Mesh of Tree: Unifying Mesh and MFPGA for Better Device Performances

Zied Marrakchi, Hayder Mrabet, Christian Masson and Habib Mehrez
LIP6, Université Pierre et Marie Curie
4, Place Jussieu, 75252 Paris, France
zied.marrakchi@lip6.fr

Abstract

In this paper we present a new clustered mesh FPGA architecture where each cluster local interconnect is implemented as an MFPGA tree network [6]. Unlike previous clustered mesh architectures, the mesh of tree allows us to consider large clusters sizes (thanks to MFPGA depopulated local interconnect). Experimentation shows that we obtain a reduction of 14% in switches number and 2 times in the placement and routing run time. Furthermore, compared to MFPGA, the mesh of tree achieves full routability of all MCNC benchmarks since we can easily control both clusters LUTs occupation and mesh channel width.

1. Introduction

Modern Mesh FPGA architectures are based on a clustered architecture, where a number of lookup tables (LUTs) are grouped together to act as the configurable logic block. The motivation of using clusters is manifold: to reduce area, to reduce critical-path delay, and to reduce CAD tool runtime [2] [3]. This trend is followed by some FPGAs from Xilinx (the Virtex and Spartan families) and Altera (the Stratix and Cyclone families). All of these FPGAs are based on clusters of 4-input lookup tables. In some FPGAs, such as Altera's APEX family, these internal cluster connections are fully populated or fully connected. This is equivalent to employing a full crossbar: a crosspoint switch exists at intersection point of every LUT input and every cluster input or feedback connection. Such a high degree of connectivity makes routing easier, but it has significant area overhead. This penality is increasing especially in the case of architectures with high clusters sizes.
Our previous studies of MFPGA architecture [6] showed that it leads to better logic density than mesh especially for small benchmark circuits. We have improved the placement strategy to enhance routability and to target benchmark circuits with higher occupation than those considered in [8]. Despite our efforts we found that the proposed architecture cannot deal with circuits exceeding 80% of logic blocks occupation. To deal with both architectures drawbacks we propose in this work to use the MFPGA sparse interconnect as an alternative switch matrix inside the Mesh clusters. Since clusters size is limited, the MFPGA interconnect topology seems to be an interesting local interconnect, which allows us to control the logic occupation inside each cluster.
In this paper section 2 presents a description of the Mesh of Tree architecture. First it presents the mesh interconnect level and then the local cluster interconnect which is similar to the MFPGA connecting networks. Section 3 describes the configuration flow to implement a netlist on the presented architecture. It insists on challenges for performing place-and-route inside clusters (MFPGA). In the experimentation section, based on MCNC benchmark, we compare the Mesh of Tree architecture to the common clustered mesh in term of switches number requirement.

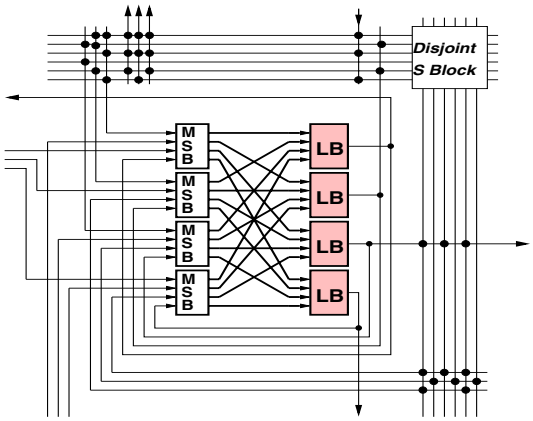

Figure 1. Mesh of Tree

2 Mesh of MFPGA Architecture

The architecture that we propose has a mesh of tree interconnect topology. It starts with a mesh of nodes and builds

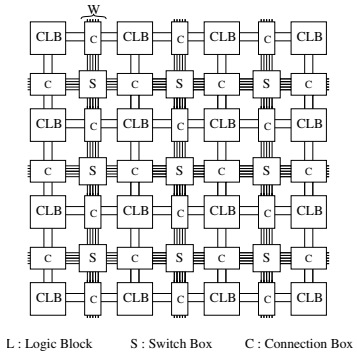

Figure 2. Mesh architecture

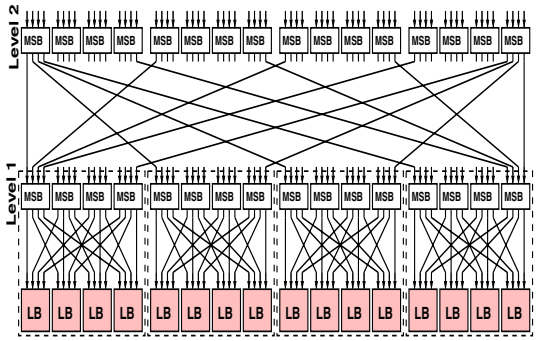

Figure 3. Downward Network

a separate hierarchical network along each row and column cluster. The resulting network corresponds to a mesh of clusters where each local interconnect is equivalent to a MFPGA interconnect. Figure 1 shows a cluster local interconnect (here with a simple one level MFPGA topology) and how it is connected to other clusters. We can consider that netlist implementation on this architecture can be run in two stages:

- Mesh stage: In this stage clusters are considered as black boxes with i inputs and j outputs. The initial netlist is partitioned into N independent sub-netlists where N corresponds to the number of clusters of the mesh architecture.
- MFPGA stage: Each one of the N sub-netlists is mapped seperately in a cluster. Since cluster local interconnect is similar to MFPGA hierarchical interconnect, clusters will be referred as MFPGA.

