage pour DAS.

Les courbes de la figure 6.3(c) présentent les résultats pour un flux de 6 flits. Dans cette configuration, les résultats montrent que DAS perd son efficacité contre WNoC-Router. Avec 6 flits, la technique SAF perd son efficacité devant la technique Wormhole.

Pour conclure, en utilisant DAS, les flux HIGH sont moins affectés par le partage de ressources avec les flux LOW par rapport à VC-Router. Ainsi, la comparaison entre DAS et WNoC-Router a montré que DAS est plus efficace par rapport à WNoC lorsqu'on considère des flux de petite taille. Enfin, DAS perd de son efficacité devant WNoC-Router pour des flux plus grands que 4 flits (Hypothèse 5).

6.4.2 Temps de communication pour les flux LOW

Dans cette section, nous évaluons le temps de communication fourni par DAS pour les flux LOW. Nous comparons DAS avec WNoC-Router et VC-Router en considérant les deux types de trafic : All-To-One et Uniforme.

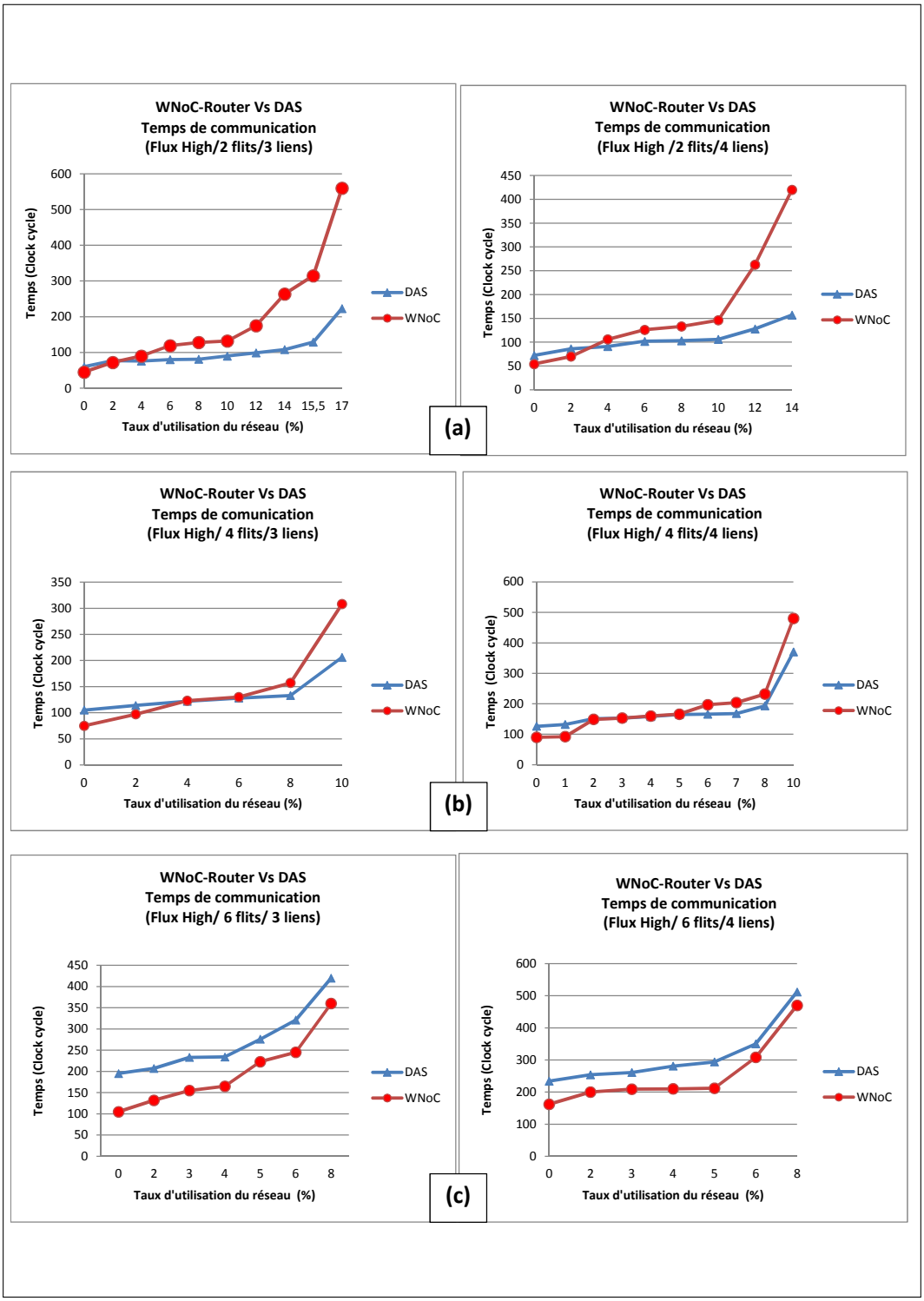


FIGURE 6.3 – Temps de communication pour FirstHigh
(a) Taille = 2 flits (b) Taille = 4 flits (c) Taille = 6 flits

6.4.2.1 Évaluation avec un trafic uniforme

Pour cette évaluation, nous avons généré un seul flux LOW. La génération de la source, de la destination, de la date d'émission et de la période est aléatoire selon Uunifast [93]. Puis, nous avons généré un ensemble de flux HIGH qui partagent les mêmes liens physiques avec le flux LOW. Dans le but d'augmenter le taux d'utilisation du réseau, le nombre des flux HIGH est augmenté et leurs périodes sont réduites. Nous avons effectué 100 simulations sous le simulateur SHoC. Pour cette évaluation, la taille du flux HIGH est 2 flits tandis que la taille du flux LOW est 8 flits.

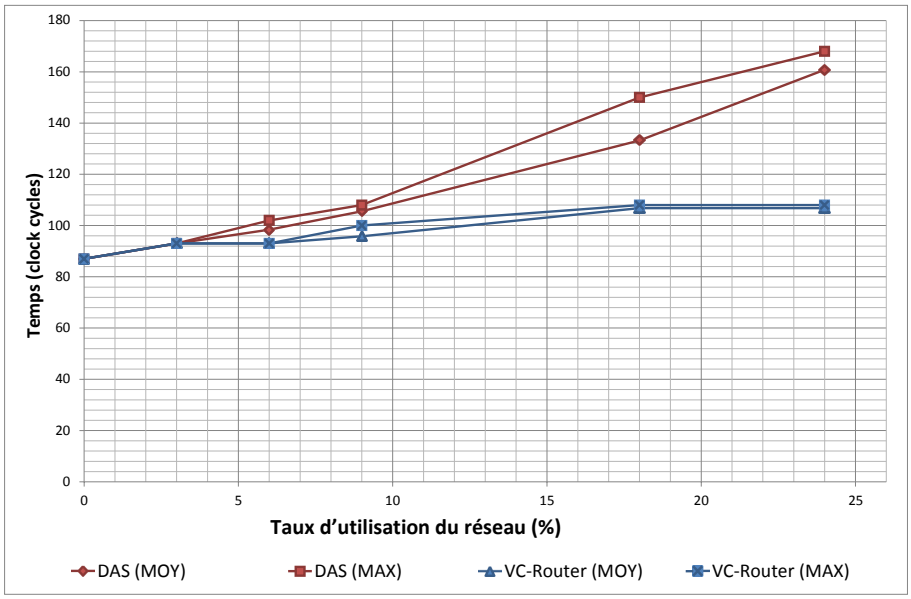


FIGURE 6.4 – Temps de communication pour les flux LOW - trafic uniforme

La figure 6.4 présente les valeurs moyennes et les bornes maximales de temps de communication du flux LOW en fonction du taux d'utilisation du réseau. VC-Router (MAX) représente la courbe des valeurs maximales pour le temps de communication en utilisant VC-Router. VC-Router (MOY) représente la courbe des valeurs moyennes pour le temps de communication en utilisant VC-Router. DAS (MAX) représente la courbe des valeurs maximales pour le temps de communication en utilisant DAS. DAS (MOY) représente la courbe des valeurs moyennes pour le temps de communication en utilisant DAS.

Les résultats montrent que DAS introduit des retards supplémentaires dans le temps de communication pour le flux LOW par rapport à VC-Router. Pour 15% du

taux d'utilisation du réseau, DAS augmente de 25% le temps de communication pour le flux LOW par rapport à VC-Router. Ceci est expliqué par l'effet de la préemption au niveau flit à la défaveur du flux LOW.

6.4.2.2 Évaluation avec un trafic All-To-One

Dans cette évaluation, nous avons généré aléatoirement un seul flux LOW selon Uunifast [93]. Ensuite, nous avons généré un ensemble de flux HIGH qui partagent les mêmes liens physiques. Cette génération est réalisée selon le trafic All-To-One. Nous avons effectué 100 simulations avec le simulateur SHoC. À chaque simulation, nous avons varié le taux d'utilisation du réseau en augmentant le nombre des flux HIGH et en réduisant les périodes des flux HIGH. Pour cette évaluation, la taille des messages du flux HIGH est 2 flits, tandis que la taille des messages du flux LOW est 8 flits. Notons ici que les dates d'émission des flux HIGH sont également générées aléatoirement en utilisant Uunifast [93].

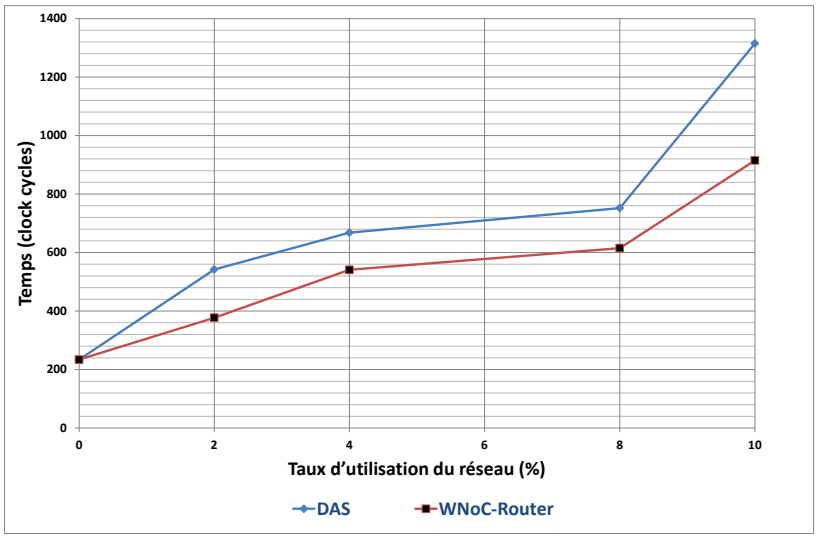


FIGURE 6.5 – Temps de communication pour les flux LOW - trafic All-To-One

La figure 6.5 présente la valeur moyenne du temps de communication pour le flux LOW en fonction du taux d'utilisation du réseau. Les résultats montrent que DAS accroit le temps de communication pour les flux LOW en comparaison avec WNoC-Router. Pour 8% du taux d'utilisation du réseau, DAS retarde de 21% le flux LOW par rapport à WNoC-Router pour une distance de 3 liens physiques.

6.4.3 Étude de cas basée sur l'application Rosace

Précédemment, nous avons évalué le temps de communication à travers DAS et nous avons étudié l'impact des conflits sur les échéances des messages. Dans cette partie, nous présentons une étude de cas dont l'objectif est d'examiner l'impact du temps de communication des flux dans le NoC utilisant DAS sur les tâches.

Nous avons utilisé un benchmark du domaine des systèmes temps réel dur appelé Rosace. Rosace est un contrôleur de commande de vol longitudinal [81]. Nous avons utilisé dans cette simulation deux NoC : un NoC avec DAS et un NoC avec WNoC-Router. Les deux réseaux ont la même dimension et topologie (grille 2D 4*4). Les deux routeurs utilisent le même algorithme de routage (le routage XY) et ils implantent 5 canaux virtuels par port.

Dans la suite, nous présentons l'application Rosace. Ensuite, nous décrivons l'allocation des tâches considérées, le modèle de tâches et de flux. Finalement, nous évaluons l'impact du temps de communication sur les deux NoC.

6.4.3.1 Rosace : un contrôleur de commande de vol

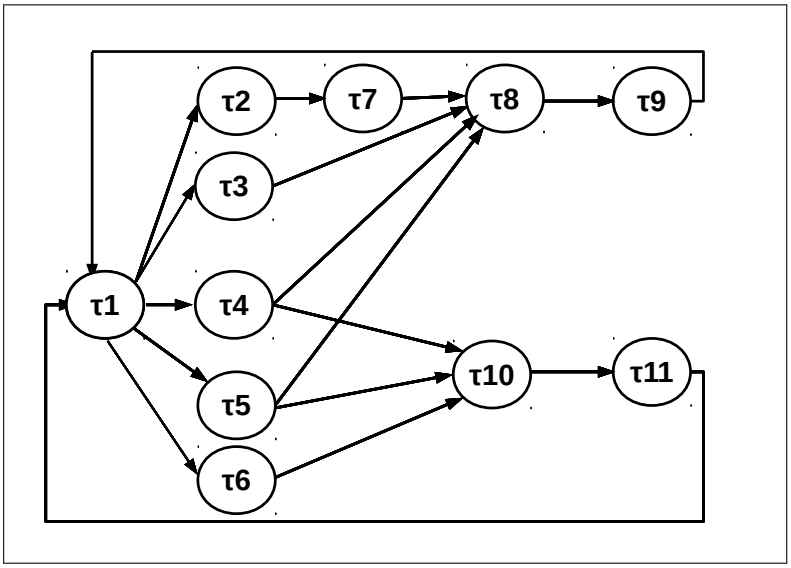


FIGURE 6.6 – Rosace : graphe de tâches

Rosace est un contrôleur de vol en phase de route pour les avions civils de moyenne portée [81]. L'objectif de cette application est de suivre, tout au long du vol, l'altitude, la vitesse verticale et les commandes de vitesse (notées respectivement h, Vz et Va).

Rosace est une application temps réel dur multi-périodique. La figure 6.7 présente le graphe de tâches pour Rosace. Cette application est composée de deux boucles

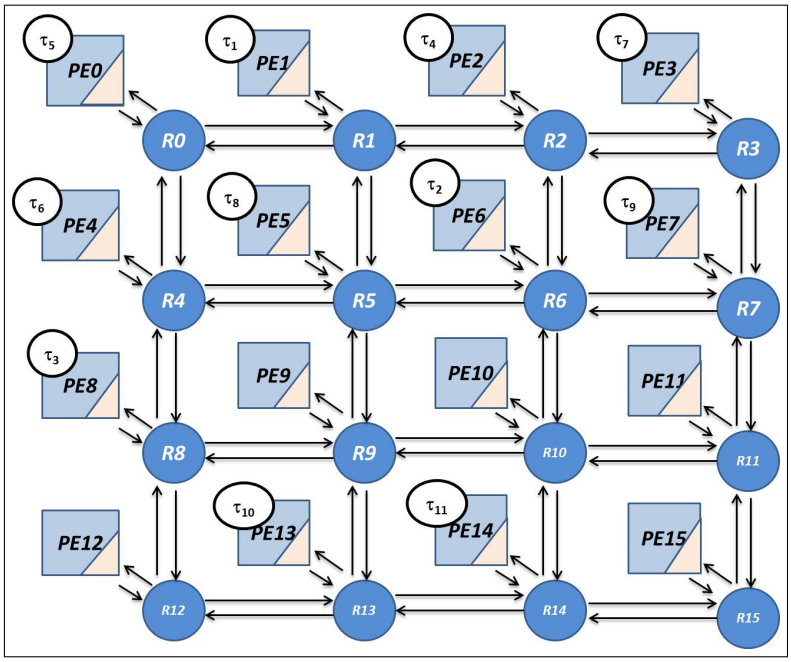


FIGURE 6.7 – Rosace : Allocation des tâches

de contrôle :

- *Boucle de contrôle d'altitude*

  Elle contient les tâches τ_1, τ_2, τ_3, τ_4, τ_5, τ_7, τ_8 et τ_9. L'objectif de cette boucle est de suivre la vitesse verticale Vz.

- *Boucle de contrôle de vitesse*

  Elle contient les tâches τ_1, τ_4, τ_5, τ_6, τ_{10} et τ_{11}. L'objectif de cette boucle est de suivre la vitesse Va.

Dans la suite, nous détaillons les paramètres et l'allocation des tâches utilisés dans cette étude de cas.

6.4.3.2 Modèle de tâche

Comme vu au chapitre 2, chaque tâche périodique τ_i est caractérisée par différents paramètres temporels (O_i, T_i, C_i, D_i, PE_i).

Rappelons que :

- T_i est la période de la tâche τ_i.
- C_i est la capacité de la tâche τ_i.

- D_i est l'échéance de la tâche τ_i.
- PE_i identifie l'unité de calcul qui exécute la tâche τ_i. Ce paramètre nous permet d'introduire l'allocation des tâches.

TABLE 6.2 – Rosace : modèle de tâche

Tâche	T_i (μs)	C_i (μs)	D_i (μs)	PE_i
τ_1	5000	200	5000	PE_1
τ_2	10000	100	10000	PE_6
τ_3	10000	100	10000	PE_8
τ_4	10000	100	10000	PE_2
τ_5	10000	100	10000	PE_0
τ_6	10000	100	10000	PE_4
τ_7	20000	100	20000	PE_3
τ_8	20000	100	20000	PE_5
τ_9	20000	100	20000	PE_7
τ_{10}	5000	100	5000	PE_{13}
τ_{11}	5000	100	5000	PE_{14}

Le tableau 6.2 présente l'allocation des tâches et les paramètres temporels, considérés dans ce travail.

6.4.3.3 Évaluation de l'impact du temps de communication sur les tâches

Contrairement aux évaluations précédentes, l'objectif de cette évaluation est de vérifier l'impact du temps de communication sur les tâches et l'intégrité du système.

Nous avons calculé, en utilisant DTFM, le modèle de flux à partir du modèle de tâches, du modèle de NoC et de l'allocation des tâches proposées précédemment.

DTFM est une méthode qui nous permet de calculer le modèle de flux transmis dans un NoC à partir du modèle du NoC, du modèle de tâches et de l'allocation des tâches. Nous détaillons DTFM dans le chapitre 7.

Une fois que le modèle de flux de Rosace est calculé, nous générons ces communications pour DAS et WNoC-Router.

À coté des flux HIGH de Rosace, nous avons généré un ensemble de 50 flux LOW selon le trafic All-To-One. Nous avons utilisé Uunifast [93] pour la génération des dates d'émission et des périodes pour les flux LOW. Nous avons effectué 100 simulations sous SHoC en variant à chaque simulation le taux d'utilisation du réseau. Notons ici que la taille des messages des flux HIGH est de 2 flits tandis que la taille des messages des flux LOW est de 8 flits.

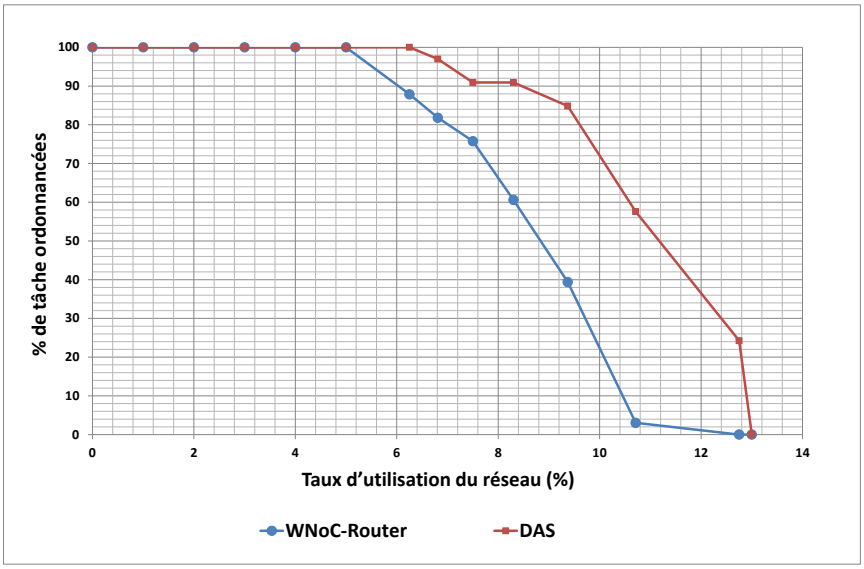


FIGURE 6.8 – Impact du temps de communication des flux sur l'ordonnancement du système

La figure 6.8 présente le pourcentage des tâches ordonnançables en fonction du taux d'utilisation du réseau. Les résultats montrent que DAS est plus efficace que WNoC-Router dans le contexte des systèmes à criticité mixte. Autrement dit, DAS nous permet d'assurer les contraintes temporelles pour les tâches temps réel dur contrairement à WNoC-Router.

