# Madeo+ : génération de netlists Verilog pour l'eFPGA M2000 Flexeos

- Rapport technique -

Lab-STICC/AS - Université de Bretagne Occidentale
CNRS UMR 3192
loic.lagadec@univ-brest.fr

## Introduction

Madeo+ est un *front-end* de synthèse reciblable pour des architectures reconfigurables de différentes natures (grain logique, organisation, modèle de programmation). Il offre un flot direct de synthèse d'une spécification applicative haut-niveau, programmée dans des langages *mainstream*, vers des netlists Verilog pour la cible eFPGA Flexeos.
Les résultats obtenus montrent que les netlists produites par Madeo+ allouent une quantité inférieure de mfcs après optimisation par rapport à une netlist produite par le traducteur EDIF vers Verilog de M2000.
La figure 1 montre le flot de Madeo+ pour la génération des netlists.

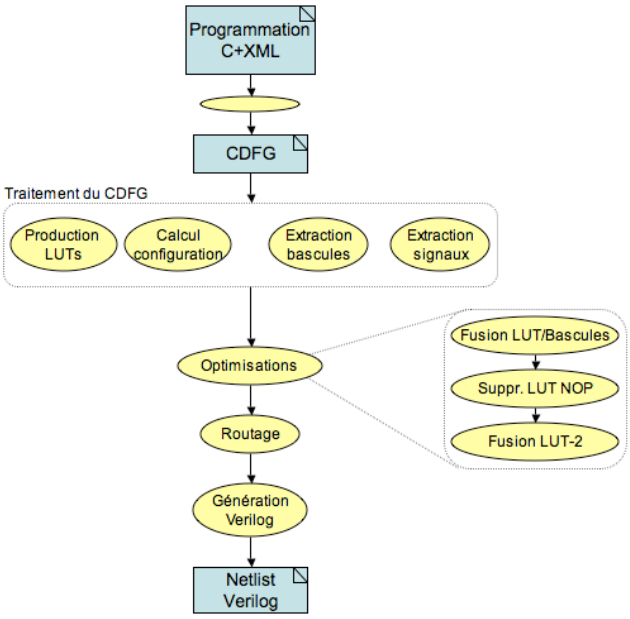


*Figure 1 : Flot global de l'outil Madeo+.*

Madeo+ prend en entrée une spécification de l'application organisée autour de trois parties correspondant à :

1. Spécification des processus de calcul dans une variante de syntaxe C orientée flot de donnée
2. Spécification des schémas d'accès aux mémoires locales en XML
3. Assemblage des différents processus par une description XML

À partir de cette description un graphe de flot de contrôle et de donnée est généré pour servir de support à la synthèse. Les nœuds de calcul du graphe sont synthétisés en logique ou *mappés* sur des opérateurs de la cible puis l'ensemble du graphe est ordonnancé. Le CDFG obtenu correspond à une description RTL de l'application proche de la netlist finale. Pour finir, le CDFG de bas niveau est utilisé pour instancier les cellules de la librairie Flexeos afin de créer un graphe de LUTs. Ce graphe est ensuite optimisé pour minimiser l'occupation des ressources. Une fois ces étapes terminées la netlist finale est générée.

## Aperçu de l'outil

L'application est programmée C et XML dans n'importe quel environnement de développement. Par exemple dans l'éditeur Emacs (cf. figure 2).

