Chapitre 7

Mécanismes POE

> La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi. Ici, nous avons réuni théorie et pratique : Rien ne fonctionne... et personne ne sait pourquoi !
>
> Albert EINSTEIN

LE SOUS-SYSTÈME organique, comme nous l'avons vu dans le chapitre précédent, inclut des caractéristiques non présentes dans les circuits programmables actuels, et qui ont pour but de faciliter l'implémentation matérielle de mécanismes bio-inspirés :

- Premièrement, la reconfiguration partielle des molécules permet une grande plasticité du circuit. Les molécules peuvent servir à stocker de l'information dans leurs bits de configuration, ou leur fonctionnalité peut être modifiée durant le fonctionnement du circuit, par d'autres molécules.
- Deuxièmement, le niveau de routage dynamique permet de créer des connexions entre différents endroits du circuit, typiquement entre des cellules. Cette nouvelle caractéristique va être mise à profit par des systèmes nécessitant une plasticité au niveau des connexions.
- Troisièmement, son implémentation n'offre aucun moyen de créer des courts-circuits, et rend le circuit sûr dans le cas de bits de configuration générés aléatoirement.
- Quatrièmement, nous connaissons en détail les bits de configuration du circuit POEtic, et pouvons en conséquence les faire évoluer dans le cadre d'applications de matériel évolutif.

En outre, le microprocesseur présent sur le circuit offre la possibilité d'exécuter des algorithmes évolutionnistes de façon rapide, et de pouvoir évaluer les individus en configurant le sous-système organique. Cette configuration est très rapide, puisqu'elle se fait par lot de 32 bits en parallèle.

A l'heure de l'écriture de ces quelques lignes, le circuit final n'a pas encore pu être testé, mais ce que nous présentons dans ce chapitre a été développé grâce au simulateur du sous-système organique. Seul le prototype PO a vu une réalisation sur un FPGA, 80 molécules, leurs unités de routage, ainsi que le processeur ayant été placés sur un circuit VirtexII de Xilinx. Ce prototype n'a pas été utilisé pour les autres mécanismes, car la simulation est basée sur le VHDL décrivant exactement le circuit final, et que le comportement des molécules est nettement plus facilement observé au travers d'une interface graphique qu'au travers d'un port série reliant la carte de la VirtexII à un PC.

Dans ce chapitre, nous allons donc présenter différents mécanismes, principalement ontogénétiques, qui pourront être utiles à l'implémentation de systèmes bio-inspirés sur le circuit POEtic. Nous mentionnerons brièvement les systèmes développés par nos partenaires européens, à savoir un neurone à impulsion, des mécanismes d'autoréparation, et une application de synthèse vocale. Chacun des exemples que nous allons présenter, et qui sont résumés dans le tableau 7.1, tire parti d'au moins une des caractéristiques de POEtic qui n'est pas offerte par les FPGAs commerciaux.

Type	**Exemple**	**Configuration partielle**	**Routage distribué**	**Non courts-circuits**	**Bits de configuration connus**	**Microprocesseur onchip**	**Section**
O	Autoréplication	X	X		X		7.1
O	Développement	X	X		X		7.2
PO	Prototype PO	X	X		X	X	7.3
P	Matériel évolutif			X	X	X	7.4
O	Autoréparation	X	X				7.5.1
P	Synthèse vocale		X			X	7.5.2
E	Neurone		X				7.5.3

Tableau 7.1 : *Exemples d'utilisation du circuit POEtic, indiquant les caractéristiques exploitées.*

7.1 Autoréplication

Un des axes de recherche concernant les systèmes bio-inspirés consiste en l'autoréplication, où un système évoluant sur un substrat particulier doit pouvoir y créer une copie de lui-même. Ce principe semble fondamental dans l'élaboration des mécanismes de la vie artificielle, étant donné qu'il est à la base des organismes biologiques, dans lesquels les cellules sont capables de se diviser de manière autonome. Ce furent les automates cellulaires qui servirent de premiers substrats à l'autoréplication, avec les travaux de Von Neumann sur son constructeur universel [247], puis vingt ans plus tard avec les boucles autoréplicatives de Langton [136]. Plus récemment, l'algorithme du petit Poucet [143] constitue une implémentation matérielle d'un système autoréplicatif capable d'embarquer une fonctionnalité.

Notre circuit POEtic, de par ses capacités d'auto-configuration partielle, permet d'implémenter des mécanismes approchants. Nous avons conçu une cellule composée de deux sous-systèmes : une partie fonctionnelle (PF), et une partie responsable de l'autoréplication (PA). Il est alors possible, si un autre composant PA est placé sur le substrat, de dupliquer la partie fonctionnelle de la cellule, et ce, en sauvegardant l'état de ses bascules.

Comme le montre la figure 7.1, la partie PA nécessite 15 molécules plus une molécule Trigger, tandis que la PF dépend de l'application considérée (dans l'exemple, nous avons 16 molécules). La partie fonctionnelle peut être quelconque, mais ses 8 bits de configuration fixes doivent être ajustés de manière à créer un chemin traversant toute la PF au niveau des bits de configuration, comme montré sur la figure. Le dernier bit de configuration de l'ensemble des molécules est récupéré par une des molécules de la PA, et peut être directement réinjecté par une molécule en mode Configure. En effet, pour qu'il y ait duplication, il est nécessaire qu'après cette action, la cellule de départ retrouve son état précédent. Elle va donc, durant tout le processus de transmission des bits de configuration, créer un registre à décalage bouclant sur lui-même, de façon à ce que sa partie fonctionnelle retrouve son état inchangé. La cellule qui se crée, quant à elle, va utiliser les bits reçus et directement les injecter dans sa PF.

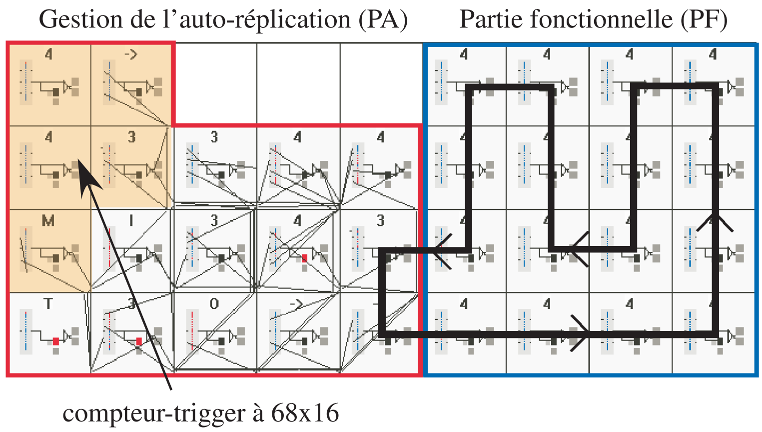

Figure 7.1 : *Une cellule capable de dupliquer sa partie fonctionnelle.*

Le mécanisme d'autoréplication est relativement simple, et exploite l'auto-configuration des molécules, ainsi que le routage distribué. Une cellule désirant s'auto-répliquer lance un routage, à partir d'une molécule Output, dont l'identifiant est 11...1. La réussite d'une duplication ne peut se faire que s'il existe quelque part sur le substrat une PA en attente, avec à ses côtés, des molécules inutilisées dont les 8 bits de configuration fixes sont identiques à ceux de la PF de la cellule de départ (gauche de la figure 7.2). Lorsque la connexion est établie, la cellule de départ décale les bits de configuration de sa partie fonctionnelle, et les envoie à la cellule d'arrivée. Cette dernière les injecte dans sa partie fonctionnelle, et ce pendant un nombre de coups d'horloge qui doit être déterminé par les PA, grâce à un compteur-trigger de valeur $68n$, pour une PF de n molécules. Dans notre exemple de 16 molécules, le compteur-trigger est réalisé grâce à 5 molécules : une molécule mémoire comptant 16, deux molécules dont on exploite les bits de configuration pour obtenir $68 = (16 + 14 + 24) + 14$, une molécule en mode Configure pour contrôler les deux précédentes, et une pour la combinaison des deux compteurs de 16 et 68.

Lorsque la configuration est terminée, la première cellule peut reprendre son acti-

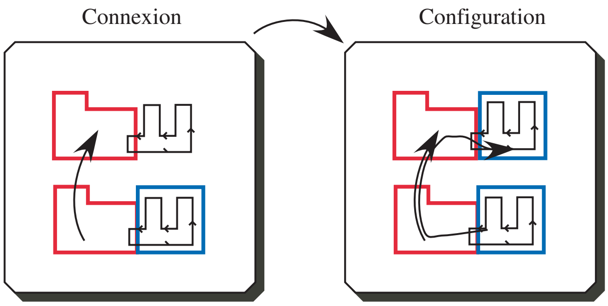

Figure 7.2 : *Duplication de la partie fonctionnelle d'une cellule, initiée par une connexion à une cellule en attente. Les bits de configuration sont ensuite envoyé à la nouvelle cellule.*

vité, et la deuxième peut, en fonction de l'application prévue, relancer une duplication ou simplement laisser travailler sa partie fonctionnelle. Il est bien clair que la partie responsable de l'autoréplication peut être modifiée en fonction du comportement désiré. Parmi les autres nombreux modes d'autoréplication qui peuvent être imaginés, citons-en deux :

- Les cellules pourraient être conçues pour s'auto-répliquer un certain nombre de fois prédéfini, en réinitialisant le routage distribué de manière à libérer leur molécule Output utilisée pour se connecter à une PA en attente.
- La duplication pourrait être lancée par la partie fonctionnelle, en envoyant un signal à la PA, par exemple lorsqu'un taux d'activation devient trop important dans le cas de réseaux de neurones à topologie variable.