En utilisant DAS, Rosace peut respecter ses contraintes temporelles jusqu'à un taux d'utilisation du réseau de 7% tandis qu'avec WNoC-Router nous ne pouvons pas dépasser 5,7%.

Nous avons évalué DAS sur deux niveaux d'abstraction différents. Dans la suite, nous allons évaluer DAS au niveau système.

6.5 Évaluation niveau système : Validation formelle de DAS

DAS doit assurer les propriétés introduites dans le chapitre 5. Ces propriétés sont coûteuses à valider sur un prototype réel de DAS. Dans ce chapitre, nous remplaçons les expérimentations sur un prototype réel par la validation sur un modèle formel [90].

Nous avons choisi le model-checking dans le but de valider les politiques d'arbitrage et le mécanisme de changement de mode implantés dans DAS.

Nous avons choisi le langage IF (Intermediate Format) [88] et ses outils pour modéliser et valider DAS.

Dans cette partie, nous présentons le langage IF. Ensuite, nous détaillons la modélisation de DAS avec le langage IF. Finalement, nous décrivons l'approche de validation de DAS avec IF et ses outils.

6.5.1 Le langage IF

IF est un langage à base d'automates temporisés communicants. Il gère les données, le temps et le parallélisme [88].

Dans cette section, nous présentons brièvement les concepts de IF. Ensuite, nous expliquons l'approche de validation proposée par le langage IF et ses outils.

6.5.1.1 Les concepts de IF

La structure globale d'une spécification IF est composée de systèmes et de processus. Le comportement d'une spécification IF est décrit en termes d'états et de transitions.

Définition 68. *(**Système**)*

Un système est composé d'un ensemble d'instances de processus actifs. Ces processus s'exécutent en parallèle [88].

Définition 69. *(**Processus**)*

Un processus est un automate temporisé étendu. La création et la destruction d'un processus peuvent être faites dynamiquement pendant l'exécution du système [88].

La communication entre ces processus se fait de manière asynchrone par échange de signaux via des routes de communication (signalroute) ou par adressage direct [88].

Définition 70. *(**Route de communication**)*

La communication entre les processus ou entre les processus et l'environnement est assurée par les routes de communication (SignalRoute) [88].

Chaque processus est caractérisé par un ensemble d'états. Un processus change d'état suite à des évènements externes ou internes. Un évènement interne peut être la satisfaction d'une garde, tandis qu'un évènement externe peut être la réception d'un signal.

Définition 71. ***(Transition)***

Le passage d'un état à un autre pour un processus est appelé transition. Le franchissement d'une transition peut être conditionné soit par une garde temporelle, soit par une condition portant sur les variables ou soit par la réception des signaux [88].

Chaque transition se termine par une action NEXTSTATE spécifiant son état destination ou par une action STOP détruisant l'instance courante du processus.

6.5.1.2 L'approche de validation

La boîte à outils du langage IF, plus précisément l'extension IFx, fournit des observateurs IF qui permettent la vérification des propriétés d'un système.

Définition 72. ***(Observateur IF)***

Les observateurs IF sont des processus reliés au reste du système par des interactions synchrones. Ils s'exécutent en parallèle avec le système [88].

Les observateurs ont toujours la priorité la plus élevé pendant l'exploration du système. Ces processus peuvent observer des évènements, des états du système (incluant les variables et les horloges), l'arrêt de la génération d'états non pertinents et la coupure de chemins d'exécution.

Les propriétés à vérifier sont décrites à l'aide d'une syntaxe spécifique aux observateurs IF. Les processus observateurs et le système observé forment le système composé. Lors de la validation d'une propriété, le simulateur effectue une exploration exhaustive de l'espace d'états de ce système composé.

Les états d'un processus observateur IF peuvent être de trois types : état ordinaire (ou en anglais ORDINARY STATE), état d'échec (ou en anglais ERROR STATE) et état de succès (ou en anglais SUCCESS STATE). Une propriété est alors satisfaite si et seulement si l'état d'échec du processus observateur est non atteignable durant l'exploration du système composé.

Dans la suite, nous présentons la modélisation de DAS avec le langage IF.

6.5.2 Modélisation de DAS

La modélisation de DAS en IF est composée de plusieurs processus que nous allons décrire par la suite :

- Le processus principal (figure 6.9)
- Plusieurs instances du processus fils (figure 6.10)

- Des instances d'arbitre d'entrée A
- Des instances d'arbitre d'entrée B (figure 6.11)
- Le switch
- Des instances d'arbitre de sortie A
- Des instances d'arbitre de sortie B

La figure 6.9 présente l'architecture globale du modèle de DAS avec le langage IF. Nous considérons les routeurs voisins et les unités de calcul locales comme environnement du système. DAS_input_message représente les messages envoyés par l'environnement vers DAS. DAS_output_message représente les messages envoyés par DAS vers l'environnement.

Divers signaux sont échangés entre les différents processus du système :

- DAS_Request représente la demande du processus fils aux autres processus pour l'envoi d'un message. Il existe plusieurs signaux de type DAS_Request tels que : DAS_Request_Input_Arb_A, DAS_Request_Input_Arb_B, DAS_Request_Out_Arb_A, DAS_Request_Out_Arb_B et DAS_Request _Switch.
- Feedback représente la notification du processus fils aux autres processus de l'état actuel du message.
- done représente la notification du processus fils aux autres processus de la fin de l'envoie du message.

Dans la suite, nous détaillons le comportement de chacune de ces entités.

6.5.2.1 Processus principal

Le rôle du processus principal consiste à calculer le port de sortie de destination du message. Quand le processus principal reçoit un message DAS_input_message envoyé par l'environnement, il crée une instance de processus fils. Dans cette instance, le paramètre input-channel-id (respectivement output-channel-id) désigne le port d'entrée (respectivement le port de sortie) utilisé dans le routeur par le message considéré.

Pour les flux HIGH, le processus fils modélise un canal virtuel. Pour les flux LOW, le processus fils modélise seulement un flit. Puisque DAS peut traiter plusieurs flux simultanément, nous pouvons avoir simultanément plusieurs instances de processus fils.

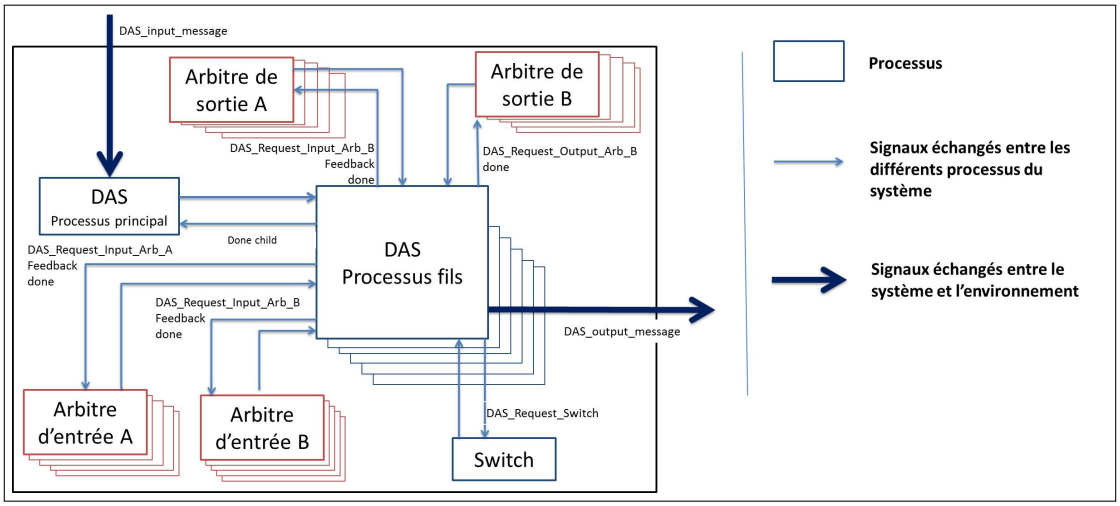


FIGURE 6.9 – L'architecture globale du modèle de DAS avec le langage IF

6.5.2.2 Processus fils

Le processus fils décrit les états possibles d'un message transmis par DAS. La figure 6.10 expose le fonctionnement et les différents états possibles pour un processus fils.

Le processus fils possède 10 états. Il utilise 5 variables booléennes liées aux réponses du switch et des arbitres suite aux demandes du processus fils. InArbiterA_Response est un exemple de variable booléenne.

L'automate commence à l'état IDLE. À partir de l'état IDLE, les processus fils HIGH qui utilisent la technique SAF passent à l'état For-High, tandis que les processus fils LOW passent à l'état For-Low. Une fois qu'un message a été stocké dans le routeur, le processus fils demande à l'arbitre d'entrée A et/ou à l'arbitre de sortie B l'envoi du message en passant vers les états Ask-Input-Arbiter-A et Ask-Input-Arbiter-B en fonction de sa criticité.

Dans l'état Ask-Input-Arbiter-B, le processus fils attend la réponse de l'arbitre d'entrée B, ensuite, il avance vers l'état Ask-Switch où le switch achemine le message vers le port de sortie.

Une fois que le port de sortie a été déterminé, le processus fils demande à l'arbitre de sortie A correspondant et/ou à l'arbitre de sortie B l'envoi du message en passant par les états Ask-Output-Arbiter-A et Ask-Output-Arbiter-B. Nous notons que, dans ces deux états, les processus fils envoient des signaux feedback aux arbitres d'entrée afin de les informer sur l'état du message.

Finalement, dans les états Send et Exit, le message est envoyé à l'environnement et le processus fils est terminé.

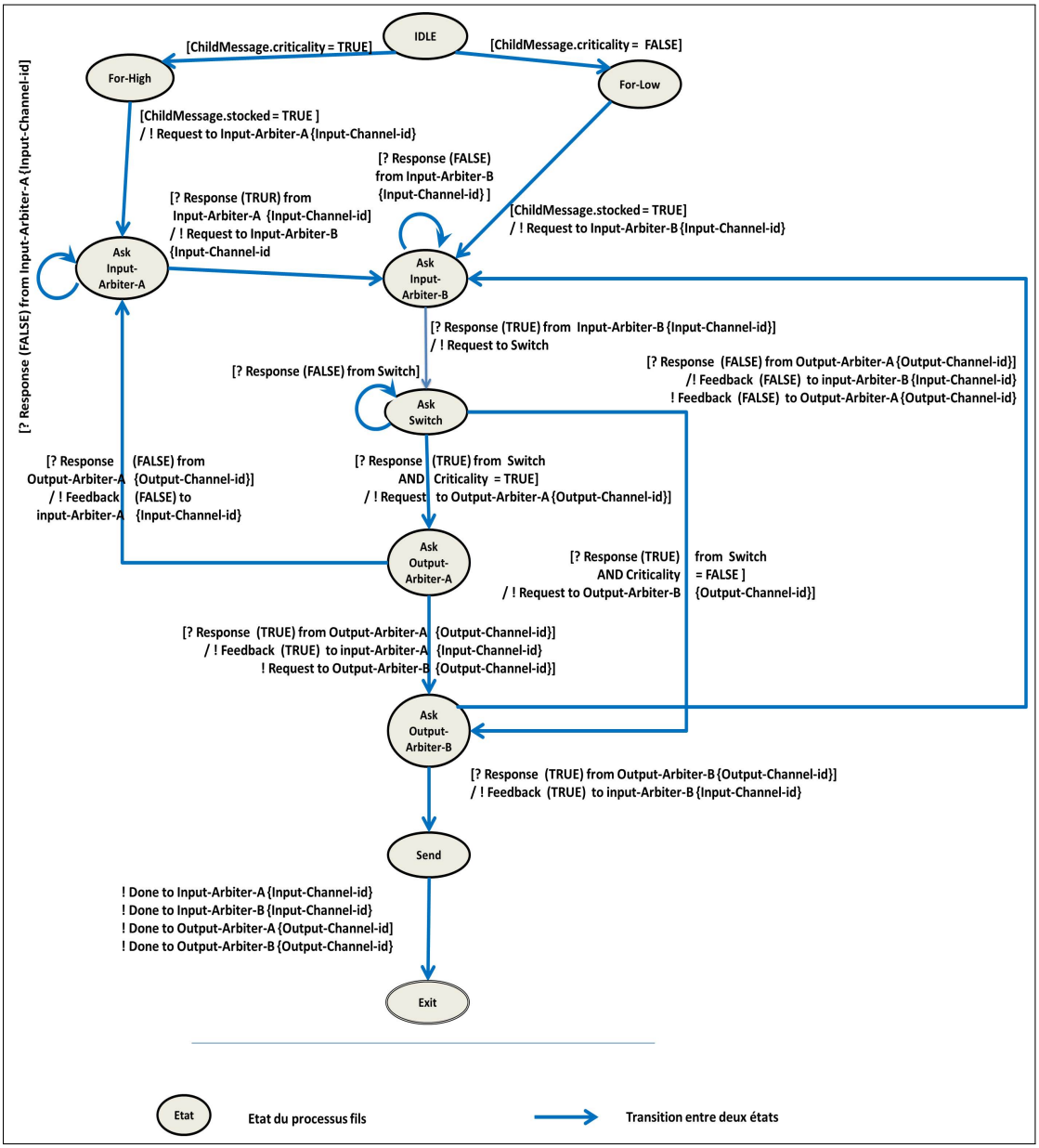


FIGURE 6.10 – Processus fils - Machine d'état

6.5.2.3 Switch

Le rôle du switch consiste à router les messages reçus par les ports d'entrée vers les ports de sortie. Les messages HIGH sont routés vers l'arbitre de sortie A du port de sortie correspondant. Les messages LOW sont routés directement vers l'arbitre de sortie B du port de sortie correspondant.

6.5.2.4 Unité d'arbitrage d'entrée

Plusieurs processus fils ayant la même input-channel-id peuvent demander au même moment l'avancement vers les ports de sortie alors que chaque input-channel-id ne peut accepter qu'un seul processus fils à chaque cycle. Le rôle principal d'une unité d'arbitrage en entrée est de sélectionner un seul processus fils pour chaque input-channel-id.

Rappelons ici que l'input-channel-id (respectivement output-channel-id) représente le port d'entrée (respectivement le port de sortie) utilisé dans le router par le message considéré.

L'unité d'arbitrage en entrée est composée de deux étages d'arbitrage : l'arbitre d'entrée A et l'arbitre d'entrée B.

L'arbitre d'entrée A est un arbitrage à priorité tournante entre tous les processus fils HIGH. L'arbitre d'entrée B est un arbitrage à priorité calculée entre les processus fils HIGH et les processus fils LOW. Nous notons ici que les processus fils HIGH sont plus prioritaires que les processus fils LOW.

La figure 6.11 présente l'automate de l'arbitre d'entrée B. Il commence à l'état IDLE. Si le processus fils est HIGH, il passe à l'état critique sinon il passe à l'état non critique.

Dès qu'il reçoit un signal feedback (true) de la part du processus fils correspondant, il passe à l'état occupé. Une fois que le message est transmis, l'arbitre d'entrée B retourne à l'état IDLE.

6.5.2.5 Unité d'arbitrage de sortie

Le rôle principal de l'unité d'arbitrage de sortie est de sélectionner un seul processus fils pour chaque output-channel-id. L'unité d'arbitrage de sortie est composée de deux étages d'arbitrage : l'arbitre de sortie A et l'arbitre de sortie B.

L'arbitre de sortie A est un arbitrage à priorité tournante entre tous les processus fils HIGH venant de différents ports d'entrée et qui demandent le même port de sortie.

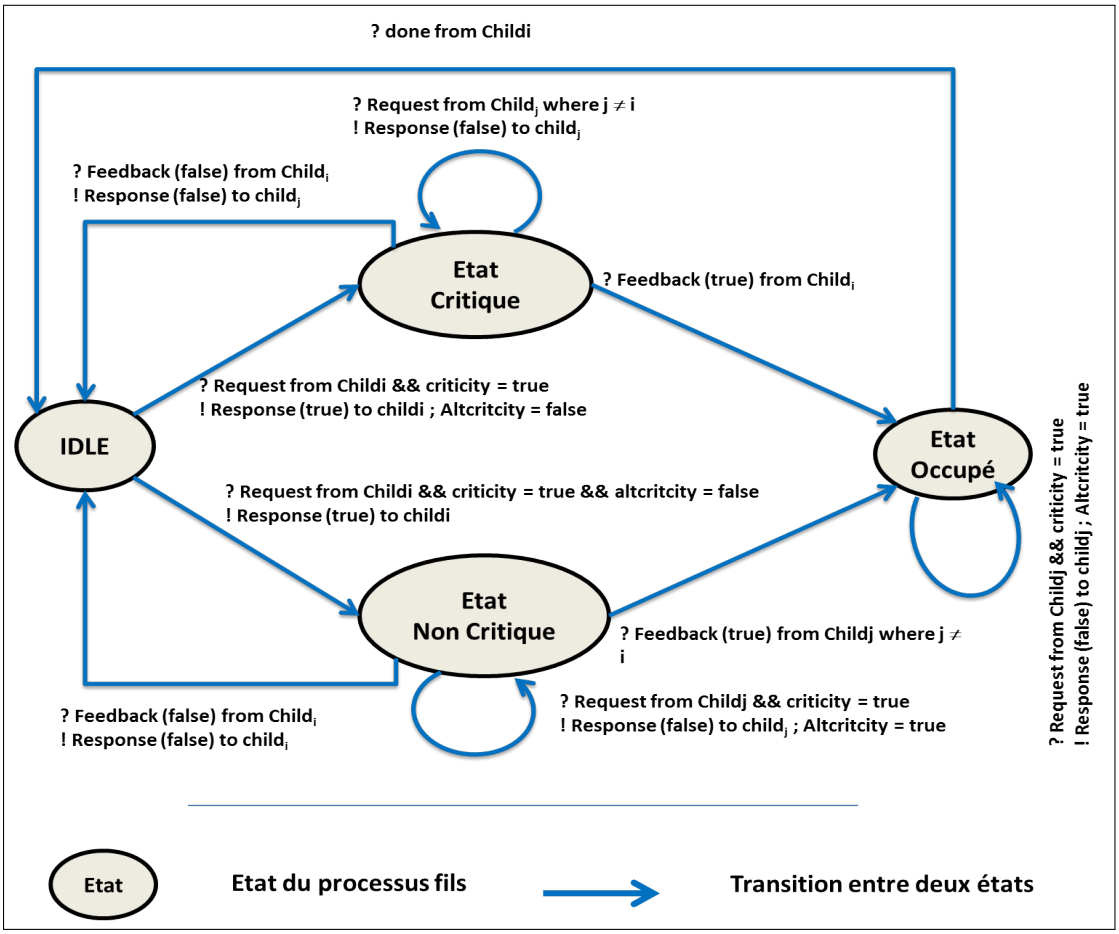


FIGURE 6.11 – Arbitre d'entrée B - Machine d'état

L'arbitre de sortie B est un arbitrage à priorité calculée. Il gère les conflits entre les processus fils HIGH et les processus fils LOW qui demandent le même port de sortie.

6.5.3 Validation

Dans la suite, nous présentons l'approche de validation utilisée. Premièrement, nous utilisons l'outil IFx pour effectuer des simulations interactives. Deuxièmement, nous effectuons une validation formelle exhaustive pour toutes les propriétés de DAS en utilisant les observateurs IF.

Nous rappelons dans la suite ces propriétés :

Propriété 1 : En mode NORMAL, les flux HIGH et LOW utilisent le routeur sans interférence.

Propriété 2 : En mode DEGRADED, les flux LOW sont suspendus.

Propriété 3 : Les flux HIGH préemptent les flux LOW au niveau flit.

Propriété 4 : Les flux LOW préemptés reprennent leur transmissions après la fin de la transmission des flux HIGH.

