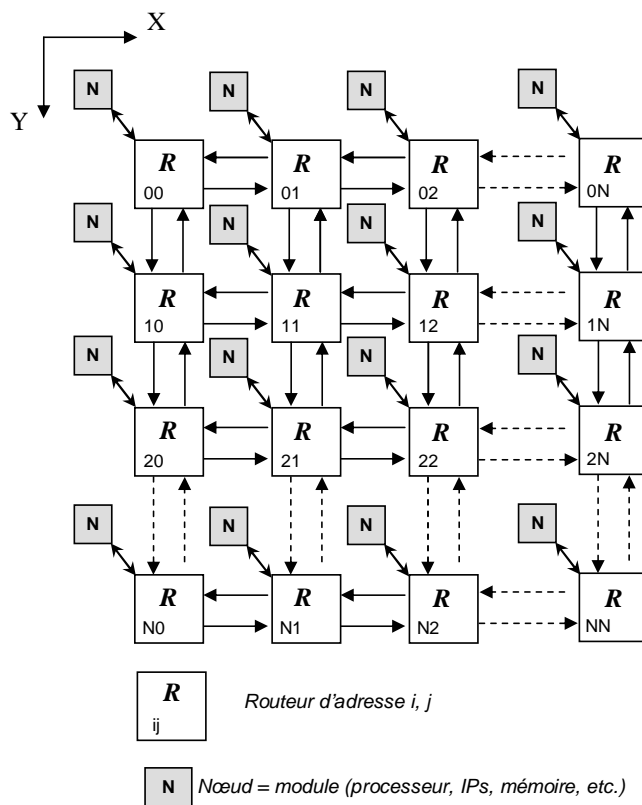




buffers d'un routeur est égale à la taille d'un *flit*. Les spécifications d'un routeur sont alors décrites de la façon suivante :

- Routeur de paquets de données de 5 *flits* de taille 9 bits (5x 9 bits).
- Canaux de données de taille 9 bits (taille d'un *phit*).
- Un buffer à chaque entrée de direction de profondeur 2 *flits* (2 x 9 bits).
- Cellule interne d'aiguillage du routeur (*Switch*) de type *Crossbar*.
- Quatre directions de transfert de données (*Est, Ouest, Sud et Nord*).
- Algorithme de routage déterministe de type *X\_Y*.
- Règle d'aiguillage de type *Wormhole* : la connexion entre une entrée et une sortie de direction d'un routeur est maintenu jusqu'à ce que tous les données élémentaires (*flits*) d'un paquet de message sont envoyés.



**Figure 2.** Structure du NoCs de type Mesh NxN.

La figure 3 donne une description de la spécification d'un routeur *NoC*. La modélisation globale du réseau s'effectue ensuite par une description structurelle *VHDL* réalisée par des instanciations du module routeur en vue d'élaborer un réseau *NoC* de taille paramétrable 3x3, 6x 6 et 10x 10 (figure 2).

### B. Algorithme de routage

L'algorithme proposé aux étudiants est un algorithme de routage statique de type *X\_Y* signifiant que les paquets de données sont dirigés vers le *PE* destinataire à travers les routeurs du réseau selon d'abord l'axe des abscisses *X* du réseau, puis selon son axe des ordonnées *Y*. La modélisation de l'algorithme de routage *X\_Y* est réalisée en *VHDL* selon l'algorithme suivant :

- If  $X_{\text{routeur}} < X_{\text{destination}}$ , direction paquets = Direction\_Est
- If  $X_{\text{routeur}} > X_{\text{destination}}$ , direction paquets = Direction\_Ouest
- If  $X_{\text{routeur}} = X_{\text{destination}}$  et  $Y_{\text{routeur}} > Y_{\text{destination}}$ , direction paquets = Direction\_Sud
- If  $X_{\text{routeur}} = X_{\text{destination}}$  et  $Y_{\text{routeur}} < Y_{\text{destination}}$ , direction paquets = Direction\_Nord
- If  $X_{\text{routeur}} = X_{\text{destination}}$  et  $Y_{\text{routeur}} = Y_{\text{destination}}$ , direction paquets = Local\_PE