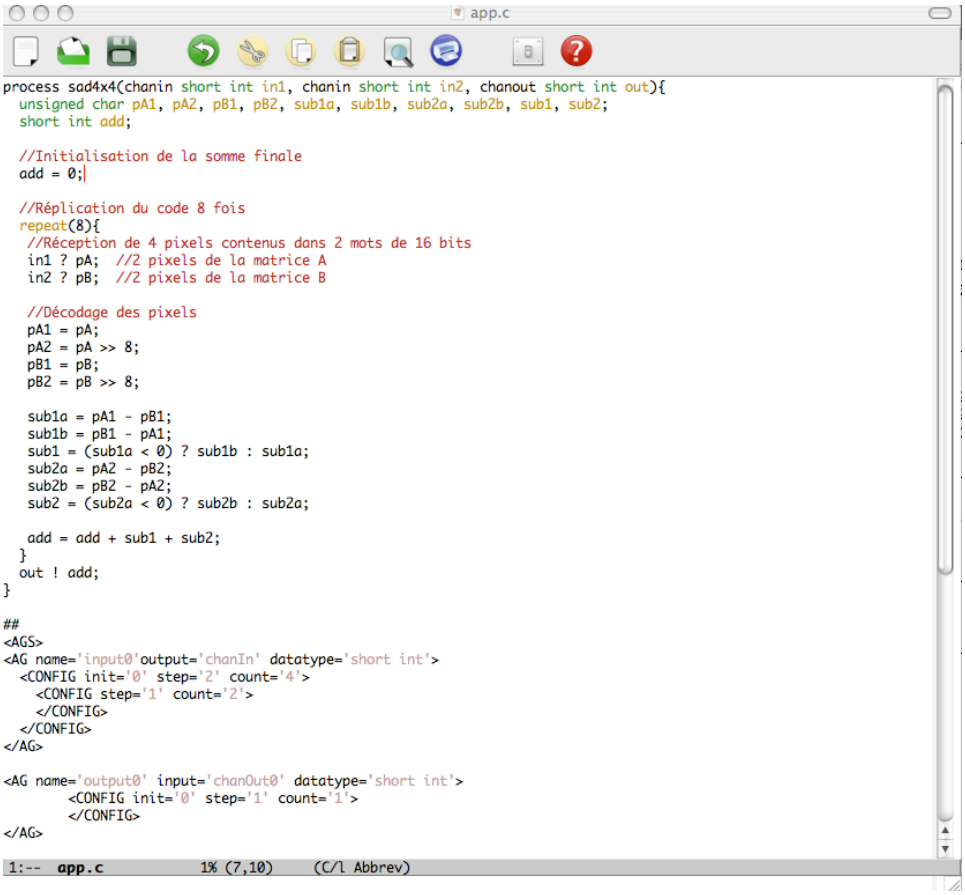


*Figure 2 : Programmation de l'application dans un environnement standard.*

Les trois parties de l'application sont décrites dans un même fichier ou dans plusieurs fichiers séparés. La description est prise en entrée de Madeo+ qui produit une représentation intermédiaire (un graphe de flot contrôle et de donnée ou CDFG). Cette représentation est qualifiée de haut-niveau car elle est indépendante d'une cible matérielle particulière.
Madeo+ fournit une interface graphique qui donne une vision en arbre du CDFG (cf. figure 3 partie A). Le CDFG haut-niveau est ensuite *mappé* sur la cible Flexeos. Cette étape inclut également la génération automatique d'un harnais mémoire pour contrôler les échanges de données entre l'accélérateur et les mémoires locales. Le harnais mémoire est reciblable sur d'autres architectures par une prise en compte des interfaces disponibles.