6.5.3.1 Validation par simulations

TABLE 6.3 – Validation de DAS par simulations : scénarios et résultats

Catégorie	**Fils**	**Criticité**	**Input and output Channel Id**	**Itérations**	**Résultats**
1- Routage simple avec HIGH	1	HIGH	1, 2, 3 and 4	> 5	**Validé**
2- Routage simple avec LOW	1	LOW	1, 2, 3 and 4	> 5	**Validé**
3- Arbitrage d'entrée entre les communications HIGH	> 3	HIGH	Mêmes ports d'entrée Différents ports de sortie	> 5	**Validé**
4- Arbitrage de sortie entre les communications HIGH	> 3	HIGH	Différents ports d'entrée Mêmes ports de sortie	> 5	**Validé**
5- Arbitrage de sortie entre les communications LOW	> 3	LOW	Différents ports d'entrée Mêmes ports de sortie	> 5	**Validé**
6- Préemption à l'entrée	3	1 HIGH 2 LOW	Mêmes ports d'entrée Différents ports de sortie	> 5	**Validé**
7- Préemption à la sortie	3	1 HIGH 2 LOW	Différents ports d'netrée Mêmes ports de sortie	> 5	**Validé**

Nous avons simulé plusieurs scénarios en utilisant l'outil IFx. Le tableau 6.3 présente les différents scénarios simulés ainsi que leurs résultats. Il détaille les

résultats de simulations, le nombre d'itérations, les ports E/S utilisés et le niveau de criticité du flux considéré. Chaque ligne du tableau 6.3 décrit les détails de chaque scénario simulé. La colonne catégorie présente le scénario considéré.

Les deux premiers scénarios (lignes 1 et 2) vérifient les cas où il n'y a aucun conflit dans le réseau pour les messages.

Les autres catégories étudient les scénarios où il y a des interférences entre les messages. Les catégories 3, 4 et 5 vérifient le comportement de DAS face aux interférences entre les messages ayant le même niveau de criticité. Les catégories 6 et 7 vérifient le comportement du routeur DAS face aux interférences entre les messages ayant des niveaux de criticité différents.

La simulation interactive en utilisant l'outil IFx nous a permis de valider le comportement de la modélisation de DAS en IF.

6.5.3.2 Validation avec les observateurs IF

Après la simulation interactive, nous avons utilisé l'outil IFx et les observateurs IF pour vérifier de manière exhaustive les propriétés de DAS.

La figure 6.12 illustre l'observateur utilisé pour la vérification de la propriété 3. Dans l'état IDLE, il observe les requêtes du processus fils par l'arbitre d'entrée B. Si les deux variables internes (altCriticality et criticality) sont égales à false, l'observateur continue la surveillance des réponses du processus fils avec une valeur booléenne (ligne 9). Si ce paramètre égale à True, l'observateur passe à l'état IDLE, sinon il passe à l'état error state et coupe l'état d'exploration.

TABLE 6.4 – Validation exhaustive de DAS en utilisant les observateurs : espaces d'états et résultats. Ces expériences ont été effectuées sur Intel(R) Core(TM) i7-6700HQ CPU @ 2.60Ghz with 32GB RAM.

Propriétés	**Nombre d'états**	**Nombre de Transitions**	**Durée (hh :mm :ss)**	**Résultats**
Propriété 1	7 300 246	4 554 916	00 :02 :51	**prouvé**
Propriété 2	1 823 025	566 238	00 :00 :43	**prouvé**
Propriété 3	378 452	858 546	00 :00 :20	**prouvé**
Propriété 4	356 684	811 734	00 :00 :18	**prouvé**

Le tableau 6.4 présente les résultats de validation des propriétés de DAS. Dans ce tableau, nous présentons le nombre d'états, le nombre de transitions et le temps pris pour la validation exhaustive pour chaque propriété. Les propriétés définies au chapitre 5 ont été validées.

Toutes ces explorations se sont terminées normalement sans passer par l'état d'erreur des observateurs IF et sans aucune coupe d'exploration.

FIGURE 6.12 – The IF Cut Observer of the Property 3

```
   cut observer obs_prop_3;
   var response boolean; var index IndexType; var
Input_Arbiter_Stage_B pid;
   state idle #start ;
   match input DAS_request_Output_Arb_B(index) in
Input_Arbiter_Stage_B;
   nextstate input_DAS_request_Input_Arb_B_matched;
   endstate;
   state input_DAS_request_Input_Arb_B_matched;
   provided (({Input_Arbiter_Stage_B}0) instate idle and
({Input_Arbiter_Stage_B}0).altCriticality = false and
({Input_Arbiter_Stage_B}0).criticality = false);
   match output DAS_response_Input_Arb_B(response);
   nextstate decision_1;
   provided ...
   nextstate decision_2;
   endstate;
   state decision_2 #unstable ;
   provided (response = true);
   informal "--Validation Success!";
   nextstate idle;
   provided (response = false);
   informal "--Validation Fail!";
   cut;
   nextstate err;
   endstate;
   state decision_2 #unstable ; ... endstate;
   state err #error ; endstate;
   endobserver;
```

6.6 Bilan et conclusion

Dans ce chapitre, nous avons évalué DAS selon plusieurs niveaux d'abstraction en utilisant différentes méthodes et outils. Nous avons mesuré le coût en surface de DAS en utilisant une synthèse Verilog HDL. Ensuite, nous avons évalué le temps de communication fourni par DAS et son impact sur l'ordonnançabilité du système avec le simulateur en SystemC SHoC. Puis, nous avons modélisé DAS avec le langage IF dans le but de vérifier formellement le fonctionnement de DAS.

L'évaluation niveau circuit a montré que la mise en oeuvre de DAS n'est pas coûteuse en termes de surface. Le sur-coût en surface pour DAS par rapport à VC-Router est d'environ 2.5%.

L'évaluation niveau transaction a montré que DAS est capable d'assurer les contraintes temporelles pour les flux critiques. Avec un taux d'utilisation de 15% du réseau, DAS réduit le temps de communication des flux HIGH de 80% par rapport à VC-Router. Avec un taux d'utilisation de 14% du réseau, DAS réduit le temps de communication pour les flux HIGH de 64% par rapport à WNoC-Router.

Pour des messages de taille importante (>6 flits), DAS perd son efficacité face à WNoC-Router.

L'évaluation niveau système nous a permis de vérifier formellement le respect des propriétés d'un système à criticté mixte par DAS. Toutes les propriétés ont été validées en utilisant les observateurs IF.

Pour conclure, DAS est capable de supporter les systèmes à criticité mixte. Il nous permet d'assurer les contraintes temporelles pour les flux critiques tout en minimisant l'impact de partage de ressources sur les messages non critiques et cela en permettant une analyse de temps de communication non pessimiste.

Dans le chapitre suivant, nous proposons un modèle de communication pour les architectures réseau sur puce.

7

Ordonnancement des systèmes à criticité mixte sur des architectures NoC

Sommaire

7.1 Introduction

Le partage de ressources dans un NoC comme les routeurs et les liens physiques introduit des nouveaux types d'interférences entre les communications. Ces interférences peuvent impacter considérablement l'ordonnancement des tâches. L'analyse des communications au sein du NoC est donc indispensable pour l'analyse d'ordonnancement du système.

Cependant, la majorité des modèles de tâches et de flux de la littérature ne comportent pas toutes les informations nécessaires pour modéliser des systèmes à criticité mixte déployés sur un NoC. Les modèles de tâches existants négligent les communications et les différentes interférences possibles dans le NoC tandis que les modèles de flux négligent les tâches et leurs impacts sur les dates d'émissions et les échéances des flux. Afin de répondre à cette problématique, nous avons proposé un nouveau modèle de tâches et de flux appelé Dual Task and Flow Model (DTFM).

En outre, les techniques d'analyse d'ordonnancement ignorent les interférences liées aux ressources partagées au sein du NoC. Dans l'objectif de résoudre cette problématique et de pouvoir analyser l'ordonnancement des systèmes à criticité mixte déployés sur DAS, nous avons proposé ECTM. Ce dernier est un modèle de communication qui nous permet de considérer les interférences au sein du NoC. Il supporte séparément les deux techniques SAF et Wormhole. Nous ne pouvons pas donc utiliser directement ECTM pour DAS parce que les deux niveaux de préemptions de DAS, la combinaison entre SAF et Wormhole et les deux étage d'arbitrage de DAS ne sont pas encore considérés dans ECTM. En revanche,

ECTM peut être utilisé pour analyser les systèmes temps réel et les systèmes à criticité mixte déployées sur des NoC qui respectent les hypothèses prises dans ce travail (les hypothèses 2, 6, 7 et celles du tableau 7.3).

Dans ce chapitre, nous donnons les grandes lignes de notre approche. Ensuite, nous détaillons le modèle de tâches et de flux proposé. Puis, nous exposons le modèle de communication ECTM. Finalement, nous évaluons l'approche proposée.

7.2 Approche générale

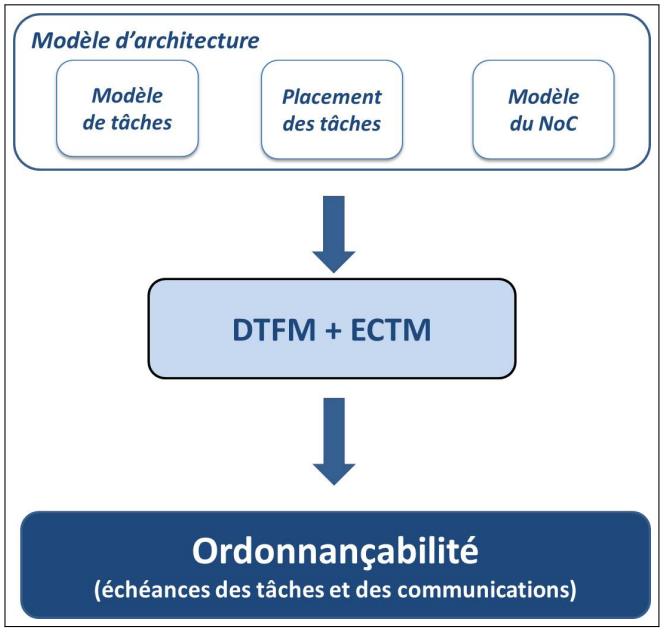


FIGURE 7.1 – Approche générale pour l'analyse d'ordonnancement d'un NoC - Objectif

L'approche générale proposée consiste à combiner DTFM et le modèle de communication ECTM. La figure 7.1 illustre l'objectif de notre approche : analyser l'ordonnançabilité d'un système temps réel déployé sur un NoC à partir d'un modèle d'architecture. Le modèle d'architecture est composé d'un modèle de NoC, d'un modèle de tâches et du placement des tâches.

La figure 7.2 détaille les différentes étapes utilisées dans notre approche.

Nous avons trois étapes :

1. DTFM calcule le modèle de flux à partir du modèle de tâches, du modèle de NoC et du placement des tâches.

2. Nous appliquons ensuite le modèle de communication ECTM au modèle d'architecture afin d'obtenir le modèle d'analyse correspondant.

   Le modèle d'architecture considéré par ECTM est celui généré lors de l'étape 1 qui est constitué d'un ensemble de tâches périodiques dépendantes déployées sur un NoC. Le modèle d'analyse produit par ECTM est, de son côté, constitué d'un ensemble de tâches périodiques dépendantes déployées sur une architecture multiprocesseur sans ressource partagée.

3. Nous effectuons l'analyse d'ordonnançabilité du système considéré en utilisant la simulation sur l'intervalle de faisabilité.

   L'intervalle de faisabilité est calculé en nous basant sur l'étude de Goossens et al [31].

   Dans cette étape, nous appliquons un algorithme d'ordonnancement multiprocesseurs qui est en adéquation avec le modèle d'analyse produit par ECTM. Pour cela, nous nous intéressons aux heuristiques d'ordonnancement par liste.

   Cette famille d'algorithmes d'ordonnancement multiprocesseurs ordonnance les systèmes temps réel avec des contraintes de précédences sur des architectures multiprocesseurs sans ressource partagée. Nous avons détaillé les caractéristiques de cette famille d'algorithme d'ordonnancement au chapitre 2.

   Highest Level First with Estimated Time (HLFET), Earliest Time First (EFT), Modified Critical Path (MCP) et Dynamic Level Scheduling (DLS) sont des exemples d'algorithmes d'ordonnancement multiprocesseurs [33].

Dans la suite, nous détaillons le modèle DTFM.

7.3 Dual Task and Flow Model (DTFM)

À partir du modèle de tâches, du modèle de NoC et du placement des tâches, DTFM calcule le modèle de flux correspondant.

Le modèle de flux généré spécifie les messages transmis dans le NoC et échangés entre les tâches. Afin de calculer le modèle de flux, DTFM utilise une fonction appelée G (cf section 7.3.3).

Dans la suite, nous présentons le modèle de tâches, le modèle de flux et les notations utilisés dans DTFM. Ensuite, nous définissons la fonction G utilisée par DTFM.

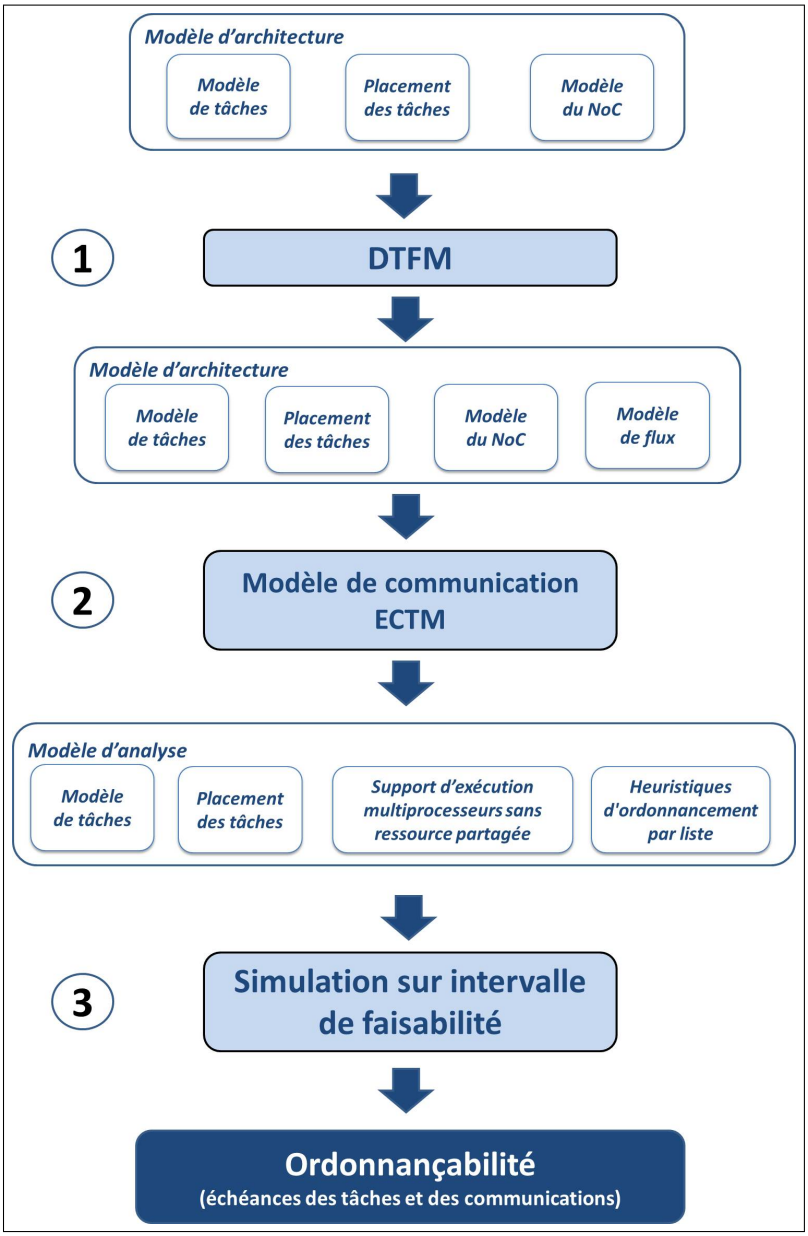


FIGURE 7.2 – Approche générale pour l'analyse d'ordonnancement d'un NoC - Moyens

7.3.1 Modèle de tâches

Nous supposons que le système est composé par un ensemble Γ de n tâches $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Chaque tâche τ_i est caractérisée par un ensemble de paramètres : $\tau_i = \{ O_{\tau_i}, C_{\tau_i}, D_{\tau_i}, P_{\tau_i}, Crit_{\tau_i}, PE_{\tau_i}, E \}$

avec :

- O_{τ_i} représente la date du premier réveil de la tâche τ_i. Elle représente la date à laquelle la tâche τ_i peut commencer son exécution.
- C_{τ_i} représente le pire temps d'exécution de la tâche τ_i.
- D_{τ_i} représente l'échéance de la tâche τ_i.
- P_{τ_i} représente la période de la tâche τ_i.
- $Crit_{\tau_i}$ représente le niveau de criticité de la tâche τ_i.
- PE_{τ_i} identifie l'unité de calcul qui exécute la tâche τ_i. Ce paramètre nous permet d'introduire le placement des tâches.
- $E : \Gamma \longrightarrow \Gamma^p$ où p $\in$ [0, n-1]

  $\tau_i \longrightarrow E(\tau_i) = \{\tau_j, \ldots, \tau_k \}$

  La fonction E nous permet d'introduire les contraintes de dépendance du modèle de tâches. La fonction E détermine, pour chaque tâche τ_i, toutes les tâches τ_j qui reçoivent des messages de la tâche τ_i.

7.3.2 Modèle de flux

Le système est également composé par un ensemble de m flux $\psi = \{\rho_1, \rho_2, \ldots, \rho_m\}$. Chaque flux est caractérisé par un ensemble de paramètres : $\rho_i = \{ O_{\rho_i}, C_{\rho_i}, D_{\rho_i}, P_{\rho_i}, Crit_{\rho_i}, NodeS_{\rho_i}, NodeD_{\rho_i}, F \}$

avec :

- O_{ρ_i} représente la date du premier réveil du flux ρ_i. Elle représente la date à laquelle le premier message du flux ρ_i peut commencer sa transmission.
- C_{ρ_i} représente le pire temps de communication de tous les messages du flux ρ_i.
- D_{ρ_i} représente l'échéance de tous les messages du flux ρ_i. Elle représente l'instant auquel la transmission d'un message doit être terminée et dont le dépassement provoque une violation de la contrainte temporelle.

- P_{ρ_i} représente la période de tous les messages du flux ρ_i. Elle représente la durée séparant deux instants de réveils successifs.
- $Crit_{\rho_i}$ représente le niveau de criticité du flux ρ_i.
- $NodeS_{\rho_i}$ est l'unité de calcul qui exécute la tâche source.
- $NodeD_{\rho_i}$ est l'unité de calcul qui exécute la tâche destination.

  Les paramètres $NodeS_{\rho_i}$ et $NodeD_{\rho_i}$ permettent à calculer les liens utilisés dans le réseau par le flux ρ_i.
- $F : \psi \longrightarrow \Omega^{links}$ où $links$ modélise le nombre de liens utilisés.

  $\rho_i \longmapsto \mathrm{F}(\rho_i) = \{\ e_{PE_jRj},\ e_{R_{p0}R_{k0}},\ \ldots,\ e_{R_{p1}R_{k1}},\ e_{RjPE_j}\ \}$

  Soit Ω l'ensemble des liens physiques du NoC. F est une fonction qui calcule les liens physiques utilisés par le flux ρ_i. Nous rappelons ici que e_{PE_jRj} représente le lien physique liant l'unité de calcul PE_j et le routeur R_j tandis que $e_{R_{p0}R_{k0}}$ représente le lien physique liant les routeurs R_{p0} et R_{k0}. La fonction F nous aide à comprendre les relations entre les flux et à détecter les interférences possibles dans le réseau. La fonction F dépend du placement des tâches et de l'algorithme de routage considéré.

7.3.3 La fonction G

En partant du modèle de tâches, du modèle de NoC et du placement des tâches, DTFM calcule le modèle de flux correspondant en utilisant la fonction G.

L'ensemble de départ de la fonction G est Γ^2 tandis que l'ensemble d'arrivée est ψ. L'image de chaque couple de tâche $(\tau_i,\ \tau_j)$ par la fonction G est un flux $\rho_{i,j}$ à condition que les tâches τ_i et τ_j soient dépendantes. Nous définissons donc la fonction G comme suit :

$\mathrm{G} : \Gamma^2 \longrightarrow \psi$

$\forall\ \mathrm{i} \in [1\ ..\ \mathrm{n}\ ];\ \forall\ \mathrm{j} \in [1\ ..\ \mathrm{n}]$

$(\tau_i,\ \tau_j) \longmapsto \mathrm{G}(\ (\tau_i,\ \tau_j)\ ) = \rho_{i,j}$

Si $\tau_j \in \mathrm{E}(\tau_i)$ alors $\mathrm{G}(\ (\tau_i,\ \tau_j)\ ) = \rho_{i,j}$

avec

- $Crit_{\rho_{i,j}} = Crit_{\tau_i}$
- $NodeS_{\rho_{i,j}} = PE_{\tau_i}$
- $NodeD_{\rho_{i,j}} = PE_{\tau_j}$

- $O_{\rho_{i,j}} = O_{\tau_i} + C_{\tau_i}$
- $P_{\rho_{i,j}} = P_{\tau_i}$
- $D_{\rho_{i,j}} = D_{\tau_i} - C_{\tau_i}$
- $C_{\rho_{i,j}}$: pire temps de communication

  La fonction G calcule le pire temps de communication de chaque flux en tenant compte de la configuration du NoC et de l'état du réseau. En d'autres termes, elle prend en compte la technique de commutation, l'utilisation ou non des canaux virtuels, le niveau de préemption adopté, la technique d'arbitrage et le nombre de flux qui partagent les mêmes liens. Elle implante les analyses de temps de communication proposées par Shi dans [32].