Par notre exemple, nous avons montré qu'il était possible d'exploiter les capacités de POEtic pour implémenter un mécanisme d'autoréplication, sous certaines contraintes. Il est clair que la structure même de POEtic n'autorise pas une autoréplication complète dans le sens de von Neumann, puisque certains bits de configuration ne sont pas modifiables par les molécules elles-mêmes. Il serait dès lors intéressant, si la conception d'un tel circuit devait se refaire, de tenter d'y inclure des mécanismes du type de l'algorithme du petit Poucet, qui offriraient une réelle autoréplication aux systèmes cellulaires qui y seraient implémentés.

7.2 Développement

Le mécanisme que nous venons de décrire est basé sur la duplication de la partie fonctionnelle d'une cellule. Dans cette section, nous nous intéressons à des mécanismes ontogénétiques pour lesquels un génome décrivant l'organisme complet est présent dans chacune des cellules. De tels mécanismes peuvent tirer avantage des trois caractéristiques suivantes :

- Premièrement, les molécules peuvent servir à stocker un génome, en implémentant un registre à décalage.
- Deuxièmement, la propriété d'auto-configuration des molécules, où une molécule a la capacité de partiellement configurer son voisinage, permet de dynamiquement modifier le comportement d'une cellule.
- Troisièmement, le niveau de routage dynamique permet de créer des connexions entre différents endroits du circuit, typiquement entre des cellules. Un système

peut alors, à partir d'une seule cellule, s'étendre jusqu'à la création d'un organisme complet.

Un processus de développement, tel qu'il nous intéresse ici, nécessite un génome, présent dans chacune des cellules. Nous allons donc tout d'abord nous intéresser à la manière de stocker le génome dans les molécules de manière efficace. Ensuite, les deux phases du processus de développement, à savoir la croissance et la différenciation cellulaire seront explicitées.

7.2.1 Stockage de génome

Le génome décrivant un organisme est, dans tous les systèmes artificiels, à l'instar des organismes vivants, stocké dans une grande chaîne d'éléments d'informations. La vie a choisi les chaînes d'ADN, composés de 4 éléments, alors que les circuits électroniques, construits sur une base binaire, utilisent des chaînes de '0' et de '1'. Les molécules peuvent être utilisées à cette fin, en stockant le génome accessible en mode sériel. Pour ce faire nous nous basons sur l'implémentation du registre à décalage présenté à la section 6.8.1, page 208. Sur cette base, plusieurs manières de stocker le génome peuvent être développées, dont deux sont présentées ici.

Premièrement, le génome peut être simplement un long registre à décalage. Dans ce cas, et si le gène décrivant la fonctionnalité d'une cellule est de taille g et que l'organisme contient n cellules, alors l'accès au gène de la cellule c nécessitera $(c+1)g$ coups d'horloge. En effet, il faut cg coups pour atteindre le bon gène, puis g coups pour le récupérer. Ensuite, il faut replacer le génome dans son état initial, ce qui s'effectue en $(n-c)g$ coups d'horloge. La configuration de l'organisme entier s'exécute donc en ng coups d'horloge, car tous les gènes doivent pouvoir être accédés et le génome doit revenir dans son état initial. L'implémentation du génome nécessite, dans cette configuration, deux compteurs, un pour la valeur g, et un pour la valeur c.

Deuxièmement, le génome peut être décomposé en n registres à décalages de taille g, chacun contenant un gène, et l'accès peut se faire à l'aide de multiplexeurs, comme indiqué à la figure 7.3. L'accès se fait alors en g coups d'horloge, puisque n'importe quelle partie du génome peut être accédé sans aucune latence. Le temps de configuration de l'organisme entier correspond donc à celui d'une seule cellule. L'implémentation nécessite un compteur de valeur g, ainsi que les multiplexeurs nécessaires à la sélection de la bonne partie du génome.

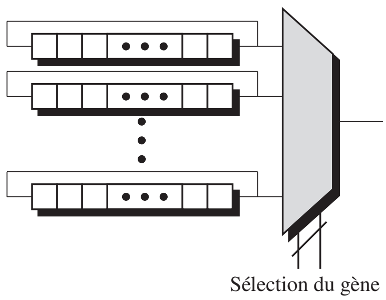

Figure 7.3 : *Un génome où un gène est directement accessible, grâce à l'utilisation d'un multiplexeur.*

Notons qu'une des différences importante entre les deux implémentations est que la première peut être implémentée avec des compteur-triggers, alors que la seconde nécessite un système contrôlant les multiplexeurs, de la forme d'un registre accessible en parallèle.

7.2.2 Croissance

Le développement d'un organisme artificielle est séparé en deux phases, la croissance et la différenciation cellulaire. La première voit le nombre de cellules croître jusqu'à atteindre le nombre nécessaire au bon fonctionnement de l'organisme, tandis que la deuxième sert à définir le comportement de chaque cellule, qui puise des informations dans le génome en fonction d'un facteur. Ce facteur dépend de la manière dont est implémenté l'ontogenèse, et peut par exemple être un identifiant unique à chaque cellule, ou des morphogènes, dans le cas d'un codage morphogénétique. Nous allons ici présenter différentes façons d'aborder la croissance, en fonction, notamment, de la manière de considérer les cellules.

Nous allons décrire deux mécanismes où chaque cellule possède un identificateur unique. Cet identificateur sert, lors de la phase de différenciation, à sélectionner la partie du génome devant être exprimée. Nous proposons les systèmes suivants :

- Un identificateur linéaire, où les cellules sont numérotées de 0 à $n-1$.
- Un système de coordonnées à 2 dimensions, l'identifiant ayant une composante en X et une en Y, du type (x, y).

Commençons par nous intéresser aux identificateurs linéaires, et décrivons comment un tel système de croissance peut aisément être construit pour POEtic. Un organisme de n cellules contiendra des cellules ayant pour identifiant tous les entiers dans $[0, n-1]$. L'idée de base étant de laisser le sous-système organique le plus indépendant possible d'un agent extérieur tel que le microprocesseur, qui ne sert qu'à lancer la croissance, en envoyant à une des cellules le premier identifiant, ainsi que le génome.

Rappelons tout d'abord qu'avant toute exécution d'un design dans le sous-système organique, le microprocesseur est en charge de configurer les molécules. Dans le cas d'un système ontogénétique tel que présenté, nous partons du principe qu'après configuration, des cellules totipotentes ont été placées sur le tissu. Elles contiennent, entre autre, une partie fonctionnelle ainsi que les molécules nécessaires au stockage du génome décrivant l'organisme, et sont, évidemment, toutes identiques.

Chaque cellule doit posséder une molécule en mode Input, en attente d'une connexion, sans toutefois initier une nouvelle connexion. Cette molécule sert de port d'entrée pour la phase de croissance, et doit avoir la même adresse dans toutes les cellules. Il faut évidemment garantir que cette adresse ne sera jamais utilisée par la suite lors du fonctionnement de l'organisme. Un agent externe, typiquement le microprocesseur, est alors chargé d'initier une première connexion, en utilisant une unité de routage du bord. Une fois la connexion établie avec la cellule totipotente la plus proche, l'agent externe lui envoie sont identifiant. La cellule le stocke, et calcule en même temps celui de la cellule suivante. Afin d'éviter que la cellule ne doive connaître la taille de l'organisme, il est proposé que le premier identifiant envoyé soit $n-1$, et que chaque cellule calcule l'identifiant suivant en effectuant une décrémentation. De ce fait, la fin de la croissance est détectée par la dernière cellule, qui possède l'identifiant 0.

Lorsque l'identifiant a été totalement reçu et que l'identifiant de la cellule suivante a été calculé, la cellule reçoit le génome, qu'elle stocke dans les molécules prévues à cet effet. Ensuite, la cellule utilise une molécule en mode Output ayant le même identifiant que celle d'entrée, pour se connecter à une autre cellule totipotente. La connexion établie, elle envoie l'identifiant fraîchement calculé et son génome[1], puis attend que l'organisme entier soit ainsi construit. Elle relâche également l'*enable* moléculaire global, qui est justement utilisé pour détecter la fin de la croissance. Les molécules prenant en charge la croissance ne sont pas sensibles à cet *enable*, alors que toutes les autres le sont. De cette manière, dès que la dernière cellule a reçu son identifiant et le génome, l'*enable* global est entièrement relâché, et le reste du circuit commence à fonctionner, lançant la phase de différenciation.

Nous proposons deux manières de coder l'identifiant : en format binaire, ou en format un parmi M. La principale différence entre ces deux codages est la manière d'effectuer la décrémentation. En effet, l'identifiant étant envoyé à la cellule de manière sérielle, le format binaire peut utiliser une molécule afin de calculer sériellement l'identifiant suivant. Pour un codage en un parmi M, décrémenter signifie simplement un décalage de un bit, ce qui nécessite une molécule de moins. Dans ce cas, la cellule 0 aura l'identifiant "1000000000000000", la cellule 1 "0100000000000000", etc, de manière à ce qu'un décalage à gauche de 1 signifie une décrémentation. Notons toutefois que, si le un parmi M économise une molécule, il ne peut être réalisé de manière plus efficace que pour des organismes ayant au plus 16 cellules.

