Chapitre 6

Le Circuit POEtic

C'est pas avec des idées poétiques qu'on fait une révolution.

Bertrand BONELLO , *Dialogue du film Le pornographe*

DANS la section 3.5, page 62, nous avons défini une architecture de cellule POEtic. Un tissu POEtic, embarquant de telles cellules, ainsi que des capacités d'évolution, pourrait évidemment être créé pour une application particulière, sous la forme d'un circuit intégré composé d'un tableau de cellules déjà spécialisées, et d'un processeur pouvant y faire évoluer des organismes artificiels. Cependant, chaque nouvelle application impliquerait la réalisation d'un nouveau circuit, avec le temps de développement et les coûts que cela implique. Il serait dès lors nettement plus avantageux de disposer d'un substrat reconfigurable sur lequel nous pourrions implémenter n'importe quel type de cellules.

Le circuit POEtic [168], dont la partie reconfigurable a été développée dans le cadre de cette thèse, a été conçu dans ce but, et propose des mécanismes facilitant l'implémentation de ces systèmes multicellulaires bio-inspirés. Un tableau d'élément simples, que nous appelons molécules, y sert à accueillir les cellules, qui ensemble forment un organisme (Figure 6.1). Ce substrat reconfigurable dispose de capacités qui ne sont pas présentes dans les FPGAs commerciaux, et qui peuvent être exploitées par ces applications spécifiques :

- Le routage dynamique permet aux cellules de créer des connexions durant la “vie” de l'organisme.
- L'auto-configuration partielle des molécules offre de la plasticité aux cellules, qui peuvent modifier leur comportement, notamment lors de la phase de différenciation des mécanismes ontogénétiques.
- Le routage inter-moléculaire est effectué via des multiplexeurs, de manière à empêcher toute possibilité de court-circuit, ce qui fait de POEtic une plateforme idéale pour de l'évolution matérielle non-contrainte.
- La configuration des molécules est exécutée par lots de 32 bits, afin d'accélérer le chargement d'un organisme.

Nous allons à présent nous intéresser à la réalisation physique de ce circuit POEtic,

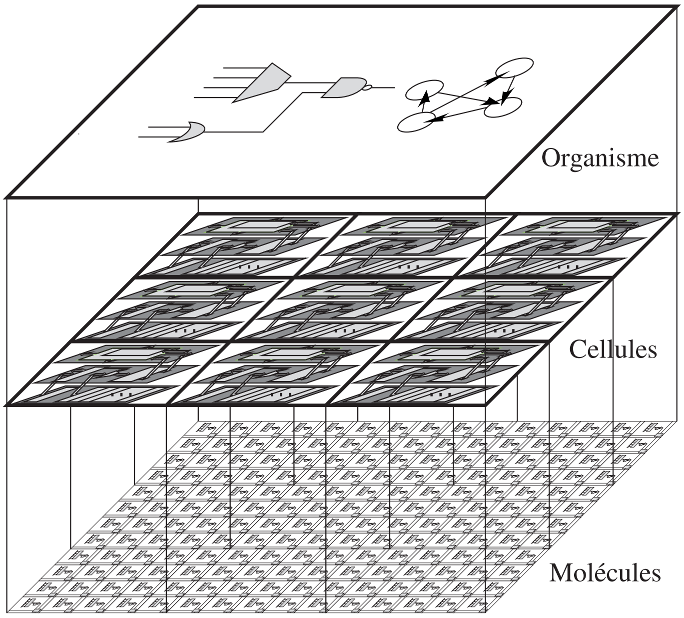


Figure 6.1 : *Les trois niveaux d'organisation d'un organisme implémenté sur le circuit POEtic.*

principalement composé d'un microprocesseur et d'un tableau d'éléments reconfigurables. Ce dernier est couplé à un système de routage distribué quasiment identique à HIDRA, que nous avons présenté en détail en page 104, et constitue donc une application directe des recherches effectuées sur le routage distribué du chapitre 5. Bien qu'il soit un System On Chip générique, nous allons présenter le circuit POEtic en partant du principe que ce sont des cellules possédant une architecture de type POEtic qui y seront implémentées, étant donné qu'il a spécialement été conçu pour accueillir des designs multicellulaires bio-inspirés.

Avant de poursuivre, il est important d'accentuer le fait que le circuit POEtic a été réalisé physiquement, et non uniquement simulé. Un premier prototype ne contenant que 12 molécules et le microprocesseur a tout d'abord été réalisé et testé avec succès avant que ne le soit le circuit final, ce dernier étant actuellement en phase de test. Ce chapitre présente donc la structure du circuit réel, qui a dû être figée dans le silicium, et n'est donc plus modifiable. Nous verrons toutefois que quelques améliorations pourraient être faites si une deuxième version de POEtic devait voir le jour.

Nous allons tout d'abord présenter la structure globale du circuit. Nous nous arrêterons ensuite sur le tableau reconfigurable, qui fut développé dans le cadre de cette thèse. Le routage distribué sera brièvement rappelé, et ses différences d'avec HIDRA explicitées, avant de définir exactement son interface le connectant au tableau reconfigurable. Nous verrons alors comment plusieurs circuits POEtic peuvent être connectés en un super-circuit. La partie reconfigurable ayant été explicitée, sous passerons rapidement sur les caractéristiques du microprocesseur embarqué, ce dernier ayant été conçu à Barcelone, ainsi que sur l'interfaçage entre ce processeur et la partie reconfigurable. Nous présenterons ensuite comment certains composants à la base de nombreux designs peuvent y être efficacement implémentés. Nous terminerons finalement par les outils de développement, qui ont été réalisés dans le but d'épauler un utilisateur final dans la conception de designs spécifiquement prévus pour notre circuit.

6.1 Structure globale

Le circuit POEtic est composé de trois parties distinctes : le sous-système environnemental, le sous-système organique, et l'interface entre les deux (Figure 6.2).

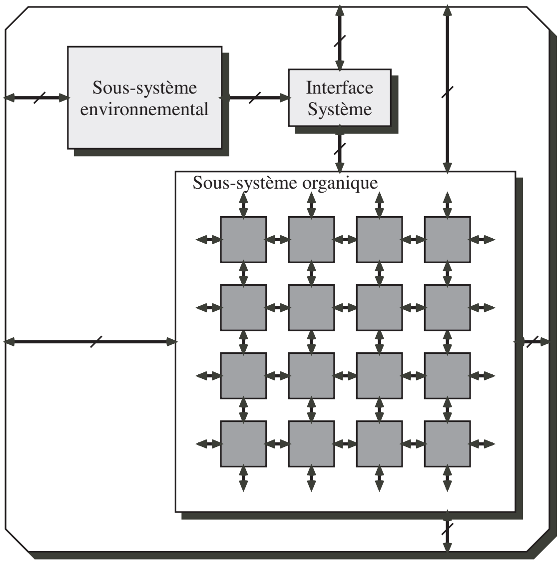


Figure 6.2 : *La structure du circuit POEtic, composé de trois parties.*

Le sous-système environnemental est chargé de gérer les entrées/sorties du circuit, en récupérant les données de l'environnement, et en les envoyant au sous-système organique, où l'organisme y étant implémenté peut les intégrer de manière à agir en conséquence. Il est également responsable des processus évolutifs et du chargement des individus dans le sous-système organique. Un microprocesseur 32 bits de type RISC constitue le cœur de ce sous-système, offrant ainsi un grande généricité au circuit, un microprocesseur étant capable d'effectuer n'importe quel traitement. En outre, des périphériques particuliers lui ont été ajoutés, tels un générateur de nombre pseudo-aléatoires, qui s'avère fort utile lors de l'exécution d'algorithmes génétiques.

Le sous-système organique est la partie dans laquelle sont chargés les individus, et où leur comportement est exprimé, et évalué. L'ontogenèse prend place dans ce sous-système, les cellules qui y sont implémentées pouvant être capables de s'auto-configurer sur la base de leur génome et des interactions qu'elles entretiennent avec les autres cellules, ainsi qu'avec leur environnement. De manière plus générale, ce sous-système permet l'implémentation de n'importe quel type de design, qu'il soit bio-inspiré ou non. Les éléments reprogrammable qui composent ce sous-système sont appelés "molécules", par analogie avec le niveau hiérarchique trouvé dans les organismes vivants.

L'interface des systèmes est, quant à lui, responsable de la bonne communication entre les deux sous-systèmes. En outre, il gère automatiquement les systèmes multi-circuits, et offre donc une meilleure scalabilité au tissu complet.

Nous allons commencer par décrire le sous-système organique, qui a été développé par nos soins, la description des deux autres parties nécessitant de connaître la structure de celui-ci.

6.2 Le sous-système organique

Notre circuit est spécifiquement dédié à des applications bio-inspirées dont la structure est multicellulaire, et les cellules ayant une architecture POEtic doivent y être facilement implémentables (Figure 6.1). Étant donné que nous désirions offrir assez de souplesse pour permettre l'implémentation de n'importe quel type de cellules et de connectique entre ces cellules, nous avons développé un nouveau type de circuit reconfigurable. Composé de trois niveaux (Figure 6.3), un tableau de molécules, un tableau d'unités de routage, et d'une interface entre les deux, il offre des caractéristiques qui n'existent pas dans les FPGAs commerciaux : du routage autonome, de l'auto-reconfiguration partielle, et l'impossibilité de créer des courts-circuits.

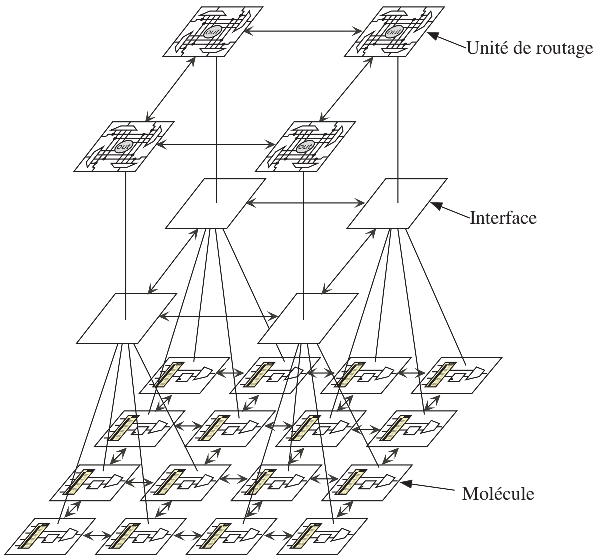


Figure 6.3 : *Les trois couches du sous-système organique.*

Le routage autonome permet aux cellules de créer de nouvelles connexions sans nécessiter l'intervention d'un contrôleur externe. De ce fait, des réseaux de neurones à topologie variable peuvent y être implémentés, de même que des mécanismes de croissance et d'autoréparation. La reconfiguration partielle, quant à elle, autorise les cellules à modifier les bits de configuration de leurs molécules, c'est-à-dire à modifier leur propre comportement, où celui d'autres cellules. Dans le cas de l'ontogenèse, il sera intéressant de pouvoir disposer de telles capacités permettant à une cellule de configurer une voisine, qui pourra ensuite la remplacer ou continuer une phase de croissance. Finalement, l'impossibilité de créer des courts-circuits va permettre l'utilisation du circuit POEtic comme base à du matériel évolutif, aucune combinaison de bits de configuration ne pouvant causer de tort au matériel. L'utilisation de bus de connexions a donc été remplacé ici par des switchboxes à base de multiplexeurs, qui ralentissent la propagation de signaux au travers du circuit, mais offrent cette garantie de sécurité.

Le premier niveau du sous-système organique est donc, comme nous l'avons déjà cité, composé de molécules. Chaque molécule (Figure 6.4) est principalement composée d'une look-up table de 16 bits, d'une bascule, et d'un switchbox lui permettant de communiquer avec d'autres molécules du même circuit.

Le niveau de routage distribué est, lui, composé d'un tableau d'unités de routage

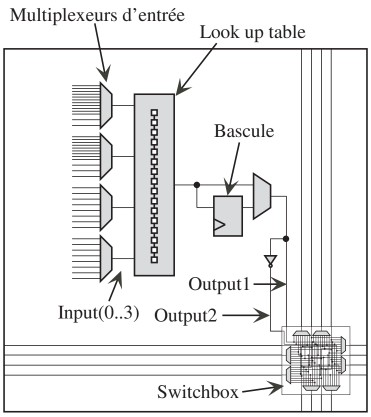


Figure 6.4 : *Structure de base d'une molécule.*

relativement semblables à celles que nous avons décrites dans l'implémentation de l'algorithme HIDRA. Chaque unité de routage est connectée à ses quatre voisines, ainsi qu'à une interface moléculaire. Ce niveau, qui fait la spécificité du circuit POEtic, permet de créer des connexions entre molécules durant le fonctionnement du système, et ce de manière totalement distribuée. Étant donné que nous allons présenter les molécules avant le routage distribué, il est important de noter une différence importante avec HIDRA. En effet, deux modes de routage sont disponibles, et sont imposés par les molécules elles-mêmes. Alors que le premier offre un comportement identique à HIDRA, le deuxième, appelé mode pseudo-statique, ne permet que de créer des communications selon un seul axe. Une source et une destination doivent y être situées sur la même ligne horizontale ou verticale, et le processus de routage complet est exécuté en 16 coups d'horloge.

Finalement, l'interface entre les molécules et les unités de routage a été conçu dans le but de pouvoir facilement paramétrer le nombre de molécules reliées à une unité de routage. Dans la réalisation physique du circuit, nous avons choisi un ratio de `4 : 1`, mais il serait extrêmement aisé de le modifier, par exemple pour relier neuf molécules à une unité de routage.

Nous allons maintenant décrire en détail le fonctionnement et l'implémentation des molécules. Nous continuerons ensuite avec les unités de routage, puis avec leur interface.

6.3 Les Molécules

Chaque molécule (Figure 6.4) contient une bascule D, une look-up table de 16 bits, et un switchbox. Les quatres entrées de la LUT, `Input(0..3)`, sont sélectionnées grâce à quatre multiplexeurs, qui sont détaillés en page 190, mais dont il est important de savoir qu'ils ont la possibilité de récupérer la sortie de la bascule. La sortie de la LUT peut passer par la bascule, en fonction de la configuration du multiplexeur de sortie. Enfin, deux signaux, `Output1` et `Output2`, sont transmis au switchbox, et sont, dans la configuration standard, la valeur sélectionnée par le multiplexeur de sortie et son inverse. Les molécules d'un même circuit peuvent communiquer en faisant passer des valeurs par les switchboxs moléculaires, qui offrent deux lignes unidirectionnelles dans chacune des quatre directions. Elle est donc reliée à ses quatre voisines, ainsi qu'à une unité de routage, pour les communications intercellulaires.

Alors que le projet embryonique [142] était basé sur une molécule dont la fonctionnalité était constituée d'un multiplexeur et d'une bascule, notre circuit offre une granularité plus grosse. Dans une cellule implémentée sur embryonique, une grande partie des molécules ne servent qu'à router des signaux, alors que la présence du switchbox dans les nôtres permet d'éviter le gaspillage de molécules à cette fin.

En comparaison des FPGAs dernier cri d'Altera et de Xilinx, qui contiennent plusieurs LUTs, notre molécule possède une structure moins complexe, mais offre l'avantage de disposer de mécanismes spécifiques permettant aux molécules de configurer leurs voisines, ainsi que d'accéder au niveau de routage distribué. De plus, la taille du circuit final étant limitée, nous devions prendre garde à ne pas développer de molécules trop grandes, qui n'auraient autorisé la présence que d'un petit nombre d'entre elles dans ce circuit.

Enfin, outre la fonctionnalité de base, qui consiste en une 4-LUT, la molécule peut être configurée selon huit modes opératoires, qui seront décrits en détails dans les section 6.3.1 à 6.3.8 :

- En mode **4-LUT**, la LUT fournit une sortie en fonction de ses quatre entrées, et la sortie peut passer par la bascule ou non (page 177).
- En mode **3-LUT**, la LUT est décomposée en deux LUTs à 8 bits. Elles partagent les mêmes trois entrées, et la sortie de la première peut passer par la bascule. La deuxième sortie, qui est reliée au switchbox, est également directement transmise à la molécule voisine au Sud, et permet d'implémenter efficacement des opérations arithmétiques nécessitant une retenue (page 178).
- En mode **Comm**, la LUT est décomposée en une LUT à trois entrées et un registre à décalage de 8 bits. Ce mode pourrait donc être utilisé pour comparer une adresse stockée dans le registre à décalage avec une entrée sérielle (page 178).
- En mode **Shift Memory**, les 16 bits de la LUT sont considérés comme un registre à décalage, et peuvent être utilisés pour stocker de l'information, comme un génome de cellule, par exemple. Deux entrées seulement sont utilisées dans ce mode, une pour le contrôle du décalage, et une comme donnée à insérer dans le registre (page 179).
- En mode **Input**, la molécule est considérée comme une entrée de la cellule, et a pour but de récupérer une valeur transmise via le niveau de routage distribué. Elle est connectée à une unité de routage, et peut ou non initier une nouvelle connexion. Dans tous les cas, elle se doit d'accepter toute connexion correspondant à son adresse, qui est stockée dans les 16 bits du registre à décalage. Seules deux entrées sont utilisées, l'une servant à initier une connexion, et l'autre servant à définir le mode de routage du tissu POEtic (page 180).
- En mode **Output**, la molécule est considérée comme une sortie de la cellule, et peut envoyer des valeurs via le niveau de routage distribué. De la même manière qu'une molécule Input, elle stocke son adresse dans son registre à décalage et possède une entrée pour initier une connexion. Sa deuxième entrée est alors la valeur à envoyer à sa destination correspondante (page 181).
- En mode **Trigger**, le registre à décalage de la molécule doit contenir la valeur "$000\cdots01$" pour une adresse codée sur 16 bits, et sert à synchroniser les unités de routage lors de la phase de comparaison d'adresse. L'une des entrées de la molécule est un *Chip enable*, capable de désactiver certaines molécules du tissu

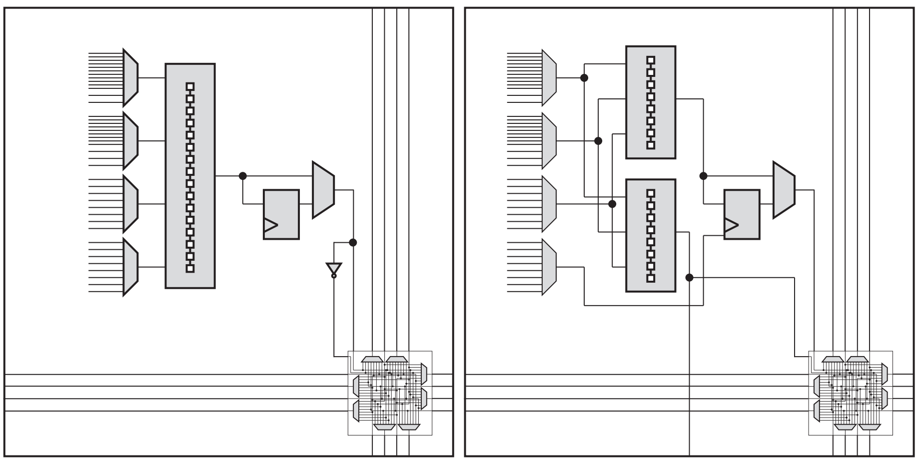
POEtic, et l'autre permet un reset du routage en détruisant toutes les connexions existantes (page 182).

