Performance Driven Technology Mapping for Lookup-Table Based FPGAs using the General Delay Model

Anmol Mathur
C. L. Liu

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue
Urbana, Illinois 61801 USA

Abstract. In this paper we present an efficient algorithm for performance driven technology mapping for table-look-up based FPGA architectures using the general delay model. Since our algorithm does not impose any restrictions on the allowed values of edge delays, it can use delay information generated by a layout phase to intelligently guide the technology mapping in an iterative loop. The algorithm uses a set of node weights to measure the criticality of the nodes in a boolean network. A biased depth first search is then used to generate a topological ordering of the nodes and this ordering is used to guide the clustering step. An important feature of our algorithm is that it exploits reconvergent paths in the network automatically. The technology mapped circuits produced by our algorithm were compared with those produced by other algorithms. In terms of delay, our algorithm produces results with delays that are 30% less than those produced by *flowmap* on the average, showing that minimizing the number of levels in the technology mapped circuit does not ensure a low delay realization. In terms of the number of logic blocks, our results are comparable to those produced by *flowmap*.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are becoming increasingly important in the design of Application Specific Integrated Circuits (ASICs) since they provide both large scale integration and user-programmability. User-programmability results in shorter turn-around time and lower manufacturing costs. FPGAs are more versatile than PLAs since they can be used to implement multi-level combinational logic.

An FPGA is a regular array of programmable logic blocks, also referred to as *configurable logic blocks* (CLBs) each of which is capable of implementing any boolean function of k inputs. These CLBs can be interconnected using programmable switches which connect the horizontal and vertical wire segments in the tracks between the CLBs in the array.

Logic synthesis for FPGAs is the process of mapping a set of boolean functions to a set of CLBs which are then programmed and interconnected to realize the functions. The process involves a *technology independent* phase in which various transformations are applied to simplify the input functions, followed by the *technology mapping* phase. Technology mapping is followed by placement of the CLBs and routing of the interconnections.

The technology mapping phase involves clustering

of a directed acyclic graph corresponding to a boolean network into components, each of which is mappable onto one CLB of the FPGA, so as to optimize various area and delay objectives. The clustering step can be guided by one or more of the following goals :

- Minimizing the number of CLBs or equivalently, minimizing the number of components in the clustered boolean network, resulting in lower area requirement.
- Minimizing the maximum delay along a path from an input to an output in the clustered boolean network.
- Producing an easily routable mapping.

Much of the early work on design automation for FPGAs focussed on technology mapping for area minimization, [7] [3]. Recently more effort has been directed towards algorithms for FPGA technology mapping which minimize the delay in the critical path. The *Chortle-d* technology mapper [4] attempts to minimize the depth of the clustered circuit by using bin packing heuristics geared towards reduction of the number of levels. *DAG-map* [1] uses Lawler's labeling algorithm to produce a low depth implementation of the given boolean network. However, this algorithm is optimal only for DAGs which are monotone with respect to the number of distinct inputs entering a logic cone. The *Flow-Map* mapper [2] uses a modified version of Lawler's labeling algorithm, computing the labels by finding feasible cuts in a network flow formulation. This algorithm produces a minimum depth implementation of the boolean network and is applicable to arbitrary networks. *TechMap* [10] uses clique partitioning in a compatibility graph to cluster the given boolean network. However, all these algorithms attempt to minimize the number of levels in the clustered circuit, without considering the actual delays which might arise when the clustered circuit is placed and routed. The underlying assumption for all these approaches is that the net delays between CLBs in the final design are relatively uniform. The scarcity of routing resources and the fact that there are substantial differences in delays among the available routing resources, make the above assumption unrealistic. In fact, our results indicate that often clusterings with larger number of levels, but a better "distribution" of the delays in the various paths result in lower critical path delay in the actual implementation.

To the best of our knowledge, the only technology mapper which takes into consideration actual delays in the technology mapping phase is the modified *mis-pga* mapper [8]. It uses a two phase approach: the first phase involves delay optimization during logic synthesis before placement, and the second uses logic resynthesis during a timing-driven placement phase.

In this paper, we study the problem of technology mapping for FPGAs, with delay minimization as the primary objective. The delay model used is different from the ones used in all previous approaches to solve this problem. We allow different interconnections to have different delays even if they belong to the same net. An efficient algorithm for integrating technology mapping and layout, based on longest path computations and ***biased*** depth first search in directed acyclic graphs is proposed.. An iterative approach is employed to use the delay information generated after placement and routing to direct technology mapping. The rest of the paper is organized as follows: Section 2 defines the terms used in the paper and formulates the problem precisely; Sections 3 presents an overview of our iterative technology mapping algorithm; results and comparison to other technology mappers on some MCNC benchmarks are presented

in Section 4.