L'algorithme 7.1 explicite les actions exécutées par chaque cellule afin de mener à bien la croissance de l'organisme :

Algorithme 7.1 Traitement effectué par chaque cellule, pour une croissance basée sur un identificateur simple

```
Attendre une connexion par le port d'entrée
Recevoir son identifiant et le stocker, et calculer l'identifiant suivant
Recevoir le génome et le stocker
Si l'identifiant courant est différent de 0 alors
    Se connecter à une cellule totipotente
    Lui envoyer son identifiant
    Lui envoyer le génome
Fin si
Relâcher l'enable moléculaire global
Attendre que l'organisme soit totalement construit
```

Passons maintenant à un système de coordonnées à 2 dimensions. Dans ce cas-ci, le routage dynamique permet de créer un tableau de cellules, en connectant celles-ci à leurs 4 voisines, par exemple, et ceci de manière virtuelle. En effet, deux cellules voisines dans le tableau n'ont pas besoin d'être voisines physiques, puisque le routage dynamique permet de créer n'importe quel chemin de données. La figure 7.4 montre un organisme composé d'un tableau de 3×3 cellules, les coordonnées virtuelles des cellules ne correspondant pas aux coordonnées réelles.

Pour l'implémentation du système de coordonnées, nous partons sur le même principe que précédemment : l'agent externe lance la croissance et doit interférer le moins possible, et toutes les cellules sont, après configuration initiale, totipotentes. De même, chaque cellule possède une molécule en mode Input, et deux en mode Output, avec une adresse similaire, du type "111...111". Il est proposé que le premier identifiant envoyé,

[1]De la même façon que pour les mécanismes d'autoréplication, le génome de la cellule source est sauvegardé, en utilisant un principe de registre à décalage bouclant sur lui-même.

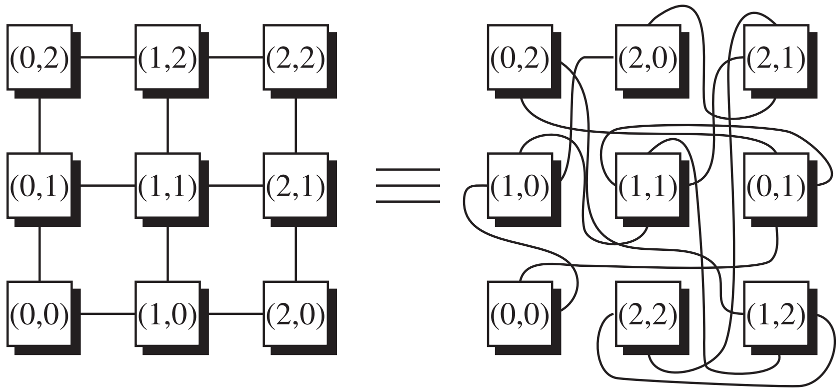

Figure 7.4 : *Le placement physique d'un tableau virtuel de* 3×3 *cellules.*

pour un tableau de $n \times m$ cellules, soit $(0, 0)$, et que chaque cellule calcule l'identifiant suivant en effectuant une incrémentation en y, de même qu'en x, pour celles ayant une coordonnée y à 0. En effet, l'argument utilisé dans le cas d'un identifiant unidimensionnel concernant le fait que la cellule n'a pas besoin de connaître la taille totale de l'organisme n'est plus valide ici, était donné qu'une cellule doit savoir si sa coordonnée en y est 0 ou $m-1$. La fin de la croissance est donc détectée par la dernière cellule, qui possède l'identifiant $(n-1, m-1)$.

Chaque cellule est chargée de se connecter à une cellule totipotente selon les conditions suivantes, pour une cellule de coordonnée (x, y) :

- Si $y \neq m-1$, alors se connecter à une nouvelle cellule, en lui envoyant l'identifiant $(x, y+1)$.
- Si $y = 0$ et $x \neq n-1$, alors se connecter à une nouvelle cellule, en lui envoyant l'identifiant $(x+1, y)$, ou $(x+1, 0)$, ce qui est équivalent.

De cette manière, nous créons un arbre de développement tel que celui de la figure 7.5.

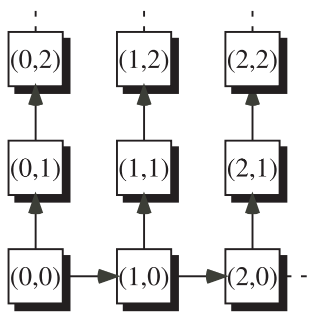

Figure 7.5 : *Arbre de développement d'un système de coordonnées.*

L'arbre de développement de la figure 7.5 place physiquement les cellules dans le tissu. Toutefois, il se peut que le placement physique diffère du placement virtuel. Afin d'obtenir un placement physique le plus proche possible du virtuel, l'entrée et les deux sorties doivent être placés tels que sur la figure 7.6. La sortie ox est alors utilisée pour connecter les cellules de la ligne 0 entre elles, et la sortie oy sert de connexion vertical. La figure 7.6 illustre bien le fait que ce placement minimise le routage nécessaire pour connecter les différentes cellules entre elles, la sortie ox, de même que la sortie oy, étant voisine immédiate de l'entrée de la cellule suivante.

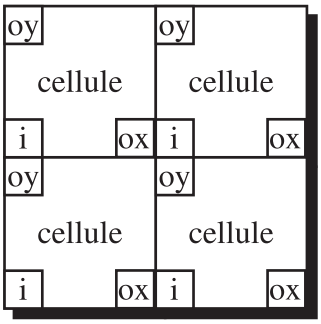

Figure 7.6 : *L'entrée et les deux sorties d'une cellule, pour obtenir un arbre de développement tel que celui de la figure 7.5.*

L'agent externe débute la phase de développement, en se connectant, via une unité de routage du bord, à la cellule totipotente la plus proche, et lui envoie ensuite son identifiant, soit la valeur $(0, 0)$ pour un tableau de $n \times m$ cellules, ainsi que le génome. La cellule récupère son identifiant, le stocke, et compare la coordonnée en Y avec 0 et $m - 1$ et la coordonnée en X avec $n - 1$. En fonction de sa coordonnée, elle se connecte à zéro, une, ou deux cellules totipotentes, et leur transmet les coordonnées calculées, de même que le génome.

De la même manière que précédemment, l'*enable* moléculaire global peut être utilisé afin de détecter la fin de la croissance. Les quelques lignes de l'algorithme 7.2 explicitent les actions exécutées par chaque cellule afin de mener à bien la croissance de l'organisme.

Algorithme 7.2 Traitement effectué par chaque cellule, pour une croissance basée sur un systèmes de coordonnées

Attendre une connexion par le port d'entrée
Recevoir son identifiant (x, y) et le stocker
Recevoir le génome et le stocker
Si $y = 0$ et $x \neq n - 1$ **alors**
    Se connecter à une cellule totipotente
    Lui envoyer l'identifiant $(x + 1, y)$
    Lui envoyer le génome
Fin si
Si $y \neq m - 1$ **alors**
    Se connecter à une cellule totipotente
    Lui envoyer l'identifiant $(x, y + 1)$
    Lui envoyer le génome
Fin si
Relâcher l'*enable* moléculaire global
Attendre que l'organisme soit totalement construit

D'autres mécanismes de croissance peuvent évidemment être envisagés. Le système morphogénétique, développé au Laboratoire de Systèmes Autonomes dans le cadre du projet POEtic, en est un exemple. Fonctionnant sur le principe de diffusion de gradients chimiques, il a été implémenté sur le substrat moléculaire. Les détails de cette implémentation peuvent être trouvés dans [192].

7.2.3 Différenciation

Contrairement aux êtres vivants dans lesquels les cellules n'attendent pas que l'organisme entier soit terminé pour se différencier, dans notre approche, la phase de différenciation est distincte de la croissance. En effet, les organismes vivants ne pourraient terminer leur développement sans que la croissance ne soit accompagnée de différentiation, car la différenciation cellulaire influe sur la croissance des tissus. En revanche, dans notre approche, les cellules ayant un identifiant unique, il est possible de générer l'identifiant de chaque cellule sans mettre en oeuvre les mécanismes naturels complexes. De plus, la création des connexions entre cellules grâce au routage dynamique implique une séparation des deux phases. En effet, si toutes les cellules ne sont pas créées, c'est-à-dire n'ont pas d'identifiant, alors certaines connexions ne pourront se créer, bloquant ainsi le système.

La phase de différenciation débute donc lorsque toutes les cellules ont acquis leur identifiant. Elles utilisent l'information contenue dans le génome pour configurer correctement certaines de leurs molécules, en fonction de leur identificateur, qui sert d'indice pointant sur le bon gène. La propriété d'auto-configuration des molécules permet alors de partiellement configurer certaines de leurs parties, comme par exemple la look-up table. Notons également que les adresses des molécules d'entrée/sortie peuvent être reconfigurées de cette manière, laissant le génome définir la topologie du réseau cellulaire.

La configuration de la cellule s'exécute en un nombre de coups d'horloge dépendant du type de stockage du génome tel que défini à la section 7.2.1. Le génome est décalé, et les molécules concernées par la configuration sont configurées avec les nouvelles valeurs. Lorsque toutes les cellules sont ainsi correctement configurées, la création du réseau cellulaire peut débuter, mettant en jeu les adresses d'entrée/sortie fraîchement définies. Plusieurs routages s'effectuent jusqu'à ce que l'organisme entier soit opératoire, la fonctionnalité de ses cellules ainsi que leur topologie étant définies.