Then the clusters netlist is placed and routed on the mesh using VPR. In the following, both architecture stages and their corresponding tools will be described.

2.1 Mesh routing interconnect

As shown in figure 2 mesh-based architecture is composed of clustered logic blocks (CLB), switch blocks (S), connection blocks (C), and I/O blocks. Interconnection between clusters is formed by the C and S blocks, comprising the horizontal and vertical routing channels. The C block is the region where the CLB input and output pins connect to the routing channels. The S block is where connections are made between the horizontal and vertical routing channels, allowing nets to turn corners or extend farther along the channel. Each routing channel contains W parallel tracks of wires, where W is called the channel width. The same width is used for all channels. The cluster inputs are connections from the external routing, carrying signals from other clusters into this one.

2.2 MFPGA routing interconnect

The MFPGA hierarchical interconnect were described previously in [6]. In the following we present a more detailed description which will be very useful to extract architecture routing constraints and to present challenges on performing place-and-route.

MFPGA contains LUT-based logic blocks and two unidirectional connecting networks. Logic Blocks (LBs) and interconnect are organised into levels. Let $nb\ell$ denote the number of levels of a given architecture. In each level we have a set of clusters, and C denotes the set of clusters in all levels. A cluster with index c belonging to level ℓ is noted by $cluster(\ell, c)$. Each $cluster(\ell, c)$ where $\ell \geq 1$ contains a set of MSBs (Mini Switch Boxes) and 4 sub-clusters. Sub-clusters of $cluster(\ell, c)$ are $cluster(\ell - 1, 4c + i)$ where $i \in \{0, 1, 2, 3\}$. The MSB with index m belonging to $cluster(\ell, c)$ is denoted $MSB(\ell, c, m)$ where $m \in \{0, \ldots, 4^\ell - 1\}$. Each MSB contains 4 inputs driven by the upper level and 1 feedback coming from a leaf output pin. Each cluster in level ℓ contains $nbMSB(\ell) = 4^\ell$. Each cluster in level 0 is denoted $cluster(0, c)$ or $leafcluster(c)$ and corresponds to the Logic Block (LB) and contains 4 inputs, 1 output, no MSBs and no sub-cluster. Each $cluster(\ell, c)$ where $\ell < nb\ell - 1$ has an owner in level ℓ' where $\ell' > \ell$ denoted $cluster(\ell', c \div 4^{(\ell' - \ell)})$. We define for each $cluster(\ell, c)$ a position inside its owner in level $\ell + 1$ (direct owner) by the following function:

$pos : C \longrightarrow \{0, 1, 2, 3\}$

$cluster(\ell, c) \longmapsto c \mod 4$

Two clusters belonging to level ℓ and having the same owner at level $\ell + 1$ have two different positions. To get the cluster owner in level ℓ' of $cluster(\ell, c)$ $(\ell < \ell' \leq nb\ell - 1)$ we define the function:

$owner : C \times \mathbb{N} \longrightarrow C$

$(cluster(\ell, c), \ell') \longmapsto cluster(\ell', c \div 4^{\ell' - \ell})$

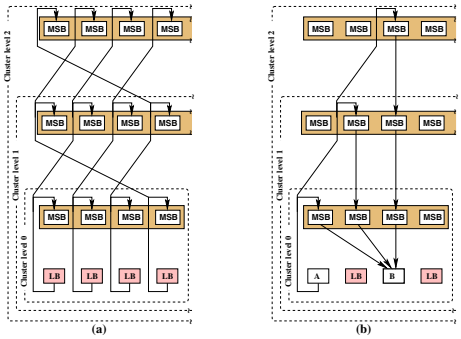

Figure 4. Upward Network

2.2.1 Downward Network

Figure 3 shows a sparse downward network based on unidirectional MSBs. The downward interconnect topology is similar to the butterfly fat tree. Each MSB of a $cluster(\ell, c)$ where $\ell > 1$ is connected to each sub-cluster in one and only one input pin. We say an $MSB(\ell', c', m')$ is the successor of an $MSB(\ell, c, m)$ where $0 < \ell' < \ell$ if there is a downward directed path from $MSB(\ell, c, m)$ to $MSB(\ell', c', m')$. The path between an MSB and its successor is unique.
We define the function:
$Mod_i : \mathbb{N} \longrightarrow \mathbb{N}$
$m \longmapsto m \mod nbMSB(i)$
Thus each $MSB(\ell, c, m)$ has a successor in each sub-cluster belonging to level ℓ' $MSB(\ell', c', m')$ where $0 < \ell' < \ell$, with:

$$m' = Mod_{\ell'} \circ \cdots \circ Mod_{\ell-1}(m) \quad (1)$$

2.2.2 Upward Network

We propose to connect the output signals of leaf clusters to specific MSBs of upper levels. Thus for each logic block output, we define a list of feedbacks. Each one enables the output to reach an MSB in a particular level. The way feedbacks are distributed has an important impact on the structure routability. Connecting an output of a leaf cluster to MSBs with different indexes increases the number of paths from a source to a destination. This specific distribution is described in figure 4-(a). Figure 4-(b) shows how the cluster leaf 'A' output can reach the cluster leaf 'B' inputs using different paths. Each leaf cluster $cluster(0, c)$ is connected to one and only one $MSB(\ell, c', m)$ in level $\ell > 0$. c' is the index of the owner of cluster(0,c) in level ℓ: $c' = c \div 4^{\ell}$; m is given by:

$$\begin{cases} m = (pos(cluster(0, c)) + \ell - 1) \mod 4+ \\ \sum_{j=1}^{\ell-1} pos(owner(cluster(0, c), j)) \times nbMSB(j) \end{cases} \quad (2)$$

