Performances comparison between Multilevel hierarchical and Mesh FPGA interconnects

Zied Marrakchi, Hayder Mrabet and Habib Mehrez
Dept ASIM-LIP6
Université Paris 6, Pierre et Marie Curie
4, Place Jussieu, 75252 Paris, France
{zied.marrakchi, hayder.mrabet, habib.mehrez}@lip6.fr

ABSTRACT

In this paper we evaluate a new multilevel hierarchical FPGA (MFPGA). The specific architecture includes two unidirectional programmable networks: A downward network based on the Butterfly-Fat-Tree topology, and a special upward network. New tools are developed to place and route several benchmark circuits on this architecture. Comparison with the traditional symmetric, manhattan mesh architecture shows that MFPGA can implement circuits with smaller area and better speed.

1. ARCHITECTURE OVERVIEW

A standard hierarchical FPGA is denoted k-HFPGA in which a cluster has k subclusters. The structure can be represented by a tree. Figure 1 is an example of a 4-HFPGA where a cluster contains 4 subclusters. A vertex in this tree is used to represent a logic or a switch block. An edge between two vertices is used to represent a routing channel which consists of a set of tracks. The logic blocks are at the bottom of the tree while the switch boxes are those vertices above the logic blocks. We propose a modified multilevel hierarchical architecture denoted MFPGA which can be more interesant in terms of area and performances. Our architecture has the following particularities:

- The lowest level of the hierarchy contains the Logic blocks and the IO pads. Each logic element contains one 4 inputs Look-Up Table (4-LUT) followed by a bypass Flip-Flop.
- The routing architecture contains only unidirectional wires and the switch boxes are depopulated.
- In each level the ratio between parent tracks and child tracks is equal to k (k is the number of slaves in the cluster).

As we use unidirectional switches, we can distinguish two connecting networks as shown in figure 1.

- A downward connecting network whose topology is equivalent to the butterfly fat tree. In this tree the edges come from the upper levels and reach the inputs of the logic blocks. The topology of this tree is equivalent to the one used in SPIN network [1] [2].
- An upward connecting network whose edges come from the leaves (outputs of logic blocks and input pads) to the switch boxes of each level.

1.1 The downward connecting network

Let us consider the case of a 2 levels tree with an arity equal to 4. In each level a cluster contains 4 slaves and a switch box. To depopulate the switch box, we divide it into 4 Mini Switch Boxes (MSB). In level 0 each MSB is in charge of connecting the upper level tracks and one input of each logic block as depicted in figure 2. Thus each MSB has 4 outputs which are equal to the number of logic blocks (slaves). The level 1 is constructed in the same manner, we connect the switch box of each cluster of level 1 to 4 clusters belonging to level 0. As each cluster in level 0 has 16 inputs, we divide the switch boxes into 16 MSB and connect each one to one input of a cluster slave. Figure 3 shows the distribution of the interconnect in level 1. The previous described butterfly fat tree has the following properties:

- From a track located in the top of a switch box we can

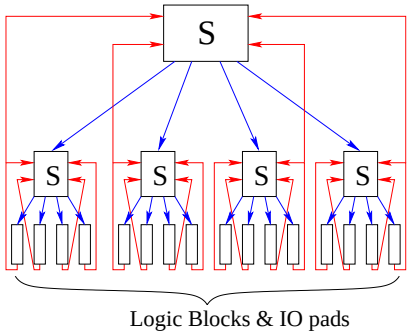

Figure 1: Connection networks

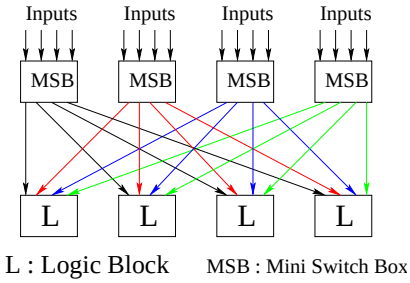

Figure 2: Top-down connecting tree in level 0

reach any slave but in only one pin.

