

Hierarchical FPGA clustering to improve routability

Zied Marrakchi, Hayder Mrabet, Habib Mehrez
LIP6-ASIM Laboratory, Université Paris 6, Pierre et Marie Curie
4 Place Jussieu, 75252 Paris, France
E-mail: zied.marrakchi@lip6.fr

ABSTRACT

In this paper we present a new clustering technique, based on the multilevel partitioning, for hierarchical FPGAs. The purpose of this technique is to reduce area and power by considering routability in early steps of the CAD flow. We will show that this technique can reduce the needed tracks in the routing step by 15% compared with the other packing tools.

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) have gained rapid commercial acceptance thanks to their reconfigurability and low cost. Speed and area efficiency of an FPGA are directly related to the granularity of its logic blocks. If the logic blocks are fine grained, the circuit to be implemented will be distributed over a larger number of logic blocks. This has a negative impact on routability since more blocks need to be interconnected. Recently FPGA vendors have introduced hierarchical FPGAs consisting of logic clusters. Examples of such devices are the Xilinx Virtex and the Apex from Altera. In these architectures several Look Up Tables (LUTs) are clustered into one logic block to provide better performances specially for communication and to exploit signal sharing among LUTs. Our work focuses on the way to adapt an existing multilevel partitioning tool to the FPGA clustering problem. In fact, some constraints imposed by the architecture (number of pins and cluster size) must be respected.

2. PREVIOUS WORK

Prior research on clustered FPGA architectures has focused mainly on area and delay optimisation. Betz et al. [1] proposed a packing/clustering algorithm: Vpack for hierarchical FPGAs [5]. The main idea in their work was to pack a technology-mapped circuit into clusters of a given size and input/output pin constraints. The same authors introduced a tool, T-Vpack [2], using a timing driven packing approach based on the idea of packing blocks on timing-critical paths to exploit fast local interconnects. The clusters generated using T-vpack use an average of 12% fewer tracks than the clusters generated using Vpack for the same array size. A recent work, R-pack [3] presented a routability-driven packing algorithm which first identifies routability factors and prioritizes these factors into an improved clustering function. This approach produces routing track counts comparable to those generated by T-Vpack.

3. ARCHITECTURE OVERVIEW

The FPGA we are targeting is of island-style structure. As it can be seen in figure 1, the circuit is composed of clustered logic blocks (CLBs), switch blocks, connection blocks, and I/O blocks. Each CLB, which implements the user's logic, has inputs and outputs connected by the routing network. It contains N basic logic elements (BLEs) grouped together. Each BLE contains a k -input lookup table (a K-LUT) followed by a bypass flip-flop. The LUT inputs are chosen from among a set of shared cluster inputs. In our case $k = 4$.

The Connection block connects the input and output pins of a CLB to the routing channel, and the Switch block connect the wires of two intersecting channels. The number of tracks between any two neighboring clusters is uniform and is called the channel width. The number of logic clusters that each wire-segment spans before going through a switch box is called the track segment length. We assume that a cluster of size n has $(2n + 2)$ input pins and n output pins. Indeed, this is sufficient to achieve full logic connectivity as shown by Betz in [1]. In addition we assume that all segments are of length 1.

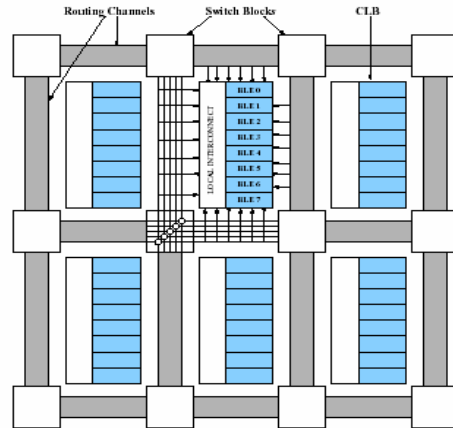


Figure 1. General architecture model

4. MULTILEVEL LOGIC CLUSTERING APPROACH

Clustering is done in 2 phases. First we apply a k -way partitioning to the circuit. So having a circuit consisting of set of modules and set of signal nets, we want to divide it into k clusters such that the number of inter-cluster signal nets is minimized. As we will use an existing partitioning

tool we can't impose several constraints in advance like the number of pins per cluster. That's why in the second phase we have to move some vertices among the partitions to respect such constraints.