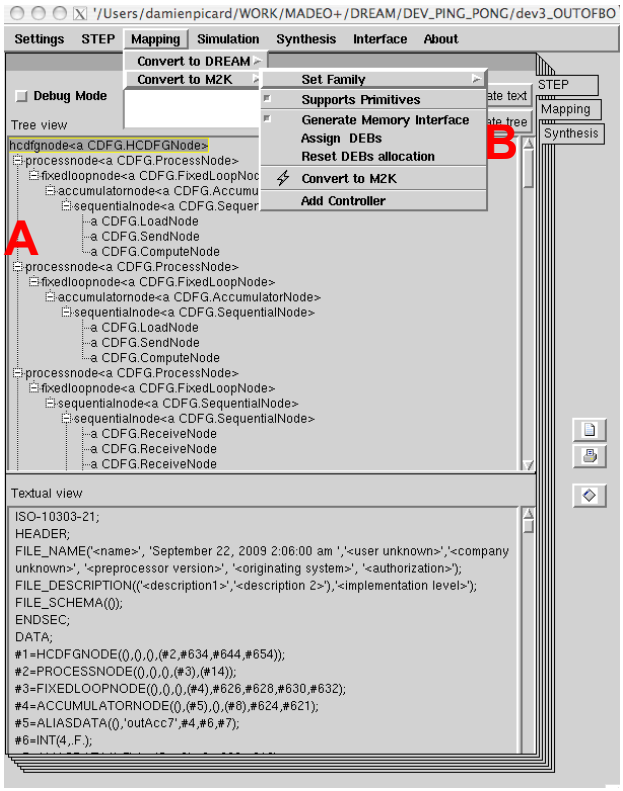


*Figure 3: Interface de visualisation du CDFG.*

Après le *mapping* le CDFG est lié à la cible et est qualifié de bas-niveau. Les propriétés de chaque nœud du graphe sont affichées dans la partie inférieure de l'interface (cf. figure 4 partie B) permettant ainsi une exploration complète de l'application. Cette capacité d'exploration est indispensable lors des phases de *debug*.

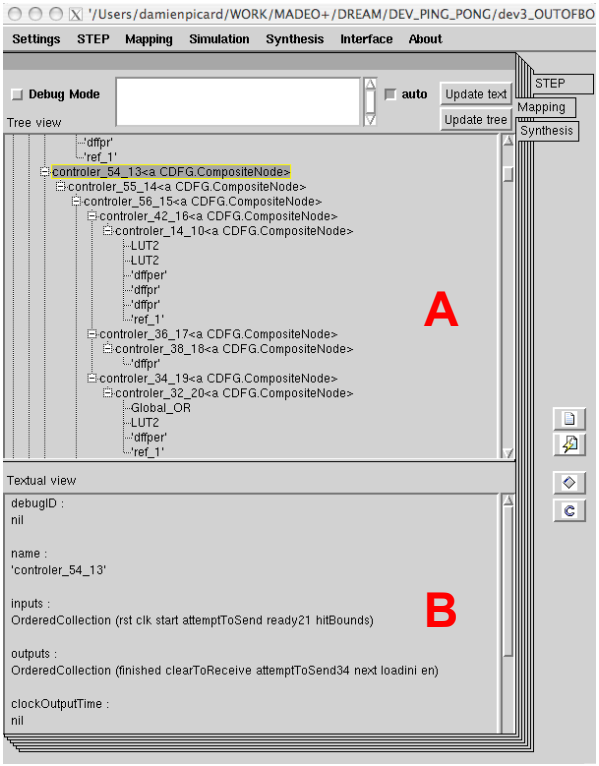


*Figure 4 : Visualisation du CDFG bas-niveau.*

La netlist Verilog est générée à partir du CDFG bas-niveau. Quatre types d'optimisation sont disponibles comme montré par la figure 5. Ces optimisations sont activables ou désactivables suivant les besoins du concepteur. En phase de *debug* il est préférable de désactiver toutes les optimisations pour avoir accès à l'ensemble des cellules du *design*.

