A new Multilevel Hierarchical MFPGA and its suitable configuration tools

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 MFPGA. The specific architecture includes two unidirectional programmable networks: A downward network based on the Butterfly-Fat-Tree topology, and a special rising 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 fewer switches and a smaller area.

Keywords

FPGA, Hierarchical, Multilevel, Clustering, Routing

1. INTRODUCTION

The architecture of an FPGA is similar to a gate array and can be programmed to specify the function of the logic blocks and their interconnections. However these arrays suffer from lower density and speed compared with application specific integrated circuits ASICs. The most common FPGA architecture is the mesh which has a symmetrical grid of logic blocks and routing channels on all four sides of the logic blocks. Less work has been done for hierarchical FPGAs (HFPGAs), which contain a hierarchy of logic blocks and routing resources. HFPGA whose channels are fully populated with switches offers lower density than the other ones but have the advantage that routing has more predictable paths as well as lower delays. In previous papers [4] [9] different FPGAs with hierarchical depopulated interconnection structures were proposed to improve these shortcomings.

In our work we are also interested by this kind of architectures. We have explored the architecture of hierarchical HFPGA with regard to the particular issue of how the switch patterns of HFPGA can be partly depopulated. In fact the structure that we propose has a better ratio between logic and routing resources occupations than conventional mesh FPGAs. Hence the density of HFPGAs can be higher than conventional ones. Moreover, because fewer switches are needed for a connection path, the timing-skew problem is alleviated.

In this paper we give a description of our proposed architecture and the suitable techniques to program it.

2. 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 four subclusters. A vertex in this tree is used to represent a logic or a switch box. An edge between two vertices is used to represent a routing channel which consists of many 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.

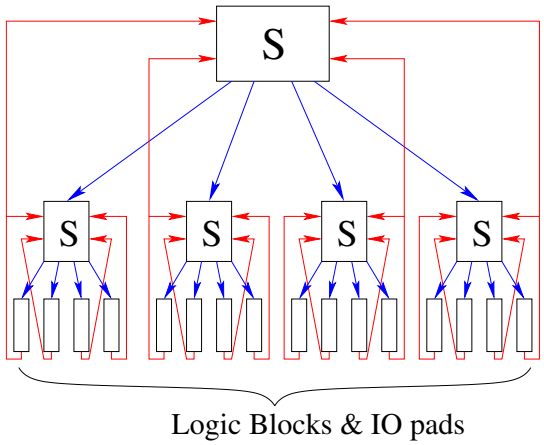

Figure 1: Connection networks

- 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.

2.1 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 boxes, we divide it into four 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

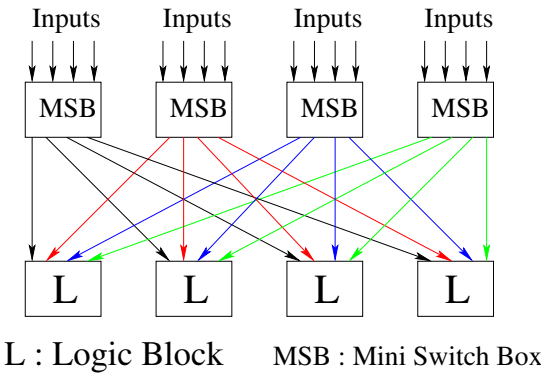

Figure 2: Top-down connecting tree in level 0

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

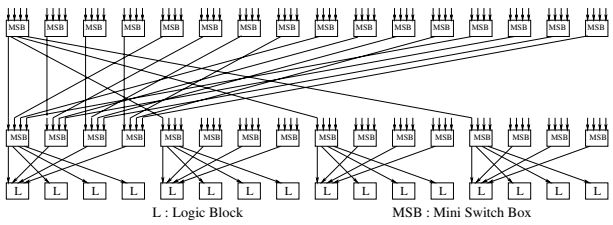

Figure 3: Top-down connecting tree in level 1

described butterfly fat tree has the following properties:

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

2.2 The upward connecting network

The next question is how can the outputs of logic blocks and the input pads reach the inputs of other logic blocks. 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. An output of a source cell can reach its destination only if it has a feedback in the lowest common level or in a higher one. 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. In fact, if in each level the feedback is connected to an MSB with an index different from the one in the previous level, we can obtain two different paths to reach a cell. In our case we distribute