Dans le cas d'un système de coordonnées en 2 dimensions, et où la topologie du réseau cellulaire correspond à un voisinage de Von Neumann (4 voisins directs), la topologie n'a pas besoin d'être stockée dans le génome, mais peut être calculée en fonction de la coordonnée de la cellule. Le génome ne sert donc qu'à la fonctionnalité de la cellule, et le gène à sélectionner peut être calculé en fonction de cette coordonnée, en concaténant les coordonnées x et y.

7.3 Un exemple concret : le prototype PO

Après avoir explicité différentes manières de stocker un génome, faire croître un organisme, et différencier des cellules, nous allons maintenant illustrer ces concepts avec un exemple concret : le prototype PO. Dans le cadre du projet POEtic, ce prototype [194] visait à montrer des capacités d'évolution (P) et de développement (O), et devait être implémenté sur FPGA. Pour ce faire, nous avons réalisé, en collaboration avec Daniel Roggen, un design comprenant le microprocesseur POEtic ainsi qu'un sous-système organique de 80 molécules. Le tout fut placé sur une Xilinx Virtex XC2V3000-4FF1152.

Notons qu'une des différences importantes entre notre implémentation et le circuit POEtic final est que dans notre prototype, le sous-système organique est configuré de manière sérielle, et non parallèle. Cette modification fut obligatoire afin de réduire les problèmes de routage, et de pouvoir placer un maximum de molécules sur le FPGA.

7.3.1 Implémentation physique

La synthèse du système donne pour résultat un usage des ressources de 5% des LUTs pour le microprocesseur, 66% pour le sous-système organique, et un petit pourcentage pour la connexion des deux modules. Après placement-routage, 97% des ressources et 51% des blocs de mémoire sont réquisitionnés. En réalité, le placement s'exécute avec succès pour 100 molécules, mais le routage ne trouve aucune solution, 33 fils ne pouvant être routés. Concernant la fréquence de fonctionnement du système, les résultats après placement-routage ne sont pas significatifs, étant donné que le sous-système organique contient des boucles combinatoires, à cause des switchboxs intermoléculaires et les switchboxs du routage dynamique. Toutefois, l'expérience montre que le système fonctionne parfaitement à 10MHz. Notons que le fonctionnement correct du système n'était pas garanti, étant donné que le placement-routage devait se faire sans contraintes de timing. En effet, les outils de Xilinx, à cause des boucles combinatoires, sont incapables de mener à bien le placement-routage en prenant en compte ces contraintes.

Il est bien clair que 80 molécules dans un FPGA contenant 14000 éléments de base est loin d'être un résultat exceptionnel. Toutefois, il faut prendre en compte le fait qu'un FPGA n'est pas fait pour émuler un autre FPGA, et ceci pour deux raisons majeures. Premièrement, le routage du FPGA émulé, qui est un des éléments critique dans un tel circuit, demande énormément de ressources. Deuxièmement, chacune de nos molécules, qui, en terme de complexité, correspond plus ou moins à un des 14000 éléments de bases du FPGA cible, contient 76 bits de configuration, et chacun de ces bits est implémenté dans un élément de la cible, ce qui implique une grande utilisation des ressources matérielles. Ces résultats montrent bien que notre système n'est qu'un prototype, et que la réalisation physique du circuit POEtic est plus que nécessaire afin d'obtenir un système efficace.

Pour ce prototype, nous avons choisi de réaliser un système cellulaire où chaque cellule est composée d'une fonction quelconque à trois entrées. L'évolution permet alors d'évoluer la fonctionnalité de chacune des cellules, ainsi que la connectivité intercellulaire. Étant donné le peu de molécules à disposition, il n'a été possible de placer que deux cellules sur les 80 molécules. La fonctionnalité de l'organisme est donc une fonction logique, dépendante de l'application, et le système a été évalué avec trois applications : un multiplexeur, un additionneur avec retenue, et une application de contrôle de robot. Nous allons décrire la structure cellulaire implémentée, ainsi que les concepts ontogénétiques utilisés, avant d'entrer plus en détail dans les différentes applications.

7.3.2 Structure cellulaire

Le principe fondamental de notre tissu PO est que chaque cellule contient le génome décrivant entièrement l'organisme. La structure cellulaire suit le principe des trois couches introduites dans [245], et que nous avons traitées en page 62. La couche génotypique contient le génome, la couche de mapping contient deux éléments, un pour la phase de croissance, et un pour la différenciation, et finalement, la couche phénotypique contient la fonctionnalité de la cellule, à savoir trois entrées, une sortie, et une look-up table à trois entrées. La figure 7.7 explicite ces trois couches, et la figure 7.8 montre l'emplacement de ces couches dans la cellule. 40 molécules sont nécessaires à l'implémentation d'une cellule, ce qui ne permet d'avoir que deux cellules dans le système. La répartition des trois couches en terme de molécules est la suivante :

- La couche génotypique occupe 16 molécules en mode mémoire. Le génome représente une organisme de 4 cellules au maximum, et donc les mêmes cellules peuvent être utilisées pour une application plus conséquente. Nous pouvons cependant noter qu'il serait possible d'exploiter les bits de configuration des molécules pour le stockage du génome, ce qui aurait pour conséquence de réduire la taille de la cellule. Ne pouvant de toute façon pas placer plus de deux cellules dans les 80 molécules, nous avons gardé les molécules en mode mémoire, ce qui permettait de simplifier le traitement effectué par le microprocesseur.
- La couche de mapping est implémentée sur 18 molécules : 8 molécules pour la croissance, et 10 pour la différenciation.
- La couche phénotypique n'est réalisée qu'avec 5 molécules : 3 pour les entrées, une pour la sortie, et une pour la partie fonctionnelle (une 3-LUT).

Couche phénotypique

Couche de mapping

Couche génotypique

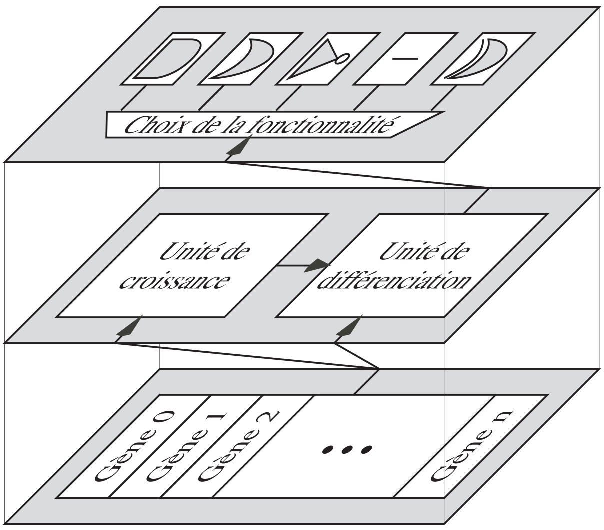

Figure 7.7 : *Les trois couches composant une cellule. La couche génotypique contient le génome, la couche de mapping s'occupe du développement, et la couche phénotypique correspond à la fonctionnalité de la cellule.*

7.3.3 Mécanisme de développement

Un des buts principaux du prototype PO étant de montrer une capacité de développement d'un organisme à partir d'une cellule, les cellules ont été spécifiquement définies dans cette optique. En effet, la partie fonctionnelle n'occupe que 12,5% de la cellule, le reste des molécules réalisant le mécanisme de développement. Le mécanisme ontogénétique est basé sur un système d'identificateur linéaire, codé en 1 parmi

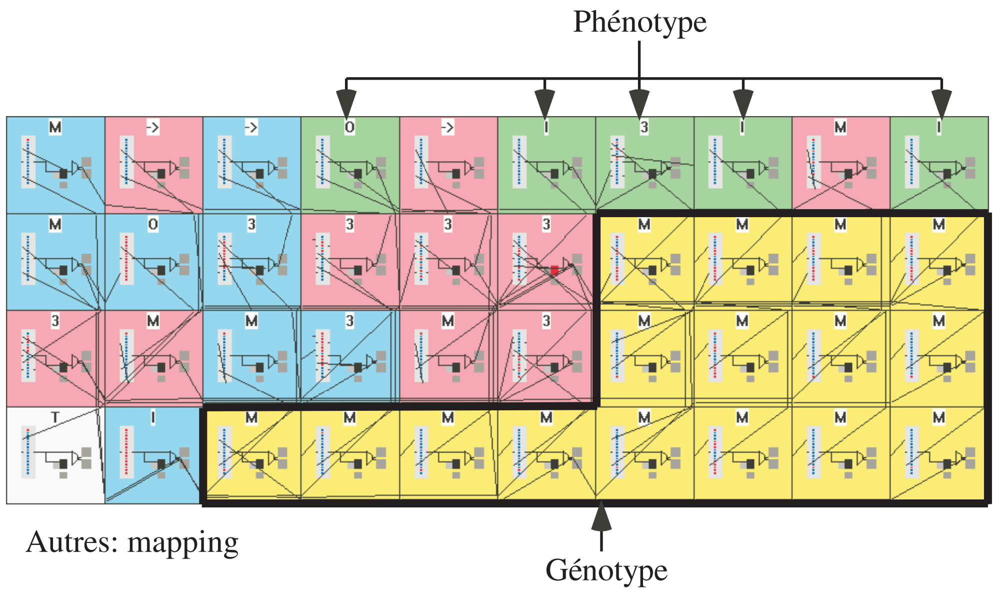

Figure 7.8 : *Implémentation d'une cellule sur les molécules, montrant les trois blocs : génome, développement, fonctionnalité.*

M, sur 16 bits. La cellule numéro 0 a un identifiant 100...0, la cellule numéro 1 a l'identifiant 010...0, etc.