4.1 Partitioning running

In this phase, we present the multilevel partitioning algorithm that we have applied to divide the BLEs into clusters. As in the FPGA partitioning problem, BLEs must be divided into k roughly equal parts, we use an algorithm that computes directly the k partitions: hMETIS-Kway [4]. The hMETIS-Kway is k -way partitioning algorithm based on the multilevel paradigm.

As shown in figure 2, the hypergraph is coarsened successively and it is directly partitioned into k parts. Then this k -way partitioning is successively refined as the partitioning is projected back into the original hypergraph.

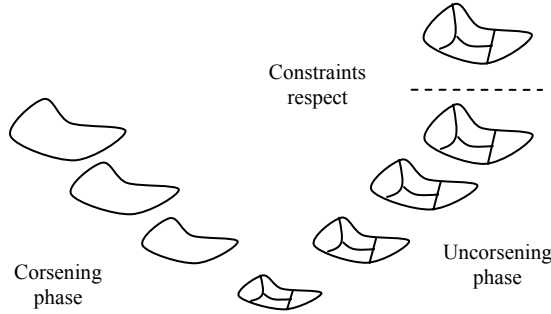


Figure 2. The various phases of the multilevel clustering approach

If during the clustering special properties of the interconnect can be exploited, significant gains can be obtained in terms of routability. when we run the partitioning we choose an objective function that minimises the number of external nets and eliminates nets with high density: A net with a larger number of terminals is harder to route.

4.2 Respecting Constraints

When the k -way hMetis partitioning is run, it's impossible to impose the constraints concerning the partitions' size and the number of inputs per cluster. As it can be seen in figure 1 a "constraints respecting" step was added at the end of the uncorsening phase. First we have to verify for each cluster if the number of blocks exceeds the limit imposed by the FPGA architecture. In this case we have to move some blocks from some clusters and to place them in other ones. In a second step we have to verify if the number of external input nets exceeds the number of cluster's input pins allowed by the architecture. In this case we have also to move some blocks. When we do such moves we will modify the partitions that we have obtained and this can have a bad effect on the objective function. So in both cases we have to select the candidates to move with the best gain. The gain is defined as the number of external nets (\sim number of pins per cluster) to reduce when we move a bloc B from a cluster C.

$$Gain(B, C) = \sum_{i \in Nets(B)} g(i, Nets(C), B)$$

Now we present how the gain for each net is computed. Note that each block output pin is accessible from outside and there is no sharing among the output pins. Therefore there would be no output pin constraints for the clusters. According to this observation, saving on output pins doesn't bring any gain.

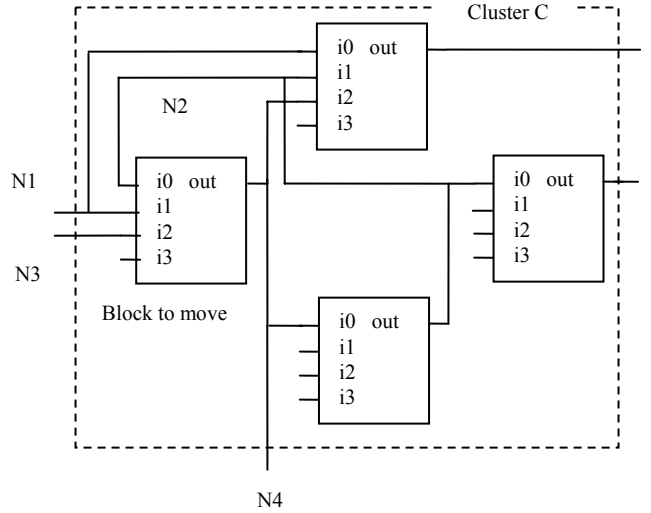


Figure 3. Logic block being moved from a cluster

In figure 3, we show a candidate basic block B and a cluster C. The nets N1, N2, N3 and N4 have different contributions to the gain of moving the block B.