2 Problem Formulation

The technology mapping phase for implementing a given set of boolean functions, represented as a directed acyclic graph (DAG), $G(V, E)$, involves clustering of the DAG into components satisfying the constraints of FPGA technology. In this paper we consider lookup-table based FPGAs, which use static RAMs to implement the configurable logic blocks. The Xilinx FPGA series is an example of such FPGAs [11]. The technology constraints in such FPGAs arise mainly from the architecture of the logic blocks and are of the form:

1. The number of distinct inputs to a logic block is at most k (k is 5 for the Xilinx 3000 series architecture).
2. The number of distinct outputs of a logic block is at most k'.
3. If more than one boolean function is being implemented in a logic block then there are constraints on the maximum number of inputs that can be shared between the functions.

For most FPGAs currently being marketed, k ranges between 4 and 9, and k' is 1 or 2.

Given a node v, we define $cone(v)$ to be the subgraph of G induced by the set of nodes consisting of v and all its predecessors in G [1]. We will use the terms cone and *logic cone* interchangeably in this paper. A *cluster* is a set of nodes $C \subset V$, such that for any nodes $u, v \in C$ all nodes on the paths between u and v are also in C. A cluster C is said to be *feasible* if it can be mapped to a logic block without violating any of the technology constraints mentioned earlier.

[1] u is a predecessor of v if there is a directed path from u to v.

Some of the terms used in the rest of the paper are illustrated in Fig. 1.

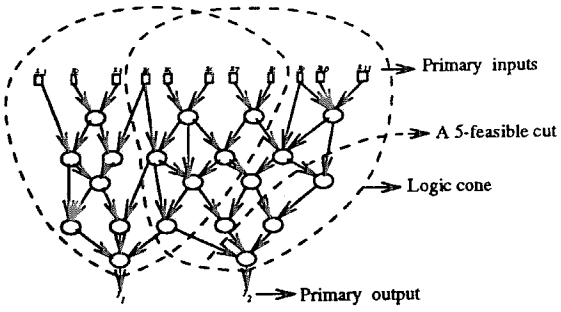

Figure 1: A boolean network illustrating some terms.

A *valid clustering* of a DAG is a set of clusters, $C = \{C_1, C_2, \cdots, C_n\}$, such that

1. All the clusters are feasible.
2. Each node of the DAG is contained in at least one cluster.
3. For each input (that is not a primary input), y, to a cluster C_i, there must be a cluster $C_j \in C$ that has y as an output.

In a valid clustering of $cone(v)$, the cluster covering v is referred to as the *apical cluster*. For a given clustering C, an edge (u, v) is *visible* with respect to a path P in the clustered network, if $u \in C_i$ and $v \in C_j$ and C_i and C_j are adjacent clusters on the path P. Otherwise, (u, v) is *invisible* with respect to path P.

It is important to note that we do not require that the clusters in a valid clustering be pairwise disjoint. Non-overlapped clustering can be arbitrarily bad with respect to both the number of clusters used and the delay in the critical path. Thus, unlike many traditional clustering algorithms, FPGA technology mapping algorithms produce overlapping

clusters. The overlap between clusters entails duplication of logic, but this is not a severe penalty in FPGAs since the logic blocks are constrained not by the amount of logic (as measured by the number of nodes) but by the number of inputs.

The result of the technology mapping process is a valid clustering of the given boolean network. Subsequently, each cluster is mapped to a logic block and the logic blocks are interconnected according to the edges between the clusters.

2.1 Delay Models

The delay model used plays an important role in the design of performance driven technology mapping algorithms, since it dictates the underlying assumptions which the algorithm uses to estimate delays. The *unit delay model* assumes the delay between CLBs to be constant for all interconnections. This assumption is not realistic when communication delays form the dominant part of the total delay. The invalidity of this assumption for the case of FPGAs is further amplified by the scarcity of routing resources, which might force logically adjacent CLBs to be physically distant. The *nominal delay model* allows different nets to have different delays, but requires that all the interconnections in the fanout of one node have the same delay. The Flow-map algorithm [2] can be modified to produce an optimal technology mapping under this model. This model is more accurate than the unit delay model, but still does not present a totally correct picture, since it is possible for different interconnections in the fanout of a node to have vastly different delay values. The *general delay model* allows for different delays on interconnections, even if they fan out from the same node. This model captures the fact that different branches of the same net may be routed on paths with different delays. In the context of FPGAs, this is a more realistic delay model since different edges of the same net might be routed along paths having significantly different delays. Thus, forcing all edges on a net to have the same delay (nominal delay model) or assuming that all edges in the network have the same delay (unit delay model) can result in erroneous identification of critical paths. The Penfield-Rubinstein model [9] also results in different delays in different branches of a net.