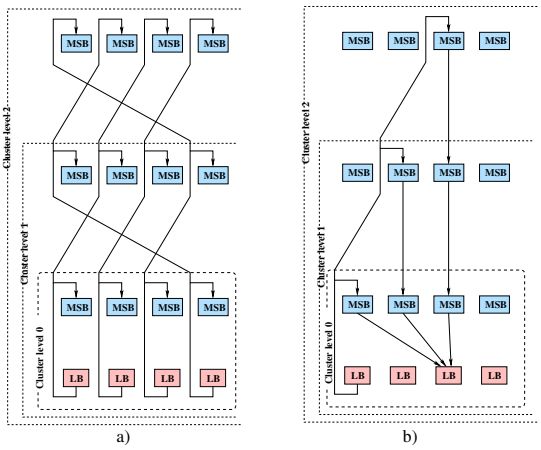

Figure 4: The rising connecting tree and different routing paths

the feedbacks as depicted in figure 4-a. We achieve a multi path structure which enhances the routability and gives us more different possibilities to reach a destination. As shown in figure 4-b, using two different feedbacks (located in two different levels), an output pin can reach a destination cell from two different paths but in two different pins. Since all the input pins in a LUT are logically equivalent, the router can complete a given connection using any one of the input pins of a LUT. Changing the ordering of the inputs in a LUT due to the connections performed by the router can be compensated by re-ordering the values of the LUT mask.

2.3 Connection with outside

As explained previously the input pads are located in the bottom of the architecture and are located inside the clusters of the level 0. These pads are connected like the outputs of the logic block in the rising network. The last point in this architecture concerns the location of the output pads. Those pads are also located inside clusters of level 0. To reduce the complexity of the routing architecture, those pads are not connected to the descending tree. Thus they have their own network interconnect. This interconnect is local to the container cluster. As shown in figure 5 each output pad can be driven only by outputs of a logic blocks belonging to the same cluster. Such a layout simplifies the task of the router but adds more constraints to the placer.

3. PLACEMENT

In this section, we present the placement technique used in the case of our hierarchical MFPGA. As explained previously our routing resources are limited and we have a few different ways to connect a source to a destination. Thus the placement of the cells has an important impact on the routability of the netlists. In fact most part of the effort will be devoted to the placement phase. Placement is done in two steps. First we apply a global placement. The aim of this phase is to balance the nets to route between clusters. It consists on a multilevel clustering and a multilevel refinement phases. Second in each level we run a detailed placement to select slots that will be occupied inside clusters. As it will be explained in the following, this second phase is important to alleviate the routing congestion.

3.1 Multilevel clustering

DeHon [6] showed that for hierarchical FPGAs, 100% logic utilisation is not necessary benefical for overall device area minimisation. He presented some initial evidence to support this claim and presented a technique for depopulating gates in a hierarchical array. 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 this work authors present 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 spatial uniformity in the clustered netlist. In the following we explain how we have extended this technique to multilevel hierachical MFPGA 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 objectif function proposed in [3]. First we identify low fanout 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. An unclustered block can be absorbed into a cluster only if cluster size and a pins constraint (balancing constraint) are satisfied. The pins constraint can lead potentially to spatial uniformity and less than 100% cluster utilisation. As it is shown in [3], the described clustering technique can reduce the number of external nets and eliminate regions (clusters) with high congestion. 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 6). Thus we use the same gain function to compute the attraction of each block to a cluster. We notice that pins constraints enforcing 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. In fact in most cases when we impose pins constraints the tool needs more than the allowed clusters to achieve the clustering. To deal with such problem, we propose to create clusters in high levels without pins constraints enforcing. As presented in figure 6, once the multilevel clustering is achieved, we run a top-down refinement. The aim of this refinement is to move some blocks between clusters to reduce the number of pins per cluster in each level (pins balancing).