- En mode **Configure**, la molécule a la capacité d'accéder aux bits de configuration d'autres molécules présentes sur le même circuit. Elle peut donc partiellement reconfigurer certaines molécules, grâce à ses deux entrées, dont une contrôle la reconfiguration tandis que l'autre correspond au bit à insérer à chaque coup d'horloge (page 184).

Nous allons à présent entrer dans le détail des modes opératoires de la molécule, avant de décrire l'implémentation des molécules et leurs spécificités.

6.3.1 Mode 4-LUT

En mode 4-LUT (Figure 6.5(a)), la molécule peut calculer n'importe quelle fonction à 4 entrées. Le signal `Input` sélectionne un bit parmi les seize du registre de la molécule. Il est donc possible d'implémenter n'importe quelle fonction à quatre variables avec une molécule 4-LUT. En ce qui concerne les entrées, les quatres valeurs de `Input` sont utilisées comme entrées de la look-up table, et les multiplexeurs sont donc séparés en 4 multiplexeurs à 8 entrées. En sortie, `Output1` correspond à la sortie de la look-up table, et peut être combinatoire ou séquentielle, tandis que `Output2` est l'inverse de `Output1`.



(a) (b)

Figure 6.5 : *A gauche, une molécule en mode 4-LUT, et à droite en mode 3-LUT.*

Entrée	Fonction
`Input(0)`	Bit de sélection 0 de la 4-LUT
`Input(1)`	Bit de sélection 1 de la 4-LUT
`Input(2)`	Bit de sélection 2 de la 4-LUT
`Input(3)`	Bit de sélection 3 de la 4-LUT
Sortie	**Fonction**
`Output1`	Résultat de la 4-LUT, séquentiel ou non
`Output2`	Inverse de `Output1`

6.3.2 Mode 3-LUT

En mode 3-LUT (Figure 6.5(b)), la molécule peut calculer deux fonctions utilisant les mêmes trois entrées. Le résultat de la première fonction peut passer par la bascule, alors que le deuxième est directement envoyé à la voisine au Sud, afin d'accélérer les opérations de type addition parallèle.

La LUT est séparée en deux LUTs de 8 bits. Le registre est donc coupé en deux, les 8 bits de poids faible définissant le comportement de la première LUT, et les 8 bits de poids fort définissant celui de la deuxième.

Les trois entrées de la première LUT sont identiques à ceux de la deuxième, et correspondent aux trois premiers signaux d'entrée `Input(0..2)`. La dernière entrée, `Input(3)`, agit comme un *enable* de la bascule, et n'est utilisée que si le bit de configuration de l'*enable* de la bascule est actif.

Entrée	Fonction
`Input(0)`	Bit de sélection 0 des deux 3-LUT
`Input(1)`	Bit de sélection 1 des deux 3-LUT
`Input(2)`	Bit de sélection 2 des deux 3-LUT
`Input(3)`	*enable* de la bascule D
Sortie	**Fonction**
`Output1`	Résultat de la première 3-LUT, séquentiel ou non
`Output2`	Résultat de la deuxième 3-LUT

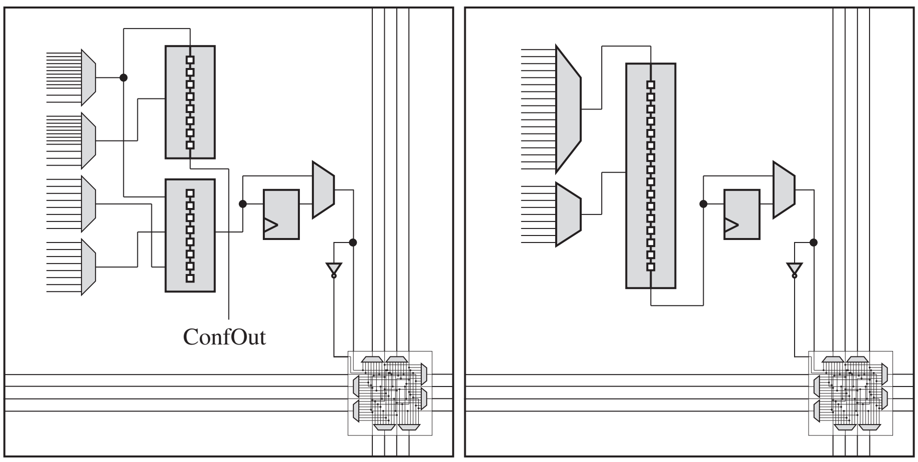
`Output1` correspond à la sortie de la première LUT, et peut être séquentielle ou non, tandis que `Output2` est la sortie de la deuxième LUT, qui peut également être directement récupérée par la molécule au Sud. Ce signal permet d'éviter de devoir passer par le switchbox moléculaire, et donc de réquisitionner inutilement des ressources, dans le cas d'application nécessitant des opérations parallèles.

6.3.3 Mode Comm

Le mode Comm a initialement été développé pour permettre l'implémentation efficace d'un mécanisme de communication par paquets, qui ne nécessiterait pas de passer par le routage distribué. Les 16 bits du registre à décalage sont coupés en un registre à décalage de 8 bits pour l'octet de poids faible, et une LUT de 8 bits pour l'octet de poids fort (Figure 6.6(a)). Une des entrées contrôle le décalage du registre, et les trois autres correspondent aux entrées de la 3-LUT. Notons toutefois que $Input(0)$ est utilisé conjointement par la 3-LUT et par le registre à décalage, dont il est le bit à insérer.

Entrée	**Fonction**
`Input(0)`	Bit de sélection 0 de la 3-LUT, et bit d'entrée du registre à décalage
`Input(1)`	Contrôle du décalage du registre
`Input(2)`	Bit de sélection 2 de la 3-LUT
`Input(3)`	Bit de sélection 1 de la 3-LUT

Sortie	**Fonction**
`Output1`	La valeur calculée par la 3-LUT
`Output2`	Inverse de `Output1`



(a) (b)

Figure 6.6 : *A gauche, une molécule en mode Comm, et à droite en mode Mémoire à décalage.*

Le bit de poids fort du registre à décalage peut être récupéré par les molécules voisines, le signal `ConfOut` de la figure 6.6(a) étant accessible via un niveau de communication normalement réservé aux bits de configuration.

Pour l'implémentation d'une communication par paquets, le registre à décalage peut contenir une adresse, sur 8 bits, qui doit être comparée avec une adresse arrivant en série. Le registre à décalage peut boucler sur lui-même, son entrée étant également utilisée par la LUT. En utilisant le bit sériel de l'adresse à comparer comme deuxième entrée de la LUT, et la bascule comme troisième, il est alors aisé de réaliser un comparateur d'adresse sériel dans une seule molécule.

6.3.4 Mode Shift Memory

Une molécule en mode Shift Memory, que nous appellerons également mode Mémoire, sert à stocker de l'information de manière sérielle. Les 16 bits du registre sont considérés comme un registre à décalage, et plusieurs de ces molécules peuvent être chaînées pour créer un registre à décalage de taille plus importante. Une entrée active à '1' gère le décalage, et l'autre correspond au bit à insérer lors d'une telle action.

Entrée	**Fonction**
`Input16(0)`	Bit d'entrée du registre à décalage
`Input16(1)`	Contrôle du décalage du registre
Sortie	**Fonction**
`Output1`	bit de poids fort du registre à décalage, où valeur de la bascule
`Output2`	Valeur inversée de Output1

Il est intéressant de noter que la sortie du registre à décalage peut passer par la bascule de la molécule avant d'être utilisée. De cette manière, il est possible de créer un registre à 17 bits dans une seule molécule.

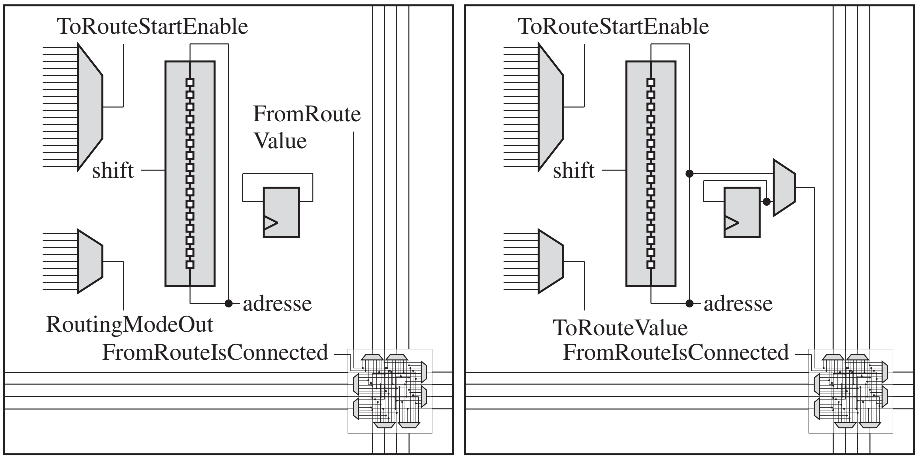
6.3.5 Mode Input

Alors que les modes précédents sont des caractéristiques qu'il est possible de retrouver dans certains FPGAs actuels (Virtex de Xilinx, par exemple), le mode Input est une particularité du circuit POEtic. Une molécule dans ce mode opératoire accède à l'unité de routage qui lui est connectée, et est considérée comme une destination. Deux entrées gèrent cette molécule : la deuxième entrée sert à indiquer le mode de routage général, tandis que la première indique si l'unité de routage doit impérativement se connecter à sa source correspondante. Si elle est inactive, l'unité de routage se doit toutefois d'accepter une connexion si une source possédant la même adresse tente de créer un nouveau chemin.

Entrée	**Fonction**
`Input16(0)`	Force la création de la connexion
`Input16(1)`	Sélection du mode de routage
Sortie	**Fonction**
`Output1`	Valeur transmise par l'unité de routage
`Output2`	Indique si la molécule est connectée à sa correspondante

Les deux sorties de la molécule sont respectivement la valeur d'entrée fournie par l'unité de routage, et une indication sur l'état de la connexion : si la deuxième sortie est à '1', cela signifie que l'unité de routage est connectée à sa correspondante.

Le registre à décalage contient ici une adresse, qui est accédée par l'unité de routage de manière sérielle, lors d'un processus de routage. L'unité de routage gère donc le décalage, et récupère le bit de poids fort du registre à décalage. Cette adresse peut être composée de 16, 8, 4, 2, ou 1 bits, en fonction du nombre de connexions potentielles du circuit. La taille est gérée par la molécule en mode Trigger, et elle définit la manière de remplir le registre de 16 bits. En effet, pour des tailles inférieures à 16 bits, l'adresse doit être répétée, comme indiqué dans le tableau 6.1, où a_i est le i-ème bit de l'adresse.



(a) (b)

Figure 6.7 : *A gauche, une molécule en mode Input, et à droite en mode Output.*

Taille de l'adresse	**Contenu du registre**
1	$a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0\ a_0$
2	$a_1a_0\ a_1a_0\ a_1a_0\ a_1a_0\ a_1a_0\ a_1a_0\ a_1a_0\ a_1a_0$
4	$a_3a_2a_1a_0\ a_3a_2a_1a_0\ a_3a_2a_1a_0\ a_3a_2a_1a_0$
8	$a_7a_6a_5a_4a_3a_2a_1a_0\ a_7a_6a_5a_4a_3a_2a_1a_0$
16	$a_{15}a_{14}a_{13}a_{12}a_{11}a_{10}a_9a_8a_7a_6a_5a_4a_3a_2a_1a_0$

Tableau 6.1 : *Contenu du registre, en fonction de la taille des adresses utilisées par le routage distribué.*

Si le mode de routage sélectionné par les molécules du circuit est pseudo-statique, le contenu du registre est directement chargé dans les bits de configuration des multiplexeurs de l'unité de routage, par un accès sériel, et ce en 16 coups d'horloge. Le tableau 6.2 définit les bits du registre responsables des multiplexeurs, et le tableau 6.3 indique le signal sélectionné par ces multiplexeurs, en fonction de leurs bits de configuration.

6.3.6 Mode Output

Le mode Output (Figure 6.7(b)) est relativement similaire au mode Input, mais correspond à une source de données qui seront transmises via le routage distribué. Le registre y est géré de la même manière, et son contenu est identique à celui d'une molécule Input. Une des deux entrées permet de forcer une connexion, tandis que la deuxième sert tout naturellement de valeur à transmettre via le routage distribué, vers la destination connectée. En sortie de la molécule, la première peut être le bit de poids fort de l'adresse (peu utile), ou la valeur contenue dans la bascule, qui peut permettre de disposer d'une valeur fixe à '0' ou à '1'. La deuxième, à l'instar du mode Input, indique si l'unité de routage est connectée ou non à sa correspondante.

bits du registre	**Multiplexeur**
2..0	Vers le Nord
5..3	Vers l'Est
8..6	Vers le Sud
11..9	Vers l'Ouest
14..12	Vers la molécule

Tableau 6.2 : *En mode de routage pseudo-statique, le contenu du registre est directement utilisé pour configurer les multiplexeurs de l'unité de routage.*

Bits de configuration	**Vers le Nord**	**Vers l'Est**	**Vers le Sud**	**Vers l'Ouest**	**Vers la molécule**
X00	Molécule	Nord	Nord	Nord	Nord
X01	Est	Molécule	Est	Est	Est
X10	Sud	Sud	Molécule	Sud	Sud
X11	Ouest	Ouest	Ouest	Molécule	Ouest

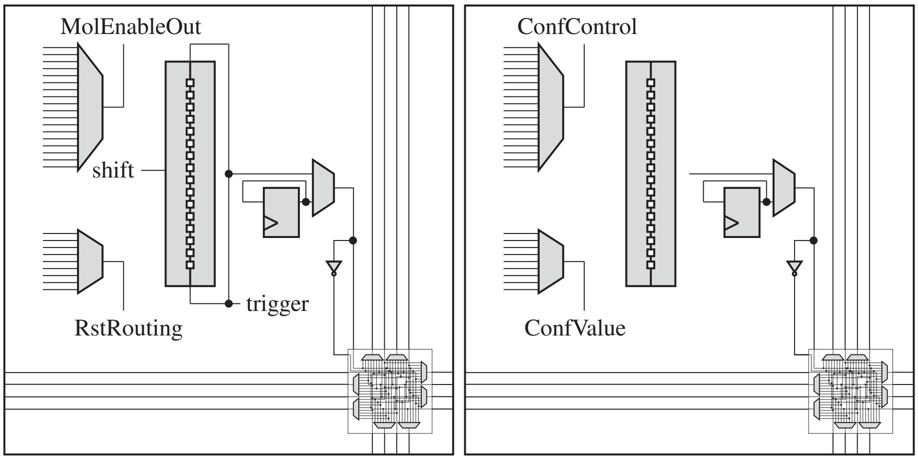
Tableau 6.3 : *Signal sélectionné par chaque multiplexeur de l'unité de routage, en fonction de ses bits de configuration.*

Entrée	**Fonction**
`Input16(0)`	Force la création de la connexion
`Input16(1)`	Valeur à transmettre à l'unité de routage
Sortie	**Fonction**
`Output1`	Si la sortie est combinatoire, il s'agit du bit de poids fort du registre. Sinon il s'agit de la valeur de la bascule
`Output2`	Indique si la molécule est connectée à sa correspondante

6.3.7 Mode Trigger

Une molécule en mode Trigger (Figure 6.8(a)) a un status un peu particulier. Elle n'exécute aucune tâche spécifique en relation avec d'autres molécules du circuit, mais est utile aux unités de routage pour synchroniser la phase de comparaison des adresses. Le contenu du registre est décalé de la même manière que les adresses des molécules Input et Output, et est en charge de fournir un '1' pour indiquer la fin de la comparaison. Pour une adresse de taille n, il doit donc contenir $n-1$ '0's, puis un '1', ce motif étant répété de façon à remplir le registre, comme indiqué dans le tableau 6.4.

Les deux entrées de la molécule ont des fonctions globales. La première agit comme un *chip enable*, par lequel certaines molécules, sensibles à ce signal, peuvent être momentanément désactivées. Cette particularité offre notamment la possibilité de séparer une cellule en une partie fonctionnelle et une partie ontogénétique, cette dernière pouvant s'occuper de construire l'organisme, tout en inhibant la partie fonctionnelle. La deuxième entrée permet de réinitialiser le routage dynamique, en forçant une



(a) (b)

Figure 6.8 : *A gauche, une molécule en mode Trigger, et à droite en mode Configure.*

Taille de l'adresse	**Contenu du registre**
1	1111111111111111
2	0101010101010101
4	0001000100010001
8	0000000100000001
16	0000000000000001

Tableau 6.4 : *Le contenu d'une molécule en mode Trigger dépend de la taille des adresses utilisées par le routage distribué.*

reconstruction des chemins de données. Une cellule peut donc, si besoin s'en fait sentir, comme durant des processus d'autoréparation, réinitialiser le routage intercellulaire de manière à prendre en compte une nouvelle cellule capable de remplacer une cellule défaillante. Le reset du routage distribué est effectué après un coup d'horloge, de manière synchrone, une bascule ayant été placée après la porte ET générale réunissant les signaux de Reset de toutes les molécules.

Entrée	**Fonction**
`Input16(0)`	Désactive certaines molécules
`Input16(1)`	Force un Reset du routage distribué
Sortie	**Fonction**
`Output1`	Si la sortie est combinatoire, il s'agit du bit de poids fort du registre. Sinon il s'agit de la valeur de la bascule
`Output2`	La valeur inversée de `Output1`

La sortie `Output1` peut être soit le bit de poids fort du trigger (peu utile), soit la valeur de la bascule, et la sortie `Output2` correspond à l'inverse de `Output1`. Ces deux sorties n'ont que peu d'utilité, une molécule Trigger n'étant présente que pour fournir une valeur à une unité de routage ainsi que deux signaux globaux, mais ont le mérite d'exister.

6.3.8 Mode Configure

Une molécule en mode Configure (Figure 6.8(b)) a la capacité de partiellement reconfigurer d'autres molécules présentes sur le même circuit. Pour ce faire, deux entrées sont utilisées : la première indique aux voisines si elles doivent effectuer une configuration partielle, et la deuxième est le bit à décaler dans les bits de configuration des molécules cibles.

Entrée	**Fonction**
`Input16(0)`	Active la configuration du voisinage
`Input16(1)`	Bit de configuration envoyé au voisinage
Sortie	**Fonction**
`Output1`	Si la sortie est combinatoire, il s'agit du bit de poids fort du registre. Sinon il s'agit de la valeur de la bascule, qui n'est pas fixe après un Reset [1]
`Output2`	La valeur inversée de `Output1`

La reconfiguration partielle, expliquée en page 192, est une des spécificités du circuit POEtic. Une cellule POEtic y a donc la possibilité de modifier son comportement en reconfigurant une partie de ses molécules. Pour l'implémentation de processus ontogénétiques, nous pouvons couper la cellule en une partie fonctionnelle et une partie chargée de la division et de la différenciation. Si un génome est stocké dans les cellules, et que ce génome représente des bits de configuration de molécules, la cellule peut sélectionner, en fonction de sa différenciation, une partie du génome, et la charger dans les molécules de la partie fonctionnelle.

6.3.9 Entrées/sorties

Les entrées/sorties d'une molécule peuvent être classées dans quatre catégories : les signaux globaux, les signaux de configuration, les communications intermoléculaires avec les quatre voisines, et les communications avec l'interface de l'unité de routage. Nous allons brièvement passer en revue ces différents signaux, afin de maîtriser l'interface des molécules.