For a path in the clustered network, we define the delay of the path to be the sum of the delays of the *visible* edges in the path. In the implementation of the clustered network, both the interconnections and the logic blocks contribute to the delay of any path. However, the delay due to the logic blocks can be added to the delays on the connections in the fanout of the logic block without altering the delay of any path passing through the logic block. This justifies the assumption of assigning delays only to edges in the boolean network. A *critical path* in the clustered network is a path with maximum delay. The minimum delay clustering problem is to find a valid clustering of the given network which has the smallest critical path delay. Notice that exact values of delays on the interconnections depend on the actual placement and routing of the clustered network. Consequently, estimates of delay values are used during the technology mapping phase.

3 An Overview of Our Approach

The labeling based approaches for minimizing the number of levels [1], [2] make use of the fact that the optimal clustering of the cone of a node can be constructed using some of the optimal clusterings of its predecessors. However, in the general delay model,

sub-optimal clusterings at some of the predecessors might be required to prevent edges with large delay from becoming visible. This causes the dynamic programming formulation for delay minimization in the unit delay model to break down for the general delay model. We now outline our heuristic for finding "good" clusterings in the general delay model.

3.1 The Iterative Improvement Algorithm

Since our technology mapping algorithm does not make any assumptions about the delay values assigned to the edges, it can be coupled with a placement and routing algorithm to form an iterative technology mapping algorithm. Getting good estimates of delays is the major bottleneck in using the general delay model in the technology mapping stage. One possible solution is to have an iterative algorithm which feeds back the delay information obtained after placement and routing to the technology mapping phase of the next iteration.

The iterative algorithm starts with some delay estimates as the edge delays. In our experiments we found that the fanout of a node is a good estimate of the delay on the edges fanning out from that node. Our technology mapping algorithm is used to obtain a technology mapped circuit which is then placed and routed. From the placed and routed circuit we recompute the edge delays. A strategy for recomputing the edge delays is to assume the delay of an edge invisible in a mapping to be zero and the delay of the visible edges to be their actual propagation delays. In any iteration, the technology mapper then uses a (weighted) mean of the delay values assigned to an edge in all the previous iterations as the delay of the edge. A flowchart for such an algorithm is shown in Fig. 2. In fact, our algorithm can be viewed as one

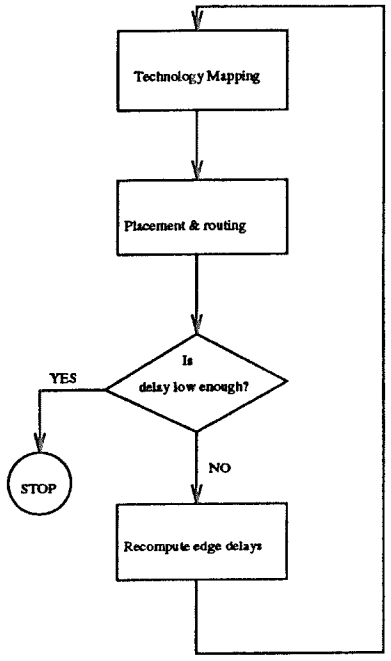

Figure 2: The flowchart for the iterative technology mapping algorithm.

that integrates the technology mapping , placement and routing steps in an iterative loop. The input to our algorithm is a boolean network and the output is a completely placed and routed design.

The experimental results we obtain indicate that such an approach is quite effective.

3.2 Biased Depth First Clustering

Given an edge weighted, k-feasible boolean network $G(V, E)$, we want to generate a clustering which minimizes the maximum value of the sum of the delays of the visible edges on any path from a primary input to a primary output. Depth first search starting from the primary outputs seems to be a natural way to achieve a low delay clustering, since it tries to grow clusters ***along*** a path as much as possible. However, a naïve depth first search cannot be expected to per-

form well because it does not use the delay information in an intelligent manner.