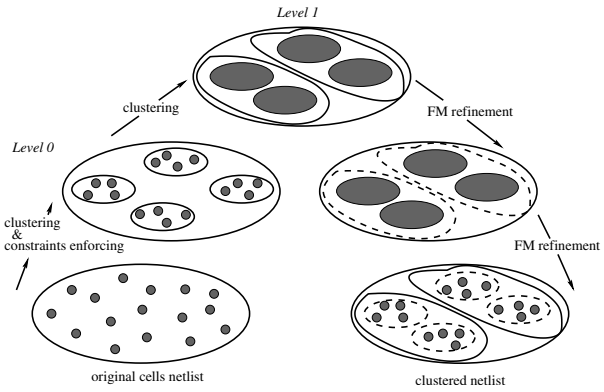

Figure 6: Multilevel clustering & refinement

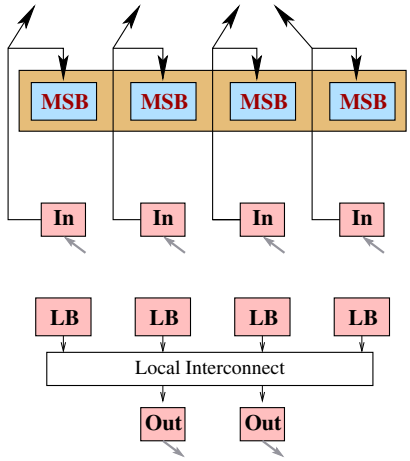

Figure 5: IO Pads connections

3.2 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 higher level. To prevent such bad effect, cells can only move between neighboring clusters. We call neighboring clusters, all clusters in a level belonging to the same supercluster. Thus in every level, neighboring clusters will be isolated and form a subgraph. In figure 6 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. An FM

algorithm [7] will be applied on a subgraph to optimise the local objective function. As described in [7] [10] this algorithm uses k(k-1) priority queues, one for each type of move. In each step of the algorithm, the moves with the highest gain are found from each of these k(k-1) queues, and the move with the highest gain that preserves or improves the balance, is performed. Note that the gain of a cell may be negative. After the move, all of the k(k-1) priority queues are updated. The balance criterion is used to select the cluster (part) from which a cell with the highest gain is to be moved. In our case, in each move we allow only one cluster to contain a number of cells exceeding the limit imposed by the architecture (the arity of the level). After all cells have been moved, the partition which has the best global gain and which respects the architecture arity is retained as the output result of this pass. 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). The hill-climbing capability of the FM algorithm serves a very important purpose. It allows movement of an entire cluster of vertices across a partition boundary. Note that it is quite possible as the cluster is moved across the partition boundary, the value of the objective function increases, but after the entire cells moves across the partition, then the overall value of the objective function comes down. As shown in figure 6, when we apply refinement, we begin from high levels down to lower ones.

3.3 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 7. In this example we have placed two logic blocks and 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 reaches the target block in the same pin. This problem can be resolved by simply changing the position of one of the driver blocks. This problem can also occur between two logic bloks located in two different clusters and trying to drive the same logic block. Our detailed placement is applied seperately in each level. For example we will present how this technique is applied in level 0. The same method can be extended to the other levels. In level 0 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.

A CCG is used to represent the routing constraints for a placement problem in a particular level. This means that the placement of vertices will depend on the positions of the adjacent vertices that have been already placed. 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
```

To eliminate positions considering the adjacent placed vertices, we define a specific function that takes in account the following parameters:

- Clusters where the common destination, the vertice to place and the placed adjacent vertices are located.
- Type of the vertices to be placed and the adjacent placed ones (logic block/Inpud pad).

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.

Vertices are sorted depending on both previous properties and then placed by the algorithm following the method that we have described. The technique is expanded on all levels. When we apply the technique we begin from high levels down to lower ones.

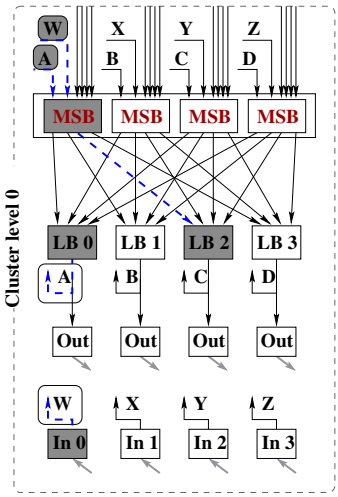

Figure 7: Impact of detailed placement on routability

4. 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 is predictable. Unlike the other architecture (mesh-connected arrays, Triptych ...) we do not need to define a directed graph to describe the routing architecture. This reduces the routing process complexity.

We have studied different routing techniques to find the most suitable one for our approach. For example the obstacle avoidance algorithm could be used: if in one path we find a used resource we can try another path by jumping to another level (using another feedback). This algorithm is easy to implement but it usually yields to many unroutable nets. Some rip-up and retry approaches have been proposed to remedy the deficiencies of this approach [5].

To route our architecture we adopted a particular iterative rip-up algorithm based on the congestion negotiation called PathFinder [8]. PathFinder was applied to the mesh architectures and we have adapted it to our architecture. Since we have only one path downwards to reach a destination, we have eliminated the breath-first search in the detailed routing part. Our detailed router corresponds to a function that determines directly the next wire to reach destination. In any way once we have chosen the corresponding feedback, only one path (only one next wire) can bring us to the destination.

Since the choice of the feedback imposes the path to follow, 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 the first iteration of the global router each feedback has a cost equal to the index of the level where it is located. This encourages the use of lower level to reduce the path length and the number of switches to cross. During this iteration individual routing resources may be used by more than one signal. During subsequent iterations the penalty to use shared resources is gradually increased so that signals will negotiate effectively for resources. In fact costs of feedbacks of a source will change: a feedback belonging to a higher level can get a cost lower than a feedback located in a lower level. The implemented algorithm is described in 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
```