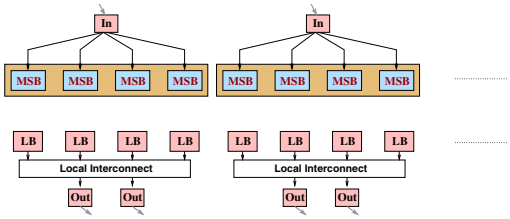

Figure 5. Connection with outside

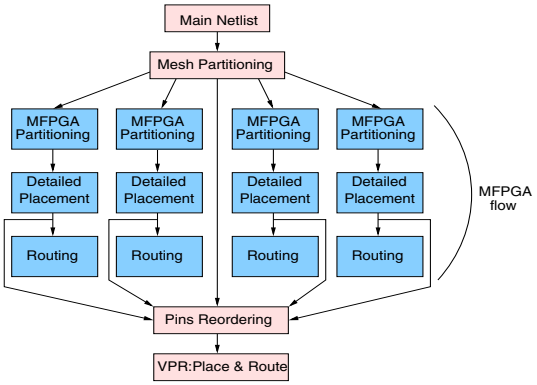

Figure 6. Mesh of Tree configuration flow

2.2.3 Connection with outside

Output pads are clustered with the logic blocks at $level0$. The number of output pads per cluster can be varied to obtain the best design fit. We use a local interconnect between the logic block outputs and the output pads. Input pads are connected directly to MSBs of the highest level. In this way each Input pad can reach all logic blocks. To add flexibility, as shown in figure 5, an input pad can be connected to more than one MSB. This enables pads to reach logic blocks from different paths and in different pins.

3 Configuration flow

In the following we present the different steps to implement a netlist on the Mesh of Tree architecture.

3.1 Mesh partitioning

The purpose of the partitioning step is to distribute netlist instances between mesh architecture clusters (subdomains) in order to reduce external communication (cut) and congestion. Since we have a balanced mesh interconnect (the same width is used for all channels), it is both mandatory to match cluster I/O resources and worthwhile to spread the congestion over all the interconnect. Despite the success of multilevel algorithms [5] in producing partitionings in

which the cut is minimized, this cut is not uniformly distributed across the different subdomains. That is, the number of hyperedges that are being cut by a particular subdomain (referred to as the Subdomain Degree) may be significantly higher than that of other subdomains. Therefore it is of great importance to produce partitioning solutions that minimize the cut but also minimize the Maximum Subdomain Degree (MSD). To address this problem we developed a multi-objective partitioning tool, in which the MSD is the highest priority objective and the cut is the second highest. We implemented a solution similar to the direct multi-phase refinement presented in [10]. Here, using hMetis tool, we first generate a partitioning solution with the cut as an objective, next we apply a multi-phase multi-objective refinement with MSD as the highest priority objective. After main netlist partitioning, we obtain a clusters netlist and N subnetlists each one describing how LBs must be connected inside each cluster. In figure 6 the main netlist is partitioned into 4 sub-netlists targeting a clustered mesh architecture with 4 clusters. Each sub-netlist is mapped separately using the MFPGA configuration flow. Then, after a pin reordering of the inter-clusters netlist (in order to match pin assignemnts done at each MFPGA level), this netlist is mapped using VPR place and routing tool.

3.2 MFPGA placement

The MFPGA placement problem can be stated as assigning to each netlist cell a logic block (leaf) in the MFPGA architecture. The way how cells are distributed has an important impact on routability. In fact once cells are placed, the router tries to find a path to connect a source LB (cluster leaf) to its destinations LBs (cluster leaf) using architecture resources. Thanks to the interconnect predictibility provided by this MFPGA architecture we can introduce, in the placement phase, some conditions to limit later conflicts in the routing phase.

3.2.1 Conflict conditions

Definition 1 *There is a resource conflict problem in level ℓ if 2 leaf clusters (or more), such as $cluster(0,c)$ and $cluster(0,c')$ reach a $cluster(\ell,c'')$ on the same pin p.*

Property 1 *The owner in level $\ell+1$ of $cluster(\ell,c'')$ has one and only one $MSB(\ell+1, c''\div 4, m)$ which can reach this $cluster(\ell,c'')$ on pin p.*

Definition 2 *Referring to the previous property, the definition 1 can be stated as:*
There is a resource conflict problem in level ℓ if 2 leaf clusters (or more) such as $cluster(0,c)$ and $cluster(0,c')$ try to reach a $cluster(\ell,c'')$ and have both already reached its owner $cluster(\ell+1, c''\div 4)$ at the same $MSB(\ell+1, c''\div 4, m)$.