Croissance Après la configuration des molécules, toutes les cellules sont identiques, totipotentes, non connectées, et non différenciées. La phase de croissance est initiée par le microprocesseur, qui utilise une unité de routage du bord pour se connecter à la cellule totipotente la plus proche. Chaque cellule possède une molécule en mode Input, avec l'adresse 11...111. Une fois la cellule connectée grâce au routage dynamique, le microprocesseur lui envoie son identifiant en mode sériel. L'opération est effectuée en 17 coups d'horloge, de manière à ce que la cellule génère l'identifiant de la cellule suivante, en décrémentant son propre identifiant. Après les 17 coups d'horloge, la cellule détecte si elle est la dernière (ID zéro) ou non, en observant le bit de poids fort de son identifiant. Si elle n'est pas la dernière, elle se connecte à une autre cellule totipotente, puis lui envoie l'identifiant qu'elle vient de calculer. En même temps, elle relâche l'*enable* global qui permet, lorsque la dernière cellule est atteinte, de lancer la différenciation (les cellules responsables de la croissance sont insensibles à cet *enable*, alors que toutes les autres le sont).

Différenciation Lorsque toutes les cellules ont reçu leur identifiant, l'*enable* global est relâché, et les molécules en charge de la différenciation prennent le relais. Le génome, composé de 16 molécules en mode mémoire, permet de décrire un organisme contenant jusqu'à quatre cellules. Chaque cellule est donc décrite par quatre molécules, soit 64 bits. Chaque adresse d'entrée de la cellule doit alors être configurée avec 16 bits, et les 16 derniers bits servent pour moitié à la définition de la fonctionnalité de la cellule.

Le génome est donc, dans chaque cellule, décalé. L'identifiant, contenu dans une LUT, est lui décalé tous les 64 coups d'horloge, et lorsqu'il vaut 0 (100...000), le gène récupéré est directement utilisé pour configurer les trois entrées de la partie fonctionnelle, ainsi que la LUT. Après 4x64 coups d'horloge, le génome est donc dans son état initial, et toutes les cellules de l'organisme sont correctement différenciées. Un routage dynamique est alors lancé jusqu'à ce que toutes les connexions intercellulaires soient créées. Le circuit est ensuite prêt à fonctionner, le microprocesseur peut forcer les entrées et récupérer la sortie, afin de calculer le fitness de l'individu testé.

7.3.4 Evolution

Alors que la majorité des circuits électroniques sont conçus par des ingénieurs, le concept de matériel évolutif délègue cette tâche à un système automatique. Le microprocesseur est ici en charge d'exécuter un algorithme génétique faisant évoluer une population d'individus de deux cellules. Le système possède six entrées et deux sorties, placés tels que sur la figure 7.9, qui sont notamment utilisées comme résultat et retenue pour l'additionneur. Une entrée d'une cellule peut donc choisir entre une des six entrées et la sortie d'une des cellules, et a donc huit choix possibles, définis par 3 bits. Comme nous l'avons vu, la cellule est définie par sa fonctionnalité, soit 8 bits, et par la connectivité de ses entrées, soit 3 bits chacune. Au total nous avons donc 17 bits par cellule, pour un total de 34 pour un individu.

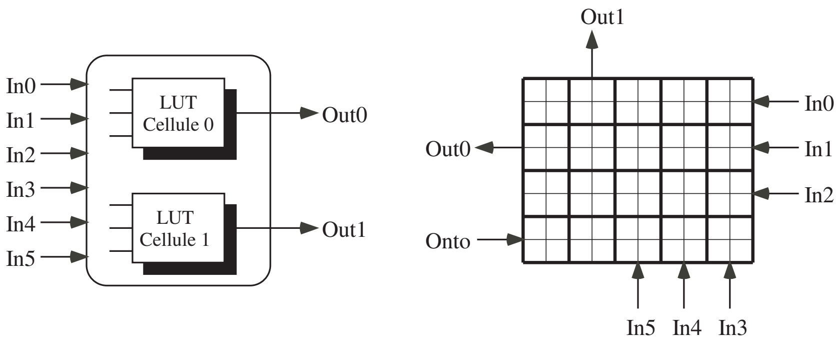

Figure 7.9 : *A gauche, la partie fonctionnelle de l'organisme, et à droite la répartition des entrées/sorties sur le circuit.*

Evolution de fonctions logiques Dans cette application, le prototype PO est utilisé pour l'implémentation de fonctions logiques, un additionneur, et un multiplexeur. Les trois premières entrées du système sont forcées par le microprocesseur, tandis que les trois suivantes sont fixées aux valeurs 0,1 et 0. L'application multiplexeur ne comporte qu'une sortie, alors que l'additionneur en compte deux, une pour le résultat, et une pour la retenue. Chaque individu est évalué en lui présentant toutes les entrées possibles, soit huit possibilités, et en vérifiant que la sortie corresponde à la fonction désirée.

Un algorithme génétique est exécuté par le microprocesseur, avec les paramètres décrits dans le tableau 7.2, afin de trouver une solution efficace. Le fitness d'un individu est calculé en comptant le nombre de sorties correctes sur les huit évaluées. Dans le cas du multiplexeur, le fitness est compris entre 0 et 8, et dans celui du multiplicateur, étant donné qu'il a deux sorties, entre 0 et 16. La figure 7.10 montre l'évolution du fitness moyen, ainsi que celui du meilleur individu, pour le multiplexeur et l'additionneur. Sur 32 exécutions de l'algorithme génétique, 31 ont réussi à trouver une

Paramètre	Valeur
taille de la population	200
taux de crossover	30%
taux de mutation	5%
sélection	au rang, dans les 20 meilleurs
élitisme	oui

Tableau 7.2 : *Paramètres de l'algorithme génétique*

solution pour le multiplexeur, en 50 générations au maximum (le seul échec a terminé avec un meilleur fitness de 7). Concernant l'additionneur, 20 exécutions sur 32 se sont terminées avec succès, les 12 échecs arrivant à un fitness de 14. Ces résultats correspondent au fait que l'additionneur, avec deux sorties, est un système plus complexe à réaliser que le multiplexeur.

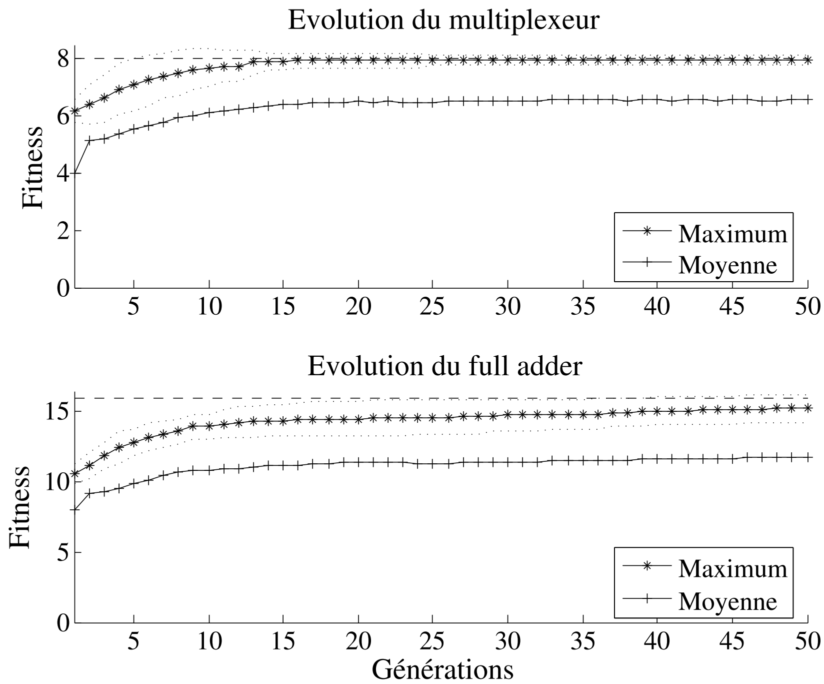

Figure 7.10 : *L'évolution du fitness moyen et maximum, pour un multiplexeur et un additionneur complet.*

Evolution d'un contrôleur de robot La deuxième application du prototype PO vise à contrôler un robot Khepera [111] dans une tâche d'évitement d'obstacle. Le Khepera est un robot qui, dans sa version de base, possède huit capteurs de proximité infrarouges, ainsi que deux roues. La figure 7.11 illustre la position des capteurs du robot, ainsi que de ses deux roues. Notons que, notre système à deux cellules ayant six entrées, les capteurs latéraux sont groupés, la valeur du capteur le plus actif étant utilisée en entrée. Une entrée de l'individu est à '1' si la valeur du capteur est supérieure à un certain seuil, et à '0' sinon. La vitesse des roues du robot est directement calculée en fonction des valeurs des capteurs de proximité infrarouge du robot, grâce aux deux cellules de notre tissu. Chacune des sorties commande une des deux roues, et les moteurs du Khepera ont une période de 100ms durant laquelle la vitesse reste constante. Le tissu POEtic met donc cette vitesse à jour à la fin de chaque période, en fonction de la sortie. Si la sortie est à '1', le moteur fonctionne à +80mm/s, tandis qu'une valeur '0' le fait fonctionner à -80mm/s. Dans notre modèle, les moteurs ne peuvent jamais s'arrêter, ce choix ayant été dicté par le petit nombre de cellules de notre tissu.