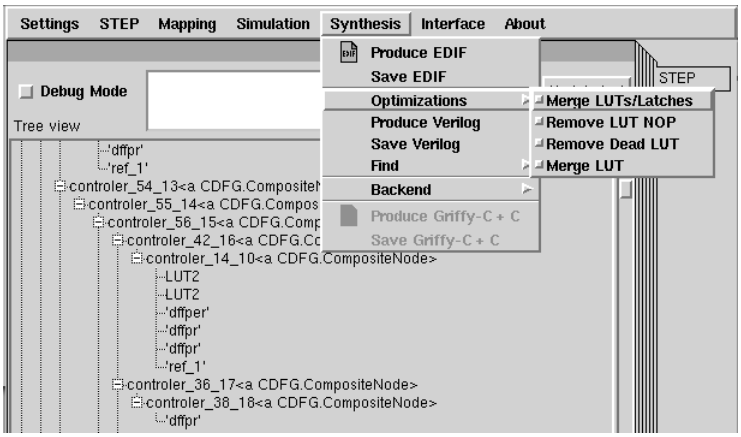


*Figure 5 : Optimisations disponibles.*

Un *log* est disponible sur la génération du Verilog détaillant les types de cellules utilisées et les résultats des optimisations (cf. figure 6).

```
###INFO: Cell summary
      ref_0     11
      ref_1     18
      lut1      7
      lut2      33
      lut3      230
      lut4      86
      dffp      1
      dffpr     18
      dffpe     12
      dffper    148
###INFO: Total mfc count: 564
###INFO: Total ref_0 collapsed: 11
###INFO: Total merged latch: 145
###INFO: Total single latch instantiated: 34
###INFO: Total mfc count after LUT/latch merging: 408
###INFO: Total mfc count after removing NOP LUTs: 406
###INFO: Total unconnected nets: 8
###INFO: Total mfc count after removing dead LUTs: 399
###INFO: Total mfc count after LUT2 merging: 354
###INFO: Generation done
######
```

*Figure 6 : Log de génération de la netlist.*

Madeo+ garde un lien bidirectionnel entre les cellules de la netlist et le CDFG. Il est donc possible à partir d'un nom de signal ou de LUT de retrouver le nœud correspondant dans le CDFG et à partir d'un nœud de retrouver le nom de son élément dans la netlist (cf. figure 7).

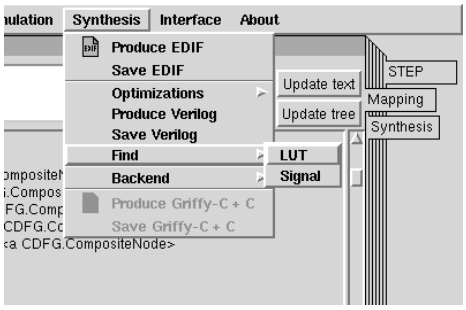


*Figure 7 : Un lien bidirectionnel est conservé entre le CDFG et la netlist.*

Par exemple, pour la *lut3_276* et le signal *sig_925* (cf. figure 8).

```
    defparam \lut3_280 .equate = 16'b1001011010010110;
mfc_lut \lut3_279 (.I1(sig_922), .I2(sig_167), .I3(1'b0), .I4(1'b0), .S(sig_955));
    defparam \lut3_279 .equate = 16'b1001011010010110;
mfc_lut \lut3_278 (.I1(sig_923), .I2(sig_160), .I3(1'b0), .I4(1'b0), .S(sig_956));
    defparam \lut3_278 .equate = 16'b1001011010010110;
mfc_lut \lut3_277 (.I1(sig_924), .I2(sig_113), .I3(1'b0), .I4(1'b0), .S(sig_957));
    defparam \lut3_277 .equate = 16'b1001011010010110;
mfc_lut \lut3_276 (.I1(sig_925), .I2(sig_83), .I3(1'b0), .I4(1'b0), .S(sig_958));
    defparam \lut3_276 .equate = 16'b1001011010010110;
mfc_lut \lut3_275 (.I1(sig_926), .I2(sig_89), .I3(1'b0), .I4(1'b0), .S(sig_959));
    defparam \lut3_275 .equate = 16'b1001011010010110;
mfc_lut \lut3_274 (.I1(sig_927), .I2(sig_139), .I3(1'b0), .I4(1'b0), .S(sig_960));
    defparam \lut3_274 .equate = 16'b1001011010010110;
```

*Figure 8 : Extrait de la netlist générée.*

Les noms sont saisis par le concepteur (cf. figure 9).