Signaux globaux

Plusieurs entrées et sorties de la molécule agissent à un niveau global (Tableau 6.5), c'est-à-dire ont un impact direct sur le comportement de toutes les molécules, ou de toutes les unités de routage.

Le `ChipEnable`, géré par le microprocesseur externe, permet de bloquer le fonctionnement du circuit entier, par exemple durant la configuration de celui-ci.

[1] Dans une version ultérieure du circuit, il est bien clair que la valeur de la bascule devrait être fixe, pour une molécule en mode Config.

`IsRouting` indique si un processus de routage est en cours, dans quel cas la fontionnalité des molécules est bloquée, et seules les molécules Input, Output et Trigger sont actives. Enfin, `MolEnableIn` permet de désactiver certaines molécules (cf. page 195).

Concernant les sorties, `RstRouting` permet à une molécule Trigger de forcer un reset du routage distribué, tandis que `RoutingModeOut`, contrôlé par les molécules en mode Input, sélectionne le type de routage, qui peut être pseudo-statique, ou semblable à l'algorithme HIDRA. Enfin, une molécule Trigger peut utiliser `MolEnableOut` pour désactiver certaines molécules, ce signal étant à mettre en relation avec `MolEnableIn`.

Entrée	bits	Description
`clk`	1	Horloge globale du système
`rst`	1	Reset du système, pour la bascule D
`ChipEnable`	1	Désactive la molécule
`IsRouting`	1	Indique si un routage est en cours
`MolEnableIn`	1	Désactive la molécule
Sortie	**bits**	**Description**
`RstRouting`	1	Force un reset du routage, par une molécule Trigger
`RoutingModeOut`	1	Mode de routage
`MolEnableOut`	1	Permet de désactiver certaines molécules, commandé par une molécule Trigger

Tableau 6.5 : *Entrées/Sorties globales d'une molécule.*

Signaux de configuration

La configuration des molécules se fait de manière parallèle, en accédant le tableau de molécules comme une RAM de 32 bits de données (Tableau 6.6). Les données en entrée `ValueIn`, et respectivement en sortie `ValueOut`, permettent au microprocesseur de configurer les molécules, et de lire leur contenu, en choisissant le mode d'accès grâce au signal `Write`. Et étant donné que la molécule possède 76 bits de configuration, les trois signaux `Cs`, dont un seul doit être actif à la fois, sélectionnent la partie des bits de configuration qui est accédée (cf. page 192).

Entrée	bits	Description
`ValueIn`	32	Bits de configuration à charger
`Write`	1	Force une écriture des bits de configuration
`Cs`	3	Sélectionne un des trois blocs de bits de configuration
Sortie	**bits**	**Description**
`ValueOut`	32	32 bits de configuration, lors d'une lecture

Tableau 6.6 : *Entrées/Sorties d'une molécule, au niveau de la configuration.*

Signaux de communication intermoléculaire

Les molécules sont organisées selon une grille régulière à deux dimensions, où chacune est directement connectée à ses quatre voisines. Parmi les signaux du tableau 6.7, les connexions sont ainsi :

- `ValInN` $\Leftarrow$ `ValOutS` de la molécule au Nord.
- `ValInE` $\Leftarrow$ `ValOutW` de la molécule à l'Est.
- `ValInS` $\Leftarrow$ `ValOutN` de la molécule au Sud.
- `ValInW` $\Leftarrow$ `ValOutE` de la molécule à l'Ouest.
- `ConfPartInN` $\Leftarrow$ `ConfPartOut` de la molécule au Nord.
- `ConfPartInE` $\Leftarrow$ `ConfPartOut` de la molécule à l'Est.
- `ConfPartInS` $\Leftarrow$ `ConfPartOut` de la molécule au Sud.
- `ConfPartInW` $\Leftarrow$ `ConfPartOut` de la molécule à l'Ouest.
- `ConfInN` $\Leftarrow$ `ConfOut` de la molécule au Nord.
- `ConfInE` $\Leftarrow$ `ConfOut` de la molécule à l'Est.
- `ConfInS` $\Leftarrow$ `ConfOut` de la molécule au Sud.
- `ConfInW` $\Leftarrow$ `ConfOut` de la molécule à l'Ouest.
- `ChainIn` $\Leftarrow$ `ChainOut` de la molécule au Nord.

Entrée	**bits**	**Description**
`ConfPartInN`	1	Force une configuration partielle venant du Nord
`ConfPartInE`	1	Force une configuration partielle venant de l'Est
`ConfPartInS`	1	Force une configuration partielle venant du Sud
`ConfPartInW`	1	Force une configuration partielle venant de l'Ouest
`ConfInN`	1	Bit de configuration à insérer durant une reconfiguration partielle venant du Nord
`ConfInE`	1	Bit de configuration à insérer durant une reconfiguration partielle venant de l'Est
`ConfInS`	1	Bit de configuration à insérer durant une reconfiguration partielle venant du Sud
`ConfInW`	1	Bit de configuration à insérer durant une reconfiguration partielle venant de l'Ouest
`ValInN`	3	Valeurs transmises du Nord
`ValInE`	3	Valeurs transmises de l'Est
`ValInS`	3	Valeurs transmises du Sud
`ValInW`	3	Valeurs transmises de l'Ouest
`ChainIn`	1	Valeur de retenue envoyée par le Nord

Sortie	**bits**	**Description**
`ConfPartOut`	1	Propose une reconfiguration partielle, envoyée aux 4 voisines
`ConfOut`	1	Bits de configuration à décaler lors d'une reconfiguration partielle
`ChainOut`	1	Retenue envoyée au Sud, pour les opérations arithmétiques ou logiques
`ValOutN`	3	Valeurs transmises au Nord
`ValOutE`	3	Valeurs transmises à l'Est
`ValOutS`	3	Valeurs transmises au Sud
`ValOutW`	3	Valeurs transmises à l'Ouest

Tableau 6.7 : *Entrées/Sorties d'une molécule, pour la communication intermoléculaire.*

Les valeurs d'entrée de `ValOut`*X*`(0..1)` correspondent aux valeurs sélectionnées par le switchbox de la molécule, qui fournit deux signaux dans chacune des directions. `ValOut`*X*`(2)`, en revanche, est identique dans chacune des directions, et correspond à la première sortie `Output1`, calculée par l'unité fonctionnelle de la molécule. A titre d'exemple, pour une molécule en mode 3-LUT, il s'agit du résultat de la première LUT, ou de la sortie de la bascule D. Le fait qu'une molécule puisse ainsi récupérer de manière directe la sortie de n'importe laquelle de ces voisines permet d'économiser des ressources au niveau du routage intermoléculaire en groupant efficacement les molécules dont les fonctions sont intimement liées.

Le signal `Chain{In/Out}`, à l'instar du `ValOut`*X*`(2)`, permet également d'économiser du routage intermoléculaire. Il transmet à la molécule présente immédiatement au Sud la valeur de sortie de la deuxième LUT de la molécule, ce qui est utile lors d'opérations nécessitant une retenue, comme une addition parallèle, qui, pour un nombre de n bits, peut être implémentée grâce à n molécules (cf. page 211).

Les deux signaux `ConfPartOut` et `ConfOut` sont utilisés lors de la reconfiguration partielle de molécules (cf. page 192), et chacun d'eux est transmis aux quatre molécules voisines. Le premier contrôle cette reconfiguration partielle, tandis que le deuxième correspond au bit de donnée à insérer dans la configuration de la molécule destination.

Signaux de communication avec l'unité de routage

Comme nous l'avons déjà mentionné, quatre molécules sont reliées à une unité de routage via une interface (cf. page 202), et seule une de ces molécules peut être en mode Input ou Output, sans quoi un conflit empêche le bon fonctionnement du système.

En entrée de la molécule (Tableau 6.8), `FromRouteValue` est exploité par une molécule en mode Input, et consiste en la valeur transmise par la molécule Output reliée à elle via le routage distribué. `FromRouteShift`, commandé par l'unité de routage, force la molécule Input, Output, ou Trigger, à décaler le contenu de son registre 16 bits, durant un processus de routage. Enfin, `FromRouteIsConnected` indique simplement à une molécule Input ou Output si son unité de routage est connectée à une unité de routage ayant la même adresse.

La sortie `ToRouteValue` est plus ou moins l'homologue de `FromRouteValue`, et est la valeur envoyée, via le routage distribué, par une molécule en mode Output. De plus, elle sert, lors du processus de routage, à envoyer le bit de poids fort du registre 16 bits de la molécule, qui peut être l'adresse, dans le cas d'une molécule Input ou Output, ou un trigger, pour une molécule Trigger. `ToRouteStartEnable`, comme son nom l'indique, permet à une molécule Input ou Output de forcer la création d'un chemin dans le niveau de routage distribué, pour la connecter à la molécule, respectivement Output ou Input, possédant la même adresse. Et finalement, `ToRouteMode`, sur 3 bits, est le mode opératoire de la molécule, qui permet à l'interface molécule/unité de routage d'envoyer les bons signaux à l'unité de routage.

6.3.10 Communication intermoléculaire

Outre par sa fonctionnalité, une molécule est également définie par la communication qu'elle entretient avec ses voisines. Pour ce faire, chaque molécule possède un

Entrée	bits	Description
`FromRouteValue`	1	Valeur récupérée de l'unité de routage par une molécule en mode Input
`FromRouteShift`	1	Indique qu'il faut décaler le registre, pour une molécule Input, Output, ou Trigger
`FromRouteIsConnected`	1	Indique si l'unité de routage reliée à la molécule est connectée à sa correspondante
Sortie	**bits**	**Description**
`ToRouteValue`	1	Valeur transmise à l'unité de routage par une molécule Input, Output, ou Trigger
`ToRouteStartEnable`	1	Force l'unité de routage à démarrer un routage, pour une molécule Input ou Output
`ToRouteMode`	3	Mode opératoire de la molécule, transmis à l'unité de routage

Tableau 6.8 : *Entrées/Sorties d'une molécule, communiquant avec l'unité de routage.*

switchbox qui lui permet d'envoyer ou de faire transiter des valeurs vers d'autres molécules. Décision fut prise d'offrir deux lignes de communication vers chacune des quatre voisines, afin d'éviter les problèmes de congestion qui furent rencontrés dans le cadre du projet Embryonique. L'expérience a montré que ce type de communication permet d'implémenter un système multicellulaire sans problèmes de congestion, car une cellule n'est composée que d'un nombre relativement restreint de molécules – Le circuit final comportant 144 molécules, et la communication entre circuits ne se faisant qu'au niveau du routage distribué, une cellule ne doit pas dépasser une taille de 144 molécules, sans quoi elle doit être répartie sur plusieurs circuits. La communication intercellulaire doit passer par le niveau de routage distribué, et le routage intermoléculaire, confiné à l'intérieur d'une cellule, n'explose donc pas. La figure 6.9 montre la manière dont les molécules sont arrangées, ainsi que la structure du switchbox.

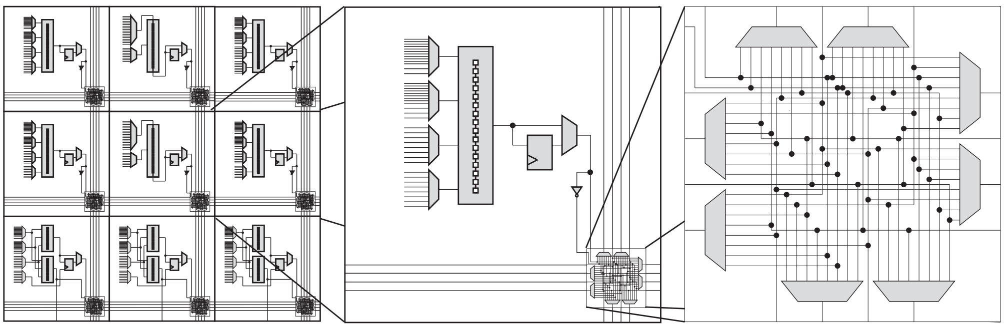


Figure 6.9 : *Un tableau de molécules, et le switchbox d'une molécule.*

En entrée du switchbox, nous trouvons huit lignes, représentant deux lignes par molécule voisine, ainsi que deux signaux résultant de l'activité de la molécule, `Output1` et `Output2`. Trois bits de configuration par multiplexeur définissent exactement la valeur à sélectionner, et donc un total de 24 bits de configuration sont nécessaires à la définition du comportement du switchbox. Le tableau 6.9 indique les signaux sélectionnés en fonction des bits de configuration.

ConfigSwitch(2..0)	ValOutN(0)	ConfigSwitch(5..3)	ValOutN(1)
000	Output1	000	Output1
001	Output2	001	Output2
010	ValInE(0)	010	ValInE(0)
011	ValInE(1)	011	ValInE(1)
100	ValInS(0)	100	ValInS(0)
101	ValInS(1)	101	ValInS(1)
110	ValInW(0)	110	ValInW(0)
111	ValInW(1)	111	ValInW(1)
ConfigSwitch(8.6)	ValOutE(0)	ConfigSwitch(11..9)	ValOutE(1)
000	ValInN(0)	000	ValInN(0)
001	ValInN(1)	001	ValInN(1)
010	Output1	010	Output1
011	Output2	011	Output2
100	ValInS(0)	100	ValInS(0)
101	ValInS(1)	101	ValInS(1)
110	ValInW(0)	110	ValInW(0)
111	ValInW(1)	111	ValInW(1)
ConfigSwitch(14..12)	ValOutS(0)	ConfigSwitch(17.15)	ValOutS(1)
000	ValInN(0)	000	ValInN(0)
001	ValInN(1)	001	ValInN(1)
010	ValInE(0)	010	ValInE(0)
011	ValInE(1)	011	ValInE(1)
100	Output1	100	Output1
101	Output2	101	Output2
110	ValInW(0)	110	ValInW(0)
111	ValInW(1)	111	ValInW(1)
ConfigSwitch(20..18)	ValOutW(0)	ConfigSwitch(23..21)	ValOutW(1)
000	ValInN(0)	000	ValInN(0)
001	ValInN(1)	001	ValInN(1)
010	ValInE(0)	010	ValInE(0)
011	ValInE(1)	011	ValInE(1)
100	ValInS(0)	100	ValInS(0)
101	ValInS(1)	101	ValInS(1)
110	Output1	110	Output1
111	Output2	111	Output2

Tableau 6.9 : *Bits de configuration du switchbox intermoléculaire.*

6.3.11 Multiplexeurs d'entrée

La molécule possède un nombre d'entrées non négligeable, dont certaines sont exploitées par la partie fonctionnelle de la molécule, comme les 8 entrées reliées au switchbox, les quatre entrées directes des voisines, la retenue venant du nord, etc. Or, au maximum quatre signaux sont nécessaires au fonctionnement de la molécule, dans les modes 4-LUT, 3-LUT, et Comm. Afin de sélectionner parmi cet ensemble de valeurs, des multiplexeurs ont été placés dans la molécule, et fournissent par exemple, les quatre signaux d'entrée de la LUT. Ce bloc de sélection des entrées fournit donc quatre valeurs, que nous nommons `Input(0..3)`.

Dans certains modes opératoires, seuls deux entrées sont nécessaires. Dans ces cas, les deux premiers multiplexeurs sont combinés pour obtenir plus de combinaisons, et les deux derniers le sont également, grâce à deux multiplexeurs visibles à droite de la figure 6.10. Les signaux `Input(0)` et `Input(2)` correspondent alors aux entrées utilisées, et nous les renommons `Input16(0)` et `Input16(1)`, pour plus de clarté.

Le bloc de sélection des entrées, observé à la figure 6.10, est donc principalement formé de quatre multiplexeurs à 8 entrées, chacun étant contrôlé par 3 bits de configuration `ConfigLutSel(3X,3X+1,3X+2)`, pour $X \in [0, 3]$. Si les deux bits de configuration `ConfigSpecialInput` et `ConfigDirectIn` sont à '0', les entrées de ces multiplexeurs correspondent plus ou moins aux entrées `ValInX(1,0)`, qui sont les valeurs envoyées par les switchboxs des quatre molécules voisines. Pour une molécule en mode 4-LUT, n'importe quelle combinaison de 4 parmi les 8 entrées standards est réalisable. Le troisième multiplexeur a également accès à la valeur de la bascule de la molécule, tandis que le deuxième peut fournir un '1' logique, qui constitue une valeur utile à plusieurs modes opératoires. La présence de la valeur de la bascule est évidemment indispensable pour la réalisation de compteurs ou de machines d'états, tandis que le '1' permet d'éviter de gaspiller une molécule pour fournir cette valeur logique.

Le nombre de signaux disponibles aux deux premiers multiplexeurs est augmenté, grâce aux bits de configuration `ConfigSpecialInput` et `ConfigDirectIn`. Le premier bit offre au premier multiplexeur les valeurs suivantes :

- La valeur calculée par la deuxième LUT de la molécule du Nord, qui permet de faciliter la réalisation d'opération nécessitant la propagation d'une retenue.
- Le bit de poids fort du registre de 16 bits de la molécule, qui est utile lorsque la molécule est en mode Mémoire. Il est alors possible d'en faire un registre à décalage bouclant sur lui-même, en sélectionnant son dernier bit comme entrée du registre.
- Le bit de donnée sélectionné par les deux bits de configuration utiles lors d'une reconfiguration partielle.
- La sortie de la bascule de la molécule.
- Et enfin la valeur logique '0', qui est notamment utile aux modes Input et Output, pour lesquels '0' est la valeur du signal `ToRouteStartEnable` pour ne pas forcer une connexion à s'établir.

Le bit `ConfigDirectIn` permet, quant à lui, au deuxième multiplexeur d'accéder aux valeurs directement transmises par chacune des molécules voisines, correspondant à leur première sortie.