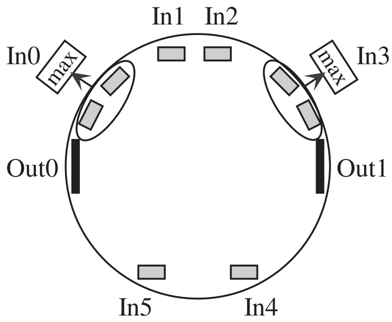

Figure 7.11 : *Les entrées (capteurs de proximité) et sorties (moteurs des roues) du robot Khepera.*

Le fitness d'un individu est évalué en fonction du comportement du robot de manière à maximiser le mouvement d'avancement, tout en minimisant les contacts avec les murs. Le microprocesseur le calcule en sommant la vitesse des moteurs lorsqu'ils tournent les deux vers l'avant [70]. Cette technique simple permet de favoriser les robots qui évitent les obstacles, car la vitesse des moteurs d'un robot bloqué contre un mur est nulle, de par le frottement avec le sol. L'algorithme génétique exécuté par le microprocesseur utilise les mêmes paramètres que pour l'application précédente ; toutefois, afin de gagner du temps, l'évolution s'est faite en simulation, seul le meilleur individu ayant été implémenté dans le robot réel. La figure 7.12 montre l'évolution du fitness moyen et maximum, et nous pouvons noter que dès la dixième génération, de très bons candidats furent obtenus.

7.3.5 Remarques conclusives

Par ce prototype du tissu POEtic, nous avons démontré que le routage dynamique ainsi que la capacité d'auto-configuration des molécules sont des caractéristiques importantes qui peuvent être exploitées par un système ontogénétique. Le système présenté, capable de développer un organisme à partir d'un tableau de cellules totipotentes sur ordre d'un agent externe, peut être une base sur laquelle construire un organisme auto-réparateur, où une cellule endommagée pourrait être remplacée par une autre. De plus, ce prototype a permis de vérifier le bon fonctionnement du microprocesseur, qui n'était alors pas encore figé dans le circuit final.

7.4 Matériel évolutif non-contraint

Le routage interne du sous-système organique de POEtic est uniquement composé de multiplexeurs. Bien qu'allongeant potentiellement les délais entre les registres, cette

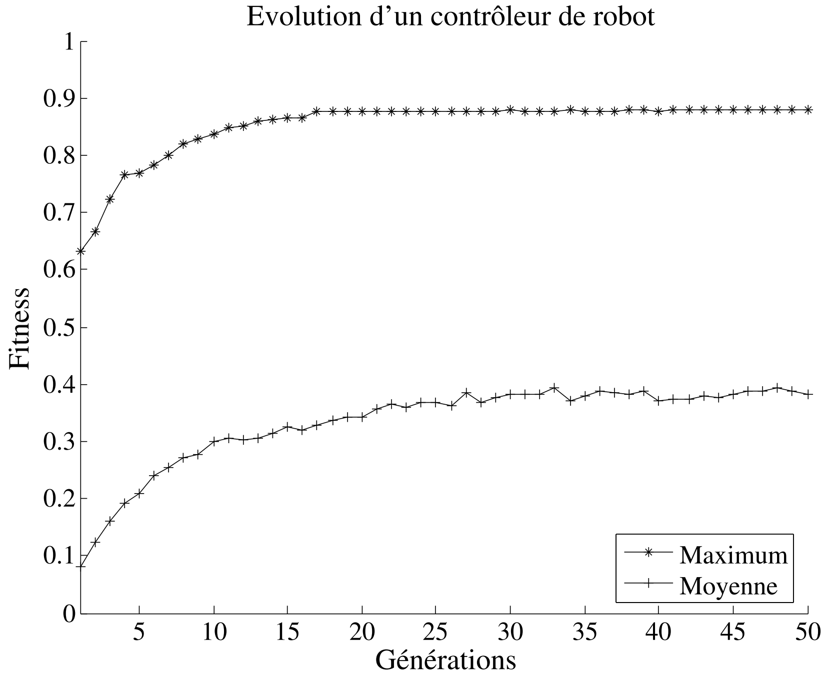

Figure 7.12 : *L'évolution du fitness moyen et maximum, pour un contrôleur de robot.*

implémentation offre l'avantage d'exclure tout court-circuit. En effet, les FGPAs actuels sont conçus sur des technologies pouvant laisser apparaître des courts-circuits si le bitstream de configuration n'est pas correct. Les outils tels que JBits [85] pour les circuits Virtex permettent de modifier les bits de configuration du FPGA à la main, mais mettent en garde l'utilisateur sur la possibilité de détruire le circuit.

La dernière FPGA conçue sans courts-circuits potentiels fut la série XC6200 de Xilinx. Depuis, aucun FPGA de ce type n'est apparu, alors que les applications de matériel évolutifs tels que celles développées par Thompson [235], ont besoin de cette qualité. Notre circuit est donc un candidat rêvé pour l'implémentation de systèmes mettant en oeuvre un évolution non-contrainte. De plus, à l'instar du XC6200, les bits de configuration sont connus, alors qu'ils ne sont pas rendus publics pour les autres FPGAs commerciaux. Il est alors trivial de ne choisir qu'une partie du bitstream à évoluer, que se soit le routage intermoléculaire, la fonctionnalité des molécules, ou d'autres. Alors que le prototype PO mettait à profit le routage distribué, dans un mécanisme d'évolution contrainte, nous ne l'exploitons aucunement dans cette section, qui vise à effectuer de l'évolution non contrainte.

Il est bien évident possible de concevoir un système faisant évoluer l'entièreté des bits de configuration d'une partie du sous-système organique. Toutefois, 76 bits définissant une molécule, un tableau de 10×10 molécules est décrit par rien moins que 7600 bits, ce qui peut poser des problèmes à un algorithme génétique qui se perdrait dans l'espace de recherche (cf. problème de scalabilité, page 51). Une solution, que nous allons développer ici, consiste en la réduction de l'espace de recherche, par la fixation de certains bits de configuration.

Nous allons expliciter la manière de fixer ces bits afin de réduire l'espace de recherche, tout en gardant une grande fonctionnalité [233]. L'élément de base à évoluer est alors tel que présenté à la figure 7.13, et contient une look-up table à trois entrées, trois multiplexeurs pour ces entrées, et un switchbox pour les sorties, chaque sortie pouvant être une des entrées des trois autres directions, ou le résultat de sortie de la molécule. Notons qu'un bit supplémentaire peut définir si la sortie est combinatoire ou séquentielle.

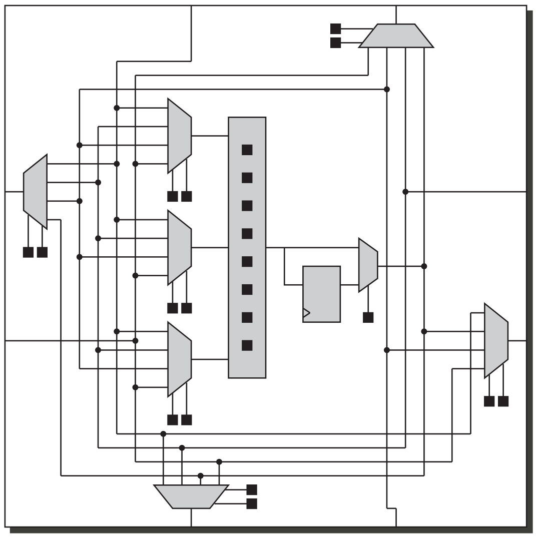

Figure 7.13 : *Le sous-ensemble de configuration d'une molécule, défini par 22 bits, ou 23 pour l'utilisation de la bascule.*

7.4.1 La look-up table

Durant la phase d'évolution, nous laissons la possibilité à la look-up table de la molécule d'évoluer. Nous pouvons choisir une 3-LUT ou une 4-LUT, la grande différence entre les deux approches étant la taille du génome. En effet, la version 3-LUT nécessite 8 bits pour la LUT, et 6 bits pour les entrées, soit 14 au total, alors que la version 4-LUT contient 16 bits pour la LUT et 8 pour les entrées, pour un total de 24. Dans l'optique de faciliter le travail de l'algorithme génétique, la version 3-LUT sera préférée, tout en n'excluant pas l'autre possibilité.

7.4.2 Le switchbox

Le switchbox d'une molécule, comme nous l'avons défini en page 187, est composé de huit multiplexeurs à huit entrées. Il y a donc deux lignes partant dans chaque direction. Nous pouvons, toujours afin de limiter l'espace de recherche, nous limiter à une seule ligne dans chaque direction. La moitié des multiplexeurs est alors inutilisée, et les quatre restants n'ont pas besoin de toutes leurs entrées, mais seulement de quatre d'entre elles. En effet, en n'utilisant que les lignes `ValInN(0)`, `ValInE(0)`, `ValInS(0)` et `ValInW(0)`, les entrées `ValIn`*X*`(1)` ne sont pas définies. Nous nous retrouvons en fin de compte avec un switchbox composé de quatre multiplexeurs à quatre entrées. La figure 7.14 montre le switchbox original, et les bits à fixer, ainsi que le switchbox résultant.