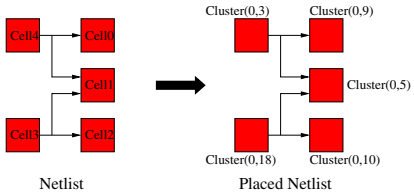

Figure 8. Netlist to route

From definition 2, we can detect a resource conflict by finding 2 leaf clusters reaching the owner cluster of a destination in the same MSB. We consider that $cluster(0,c)$ reaches $cluster(\ell,c'')$ in $MSB(\ell,c'',m)$ using the level ℓ_{up}, and that $cluster(0,c')$ reaches the same cluster destination in $MSB(\ell,c'',m')$ using level ℓ'_{up}. ¿From equation (2) and (1) in this order we get:

$$\begin{cases} m = (pos(cluster(0,c)) + \ell_{up} - 1) \bmod 4 + \\ \quad \sum_{\jmath=1}^{\ell-1} pos(owner(cluster(0,c),\jmath)) \times nbMSB(\jmath) \\ m' = (pos(cluster(0,c')) + \ell'_{up} - 1) \bmod 4 + \\ \quad \sum_{\jmath=1}^{\ell-1} pos(owner(cluster(0,c'),\jmath)) \times nbMSB(\jmath) \end{cases} \quad (3)$$

thus,

$$\begin{cases} m = m' \\ \Updownarrow \\ (pos(cluster(0,c)) + \ell_{up})[4] = (pos(cluster(0,c')) + \ell'_{up})[4] \\ pos(owner(cluster(0,c),\jmath)) = pos(owner(cluster(0,c'),\jmath)) \\ \forall \jmath \in \{1,\dots,\ell-1\} \end{cases} \quad (4)$$

Proof : Equations (3) correspond to the decomposition of m and m' in the base 4 because:
- $0 < (pos(cluster(0,c)) + \ell_{up})[4] < 4$
- $0 < pos(owner(cluster(0,c),\jmath)) < 4 \ \forall j,c$
- $nbMSB(\jmath) = 4^{\jmath}$

Therefore we obtain results presented in equation (4).

Lemma 1 *We say 2 leaf clusters $cluster(0,c)$ and $cluster(0,c')$ are in conflict to drive a common destination $cluster(\ell,c'')$* **if and only if:**

$$\begin{cases} pos(cluster(0,c)) - pos(cluster(0,c')) = (\ell'_{up} - \ell_{up})[4] \\ pos(owner(cluster(0,c),\jmath)) = pos(owner(cluster(0,c'),\jmath)) \\ \forall \jmath \in \{1,\dots,\ell\} \end{cases}$$

3.2.2 Placement example

We refer to the netlist presented in figure 8. We propose to place cells as shown in figure 7. In this example $cell0$, $cell1$, $cell2$ and $cell3$ are placed respectively in $cluster(0,9)$, $cluster(0,5)$, $cluster(0,10)$,

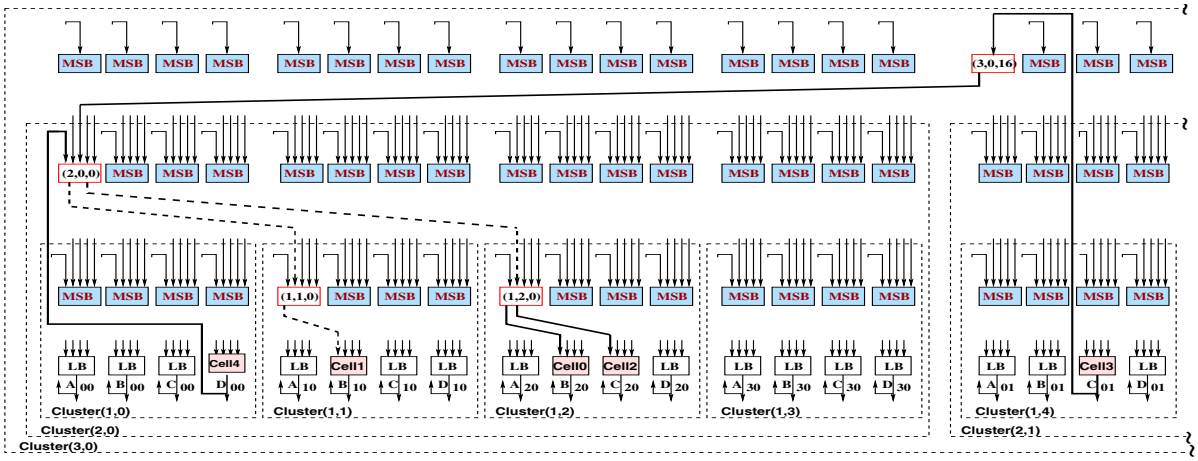

Figure 7. Detailed placement example

$cluster(0,18)$, and $cluster(0,3)$.
Referring to the clustered netlist, $cluster(0,18)$ and $cluster(0,3)$ have a common destination $cluster(0,5)$ at level 0.
$cluster(1,2)$ and $cluster(1,1)$ are also two common destinations at level 1. We check now referring to Lemma 1 whether there is a resource conflict to connect both sources to the three destinations. We propose to use the lowest possible level to connect a source to its destinations. To reach $cluster(0,5)$, $cluster(0,3)$ must go up to level 2 ($\ell_{up} = 2$) and $cluster(0,18)$ to level 3 ($\ell'_{up} = 3$).
Since $pos(cluster(0,3)) = 3$ and $pos(cluster(0,18)) = 2$, we get $pos(cluster(0,3)) - pos(cluster(0,18)) = \ell'_{up} - \ell_{up}$. Thus the condition of lemma 1 is satisfied and there is a resource conflict in level 0 to reach $cluster(0,5)$. The second common destination is $cluster(1,2)$. To reach this destination, $cluster(0,3)$ must be connected up to level 2 ($\ell_{up} = 2$) and $cluster(0,18)$ to level 3 ($\ell'_{up} = 3$).
Since $pos(cluster(0,3)) = 3$ and $pos(cluster(0,18)) = 2$, we get $pos(cluster(0,3)) - pos(cluster(0,18)) = \ell'_{up} - \ell_{up}$. Thus the first condition in lemma 1 is satisfied. We check now the second condition of lemma 1 since destination $cluster(1,2)$ belongs to level 1 ($\ell > 0$). We have $owner(cluster(0,3),1) = cluster(1,0)$ and $owner(cluster(0,18),1) = cluster(1,4)$.
Since $pos(cluster(1,0)) = pos(cluster(1,4)) = 0$ the second condition of lemma 1 is verified too. Thus there is a resource conflict at level 1 to connect $cluster(0,3)$ and $cluster(0,18)$ to $cluster(1,2)$.
We have the same problem with the third common destination $cluster(1,1)$. The routing solution of the placed netlist using the lowest levels is presented in figure 7. The dashed arrows present the resource conflicts. To prevent resource conflicts we propose:
- To change positions of the leaf cluster sources.
- To change positions of the sources owners in level 1.
When we try to resolve a congestion problem to reach a destination we can introduce unexpected problems to reach other destinations. The aim of the following sections is to present a method to model all placement constraints and to perform the optimal positions assignement.