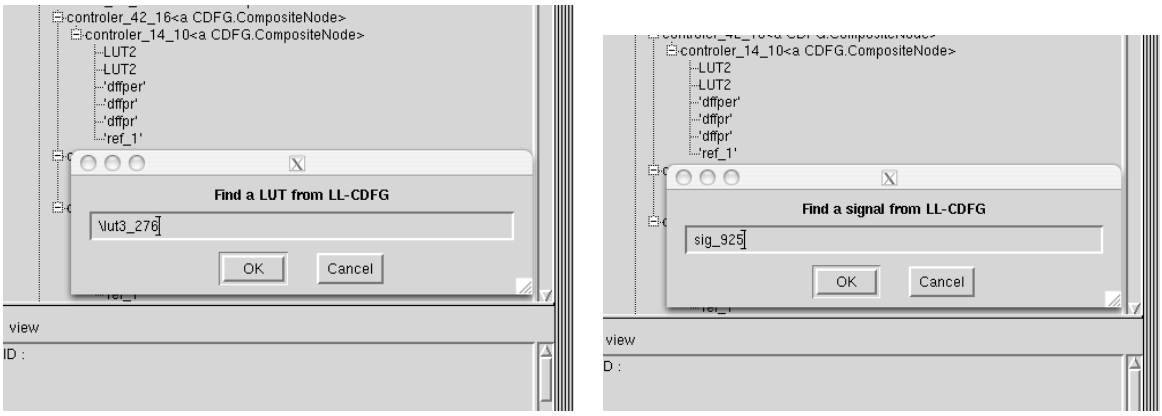


*Figure 9 : Recherche d'une LUT ou d'un signal à partir de son nom.*

Si les éléments existent alors des fenêtres d'exploration s'ouvrent donnant accès à l'ensemble des informations concernant les éléments. Pour une LUT ses entrées/sorties et son équation logique (cf. figure 10).

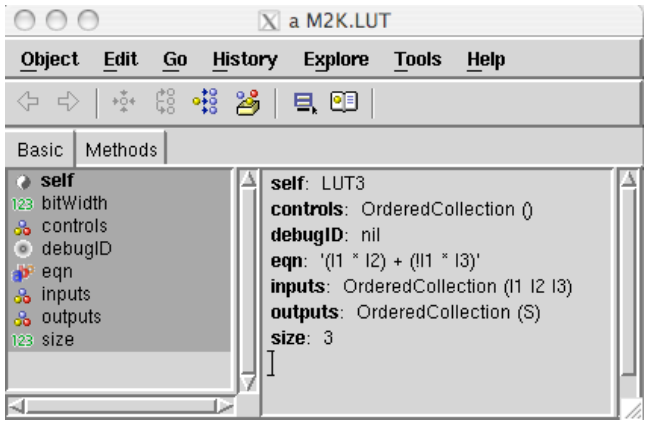


*Figure 10 : Détails d'une LUT du CDFG bas-niveau.*

Pour un signal ses entrées/sorties (cf. figure 11).

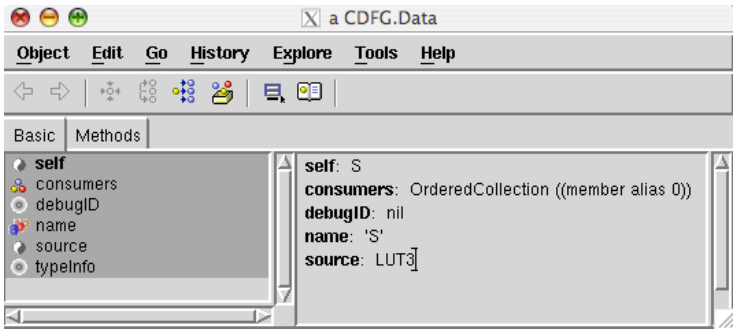


*Figure 11 : Détails d'un signal.*

Les entrées/sorties peuvent également être explorées pour accéder aux autres nœuds connectés.