- From a track of a switch box we have only one path to reach a particular slave. Due to the regularity of the architecture, this path is easily predicted.

- In each level the interconnect resources are balanced between clusters.

1.2 The upward connecting network

We propose to connect the output signals to specific switch boxes of upper levels. Thus for each logic block output (and input pad), we define a list of feedbacks, each one enables the output to reach a switch box in a particular level. The way how we distribute the feedbacks on each level has an important impact on the number of different paths to reach a destination logic block from a source. As shown in figure 4, two feedbacks can connect a source to a destination using two different paths.

2. PLACEMENT

2.1 Multilevel clustering

DeHon [4] showed that for hierarchical FPGAs, 100% logic utilisation is not necessary benefical for overall device area minimisation. His results indicate that a careful partitioning of designs and depopulation of logic clusters can result in better FPGA resources utilisation. This remark was confirmed by results obtained by Singh [3] in the case of clustered mesh FPGA. In their work authors presented the routability-driven bottom-up clustering technique. The aim of their technique is to alleviate routing congestion by absorbing as many nets into clusters as possible, and depopulating clusters according to Rent's rule in order to achieve

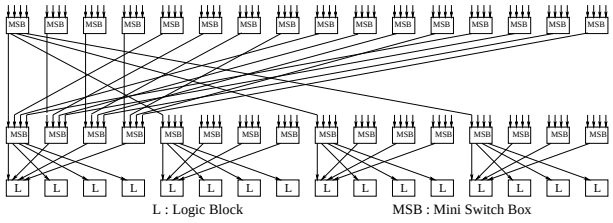

Figure 3: Top-down connecting tree in level 1

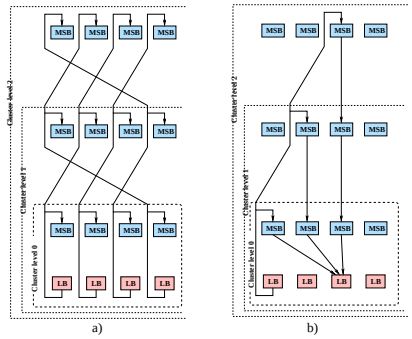

Figure 4: The upward connecting network and different routing paths

spatial uniformity in the clustered netlist. In our work we have extended this technique to the multilevel hierachical FPGA clustering.
The clustering algorithm begins by choosing a logic block as a seed (the block with the highest separation) and assigning it to the first available slot in a cluster. We use the same objective function proposed in [3]. First we identify low fan-out nets and then we absorb them into a cluster. In order to garantee spatial uniformity of the clustered netlist, we limit the number of available pins. Since in each level the interconnect is balanced between clusters, an attempt is made to spread the logic evenly across all clusters in the level while limiting the number of pins available. Once clustering has been done, the original netlist is reduced to a new netlist with each node corresponding to a cluster. We propose to apply the same technique to construct super-clusters of clusters (see figure 5). We notice that pins limit strategy is inefficient when it is applied in high levels. This is due essentially to the bottom-up and the greedy aspect of our clustering technique. To deal with such problem, we propose to create clusters in high levels without pins-limit enforcing. As presented in figure 5, once the multilevel clustering is achieved, we run a top-down refinement.