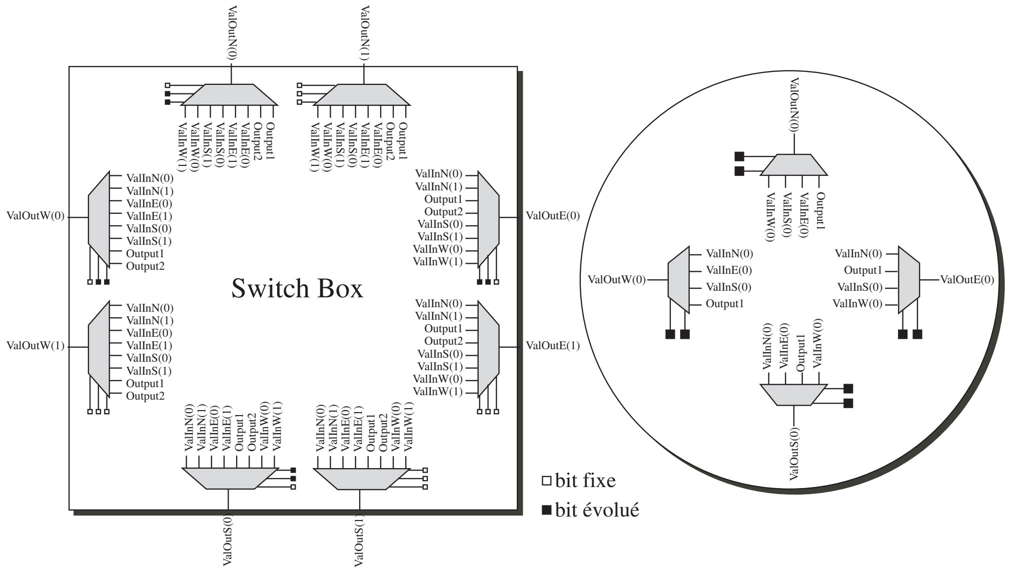

Figure 7.14 : *A gauche le switchbox d'une molécule, avec certains bits de configuration fixés. A droite, le switchbox résultant des fixations.*

7.4.3 Les entrées

Les entrées de la look-up table sont gérées par des multiplexeurs à 8 entrées, plus quelques autres. Etant donné que nous avons quatre entrées, venant chacune d'une des quatre voisines, les multiplexeurs peuvent être réduits en fixant certains bits de configuration. La figure 7.15 montre les multiplexeurs originaux, et les bits à fixer, ainsi que les multiplexeurs résultants pour les deux premières entrées.

7.4.4 Représentation du génome

Dans cette section nous présentons une manière de représenter le génome, afin d'optimiser le temps d'exécution de l'algorithme génétique.

Approche conventionnelle La première approche consiste simplement en une représentation compacte du génome. Les 22 bits sont stockés de manière optimale dans la mémoire du microprocesseur, et l'ensemble des bits de configuration d'une molécule, sur 76 bits, est reconstruit à partir de ces 22 bits. Cette méthode peut être plus efficace en terme de temps d'exécution si les opérateurs de crossover et mutation sont extrêmement rapides. Le mapping entre le génome et les bits de configuration réels est toutefois relativement complexe.

Nouvelle approche Nous proposons une nouvelle approche dans laquelle nous cherchons à optimiser la phase de mapping. En effet, passer des 22 bits d'un gène aux 76 décrivant la configuration d'une molécule est une tâche lourde en terme de temps de calcul. Nous proposons donc de travailler avec un génome composé de gènes de $3 \times 32 = 96$ bits. Sur ces 96 bits, seuls 22 représentent de l'information utile pour l'évolution, 54 sont des bits de configuration fixes, et les 20 restants ne sont tout simplement pas utilisés. L'idée est donc de voir un gène comme étant constitué d'une

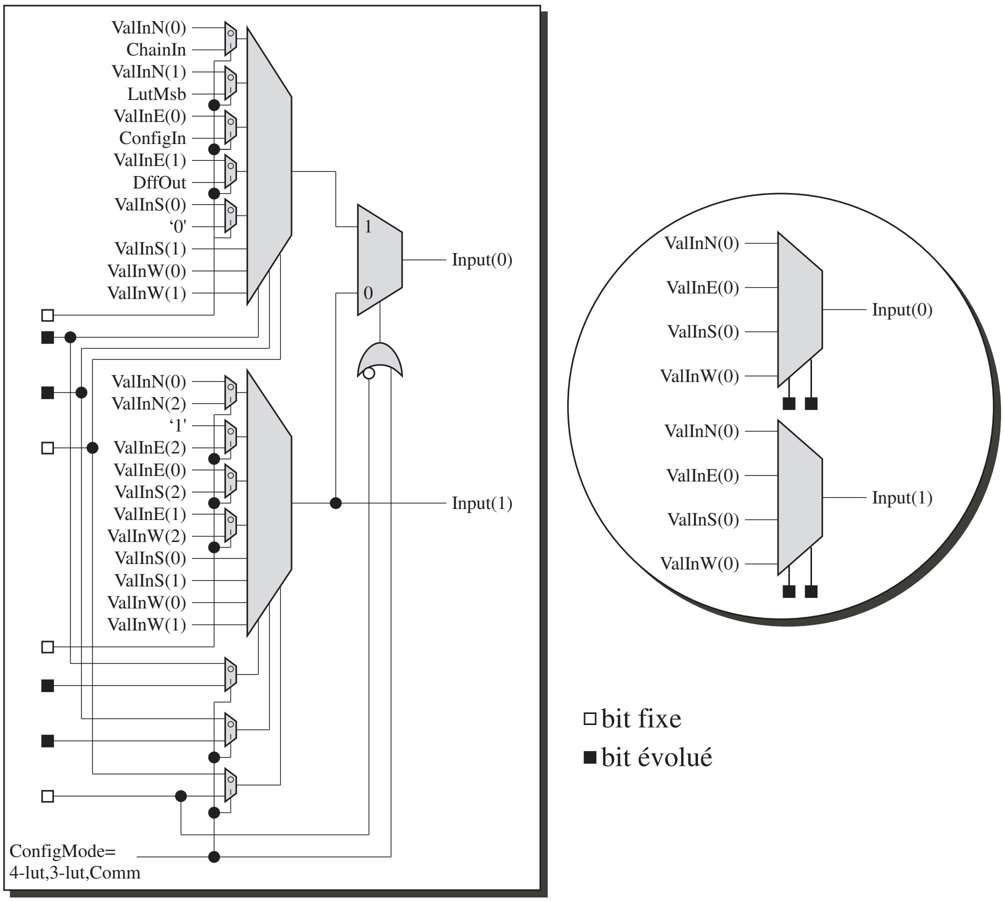

Figure 7.15 : *A gauche, les entrées d'une molécule, avec certains bits de configuration fixés. A droite, les entrées résultantes des fixations.*

partie utile, et d'une partie inutile, à la façon des organismes vivants, où une partie de l'ADN, appelée ADN poubelle, ne code aucune information (selon les recherches actuelles). Si nous reprenons le mapping mémoire des bits de configuration d'une molécule tel que défini à la page 192, nous découpons les 96 bits d'un gène de la façon décrite à la figure 7.16.

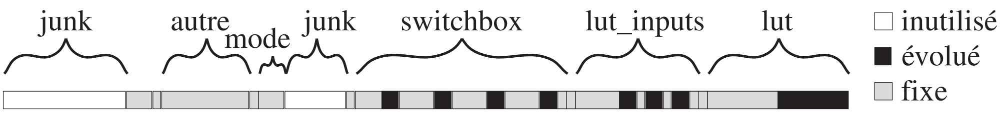

Figure 7.16 : *Un gène de 96 bits, décomposé en bits à évoluer, fixes, et inutilisés.*

Lors d'un crossover, le gène entier est considéré, de même que lors d'une mutation, et le mapping du gène aux bits de configuration peut se faire grâce à des opérations logiques simples. En effet, il suffit de garantir que les bits de configuration fixes le sont, et que les bits évolués sont bien exploités. Pour ce faire, il suffit de disposer de deux masques complémentaires, appliqués aux bits fixes et aux bits évolués, et d'opérer un OU logique entre les deux résultats. La figure 7.17 résume les opérations à effectuer sur un gène afin d'obtenir les bits de configuration de la molécule. Notons que l'application du masque aux bits fixes peut n'être effectuée qu'une seule fois, car son résultat est réutilisé pour toutes les molécules.

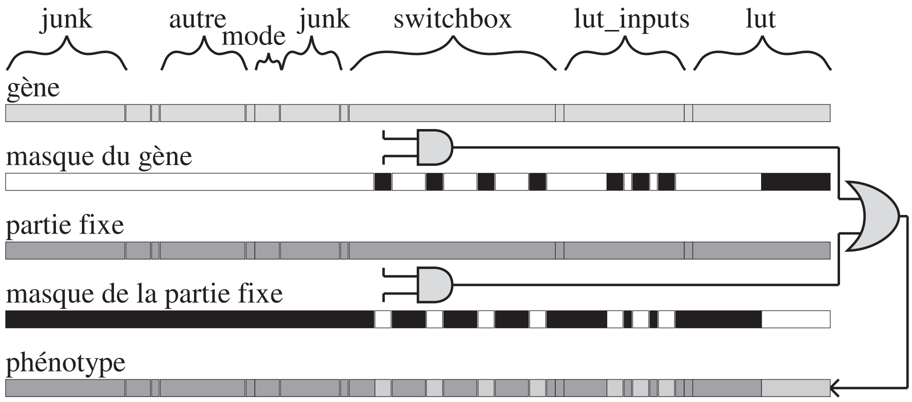

Figure 7.17 : *Les opérations logiques nécessaires au mapping d'un gène aux bits de configurations.*

Comparaison Lorsque le circuit POEtic sera opérationnel, il sera possible de comparer ces deux approches, afin de définir la plus efficace. Nous donnons ici une piste quant à leur évaluation. Le temps d'exécution de l'approche conventionnelle est dénotée par Tc, et celui de la nouvelle par Tn.