Our algorithm employs the idea of a *biased depth first search* to produce a low delay clustering. We use the delay information to compute node weights which bias the depth first search to explore paths with greater delays first. The main motivation for this is that exploring high delay paths preferentially will cause "long" clusters to be generated along these paths, resulting in fewer visible edges on these paths. This should insure that the delay along long paths is not too large in the clustered network. Implicit in this approach is the following assumption about the relation between the critical paths in the original network and the clustered network:

- The critical paths in the original network are the paths whose nodes form the clusters on the critical paths in any optimal clustering of the network.

Since the clusters are constrained by the number of inputs, the above assumption translates into the requirement that the number of distinct inputs to nodes on a path in the boolean network be proportional to the length of the path. This requirement is satisfied for most real circuits, although it is possible to contrive boolean networks having long paths with very few distinct inputs. Thus, biased depth first search should perform well on networks which satisfy this assumption.

The algorithm performs technology mapping for the logic cone of each primary output separately. For each logic cone, node weights are computed based on the delays; a biased depth first search is used to generate a topological ordering of the logic cone; densities of crossing edges are computed and a feasible cut of minimum weight is found to generate the apical cluster. Deleting the apical cluster from the logic cone

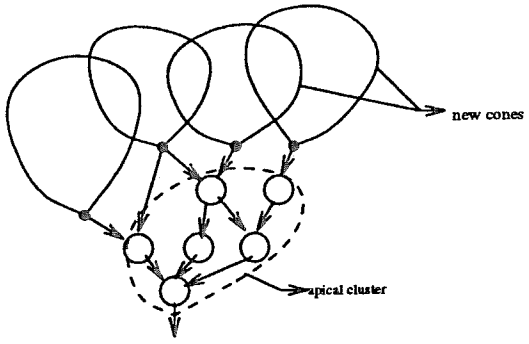

Figure 3: An apical cluster and the logic cones generated on deleting the apical cluster.

results in a collection of (overlapping) cones each of which is clustered recursively (see Fig. 3). However, the biasing weights and biased topological order do not need to be recomputed for the new cones.

Here is the pseudocode describing the clustering algorithm at the top level:

```
for each primary output in the boolean network {
    Label the logic cone of the primary output;
    Sort_array = topological_sort(logic cone);
    Compute_biasing_weights(sort_array);

    /* Now we do the biased DFS */
    Biased_sort_array = topological_sort(weighted logic cone);

    /* Next we find the apical cluster */
    Find_apical_cluster(biased_sort_array);

    Recursively cluster logic cones generated by deleting
    the apical cluster;
}
```

The biasing weight of a node in the DAG is the length of the longest path passing through the node. These weights can be computed for all the nodes in the DAG in linear time using a dynamic programming

based algorithm. The biased depth first search produces a linear ordering of the nodes in the cone with the property that nodes with higher weight occur as far to the left (see Fig. 5), as long as the topological sorting constraint is not violated. Finding an apical cluster requires the computation of a cut that has at most k crossing edges, so that all the nodes on one side of the cut form a feasible apical cluster. In order to find a "good" apical cluster, we want a feasible cut that exposes edges having low delays, and puts a large number of nodes in the apical cluster. In our algorithm, we use a combination of the volume of the cut and the delays on the edges it exposes to choose a good cut.

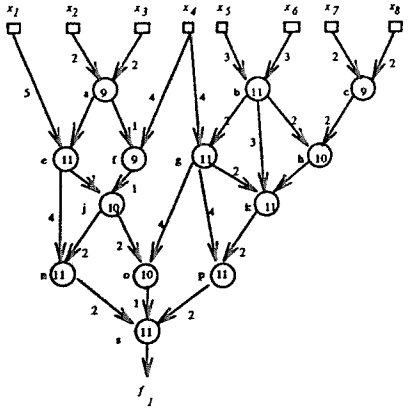

Figure 4: A logic cone with biasing weights.

Figures 4 and 5 show an example of a boolean network with edge delays, biasing weights, and the resulting biased topological sorting that is used to compute the apical cluster.

Details of these steps are omitted due to lack of space.

Note that since the biased topological sort orders the nodes in the cone according to decreasing criticality, using the maximum volume cut to define the apical cluster is a good strategy with respect to delay reduction because it attempts to reduce the number of visible edges on the longer paths. Also, our algorithm takes a more global view in finding the cut defining the apical cluster, since it examines the densities at all points in the linear arrangement produced by the biased topological sort. This enables it to "climb" over some "hills" in the density function. This "hill climbing" effect is visible in Fig. 5, where the increase in density after node h to a value larger than 5 does not prevent the algorithm from finding the feasible cut after node j.