3.2.3 Partitioning

The way in which we distribute logic blocks between MFPGA clusters has an important impact on routing congestion reduction. Based on the upward interconnect specificity we notice that the number of different paths to connect a source to a destination depends on their enclosing clusters. If they are packed in the same cluster, the source can use more levels to reach its destination and therefore more paths. From this remark we can consider that the netlist cut reduction is an important factor for routability improvement.
A second partitioning objective is deduced from routability conditions presented in Lemma 1. Let us consider 2 leaf sources $cluster(0,c)$ and $cluster(0,c')$ driving a common destination $cluster(0,c'')$. If we pack both sources in the same cluster we obtain on the one hand $\ell_{up} - \ell'_{up} = 0$ (ℓ_{up} is the lowest used level to reach the destination). On the other hand we get
$pos(cluster(0,c)) - pos(cluster(0,c')) \neq 0$. In this case referring to Lemma 1, the conflict condition is not verified and no resource conflict occurs.
To include this objective in the clustering technique, we propose to construct a Cells Constraints Graph (CCG). The CCG denoted as $G_n = (V, E_n)$ consists of a set of vertices and weighted edges derived from the netlist. An edge is established between two vertices when they drive the same destination cluster, which are called adjacent. Each edge contains a weight equal to the number of common destinations between two adjacent vertices. Using only this graph

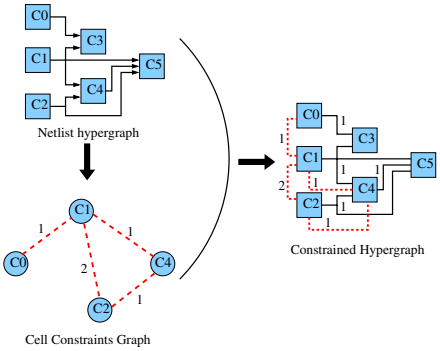

Figure 9. CCH: Cell Constraints Hypergraph

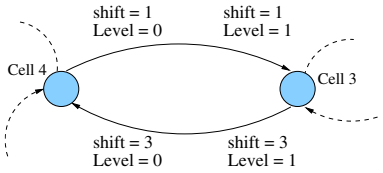

Figure 10. ACCG: Advanced Cell Constraints Graph

in the partitioning weakens the obtained clusters netlist results in term of external communication. To take this in consideration, we propose, as presented in figure 9, to generate from the initial netlist hypergraph and the CCG a new Constrained Cells Hypergraph CCH. In this hypergraph, vertices are cells (as in the netlist) and it contains all hyperedges of the netlist and all edges of the CCG. This constrained hypergraph is partitioned by hMetis and objectives priority will be defined by hyperedges weights.
We propose to use a top-down patitioning approach (global connectivity informations). We first construct clusters of the top level and then each cluster is partitioned into subclusters. This is done until the bottom of the hierarchy is reached. To run partitioning we used hMetis [5] since it generates a good solution in a short time due to its multiphase refinement approach.

3.2.4 Detailed placement

If during detailed placement we take lemma 1 in account, significant gain can be obtained in term of routability and congestion reduction. For this purpose we introduce the Advanced Cells Constraints Graph (ACCG) which is associated to a given a multilevel clustered netlist (previous section) and a placement problem.

Advanced Cell Constraints Graph:
We can say that an ACCG is a CCG that contains extra cells partitioning informations required to check conditions of Lemma 1. An ACCG denoted as $G_n = (V, E_n)$ consists of a set of vertices and directed edges derived from the netlist and the way its cells are partitioned between clusters in each level, where each vertex corresponds to a cell of the netlist. A pair of opposite directed edges is established between two vertices when they drive the same destination cluster (located at any level), which are then called adjacent. To be able to verify conditions proposed in lemma 1, we need to add some informations to the constraints graph. Those informations are stored in each directed edge connecting two adjacent vertices as a list of pairs (shift, level), featuring:
- The forbidden shift between adjacent vertices positions. $shift = (\ell_{up} - \ell'_{up}) \mod 4$.
- The level where is located the common destination cluster.
It is worthwhile to use the lowest level feedback link to connect a source to its destination, since it has an important impact on delay reduction. That's why when we construct the ACCG as described in figure 11, ℓ_{up} corresponds to the lowest level where the source has to go up to reach its destination. Reducing the conflict between sources using the lowest level is beneficial for the first routing iteration. In fact, as explained in [8] we use an iterative rip-up routing algorithm based on the congestion negotiation. We assign an adjustable cost to each feedback. A lower level induces lower cost; consequently in the first routing iteration, signals will be routed using the lowest levels. Using the lowest levels to construct the ACCG has two advatages:
- Fewer switches will be crossed to route signals.
- A good initial solution for the iterative router exists: first iteration is run with the least number of resource conflicts.
Figure 10 presents the Advanced Cell Constraints Graph

```
for each leaf cluster cl
  for each level l
    for each receiver rc of cl in level l
      for each leaf driver dr of rc
        //cl and dr both drive rc
(*)     if rc has no common subreceiver of dr and cl
          if No edge between cl and dr
            create edge e between cl and dr
          end if
          level = GetLevel(rc)
          shift = ShiftCompute(cl,dr,rc)
          append pair(shift,level) to e
        end if
      end for
    for
  for
for
```