The algorithm is based on two simple and basic functions that are very depended on our MFPGA routing architecture. The first one belongs to the global router and determines the feedback that the source will use to reach the destination. Knowing the source cell index, the sink cell index, this function return the best level to jump to, in other words the feedback with the lowest cost. The second function belongs to the detailed router and determines the next wire to use to reach destination knowing the actual wire index.

5. RESULTS

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 have noticed that in most cases the FM refinement alleviates congestion and leads to full routability. 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).

To have an idea of the area efficienty of our architecture, we have compared switches and area requirements between our MFPGA architecture and the traditional mesh topology. The mesh is similar to the vpr422_challange_arch architecture with uniform routing with single-length segments and a subset switch box. Each Logic Block contains only one 4-LUT. One input appears in each side, and the output appears on the top and the right side. Both the inputs and the outputs are fully populated ($F_c = 1$). The IO pads are fully populated too.

We use the channel minimising VPR 4.3 router to route the mesh, and we vary the IO_ratio to achieve the optimal array size.

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
b1	4	2	3	2	300	1284	4	100	100	100	32	288
cm138a	9	3	4	2	824	3344	4x4	56	100	100	512	2032
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

Table 1: Benchmark statistics

VPR chooses the optimal size as well as the optimal channel width needed to place and route each benchmark. For the MFPGA we choose the structure that is large enough to support the benchmark circuit. MFPGA structures can be varied by changing the number of level, the arity of each level.
In both cases the number of switches consumed by each benchmark corresponds to the total number of switches used by the overall optimal target architecture.
We compare the area of both architectures using both a simple cost model based on routing switches count, and a more refined model that estimates effective circuit area. The mesh area is the sum of its basic cells area like SRAMs, Tri-states and multiplexers. The same thing with the MFPGA composed primarily of SRAMs and Multiplexers. We use the same cells library for both architectures.
Column 9 of table 1 shows the occupation average of each circuit in the target MFPGA. There is a low occupation average in the majority of the benchmarks. This is due to the depopulation of the interconnect. As mentioned previously we under-utilise the logic resources in this type of structure. In addition, the size of the smallest MFPGA that can contain the circuit under investigation is penalised due to the coarse granularity of this architecture. In spite of these constraints we achieve a gain in area efficiency compared to the mesh. Columns untitled "Switches number" and "Area" in table 1 show the difference in number of switches and the total area in the Mesh and the MFPGA structures.
It is clear from this comparison that the new architecture will be more efficient in terms of area if we can increase the Logic utilisation.

6. CONCLUSION

This paper described a new hierarchical multilevel MFPGA architecture and its suitable configuration tools. The preliminary results show that good balancing of the LUT and the interconnect utilisation reduces area compared with traditional Mesh architectures.
The new topology based on two hierarchical unidirectional networks seems to be more robust and can achieve better speed than symmetrical FPGA architectures.
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 between sources and destinations. This can leads to area increasing, but can be compensated by applying the Rent's rule to reduce the cluster inputs/outputs

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. A. Aggarwal and D. M. Lewis. Routing architecture for hierarchical field programmable gate arrays. *Pro. IEEE Custom Integrated Circuits Conference*, 1994.

[5] W. Dees and R. Smith. Performance of interconnection rip-up and reroute strategies. *Proc. DAC*, pages 382–390, June 1981.

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

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

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

[9] Y. T. Lai and P. T. Wang. Hierarchical interconnection structures for field programmable gate array. *IEEE Trans. VLSI*, pages 186–196, 1997.

[10] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. on computers*, 38(1), January 1989.