# Description de l'application

Le frontal de programmation de Madeo+ permet à l'utilisateur de spécifier les applications sous forme de réseaux de processus interconnectés par des canaux de type CSP (synchronisation par *rendez-vous*) tels que définis par Hoare. Une application est composée d'un ou plusieurs processus de calcul alimentés par des processus de communication similaires à des générateurs d'adresse qui lisent et écrivent les données dans des mémoires locales.
Les fonctions à accélérer sont écrites dans une variante du langage C (proche de Handel-C) et les schémas d'accès mémoire en XML. Ces deux langages sont largement connus et standardisés évitant ainsi l'apprentissage de nouveaux langages pour le concepteur d'application. De plus, ils bénéficient de nombreux supports dans les librairies de programmation permettant la génération des fonctions accélérées et des générateurs d'adresse par des outils tiers. Ces deux langages sont donc utilisables comme format d'échange neutre entre les outils.

## Exemple de description d'une fonction de calcul

La fonction considérée ici est une somme de différences absolues (SAD) couramment utilisée dans les algorithmes de codage vidéo. La SAD est effectuée entre deux matrices, A et B, de taille 4x4 avec (r, s) qui représente le *motion vector* :

$$SAD(x, y, r, s) = \sum_{i=0}^{i=3} \sum_{j=0}^{j=3} |A_{(x+i,\, y+j)} - B_{((x+r)+i,(y+s)+j)}|$$

Le code C de la fonction SAD est le suivant :

```
process sad4x4(chanin short int in1, chanin short int in2, chanout short int out){
  unsigned char pA1, pA2, pB1, pB2, sub1a, sub1b, sub2a, sub2b, sub1, sub2;
  short int add;

  //Initialisation de la somme finale
  add = 0;

  //Réplication du code 8 fois
  repeat(8){
   //Réception de 4 pixels contenus dans 2 mots de 16 bits
   in1 ? pA;  //2 pixels de la matrice A
   in2 ? pB;  //2 pixels de la matrice B

   //Décodage des pixels
   pA1 = pA;
   pA2 = pA >> 8;
   pB1 = pB;
```

```
    pB2 = pB >> 8;

    sub1a = pA1 - pB1;
    sub1b = pB1 - pA1;
    sub1 = (sub1a < 0) ? sub1b : sub1a;

    sub2a = pA2 - pB2;
    sub2b = pB2 - pA2;
    sub2 = (sub2a < 0) ? sub2b : sub2a;

    add = add + sub1 + sub2;
  }
  out ! add;
}
```

Le calcul de SADs pour des matrices de plus grande taille est réalisé par la combinaison de ces fonctions. Par exemple une SAD sur des matrices de taille 16x16 nécessiterait 16 processus de calcul *sad4x4* et un processus (ou plusieurs) relisant les sommes partielles pour calculer le résultat final.

## Exemple de description des processus de communication

Les processus de communication sont équivalents à des générateurs d'adresse matériels qui alimentent en données les fonctions de calcul accélérées. Ils effectuent soit des opérations de lecture (le type du processus est *input*) ou des opérations d'écritures (le type du processus est *output*) dans les mémoires locales connectées à l'accélérateur.
Le code suivant montre un exemple de spécification d'un processus de lecture et d'écriture (balise AG). Le processus *input0* lit une matrice de pixel 4x4 par paquet de 16 bits et envoie les données sur son canal de sortie, *chanIn*. La lecture dans un espace à 2 dimensions est réalisée par l'imbrication hiérarchique des configurations d'accès (balise CONFIG). Le second processus écrit un résultat en mémoire qui correspond au résultat de la fonction *sad4x4*.

```
<AGS>
 <AG name='input0'output='chanIn' datatype='short int'>
  <CONFIG init='0' step='2' count='4'>
    <CONFIG step='1' count='2'>
    </CONFIG>
  </CONFIG>
 </AG>

 <AG name='output0' input='chanOut0' datatype='short int'>
  <CONFIG init='0' step='1' count='1'>
  </CONFIG>
</AG>
```