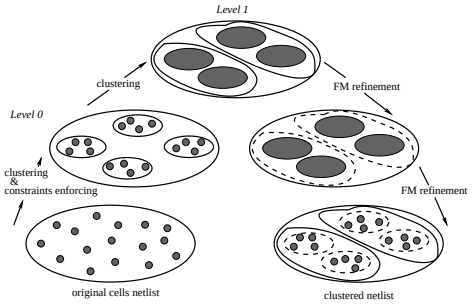

Figure 5: Multilevel clustering & refinement

2.2 Multilevel refinement

After the clustering phase, we obtain k clusters in each level. We can consider that the k clusters in a level present an initial solution to a k-way partitioning problem. During the refinement phase, cells will be moved between clusters (parts) to optimise an objective function without violating the constraints imposed by the cluster size. In a level, cells are not allowed to move between all clusters, because this can decrease the quality of the solution obtained in the upper level. Thus in every level, adjacent clusters (having the same supercluster) will be isolated and form a subgraph. In figure 5 those subgraphs are presented by the continued lines and partition by the dashed ones. A cell is allowed only to move across dashed lines. The objective function is local to each subgraph and corresponds to the maximum number of pins of all clusters (parts) belonging to the same subgraph (Maximum Subdomain degree). An FM algorithm [5] will be applied on a subgraph to optimise the local objective function. The complexity of our k-way refinement is reduced since we do not apply it for all the graph but successively for each subgraph (in each subgraph there are small values of parts: Arity of the architecture).

2.3 Detailed placement

Now that we have obtained clusters with minimum number of pins and containing highly connected cells in each level, we proceed to the detailed placement. As in the proposed architecture we do not have full cross bar connection boxes, we can not place cells randomly inside the clusters. In fact the way cells are placed has an important impact on the routability. If during the detailed placement special properties of the netlist and the interconnect can be exploited, significant gains can be obtained in terms of routability and congestion reduction. The effect of the detailed placement on routability can be explained by the example shown in figure 6. In this example we have placed two logic blocks and

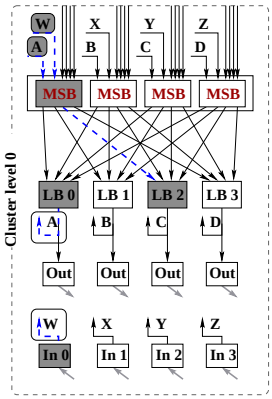

Figure 6: Impact of detailed placement on routability

an Input Pad in the same cluster. The logic block in position 0 LB0 and the Input pad in position 0 In0 drive the logic block placed in position 2 LB2. With the present placement we can not route both signals using only the switch box in level 0. In fact both signals reach the target block in the same pin (dashed lines). This problem can be resolved by simply changing the position of one of the driver blocks. This problem can also occur between two logic blocks located in two different clusters and trying to drive the same logic block. Our detailed placement is applied seperately in each level. In level l we notice that congestion is generated by cells that drive the same destination cell. Thus we introduce the notation of Cells Constraints Graph (CCG). Given a clustered netlist and a placement problem, a CCG denoted as $Gn = (V, E_n)$ consists of a set of vertices and edges, can be constructed from a netlist. In a CCG, a vertex is used to represent a cell of the netlist and an edge is constructed between two vertices which drive the same destination cell. Those cells are called adjacent cells.
Thus the placement of vertices will introduce constraints on the placement of the adjacent vertices. For each vertice we reserve a list of allowed positions (slots) inside the cluster. Initially each vertice has k possibilities (arity of the cluster). Our algorithm is as the following:

```
Loop over not placed vertices
  Eliminate occupied positions
  Loop over placed adjacent vertices
    Find positions to avoid
  End
END
```

In our technique the order of placing vertices is very important and has a great impact on the efficienty of the method. We give priority to vertices that have the following properties:

- Vertices located far from their destinations. We know that these vertices have less paths to reach their destinations.

- Vertices with high number of adjacent vertices: more constraints.