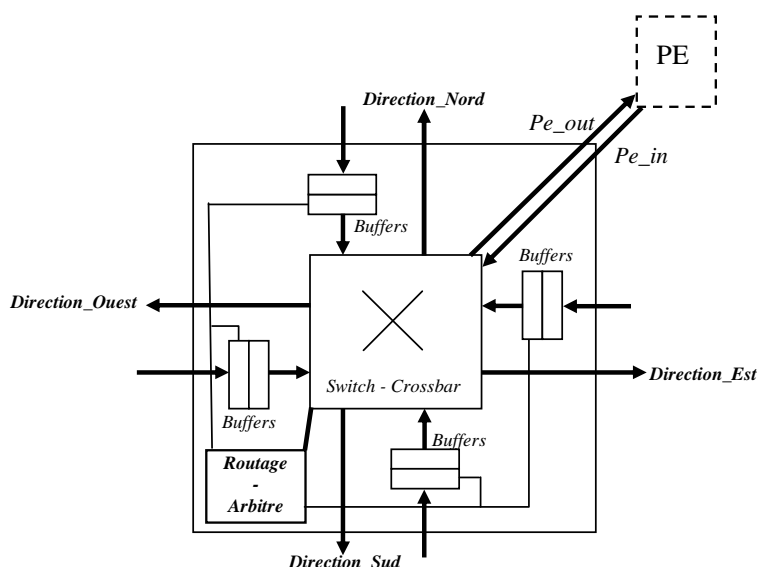


Figure 3. Illustration du modèle structurel d'un Routeur NoC.

La figure 4 illustre l'algorithme de routage  $X_Y$  d'un paquet entre les  $PE_{00}$  et  $PE_{22}$ . Une mauvaise description de cet algorithme peut faire apparaître au cours de la simulation du NoC des interblocages (*Deadlock*) ou des bouclages (*Livelock*) de paquets [1].

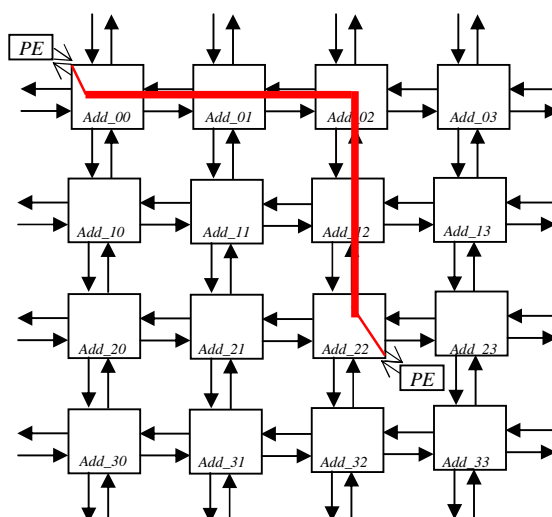


Figure 4. Illustration de l'algorithme  $X_Y$ .