Concernant le contrôle des multiplexeurs, en mode 4-LUT, 3-LUT, et Comm, chacun est géré par trois bits de configuration. En revanche, dans tous les autres modes,

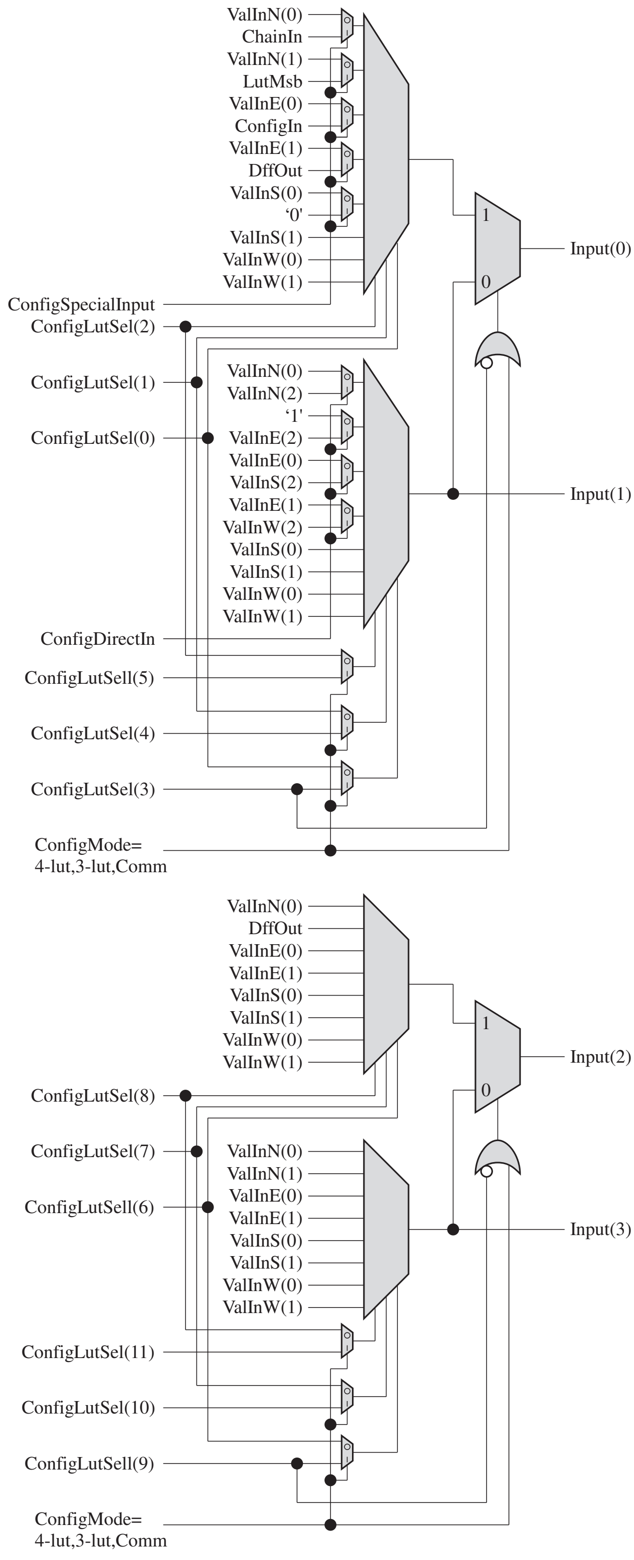


Figure 6.10 : *Multiplexeurs d'entrée.*

qui ne nécessitent l'utilisation que de deux entrées, les deux premiers multiplexeurs sont commandés par les mêmes trois bits (`ConfigLutSel(0,1,2)`), de même que

Mode	**Input(0)**	**Input(1)**	**Input(2)**	**Input(3)**
	Input16(0)		**Input16(1)**	
4-LUT	SelLut(0)	SelLut(1)	SelLut(2)	SelLut(3)
3-LUT	SelLut(0)	SelLut(1)	SelLut(2)	*Enable* de la bascule
Comm	SelLutHigh(0) et bit d'entrée	Décalage	SelLutHigh(2)	SelLutHigh(1)
Memory	Bit à insérer		Contrôle le décalage	
Input	Force la connexion		Mode de routage	
Output	Force la connexion		Valeur envoyée	
Trigger	*Enable* moléculaire		Reset du routage	
Configure	Contrôle de la configuration		Bit à insérer	

Tableau 6.10 : *Utilisation des valeurs fournies par les multiplexeurs d'entrée, en fonction du mode opératoire.*

les deux derniers le sont par `ConfigLutSel(6,7,8)`. Un petit multiplexeur à deux entrées sert ensuite à sélectionner la sortie d'un des deux multiplexeurs, en fonction d'un quatrième bit de configuration. L'avantage de ce couplage de multiplexeurs en un plus imposant réside dans le nombre d'entrées à disposition. En effet, si nous n'avions pas introduit le multiplexeur à deux entrées, les molécules n'utilisant que deux entrées n'auraient pas pu avoir le choix d'accéder les entrées directes des voisines, ou la retenue du Nord, par exemple.

En fonction du mode opératoire de la molécule, les entrées sélectionnées par nos multiplexeurs ont différentes fonctions, dont le tableau 6.10 offre un résumé.

6.3.12 Bits de configuration et reconfiguration partielle

Le comportement d'une molécule est défini non pas seulement par le contenu de son registre, mais par un total de 76 bits de configuration. Une des spécificités du circuit POEtic étant sa capacité d'autoreconfiguration partielle, nous avons séparé ces 76 bits en 5 blocs distincts, indépendamment reconfigurables par les molécules elles-mêmes. Chacun de ces blocs possède un bit supplémentaire indiquant s'il peut être reconfiguré ou non. De plus, au niveau de la molécule, trois bits de configuration spéciaux servent lors d'une reconfiguration partielle, pour indiquer son origine, et si elle doit être transmise à d'autres voisines. Au total, nous avons donc $5+3=8$ bits intouchables par une reconfiguration partielle, les autres étant modifiables par les molécules.

Le tableau 6.11 détaille les bits de configuration, séparés en cinq blocs reconfigurables (les trois premiers bits de la liste sont fixes). L'*enable* (qui correspond à un chip select) et le numéro indiqués sont utilisés lors de la configuration des molécules par le microprocesseur, présentée en page 206.

Le principe de la reconfiguration partielle est de laisser à une molécule en mode

Nombre de bits	*Enable* d'accès	Numéro de bit	Nom	Description
1	cs2	17	*ConfigPartialEnable*	*Enable* de configuration partielle
2	cs2	15..16	*ConfigPartialIn*	Origine d'une configuration partielle
1	cs0	16	*ConfigPartialLut*	*Enable* de configuration partielle de la LUT
16	cs0	0..15	LUT	Contenu du registre à décalage de 16 bits
1	cs0	31	ConfigPartialLutSel	*Enable* de configuration partielle des entrées
12	cs0	17..28	ConfigLutSel	Sélection des entrées de la molécules
1	cs0	29	ConfigSpecialInput	Sélection d'une entrée spéciale
1	cs0	30	ConfigDirectIn	Sélection d'une entrée comme valeur d'une voisine
1	cs1	24	*ConfigPartialSwitch*	*Enable* de configuration partielle du switchbox
3	cs1	0..2	ConfigSwitch(0..2)	Sélection pour la ligne ValOutN(0)
3	cs1	3..5	ConfigSwitch(3..5)	Sélection pour la ligne ValOutN(1)
3	cs1	6..8	ConfigSwitch(6..8)	Sélection pour la ligne ValOutE(0)
3	cs1	9..11	ConfigSwitch(9..11)	Sélection pour la ligne ValOutE(1)
3	cs1	12..14	ConfigSwitch(12..14)	Sélection pour la ligne ValOutS(0)
3	cs1	15..17	ConfigSwitch(15..17)	Sélection pour la ligne ValOutS(1)
3	cs1	18..20	ConfigSwitch(18..20)	Sélection pour la ligne ValOutW(0)
3	cs1	21..23	ConfigSwitch(21..23)	Sélection pour la ligne ValOutW(1)
1	cs2	3	*ConfigPartialMode*	*Enable* de configuration partielle du mode
3	cs2	0..2	ConfigMode	Mode opératoire de la molécule
1	cs2	14	*ConfigPartialOthers*	*Enable* de configuration partielle des bits divers
1	cs2	0	ConfigSequential	Sortie séquentielle ou combinatoire
1	cs2	1	ConfigRstValue	Valeur de Reset de la bascule
1	cs2	2	ConfigLocalRstEnable	Utilisation ou non de l'*enable* local de la bascule
1	cs2	3	ConfigClockEdge	Flanc de fonctionnement de la bascule
3	cs2	4..6	ConfigRstOrigin	Origine du Reset local
1	cs2	7	ConfigRstEnable	*Enable* du Reset local
1	cs2	8	ConfigSynchRst	Reset de la bascule synchrone ou asynchrone
1	cs2	9	ConfigMolEnable	Autorisation de l'*enable* moléculaire
1	cs2	18	DffOut	Valeur de la bascule

Tableau 6.11 : *Les bits de configuration d'une molécule, classés selon l'ordre en vigueur lors d'une reconfiguration partielle. Les signaux en italique ne peuvent être modifiés durant une reconfiguration partielle.*

Configure la possibilité de forcer la modification des bits de configuration d'autres molécules du circuit. La molécule en mode Configure utilise deux signaux, l'un activant la reconfiguration, et l'autre envoyant le bit à insérer. Dans les molécules atteintes par la reconfiguration, l'accès s'effectue de façon sérielle, en traversant les cinq blocs de bits de configuration, qui peuvent ou non être touchés par la reconfiguration. Les 8 bits fixes permettent de définir une direction par laquelle la configuration partielle peut se faire, ainsi que les blocs à modifier. Si aucun des blocs ne l'est, la molécule sert simplement à transmettre l'information, de façon à ce qu'une molécule puisse influencer des parties du circuit qui ne lui sont pas directement accolées.

La figure 6.11 montre comment la molécule de coordonnées $(1,1)$ peut reconfigurer quatre autres, aux coordonnées $(4,2),(5,2),(5,0),(6,0)$, qui ne sont pas directement contiguës. Pour ce faire, les molécules intermédiaires ont leur bit de configuration `ConfigPartialEnable` actif, et propagent donc le signal envoyé par la molécule $(1,1)$. Il est important de noter que ces molécules peuvent continuer à effectuer leur tâche principale alors même qu'elles servent à faire transiter les bits de configuration. Les flèches en entrée des molécules correspondent à l'origine sélectionnée par `ConfigPartialIn`. Il aurait été possible de faire l'usage d'un démultiplexeur en sortie de la molécule plutôt que d'un multiplexeur en entrée, ce qui aurait impliqué qu'une molécule aurait pu choisir vers quelle direction propager la configuration, plutôt que de choisir la provenance de celle-ci. L'option retenue possède toutefois l'avantage que plusieurs molécules peuvent être reconfigurées avec les mêmes données, comme le montre la figure 6.11, ce qui ne serait pas possible si une seule sortie de configuration était activable.

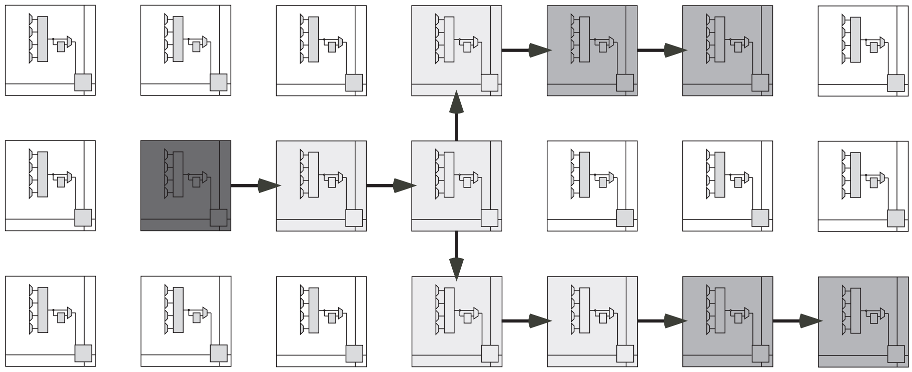


Figure 6.11 : *La molécule la plus foncée reconfigure les quatre molécules de grisé intermédiaire.*

Un bloc de bits de configuration, comme présenté à la figure 6.12, est composé d'un registre à décalage, qui est décalé lorsque le signal `conf` est à '0' et que le bit de configuration `ConfigPartial` est à '1'. Ce bit ne peut être accédé que par le sous-système environnemental, et définit donc le comportement du bloc lors d'une reconfiguration partielle. Nous pouvons observer que le bloc est *bypassé* s'il n'est pas décalé, de manière à pouvoir servir de transit à une reconfiguration devant affecter d'autres blocs ou d'autres molécules.

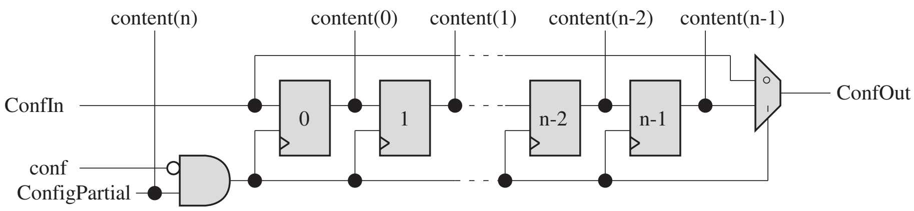


Figure 6.12 : *Un bloc de bits de configuration implémenté grâce à un registre à décalage.*

Les cinq blocs sont chaînés dans l'ordre de la figure 6.13, qui montre une reconfiguration partielle de deux blocs dont les bits de contrôle de reconfiguration sont actifs (noirs). Les deux bits de configuration `ConfigPartialIn` sélectionnent la molécule de par qui une reconfiguration partielle peut être exécutée, en sélectionnant le bit à insérer, et le signal de contrôle. Le bit de configuration `ConfigPartialEnable` indique, quant à lui, si la configuration partielle doit être propagée aux autres voisines. Ceci permet de chaîner plusieurs molécules au niveau des bits de configuration, ou de laisser une molécule reconfigurer une cousine lointaine, en traversant d'autres molécules sans les modifier.

La capacité de reconfiguration partielle, telle que nous venons de la décrire, constitue une des innovations du circuit POEtic. Elle pourra être plus qu'utile lors de l'implémentation de mécanismes ontogénétiques, et ce de plusieurs façons. Premièrement, nous verrons, en page 208, que la réalisation de registres à décalage pouvant stocker un génome peut en tirer profit, en mémorisant jusqu'à 54 bits par molécule. Deuxièmement, la phase de différenciation d'une cellule pourra se faire en reconfigurant certaines molécules de sa partie fonctionnelle.

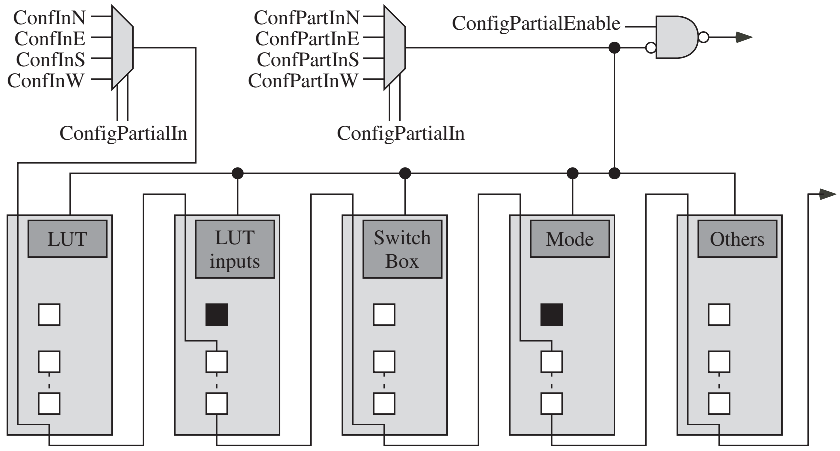


Figure 6.13 : *Les blocs de bits de configuration sont chaînés.*

6.3.13 Enable moléculaire

Une des particularités du circuit POEtic est son *enable* moléculaire, géré par les molécules en mode Trigger (page 182). Actif à '1', il est calculé en passant les sorties `MolEnableOut` de chaque molécule dans une porte ET, comme montré à la figure 6.14. De ce fait, une seule sortie à '0' force une désactivation de certaines molécules.

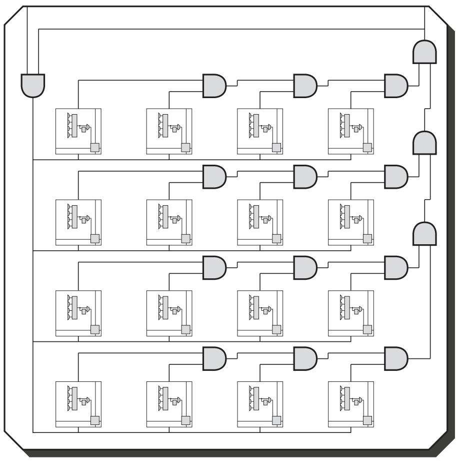


Figure 6.14 : *L'enable moléculaire global est calculé grâce à une grande porte ET.*

Dans la liste de ses bits de configuration, chaque molécule en possède un, nommé `ConfigMolEnable`, qui indique si la molécule doit être sensible ('1') ou non ('0') à l'*enable* moléculaire. Si tel est le cas, lorsque l'*enable* est à '0', les fonctions suivantes de la molécule sont affectées :

- Si la molécule est en mode Memory ou Comm, le décalage standard de la LUT est bloqué. Toutefois, la reconfiguration partielle de la molécule est autorisée.
- Si la molécule est en mode Configure, elle n'est plus autorisée à agir en tant que telle.
- La bascule de la molécule ne peut plus être chargée par un Load habituel, mais peut accepter un Reset.

Dans le cas d'un système multi-circuits, il est possible de combiner tous les signaux d'*enable* en sortie, dans une grande porte ET, et de réinjecter le résultat en entrée de chacun (signal entrant en haut à gauche de la figure 6.14). De ce fait, toutes les molécules présentes dans un tableau de circuits POEtic obéissent au même *enable* moléculaire.

Cette particularité de l'*enable* moléculaire sera notamment exploitée par les processus ontogénétiques que nous présenterons plus loin. Une cellule peut y être décomposée en une partie responsable de l'ontogenèse, et une partie fonctionnelle. Lors du processus de croissance de l'organisme, les parties fonctionnelles peuvent être inhibées, et ce dans toutes les cellules. Après la fin de la croissance, les cellules relâchent l'*enable*, et les parties fonctionnelles peuvent commencer le traitement pour lequel elles ont été configurées. De plus, des mécanismes d'autoréparation peuvent également en tirer parti, en bloquant la partie fonctionnelle des cellules, le temps que la cellule défectueuse soit remplacée par une autre.

6.3.14 Gestion de la bascule

Sur le plan fonctionnel, une molécule possède, outre une LUT, une bascule D. Elle permet d'implémenter des systèmes séquentiels, en y faisant passer la sortie de la LUT. La figure 6.15 illustre le contrôle de son fonctionnement, qui obéit notamment à plusieurs Reset.

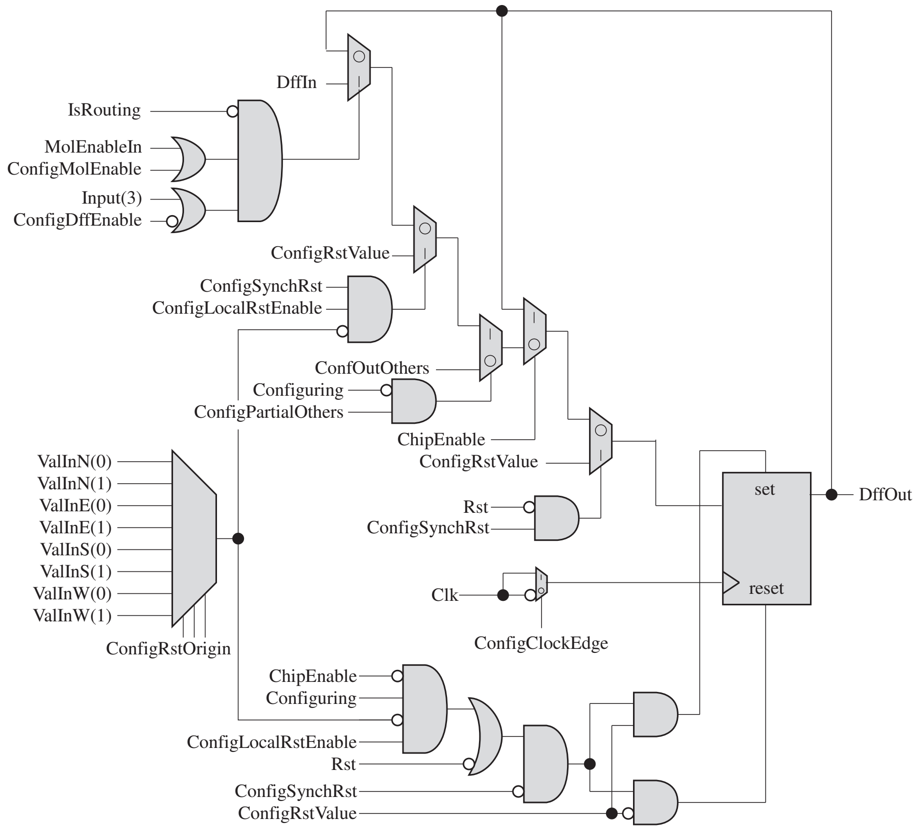


Figure 6.15 : *Gestion de la bascule d'une molécule.*

Tout d'abord, il est possible, grâce au bit de configuration `ConfigClockEdge`, de sélectionner le flanc d'horloge auquel est chargée la bascule : s'il est à '1', le flanc est montant, et s'il est à '0', le flanc est descendant.