La fonction G est une fonction surjective. Chaque élément du domaine d'arrivée possède au moins un antécédent dans le domaine de départ.

$$\forall\ \rho \in \psi\,;\ \exists\ (\tau_i,\ \tau_j) \text{ tel que } \rho = \mathrm{G}(\,(\tau_i,\ \tau_j)\,)$$

La fonction G n'est pas injective. Nous pouvons trouver des éléments du domaine de départ qui n'ont pas d'image dans le domaine d'arrivé. En d'autres termes, deux tâches indépendantes ne possèdent pas une image dans l'ensemble de flux.

Dans la suite, nous présentons un exemple d'utilisation de DTFM.

7.3.4 Exemple

Dans cet exemple, nous considérons un NoC grille 2D 4×4. L'algorithme de routage considéré est XY. Nous illustrons figure 7.3 le modèle de tâches utilisé.

Nous considérons dans cet exemple 5 tâches périodiques dépendantes $\{\ \tau_1,\ \tau_2,\ \tau_3,\ \tau_4,\ \tau_5\ \}$.

Les paramètres des tâches sont présentés dans le tableau 7.1. Pour rappel, le paramètre E spécifie les contraintes de dépendance entre les tâches :

- $\mathrm{E}(\tau_1) = \{\tau_2, \tau_3\}$; La tâche τ_1 envoie des messages aux tâches τ_2 et τ_3.
- $\mathrm{E}(\tau_2) = \{\tau_5\}$; La tâche τ_2 envoie des messages vers la tâche τ_5.
- $\mathrm{E}(\tau_3) = \{\tau_4, \tau_5\}$; La tâche τ_3 envoie des messages aux tâches τ_4 et τ_5.
- $\mathrm{E}(\tau_4) = \{\tau_5\}$; La tâche τ_4 envoie des messages vers la tâche τ_5.

À partir du modèle de tâches, du modèle de NoC et du placement des tâches, nous calculons le modèle de flux correspondant en utilisant la fonction G.

Nous avons 6 flux $\{\ \rho_{1,2},\ \rho_{1,3},\ \rho_{2,5},\ \rho_{3,4},\ \rho_{3,5},\ \rho_{4,5}\ \}$ calculés comme suit :

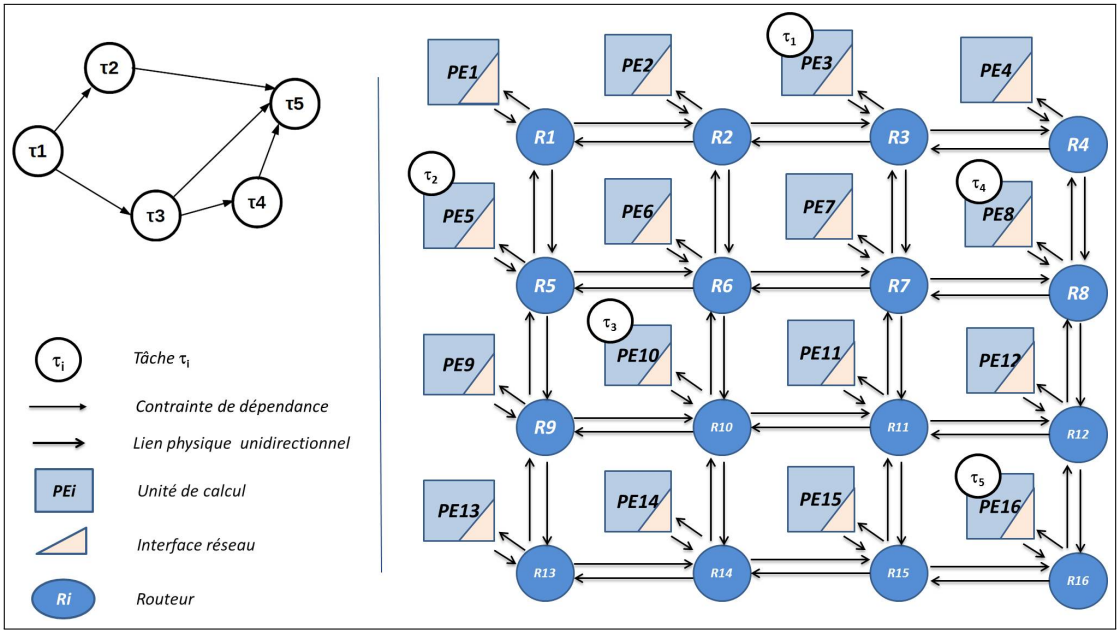


FIGURE 7.3 – Dual Task and Flow Model (DTFM)
Exemple - Modèle de tâches et modèle de NoC.
L'unité de calcul PE3 exécute la tâche τ_1. L'unité de calcul PE5 exécute la tâche τ_2. L'unité de calcul PE8 exécute la tâche τ_4. L'unité de calcul PE10 exécute la tâche τ_3. L'unité de calcul PE16 exécute la tâche τ_5.

Tâche	O_{τ_i}(s)	P_{τ_i}(s)	C_{τ_i} (μs)	D_{τ_i}(s)	PE_{τ_i}	E
τ_1	1	6	100	2	$PE3$	τ_2 τ_3
τ_2	3	2	100	2	$PE5$	τ_5
τ_3	3	2	300	2	$PE10$	τ_4 τ_5
τ_4	5	2	500	2	$PE8$	τ_5
τ_5	7	2	100	2	$PE16$	–

TABLE 7.1 – Dual Task and Flow Model (DTFM)
Exemple - Modèle de tâches - Paramètres

Flux	$NodeS_{\rho_{i,j}}$	$NodeD_{\rho_{i,j}}$	F
$\rho_{1,2}$	PE3	PE5	e_{PE3R3} e_{R3R2} e_{R2R1} e_{R1R5} e_{R5PE5}
$\rho_{1,3}$	PE3	PE10	e_{PE3R3} e_{R3R2} e_{R2R6} e_{R6R10} $e_{R10PE10}$
$\rho_{2,5}$	PE5	PE16	e_{PE5R5} e_{R5R6} e_{R6R7} e_{R7R8} e_{R8R12} e_{R12R16} $e_{R16PE16}$
$\rho_{3,4}$	PE10	PE8	$e_{PE10R10}$ e_{R10R11} e_{R11R12} e_{R12R8} e_{R8PE8}
$\rho_{3,5}$	PE10	PE16	$e_{PE10R10}$ e_{R10R11} e_{R11R12} e_{R12R16} $e_{R16PE16}$
$\rho_{4,5}$	PE8	PE16	e_{PE8R8} e_{R8R12} e_{R12R16} $e_{R16PE16}$

TABLE 7.2 – Dual Task and Flow Model (DTFM)
Exemple - Modèle de flux calculé par DTFM

- $G((\tau_1, \tau_2)) = \rho_{1,2}$
- $G((\tau_1, \tau_3)) = \rho_{1,3}$
- $G((\tau_2, \tau_5)) = \rho_{2,5}$
- $G((\tau_3, \tau_4)) = \rho_{3,4}$
- $G((\tau_3, \tau_5)) = \rho_{3,5}$
- $G((\tau_4, \tau_5)) = \rho_{4,5}$

Le tableau 7.2 présente les paramètres de chaque flux. Le paramètre F spécifie les liens physiques utilisés par chaque flux.

7.4 Modèles de communication pour les architectures NoC

Les communications au sein du NoC et les différents délais causés par le partage de ressources doivent être considérés dans les techniques d'analyse d'ordonnancement existantes.

Shi et al ont proposé un modèle de communication (WCCTM) pour les architectures NoC [55]. Ce modèle est basé sur les scénarios pire cas.

Dans la suite, nous formalisons le modèle proposé par Shi. Puis, nous proposons ECTM, un modèle de communication pour les architectures NoC qui améliore la précision de WCCTM.

La figure 7.4 présente l'approche générale de ces modèles de communications. Le modèle d'entrée est la description de l'architecture du système temps réel à déployer sur un NoC. À partir du modèle d'architecture, nous proposons de générer un modèle d'analyse qui peut être exploité pour l'analyse d'ordonnançabilité.

Dans la suite, nous décrivons en détail le modèle d'architecture et le modèle d'analyse.

7.4.1 Modèle d'architecture

Nous supposons que le système à analyser est constitué d'un ensemble de tâches périodiques dépendantes déployées sur un réseau sur puce. Les tâches et les flux sont conformes aux modèles introduits dans DTFM.

ECTM et WCCTM supportent les NoC Wormhole et les NoC SAF. Les modèles proposés dans ce chapitre sont :

- Worst Case Communication Time Model (WCCTM)

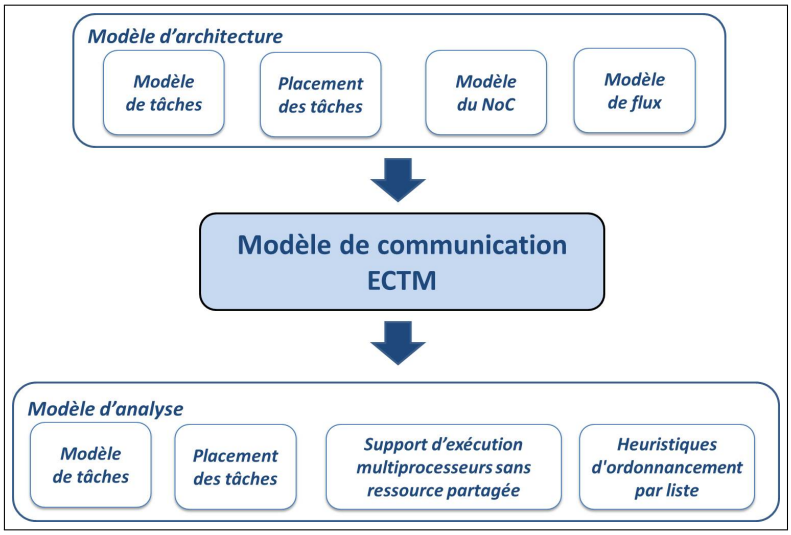


FIGURE 7.4 – Modèle de communication pour les architectures NoC - Approche générale

Modèle de communication	WCCTM	$ECTM_{SAF}$	$ECTM_{Wormhole}$
Topologie et dimension	2D mesh	2D mesh	2D mesh
Algorithme de routage	XY	XY	XY
Technique de commutation	Wormhole / SAF	SAF	Wormhole
Technique d'arbitrage	PF / RR	PF / RR	PF / RR
Canaux virtuels	Avec / Sans	Avec	Avec
Niveau de préemption	Flit / paquet	Paquet	Flit

TABLE 7.3 – Configurations NoC considérées

- Exact Communication Time Model pour un NoC SAF ($ECTM_{SAF}$)
- Exact Communication Time Model pour un NoC Wormhole ($ECTM_{Wormhole}$)

Nous détaillons dans tableau 7.3 la configuration des NoC considérés par ECTM et WCCTM.

7.4.2 Modèle d'analyse

Le modèle d'analyse est constitué d'un ensemble de tâches périodiques dépendantes déployées sur une architecture multiprocesseur de type d'interconnexion bus sans ressource partagée.

Nous justifions le choix de ce modèle d'analyse par deux arguments. Premièrement, l'un des avantages de ce type d'interconnexion est d'avoir une latence des communications fixe quels que soient l'émetteur et le récepteur. Deuxièmement, plusieurs

algorithmes multiprocesseurs ont été proposés qui considèrent ce type d'interconnexion. Nous pouvons citer à titre d'exemple : Highest Level First with Estimated Time (HLFET), Earliest Time First (EFT), Modified Critical Path (MCP) et Dynamic Level Scheduling (DLS).

Après avoir calculé le modèle d'analyse, nous pouvons vérifier l'ordonnançabilité de ce modèle d'analyse en utilisant une simulation de l'ordonnancement sur l'intervalle de faisabilité [31].

Dans la suite, nous détaillons comment produire le modèle d'analyse pour chacun des modèles de communication WCCTM, $ECTM_{SAF}$ et $ECTM_{Wormhole}$.

7.4.3 Worst Case Communication Time Model

Pour WCCTM, le modèle d'analyse est produit par les règles suivantes :

1. **Règle 1**

   Chaque unité de calcul du modèle d'architecture est conservée dans le modèle d'analyse. Cependant, tous les routeurs et les liens physiques du support d'exécution du modèle d'architecture sont supprimés dans le modèle d'analyse.

2. **Règle 2**

   Chaque tâche τ_i du modèle d'architecture est conservée dans le modèle d'analyse. Cependant, chaque flux ρ_i du modèle d'architecture est transformé en un couple tâche/unité de calcul (τ_{ρ_i} et $PE\rho_i$) dans le modèle d'analyse.

3. **Règle 3**

   Les paramètres de la tâche τ_{ρ_i} sont calculés en fonction du flux ρ_i.

   Supposons que le flux ρ_i est envoyé par la tâche τ_{Source} vers la tâche $\tau_{Destination}$. τ_{ρ_i} est donc caractérisée par :

   - $O_{\tau_{\rho_i}} = O_{\rho_i}$
   - $P_{\tau_{\rho_i}} = P_{\rho_i}$
   - $C_{\tau_{\rho_i}} = C_{\rho_i}$

     Dans ce modèle, la capacité de la tâche τ_{ρ_i} est égale au pire temps de communication du flux ρ_i. Nous notons ici que le pire temps de communication dépend de la technique de commutation adoptée par le NoC considéré.

     C_{ρ_i} est calculé avec les méthodes proposées dans [38, 32] pour les NoC Wormhole ou dans [32] pour les NoC SAF.

- $D_{\tau_{\rho_i}} = D_{\rho_i}$
- $Node_{\tau_{\rho_i}} = PE\rho_i$
  Notons ici que pour chaque flux de modèle d'architecture, WCCTM crée une nouvelle unité de calcul appelée $PE\rho_i$. Cette dernière est le support d'exécution de la tâche τ_{ρ_i}.
- $E(\tau_{\rho_i}) = \tau_{Destination}$

7.4.3.1 Exemple

La figure 7.5 illustre un exemple pour le modèle WCCTM. Dans cet exemple, nous considérons un NoC constitué de quatre unités de calcul (PE_1, PE_2, PE_3, PE_4) connectées entre elles via quatre routeurs (R1, R2, R3, R4) et des liens physiques unidirectionnels. Nous considérons également quatre tâches périodiques dépendantes τ_1, τ_2, τ_3 et τ_4. La tâche τ_1 s'exécute sur l'unité de calcul PE_1. La tâche τ_2 s'exécute sur l'unité de calcul PE_3. La tâche τ_3 s'exécute sur l'unité de calcul PE_2. La tâche τ_4 s'exécute sur l'unité de calcul PE_4. La tâche τ_1 envoie les messages de ρ_1 vers la tâche τ_2 et la tâche τ_3 envoie les messages de ρ_2 vers la tâche τ_4.

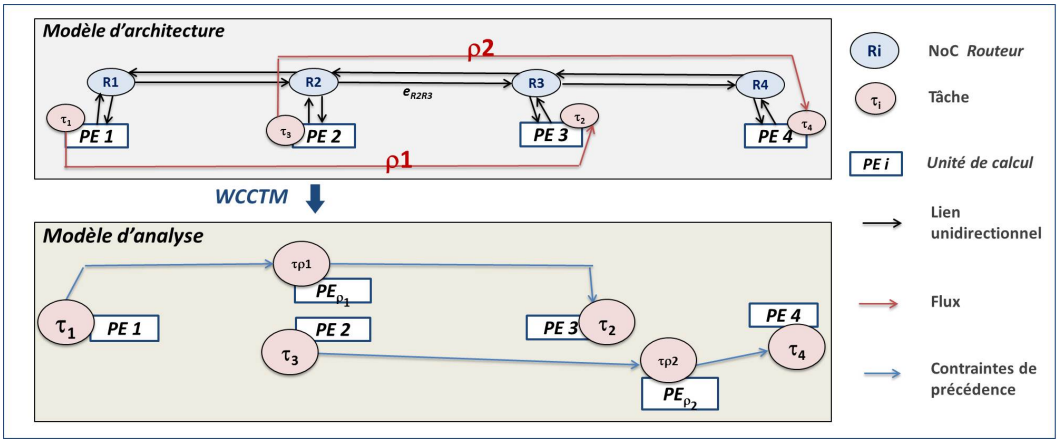


FIGURE 7.5 – Worst Case Communication Time Model (WCCTM) - Exemple de transformation

Pour cet exemple, le modèle d'analyse est produit ainsi :

- (règle 1) Tous les routeurs et les liens physiques du NoC du modèle d'architecture sont supprimés dans le modèle d'analyse. En gardant les mêmes unités de calcul, le modèle d'analyse comporte un support d'exécution multiprocesseur de type bus sans ressource partagée.

- (règles 2 + 3) Ensuite, le flux ρ_1 du modèle d'architecture est transformé en une tâche τ_{ρ_1} et une unité de calcul PE_{ρ_1} dans le modèle d'analyse. La tâche τ_{ρ_1} s'exécute sur l'unité de calcul PE_{ρ_1}.

  De la même façon, le flux ρ_2 est transformé en une tâche τ_{ρ_2} et une unité de calcul PE_{ρ_2} dans le modèle d'analyse. La tâche τ_{ρ_2} s'exécute sur l'unité de calcul PE_{ρ_2}.

Le partage du lien physique e_{R2R3} entre les flux ρ_1 et ρ_2 dans le modèle d'architecture est considéré dans les temps d'exécution des tâches τ_{ρ_1} et τ_{ρ_2} dans le modèle d'analyse. Nous notons ici qu'il n'y a pas de partage de ressources entre les tâches τ_{ρ_1} et τ_{ρ_2} dans le modèle d'analyse.

7.4.4 Exact Communication Time Model pour les NoC SAF

$ECTM_{SAF}$ est conçu pour les NoC SAF. Pour $ECTM_{SAF}$, le modèle d'analyse est produit par les règles suivantes :

1. **Règle 1**

   Chaque unité de calcul du modèle d'architecture est conservée dans le modèle d'analyse. Cependant tous les routeurs de support d'exécution du modèle d'architecture sont supprimés dans le modèle d'analyse.

2. **Règle 2**

   Chaque lien physique unidirectionnel liant deux routeurs e_{xy} du modèle d'architecture est remplacé par une unité de calcul $PE_{R_xR_y}$ dans le modèle d'analyse.

   Chaque lien physique unidirectionnel liant un routeur à une unité de calcul $e_{R_xPE_x}$ (respectivement $e_{PE_xR_x}$) du modèle d'architecture est remplacé par une unité de calcul $PE_{R_xPE_x}$ (respectivement $PE_{PE_xR_x}$) dans le modèle d'analyse.

   Nous notons ici que ces unités de calcul utilisent un algorithme d'ordonnancement en accord avec la technique d'arbitrage adoptée dans le NoC.

3. **Règle 3**

   Chaque tâche τ_i du modèle d'architecture est conservée dans le modèle d'analyse. Chaque flux ρ_i du modèle d'architecture est remplacé par un ensemble de $nbrlink_{\rho_i}$ tâches dans le modèle d'analyse. $nbrlink_{\rho_i}$ est le nombre de liens physiques utilisés par le flux.

   Dans la suite de cette partie, nous appelons cet ensemble de tâches : tâches$_{paquet}$

   En appliquant $ECTM_{SAF}$, un flux qui utilise 3 liens physiques se transforme donc en un ensemble tâches$_{paquet}$ de 3 tâches.

4. **Règle 4**

   Nous calculons les paramètres des tâches$_{paquet}$ en fonction des paramètres des flux.

   Chaque flux ρ_i du modèle d'architecture qui utilise $nbrlink_{\rho_i}$ liens physiques et qui est émis par la tâche τ_{source} vers la tâche $\tau_{destination}$, se transforme en un ensemble de $nbrlink_{\rho_i}$ tâches dans le modèle d'analyse : $\Gamma_{\rho_i} = \{ \tau_{\rho_{i,1}}, \tau_{\rho_{i,2}}, \ldots, \tau_{\rho_{i,nbrlink_{\rho_i}}} \}$.