3. ROUTING

The routing problem can be stated as assigning signals to routing resources in order to successfully route all signals. This goal is difficult to achieve in our architecture because of the lack of routing resources (depopulated switch boxes). In fact the number of paths to reach a destination from a source is significantly reduced and those paths depend on the location of cells and the number of levels in the architecture. Thus signals will compete for the same resources and the challenge is to find a way to allocate resources so that all signals can be routed. Despite this disadvantage we have a great advantage in our architecture since our unique path connecting an MSB to a logic block is predictable.
To route our architecture we adopted a particular iterative rip-up algorithm based on the congestion negotiation called PathFinder [6]. PathFinder was applied to the mesh architectures and we have adapted it to our architecture. Since we have a unique downward path to reach a destination from an MSB, we have eliminated the breath-first search in the detailed routing part.
Since the choice of the feedback imposes the path to use, our negotiation must be done on the choice of the feedback that leads to a path with less congestion. According to this remark, we assign to each feedback an adjustable cost. The global router dynamically adjusts the congestion penalty for each feedback. During each iteration individual routing resources may be used by more than one signal. The penalty to use shared resources is gradually increased so that signals will negotiate effectively for resources. The implemented algorithm is described by the following:

```
While shared resources exist
  /*global router*/
  Loop over all signals i
    Loop until all sinks tij are found
      Rip up branch Bij
      Find feedback fij with lowest cost
      Bij <- fij
      /*detailed router*/
      Loop until new tij is found
        Find next_wire
        Add next_wire to Bij
      End
    End
  End
  /*backtrace*/
  Loop over nodes in Bij
  /*path from tij to si*/
    Update cost of fij
  END
END
```

Benchmark		Mesh					MFPGA					
Name	LUTs	$\sqrt{N}$	W	IO ratio	Switches number	Area (λ^2) x10^3	Arch	Occup-ation%	R%	R% +ref %	Switches number	Area (λ^2) x10^3
cm42a	10	4	3	1	948	4344	4x4	63	100	100	512	2032
pcle	29	6	5	2	3700	15316	4x2x2x4	46	100	100	3584	11968
decod	32	6	4	1	2768	11822	4x4x4	50	95	100	3584	11648
cc	33	6	5	2	3700	15316	4x4x4	52	92	100	3584	11648
count	37	7	5	2	4950	20577	4x4x2x2	58	98	100	4096	12608
my_adder	49	7	4	2	3960	16680	4x4x4	77	100	100	3584	11648
b9	61	8	5	4	7020	28656	4x4x4	96	90	98	3584	11648
i4	110	11	7	5	18298	71289	4x4x4x4	42	87	100	20480	46080
c2670	363	20	8	5	63968	249172	4x4x4x4x4	35	92	100	106496	299008
i9	471	22	8	2	72480	286356	4x4x4x4x4	46	85	100	106496	299008
alu4	584	25	8	1	91568	363547	4x4x4x4x4	57	95	100	106496	299008
tseng	1085	33	9	1	178758	709785	4x4x4x4x4x2	53	88	100	253952	679936

Table 1: Area Comparison between MFPGA and Mesh architectures

4. TIMING ANALYSIS

Timing analysis is used for two basic purposes:

- To detrmine the speed of circuits which have been completely placed and routed, and
- To estimate the slack of each source-sink connection during routing (or other parts of the CAD flow) in order to decide which connections must be made via fast paths to avoid slowing down the circuit.

In our work we are interested to study the performances of our proposed architecture MFPGA in term of speed. Thus once an application was completely placed and routed we estimte the minimum feasible clock period to run it. Ideally, one would use a circuit simulator such as SPICE to obtain highly accurate delay estimates, but the CPU time required to run SPICE on the thousands of nets in a typical circuit is prohibitive. Instead since we have a predictive upward and downward connecting networks, we propose to divide a path into several sub-paths. Each sub-path connects a source to a sink and consists on going from a source up to a particular level and then down to the sink. In each architecture the number of sub-paths depends on the number of levels. The first step consists on estimating delays on each sub-path. Second we compute the delay of each path composed of several sub-paths.