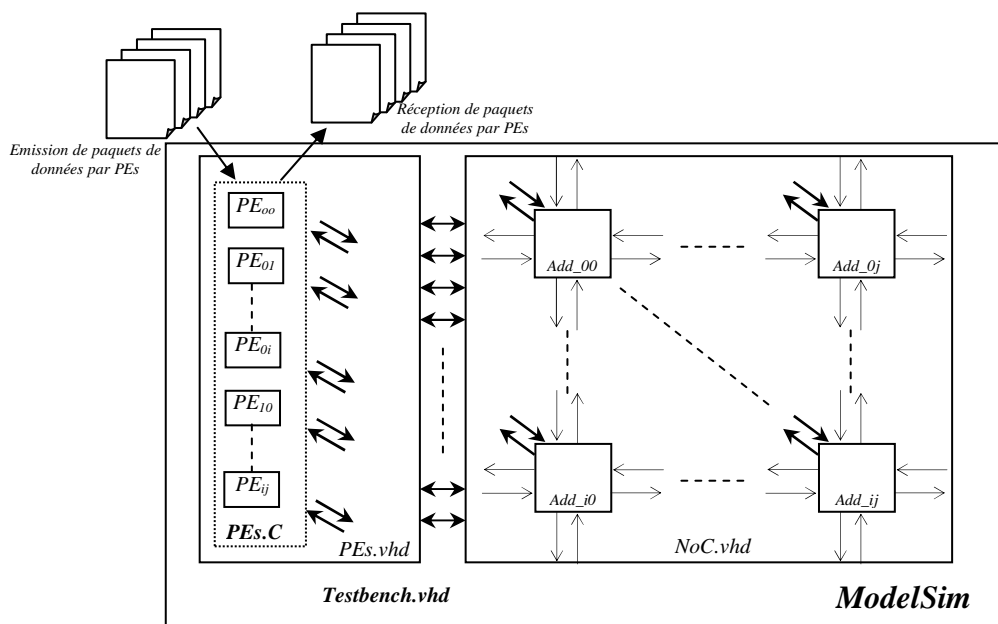
### C. Police d'arbitrage

L'élaboration des règles d'arbitrage de priorité de l'ordre de traitement des paquets entrant dans un routeur et routés selon les sorties de direction, est laissée à l'initiative des étudiants. Ces règles doivent être associées à l'algorithme de routage, au nombre de *flits* constituant les paquets et aux règles d'aiguillage *Wormhole*. L'objectif est d'amener les étudiants à une description itérative progressive du module de routage et d'arbitrage au fur et à mesure des phases de modélisation – simulation mettant en évidence les phénomènes de blocage temporaire, de verrouillage (*starvation*) et leurs impacts sur la notion de latence de transfert de paquets de données.

### 2. 2. Co-Simulation sous environnement ModelSim

A partir d'une description en langage C modélisant l'envoi et la réception de paquets de données par l'ensemble des *PEs* associés à chaque routeur du NoC, les étudiants réalisent une modélisation comportementale *VHDL* détaillée d'un NoC (routeurs, contrôleurs de flux données, etc.) et une validation par co-simulation *C-VHDL*

grâce à l'interfaçage *VHDL FLI (Foreign Language Interface)* de *ModelSim* [3]. La modélisation et la simulation du réseau *NoC* développé par les étudiants sont assurées par la description structurée *VHDL* du réseau associée à une modélisation en langage *C/C++* de l'ensemble des *PEs* du réseau assurant les émissions et les réceptions des paquets de données. La figure 5 illustre l'interfaçage et l'association d'une co-simulation sous environnement *ModelSim*. A partir des fichiers contenant les paquets de données à transmettre par des *PE* émetteurs vers *PEs* destinataires, un interfaçage entre la description *VHDL* du *NoC* et le model *C/C++* des fonctionnalités d'émission et de réception de paquets par chaque *PE* est exécuté sous *ModelSim*.



**Figure 5.** Illustration de la co-simulation *C-VHDL* sous *ModelSim*.

En pratique, le model abstrait des *PEs* est mis en œuvre à travers une compilation *DLL* de la modélisation *C/C++* des *PEs*. Cette dernière est ensuite définie comme la partie « architecture » de la description *VHDL* des *PEs* du *NoC* sous *ModelSim*. La figure 6 montre l'intégration de fonctionnalité abstrait *C/C++* dans la partie « architecture » de la description *VHDL* des *PEs* du *NoC*.

```
architecture arch_c_module_10x10_xy of c_module_10x10_xy is
|attribute foreign:string;
attribute foreign of arch_c_module_10x10_xy: architecture is "c_module_10x10_xy_init ./c_module_10x10_xy_with_lat_vf_6x6_5flits.dll";
begin
end;
```

**Figure 6.** Attribution *FLI* d'un model abstrait par *DLL* dans une description *VHDL*.

La figure 7 présente le contenu d'un fichier associé à un *PE* et contenant les paquets (constitués de 5 *flits*) à transmettre. Le premier *flit* correspond à l'adresse du *PE* émetteur, le second *flit* à l'adresse du *PE* destinataire suivi ensuite des *flits* de données à transmettre.

```
out_10x10_xy_0 - Bloc-notes
Fichier Edition Format Affichage ?
100 017 0A1 0A2 0A3
100 018 0A1 0A2 0A3
100 040 0A1 0A2 0A3
100 050 0A1 0A2 0A3
100 016 0A1 0A2 0A3
100 018 0A1 0A2 0A3
100 02B 0A1 0A2 0A3
100 023 0A1 0A2 0A3
100 05A 0A1 0A2 0A3
100 017 0A1 0A2 0A3
100 01B 0A1 0A2 0A3
100 045 0A1 0A2 0A3
100 04B 0A1 0A2 0A3
100 01E 0A1 0A2 0A3
```

**Figure 7.** Exemple de paquets de données à transmettre par le *PE* d'adresse 00.

Au cours des phases de co-simulation, les étudiants sont sensibilisés aux risques d'interblocage (*Deadlock*), de bouclage (*Livelock*) et de verrouillage (*Starvation*) des paquets de données dans un *NoC* [1]. Ces risques sont principalement dus au partage de ressources limitées et aux règles d'accès à ces ressources. Ils sont couramment étudiés et analysés dans le développement d'un *NoC* adapté pour la conception d'un *MPSoC* spécifique.