Pour la modification du contenu de la bascule, le Reset global du circuit (`Rst`), actif à '0', a la priorité sur les autres actions. S'il n'est pas actif, toute autre action est conditionnée par le *chip enable* `ChipEnable`, qui bloque le fonctionnement des molécules s'il est inactif ('1').

Un Reset local offre aux molécules la capacité de forcer un reset de n'importe quelle autre, en faisant passer un signal par les switchboxes du routage intermoléculaire. Trois bits de configuration, `ConfigRstOrigin`, sélectionnent parmi les huit entrées du switchbox, et un Reset peut avoir lieu si ce signal est à '0' et que le bit de configuration `ConfigLocalRstEnable` est à '1'. Lors d'un reset, qu'il soit global ou local, la valeur de `ConfigRstValue` est chargée dans la bascule, de manière synchrone ou asynchrone, en fonction de la valeur de `ConfigSynchRst` ('1'=synchrone, '0'=asynchrone).

A l'instar des Resets, une configuration partielle peut modifier la valeur de la bascule. En effet, dans la chaîne des bits de configuration modifiables par les autres molécules, la bascule est placée en fin du bloc de bits de configuration divers. Dès lors, si ce bloc est soumis à une reconfiguration partielle, la bascule y participe, et elle se charge avec la valeur de `ConfigMolEnable`, qui est le dernier bit du bloc de configuration divers. La sortie de la bascule est alors récupérée par le système de configuration partielle et est envoyée aux quatre voisines, candidates potentielles à une telle configuration.

Enfin, si aucun Reset n'est actif, que le *chip enable* est actif, et qu'aucune configuration partielle n'a lieu, la bascule se trouve en fonctionnement normal, et obéit à un signal de chargement dépendant de plusieurs facteurs. Un chargement ne s'effectue que si aucun processus de routage n'est en cours, ce qui prévient le circuit de se trouver dans un état où des molécules Input ou Output seraient en cours de configuration partielle durant un processus de routage, ce qui aurait des conséquences néfastes sur le fonctionnement de l'algorithme. De plus, nous garantissons ainsi que les molécules exécutent leur tâche uniquement si toutes les connexions demandées via le routage distribué ont été créées. L'*enable* moléculaire peut également bloquer le chargement de la bascule. Le chargement n'est autorisé que si `MolEnableIn` est actif ('0'), et que `ConfigMolEnable` est également à '0'. Enfin, si `ConfigDffEnable` est à '1', la molécule doit prendre en compte un *enable* local, qui n'est autre que la sortie du quatrième multiplexeur d'entrée, `Input(3)`.

Lors d'un chargement de la bascule, le signal `DffIn` est calculé comme montré dans le schéma gauche de la figure 6.16. Si la molécule est en mode Input, Output, ou Trigger, la bascule mémorise sa valeur, et dans les autres modes, elle charge la valeur calculée par la LUT. Nous pouvons noter que le schéma de droite aurait été nettement plus judicieux, permettant à une molécule Input de faire passer la valeur reçue de l'unité de routage par la bascule.

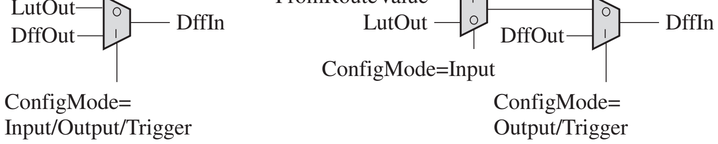


Figure 6.16 : *Valeur chargée dans la bascule, à gauche dans l'implémentation finale, et à droite telle qu'elle devrait être.*

6.3.15 Look-up table

La look-up table de la molécule est séparée en deux 3-LUTs, comme illustré à la figure 6.17. Chacune de ces LUTs possède trois entrées qui sélectionnent le bit de configuration à placer en sortie, une entrée contrôlant le décalage de ses bits de configuration, et un bit qui est inséré lors d'un décalage. En sortie, chacune fournit la valeur sélectionnée par les trois entrées, ainsi que le bit de poids fort de son registre.

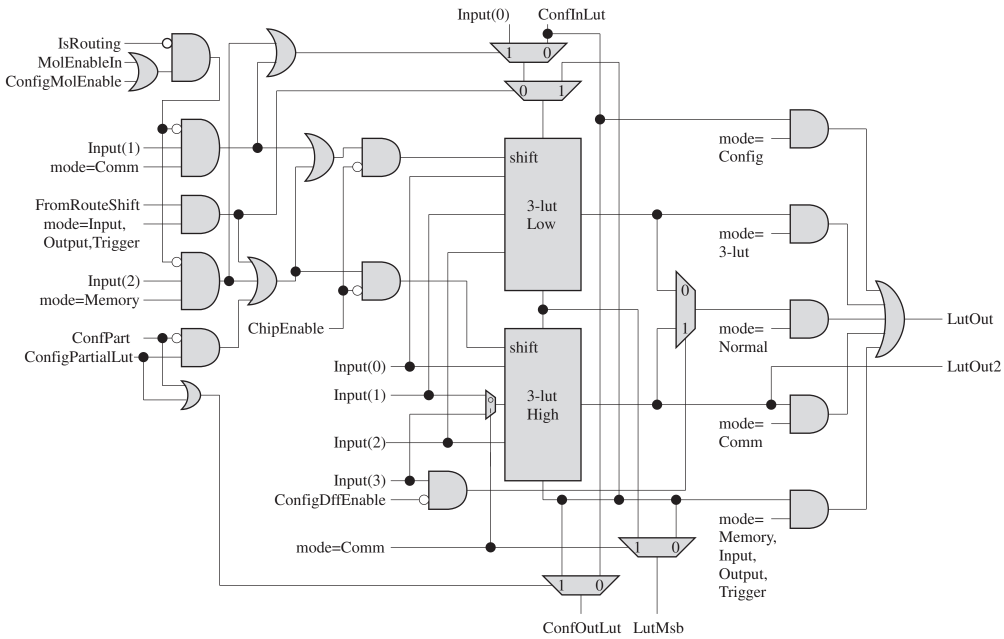


Figure 6.17 : *Mécanisme de contrôle des deux 3-LUTs.*

La première sortie dépend du mode opératoire de la molécule, et peut être la valeur calculée par la première 3-LUT, par la deuxième 3-LUT, le bit de poids fort de la deuxième 3-LUT, ou encore le bit de configuration partielle en entrée du bloc LUT [2]. La deuxième sortie n'est autre que la valeur calculée par la deuxième 3-LUT, qui est transmise à la molécule du Sud, ainsi qu'au switchbox, en mode 3-LUT. La sortie `LutMsb` est, quant à elle, le bit de poids fort du registre à décalage, qui correspond au bit de poids fort de la première 3-LUT en mode Comm, ou à celui de la deuxième dans les autres modes.

6.4 Le routage distribué

L'une des faiblesses des circuits reconfigurable existants est l'impossibilité qu'ils ont de se configurer dynamiquement, tant au niveau de la fonctionnalité des blocs de base, que du routage. Nous avons vu que nos molécules sont capables de configurer leurs voisines, ce qui offre des possibilités de configuration dynamique. Concernant le routage, nous avons couplé la grille de molécules à une grille d'unités de routage, qui implémentent l'algorithme HIDRA, décrit à la page 104. Nous n'allons pas représenter

[2] Cette dernière option n'est sélectionnée que dans le mode Config, et aurait dû permettre à la molécule de récupérer les bits de configuration d'une voisine pour directement les injecter dans une autre. Une erreur de design l'en empêche malheureusement.

son fonctionnement, mais uniquement mettre l'accent sur les quelques modifications que nous avons apportées aux unités de routage, qui offrent, outre le routage de type HIDRA, ce que nous appelons le routage pseudo-statique.

6.4.1 Routage pseudo-statique

Le routage selon HIDRA offre une grande souplesse, étant donné que les connexions se font sur la base d'identifiants, qui doivent être identiques pour qu'une source se connecte à une destination. Dans le cas où nous désirons réaliser un système multicellulaire où les cellules sont uniquement reliées à leurs voisines immédiates, il est nettement plus simple de pouvoir le faire avec des lignes droites (Figure 6.18) plutôt que par la connaissance des identifiants des cellules voisines.

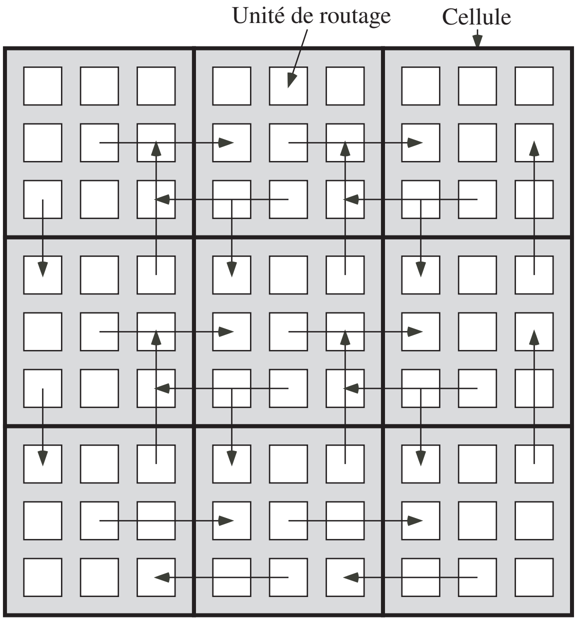


Figure 6.18 : *Connexions possibles de cellules avec leurs 4 voisines, grâce au routage pseudo-statique.*

Pour ce faire, les molécules en mode Input ont le pouvoir de forcer le routage à se faire dans ce mode, si au moins l'une d'elles place sa deuxième entrée à '0'. Pour que l'algorithme fonctionne, il est impératif que les bits de contrôle des multiplexeurs de l'unité de routage sélectionnent la valeur de l'entrée opposée, après un Reset du routage. Lorsqu'un processus de routage est lancé, le contenu du registre de chaque molécule Input et Output est chargé dans ces bits de contrôle, de manière sérielle. La configuration se termine après 16 coups d'horloge, lorsque la sortie des molécules Trigger, dont le registre doit contenir la valeur "000...001", s'active. Le circuit est alors opérationnel, et si le contenu des registres des molécules Input et Output ont été correctement définis, des liaisons en lignes droites directes sont créées entre les sources et les destinations, à la manière de la figure 6.18.

Sur le plan de l'implémentation, seules les commandes effectuées dans l'état sInit du contrôleur de HIDRA sont modifiées, de même que les bits de contrôle des multiplexeurs du switchbox. Au lieu de n'être que des registres chargés par le contrôleur, ils offrent également la possibilité de décaler leur contenu, à la manière d'un registre à décalage.

6.4.2 Entrées/sorties et systèmes multi-chip

Comme nous l'avons déjà mentionné, les cellules sont implémentées à l'aide de molécules. La communication intercellulaire est, quant à elle, réalisée via le niveau de routage distribué. Le circuit POEtic final n'étant pas de taille infinie, nous avons prévu de pouvoir en relier plusieurs entre eux, en un tableau de taille quelconque. Le nombre d'entrées/sorties étant également limité, il était impossible de laisser les molécules du bord du circuit communiquer directement avec leurs voisines immédiates des circuits voisins. La communication inter-circuit, pour les sous-systèmes organiques, se fait donc uniquement au travers du routage distribué, qui nécessite un nombre nettement plus réduit d'entrées/sorties.

La figure 6.19 montre l'organisation du niveau de routage distribué, et plus particulièrement la manière dont sont gérées les entrées/sorties. Chaque unité de routage envoie deux signaux vers chacune de ses voisines, comme nous pouvons l'observer au centre de la figure.

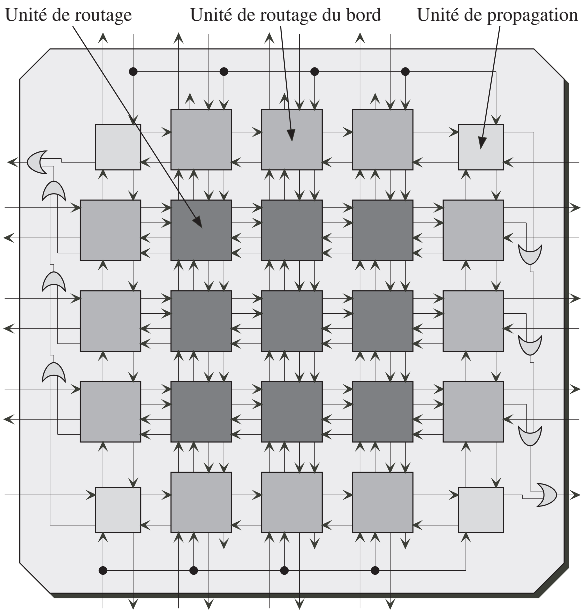


Figure 6.19 : *Les connexions entre unités de routage, ainsi que les entrées/sorties du circuit.*

Sur les bords du circuits sont disposés des unités de routage spéciales. Elles permettent de relier plusieurs circuits entre eux (Figure 6.20), et de gérer les entrées/sorties au niveau cellulaire. Elles sont composées d'un contrôleur légèrement simplifié par rapport aux unités de routage standards, ainsi que de quelques bits de configuration. Leur principe de fonctionnement dépend de ce à quoi elles sont connectées :

- Si le bord du circuit est relié à un autre circuit POEtic, les unités de routage du bord sont bypassées, et transmettent directement les signaux à l'unité standard à laquelle elles sont connectées.

- Si le bord du circuit n'est pas relié à un autre circuit POEtic, l'unité du bord peut agir comme une entrée, une sortie, ou une unité inactive.

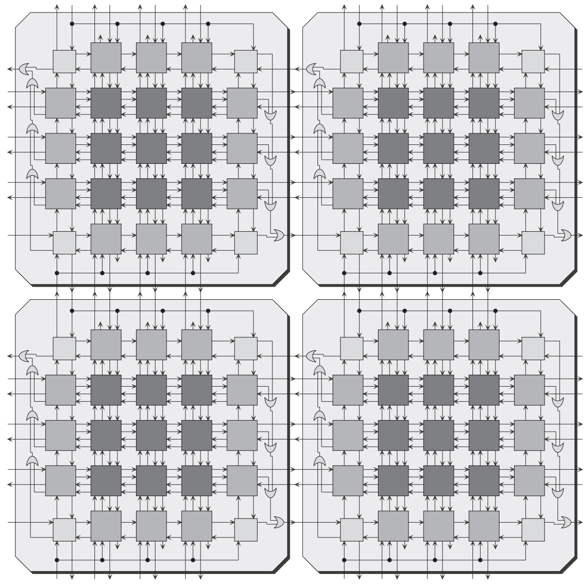


Figure 6.20 : *Connexions entre plusieurs circuits, au niveau du routage distribué.*

Le premier cas est trivial, et ne nécessite que la présence de quelques multiplexeurs sélectionnant les bons signaux à transmettre. Le deuxième, par contre, voit l'unité de routage du bord agir comme n'importe quelle autre unité de routage. Trois bits de configuration, dénommés `action(0..2)`, servent à définir exactement son comportement. Elle peut donc ne rien faire, être une sortie ou une entrée, et forcer une connexion ou non, et le tableau 6.12 définit exactement la fonctionnalité de l'unité en fonction des trois bits de configuration `action`.

action	**Description**
`001`	Sortie, qui ne nécessite pas forcément d'être connectée
`01X`	Entrée, qui ne nécessite pas forcément d'être connectée
`101`	Sortie, qui doit se connecter
`11X`	Entrée, qui doit se connecter
`X00`	Inutilisée, donc bypassée

Tableau 6.12 : *Bits de configuration d'une unité de routage du bord définissant son fonctionnement.*

Outre ces trois bits de configuration, une adresse est également nécessaire dans le cas ou l'unité de routage du bord est une entrée ou une sortie. Un registre à décalage de 16 bits est donc présent, et peut être chargé par le sous-système environnemental, en même temps que `action`, par l'interface parallèle.

La figure 6.21 montre le schéma d'une unité de routage du bord, avec en son centre le contrôleur simplifié. Les signaux globaux `Clk`, `Rst`, `ChipEnable`, `TriggerIn`, `RoutingMode`, `Congestion`, ainsi que ceux responsables de la configuration parallèle ont été omis, afin de ne pas surcharger le dessin. Le multiplexeur présent à droite du schéma permet de bypasser l'unité de routage, lorsqu'elle n'est pas utilisée comme entrée ou sortie du système. La valeur envoyée à l'unité de routage lui étant directement connectée à l'intérieur du circuit ne vient du contrôleur que lors d'un routage, si l'unité du bord est une entrée ou une sortie. De même, la porte ET présente en bas à gauche remplit la même fonction pour la valeur envoyée à l'extérieur du circuit.

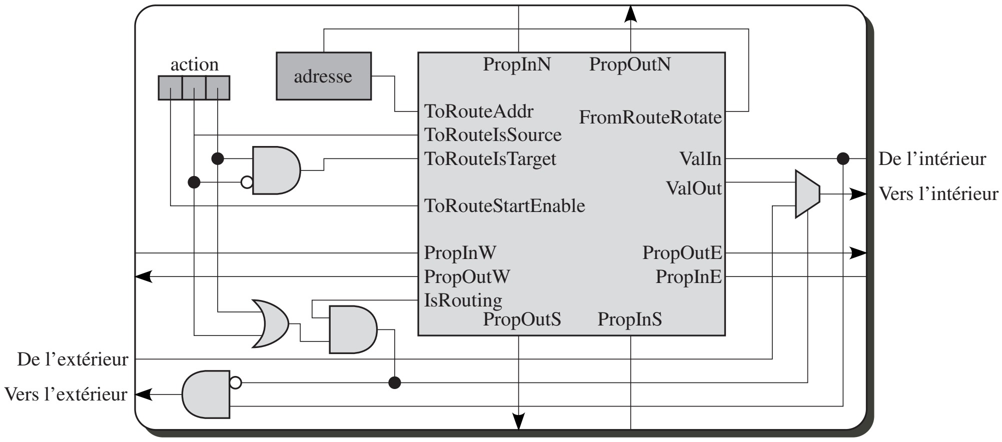


Figure 6.21 : *Une unité de routage du bord, composée d'un contrôleur, de 16 bits d'adresse, et de trois bits d'action.*

6.4.3 Interface molécule/unité de routage

Maintenant que nous avons décortiqué les molécules et les unités de routage, voyons comment les relier ensemble. Étant donné que quatre molécules ont accès à la même unité de routage, une interface est nécessaire, de façon à combiner correctement les différents signaux.