Figure 11. ACCG construction

constructed from the placed netlist in figure 8. In line (*)

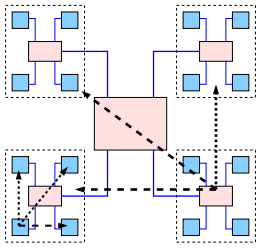

Figure 12. Moves range limiters

of the algorithm, we test whether the common receiver rc has already a sub-cluster (slave) which is also a common receiver of cl and dr. This verification is important to avoid computing many times the same conflict to reach a destination. In fact the conflict can occur when reaching the destination or its owners. In the netlist described in section 3.2, we have a conflict driving $cluster(0,5)$ and its owner $cluster(1,1)$. In the routing phase this conflict will be considered only once. That is why in the generated ACCG we append in the edge only the couple $(1,0)$ corresponding to destination $cluster(0,5)$ and the couple $(1,1)$ corresponding to destination $cluster(1,2)$, but we do not append the couple $(1,1)$ corresponding to destination $cluster(1,1)$. In addition referring to lemma 1, If there is no conflict to reach $cluster(l,c)$, there will be no conflict to reach any one of its owners.

Simulated Annealing technique:

A detailed placement consists in assigning a position for each cell and each cluster of cells inside its owner. The objective is to reduce the number of resource conflicts. To compute this number we take each vertice in the ACCG and we check whether conditions of lemma 1 are verified, if such the global cost function is incremented by 1. Computing this cost for a specific detailed placement is given by the pseudo-code of figure 13. The cost is updated

```
cost = 0
for each vertice v
  mark vertice v visited
  for each adjacent vertice adj of v
    if adj was not visited
      for each couple (level,shift)
(*)     if conflict(v,adj,shift,level)
          cost++
        end if
      end for
    end if
  end for
end for
```

Figure 13. Incremental cost computing

incrementally in the sequel.

To check whether there is a resource conflict (*), we must check conditions of lemma 1. To do this we need informations about the first source position, the second source position (adjacent), the forbidden shift and the destination level. All those informations are provided by the ACCG.

To find the best detailed placement combination we propose to use an adaptive simulated annealing algorithm [7] [1]. In this algorithm the operating parameters are controled using statistical techniques.

Moves are randomly applied to the configuration and consist in assigning new positions. First we choose randomly an element to be moved, it can be a basic cell or a cluster of cells (located at any level). Second we choose randomly the new position inside the direct cluster owner, if it is occupied we swap both elements positions. The cost function is updated incrementally by evaluating the incremental cost of the moved vertices and their adjacents. Moving a cells cluster is important referring to lemma 1 and can lead to cost reduction. In this case, since the ACCG vertices correspond only to basic elements, we update the cost by visiting all basic elements of the moved cluster and their adjacents. We adopt a hard windowing move restriction approach. As presented in figure 12, a cell or a cluster can only move inside its direct owner. This restriction is important to keep constant the partitioning result obtained by the tool described in section 5. In addition, by respecting this restriction, we do not have to update the ACCG since the common receivers and the levels to use to reach them always remain the same. This yields important run time reduction for the cost updating phase.

3.2.5 Logic replication

The idea behind logic replication is that by making copies of one or more logic cells, one can maintain the logical behavior of a netlist while, hopefully, enabling additional optimization. Consider a logic cell a and suppose that we have created a duplicate cell a'. The cell a' takes precisely the same inputs as a and produces exactly the same boolean function of those inputs as its output. In this situation, the pins in the circuit that need to receive this signal may now obtain it from either the output of a or a'. This adds freedom and enhances routability since it enables to reach destination cells using additional routing resources. In addition if we place the duplicate logic block a' inside the super cluster (owner) containing the original logic block a, we will not add routing congestion to connect a' inputs. This means that logic replication must be done after original logic blocks partitioning. Thus we have to estimate which are logic blocks that need to be duplicated before

partitioning to reserve vacant positions and depopulate the containing clusters. To consider this we use the CCG graph to attribute weights to logic blocks. Logic block weight is equal to the number of its adjacent vertices in the CCG. Vertices weights are added to the CCH hypergraph and the partitioner will distribute vertices and controle clusters population based on these informations. Once logic blocks are partitioned between clusters, we run the detailed placement. After the last placement iteration we define logic blocks arrangement inside clusters and we evaluate the number of conflicts that will occur in the first routing iteration. Using the ACCG we can easily identify logic blocks (drivers) leading to these conflicts and duplicate them inside their clusters owners. When the routing iterations are progressing, the router can use the duplicate logic blocks to reach some destinations.