Les Figures 8 et 9 donnent respectivement des exemples de résultat de co-simulation sous ModelSim. La figure 8 présente un aiguillage *Wormhole* d'un paquet de données selon le routage *X\_Y*. La figure 9 montre la réception d'un paquet de données par un *PE* du *NoC*.

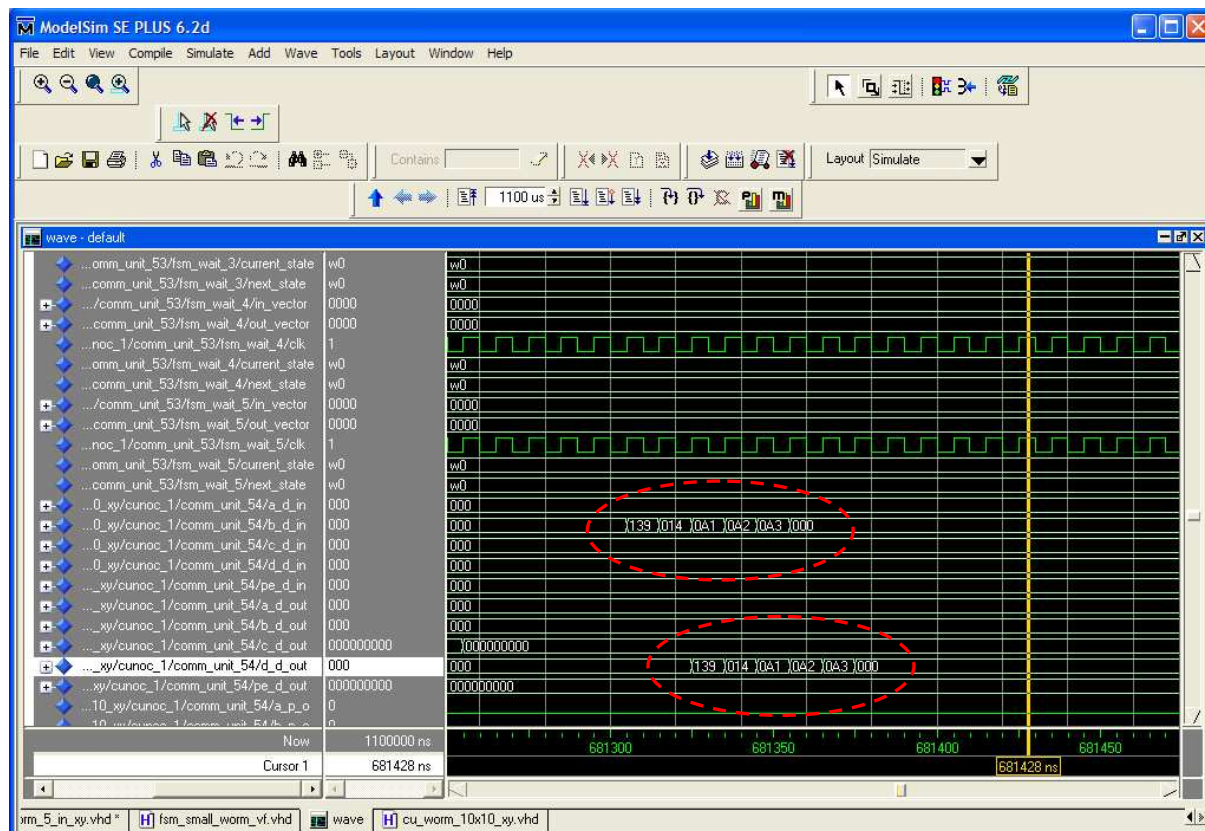


Figure 8. Exemple de résultat de simulation de l'aiguillage *Wormhole* d'un paquet selon *X\_Y*.