N1 is connected to input pins of two blocks inside the cluster C. So N1 has no improving effect since the external input pin will be kept. So the gain obtained by moving B from cluster C corresponding to net N1 is 0. However since all terminals of the net N2 are inside the cluster (internal net), one input pin of the cluster would be used for N2 if we move the block B. So the gain of moving logic block B to the cluster due to N2 in terms of used input pins per cluster is -1. N3 is connected to only an input pin of the block B in the cluster C. So when we move B an input pin gets free and the gain is 1. The driving pin of N4 is the output of the logic block B. There is an input pin of net N4 inside the cluster C. If we move the block B we need to use an input pin of the cluster to connect the net N4 to other terminals of the net outside the cluster. According to this reason the net N4 has a contribution to the block gain equal to -1.

All different cases yielding different gains are presented in table 1, for one net connected to the candidate block.

Table 1. Gain of moving a block according to a single net

pin in B -> pin in C	Staus	in-pin gain
out -> out	B has the only pin	0

out -> out	Other in pins in C	-1
in -> in	B has the only pin	1
in -> in	Other in-pins in C	0
in -> in	All pins in C	-1
in -> out	-	0

Once we have selected the candidate block to move, we must select the best cluster receiver. The cluster receiver must verify the following conditions:

- Number of blocks is less than the limit number imposed by the architecture;
- Number of input pins doesn't exceed the number imposed by the architecture.

When we add a block to a cluster we must not exceed the number of inputs. So we compute a gain function for each cluster. The gain is the number of the inputs that will be added (negative value) or reduced (positive value) when we insert the block into the cluster.

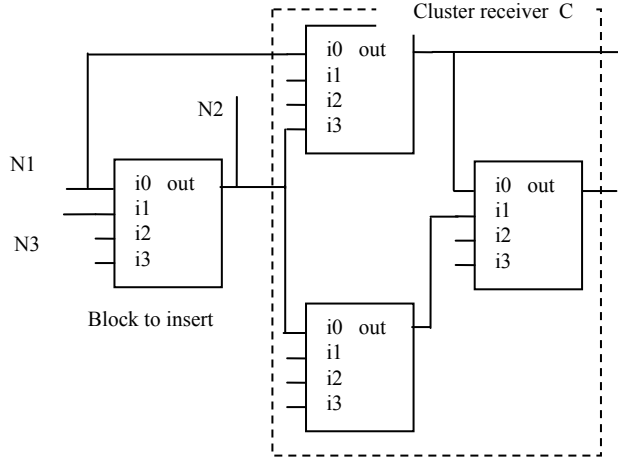


Figure 4. Logic block being inserted in a cluster

In figure 4, a block B and a candidate cluster C are presented. Block B has two common nets with cluster C: N1 and N2. An input pin of the net N1 is inside the cluster C. By adding the block B to the cluster, another input terminal of the net N1 would be inside the cluster. This will not lead to any change concerning the input pins, the gain is equal to 0. The driving pin of net N2 is the output of the logic block B. There is an input pin of net N2 inside the cluster C. If we insert the block B there will be no need to use an input pin of the cluster to connect the net N2 to other terminals of the net outside the cluster. By adding block B to cluster C, an input pin of the cluster gets free and can be used for another net connection. The gain corresponding to this net is equal to 1. Net N3 has no pins inside the cluster C. One pin of the cluster will be used for N3. So the gain of moving logic block B to the cluster due to N3 in terms of used input pins per cluster is -1.

The gain for each block inserting in cluster C can be computed as follows:

$$Gain(B, C) = \sum_{i \in Nets(B)} g(i, Nets(C), B)$$

$g(i, Nets(C), B)$ is defined as the gain obtained in input pins of cluster C as defined in table 2.

Table. 2. Gain of inserting a block according to a single net

pin in B -> pin in C	Status	in-pin gain
in -> out	-	0
in -> in	-	0
out -> in	-	1
in	New net	-1
out	-	0

5. EXPERIMENTAL RESULTS

We have implemented our clustering technique, on the top of VPR [5], we have used a Pentium-4 machine 3 GHz. We placed and routed 18 of the largest MCNC random benchmark circuits [6] on clustered FPGAs. Cluster size 8 was used in our experiments.