   Pour $j \in [\ 1 \ldots nbrlink_{\rho_i}\ ]$, $\tau_{\rho_{i,j}}$ est caractérisée par :

   - $O_{\tau_{\rho_{i,j}}} = O_{\rho_i}$
   - $P_{\tau_{\rho_{i,j}}} = P_{\rho_i}$
   - $C_{\tau_{\rho_{i,j}}} = PD_{Onelink}$
     $PD_{Onelink}$ représente le temps de trajet pour un seul lien physique. C'est le temps de communication de flux ρ_i pour un lien physique en l'absence de contention de communication dans le réseau.
   - $D_{\tau_{\rho_{i,j}}} = D_{\rho_i}$
   - $PE_{\tau_{\rho_{i,j}}}$: unité de calcul qui exécute la tâche $\tau_{\rho_{i,j}}$.

     $$PE\tau_{\rho_{i,j}} = \begin{cases} PE_{R_xPE_x} & \text{si j} = nbrlink_{\rho_i} \\ PE_{PE_xR_x} & \text{si j =1} \\ PE_{R_xR_y} & \text{si 1<j<}nbrlink_{\rho_i} \end{cases}$$

     Nous notons ici que $e_{R_xR_y}$ représente un des liens physiques utilisés par le flux ρ_i et qui est transformé en unité de calcul $PE_{R_xR_y}$ dans le modèle d'analyse.
   - $E(\tau_{\rho_{i,j}})$: successeurs de la tâche $\tau_{\rho_{i,j}}$.

     $$E(\tau_{\rho_{i,j}}) = \begin{cases} \tau_{\rho_{i,j+1}} & \text{si j<}nbrlink_{\rho_i} \\ \tau_{destination} & \text{si j} = nbrlink_{\rho_i} \end{cases}$$

7.4.4.1 Exemple

Les figures 7.6 et 7.7 illustrent un exemple pour le modèle $ECTM_{SAF}$. Dans cet exemple, nous considérons le même système d'architecture que dans l'exemple précédent (figure 7.5). Nous considérons un seul flux dans figure 7.6 et deux flux dans figure 7.7.

Pour cet exemple, le modèle d'analyse est produit ainsi :

- (règles 1 + 2) Dans le modèle d'analyse produit, $ECTM_{SAF}$ supprime tous les routeurs du modèle d'architecture. Ensuite, chaque lien physique du

NoC du modèle d'architecture est transformé en une unité de calcul dans le modèle d'analyse.

- (règles 3 + 4) Puisque le flux ρ_1 utilise 4 liens physiques (e_{PE1R1}, e_{R1R2}, e_{R2R3} et e_{R3PE3}), le flux ρ_1 du modèle d'architecture est transformé en 4 tâches dans le modèle d'analyse : $\tau_{\rho_1,1}$, $\tau_{\rho_1,2}$, $\tau_{\rho_1,3}$ et $\tau_{\rho_1,4}$.

  La tâche $\tau_{\rho_1,1}$ s'exécute sur l'unité de calcul PE_{PE1R1}. La tâche $\tau_{\rho_1,2}$ s'exécute sur l'unité de calcul PE_{R1R2}. La tâche $\tau_{\rho_1,3}$ s'exécute sur l'unité de calcul PE_{R2R3} et la tâche $\tau_{\rho_1,4}$ sur l'unité de calcul PE_{R3PE3}.

  De la même façon, le flux ρ_2 est transformé en 4 tâches dans le modèle d'analyse : $\tau_{\rho_2,1}$, $\tau_{\rho_2,2}$, $\tau_{\rho_2,3}$ et $\tau_{\rho_2,4}$.

Le partage du lien physique e_{R2R3} entre les flux ρ_1 et ρ_2 dans le modèle d'architecture se traduit dans le modèle d'analyse par le partage de l'unité de calcul PE_{R2R3} entre les tâches $\tau_{\rho_1,3}$ et $\tau_{\rho_2,2}$.

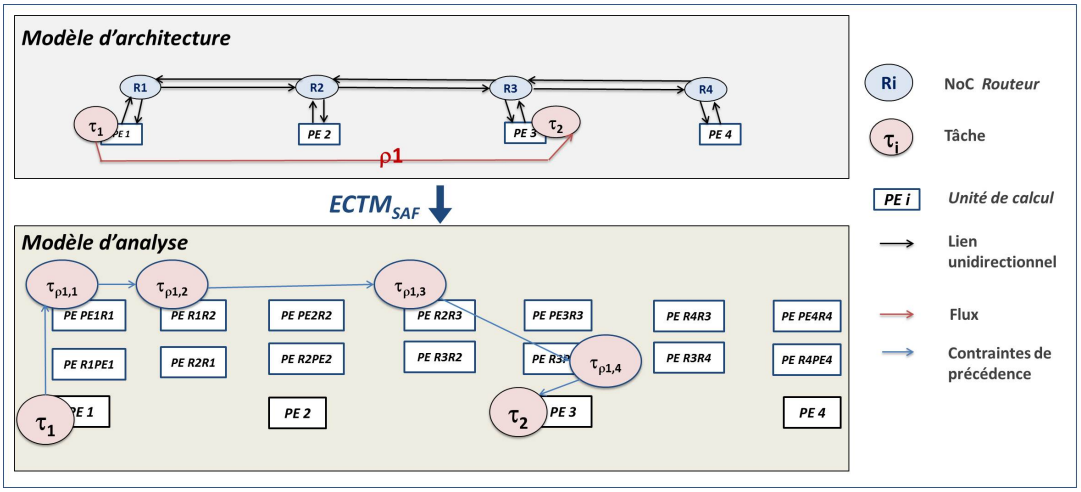


FIGURE 7.6 – $ECTM_{SAF}$ - Exemple de transformation - 1 flux (2 flits)

7.4.5 Exact Communication Time Model pour les NoC Wormhole

Ce modèle est conçu pour les NoC Wormhole. Pour $ECTM_{Wormhole}$, le modèle d'analyse est produit par les règles suivantes :

1. **Règle 1**

   Chaque unité de calcul du modèle d'architecture est conservée dans le modèle d'analyse. Cependant, tous les routeurs de support d'exécution du modèle d'architecture sont supprimés dans le modèle d'analyse.

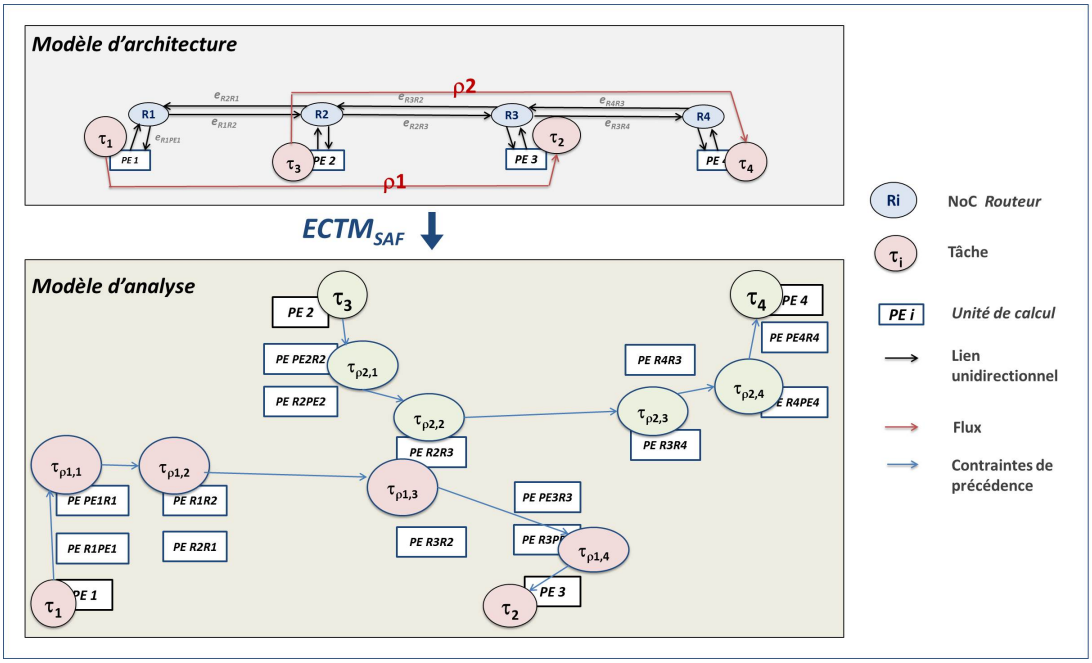


FIGURE 7.7 – $ECTM_{SAF}$ - Exemple de transformation - 2 flux (2 flits)

2. **Règle 2**

   Chaque lien physique unidirectionnel liant deux routeurs $e_{R_xR_y}$ du modèle d'architecture est remplacé par une unité de calcul $PE_{R_xR_y}$ dans le modèle d'analyse.

   Chaque lien physique unidirectionnel liant un routeur à une unité de calcul $e_{R_xPE_x}$ (respectivement $e_{PE_xR_x}$) du modèle d'architecture est remplacé par une unité de calcul $PE_{R_xPE_x}$ (respectivement $PE_{PE_xR_x}$) dans le modèle d'analyse.

   Nous notons ici que ces unités de calcul utilisent un algorithme d'ordonnancement en accord avec la technique d'arbitrage adoptée dans le NoC.

3. **Règle 3**

   Chaque tâche τ_i du modèle d'architecture est conservée dans le modèle d'analyse. Chaque flux ρ_i du modèle d'architecture est remplacé par un ensemble de nb tâches dans le modèle d'analyse. nb est le produit du nombre de liens physiques utilisés par la taille du flux en flit.

$$nb = \text{nombre de liens physiques utilisés} \cdot \text{taille du flux}$$

   Pour $ECTM_{Wormhole}$, un flux de 2 flits et qui utilise 3 liens physiques est transformé en un ensemble de 6 tâches.

4. **Règle 4**

   Chaque flux ρ_i du modèle d'architecture de taille $size_{\rho_i}$ qui utilise $nbrlink_{\rho_i}$ liens physiques et qui est émis par la tâche τ_{source_i} vers la tâche $\tau_{destination_i}$ est transformé en un ensemble de $nbrlink_{\rho_i}$. $size_{\rho_i}$ tâches (Γ_{ρ_i}) dans le modèle d'analyse.

   $\Gamma_{\rho_i} = \{ \tau_{\rho_{i,1,1}}, \tau_{\rho_{i,a,b}}, \ldots, \tau_{\rho_{i,size_{\rho_i},nbrlink_{\rho_i}}} \}$ pour (a,b) $\in [1, size_{\rho_i}]$ x $[1,nbrlink_{\rho_i}]$

   La tâche $\tau_{\rho_{i,a,b}}$ dans le modèle d'analyse est l'image du a^{ieme} flit qui utilise le b^{ieme} lien du flux ρ_i du modèle d'architecture.

   Nous calculons les paramètres des tâches de l'ensemble Γ_{ρ_i} en fonction des paramètres du flux ρ_i. Pour (a,b) $\in [1, size_{\rho_i}]$ x $[1,nbrlink_{\rho_i}]$, la tâche $\tau_{\rho_{i,a,b}}$ est caractérisée par :

   - $O_{\tau_{\rho_i,a,b}} = O_{\rho_i}$
   - $P_{\tau_{\rho_i,a,b}} = P_{\rho_i}$
   - $C_{\tau_{\rho_i,a,b}} : PD_{Oneflit/Onelink}$
     $PD_{Oneflit/Onelink}$ représente le temps de trajet d'un flit sur un lien. C'est le temps de communication d'un flit pour un seul lien physique sans considérer les possibles contentions de communication dans le NoC.
   - $D_{\tau_{\rho_i,a,b}} = D_{\rho_i}$
   - $PE_{\tau_{\rho_i,a,b}}$: unité de calcul qui exécute la tâche $\tau_{\rho_{i,a,b}}$.

     $$PE_{\tau_{\rho_i,a,b}} = \begin{cases} PE_{R_xPE_x} & \text{si j} = nbrlink_{\rho_i} \\ PE_{PE_xR_x} & \text{si j} = 1 \\ PE_{R_xR_y} & \text{si } 1 < \text{j} < nbrlink_{\rho_i} \end{cases}$$

     Nous notons ici que $e_{R_xR_y}$ représente un des liens physiques utilisés par le flux ρ_i et qui est transformé en unité de calcul $PE_{R_xR_y}$ dans le modèle d'analyse.

   - $E(\tau_{\rho_i,a,b})$: successeurs de la tâche $\tau_{\rho_{i,a,b}}$.

     $$E(\tau_{\rho_i,a,b}) = \begin{cases} \tau_{\rho_i,a+1,b},\ \tau_{\rho_i,a,b+1} & \text{si a} < size_{\rho_i} \text{ et b} < nbrlink_{\rho_i} \\ \tau_{\rho_i,a+1,b} & \text{si a} < size_{\rho_i} \text{ et b} = nbrlink_{\rho_i} \\ \tau_{\rho_i,a,b+1} & \text{si a} = size_{\rho_i} \text{ et b} < nbrlink_{\rho_i} \\ \tau_{destination} & \text{si a} = size_{\rho_i} \text{ et b} = nbrlink_{\rho_i} \end{cases}$$

     Nous notons ici que certaines tâches de l'ensemble Γ_{ρ_i} possèdent un seul successeur alors que d'autres peuvent avoir deux successeurs.

7.4.5.1 Exemple

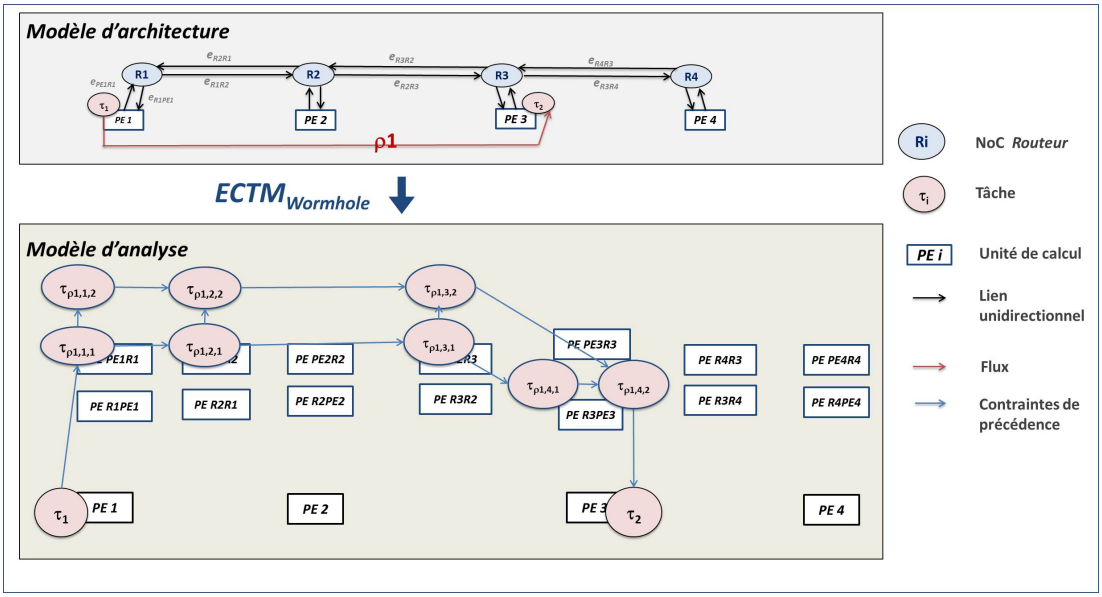


FIGURE 7.8 – $ECTM_{Wormhole}$ - Exemple de transformation - 1 flux

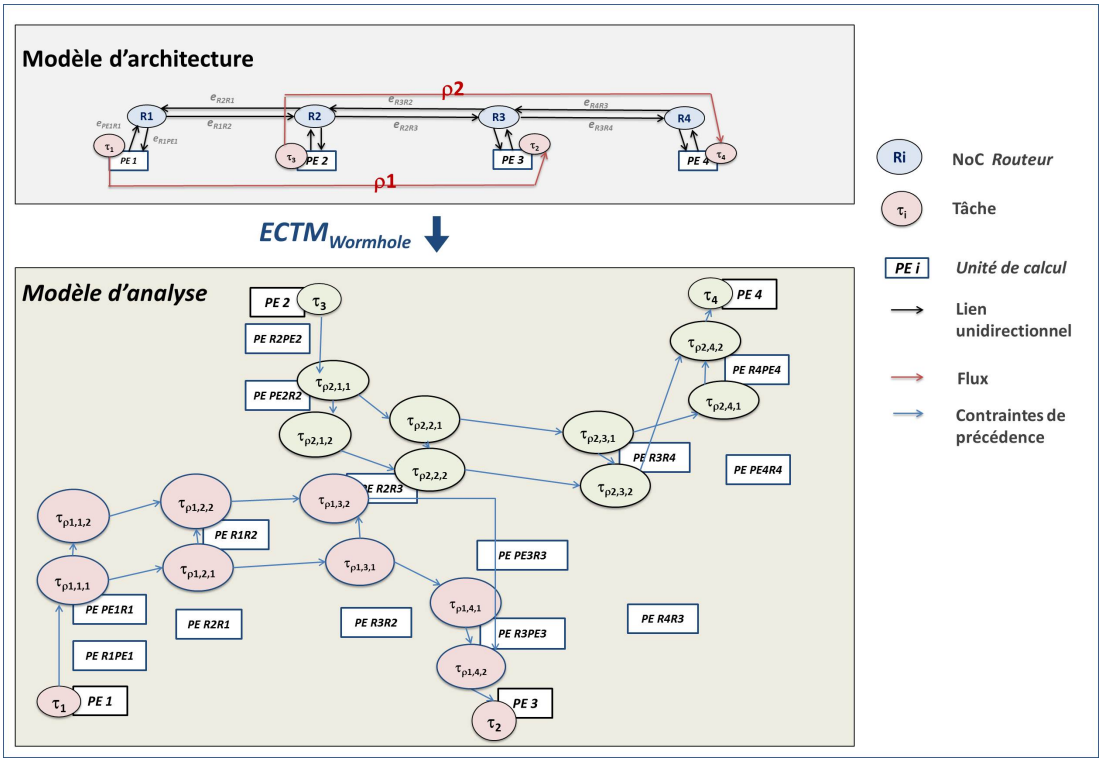


FIGURE 7.9 – $ECTM_{Wormhole}$ - Exemple de transformation - 2 flux

Les figures 7.8 et 7.9 illustrent un exemple pour le modèle $ECTM_{Wormhole}$. Dans cet exemple, nous considérons la même architecture que le premier exemple (figure 7.5). Nous considérons un flux dans la figure 7.8 et deux flux la figure 7.9.

Pour cet exemple, le modèle d'analyse est produit ainsi :

- (règles 1 + 2) Tous les routeurs du modèle d'architecture sont supprimés dans le modèle d'analyse. Ensuite, chaque lien physique du NoC du modèle d'architecture est transformé en une unité de calcul dans le modèle d'analyse.

- (règles 3 + 4) Supposons que les flux ρ_1 et ρ_2 sont de taille de 2 flits. Puisque que le flux ρ_1 utilise 4 liens physiques (e_{PE1R1}, e_{R1R2}, e_{R2R3} et e_{R3PE3}) et est de taille de 2 flits, $ECTM_{Wormhole}$ transforme le flux ρ_1 en 8 (4×2) tâches : $\tau_{\rho_1,1,1}$, $\tau_{\rho_1,1,2}$, $\tau_{\rho_1,2,1}$, $\tau_{\rho_1,2,2}$, $\tau_{\rho_1,3,1}$, $\tau_{\rho_1,3,2}$ $\tau_{\rho_1,4,1}$ et $\tau_{\rho_1,4,2}$.

  Les tâches $\tau_{\rho_1,1,1}$ et $\tau_{\rho_1,1,2}$ s'exécutent sur l'unité de calcul PE_{PE1R1}. Les tâches $\tau_{\rho_1,2,1}$ et $\tau_{\rho_1,2,2}$ s'exécutent sur l'unité de calcul PE_{R1R2}. Les tâches $\tau_{\rho_1,3,1}$ et $\tau_{\rho_1,3,2}$ s'exécutent sur l'unité de calcul PE_{R2R3}. Les tâches $\tau_{\rho_1,4,1}$ et $\tau_{\rho_1,4,2}$ s'exécutent sur l'unité de calcul PE_{R3PE3}.