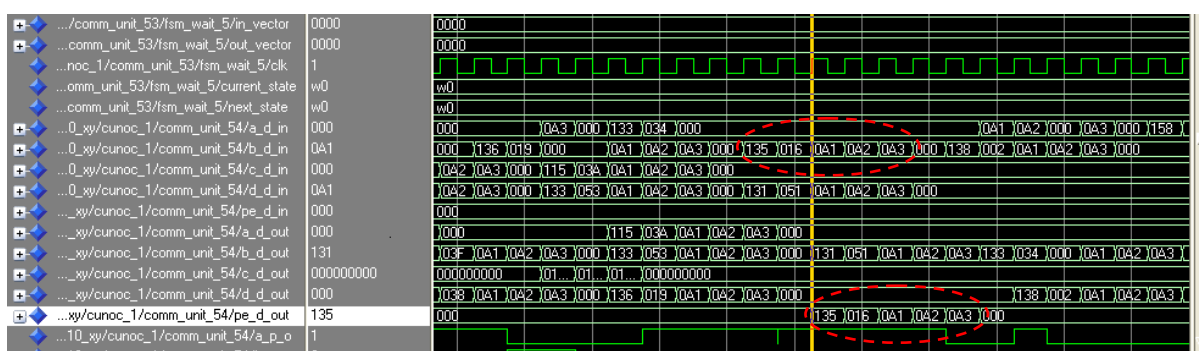


Figure 9. Exemple de simulation de réception d'un paquet données par le *PE*<sub>54</sub>.

Cette co-simulation permet une validation accélérée comportementale du *NoC* tout en mettant en évidence sa caractérisation à travers les notions de latence et de débit de données. La figure 10 montre un exemple des

résultats de temps de latence en nombre de cycle des paquets transmis dans le *NoC* par un *PE* émetteur vers des *PE\_destinataires*.

out_10x10_xy_0 - Bloc-notes					out_lat_10x10_xy_0 - Bloc-notes				
Fichier	Edition	Format	Affichage	?	Fichier	Edition	Format	Affichage	?
100	017	0A1	0A2	0A3	100	017	10		
100	018	0A1	0A2	0A3	100	018	16		
100	040	0A1	0A2	0A3	100	040	22		
100	050	0A1	0A2	0A3	100	050	58		
100	016	0A1	0A2	0A3	100	016	8		
100	018	0A1	0A2	0A3	100	018	12		
100	02B	0A1	0A2	0A3	100	02B	18		
100	023	0A1	0A2	0A3	100	023	24		
100	05A	0A1	0A2	0A3	100	05A	57		
100	017	0A1	0A2	0A3	100	017	10		
100	01B	0A1	0A2	0A3	100	01B	26		
100	045	0A1	0A2	0A3	100	045	38		
100	04B	0A1	0A2	0A3	100	04B	32		
100	01E	0A1	0A2	0A3	100	01E	26		

**Figure 10.** Latences simulées en nombre de cycle d'horloge des paquets transmis par le PE\_00.

### 3. CONCLUSION

Cet enseignement propose de modéliser, de simuler et d'évaluer l'architecture d'un réseau de communication sur puce à l'aide d'une co-simulation *C-VHDL* avec l'outil *Modelsim*. Il permet de sensibiliser les étudiants au rôle fondamental des interconnectes sur les performances attendues d'un *MPSoC* lors de sa conception. La modélisation des transactions des paquets de données est réalisée en *C/C++* ce qui permet une simulation rapide nécessaire pour vérifier les fonctionnalités et les performances d'un réseau de communication sur puce pour des systèmes intégrés complexes tandis que les modules du *NoC* à réaliser sont décrits en *VHDL* en vue d'une description synthétisable sur une technologie donnée. L'objectif de cet enseignement est d'une part de sensibiliser les étudiants sur les phases de développement et de conception microélectronique de l'interconnecte, constituant l'élément clé des performances d'un *MPSoC*. D'autre part, de montrer l'intérêt d'utiliser des outils tel que *ModelSim* permettant une co-simulation dans le but de simuler et de valider une architecture dans son contexte environnemental de fonctionnement. Cet enseignement est complémentaire à l'enseignement de modélisation de système haut niveau à partir d'un langage tel que *SystemC*. En effet, il complète l'enseignement de la conception microélectronique en présentant des outils permettant d'accélérer les phases de développement de systèmes qui deviennent de plus en plus complexes par l'intégration à la fois de parties matérielles et logicielles.

### BIBLIOGRAPHIE

- [1] G. De Micheli et L. Benini, « Networks on Chips, Technology and tools », Morgan Kaufmann publishers, 2006.
- [2] Mentor Graphics, « Modelsim SE User's Manuel, Software, Version 6. 4 », 2008. <http://www.mentor.com/>.
- [3] Mentor Graphic, « ModelSim, Foreign Language Interface, version 5.6d », August 2002.