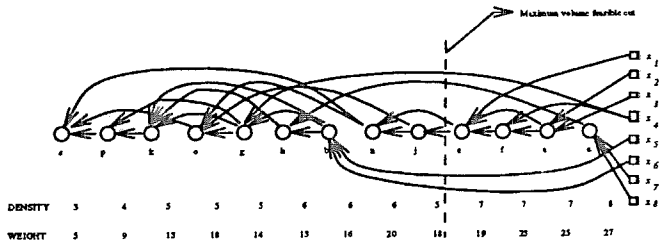

Figure 5: Topological order for the cone in Fig. 4 along with cut densities and cut weights.

Another salient feature of our algorithm is that it exploits reconvergent paths ([3]) automatically. Also, predecessor packing which is carried out as a post-processing step in [2] is carried out in the clustering phase itself.

4 Experimental Results

We have implemented the biased depth first search clustering algorithm and tested it on a set of MCNC benchmark examples. The implementation of our performance driven clustering algorithm was integrated as part of the *misII* program and can be invoked using the command *bias_clus* command. We used a version of *misII* that had the *flowmap*, *dagmap* and *flowpack* commands available.

The following sequence of steps was used in gener-

ating the experimental results:

1. Read in the benchmark boolean network using *read_blif* or *read_pla* after invoking *misII*.

2. Transform the boolean network into a simple and-or circuit using *tech_decomp -a 1000 -o 1000*.

3. Transform the and-or circuit into a 2-feasible boolean network using *tech_decomp -a 2 -o 2*.

4. Use *bias_clus* to cluster the 2-feasible network.

5. Use *write_xnf* command to output the clustered network in the Xilinx netlist format.

6. Transform the *.xnf* file produced to a *.lca* file using the sequence of Xilinx XACT commands *xnfmap* and *map2lca*.

7. Use the Xilinx automatic place and route tool *apr* to place and route the design obtained as a result of the previous step. We used the command
   `apr -i 0.0`
   to produce a path delay table in the report generated by *apr*. The path delay table contains the delays (in nanoseconds) for all the paths from primary inputs to primary outputs. This table was used to compute the worst case delay from primary input to primary output.

Table 1 shows a comparison between our results and the results in [5] where *flowmap* was used to perform the technology mapping and then *apr* was used to do the placement and routing. The column *CLBs* is the number of CLBs in the technology mapped circuit; the column *depth* is the maximum number of CLBs on a path in the clustered circuit, and the column *delay* is the delay in the critical path of the circuit. Although our algorithm produced circuits with larger depth than *flowmap* (*flowmap* produces minimum depth circuits), our results on delays in the critical path are consistently better (with the exception the circuit *bw*) to those for *flowmap*.

Thus, the critical path delay after placement and routing is about 30% better, on the average, than the results obtained when *flowmap* was used as the technology mapping algorithm. This supports our claim that minimizing the number of levels in the technology mapping process is not enough to guarantee a low delay in the critical path after placement and routing . Instead, it is often useful to use a more accurate estimate of the actual delays and perform performance driven technology mapping in the general delay model.

Table 2 shows the comparison of our results with those in [8] for the circuits which we were able to place and route using *apr*. The circuits that we were not able to place using *apr* had more CLBs than the limit imposed by the Xilinx 3000 series FPGAs. The results produced by our algorithm are better than those for *xln_p* on 6 of the benchmark circuits. However, the delay is much larger for the circuits *5xp1* and *misex2*. This is probably due to the fact that our algorithm used a significantly larger number of CLBs for these two circuits, resulting in a much harder placement and routing problem.

We tested our iterative algorithm on the benchmark – *count* with 129 nodes (in the unclustered feasible circuit). In three iterations, the delays in the critical paths show significant decreases. Table 3 shows the delays of the five most critical paths after each of the three iterations. Tables 4 and 5 show how the delays in each critical path changes in successive iterations. The first column of Table 4 shows the delays of the five most critical paths after the first iteration. The second column shows the delays of these paths

Circuit	flowmap			bias_clus		
	CLBs	depth	delay	CLBs	depth	delay
cordic	17	4	58.3	21	6	38.3
count	34	5	87.0	30	5	61.0
bw	28	1	39.3	28	1	68.5
f51m	42	7	99.9	48	9	84.7
frg1	60	6	87.1	52	6	47.7
comp	68	7	105.0	69	7	60.8
myadder	24	8	113.1	31	8	45.0
9sym	96	5	98.3	75	5	52.0
term1	117	5	96.1	79	5	86.5