  De même façon, le flux ρ_2 est transformé en 8 (4×2) tâches : $\tau_{\rho_2,1,1}$, $\tau_{\rho_2,1,2}$, $\tau_{\rho_2,2,1}$, $\tau_{\rho_2,2,2}$, $\tau_{\rho_2,3,1}$, $\tau_{\rho_2,3,2}$ $\tau_{\rho_2,4,1}$ et $\tau_{\rho_2,4,2}$.

Le partage du lien physique e_{R2R3} entre les flux ρ_1 et ρ_2 dans le modèle d'architecture est traduit dans le modèle d'analyse par le partage de l'unité de calcul PE_{R2R3} entre les tâches $\tau_{\rho_2,2,1}$, $\tau_{\rho_2,2,2}$, $\tau_{\rho_1,3,1}$, $\tau_{\rho_1,3,2}$.

7.5 Validation des modèles de communications

ECTM transforme chaque flux ρ_i du modèle d'architecture en un ensemble de tâches Γ_{ρ_i} dans le modèle d'analyse. Nous montrons, dans cette partie, que les interférences de communications dues au partage de ressources dans le modèle d'architecture sont conservées dans le modèle d'analyse produit par ECTM.

Avant d'entamer la validation de notre approche, nous donnons quelques définitions nécessaires pour cette validation.

Définition 73. ***(Temps de réponse d'un DAG)***

Le temps de réponse C_Γ d'un DAG Γ est la différence entre l'instant où la dernière tâche de sortie de l'ensemble Γ termine son exécution et l'instant où la première tâche d'entrée de l'ensemble Γ est activée.

Dans ce chapitre, nous employons des DAG linéaires :

Définition 74. *(**DAG linéaire**)*

Soit Γ *un DAG composé de n tâches périodiques* $\Gamma = \{ \tau_1, \cdots \tau_n \}$. *Un DAG linéaire est un DAG tel que :*

- Γ *possède une seule tâche d'entrée et une seule tâche de sortie.*
- *À l'exception de la tâche d'entrée et de la tâche de sortie, chaque tâche de* Γ *possède un seul successeur et un seul prédécesseur.*

  $\forall i \in [2, n-1], E(\tau_i) = \tau_{i+1}$

Cette définition du DAG linéaire est inspirée des transactions linéaires [94].

Définition 75. *(**Temps de réponse d'un DAG linéaires**)*

Le temps de réponse d'un DAG linéaire est égal à la somme des temps de réponse des tâches qui constituent le DAG.

Dans cette partie, nous montrons le théorème suivant :

Théoreme 1. *Supposons un flux* ρ_i *envoyé par la tâche* τ_1 *vers la tâche* τ_2. *ECTM transforme le flux* ρ_i *du modèle d'architecture en un ensemble de tâches* Γ_{ρ_i}. *Le temps de réponse de l'ensemble de tâches* Γ_{ρ_i} *est égal au temps de communication effectif du flux* ρ_i.

Ce théorème peut être traduit par l'inéquation suivante :

$$PD_{\rho_i} \leq C_{eff_{\rho_i}} = C_{ECTM_{\rho_i}} \leq C_{\rho_i} \tag{7.1}$$

avec

- PD_{ρ_i} est le temps de trajet du flux ρ_i. Il présente le temps de communication de flux en l'absence de contention de communication dans le NoC.
- $C_{eff_{\rho_i}}$ représente le temps de communication effectif du flux ρ_i.
- $C_{ECTM_{\rho_i}}$ est le temps de réponse de l'ensemble de tâches Γ_{ρ_i}. $(C_{ECTM_{\rho_i}} = C_{\Gamma_{\rho_i}})$
- C_{ρ_i} représente le pire temps de communication du flux ρ_i.

Dans la suite, nous démontrons cette inéquation pour les NoC SAF et Wormhole. Pour le faire, nous considérons deux cas : sans et avec interférences.

7.5.1 Exact Communication Time Model pour SAF

Considérons deux tâches dépendantes τ_1 et τ_2. La tâche τ_1 envoie un flux ρ_i à la tâche τ_2. ρ_i utilise n liens physiques. En appliquant le modèle $ECTM_{SAF}$, le flux ρ_i du modèle d'architecture est transformé en un ensemble de n tâches dans le modèle d'analyse $\Gamma_{\rho_i} = \{ \tau_{\rho_i,1}, \cdots \tau_{\rho_i,n}\}$.

Rappelons que sous cette configuration (tableau 7.3 : utilisation des canaux virtuels avec la technique SAF), nous ne pouvons pas avoir d'interférence indirecte [90]. En outre, nous ne pouvons pas avoir de préemption au niveau flit puisque le niveau de préemption utilisé est le paquet.

7.5.1.1 Sans interférence

Avant de commencer notre démonstration, nous rappelons dans la suite les équations de temps de trajet PD_{ρ_i} pour le flux ρ_i pour une telle configuration du NoC [32] :

$$PD_{\rho_i} = (size_{max}/B_{link} + R_{hop}).n = PD_{Onelink}.n \tag{7.2}$$

où

- $size_{max}$ est la taille maximale du flux ρ_i.
- B_{link} représente la bande passante du lien entre deux routeurs voisins.
- R_{hop} représente la constante de temps de routage pour chaque routeur.
- n est le nombre de saut entre la source et la destination pour le flux ρ_i.

Dans le cas sans interférence, nous avons :

$$C_{ECTM_{\rho_i}} = C_{\Gamma_{\rho_i}} \qquad \text{(par définition de } C_{ECTM_{\rho_i}})$$
$$C_{\Gamma_{\rho_i}} = \sum_{j=0}^{n} C_{\tau_{\rho_i,j}} \qquad \text{(définition 75)}$$

L'ensemble de tâche Γ_{ρ_i} peut être considéré comme un DAG linéaire et par la suite le temps de réponse de l'ensemble Γ_{ρ_i} est égal à la somme de temps d'exécution de toutes les tâches de cet ensemble.

$$
\begin{aligned}
C_{\Gamma_{\rho_i}} &= \sum_{j=0}^{n} C_{\tau_{\rho_i,1}} && (\text{car } C_{\tau_{\rho_i,1}} = C_{\tau_{\rho_i,2}} = \cdots = C_{\tau_{\rho_i,n}}) \\
C_{\Gamma_{\rho_i}} &= \sum_{j=0}^{n} PD_{Onelink} && (\text{règle 4 d'ECTM i.e. } C_{\tau_{\rho_i,1}} = PD_{Onelink}) \\
C_{\Gamma_{\rho_i}} &= n \cdot PD_{Onelink} \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_1} && (\text{voir éq. 7.2})
\end{aligned}
$$

et puisque dans le cas sans interférence, nous avons $PD_{\rho_i} = C_{eff_{\rho_i}} = C_{\rho_i}$
Donc nous pouvons déduire que :

$$PD_{\rho_i} \leq C_{eff_{\rho_i}} = C_{ECTM_{\rho_i}} \leq C_{\rho_i}$$

7.5.1.2 Avec interférences

Dans ce paragraphe nous supposons que le flux ρ_i partage certains liens physiques avec j autres flux, mais qu'il n'interfère dans la réalité qu'avec k flux (avec $k \leq j$).

Nous rappelons dans la suite, les équations du pire temps de communication et de temps de communication effectif pour le flux ρ_i pour une telle configuration du NoC [32] :

$$C_{\rho_i} = PD_{\rho_i} + \sum_{1}^{j} PD_{Onelink} \tag{7.3}$$

$$C_{eff_{\rho_i}} = PD_{\rho_i} + \sum_{1}^{k} PD_{Onelink} \tag{7.4}$$

Dans ce cas, nous avons :

$$
\begin{aligned}
C_{ECTM_{\rho_i}} &= C_{\Gamma_{\rho_i}} && (\text{par définition}) \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_1} + \sum_{x=1}^{k} C_{\tau_x} && (\text{règle 3 d'ECTM})
\end{aligned}
$$

Nous notons ici que l'unité de calcul qui exécute les tâches τ_x est une abstraction du routeur NoC. Il utilise un algorithme d'ordonnancement en accord avec la technique d'arbitrage adoptée par le routeur afin d'assurer que l'intervalle d'exécution de toutes les tâches de l'ensemble Γ_{ρ_i} est égal à l'intervalle de transmission du message correspondant dans le modèle d'architecture.

En considérant la règle 4 d'$ECTM_{SAF}$, nous avons :

$$C_{\Gamma_{\rho_i}} = PD_{\rho_1} + \sum_{x=1}^{k} PD_{Onelink} \qquad \text{(règle 4 d'ECTM i.e. } C_{\tau_{\rho_i,1}} = PD_{Onelink})$$

$$C_{\Gamma_{\rho_i}} = PD_{\rho_1} + k \cdot PD_{Onelink} \leq PD_{\rho_1} + j \cdot PD_{Onelink} \qquad (k \leq j)$$

$$C_{\Gamma_{\rho_i}} = C_{eff_{\rho_i}} \leq C_{\rho_i} \qquad \text{(éq. 7.4 et Eq. 7.3)}$$

D'où l'inéquation 7.1 :

$$PD_{\rho_i} \leq C_{eff_{\rho_i}} = C_{ECTM_{\rho_i}} \leq C_{\rho_i}$$

Pour conclure, ECTM conserve bien les interférences de communication possibles dans le modèle d'analyse produit pour un NoC SAF et il fournit un temps de communication exactement égal au temps de communication effectif du flux ρ_i.

7.5.2 Exact Communication Time Model pour Wormhole

Considérons deux tâches dépendantes τ_1 et τ_2. La tâche τ_1 envoie les messages d'un flux ρ_i à la tâche τ_2. Les messages de ρ_i sont de taille n flits et passent par m liens physiques.

En appliquant le modèle $ECTM_{Wormhole}$, le flux ρ_i est transformé en un ensemble de $(n \cdot m)$ tâches : $\Gamma_{\rho_i} = \{ \tau_{\rho_i,1,1}, \cdots \tau_{\rho_i,n,m} \}$

Rappelons ici que sous cette configuration, nous ne pouvons pas avoir d'interférence indirecte (tableau 7.3 : suite à l'utilisation d'un canal virtuel pour chaque flux) [90].

7.5.2.1 Sans interférence

Dans la suite, nous rappelons l'équation du temps de trajet pour une telle configuration du NoC [32] :

$$PD_{\rho_i} = n.PD_{Oneflit/Onelink} + R_{hop}.(m-1) \qquad (7.5)$$

avec

- $PD_{Oneflit/Onelink}$ représente le temps de communication d'un flit pour un seul lien sans considérer les possibles contentions de communication dans le NoC.
- n représente la taille du flux ρ_i.
- m représente le nombre de liens utilisés par le flux ρ_i.
- R_{hop} représente la constante de temps de routage pour chaque routeur.

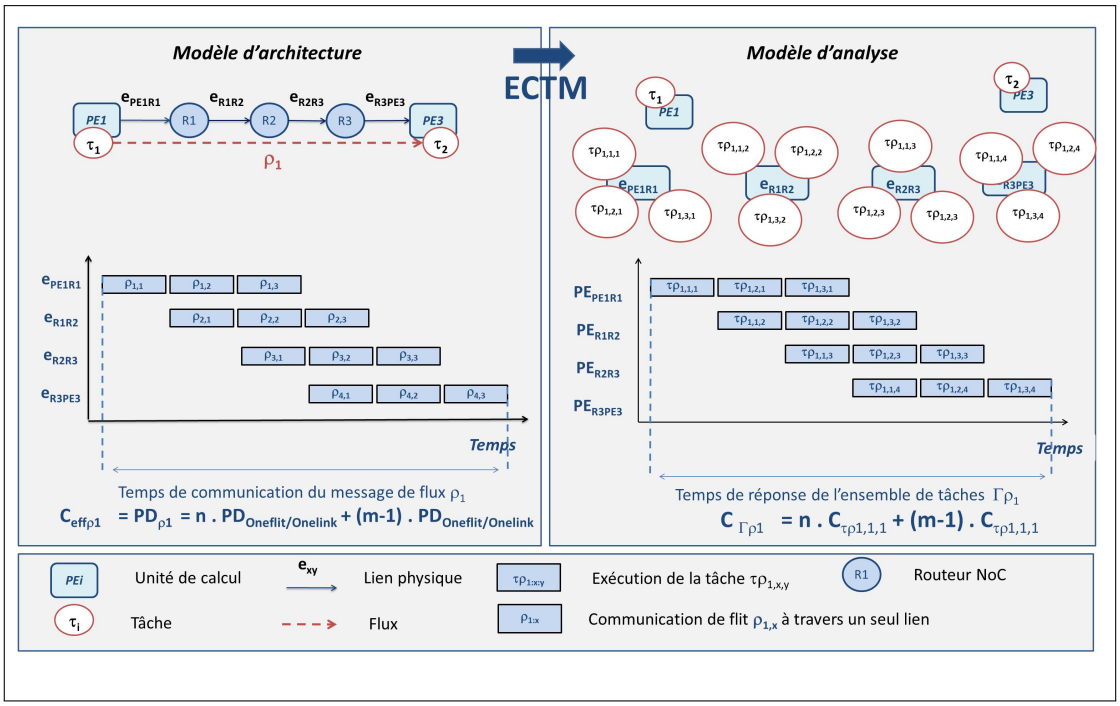


FIGURE 7.10 – $ECTM_{Wormhole}$ - Validation

La figure 7.10 illustre le temps de réponse de l'ensemble des tâches Γ_{ρ_i} dans le cas sans interférence en supposant que $n = 3$ et $m = 4$. Nous supposons également que $R_{hop} = PD_{Oneflit/Onelink}$. Dans la partie gauche de cette figure, nous décrivons le temps de communication au niveau flit pour un message du flux ρ_i tandis que, dans la partie droite, nous exposons le temps d'exécution de chaque tâche de l'ensemble Γ_{ρ_i} qui abstrait le flux ρ_i dans le modèle d'analyse. Dans la partie gauche, nous considérons deux tâches τ_1 et $\tau 2$. τ_1 envoie un flux ρ_1 vers la tâche τ_2. Nous présentons ensuite le temps de communication du flux ρ_1 pour un Wormhole NoC. $C_{eff_{\rho_i}}$ présente le temps de communication du message du flux ρ_1.

Dans le cas sans interférence, nous avons :

$$
\begin{aligned}
C_{ECTM_{\rho_i}} &= C_{\Gamma_{\rho_i}} && \text{(par définition de } C_{ECTM_{\rho_i}}) \\
C_{\Gamma_{\rho_i}} &= n \cdot C_{\tau_{\rho_i,1,1}} + (m-1) \cdot C_{\tau_{\rho_i,1,1}} && \text{(fig. 7.10)} \\
C_{\Gamma_{\rho_i}} &= n \cdot C_{\tau_{\rho_i,1,1}} + (m-1) \cdot R_{hop} && (R_{hop} = PD_{Oneflit/Onelink}) \\
C_{\Gamma_{\rho_i}} &= n \cdot PD_{Oneflit/Onelink} + (m-1) \cdot R_{hop} && (C_{\tau_{\rho_i,1,1}} = PD_{Oneflit/Onelink}) \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_i} && \text{(voir éq. 7.5)}
\end{aligned}
$$

et puisque dans le cas sans interférence, nous avons $PD_{\rho_i} = C_{eff_{\rho_i}} = C_{\rho_i}$

Donc nous pouvons déduire que :

$$PD_{\rho_i} \leq C_{eff_{\rho_i}} = C_{ECTM_{\rho_i}} \leq C_{\rho_i}$$

7.5.2.2 Avec interférences

Dans ce paragraphe nous supposons que le flux ρ_1 partage certains liens physiques avec j autres flux mais dans la réalité, il n'interfère qu'avec k flux (où $k \leq j$).

Dans ce cas, les j autres flux sont transformés en des ensembles de tâches Γ_1, $\cdots$, Γ_j . Ces ensembles de tâches partagent les mêmes supports d'exécution que l'ensemble Γ_{ρ_i}.

Nous rappelons dans la suite les équations du pire temps de communication et de temps de communication effectif pour le flux ρ_i pour une telle configuration du NoC [32] :

$$C_{\rho_i} = PD_{\rho_i} + \sum_{x=1}^{j} PD_{Oneflit/Onelink}.size_{max_x} \tag{7.6}$$

$$C_{eff_{\rho_i}} = PD_{\rho_i} + \sum_{x=1}^{k} PD_{Oneflit/Onelink}.size_{max_x} \tag{7.7}$$

avec $size_{max_x}$ étant la taille maximale de tous les messages du flux ρ_x.

Dans le cas avec interférences, nous avons

$$
\begin{aligned}
C_{ECTM_{\rho_i}} &= C_{\Gamma_{\rho_i}} \\
&\quad \text{(par définition de } C_{ECTM_{\rho_i}}\text{)} \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_1} + \sum_{y=1}^{k} \sum_{x=1}^{size_{max_x}} C_{\tau_{\rho_i,y,x}} \\
&\quad \text{(règle 3 d'ECTM)} \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_1} + \sum_{y=1}^{k} C_{\tau_{\rho_i,y,1}} \cdot size_{max_x} \\
&\quad (C_{\tau_{\rho_i,y,x}} = C_{\tau_{\rho_i,y,1}}) \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_1} + k \cdot C_{\tau_{\rho_i,1,1}} \cdot size_{max_x} \\
&\quad (C_{\tau_{\rho_i,y,1}} = C_{\tau_{\rho_i,1,1}}) \\
C_{\Gamma_{\rho_i}} &= PD_{\rho_1} + k \cdot PD_{Oneflit/Onelink} \cdot size_{max_x} \\
&\quad (C_{\tau_{\rho_i,1,1}} = PD_{Oneflit/Onelink}) \\
C_{\Gamma_{\rho_i}} &= C_{eff_{\rho_i}} \leq PD_{\rho_1} + j \cdot PD_{Oneflit/Onelink} \cdot size_{max_x} \\
&\quad \text{(voir éq. 7.7 ; } k \leq j\text{)} \\
C_{\Gamma_{\rho_i}} &= C_{eff_{\rho_i}} \leq C_{\rho_i} \\
&\quad \text{(voir éq. 7.6)}
\end{aligned}
$$

D'où l'inéquation 7.1 :

$$PD_{\rho_i} \leq C_{eff_{\rho_i}} = C_{ECTM_{\rho_i}} \leq C_{\rho_i}$$

Pour conclure, ECTM conserve bien les possibles interférences de communication de modèle d'architecture dans le modèle analyse pour un NoC Wormhole et il fournit un temps de communication exactement égal au temps de communication effectif du flux ρ_i.

Dans la suite, nous détaillons la mise en œuvre de notre approche. Ensuite, nous présentons les évaluations réalisées et nous discutons des résultats obtenus.

7.6 Implantation

Dans l'objectif d'évaluer les performances de notre approche, nous avons développé et intégré DTFM et les modèles de communication proposés dans l'outil de simulation Cheddar [28].

Nous rappelons que Cheddar est un outil de simulation conçu à l'Université de Bretagne Occidentale. Il nous permet d'appliquer des tests de faisabilité sur un jeu de tâches donné pour un algorithme d'ordonnancement choisi [28].

Pour modéliser l'architecture du système à analyser, Cheddar fournit un langage de conception appelé CheddarADL [28]. CheddarADL permet aux utilisateurs de décrire à la fois les composants logiciels et les composants matériels du système à analyser.

La figure 7.11 illustre l'architecture logicielle de notre prototype et d'un sous-ensemble des bibliothèques de Cheddar. Le module NoC nous permet de générer un jeu de tâches périodiques dépendantes déployées sur un NoC. Nous utilisons également les modules DTFM, WCCTM, $ECTM_{SAF}$ et $ECTM_{Wormhole}$ pour effectuer nos transformations de modèles. Le module HLFET est utilisé pour effectuer une simulation de l'ordonnancement sur un intervalle de faisabilité donné.