Les entrées/sorties de la molécule ont été présentées dans le tableau 6.8, à la page 188. Le tableau 6.13 montre, quant à lui, les entrées/sorties d'une unité de routage vers son élément externe.

Pour la suite de cette section, nous nommerons les signaux de la i-ème molécule $\texttt{NomDuSignal}_i$, pour plus de clarté. Les signaux en direction des molécules sont simplement identiques à ceux sortant le l'unité de routage. Il est donc du ressort de la molécules de les gérer correctement si elle n'est pas de type Input, Output, ou Trigger. Concernant ceux en sens inverse, la figure 6.22 illustre les opérations appliquées sur les sorties des molécules.

A part `InsideTrigger` et `IsTrigger`, tous les signaux calculés sont transmis à l'unité de routage reliée à l'interface. Concernant le trigger, toutes les unités de routage ont besoin d'un tel signal pour synchroniser les comparaisons d'adresse. Or, disposer d'une molécule Trigger pour chaque unité de routage serait impossible. Nous avons donc inclus, dans l'interface, un mécanisme de propagation du trigger, dans les directions Nord et Est. Chaque interface possède donc, outre des liaisons avec les molécules et les unités de routage, des connexions avec ses quatre voisin. En fonction de ses entrées Sud et Ouest, et du signal `InsideTrigger`, calculé comme indiqué en bas à droite de la figure 6.22, le trigger à transmettre à l'unité de routage ainsi qu'aux interfaces Nord et Est est calculé.

Entrée	bits	Description
`IsSource`	1	Indique si l'unité de routage est une source
`IsTarget`	1	Indique si l'unité de routage est une destination
`Address`	1	Bit d'adresse transmis à l'unité de routage durant la phase de comparaison d'adresse
`ValueIn`	1	Valeur envoyée par une unité de routage source
`StartEnable`	1	Signal forçant la création d'une connexion
`Trigger`	1	Trigger signifiant la fin de la comparaison d'adresse

Sortie	bits	Description
`ValueOut`	1	Valeur transmise à la molécule Input lorsqu'il y en a une
`Shift`	1	Indique aux molécules Input, Output, et Trigger qu'elles doivent décaler leur registre
`IsConnected`	1	Indique si l'unité de routage est connectée à sa correspondante

Tableau 6.13 : *Entrées/Sorties d'une unité de routage, communiquant avec l'interface molécule/unité de routage.*

La figure 6.23 montre le petit dispositif dédié à la propagation du trigger. Avec très peu de logique, il permet de créer des espaces d'influence des molécules Trigger. Dans la figure de gauche, les ronds noirs sont des molécules en mode Trigger, et les flèches représentent la direction de propagation de la valeur de ces triggers. L'avantage de ce système est qu'il est scalable, si l'on place des molécules Trigger de manière pas trop éloignée. Le développeur doit toutefois obligatoirement placer une molécule Trigger dans le coin Sud-Ouest de chacun de ses designs mettant en jeu le routage distribué.

Nous avons fait le choix de propager le trigger dans les interfaces plutôt que directement dans les unités de routage, ceci dans l'optique de rendre le système de routage le plus indépendant possible de l'implémentation des triggers et du registre d'adresse. De plus, la présence de cet interface entre molécules et unités de routage rend aisé la modification du nombre de molécules reliées à une unité de routage. Le code VHDL a été séparé en trois tableaux, contenant respectivement les unités de routage, les interfaces, et les molécules. De ce fait, seul le tableau des interfaces ne nécessiterait une modification dans le cas d'une réutilisation ultérieure du code.

Finalement, nous pouvons noter qu'un conflit peut naître, si plusieurs molécules Input et Output tentent d'accéder à la même unité de routage. Dans un tel cas, la valeur et l'adresse envoyées à cette unité seraient un OU logique entre les sorties des différentes molécules. Il est donc du ressort du développeur de prêter une attention particulière lors du placement des molécules accédant au niveau de routage. Ce choix a été dicté par les contraintes de place sur le silicium. Dans une autre version, il serait évidemment possible de concevoir un système de priorité, qui permettrait d'éviter les contentieux, mais qui nécessiterait un plus grand nombre de portes logiques.

6.5 Le sous-système environnemental

Le sous-système environnemental est décomposé en deux modules, un microprocesseur et un sous-système d'acquisition. Ce dernier s'occupe de récupérer les données des senseurs, et de les transmettre au sous-système organique. Il a ensuite accès aux valeurs calculées par le sous-système organique, dans les cellules, et peut les envoyer

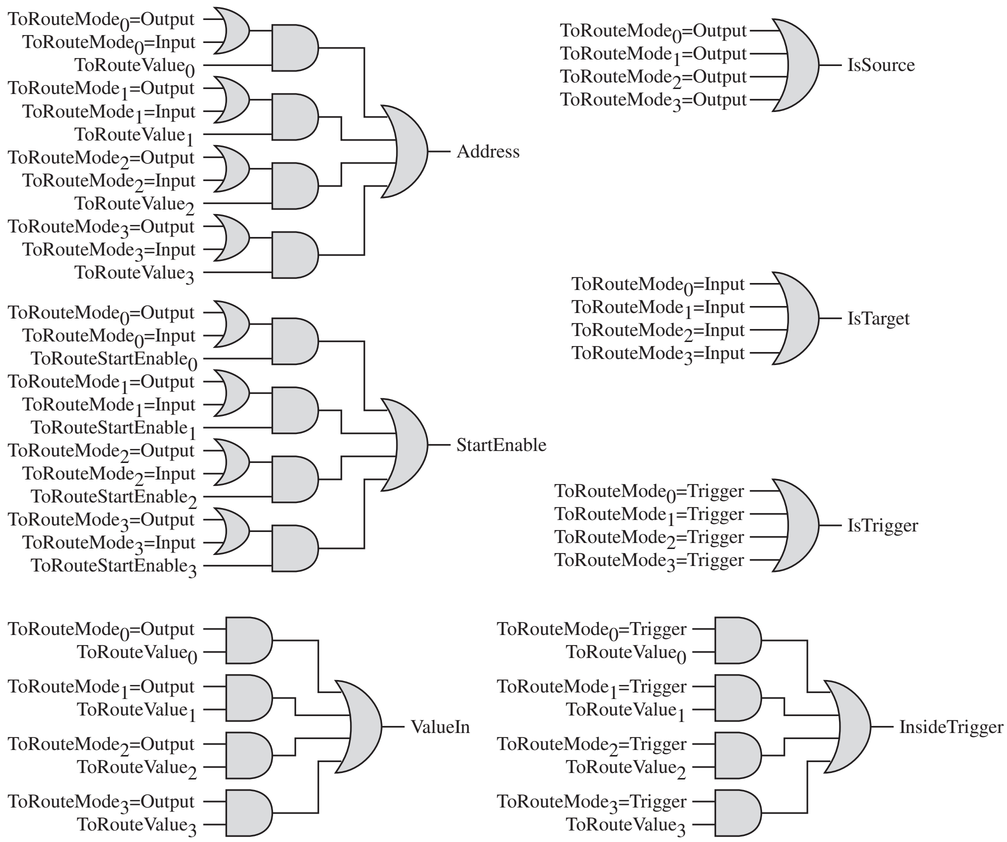


Figure 6.22 : *Les opérations appliquées aux signaux passant de quatre molécules à une unité de routage.*

sur les pins du circuit, afin d'agir sur de potentiels actuateurs. Il possède seize pins bidirectionnels, qui peuvent être utilisés indépendamment en entrée ou en sortie. Chaque ligne peut en outre passer par un registre, et est reliée à une unité de routage semblable à celles présentes dans le sous-système organique. Ces unités de routage, reliées au sous-système organique, permettent donc à ce dernier d'accéder aux entrées/sorties depuis n'importe quelle molécule.

Le microprocesseur, basé sur une architecture RISC 32 bits, possède les caractéristiques suivantes :

- Chaque instruction est exécutée en un coup d'horloge.
- Il contient un pipeline à cinq niveaux : Fetch, Decode, Execute, Memory, et Write-back.
- Les instruction sont codées sur 32 bits.
- Le code d'opération est codé sur 6 bits.
- L'architecture du processeur est basée sur une banque de registres à trois ports, de 32×32 bits, permettant trois accès simultanés, deux en lecture et un en écriture. De ce fait, un seul coup d'horloge est nécessaire à la lecture de deux opérandes et au chargement d'un résultat à une adresse différente des opérandes.
- La communication entre le processeur et la mémoire et/ou ses périphériques s'exécute selon un schéma LOAD/STORE.
- La communication entre le processeur et son environnement est réalisée via une

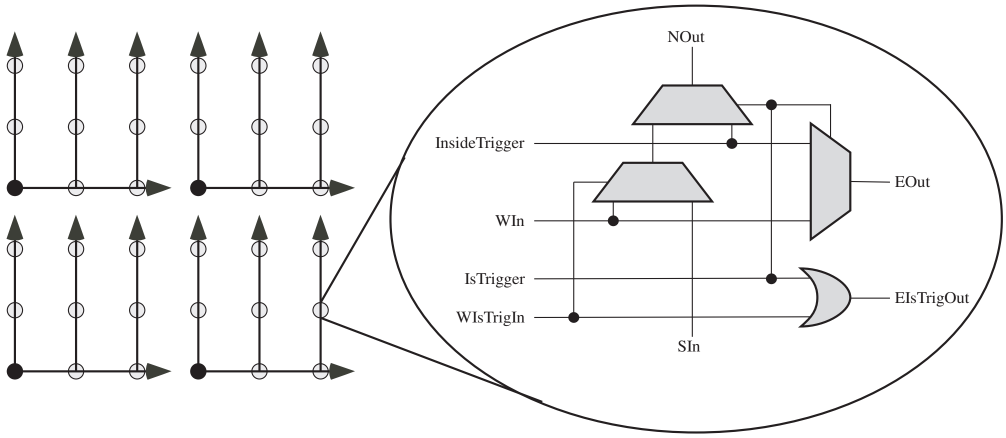


Figure 6.23 : *A droite, l'espace d'influence d'un trigger, et à gauche, l'implémentation du mécanisme de gestion du trigger.*

interface semblable au bus AMBA [13]. Le microprocesseur possède donc un contrôleur AHB interne, ainsi qu'un pont APB, qui permet l'utilisation d'un nombre indéterminé de périphériques via le bus APB.

- De la mémoire peut être ajoutée, en la connectant sur le bus AHB externe. Deux timers à 16 bits et un multiplicateur booth de taille 16×16 ont été intégrés comme périphériques, dans le circuit.
- Jusqu'à 5 sources d'interruption sont gérables par le microprocesseur, une priorité leur étant donnée grâce à un masque d'interruption. Un stack interne dédié aux interruptions a été inclus au microprocesseur, et permet dès lors au séquenceur d'accepter jusqu'à 32 interruptions enchaînées.
- L'ALU du microprocesseur contient un générateur de nombres pseudo-aléatoires, ces derniers étant grandement utilisés par les algorithmes évolutionnistes. Il est implémenté grâce à un Linear Feedback Shift Register de 32 bits, et deux instructions permettent de l'initialiser et de récupérer une valeur pseudo-aléatoire.

La figure 6.24 illustre la structure interne du microprocesseur.

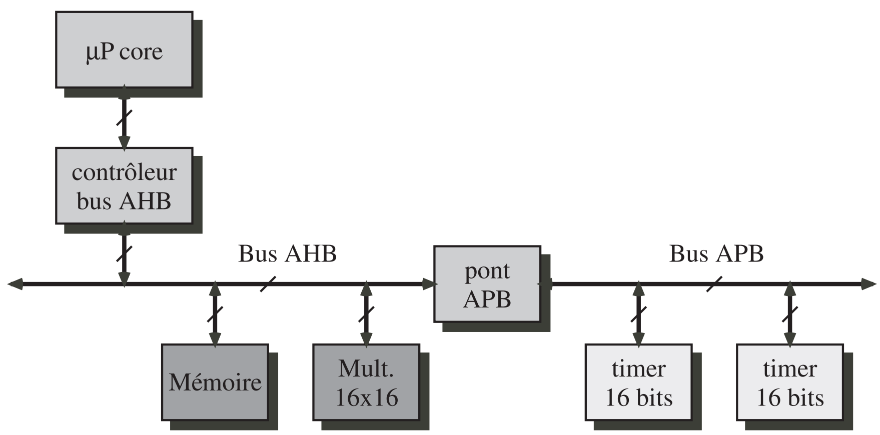


Figure 6.24 : *La structure interne du microprocesseur POEtic.*

6.6 L'interface du système

La communication entre le sous-système environnemental et le sous-système organique est gérée par une interface illustrée par la figure 6.25, que nous appellerons unité de configuration. Elle est accessible via le bus AHB, et tout accès au sous-système organique passe par elle. Elle permet également de connecter plusieurs circuits POEtic, et rend ainsi le système scalable. Cette spécificité est rendue possible par la présence de l'unité de gestion de coordonnées, qui calcule la position du circuit dans une grille de circuits interconnectés.

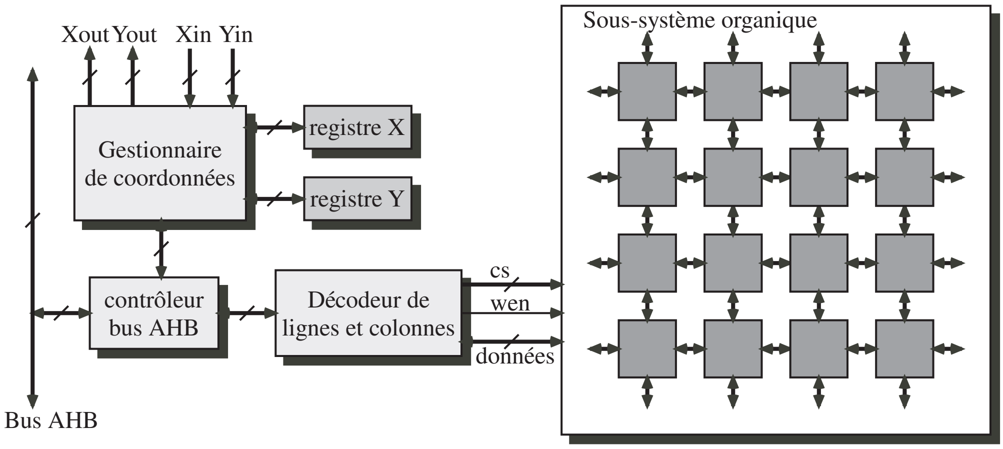


Figure 6.25 : *L'interface des systèmes.*

Le microprocesseur accède au sous-système organique grâce à plusieurs signaux : un chip select (cs), un *write enable* (wen), et les données de configuration. Étant donné qu'une molécule contient 76 bits de configuration, et que le bus de données du processeur est sur 32 bits, trois accès sont nécessaire pour définir l'entièreté d'une molécule. Le signal chip select d'une molécule est donc séparé en trois fils, qui permettent de sélectionner une des trois parties de ses bits de configuration. Le signal wen indique si l'accès se fait en lecture ou en écriture, et le bus de données contient les données à écrire dans la molécule, ou lues depuis cette molécule.

Le décodeur d'adresse permet d'effectuer de la configuration partielle d'un sous-ensemble des molécules. Le microprocesseur définit un masque de lignes et de colonnes, et de ce fait il est possible de configurer simultanément plusieurs molécules, ce qui peut accélérer le processus de configuration lors de l'utilisation d'un système cellulaire où toutes les cellules sont identiques. La figure 6.26 montre deux accès en écriture, où à chaque fois deux colonnes et deux lignes sont sélectionnées. Les quatre molécules présentes aux intersections des lignes et colonnes sélectionnées sont alors modifiées.

Concernant la scalabilité du tissu POEtic, il est nécessaire qu'un grand nombre de cellules soient implémentables. Les coûts de fabrication du circuit ne nous ont permis de réaliser qu'un système composé d'un tableau de 12×12 molécules, ce qui n'est évidemment pas suffisant pour la création d'un système possédant un grand nombre de cellules. L'unité de gestion de coordonnées a donc été prévue dans le but de réaliser une grille de circuits POEtic, afin de disposer d'un tableau reconfigurable plus important. Chaque circuit est connecté à ses quatre voisins, et les sous-systèmes organiques

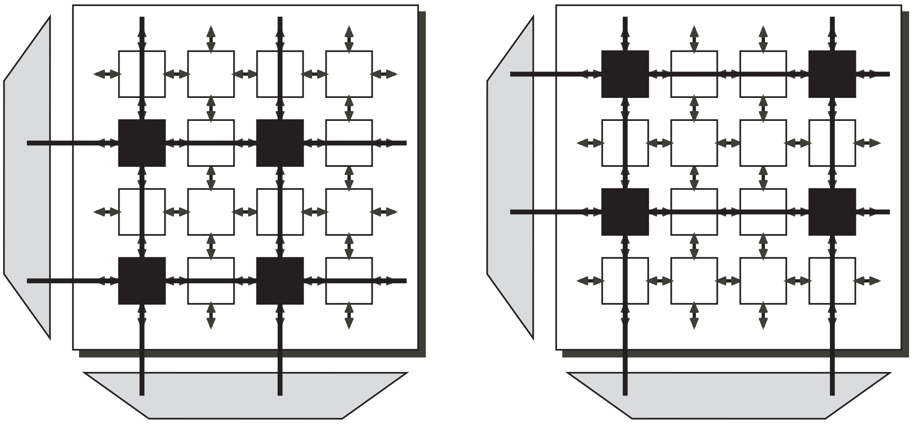


Figure 6.26 : *Configuration simultanée de plusieurs molécules.*

le sont par les unités de routage. De plus, chaque circuit est capable de calculer sa position exacte, grâce aux lignes sérielles Xout, Yout, Xin, et Yin. Lors de la mise en marche d'un tel système, un seul microprocesseur constate qu'il est le maître, et démarre le processus de propagation de coordonnées. Après un certain nombre de coups d'horloge, chaque circuit est identifié de manière unique par sa paire de coordonnées (X,Y). Ces coordonnées définissent ensuite une partie de l'adresse de mémoire réservée pour une unité de configuration donnée. Le processeur maître peut alors accéder à l'ensemble des molécules présentes.

Si nous posons que l'espace d'adressage du bus AHB/APB débute à l'adresse 0x00010000, que le tissu est composé d'un tableau de 16 $\times$ 16 circuits POEtic, et que l'espace d'adressage des molécules est placé à la fin de l'espace mémoire, l'adresse de base de l'espace mémoire d'un sous-système organique est 1_$X_3X_2X_1X_0$_$Y_3Y_2Y_1Y_0$_0000_0000_0000_0000_0000_000, où (X, Y) correspond aux coordonnées du circuit concerné. Lors d'un accès à une molécule, le microprocesseur place son adresse sur le bus AHB, et l'unité de configuration qui détecte la bonne coordonnée (X,Y) autorise l'accès à la molécule correspondante.

6.7 Fabrication du circuit

Un premier circuit de test a tout d'abord été réalisé, afin de valider le microprocesseur, les molécules et les unités de routage. Ce circuit, outre le microprocesseur, contient 12 molécules et 3 unités de routage, regroupés en 3 groupes, visibles au bas de la figure 6.27. Sa taille est de 13 mm^2, pour une technologie CMOS 0.35μm, une couche de polysilicium et 5 couches de métal. Il a servi à valider la conception des éléments principaux du circuit final, qui ont montré un fonctionnement irréprochable.