3.3 MFPGA routing

Once netlist logic blocks are placed, the router tries to connect signals using MFPGA interconnect resources. As explained in [8], the router is an adaptation of pathfinder [9]. Since the placement objective is to minimize resources conflict number in the first routing iteration, the router will start with a good initial solution (the first iteration consists in conecting a source to a destination using the lowest level). In the next iterations, the router will try to resolve conflicts using higher levels (congestion negotiation)

3.4 Pins reordering

As we do not have a full crossbar inside clusters (MFPGA topology), inputs and outputs cannot be considered as logically equivalent. As presented in figure 6, each sub-netlist is placed and routed seprately. In the detailed placement phase, sub-netlist inputs/outputs are assigned to specific cluster inputs and outputs pins (MFPGA input pads placement). This new ordering must be back annotated in the clusters interfaces of the inter-clusters netlist. The pins ordering constraint is very penalizing in the clusters netlist routing (see next section). To alleviate the effect of this penality we provide groups of equivalent cluster input pins. Input pins (MFPGA Input pads) are equivalent if they drive the same MSBs. For example if we consider the cluster presented in figure 1 we notice that it contains four groups of logically equivalent input pins. Each group is composed of 3 inputs connected to the same MSB. To enhance inter-clusters routability we distribute the equivalent pins over the four cluster sides.

3.5 Mesh placement and routing

To place and route the clusters netlist we use VPR tool [4]. After clusters placement, VPR achieves routing with the lowest channel width. Having some equivalent cluster pins gives more flexibility to the router and has an important impact to reduce channel routing width.

4 Experimental results

To evaluate architectures and tools performances, we placed and routed some of the largest MCNC benchmark circuits.

4.1 MFPGA routability evaluation

First we evaluated the efficiency of MFPGA flow and especially placement phase. It is a real challenge to perform place-and-route on MFPGA especially with high LUTs occupation. We want to check whether the new iterative detailed placement technique (presented in section 3.2.4) can improve routability compared to the constructive technique presented in [8]. In table 1 we present the effect of both placement techniques (iterative and constructive) on circuits routability. We notice that constructive technique is inefficient with circuits with high occupation. It fails with circuits with occupation more than 50%. The iterative technique is more efficient and can deal with circuits having until 80% of occupation. To deal with circuits having high occupation is also important in the case of Mesh of Tree architecture since it allows to pack a higher number of LUTs into each cluster and consequently to reduce external communication and the number of clusters in the mesh level. This has an important impact on area reduction.

4.2 Mesh of Tree vs Mesh

We use the same benchmark circuits to compare switches requirement between Mesh of Tree and clustered Mesh architectures. The clustered mesh architecture uses a uniform routing with single-length segments and a disjoint switch box. Each cluster logic block contains four 4-LUTs. 10 inputs and 4 outputs are distributed over the cluster sides. LUTs pins are connected to cluster pins using a full local crossbar. For connection blocks (C), $F_c = O.5$ and $F_{c_{out}} = 0.25$ are chosen. These switch density choices are made to be consistent with previous work [2]. We use t-vpack to construct clusters and the channel minimizing router VPR 4.3 to route the mesh. VPR chooses the optimal size as well as the optimal channel width to place and route benchmark circuits. Concerning the mesh of Tree architecture, mesh interconnect level is uniform with single-length segments and disjoint switch box. Each cluster contains 256 4-LUTs. The choice of the optimal number of clusters inputs/outputs is not obvious and depends on the complexity of netlist to implement. When we examine the

largest MCNC benchmark we notice that they can be divided into two categories: those with low interconnect utilization (such as s38417, s38584.1 and bigkey) and those with high interconnect utilization (such as spla and pdc). Based on this remark we propose to target two different Mesh of Tree architectures:

- Architecture with clusters having 64 inputs and 64 outputs to implement circuits with low interconnect utilization.

- Architecture with clusters having 128 inputs and 64 outputs to implement circuits with high interconnect utilization.

In both cases input pins are divided into groups of 4 pins. In each group those four input pins are logically equivalent (connected to the same MSBs). To give more flexibility to the external router, the four equivalent pins are distributed over the 4 cluster sides. Since we do not have a full equivalence between input pins we use an external interconnect with full flexibility: $F_c = 1$.

As shown in table 2 we placed and routed the same benchmark circuits on both architectures. All circuits were totaly routed. In fact in the case of the Mesh of Tree architecture, we can control clusters occupation with the mesh partitioner (section 3.1). Concerning the external mesh interconnect, VPR chooses the minimum channel width.
In both cases we have computed the required switches number by each architecture. In the case of the Mesh of Tree we obtain better density and the number of the switches is reduced by 14% compared to the mesh.
As cluster size is limited (< 256), the MFPGA flow run time is very short (about 10 s). We can consider that this is the required time to place and route logic blocks inside clusters, since this can be done separately (in parallel). In addition, since the clusters size of the Mesh of Tree is 64 times bigger than in the case of clustered Mesh, the netlist that must be placed and routed by VPR is smaller in term of instances and nets number. This explains why the run time in the case of the Mesh of Tree was two times reduced.

4.3 Partitioning objectives comparison

As explained in section 3.1, the mesh partitioning strategy has an important impact on inter-clusters routing phase. In table 3 we present a comparison between a partitioning minimizing the cut and a partitioning which considers both the cut and the Max Subdomain Degree (MSD). In the last case we notice that the MSD is reduced by 9% and the CUT is increased by 1.5%. It means that the external communication is slightly increased and the congestion is better distributed. Consequently we obtain a 4% reduction in the mesh routing channel width.