4.1 Sub-paths delays evaluation

As explained previously a sub-path consists on connecting a source to a sink crossing several switch boxes of the interconnect. The number of different sub-paths existing in the architecture is limited and depends on the number of levels. So given an architecture with n levels, we can isolate the n different sub-paths. In figure 7 we show the 3 isolated sub-paths of an architecture containing 3 levels. We use circuit simulator SPICE to obtain highly accurate delay estimation in each sub-path. Each architecture is composed of combinational sub-paths that either start at a logic block (Combinational/Sequential) or an input pad *pi* and either end at a logic block (Combinational/Sequential) or an output pad *po*. To ensure proper circuit function we additionaly have to take register setup-times t_{set} and sequential propagation delays d_{seq} (Sometimes denoted as "Clock-to-Q" delays) into account. The classification of sub-paths and the resulting delays is given as the following:

1. Combinational logic block → Combinational logic block
$d(p) = d(switches)$
2. Combinational logic block → Output-pad
$d(p) = d(switches) + d(po)$
3. Input-pad → Combinational logic block
$d(p) = d(pi) + d(switches)$
4. Sequential logic block → Sequential logic block
$d(p) = d_{seq} + d(switches) + t_{set}$
5. Sequential logic block → Combinational logic block
$d(p) = d_{seq} + d(switches)$
6. Sequential logic block → Output-pad
$d(p) = d_{seq} + d(switches) + d(po)$
7. Input-pad → Sequential logic block
$d(p) = d(pi) + d(switches) + t_{set}$
8. Combinational logic block → Sequential logic block
$d(p) = d(switches) + t_{set}$

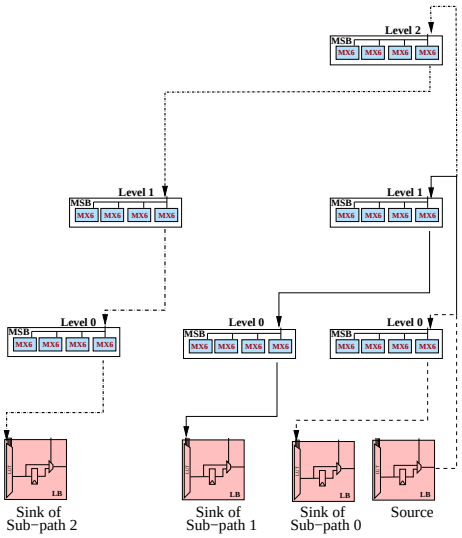

Figure 7: Sub-paths timing caracterisation

Delays on sub-paths depend on the length of wires connecting MSB and logic blocks. The length of those wires is estimated by making the layout of MFPGA architectures with different sizes (different number of levels).