Le circuit POEtic final a été réalisé physiquement grâce à la même technologie, pour une taille plus imposante : 53.77 mm^2 (7.647 mm $\times$ 7.032 mm). Il n'a pas encore été testé, mais nous pouvons présenter le layout final, à la figure 6.28. Nous y distinguons le microprocesseur, qui occupe toute la largeur du circuit, sur le bas du schéma. Le sous-système organique représente la partie la plus importante, avec ses 144 molécules. Elles sont organisées selon des groupes de 4 molécules (Figure 6.29) reliées à une unité de routage, ces groupes étant arrangés en 9 colonnes de 4 groupes de molécules.

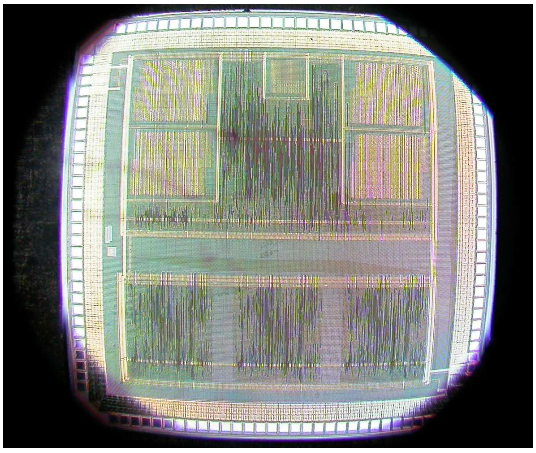


Figure 6.27 : *Photographie du circuit de test.*

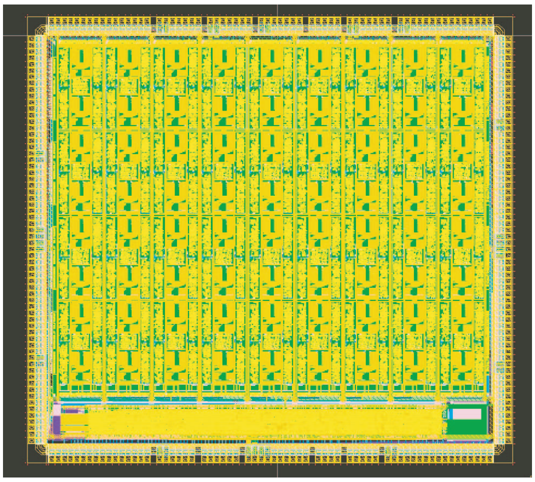


Figure 6.28 : *Le layout du circuit POEtic.*

6.8 Implémentation de composants de base

Après avoir présenté toutes les spécificités du tissu POEtic, nous nous intéressons maintenant à l'implémentation de certains composants de base tels que le registre à décalage ou le compteur. La réalisation de ces éléments peut, suivant les systèmes, être critique en terme de nombre de molécules, c'est pourquoi les caractéristiques spéciales de ces molécules aident à cette optimisation.

6.8.1 Le registre à décalage

L'importance de l'implémentation d'un registre à décalage de manière efficace n'est plus à démontrer. En effet, dans un système bio-inspiré nécessitant un génome présent dans chaque cellule, la place allouée au stockage de celui-ci peut vite prendre des proportions peu réalistes. C'est pourquoi nous présentons ici la meilleure manière d'implémenter un tel registre à décalage.

Comme nous l'avons déjà présenté, un des modes opératoires de la molécule, appelé mode Mémoire, transforme la look-up table en un registre à décalage de 16 bits.

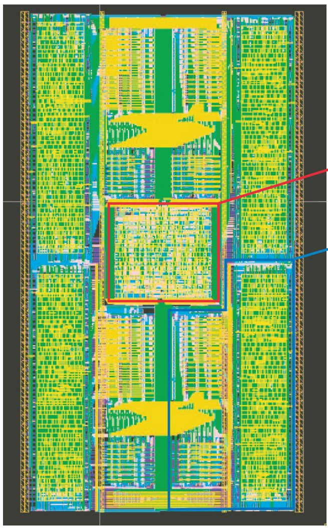


Figure 6.29 : *Un groupe de quatre molécules, et leur unité de routage.*

De plus, la bascule de la molécule peut être accolée à la look-up table, de manière à créer un registre à 17 bits. Il est ensuite possible de chaîner plusieurs molécules afin de créer un registre de $16r0 + 17r1$ bits, où $r0$ et $r1$ sont des nombres entiers positifs. Toutefois, dans le cas où le nombre de bits à stocker est considérable, le nombre de molécules nécessaires peut devenir très important. C'est pourquoi une bonne solution en terme de taille est donnée par la faculté d'auto-configuration partielle des molécules.

En effet, cette auto-configuration se fait de manière sérielle, une molécule injectant les bits de configuration un à un. Notons également qu'un des multiplexeurs d'entrées de la molécule a la possibilité de récupérer le dernier bit de la configuration partielle d'une de ces quatre voisines. Il est donc possible de chaîner plusieurs molécules sur le plan de leurs bits de configuration, en ayant une molécule supplémentaire contrôlant le décalage tout en réinjectant le génome de manière à avoir une mémoire à décalage cyclique.

Une reconfiguration partielle, comme décrite au chapitre 6.3.12, peut agir sur 5 blocs, à savoir la look-up table, les entrées de la molécule, le mode opératoire, le switchbox, et un bloc de bits divers. Il est donc possible de combiner un ou plusieurs de ces blocs afin de créer un grand registre à décalage dans la molécule. Il faut toutefois pouvoir garantir une bonne marche du circuit dans n'importe laquelle des configurations possibles. Pour ce faire, les trois points suivants doivent être respectés :

- Le bloc décrivant le mode opératoire ne peut être utilisé à cette fin, car si une molécule se trouve par hasard dans le mode trigger, input ou output, le routage dynamique a de fortes chances de se trouver paralysé par cette molécule ayant une configuration aléatoire.
- Le bloc décrivant les bits divers ne peut également pas être utilisé, car, lors d'une reconfiguration partielle, il est couplé à la valeur de la bascule D de la molécule. La limitation vient ici du fait que la valeur de cette bascule peut changer lorsque le génome n'est pas décalé. En effet, si les bits stockés dans les bits divers permettent un reset local, un tel reset peut être accidentellement exécuté, et donc changer l'état de la bascule. De plus, par défaut, la bascule se charge à chaque flanc d'horloge, et ce sont les bits divers, qui sont justement incontrôlables en

cas de stockage d'information, qui gèrent le chargement de la bascule.
- Finalement, si les bits contrôlant les multiplexeurs du switchbox servent à stocker de l'information, il est de la plus haute importance de garantir que le switchbox n'est pas impliqué dans du routage intermoléculaire. En effet, ce routage, se trouverait modifié à chaque décalage de la configuration, détruisant les chemins de données existants.

Ces trois limitations définies, nous avons donc trois blocs à dispositions :
- La look-up table, à savoir 16 bits,
- Les entrées de la molécule, soit 14 bits,
- Et le switchbox, fournissant 24 bits.

Il est dès lors possible de construire un registre à décalage de taille $16l+14e+24s$, où $l \in \mathbb{N}$, $e \in \mathbb{N}$, $s \in \mathbb{N}$. Le nombre de molécules nécessaires est alors $max(l,e,s)+1$, le "$+1$" représentant la molécule chargée de la gestion de la configuration. La figure 6.30 montre 5 molécules implémentant un registre à décalage de 160 bits, alors que 10 molécules en mode mémoire seraient nécessaires pour atteindre la même capacité.

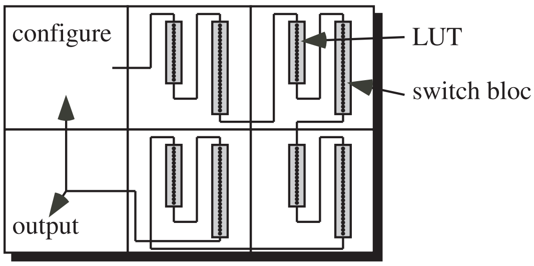


Figure 6.30 : *Un registre à décalage de 160 bits implémenté grâce à 5 molécules.*

N'oublions pas une troisième possibilité de stockage qui est tout simplement la bascule présente dans toute molécule. Elle permet de stocker un bit, ce qui est loin d'être efficace, mais permet de compléter les solutions précédentes, de manière à pouvoir construire un registre à décalage de taille quelconque.

Nous pouvons donc construire un registre à décalage en combinant les trois stockages différents, en essayant de minimiser le nombre de molécules nécessaires de la manière suivante :
- $16r0 + 17r1$ bits, dans $r0 + r1$ molécules en mode mémoire,
- b bits, dans b molécules en mode 3-LUT, où l'on n'utilise que la bascule,
- et $16l + 14e + 24s$ bits, dans $max(l,e,s) + 1$ molécules dont on utilise les bits de configuration.
- Au total, nous pouvons disposer de $16r0 + 17r1 + b + 16l + 14e + 24s$ bits, dans un nombre m de molécules défini ci-dessous :

$$m = \begin{cases} r0 + r1 + b & \text{si } l + e + s = 0 \\ r0 + r1 + b + max(l,e,s) + 1 & \text{sinon} \end{cases}$$

Pour un registre à décalage de taille n, il faut donc trouver une solution à l'équation $16r0 + 17r1 + b + 16l + 14e + 24s = n$, tout en minimisant la valeur de m telle que définie.

6.8.2 Le compteur

Dans la liste des éléments indispensables à grand nombre de designs figure le compteur. Le mode opératoire 3-LUT des molécules a été tout particulièrement conçu afin d'optimiser la réalisation de ce composant. En effet, un compteur binaire de n bits peut être construit avec n molécules, et donc un compteur à x est implémenté avec $\lceil \log_2(x) \rceil$ molécules.

Une molécule en mode 3-LUT est décomposée en deux look-up tables, la première servant au calcul du bit courant, la deuxième calculant le bit de retenue. Cette retenue est directement transmise à la molécule voisine immédiate au sud, sans le besoin de passer par un switchbox. De ce fait, la création d'un compteur en utilisant une colonne de molécules est optimale, étant donné qu'aucun switchbox n'est nécessaire. La figure 6.31 présente un compteur-trigger à 4 bits, qui, si nous modifions le contenu des LUTs, peut agir comme un compteur binaire standard.

Notons que la réalisation d'un compteur incrémenteur ou décrémenteur ne se différencie que par les valeurs présentes dans leurs LUTs.

6.8.3 Le compteur-trigger

Un compteur standard, comme présenté précédemment, permet l'accès à sa valeur en tout temps. Toutefois, certains systèmes ne nécessitent que le comptage d'un certain nombre fixe de coups d'horloge, ou d'événements. Dans ce cas il n'est pas indispensable d'avoir accès à tous les bits du compteur, mais seulement à un signal qui passe à '1' tous les n événements, en fonction d'un *enable* d'entrée. Nous appellerons un tel compteur un compteur-trigger, que nous abrégerons CT.

Les CTs, contrairement aux compteurs nécessitant un accès à la valeur courante, peuvent être optimisés, en prenant en compte les caractéristiques spéciales des molécules. En effet, puisqu'ici il s'agit d'avoir un signal qui passe à '1' durant un coup d'horloge lorsque le compteur atteint une certaine valeur, certaines optimisations peuvent se faire. Outre le compteur binaire, un registre à décalage peut implémenter un CT, de même qu'une combinaison de plusieurs CTs. Nous allons présenter les trois approches avant de les comparer, afin de pouvoir déterminer, pour n'importe quelle valeur de CT, la meilleure option.

Implémentation : compteur binaire

La première option consiste en l'implémentation d'un compteur binaire. Nous proposons de le réaliser grâce à un décompteur initialisé au reset à la valeur $CTVal - 1$. Il suffit alors de détecter le passage de '0' à '1' du bit de poids fort qui interviendra après $CTVal$ *enables*, et d'utiliser ce signal pour réinitialiser la valeur du compteur à $CTVal - 1$. La figure 6.31 montre un CT binaire de 4 bits, et le tableau 6.14 défini le contenu de chacune des LUTs.

Pour un CT de n bits, la première LUT de la molécule implémentant le bit de poids faible est de type 0, la première LUT de la molécule de poids fort est de type 3, et toutes les autres premières LUTs sont de type 2. La deuxième LUT de la molécule de poids fort est de type 4, et toutes les autres deuxièmes LUTs sont de type 1. Les valeurs de reset des différentes molécules définissent la valeur du compteur. Par exemple, pour un compteur à 13, les molécules doivent être initialisées à 12, et donc les valeurs de reset sont 1100.

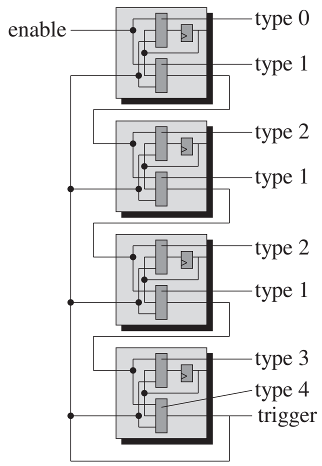


Figure 6.31 : *Un compteur-trigger binaire à 4 bits, implémenté dans 4 molécules.*

Une compteur-trigger de valeur CTVal est implémenté, conformément à l'explication de la section 6.8.2, à l'aide de $\lceil \log_2(CTVal) \rceil$ molécules.

Inputs	Type 0	Type 1	Type 2	Type 3	Type 4
000	1	0	1	Rst_value	1
001	1	1	0	0	0
010	Rst_value	0	Rst_value	Rst_value	1
011	Rst_value	1	Rst_value	0	0
100	0	1	0	0	0
101	0	1	1	1	0
110	Rst_value	1	Rst_value	0	0
111	Rst_value	1	Rst_value	1	0

Tableau 6.14 : *Valeurs des LUTs en fonction du type d'implémentation, pour un compteur-trigger décrémenteur*

Implémentation : registre à décalage

Comme nous l'avons vu à la section 6.8.1, les molécules permettent la réalisation efficace de registres à décalage. En plaçant dans le registre un ou plusieurs '1' séparés par un nombre égal de '0', nous pouvons construire un compteur-trigger, comme le montre la figure 6.30, où il suffit de placer un des bits à '1', tous les autres étant mis à '0'. Un compteur-trigger à 16 peut effectivement être simplement implémenté grâce à une molécule en mode mémoire, ce qui économise 3 molécules par rapport à une implémentation par compteur. De plus, un CT à 256 peut être réalisé grâce à 2 molécules en mode mémoire et une molécule en mode 3-LUT, ce qui économise 5 molécules par rapport à une implémentation par compteur.

Avec cette solution, un compteur-trigger de CTVal=$16r0+17r1+b+16l+14e+24s$ nécessite un registre à décalage de taille n, qui, comme nous l'avons déjà montré, est implémenté à l'aide du nombre suivant de molécules :

$$m = \begin{cases} r0 + r1 + b & \text{si } l + e + s = 0 \\ r0 + r1 + b + max(l, e, s) + 1 & \text{sinon} \end{cases}$$

Nous pouvons également observer qu'un CT de CTVal quelconque peut être réalisé avec un registre à décalage de taille $d{*}CTVal, d \in \mathbb{N}$, en plaçant un '1' tous les CTVal emplacements. De cette manière, un compteur-trigger de CTVal=$(16r0 + 17r1 + b + 16x + 14y + 24z)/d$ peut être implémenté avec le nombre suivant de molécules :

$$m = \begin{cases} d \cdot (r0 + r1 + b) & \text{si } l + e + s = 0 \\ d \cdot (r0 + r1 + b + max(l, e, s)) + 1 & \text{sinon} \end{cases}$$

Par exemple, un compteur-trigger à 7 est réalisé avec 2 molécules, en utilisant les bits de configuration des entrées d'une molécule (14), et une molécule gérant la configuration, ce qui est plus efficace qu'une implémentation faite avec une compteur binaire, qui nécessite 3 molécules.

Implémentation : combinaison

Les deux premières méthodes sont pour ainsi dire triviales, alors que la troisième offre d'intéressantes possibilités d'optimisation. En effet, un compteur binaire ou un registre à décalage peuvent être considérés comme des compteur-triggers, mais pour certains entiers factorisables, une combinaison des deux, ou de plusieurs registres à décalages peut s'avérer nettement plus efficace en terme d'espace.

Le compteur-trigger a, comme nous l'avons déjà mentionné, un *enable*, qui peut tout-à-fait provenir d'un autre de ces CT. Nous pouvons dès lors chaîner plusieurs CTs afin d'en créer un nouveau. Si nous chaînons n CTs, ayant chacun pour valeur $CTVal_i, i \in [1, n]$, alors nous créons un CT de valeur $CTVal = \prod_{i=1}^{n} CTVal_i$. Il faut toutefois être vigilant, et ne pas simplement chaîner sans autre les CTs. En effet, le n-ième CT ne doit être activé que si l'*enable* d'entrée, de même que les sorties de tous les CTs d'index inférieur est activé. Pour ce faire, des molécules en mode 3-LUT sont connectées aux différents CTs comme indiqué à la figure 6.32. La première LUT est chargée de détecter que les deux premières entrées sont à '1', et contient donc la valeur 10001000. La deuxième LUT, elle, est en charge de détecter que les trois entrées sont à '1', et contient donc la valeur 10000000.

Afin de calculer l'espace requis par un CT construit en combinaison, en terme de molécules, il faut tout d'abord savoir si une telle décomposition est possible. Pour ce faire, il suffit de trouver tous les facteurs premiers de notre nombre n, puis de comparer les tailles d'implémentation de toutes les combinaisons possibles de ces décompositions. En d'autre termes, il faut évaluer la taille de toutes les décompositions satisfaisant la condition suivante : $n = \prod_{i=1}^{j} CTVal_i, j \in [2, n/2], CTVal_i \in [2, n/2]$.

Conformément à la figure 6.32, le nombre de molécules nécessaires à la réalisation d'un CT de CTVal= $\prod_{i=1}^{nbcomb} CTVal_i$ est :

$$m = \sum_{i=1}^{nbcomb} nbmol(comb_i) + \lceil (nbcomb - 1)/2 \rceil$$

Où $nbcomb$ est le nombre de CT combinés, et $nbmol(comb_i)$ est le nombre de molécules nécessaires à l'implémentation du i-ème CT. Le terme $\lceil (nbcomb - 1)/2 \rceil$ se justifie de la manière suivante : Pour une combinaison de $nbcomb$ CTs, il faut

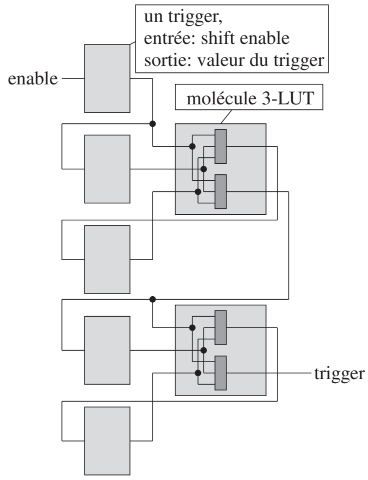


Figure 6.32 : *Un compteur-trigger obtenu par combinaison de 4 compteur-triggers.*

$nbcomb - 1$ look-up tables. Étant donné qu'une molécule en mode 3-LUT contient 2 look-up tables, $\lceil (nbcomb - 1)/2 \rceil$ molécules sont nécessaires à l'implémentation de $nbcomb - 1$ look-up tables.