5 Conclusion

In this work we have shown that, for MFPGA, by focusing most of the effort on the placement phase, the predictability of the hierarchical interconnect allows us to obtain good routability results despite depopulated routing resources. In fact we succeeded to route netlists up to 80% LUTs occupation (and less than 2K LUTs). This has led us to propose an architecture that unifies the merits of MFPGAs and Mesh architectures. This architecture is called Mesh of Tree and ensures routability of all MCNCs benchmarks (thanks to the mesh channel width flexibility) while being 14% smaller in term of switches number.
Notice that solving independently intra-clusters (MFPGA level) and inter-clusters (Mesh level) is penalizing. Indeed, performances can be significantly improved by a better interaction between these two designs flows (at the pins assignement level). Therefore, we can consider our results as a lower bound of the the quality of the proposed architecture. The Mesh of tree can be very promising architecture especially when we target very large netlists implementation ($> 20K$ LUTs).

References

[1] Aarts, Debont, and Habers. Statistical cooling: A general approach to combinatorial optimisation problems. *Philips Journal*, pages 193–226, 1985.

[2] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pages 3–12, 2000.

[3] V. Betz, A. Marquardt, and J. Rose. Architecture and CAD for Deep-Submicron FPGAs. *Kluwer Academic Publishers*, January 1999.

[4] V. Betz and J. Rose. VPR: A New Packing Placement and Routing Tool for FPGA research. *International Workshop on FPGA*, pages 213–22, 1997.

[5] G.Karypis and V.Kumar. Multilevel k-way hypergraph partitioning. *Design automation conference*, 1999.

[6] H.Mrabet, Z.Marrakchi, P.Souillot, and H.Mehrez. Performances improvement of FPGA using novel multilevel hierarchical interconnection structure. *ICCAD, San Jose*, 2006.

[7] Kirkpatrick, Gelatt, and Hecchi. Optimisation by simulated annealing. *Science 220*, pages 671–680, 1983.

[8] Z. Marrakchi, H. Mrabet, and H. Mehrez. A new Multilevel Hierarchical MFPGA and its suitable configuration tools. *Proc. ISVLSI, Karlsruhe, Germany*, March 2006.

[9] L. McMurchie and C. Ebeling. Pathfinder: A Negotiation-Based Performance-Driven Router for FPGAs. *Proc.FPGA'95*, 1995.

[10] N.Selvakkumaran and G.Karypis. Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimisation. *IEEE transactions on computer aided design*, 2005.

				Constructive		iterative	
circuits	LUTs	arch	occup %	R%	R-Time (s)	R%	R-Time (s)
i9	471	4x4x4x4x4	46	100	2	100	18
alu4	584	4x4x4x4x4	57	100	2	100	24
C5315	725	4x4x4x4x4	69	90	3	100	42
tseng	1047	4x4x4x4x4x2	51	100	3	100	60
ex5p	1064	4x4x4x4x4x2	51	98	9	100	180
apex4	1262	4x4x4x4x4x2	61	95	12	100	420
dsip	1370	4x4x4x4x4x2	66	93	5	100	120
misex3	1379	4x4x4x4x4x2	68	91	5	100	150
diffeq	1497	4x4x4x4x4x2	73	90	5	100	210
bigkey	1707	4x4x4x4x4x2	80	88	11	100	240
apex2	1878	4x4x4x4x4x2	90	88	10	94	630
s298	1931	4x4x4x4x4x2	94	86	23	96	600
frisc	3556	4x4x4x4x4x4	86	82	25	93	900
spla	3690	4x4x4x4x4x4	90	82	20	93	1020

Table 1. MFPGA detailed placement evaluation

		Clustered Mesh Cluster size 4			Mesh of Tree Cluster size 256		
MCNC Benchmark	LUTs Number	arch Clusters	Switches $\times 10^3$	R-Time (s)	arch clusters	Switches $\times 10^3$	R-Time (s)
s38417	6406	41 x 41	1269	218	8 x 4	992	36
s38584	6447	41 x 41	1236	273	8 x 4	1090	74
bigkey	1707	21 x 21	358	27	3 x 3	291	16
clma	8383	46 x 46	2123	625	9 x 5	1841	273
pdc	4575	34 x 34	1251	326	7 x 5	1097	311
ex1010	4589	34 x 34	1007	215	5 x 5	990	106
spla	3690	31 x 31	912	172	5 x 4	925	80
frisc	3556	34 x 34	1108	150	5 x 4	971	115
apex2	1878	22 x 22	404	54	3 x 4	415	25

Table 2. Clustered Mesh vs Mesh of Tree

			Partitioning CUT obj			Partitioning CUT & MSD obj		
MCNC Benchmark	LUTs Number	Clusters number	MSD	CUT	Channel width	MSD	CUT	Channel width
s38417	6406	32	99	664	52	99	664	52
s38584	6447	32	107	840	61	94	960	59
bigkey	1707	9	86	425	60	76	433	54
clma	8383	45	150	1317	98	122	1352	90
pdc	4575	35	166	1029	135	158	1099	135
ex1010	4589	25	135	1043	77	130	1085	76
spla	3690	20	171	684	105	146	709	100
frisc	3556	18	167	488	111	161	520	109
apex2	1878	10	142	353	72	132	365	67

Table 3. Partitioning objectives comparison