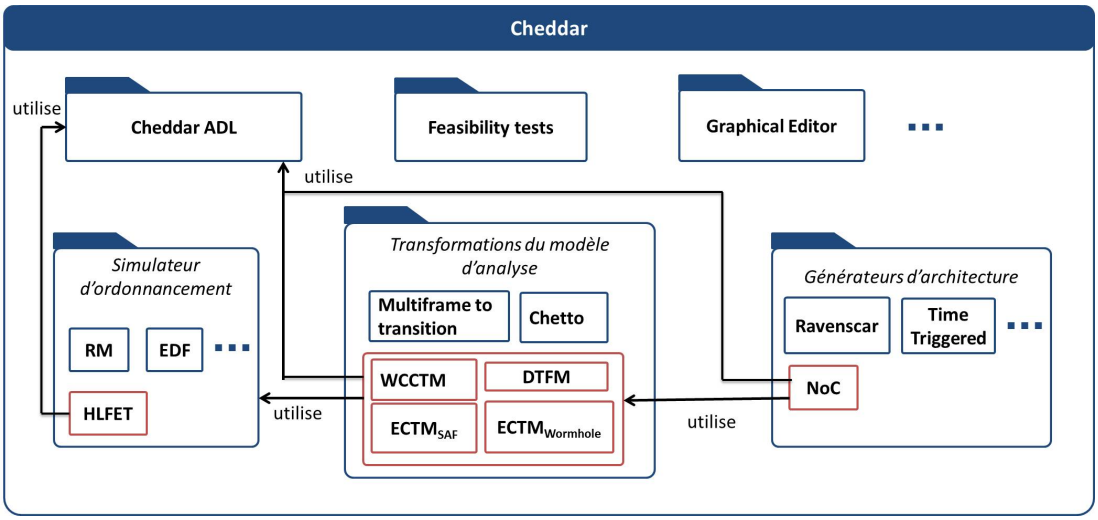


FIGURE 7.11 – Implantation de DTFM, ECTM et WCCTM dans Cheddar

7.7 Évaluations

Dans cette partie, nous évaluons les performances de notre approche pour plusieurs jeux de tâches et plusieurs configurations de NoC. Cette évaluation comprend une étude comparative entre les différents modèles de communication proposés en se basant sur plusieurs métriques comme le temps de calcul nécessaire à la transformation et le taux d'ordonnançabilité atteint.

Dans cette partie et pour chaque évaluation, nous appliquons la même démarche :

1. Nous générons un jeu de tâches dépendantes périodiques déployées sur un NoC.

2. Nous appliquons DTFM au jeu de tâches généré dans l'objectif de calculer le modèle de flux.
3. Nous appliquons les modèles de communications ECTM et WCCTM afin de calculer le modèle d'analyse.
4. Nous étudions l'ordonnançabilité du système en utilisant la simulation de l'ordonnancement sur l'intervalle de faisabilité. Nous calculons cet intervalle en nous basant sur Goossens et al [31].

   L'algorithme d'ordonnancement considéré dans notre évaluation est l'algorithme HLFET.

7.7.1 Évaluation du taux d'ordonnançabilité

Dans cette partie, nous évaluons le taux d'ordonnaçabilité généré par notre approche.

Nous avons utilisé dans cette évaluation deux types de trafic : All-To-One et One-To-One.

7.7.1.1 Trafic All-To-One

Dans cette expérimentation, nous générons aléatoirement un ensemble de tâches périodiques dépendantes en utilisant Uunifast [93].

Nous affectons ces tâches sur un NoC 4×4 de façon à avoir au maximum 2 tâches par unité de calcul. Le modèle de trafic généré est All-To-One. La taille du flux est égale à 4 flits.

Nous utilisons dans cette expérimentation deux NoC : Un NoC Wormhole et un NoC SAF.

Nous lançons l'expérimentation en appliquant DTFM, les modèles de communication et l'algorithme HLFET qui sont intégrés dans le simulateur Cheddar[28].

La figure 7.12 présente le taux d'ordonnançabilité en fonction de taux d'utilisation des unités de calcul. Elle présente les résultats des modèles de communications pour les deux NoC considérés.

La figure 7.12 montre que le modèle WCCTM est pessimiste par rapport à ECTM. En utilisant $WCCTM_{SAF}$, le système n'est plus ordonnançable à partir d'un taux égale à 0.15. Cependant avec le modèle $ECTM_{SAF}$, les modèles sont tous ordonnaçables pour un taux d'utilisation des unités de calcul de 0.2. Pour conclure, en termes de vérification d'ordonnançabilité, $ECTM_{SAF}$ améliore l'ordonnançabilité de 30% vis à $WCCTM_{SAF}$.

Cette figure confirme également l'efficacité de la technique de Wormhole face à la technique SAF.

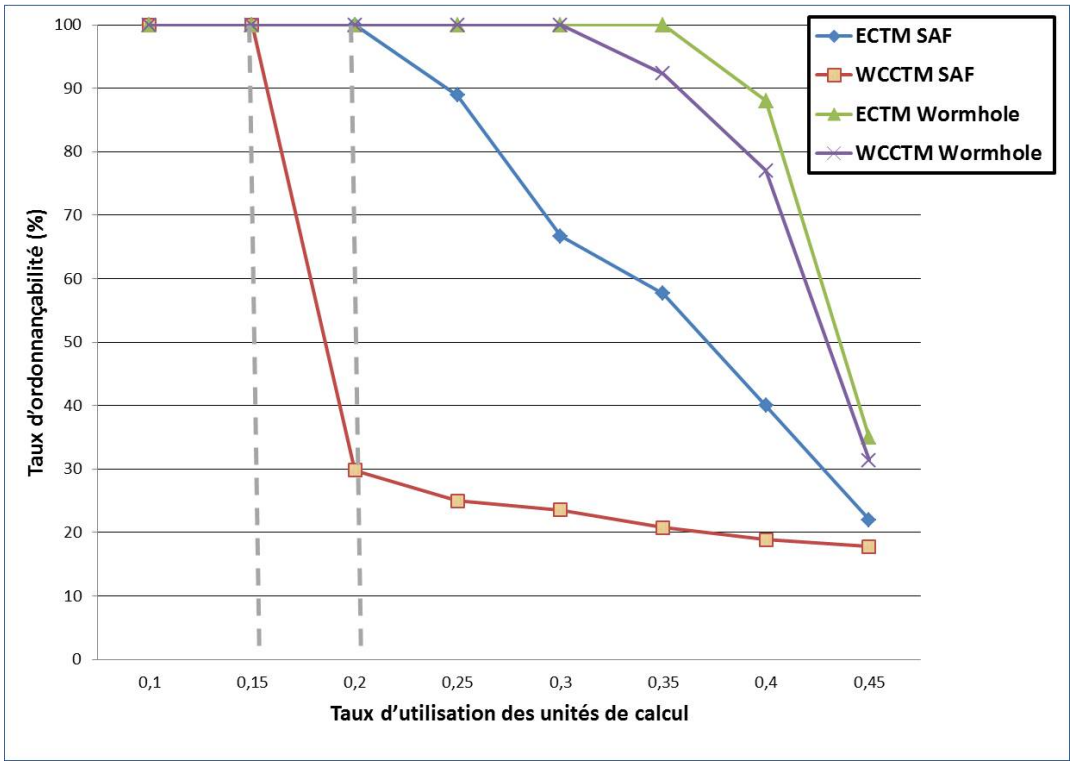


FIGURE 7.12 – Taux d'ordonnançabilité pour le modèle All-To-One

7.7.1.2 Trafic One-To-One

Dans cette expérimentation, nous considérons le modèle de trafic One-To-One. Par ailleurs, nous gardons la même configuration que l'expérimentation précédente.

La figure 7.13 présente le taux d'ordonnançabilité des systèmes en fonction du taux d'utilisation des unités de calcul. Elle présente les résultats des modèles de communications pour les deux NoC considérés.

La figure 7.13 montre que WCCTM est pessimiste par rapport à ECTM. En utilisant $WCCTM_{Wormhole}$, le système n'est plus ordonnançable à partir d'un taux égale à 0.08. Cependant avec le modèle $ECTM_{Wormhole}$, les modèles sont tous ordonnançables jusqu'à 0.16 du taux d'utilisation des unités de calcul. Donc, en termes de vérification d'ordonnançabilité, $ECTM_{Wormhole}$ améliore l'ordonnançabilité de 50% vis à vis $WCCTM_{Wormhole}$.

7.7.2 Évaluation du temps de calcul

L'objectif de cette évaluation est d'étudier le passage à l'échelle des modèles de communications proposés. Nous avons mesuré le temps de calcul du modèle d'analyse en appliquant les modèles de communication WCCTM et ECTM. Dans

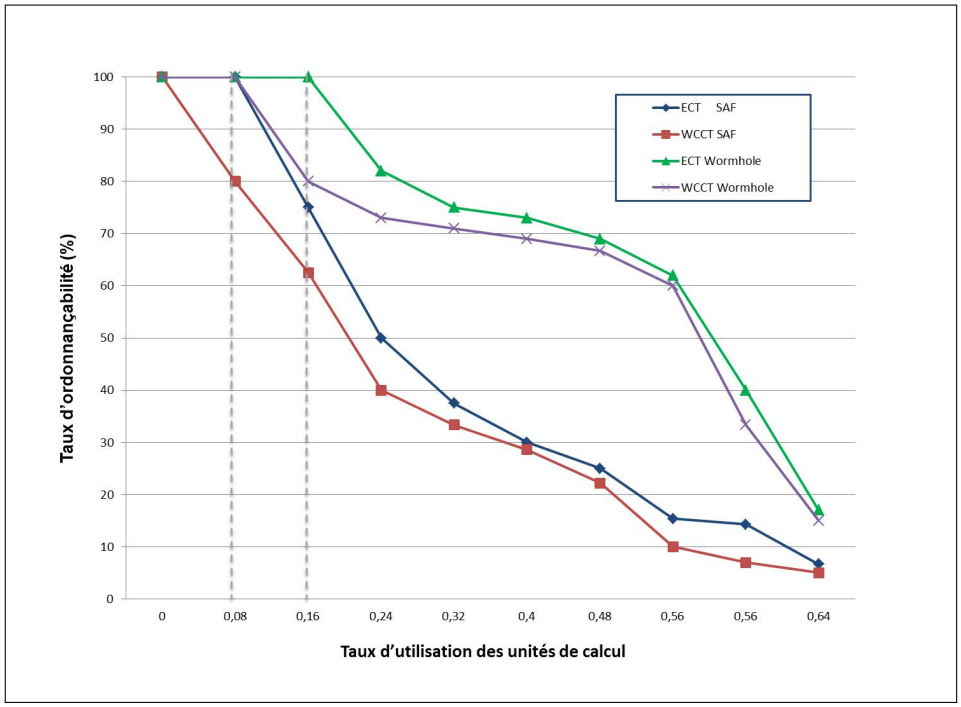


FIGURE 7.13 – Taux d'ordonnançabilité pour le modèle One-To-One

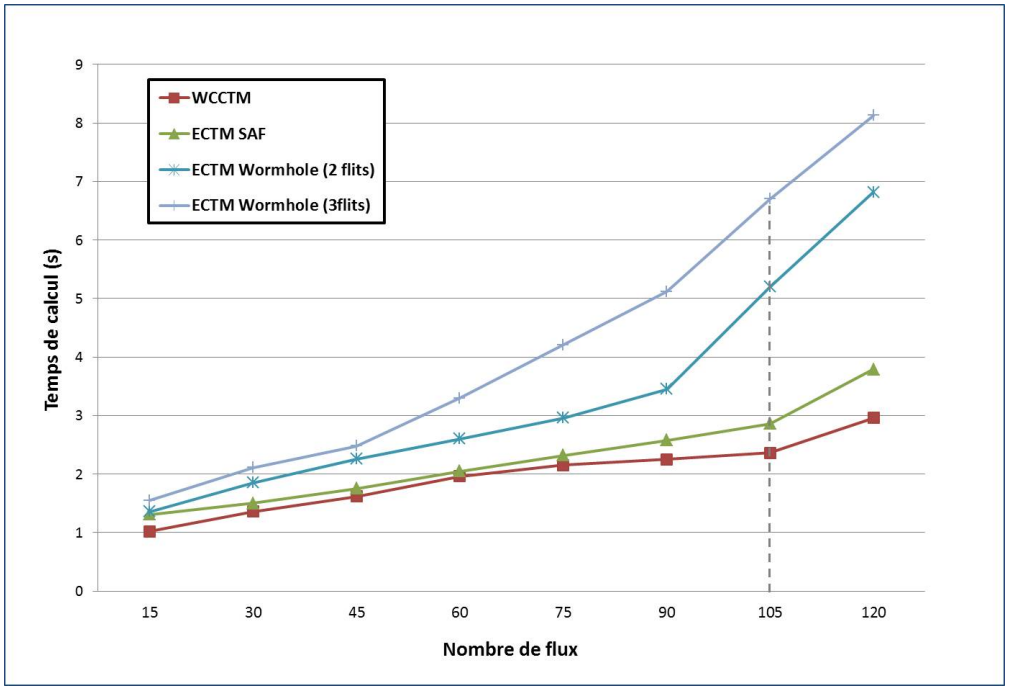


FIGURE 7.14 – Temps de calcul pour les modèles de communications

cette évaluation, nous avons gardé la même configuration que pour les expériences précédentes.

La figure 7.14 présente le temps de calcul pour construire le modèle d'analyse en fonction du nombre de flux. Le nombre de flux est compris entre 15 et 120.

WCCTM nécessite un temps de calcul plus court qu'ECTM. Pour 105 flux, WCCTM prend 2,32 secondes pour calculer le modèle d'analys tandis qu'$ECTM_{SAF}$ prend 2,86 secondes et $ECTM_{Wormhole}$ prend 6,80 secondes pour une taille de deux flits et 8,13 secondes pour une taille de 3 flits.

En termes de temps de calcul, WCCTM est donc plus rapide que ECTM, avec une réduction de 17% pour les NoC SAF et de 54% pour les NoC Wormhole.

7.8 Conclusion

L'analyse d'ordonnancement des systèmes à criticité mixte sur des architectures NoC présente un défit complexe. D'une part, la majorité des modèles de tâches et des modèles de flux ne peuvent pas modéliser un système temps réel déployé sur un NoC. D'autre part, la plupart des techniques d'analyse d'ordonnancement existantes négligent l'analyse des communications et leur impact sur l'exécution des tâches.

Afin de résoudre ces problématiques, nous avons proposé dans un premier temps un modèle de tâches et de flux appelé Dual Task and Flow Model (DTFM). Ce modèle est capable de modéliser un système temps réel exécuté sur un NoC tout en tenant compte des communications, de la configuration du NoC, de l'allocation des tâches et des différents conflits possibles à cause du partage de ressources.

Dans un second temps, nous avons formalisé le modèle de communication Worst Case Communication Time Model (WCCTM). Nous avons alors proposé un nouveau modèle de communication pour les architectures NoC : Exact Communication TIME Model (ECTM).

Notre approche consiste à combiner DTFM et les modèles de communication proposés dans l'objectif de vérifier l'ordonnançabilité des systèmes temps réel déployés sur des architectures NoC.

Nous avons développé et intégré DTFM, WCCTM et ECTM dans le simulateur Cheddar [28]. Ensuite, nous avons évalué l'exactitude et le passage à l'échelle pour ECTM. Les résultats ont montré qu'ECTM est plus précis dans la vérification d'ordonnançabilité par rapport à WCCTM. Pour un NoC SAF et en termes de vérification d'ordonnançabilité, $ECTM_{SAF}$ améliore l'ordonnançabilité de 30% vis à vis $WCCTM_{SAF}$. Pour un NoC Wormhole, $ECTM_{Wormhole}$ améliore l'ordonnançabilité de 50% vis à vis de $WCCTM_{Wormhole}$. Par contre, en termes de

temps de calcul, WCCTM est plus rapide qu'ECTM, avec une réduction de 17% pour les NoCs SAF et de 54% pour les NoCs Wormhole.

Notre approche supporte les NoC Wormhole et les NoC SAF. Il peut être utilisé pour analyser l'ordonnancement des systèmes temps réel et des systèmes à criticité mixte déployés sur des NoC qui respectent les hypothèses prises dans ce travail (les hypothèses 2, 6, 7 et celles du tableau 7.3). Par contre, ECTM ne peut pas être directement appliqué à DAS car ce dernier utilise deux techniques de commutation simultanément. L'adaptation d'ECTM au routeur DAS est l'un de nos futurs travaux dans la continuité de cette thèse.

8

Conclusion et perspectives

Conclusion

Dans le cadre de cette thèse, nous nous intéressons au challenge consistant à exécuter des systèmes à criticité mixte sur des architectures NoC. L'intégration des systèmes à criticité mixte sur des architectures NoC présente une solution prometteuse en termes de consommation d'énergie, de performance de calcul, de poids et de surface.

Plusieurs obstacles peuvent être rencontrés lors du déploiement des systèmes à criticité mixte sur un NoC. Premièrement, les architectures des routeurs NoC n'assurent pas les contraintes temporelles pour les communications critiques en limitant leur réservation des ressources. Deuxièmement, les modèles de tâches et les modèles de flux ne sont pas complets. Les modèles de tâches existants ne prennent pas en compte les communications à travers le NoC, tandis que les modèles de flux existants ignorent l'ordonnancement des tâches. Pour finir, les techniques d'analyse d'ordonnancement temps réel classiques ignorent l'analyse des communications au sein du NoC.

Afin de résoudre ces problématiques et de rendre cette intégration possible, nous avons proposé plusieurs contributions sous la forme d'un routeur, de modèles de tâches, de flux et de communications pour les NoC :

DAS, un routeur pour les systèmes à criticité mixte

DAS est un routeur conçu pour exécuter des systèmes à criticité mixte. Il supporte deux niveaux de criticité et il fonctionne sous deux modes de criticité. Pour cela, il intègre plusieurs canaux virtuels avec deux étages d'arbitrages. En outre, il

combine deux techniques de commutation (SAF et Wormhole) avec deux niveaux de préemption (paquet et flit).

Nous avons fourni dans le cadre de cette thèse une évaluation de DAS sur plusieurs niveaux d'abstraction : niveau circuit, niveau TLM et niveau système.

- Au niveau circuit, nous avons développé DAS en Verilog HDL. Une étude comparative avec les routeurs classiques a montré que DAS est plus coûteux en surface par rapport à VC-router dans une proportion de 2.5%. Ce pourcentage paraît raisonnable en considérant les avantages de DAS devant le routeur VC-router.
- Au niveau TLM, nous avons développé DAS en SystemC. Ensuite, nous avons intégré DAS dans un simulateur de NoC (SHOC). Nous avons comparé les temps de communication fournis par DAS avec ceux des routeurs classiques en faisant varier plusieurs paramètres comme l'état du réseau, la taille et la criticité des paquets. Cette évaluation a montré que DAS fournit non seulement le meilleur temps de communication mais aussi le taux d'utilisation de ressources le plus élevé en comparaison avec WNoC-Router et VC-Router. Cependant, il perd son efficacité face à WNoC-Router pour des messages de taille importante (>6 flits).
- Au niveau système, nous avons modélisé DAS en utilisant le langage IF [88]. Nous avons validé le comportement de DAS par model-checking en utilisant IF et ses outils. Cette évaluation nous a permis de valider formellement 4 propriétés de DAS.

DTFM, un modèle de tâches et de flux

DTFM est un modèle de tâches et de flux conçu pour modéliser les systèmes temps réel déployés sur une architecture NoC. Il complète les modèles classiques de tâches et de flux. DTFM calcule le modèle de flux correspondant au modèle de tâches, du modèle de NoC et du placement des tâches.

Dans le calcul du modèle de flux, DTFM intègre les analyses de temps de communication existant en tenant compte de la configuration du NoC. Il supporte plusieurs configurations telles que Wormhole et SAF. Nous avons intégré DTFM dans le simulateur d'ordonnancement Cheddar [28].

ECTM, un modèle de communication pour les architectures NoC

L'objectif de ce modèle est de prédire le temps de communication sur des architectures NoC. ECTM abstrait les communications par un DAG tout en prenant en compte le modèle de tâches et le modèle du NoC utilisés.

Nous avons intégré ce modèle de communication dans le simulateur d'ordonnancement Cheddar. Puis, nous avons évalué l'exactitude et le passage à l'échelle pour ECTM. Les résultats d'évaluation ont montré qu'ECTM est plus précis dans la vérification d'ordonnaçabilité par rapport à WCCTM avec une amélioration de 30% pour les NoC SAF et de 100% pour les NoC Wormhole. Par contre, en termes de temps de calcul, ECTM est moins rapide que WCCTM, avec une augmentation du temps de calcul de 17% pour les NoC SAF et de 54% pour les NoC Wormhole.

Perspectives

Nos travaux futurs vont porter principalement sur l'ajout d'un nouveau mode de criticité pour DAS, de la gestion du nombre de canaux utilisés dans DAS et de l'adaptation du modèle de communication ECTM proposé avec les spécifications de DAS. Nous détaillons dans la suite ces travaux.