Posons :

Tx_{mut} est le temps d'exécution de l'opérateur de mutation, sur un génome, pour l'approche x.

Tx_{cross} est le temps d'exécution de l'opérateur de crossover pour deux individus, pour l'approche x.

Tx_{sel} est le temps d'exécution de la sélection de tous les individus d'une nouvelle génération, pour l'approche x.

Tx_{fit} est le temps d'exécution du calcul de fitness d'un individu, pour l'approche x.

Tx_{map} est le temps nécessaire à la création des bits de configuration de la molécule, à partir du génome, pour l'approche x.

Tx_{tot} est le temps d'exécution d'une phase de l'algorithme génétique, c'est-à-dire du calcul d'une nouvelle population.

Le temps d'exécution d'une phase de l'algorithme génétique correspond, pour une population de taille sp à ceci :

$$Tx_{tot} = sp * Tx_{mut} + sp/2 * Tx_{cross} + Tx_{sel} + sp * Tx_{map} + sp * Tx_{fit}$$

Or, nous avons que :

$$\begin{aligned} Tc_{sel} &= Tn_{sel} \\ Tc_{fit} &= Tn_{fit} \end{aligned}$$

La deuxième approche est plus efficace en terme de temps d'exécution si et seulement si $Tn_{tot} < Tc_{tot}$, ce qui, en regard des deux égalités, est équivalent à :

$$Tn_{mut} + Tn_{cross}/2 + Tn_{map} < Tc_{mut} + Tc_{cross}/2 + Tc_{map}$$

7.4.5 Caractéristiques de l'évolution

Lorsque le circuit POEtic sera opérationnel, nous serons en mesure de mettre en œuvre le mécanisme d'évolution non-contrainte que nous venons de présenter. Nous pouvons toutefois déjà caractériser le circuit POEtic, en fonction des paramètres des systèmes de matériel évolutif définis par Torresen dans [240, 241] :

- Le microprocesseur peut exécuter un *Algorithme génétique*.
- La technologie cible est *digitale*.
- L'architecture appliquée à l'évolution peut être *Complete Circuit Design*, où des blocs de base et le routage sont évolués, ou *Circuit Parameter Tuning*, ou seuls des paramètres configurables sont évolués.
- Les blocs de base peuvent être des portes logiques (les LUTs), où des fonctions (neurones, ...).
- L'évolution est exécutée *online*, car chaque individu peut être testé sur le matériel reconfigurable.
- L'évolution est *on-chip*, étant donné que le microprocesseur est incorporé au circuit reconfigurable.
- L'évolution peut y être *contrainte* (prototype PO), ou *non contrainte*, comme nous venons de le suggérer.
- Le domaine d'évolution peut être *statique* ou *dynamique*, dépendant du type d'application.

7.5 Autres exemples

Nous désirons conclure notre survol des mécanismes tirant parti des caractéristiques de POEtic par trois applications qui furent développées par les partenaires du projet. Nous n'entrerons pas dans les détails, laissant le lecteur les trouver dans les références indiquées.

7.5.1 Autoréparation

L'autoréparation est un des points de recherche important dans le cadre des systèmes bio-inspirés. La nature est plus qu'efficace pour résoudre des pannes, comme nos corps qui sont capables de cicatriser rapidement, et de faire face aux maladies. Les systèmes artificiels ne sont que rarement capables de gérer les pannes de manière efficace. Une des solutions matérielles, notamment utilisée en aéronautique, se base sur la duplication ou la *triplication* du système critique. Deux unités permettent alors de détecter une panne, et trois offrent la possibilité, via un vote, de définir quelle unité est défectueuse.

Les circuits reconfigurables ne sont pas intrinsèquement tolérants aux pannes, si ce n'est le tableau de cellules embryoniques qui furent développé au Laboratoire de Systèmes Logiques [142]. Le circuit POEtic n'offre pas de mécanisme matériel de gestion des pannes comme c'est le cas dans Embryonique, mais des travaux ont permis de montrer qu'il était possible de démontrer des mécanismes d'autoréparation en tirant parti de la reconfiguration partielle et du routage dynamique [20]. Deux approches ont été développées et implémentées grâce au simulateur du sous-système organique.

La première a vu la création d'un système de type embryonique, où une cellule contient un génome décrivant l'organisme entier. Un mécanisme de différentiation semblable à celui que nous avons décrit précédemment laisse la cellule sélectionner la partie du génome correspondant à son identifiant. Les cellules sont alors reliées entre elles en une chaîne, de la même manière que lors de la croissance basée sur un identifiant à une dimension. Deux parties fonctionnelles calculent les sorties de la cellule, et une comparaison est effectuée, afin de détecter une erreur. Lorsqu'une faute survient, la cellule concernée envoie un message à la cellule suivante, avec son identifiant. La cellule atteinte met alors à jour son identifiant et fait suivre le décalage. Toutes les cellules faisant partie de l'organisme, et étant positionnées après la cellule touchée, modifient donc leur comportement, et une nouvelle cellule totipotente est exploitée de manière à combler le manque de la cellule morte.

La deuxième réalisation se base sur la duplication d'une cellule, à la manière que nous avons déjà présentée. Aucun génome n'est présent dans la cellule, mais trois parties fonctionnelles permettent de détecter une faute et de sauvegarder l'état du système. Lorsqu'une faute survient, la cellule bloque le fonctionnement de la partie fonctionnelle de toutes les cellules de l'organisme, en utilisant l'*enable* moléculaire global. Elle se connecte ensuite à une nouvelle potentielle cellule, qui reconstruit trois nouvelles parties fonctionnelles en se basant sur les bits de configuration envoyés par la cellule endommagée. Par ce mécanisme, la fonctionnalité de l'organisme est garantie, puisque son état est sauvegardé en tout temps. La configuration initiale du circuit doit ici contenir l'ensemble des cellules utiles à l'implémentation d'un organisme complet, ainsi que des parties de cellules capables de prendre le relais lors de fautes. Ces parties de cellules sont également couplées à des molécules dont les bits de configuration forment une chaîne qui peut être reconfigurée par la cellule endommagée.

Ces deux approches ont exhibé une manière d'exploiter l'auto-reconfiguration partielle des molécules, ainsi que le routage distribué de POEtic. Il est bien clair cependant que ces deux systèmes partent du principe qu'aucune erreur ne peut survenir dans les unités de routage, ni dans les molécules responsables de la gestion de l'autoréparation, aucun mécanisme matériel n'ayant été mis en place dans POEtic pour parer à des erreurs éventuelles.

7.5.2 Synthèse vocale

Un système de synthèse vocale basée sur une grille d'éléments reliés à leurs quatre voisins a été développé par Cooper [44], à York. De l'évolution est appliquée à la forme de la grille, qui reçoit en entrée du bruit blanc, et est capable de générer un signal sonore approchant une courbe de fréquence présentée au système. L'implémentation est réalisée à l'aide de 16 molécules par élément, qui sont reliés entre eux via le routage dynamique.

Notons toutefois que l'architecture cellulaire ne tire pas parti des spécificités du sous-système organique, mais que cette réalisation exploite la présence du microprocesseur pour accélérer l'évolution et la configuration du circuit.

7.5.3 Neurone à impulsion

Finalement, sur l'axe épigénétique, un nouveau type de neurones à impulsion a été développé par les biologistes [69], puis implémenté grâce aux molécules du circuit POEtic [238]. 80 molécules sont nécessaires à la réalisation d'un seul neurone, qui est une implémentation sérielle de celui décrit par les biologistes. Cette taille non négligeable implique qu'un seul neurone peut être présent dans un circuit, qui possède 144 molécules. De ce fait, la possibilité de relier plusieurs circuits entre eux est une caractéristique essentielle de POEtic, qui offre ainsi un tableau reconfigurable de taille acceptable. Enfin, le réseau de neurones basé sur ces cellules exploite donc le routage dynamique pour la création des interconnexions cellulaires.

7.6 Conclusion

Après avoir présenté le circuit POEtic, nous avons, dans ce chapitre, tenté de mettre en avant des applications ou mécanismes tirant parti de ses caractéristiques le différenciant des FPGAs commerciaux. Cependant, toutes ces implémentations, hormis le prototype PO, ont été effectuées à l'aide du logiciel, développé par nos soins, qui permet de concevoir des systèmes pour POEtic, en simulant le comportement du sous-système organique. A l'écriture de ces lignes, le circuit final a été reçu, mais n'est pas encore opérationnel, ni même testé. Nous ne pouvons donc qu'espérer qu'il fonctionne correctement, et que toutes ces applications puissent être, dans un futur proche, chargées dans le circuit POEtic.

Concernant les caractéristiques de POEtic, nous pourrions suggérer quelques améliorations, dans le cas où une deuxième version devait voir le jour. Premièrement, il serait intéressant de disposer d'une auto-configuration complète des molécules, ce qui permettrait aux mécanismes de duplication de ne pas nécessiter que des molécules doivent être pré-configurées sur le plan des 8 bits de configuration fixes. Deuxièmement, à la manière d'Embryonique, un mécanisme d'autoréparation matériel pourrait être envisagé, afin d'aider à la réalisation de systèmes réellement tolérants aux pannes.