Théorème 1. *Un compteur-trigger optimal obtenu par combinaison contient au maximum un compteur-trigger binaire.*

Démonstration. Supposons que nous disposons de deux compteur-triggers implémentés par des compteurs binaires de valeurs $c1$ et $c2$. Le nombre de molécules nécessaires à chacun d'eux est respectivement $\lceil \log_2(c1) \rceil$ et $\lceil \log_2(c2) \rceil$. De plus, une molécule en mode 3-LUT au moins est nécessaire à tout CT combinaison. Le nombre total de molécules est donc : $nb1 = \lceil \log_2(c1) \rceil + \lceil \log_2(c2) \rceil + 1$.

La réalisation d'un CT avec un seul compteur binaire de $c1 + c2$ nécessite $nbtot = \lceil \log_2(c1 + c2) \rceil$ molécules.

Il faut donc vérifier que $nbtot \leq nb1$.

Nous avons $c1 \geq 2$ et $c2 \geq 2$.

$\Rightarrow c1 \cdot c2 \geq c1 + c2$

$\Rightarrow 2 \cdot c1 \cdot c2 \geq 2(c1 + c2)$

$\Rightarrow 2^{\log_2(c1)+\log_2(c2)+1} \geq 2^{\log_2(c1+c2)+1}$

$\Rightarrow \log_2(c1) + \log_2(c2) + 1 \geq \log_2(c1 + c2) + 1$

Or, $\log_2(c1) + \log_2(c2) + 1 \leq \lceil \log_2(c1) \rceil + \lceil \log_2(c2) \rceil + 1$

Et $\log_2(c1 + c2) + 1 \geq \lceil \log_2(c1 + c2) \rceil$

Donc $\lceil \log_2(c1) \rceil + \lceil \log_2(c2) \rceil + 1 \geq \lceil \log_2(c1 + c2) \rceil$

La présente démonstration est similaire pour un nombre plus grand de CT compteurs. Si un CT en combinaison contient plus d'un compteur binaire, ces compteurs peuvent donc être fusionnés en un seul compteur binaire, de manière à réduire la taille du CT global. □

Résumé

Nous avons donc 3 façons de créer un compteur-trigger :

- Un compteur binaire.
- Un registre à décalage contenant un ou plusieurs '1'.

- Une combinaison d'un compteur binaire et d'un ou plusieurs registres à décalage.

L'encadré ci-dessous résume ces trois manières d'implémenter un compteur-trigger, et le nombre de molécules nécessaires pour chacune. Lors de la réalisation d'un de ces composants, il suffit de trouver une solution qui satisfasse la valeur du compteur, tout en minimisant le nombre de molécules nécessaire.

Compteur binaire

Valeur $= n$

Nb de molécules $= \lceil \log_2(n) \rceil$

Registre à décalage

Valeur $= (16r0 + 17r1 + b + 16l + 14e + 24s)/d$

$$\text{Nb de molécules} = \begin{cases} d \cdot (r0 + r1 + b) & \text{si } l + e + s = 0 \\ d \cdot (r0 + r1 + b + max(l, e, s)) + 1 & \text{sinon} \end{cases}$$

Combinaison

$$\text{Valeur} = \prod_{i=1}^{nbcomb} CTVal_i$$

$$\text{Nb de molécules} = \sum_{i=1}^{nbcomb} nbmol(comb_i) + \lceil (nbcomb - 1)/2 \rceil$$

6.9 Les outils de développement

Un circuit reconfigurable n'est rien s'il n'est pas possible d'y implémenter des applications. Pour ce faire, un logiciel de développement appelé POEticMol a été créé par nos soins, afin de permettre à un utilisateur de concevoir un design spécifiquement pour le tissu POEtic, et de le tester.

6.9.1 Design

Lors de la phase de design, POEticMol offre une interface graphique montrant la configuration des molécules du circuit (Figure 6.33). La fonctionnalité exacte d'une molécule peut être définie grâce à une fenêtre de dialogue atteignable en cliquant sur la molécule. Cette fenêtre est séparée en sections correspondant aux cinq blocs de bits de configuration, de manière à en faciliter l'usage.

Concernant le routage intermoléculaire, un algorithme de recherche du plus court chemin du type Lee a été implémenté en logiciel pour permettre à l'utilisateur de créer un chemin de donnée en cliquant simplement sur une source et une destination. L'entrée qui a été sélectionnée dans la molécule destination mémorise la sortie correspondante de la molécule source, et de cette manière un routage intermoléculaire peut être relancé après avoir déplacé des molécules, ou supprimé des chemins.

Diverses fonctionnalités ont été ajoutées au logiciel pour en faciliter l'utilisation, dont les suivantes :

- Il est possible de grouper des molécules et d'afficher les différents groupes grâce à des couleurs distinctes.
- La sélection de molécules grâce au pointeur de la souris permet de les déplacer dans le tableau, sans en écraser d'autres déjà définies.
- Le copier-coller permet de récupérer une partie de schéma et de le dupliquer ou de l'inclure dans un autre design (l'application gère plusieurs fichiers ouverts).

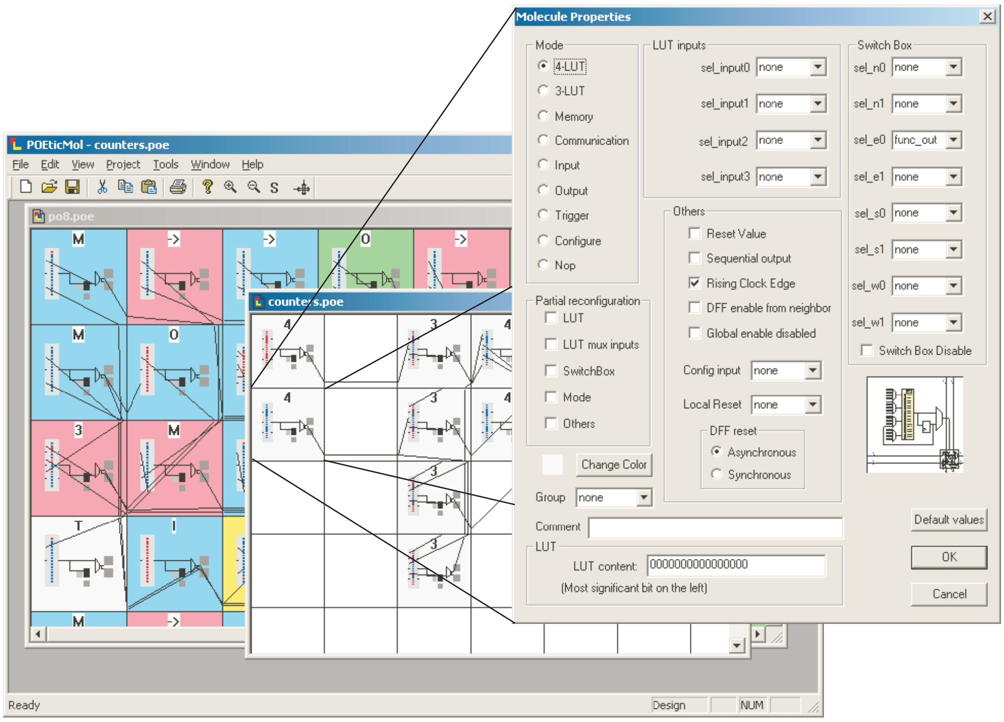


Figure 6.33 : *L'interface graphique de POEticMol.*

Une fois un design entièrement réalisé, la traduction en un fichier de bits de configuration est immédiate, étant donné que l'utilisateur a exactement spécifié la fonctionnalité des molécules.

6.9.2 Simulation

Lorsqu'un design a été réalisé, il est possible d'en simuler le fonctionnement. Pour ce faire, le code VHDL décrivant exactement le sous-système organique est chargé dans Modelsim, un logiciel de simulation. Nous y exploitons le Foreign Language Interface de Modelsim, de par lequel il est possible d'implémenter un composant avec une DLL windows. Cette DLL est un fichier contenant du code compilé offrant à Modelsim des fonctions à appeler au démarrage et lorsque certains signaux changent d'état. Cette DLL, que nous avons développée en C++ permet dès lors de récupérer la valeur de n'importe lequel des signaux présents dans le design simulé, ainsi que de forcer leur valeur.

Notre DLL, appelée POEtic_vhdl.dll, crée un fichier pipe avec POEticMol. Il leur sert de moyen de communication, POEticMol pouvant ordonner un nombre de coups d'horloge à exécuter, et la DLL renvoyant l'état des molécules et des unités de routage. Une deuxième DLL, également chargée par Modelsim, POEtic_IO.dll permet de forcer les entrées du sous-système organique, et de récupérer ses sorties. Par défaut, elle place toutes les entrées à zéro et n'exécute aucune tâche. L'utilisateur n'a qu'à en modifier le code, grâce à des fonctions de haut niveau fournies, pour interfacer avec n'importe quel programme. Dans l'exemple de la figure 6.34, un simulateur de robot Khepera est connecté à la DLL via un pipe, et peut donc envoyer la valeur de capteurs au tissu POEtic et récupérer des données afin d'agir sur les roues du robot.

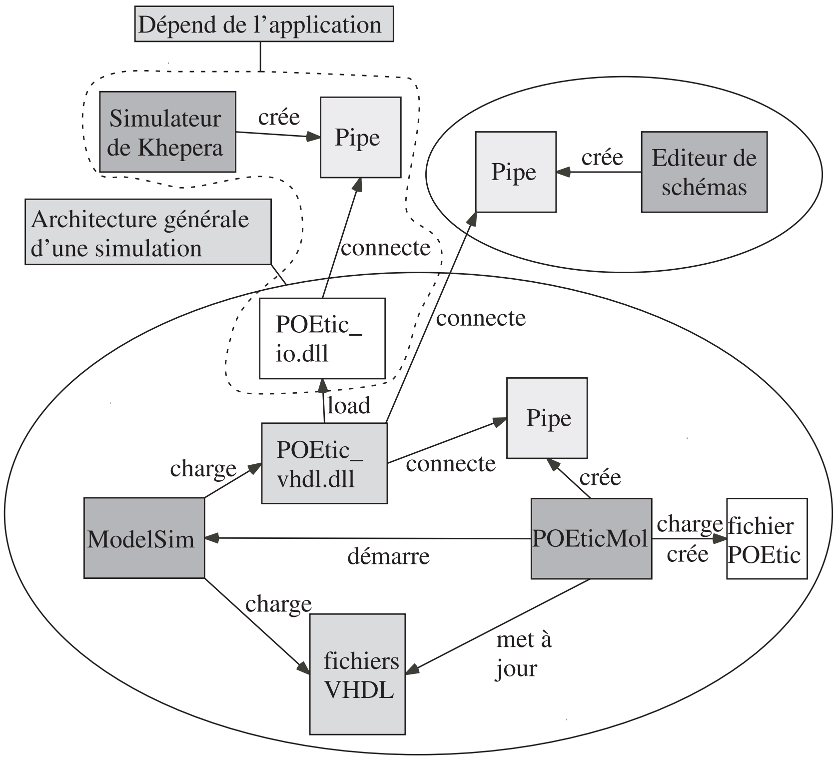


Figure 6.34 : *Réseau d'applications pour la simulation d'un design avec POEticMol.*

L'avantage de notre approche sur l'intégration de tout le système dans POEticMol réside dans la validité du VHDL. Ce dernier a été utilisé pour la réalisation du layout du circuit final, et correspond donc parfaitement à ces spécifications. De plus, durant toute la phase de développement des molécules et du routage, notre outil a été de grande utilité pour tester le VHDL.

Finalement, un outil d'édition de schémas a été développé par l'équipe de York. Il permet de créer un design au niveau des composants tels que compteurs, triggers, ou portes logiques, et de les relier entre eux. Ce logiciel peut ensuite synthétiser et placer/router le design sur les molécules et importe le résultat dans POEticMol. Lors d'une simulation il est alors possible de visualiser simultanément le résultat dans les molécules et dans l'éditeur de schémas (Figure 6.34).

Sur le plan de la programmation du processeur, un assembleur et un compilateur C, dérivé du meta-compilateur LCC [88], ont été réalisé. De plus, un émulateur a été mis au point, de manière à pouvoir tester un programme écrit pour POEtic. La suite d'outils est donc complète, est permet à un utilisateur de prendre en main aisément toutes les phases de conception d'un système à implémenter sur POEtic. Le lecteur désirant plus d'information sur ces différents outils pourra en trouver une description plus approfondie dans [232].

6.10 Conclusion

La partie reconfigurable du circuit POEtic ayant été définie, nous désirons la comparer aux approches d'autres travaux, au niveau de l'auto-configuration et du routage dynamique. Nous proposerons ensuite des améliorations possibles, qui pourraient servir lors de la réalisation d'un nouveau circuit POEtic.

6.10.1 Comparaison

Le circuit POEtic, tel que présenté, offre des caractéristiques d'auto-configuration et de routage dynamique, caractéristiques que les FPGAs commerciaux n'offrent pas encore. Concernant la première, d'autres équipes ont créé des FPGAs de ce type, basées sur des contextes multiples. Alors que dans un FPGA standard il n'existe qu'une configuration chargée, dans un FPGA multi-contexte, plusieurs configurations peuvent être présentes dans le circuit, et une seule d'entre elles est active à un instant donné. L'avantage y est de pouvoir changer de contexte de façon très rapide, en un temps situé entre 5 et 100 nanosecondes, alors que le chargement de la configuration d'un FPGA standard est de l'ordre de la milliseconde.

Dans certains de ces FPGAs multi-contextes, [170], [207], et [243], les éléments logiques ont la possibilité d'accéder aux contextes inactifs et de les modifier, en effectuant des écritures comme s'il s'agissait de RAM. Le désavantage de cette approche est toutefois la quantité de logique nécessaire au stockage et à la gestion de plusieurs contextes. De plus, l'accès aux bits de configuration des contextes n'est pas trivial, et nécessite de nombreuses ressources en terme d'éléments logiques du contexte actif, qui doit effectuer des accès mémoire parallèles. Leur utilité a toutefois été démontrée par Sidhu [215], qui tire profit des multi-contextes pour accélérer des applications de programmation génétique.

Notre approche est toute différente, nos molécules ne possédant qu'un seul contexte. L'auto-configuration s'effectue durant le fonctionnement du circuit, et l'utilisateur doit faire attention à ce qu'aucune molécule ne se trouve dans un état non désiré à la fin d'une configuration partielle, sans quoi le comportement du système peut se trouver perturbé (par exemple par un Reset du routage intercellulaire). L'avantage de notre implémentation tient en la facilité d'accès des bits de configuration. Une seule molécule, en mode Configure, peut accéder aux bits de configuration d'un nombre quelconque de voisines, et ce au travers d'un registre à décalage. Dans le cadre de systèmes POEtic, il est alors aisé, par un mécanisme ne nécessitant que peu de molécules, de recopier l'entièreté d'une cellule dans un espace libre où les bits de configuration gérant la configuration partiel ont été définis de manière identique que dans la cellule source. La reconfiguration partielle offre également des perspectives dans le contexte de l'autoréparation, où une cellule défectueuse dont la partie fonctionnelle serait tripliquée pourrait reconfigurer une cellule inutilisée avec une de ses parties fonctionnelles valides.

Concernant le routage distribué, le circuit POEtic est, à notre connaissance, le seul à proposer une approche basée sur des adresses. Une équipe japonaise a présenté en 2001 la réalisation d'un VLSI implémentant une Plastic Cell Architecture [127]. Leur circuit est également composé de deux plans, dont un contient un tableau de Plastic Part (PP), qui ne sont autre que des éléments logiques, alors que l'autre prend en charge un mécanisme de routage dynamique, par le biais de Built-in Parts (BP). La granularité des PPs y est nettement plus grosse que dans notre implémentation, puisqu'un PP contient 8×8 cellules, où chaque cellule est composée de quatre 2-LUTs. Aucune bascule n'est présente dans les PPs, et donc tout design y étant implémenté est asynchrone, étant donné qu'il n'existe pas d'horloge globale. Les LUTs peuvent alors servir à réaliser des latchs ou des éléments Muller (cellule Muller-C). Le niveau de routage dynamique est également asynchrone, et utilise des techniques de Wormhole [177] pour faire transiter des paquets d'information entre différentes parties du circuit. Le routage s'effectue sur la base de l'adresse de destination du message, qui correspond aux coordonnées X et Y de la cellule destinataire.

L'avantage de leur approche est évidemment la scalabilité, puisque tout y est asynchrone. De plus, les problèmes de congestion que nous pouvons rencontrer dans le circuit POEtic sont évités grâce au routage dynamique de type wormhole. En revanche, la taille de leurs BPs est nettement supérieure à celle de nos unités de routage, et ce d'un facteur 10. Et l'asynchronisme, qui est un atout en ce qui concerne la scalabilité, force la réalisation de systèmes plus complexes que les systèmes synchrones.

6.10.2 Améliorations

Comme nous l'avons déjà signalé, le circuit POEtic a été physiquement réalisé, et certaines applications ont montré quelques-unes de ses imperfections, que nous avons mentionnées au cours de ce chapitre. Nous ne reviendrons pas sur ces modifications mineures, mais allons plutôt proposer des améliorations majeures qui pourraient être faites.

Premièrement, la scalabilité est un des problèmes de notre circuit. Si plusieurs circuits doivent être reliés entre eux en un super-circuit, la fréquence d'horloge faisant fonctionner le sous-système organique doit être baissée, de manière à garantir que les signaux combinatoires aient le temps de parcourir plusieurs centimètres avant la mise-à-jour des bascules présentes aussi bien dans les molécules que dans les unités de routage. Étant donné que la communication intermoléculaire ne peut traverser les frontières du circuit, c'est à la communication intercellulaire d'offrir une meilleure scalabilité. Il serait dès lors possible d'utiliser des unités de routage de type HIDRA-L, qui ne fonctionnent que grâce à des communications locales. Une approche totalement asynchrone de ces unités de routage serait également envisageable, pour autant que le coût matériel ne soit pas trop important, en terme de nombre de transistors et de nombre de liaisons entre unités de routage.

Deuxièmement, sur le plan de la reconfiguration partielle, sur les 76 bits de configuration, 8 ne peuvent être modifiés par ce biais-là. Il sont en effet indispensables au guidage des bits de configuration, en définissant d'où une configuration est acceptée, et quels sont les bits touchés. Un mécanisme semblable à l'algorithme du petit poucet, développé au Laboratoire de Systèmes Logiques [143], pourrait autoriser la modification de tous les bits de configuration d'une molécule, et donc il serait possible d'y construire des cellules entièrement duplicables. Durant la phase d'exploration du projet une telle approche a traversé les esprits, mais la quantité de matériel qu'il aurait fallu ajouter aux molécules n'aurait pas été acceptable.

Finalement, il serait intéressant de disposer d'autoréparation au niveau moléculaire. Dans la réalisation actuelle, une cellule doit implémenter elle-même des mécanismes d'autoréparation, alors qu'il serait nettement plus élégant que les molécules soient capables d'en gérer elles-mêmes. Le problème vient ici de la complexité des molécules. Dans le projet Embryonique, les molécules possédaient de telles caractéristiques, au prix d'un ajout de logique non négligeable. Nos molécules étant nettement plus complexes que celles d'Embryonique, qui, nous le rappelons, ne sont composées que d'un multiplexeur et d'une bascule, la quantité de logique risquerait fort d'exploser.