## Assemblage des processus

Les processus de calcul et de communication sont regroupés dans des étages (balise STAGE) et interconnectés par des canaux de type CSP. À chaque étage est assigné un ensemble de mémoires locales (LMEM) qui permettent de lire/écrire les données reçues/produites. L'exemple suivant montre l'interconnexion des processus dans un étage pour l'application SAD.

```
<NETWORK>
 <STAGE name='SAD' inputLMEMs='LM0, LM1' outputLMEMs='LM2'>
  <AG name='CtrlIn0' connections='Fsad4x4.in1' behaviour='input0' memory='LM0'></AG>
  <AG name='CtrlIn1' connections='Fsad4x4.in2' behaviour='input0' memory='LM1'></AG>
  <FUNCTION name='Fsad4x4' connections='CtrlOut.chanOut0' behaviour='sad4x4'></FUNCTION>
  <AG name='CtrlOut' connections='' behaviour='output0' memory='LM2'></AG>
 </STAGE>
```

Les mémoires locales sont également utilisées pour les communications entre étage. La composition des étages est effectuée dans la balise MAIN.

```
 <MAIN>
  <STAGE name='SAD' inputLMEMs='LMEM1, LMEM2' outputLMEMs='LMEM3'></STAGE>
 </MAIN>
</NETWORK>
```

La figure 2 illustre un équivalent graphique de l'application SAD. La structure est un graphe de flot de donnée ayant pour source et destination les mémoires locales accédées par l'accélérateur.

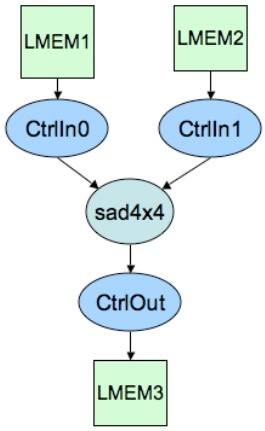


*Figure 12 : Réseau de processus pour l'application SAD.*

Le réseau de processus SAD constitue une brique de base pour traiter de plus grandes matrices de pixels.

# Optimisation de la netlist

## Suppression des LUTs NOP

En fonction du contexte de la synthèse (paramètre du circuit) certaines LUTs peuvent devenir inutiles car effectuant des opérations nulles. Nous qualifions de LUT NOP (*No OPeration*) une LUT qui effectue une opération d'identité entre une seule de ses entrées (les autres étant forcées à 0) et sa sortie. Par conséquent, ces LUTs sont supprimées et remplacées par une connexion sans affecter le bon fonctionnement du circuit.

## Fusion des LUTs et des bascules

Le LL-CDFG décrit une application au niveau RTL et contient des éléments bascules. Or, l'architecture Flexeos F5 ne permet pas l'instanciation de bascules puisqu'elles sont obligatoirement connectées à la sortie d'une LUT. L'étape de fusion consiste à former des couples LUT et bascule connectées entre elles. En fonction du type de la bascule une cellule *mfc* correspondante est instanciée. Elle a pour sortie celle de la bascule et pour entrées celles de la LUT.

## Fusion de LUTs à 2 entrées

Cette opération consiste à fusionner deux LUTs à deux entrées qui forment un chemin combinatoire. Le résultat de cette optimisation est le remplacement des deux LUTs par une seule LUT à trois entrées au comportement équivalent (voir figure 13).