DAS, un nouveau mode de criticité pour les communications non critiques

DAS supporte deux modes de criticité : NORMAL et DEGRADED. Dans le mode NORMAL, DAS garantit l'ordonnancement des communications critiques et non critiques tandis que dans le mode DEGRADED, le respect des contraintes temporelles pour les communications non critiques n'est plus garanti. Sous ces deux modes de criticité, DAS limite la réservation des ressources pour les flux critiques dans l'objectif d'assurer un partage de ressources dans le NoC qui respecte les exigences des systèmes à criticité mixte.

Le monitoring présente un moyen possible pour encore améliorer le partage de ressources dans le NoC. Pour cela, nous proposons d'ajouter un autre mode de criticité pour DAS. Dans ce nouveau mode, les communications non critiques utilisent les ressources non utilisées par les communications critiques. L'objectif de ce mode est de minimiser le gaspillage de ressources par les communications critiques tout en assurant leurs contraintes temporelles. Pour le faire, nous proposons d'intégrer des moniteurs dans l'architecture de DAS. Le rôle de ces moniteurs est de contrôler l'utilisation des ports d'entrée/sortie du routeur et de gérer l'affectation des canaux virtuels aux communications [95].

DAS, réduire le nombre des canaux virtuels

DAS implante plusieurs canaux virtuels. Le nombre de ces canaux dépend essentiellement du placement des tâches et du nombre de communications cri-

tiques. Gérer le nombre des canaux virtuels utilisés présente un autre point d'amélioration possible pour DAS. Réduire le nombre de canaux virtuels utilisés dans DAS peut conduire à un gain significatif en termes de coût en surface.

Pour cela, nous proposons d'adapter un des algorithmes de placement multi-critères avec nos objectifs [84, 83]. Les objectifs considérés dans notre approche sont principalement : la réduction du nombre de communications critiques qui partagent les mêmes liens physiques et la réduction des distances entre les tâches sources et les tâches destinations.

ECTM, conforme aux spécifications de DAS

Le troisième axe de nos futurs travaux consiste à adapter les transformations d'ECTM pour DAS. DAS supporte actuellement deux niveaux de criticité grâce à l'utilisation de deux techniques de commutation, deux niveaux de préemption et deux étages d'arbitrage.

Le modèle de communication proposé par ECTM supporte séparément les deux techniques de commutation SAF et Wormhole. Par contre, ECTM ne peut pas supporter la combinaison de ces deux techniques. Ainsi, ECTM n'est pas capable d'analyser l'ordonnancement d'un système à criticité mixte déployé sur DAS.

Afin d'adapter ECTM pour DAS, nous devons intégrer les spécifications de DAS dans les transformations d'ECTM. En d'autres termes, ECTM doit considérer simultanément dans ses transformations deux niveaux de préemption, deux étages d'arbitrage et deux techniques de commutation pour chaque routeur.

Bibliographie

[1] J. A. Stankovic. Misconceptions about real-time computing : a serious problem for next-generation systems. *Computer*, 21(10) :10–19, Oct 1988.

[2] Vincent Legout. *Energy-aware real-time scheduling of multiprocessor embedded systems*. Theses, Télécom ParisTech, April 2014.

[3] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the* 28^{th} *International Real-Time Systems Symposium (RTSS)*, pages 239–243. IEEE, Dec 2007.

[4] Haohan Li and Sanjoy Baruah. Load-based schedulability analysis of certifiable mixed-criticality systems. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '10, pages 99–108, New York, NY, USA, 2010. ACM.

[5] Sheng MA, Libo Huang, Mingche Lai, and Wei Shi. *NETWORKS-ON-CHIP From implementation to programming paradigms*. Morgan Kaufmann, 2014.

[6] Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6) :82 :1–82 :37, November 2017.

[7] Alan Burns and Robert Davis. Mixed criticality systems-a review, 9^{th} ed. Technical report, Department of Computer Science, University of York, Jan 2017. http://www-users.cs.york.ac.uk/burns/review.pdf.

[8] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pages 161–170, April 2008.

[9] Navonil Chatterjee, Suraj Paul, and Santanu Chattopadhyay. Task mapping and scheduling for network-on-chip based multi-core platform with transient faults. *Journal of Systems Architecture*, 83 :34 – 56, 2018.

[10] Ruxandra Pop and Shashi Kumar. A survey of techniques for mapping and scheduling applications to network on chip systems. Technical Report 04 :4, Department of Electronics and Computer Engineering School of Engineering, Jönköping University, 2004.

[11] A. Burns, J. Harbin, and L. S. Indrusiak. A wormhole noc protocol for mixed criticality systems. In *2014 IEEE Real-Time Systems Symposium*, pages 184–195, Dec 2014.

[12] Krunal Jetly. *Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA's.* PhD thesis, University of Windsor, Nov 2013.

[13] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems : Predictable Scheduling Algorithms and Applications.* Springer Publishing Company, Incorporated, 3rd edition, 2011.

[14] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3) :181–194, Sep 1990.

[15] M. Spuri and J. A. Stankovic. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers*, 43(12) :1407–1412, Dec 1994.

[16] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4) :406–471, 1999.

[17] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, pages 250–256, 2000.

[18] A. Gamati, C. Brunette, R. Delamare, T. Gautier, and J. Talpin. A modeling paradigm for integrated modular avionics design. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pages 134–143, Aug 2006.

[19] Abdoulaye Gamatié and Thierry Gautier. Synchronous Modeling of Modular Avionics Architectures using the SIGNAL Language. Research Report RR-4678, INRIA, 2002.

[20] Abdoulaye Gamatié, Thierry Gautier, Paul Le Guernic, and Jean-pierre Talpin. Polychronous design of embedded real-time applications. *ACM Transaction Software Engineering and Methodology*, 16, 04 2007.

[21] R. Ernst and M. Di Natale. Mixed criticality systems a history of misconceptions ? *IEEE Design Test*, 33(5) :65–74, Oct 2016.

[22] A. Burns. System mode changes - general and criticality-based. In L. Cucu-Grosjean and R. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, pages 3–8, 2014.

[23] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1) :46–61, January 1973.

[24] S. Lauzac, R. Melhem, and D. Mosse. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *Proceeding. 10th EUROMICRO Workshop on Real-Time Systems (Cat. No.98EX168)*, pages 188–195, June 1998.

[25] J. A. Stankovic, M. Spuri, M. Di Natale, and G. C. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6) :16–25, June 1995.

[26] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 63–72, Dec 2012.

[27] Navet Nicolas. *Évaluation de performances temporelles et optimisation de l'ordonnancement de tâches et messages.* Theses, Université de Lorraine, Institut National Polytechnique de Lorraine, November 1999.

[28] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a flexible real-time scheduling framework. *ACM SIGAda Ada Letters*, 24(4) :1–8, Dec 2004.

[29] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano. Mast : Modeling and analysis suite for real time applications. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 125–134, June 2001.

[30] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. SimSo : A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. In *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, page 6 p., Madrid, Spain, July 2014.

[31] Joël Goossens, Emmanuel Grolleau, and Liliana Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-Time Systems*, 52(6) :808–832, Nov 2016.

[32] Zheng Shi. *Real-Time Communication Services for Networks on Chip*. PhD thesis, University of York, Nov 2009.

[33] Thomas L. Adam, K. Mani Chandy, and J. R. Dickson. A comparison of list schedules for parallel processing systems. *Commun. ACM*, 17 :685–690, 1974.

[34] Tarek Hagras and J. J. Janecek. Static vs. dynamic list-scheduling performance comparison. 2003.

[35] L. S. Indrusiak, A. Burns, and B. Nikolić. Buffer-aware bounds to multi-point progressive blocking in priority-preemptive nocs. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 219–224, March 2018.

[36] Pradip Kumar Sahu and Santanu Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1) :60 – 76, 2013.

[37] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Mapping hard real-time applications on many-core processors. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 235–244, New York, NY, USA, 2016. ACM.

[38] Zheng Shi and Alan Burns. Improvement of schedulability analysis with a priority share policy in on-chip networks. In *17th International Conference on Real-Time and Network Systems RTNS'2009*, 26-27 October, 2009.

[39] Frederic Giroudot and Ahlem Mifdaoui. Tightness and computation assessment of worst-case delay bounds in wormhole networks-on-chip. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, RTNS '19, page 19–29, New York, NY, USA, 2019. Association for Computing Machinery.

[40] Girish Varatkar and Radu Marculescu. Communication-aware task scheduling and voltage selection for total systems energy minimization. In *Proceedings of the 2003 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '03, pages 510–, Washington, DC, USA, 2003. IEEE Computer Society.

[41] Tang Lei and Shashi Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the Euromicro Symposium on Digital Systems Design*, DSD '03, pages 180–, Washington, DC, USA, 2003. IEEE Computer Society.

[42] Z. Stamenkovic. System-on-chip design : Engineering or art. In *2006 25th International Conference on Microelectronics*, pages 372–379, May 2006.

[43] L. Benini and G. De Micheli. Networks on chips : a new soc paradigm. *Computer*, 35(1) :70–78, Jan 2002.

[44] J. Flich, T. Skeie, A. Mejia, O. Lysne, P. Lopez, A. Robles, J. Duato, M. Koibuchi, T. Rokicki, and J. C. Sancho. A survey and evaluation of topology-agnostic deterministic routing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 23(3) :405–425, March 2012.

[45] Lounis Zerioul. *Behavioral modelling of a network on chip based on RF interconnects*. Theses, Université de Cergy Pontoise (UCP), September 2015.

[46] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proceedings - Computers and Digital Techniques*, 150(5) :294–302–, Sept 2003.

[47] Samuel Evain. *μSpider Environnement de Conception de Réseau sur Puce*. Theses, INSA de Rennes, November 2006.

[48] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. F. Brown III, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27(5) :15–31, Sept 2007.

[49] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger. On-chip interconnection networks of the trips chip. *IEEE Micro*, 27(5) :41–50, Sept 2007.

[50] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 43(1) :29–41, Jan 2008.

[51] A. Lines. Asynchronous interconnect for synchronous soc design. *IEEE Micro*, 24(1) :32–41, Jan 2004.

[52] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Qnoc : Qos architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2) :105 – 128, 2004. Special issue on networks on chip.

[53] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 120–129, Aug 2008.

[54] A. Hansson, M. Subburaman, and K. Goossens. Aelite : A flit-synchronous network on chip with composable and predictable services. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 250–255, April 2009.

[55] Zheng Shi and Alan Burns. Schedulability analysis and task mapping for real-time on-chip communication. *Real-Time Systems*, 46(3) :360–385, Dec 2010.

[56] Y. Qian, Z. Lu, and W. Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pages 44–53, May 2009.

[57] Zheng Shi and Alan Burns. Real time communication analysis for on-chip networks with wormhole switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 161–170, Nov 2008.

[58] Q. Xiong, F. Wu, Z. Lu, and C. Xie. Extending real-time analysis for wormhole nocs. *IEEE Transactions on Computers*, 66(9) :1532–1546, Sept 2017.

[59] Zheng Shi and Alan Burns. Real-time communication analysis with a priority share policy in on-chip networks. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, pages 3–12, July 2009.

[60] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proceedings - Computers and Digital Techniques*, 150(5) :294–302–, Sept 2003.

[61] D. Wiklund and Dake Liu. Socbus : switched network on chip for hard real time embedded systems. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 8 pp.–, April 2003.

[62] S. Penolazzi and A. Jantsch. A high level power model for the nostrum noc. In *9th EUROMICRO Conference on Digital System Design (DSD'06)*, pages 673–676, 2006.

[63] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 890–895 Vol.2, Feb 2004.

[64] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *Design, Automation and Test in Europe*, pages 1226–1231 Vol. 2, March 2005.

[65] I. M. Panades, A. Greiner, and A. Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the gals approach. In *2006 1st International Conference on Nano-Networks and Workshops*, pages 1–5, Sept 2006.

[66] C. Schuck, S. Lamparth, and J. Becker. artnoc - a novel multi-functional router architecture for organic computing. In *2007 International Conference on Field Programmable Logic and Applications*, pages 371–376, Aug 2007.

[67] E. Beigne, F. Clermidy, H. Lhermet, S. Miermont, Y. Thonnart, X. T. Tran, A. Valentian, D. Varreau, P. Vivet, X. Popon, and H. Lebreton. An asynchronous power aware and adaptive noc based circuit. *IEEE Journal of Solid-State Circuits*, 44(4) :1167–1177, April 2009.

[68] T. Marescaux and H. Corporaal. Introducing the supergt network-on-chip ; supergt qos : more than just gt. In *2007 44th ACM/IEEE Design Automation Conference*, pages 116–121, June 2007.

[69] Samuel Evain, Jean-Philippe Diguet, and Dominique Houzet. Noc design flow for tdma and qos management in a gals context. *EURASIP Journal on Embedded Systems*, 2006(1) :063656, Oct 2006.

[70] S. Evain, J. P. Diguet, and D. Houzet. muspider : a cad tool for efficient noc design. In *Proceedings Norchip Conference, 2004.*, pages 218–221, Nov 2004.

[71] L. S. Indrusiak, J. Harbin, and A. Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 47–56, July 2015.

[72] Hamidreza Ahmadian and Roman Obermaisser. Time-triggered extension layer for on-chip network interfaces in mixed-criticality systems. *2015 Euromicro Conference on Digital System Design*, pages 693–699, 2015.

[73] S. Tobuschat and R. Ernst. Efficient latency guarantees for mixed-criticality networks-on-chip. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 113–122, April 2017.

[74] T. Hollstein, S. P. Azad, T. Kogge, and B. Niazmand. Mixed-criticality noc partitioning based on the nocdepend dependability technique. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, June 2015.

[75] S. Avramenko, S. P. Azad, S. Esposito, B. Niazmand, M. Violante, J. Raik, and M. Jenihhin. Qosinnoc : Analysis of qos-aware noc architectures for mixed-criticality applications. In *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 67–72, April 2018.

[76] Olivier Cros. *Mixed criticality management into real-time and embedded network architectures : application to switched ethernet networks*. Theses, Université Paris-Est, December 2016.

[77] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany. Survey on real-time networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5) :1500–1517, May 2017.

[78] Alan Burns. System mode changes - general and criticality-based. In *Proceedings of the 2nd workshop on Mixed Criticality Systems (WMC)*, pages 3–8, Dec 2014.

[79] R. De Andrade, K. N. Hodel, J. F. Justo, A. M. Laganá, M. M. Santos, and Z. Gu. Analytical and experimental performance evaluations of can-fd bus. *IEEE Access*, 6 :21287–21295, 2018.

[80] M. S. Santos and R. d'Amore. Error detection method for the arinc429 communication. In *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pages 1–6, March 2018.

[81] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The ROSACE case study : from simulink specification to multi/many-core execution. In *Proceedings of the* 20^{th} *International Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 309–318. IEEE, April 2014.

[82] Peter H Feiler and David P Gluch. *Model-Based Engineering with AADL : An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[83] Mohammad Abdullah Al Faruque and Jorg Henkel. Minimizing virtual channel buffer for routers in on-chip communication architectures. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pages 1238–1243, Mar 2008.

[84] Mohammad Abdullah Al Faruque and Jorg Henkel. Transaction specific virtual channel allocation in QoS supported on-chip communication. In *Proceedings of the IEEE International Conf on Application-specific Systems, Architectures and Processors (ASAP)*, pages 48–53, July 2007.

[85] Mourad Dridi, Mounir Lallali, Stéphane Rubini, MJ Sepúlveda, Frank Singhoff, and Jean-Philippe Diguet. Modeling and validation of a mixed-criticality noc router using the if language. In *10th International Workshop on Network on Chip Architectures (NoCArc)*, Oct 2017.

[86] Ieee standard for verilog hardware description language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pages 1–560, 2006.

[87] Martha Johanna Sepúlveda, Marius Strum, and Wang Jiang Chau. Performance impact of QoSS (quality-of-security-service) inclusion for NoC-based systems. In *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 12–14, Oct 2009.

[88] Marius Bozga, Susanne Graf, and Laurent Mounier. IF-2.0 : A Validation Environment for Component-Based Real-Time Systems. In *Proceedings of CAV'02*, pages 343–348, London, UK, 2002. Springer-Verlag.

[89] Ieee standard verilog hardware description language. *IEEE Std 1364-2001*, pages 1–856, 2001.

[90] Mourad Dridi, Stéphane Rubini, Mounir Lallali, Martha Johanna Sepúlveda Flórez, Frank Singhoff, and Jean-Philippe Diguet. Design and multi-abstraction-level evaluation of a noc router for mixed-criticality real-time systems. *J. Emerg. Technol. Comput. Syst.*, 15(1) :2 :1–2 :37, February 2019.

[91] Jeffrey Lin Steven Elzinga and Vinita Singhal. Design tips for hdl implementation of arithmetic functions. Technical report, XILINX, June 28 2000. https://www.xilinx.com/support/documentation/application_notes/xapp215.pdf.

[92] Design compiler user guide. Technical report, Synopsys, June 2010. http://eclass.uth.gr/eclass/modules/document/index.php?course=MHX303&download=/5346dc69nktr/5346dc86FWh3.pdf.

[93] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2) :129–154, 2005.

[94] J. J. G. Garcia, J. C. P. Gutierrez, and M. G. Harbour. Schedulability analysis of distributed hard real-time systems with multiple-event synchronization. In *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*, pages 15–24, June 2000.

[95] José Rufino, António Casimiro, Antónia Lopes, Frank Singhoff, Stéphane Rubini, Valérie-Anne Nicolas, Mounir Lallali, Mourad Dridi, Jalil Boukhobza, and Lyes Allache. North-non-intrusive observation and runtime verification of cyber-physical systems. *Ada User Journal*, 39(4), 2018.

Titre : Vers le support des systèmes à criticité mixte sur des architectures *NoC*

Mots clés : Réseau sur puce, analyse de temps de communication, système à criticité mixte.

Résumé :

Nous nous intéressons dans le cadre de ce travail au challenge consistant à intégrer des systèmes à criticité mixte sur des architectures NoC. Cette intégration exige l'assurance des contraintes temporelles pour les applications critiques tout en minimisant l'impact de partage de ressources sur les applications non critiques.

Afin d'exécuter des systèmes à criticité mixte sur des architectures NoC, nous avons proposé plusieurs contributions sous la forme d'un routeur, de modèles de tâches et de communications pour les architectures NoC.

Nous avons proposé DAS, un routeur NoC conçu pour exécuter des systèmes à criticité mixte sur des architectures NoC. Il assure les contraintes temporelles pour les communications critiques tout en limitant la réservation des ressources pour les communications non critiques.

DAS implante deux modes de fonctionnement, deux niveaux de préemption, deux techniques de contrôle de flux et deux étages d'arbitrage. Nous avons implanté DAS dans un simulateur de NoC appelé SHoC. Ensuite, DAS a été evalué sur plusieurs niveaux d'abstraction et selon plusieurs critères.

Nous avons ensuite proposé DTFM : un modèle de tâche et de flux pour les systèmes temps réel déployés sur un NoC. À partir du modèle de tâches, du modèle de NoC et du placement des tâches, DTFM calcule automatiquement le modèle de flux correspondant.

Finalement, nous avons proposé ECTM : un modèle de communications pour les architectures NoC. ECTM conduit à une analyse d'ordonnancement efficace. Il modélise les communications sous la forme d'un graphe de tâches tout en tenant compte du modèle de NoC utilisé. Nous avons implanté ECTM et DTFM dans un simulateur d'ordonnancement appelé Cheddar.

Title: Mixed Criticality System Scheduling Over NoC Architectures

Keywords: Network On Chip (NoC), Communication Time analysis, Mixed Criticality System.

Abstract:

This thesis addresses existing challenges that are associated with the implementation of Mixed Criticality Systems over NoC architectures. In such system, we must ensure the timing constraints for critical applications while limiting the bandwidth reservation for them.

In order to run Mixed Criticality systems on NoC architectures, we have proposed several contributions in the form of a NoC router, a task and flow model, and a communications model.

First, we propose a NoC router called DAS (Double Arbiter and Switching), designed to efficiently run mixed criticality applications on Network On Chip. To enforce MCS requirements, DAS implements automatic mode changes, two levels of preemption, two flow control techniques and two stages of arbitration.

We have implemented DAS in the cycle accurate SystemC-TLM simulator SHOC. Then, we have evaluated DAS with several abstraction-level methods.

Second, we propose DTFM, a Dual Task and Flow Model in order to overcome the limitation of existent task and flow models. DTFM allows us, from the task model, the NoC model and the task mapping, to compute the corresponding flow model.

Finally, we propose a new NoC communication model called Exact Communication Time Model (ECTM) in order to analyze the scheduling of periodic tasks exchanging messages over a NoC. We have implemented our approach in a real-time scheduling simulator called Cheddar.