Table 1: Comparison of *flowmap* and *bias_clus*. All the delays are in nanoseconds

Circuit	xln_p	bias_clus
z4ml	31.0	25.2
misex1	36.2	34.0
vg2	76.3	65.3
5xp1	35.9	76.4
misex2	53.7	82.0
count	79.02	49.3
9symml	54.0	61.0
9sym	53.7	52.0
f51m	76.6	74.7

Table 2: Comparison of *xln_p* and *bias_clus*. All delays are in ns.

Iteration number		
1	2	3
61.0	56.3	54.0
58.3	54.0	53.5
55.8	54.0	53.2
55.2	53.2	52.1
54.0	52.0	50.0

Table 3: Delays (in ns) of the 5 most critical nets after each of the 3 iteratio ns for circuit *count*.

Iteration 1	Iteration 2
61.0	50.3
58.3	55.2
55.8	56.3
55.2	56.2
54.0	55.0

Table 4: Change in the delays (in ns) of the 5 most critical paths between iterations 1 and 2.

Iteration 2	Iteration 3
56.3	49.0
54.0	53.2
54.0	51.0
53.2	54.0
52.0	50.0

Table 5: Change in the delays (in ns) of the 5 most critical paths between iterations 2 and 3.

after the second iteration. Similarly, the first column of Table 5 shows the delays of the five most critical paths after the second iteration and the second column shows the delays of these paths after the third iteration. Note that the most critical paths after one iteration do not remain the most critical paths after the next iteration because our algorithm is biased to cluster along the most critical paths.

5 Conclusions

In this paper we have presented a performance driven technology mapping algorithm for table-look-up based FPGAs using the general delay model. Our biased depth first clustering automatically exploits reconvergent paths and does predecessor packing making it quite competitive with respect to the number of CLBs produced in the technology mapped circuit. As is evident from the results, reducing the depth of the technology mapped circuit alone is no guarantee for a low delay realization of the original circuit. It is crucial to take some estimate of the actual delay incurred when a particular edge is exposed into consideration in the technology mapping process.

References

[1] J. Cong, A. Kahng, P. Trajmar, K. C. Chen, *Graph Based FPGA Technology Mapping for Delay Optimization*, ACM International Workshop on Field Programmable Gate Arrays, 1992, pp. 77–82.

[2] J. Cong, Y. Ding, *An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs*, Proc. of ICCAD, (1992), pp. 48–53.

[3] R. Francis, J. Rose, Z. Vranesic, *Chortle-crf : Fast Tech nology Mapping for Lookup Table-Based FPGAs*, Proc. of 28th ACM/IEEE Design Aut omation Conference, (1991), pp. 227–233.

[4] R. J. Francis, J. Rose, Z. Vranesic, *Technology Mapping for Delay Optimization of Lookup Table-Based FPGAs*, Proc. of ICCAD, (1991).

[5] T. Gao, K.-C. Chen, J. Cong, *Placement and Placement Driven Technology Mapping Algorithms for Xilinx FPGAs*, IEEE ASIC Conference, 1993.

[6] E. L. Lawler, K. N. Levitt, J. Turner, *Module Clustering to Minimize Delay in Digital Networks*, Proc. IEEE Transactions on Computers, C-18(1) (1969), pp. 47–57.

[7] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, A. Sangiovanni-Vincentelli, *Logic Synthesis for Programmable Gate Arrays*, Proc. of 27th ACM/IEEE Design Automation Conference, (1990), pp. 620–625.

[8] R. Murgai, N. Shenoy, R. K. Brayton, A. Sangiovanni-Vincentelli, *Performance Directed Synthesis for Table Look Up Programmable Gate Arrays*, Proc. of ICCAD, (1991), pp. 572–575.

[9] J. Rubinstein, P. Penfield, M. A. Horowitz, *Signal Delay in RC Tree Networks*, IEEE Transactions on CAD, July 1983, pp. 119–127.

[10] P. Sawkar, D. Thomas, *Area and Delay Mapping for Table-Look-Up Based Field Programmable Gate Arrays*, Proc. of 29th ACM/IEEE Design Automation Conference, (1992), pp. 368–373.

[11] Xilinx Programmable Gate Array Data Book, Xilinx Corporation, 1991.