4.2 Critical path extraction

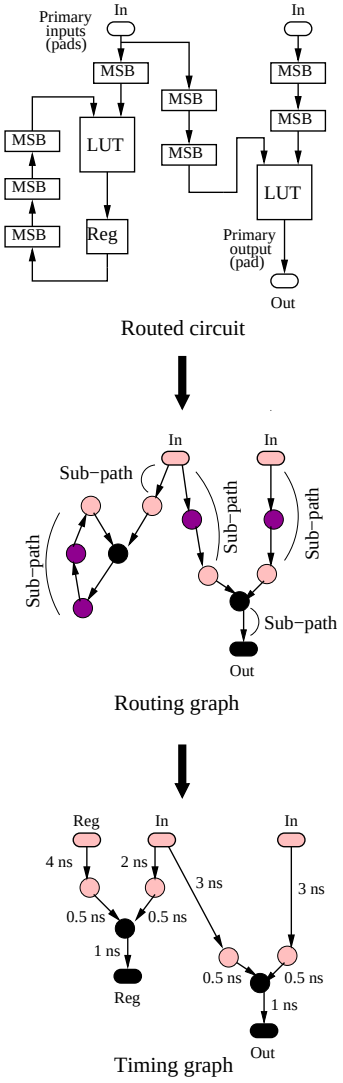

Figure 8: Graph modeling of a simple circuit

Once circuit have been completely placed and routed we obtain a direct graph called "routing graph". This resulting graph describes wires that will be used to connect logic block pins as described in the netlist and using architecture routing resources. Each wire and each logic block pin becomes a node in this "routing graph" and each passing switch (inside the MSB) becomes a directed edge. Edges are also added between the inputs of logic blocks and their outputs. Figure 8 shows a simple circuit implemented via 2-input LUTs and registers, and the corresponding "routing graph". On this graph we can isolate easily different sub-paths. We use a depth-first traversal of the graph. We replace each sub-path by only one edge annotated with the sub-path delay.
Thus we obtain a new direct acyclic graph called "timing graph". In this graph nodes represent the input pins and output pins of basic circuit elements, such as registers and LUTs. Register input pins are not joined to register output pins. Register outputs have no edges incident to them , and register inputs have no edges leaving them (acyclic graph). Similarly, primary inputs (input pads) have no incident edges, and primary outputs (out pads) have no exiting edges. Each edge is annotated with the delay required to pass through circuit element or routing (sub-path delay). Figure 8 shows the obtained "timing graph" of the routed circuit.
One can determine the minimum required clock period with $O(n)$ computation for a "timing graph" with n nodes via a breadth-first traversal. The traversal begins at nodes with no incident edges (primary inputs and register outputs) and labels each with a signal arrival time, $T_{arrival}$, of 0. Each node which has incident edges only from labelled nodes is then labelled with its arrival time according to

$$T_{arrival}(i) = \max_{j \in fanin(i)} \{T_{arrival}(j) + delay(j,i)\}$$

where node i is the node being labelled, and $delay(j,i)$ is the delay value marked on the edge joining node j to node i. This procedure continues until every node in the graph has been labelled. The node with the largest arrival time, which will be always a primary output or a register input, then defines the maximum delay, D_{max}, (= minimum clock period) through the circuit. In figure 8, for example, the arrival time at node *Reg* is 5.5 ns, and this is the largest arrival time, and hence the maximum delay, in the circuit.

5. RESULTS

5.1 Tools validation

To validate and study the performances of our tools, we placed and routed some of the MCNC benchmark circuits. As shown in table 1 results are very promising since we were able to route circuits that occupies until 77% of the logic area. We have tested the effect of the refinement phase which was run after the multilevel clustering. So we have tried to rout resulting placed netlist in both cases:

- Multilevel clustering without FM refinement (column 10 of table 1).
- Multilevel clustering followed by FM refinement (column 11 of table 1).

We notice that in most cases the FM refinement alleviates congestion and leads to better routability results. Nevertheless the router failed to route benchmarks with very high occupation like b9. In this case the router routes a large amount of the nets (until 98%). To improve the performances of the router we propose to:

- Better use routing resources by modifying the distribution of the rising interconnect or increasing the number of feedbacks in each level (An output can have more than one feedback in a level).
- Improve the placement and especially the detailed one (alleviate congestion).

5.2 Area efficiency

To have an idea of the area efficiency, we use the same MCNC benchmark circuits to compare the switch and the area requirements between our MFPGA architecture and traditional Mesh topology.
The Mesh is similar to the vpr422_challenge_arch architecture [7] with uniform routing with single-length segments and a subset switch box. Each logic block contains a single 4-LUT. One input appears in each side, and the output appers on the top and the right side. Both inputs and outputs are fully connected ($F_c = 1$), and IO pads are fully connected too.
We use the channel minimising VPR 4.3 [7] router to route circuit on the mesh architecture, and we vary the IO ratio to achieve the opotimal array size.
VPR chooses the optimal size and the optimal channel width needed to place and route each circuit. For the MFPGA we choose a structure large enough to support the benchmark circuit. MFPGA structure can be varied by changing the number of levels or the arity of each level.
We compare areas of both architectures using successively a simple cost model based on routing switches count and a more refined model that estimates the effective circuit area. The mesh area is the sum of its basic cells areas like SRAMs, Tri-states and multiplexers. The same evaluation is made for the MFPGA which is composed basicly of SRAMs and Multiplexers. We use the same cells library for both architectures. Table 1 summarises the basic results for the Mesh and for the MFPGA.
Given a benchmark with a fixed size, we implement it on a specified MFPGA architecture, we take area results as summarised in the right part of table 1. Column 9 shows the occupation average of each circuit in the target MFPGA. We notice that we have a low occupation average with the majority of the benchmark circuits. This is due essentially to the depopulation of the interconnect. As said previously we under-utilise the logic resources in this type of structure. In addition, the size of the smallest MFPGA is penalised by the coarse granularity of the architecture. In spite of these constraints, we achieve a gain in area compared to the mesh architecture. We notice in table 1 that for a low occupation average of the logic resources (LBs), the total MFPGA area is smaller than its equivalent mesh (which has a better occupation ratio) since it requires less interconnect resources.