5.1 Routability and power consumption

Table 3 shows the routing tracks results for T-VPack [2], R-Pack [3] and our clustering technique. For the same number of clusters T-VPack and R-Pack use about 15% more tracks than our clustering technique. Less number of tracks does not only mean saving wiring area but also decreasing the size of the routing switches. The reduction of needed tracks is directly related to the amount of interconnect power saving.

Table. 3. Routing tracks

		T-Vpack	R-Pack	Ours
Circuit s	Clusters	Channel	Channel	Channel
alu4	192	26	34	26
apex2	240	34	35	33
Apex4	165	37	35	35
bigkey	214	17	15	11
des	200	17	18	15
diffeq	189	20	19	17
dsip	172	14	24	11
elliptic	454	37	32	30
ex1010	599	41	42	31
ex5p	139	37	36	31
frisc	446	39	34	36
misex3	178	29	32	29
pdc	582	52	56	51
s38417	802	29	25	18
s38584	806	32	26	20
seq	221	33	37	34

spla	469	42	48	39
tseng	133	21	21	13
average	344.48	30.49	31.16	25.94

5.2 Circuit speed and run time

Now we will check the effect of our clustering method on circuit speed. We placed and routed the same benches on clustered FPGAs using a timing-driven algorithm. Table 4 shows the critical path results for T-VPack [2] and our clustering technique. We have almost the same speed performance. Our clustering method can deal with the area and speed problems in the same time.

Table. 4. Critical path and run time

Circuits	T-VPack		Ours	
	C.Path (ns)	Run time (s)	C.Path (ns)	Run time (s)
alu4	59.8	47.3	55.3	50.2
apex2	76.0	100.2	69.1	90.3
Apex4	65.7	71.5	63.4	57
bigkey	43.6	58.2	29.6	38
des	46.5	68.3	54.9	62.4
diffeq	43.7	41.5	56.7	39.1
dsip	52.4	45.4	33.4	41.9
elliptic	81.29	260.5	99.7	265.3
ex1010	86.8	525	98.6	495.4
ex5p	110.6	65.2	108.3	56.7
frisc	53.9	280.3	60.0	290.8
misex3	59.3	59.3	57.5	60.7
pdc	126.7	818.7	121.7	825.5
s38417	87.3	360	77.7	341.4
s38584	59.3	420.3	55.9	413.8
seq	63.5	79.5	62.7	88.2
spla	98.1	314	98.7	320.7
tseng	51.3	33.3	50.4	35
average	70.2	202.7	69.6	198.5

The time consumed in our partitioning method is relatively important and this is due essentially to the “uncoarsening” and the “constraints respecting” phases. In table 4 we show the time consumed to run clustering, placement and routing in both cases. We notice that by spending more time in the clustering stage we reduce the time that will be consumed in the placement and routing phases.

6. CONCLUSION

In this paper we proposed a multilevel partitioning method for cluster-based FPGAs. This method improves the routability by decreasing the number of required tracks in the FPGA routing. This method has also a good effect on optimising the critical path. Those improvements were achieved thanks to the multilevel partitioning and the

choice of the objective function. Those results in terms of area and power reduction are important for FPGA embedding on SOC.

7. REFERENCES

- [1] V.Betz, J.Rose and A.Marquardt, “Architecture and CAD for Deep-Submicron FPGAs”, Kluwer Academic Publishers, 1999.
- [2] A.Marquardt, V.Betz and J.Rose “Using Cluster-Based Logic blocks and Timing-Driven Packing to improve FPGA speed and density” ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Montrey, CA, February 1999, pp.37-46.
- [3] E.Bozozzadeh, S.Ogrenci-Memik, M.Sarrafzadeh, “RPack: Routability-driven Packing for cluster-based FPGAs”, Proceedings, Asia-South Pacific Design Automation conference, January 2001.
- [4] G.Karypis and V.Kumar, “Multilevel k-way Hypergraph Partitioning”, DAC99, New Orleans Louisiana.
- [5] V.Betz, J.Rose “A New Packing, Placement and Routing tool for FPGA research”, Proc Seventh FPLA, pp.213-222, 1997.
- [6] S.Yang, “Logic synthesis and optimization benchmarks” user guide version 3.0. MCNC, Jan. 1991.