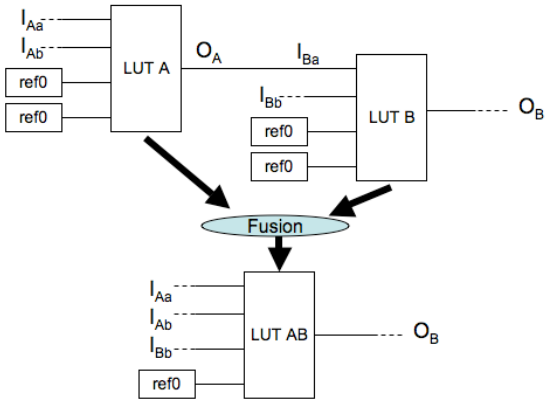


*Figure 13 : Fusion de deux LUT-2 remplacées par une LUT-3.*

# Évaluations et résultats

Les évaluations se fondent sur les résultats donnés par Madeo+ et les outils M2000 Flexeos pour la génération de netlists Verilog. L'outil M2000 prend comme entrée une netlist au format EDIF et produit un équivalent Verilog prêt pour la synthèse. Madeo+ produit directement la netlist Verilog à partir du CDFG bas-niveau. Nous prenons comme exemple de référence la synthèse d'un incrémenteur 32 bits.

Par défaut il n'est pas possible de contrôler les optimisations appliquées par l'outil M2000, en revanche les optimisations de Madeo+ sont activables ou désactivables individuellement. De ce fait chaque optimisation de Madeo+ peut être comparée au résultat de l'outil M2000.

Les résultats présentés par la figure 15 sont normalisés par rapport au nombre de cellules *mfc* occupés par un incrémenteur 32 bits généré par l'outil M2000 (394 *mfcs* sont alloués incluant les générateurs d'adresse, la fonction de calcul et le harnais mémoire). Les résultats positifs indiquent le nombre de LUTs additionnelles et les résultats négatifs indiquent le gain par rapport à l'incrémenteur de référence.

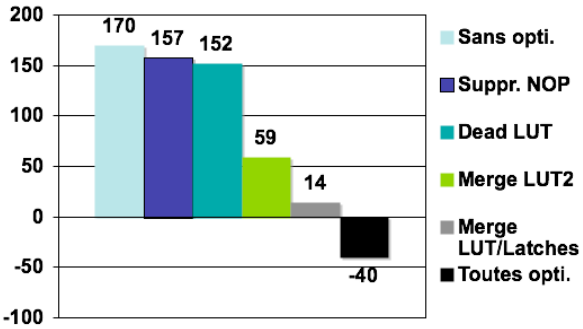


*Figure 14 : Résultats normalisés donnant le gain en nombre de LUTs pour un incrémenteur.*

La figure 15 montre l'impact de chaque optimisation sur la netlist. On constate que sans l'application d'optimisations nous obtenons un surplus de 170 *mfcs* par rapport à l'incrémenteur de référence. Cela est principalement dû à l'intanciation séparée des LUTs et des bascules, ce qui a pour conséquence d'augmenter le nombre de ressources utilisées. La fusion des LUTs et des bascules permet de réduire jusqu'à 92% le nombre de *mfcs* supplémentaires du cas sans optimisations et donne un surplus de 3,5% par rapport au résultat de référence. Les suppressions de LUTs NOP et de LUTs mortes ne concernent qu'un nombre restreint de LUTs ce qui limite les gains. En revanche la fusion des LUT-2 permet de réduire jusqu'à 65% le nombre de *mfcs* du cas sans optimisations avec un surplus de 15%. Enfin la

combinaison de toutes les optimisations donne un nombre de *mfcs* alloués inférieur à l'outil M2000. Le gain total est de 40 *mfcs* ce qui représente une amélioration de 10%.

# Bibliographie

[1] Loïc Lagadec, Damien Picard et Bernard Pottier, « Dynamic System Reconfiguration in Heterogeneous Platforms, The MORPHEUS Approach », dans chapitre 13: Spatial Design, Springer, 2009.
[2] Loïc Lagadec et Damien Picard, « Software-Like Debugging Methodology for Reconfigurable Platforms (short paper) » dans RAW, International Symposium on Parallel & Distributed Processing, Italie, 2009.
[3] Damien Picard et Loïc Lagadec, « Multi-Level Simulation of Heterogeneous Reconfigurable Platforms », dans International Journal of Reconfigurable Computing, 2009.