5.3 Speed performances

It is clear from the previous comparison that the new architecture is more efficient in term of area and this can have a good effect on circuit speed: smaller is the area, shorter are the connecting wires. In addition thank's to the hierarchy aspect of our MFPGA architecture, logic blocks will be partitioned between clusters to reduce external communication. This has an important impact on circuit speed improvement since communication will be faster. We have compared the speed of the MFPGA architecture to the Mesh . We have implemented the same circuits and we have used our timing analysing tool for the MFPGA (section 4) and the one proposed in VPR for the mesh. Timing results are presented in table 2 for both architectures. We notice that MFPGA largely outperforms Mesh architecture in term of speed. As explained previously this is due essentially to the multilevel hierarchy aspect of the MFPGA. Modern Mesh FPGAs (cluster based FPGA) have two levels of hierarchy and connecting segments with different length (> 1). This kind of architecture has better speed performances than the common ones (used here as reference). So in the next step we will make comparision with this kind of architectures specially with larger benchmark circuits.

Circuit	Mesh T(ns)	MFPGA T(ns)
cm42a	11.65	2.12
pcle	14.90	4.34
decod	13.50	3.56
cc	16.50	4.25
count	26.52	7.78
my_adder	55.12	27.45
b9	20.47	6.98
i4	22.14	6.03
c2670	48.84	16.67
i9	44.28	13.90

Table 2: Speed Comparison ($0.13\mu m$ CMOS, $1.2V$)

6. CONCLUSION

The downward network is a predictable interconnect which has a very interesting impact on accelarating the routing phase. The routing key of the proposed architecture is the upward network. Enhancing the routability needs to populate the upward network to increase paths number between sources and destinations. This can lead to area increasing, but can be compensated by applying the Rent's rule to reduce the cluster inputs/outputs
The preliminary results show that good balancing of the LUT and the interconnect utilisation reduces area compared with traditional Mesh architectures. Thank's to the hierarchy aspect of the MFPGA we have a very good performances in term of speed compared with the common Mesh architecture.

7. REFERENCES

[1] A. G. A. Adrijean. Micro-network for soc: Implementation of a 32-port spin network. *Proc. DATE'03*, pages 1128–1129, march 2003.

[2] A. G. A. Adrijean. Spin: a scalable, packet switched, on-chip micro-network. *Proc. DATE'03*, pages 70–73, march 2003.

[3] M. M.-S. A. Singh. Efficient circuit clustering for area and power reduction in fpgas. *Proc. FPGA'02*, February 2002.

[4] A. DeHon. balancing interconnect and computation in a reconfigurable array (or why you don't really want 100% lut utilisation). *Proc. FPGA'99*, 1999.

[5] C. M. Fiduccia and R. M. Mattheyeses. A linear-time heuristic for improving network partitions. *Proc. DAC*, pages 175–181, 1982.

[6] C. E. L. McMurchie. Pathfinder: A negotiation-based performance-driven router for fpgas. *Proc.FPGA'95*, 1995.

[7] J. R. V.Betz, A. Marquardt. Architecture and cad for deep-submicron fpgas. *Kluer Academic Publishers*, January 1999.