Dominator-based Partitioning for Delay Optimization [*]

David Bañeres
Univ. Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Univ. Politècnica de Catalunya
Barcelona, Spain

Mike Kishinevsky
Strategic CAD Lab, Intel Corp.
Hillsboro, OR USA

ABSTRACT

Most of the logic synthesis algorithms are not scalable for large networks and, for this reason, partitioning is often applied. However traditional mincut-based partitioning techniques are not always suitable for delay and area logic optimizations. The paper presents an approach that uses a dominator-based partitioning and conventional logic synthesis techniques for delay optimization of large networks. The calculation of dominators is crucial to find topologically ordered clusters suitable for logic restructuring. As a result, a scalable and efficient strategy for delay optimization is proposed and evaluated, showing tangible improvements with respect to existing techniques. A comparison with a standard mincut-based partitioning technique is also presented.

Categories and Subject Descriptors: B.6.3 [Hardware]: Logic Design - Design Aids; J.6 [Computer Applications]: Computer-aided engineering.

Terms: Algorithms, Design.

Keywords: logic partitioning, logic design, timing optimization.

1. INTRODUCTION

Graph partitioning is the traditional *divide-and-conquer* strategy proposed to afford different synthesis problems in large Boolean networks [1–5]. Many of these approaches are reduced to the *mincut* problem, that has been extensively used in placement [6] or FPGA partitioning [7, 8], where the minimization criteria is the number of wires among different parts of the circuit. However, a cut-size partition does not control the type of input-output interactions produced by the external connections. This problem is specially important for timing optimization in which logic restructuring must be performed guided by the critical paths that traverse the entire circuit from inputs to outputs.

Different approaches have been developed for timing optimization [9–13]. However, most of them cannot be used in large circuits because of their complexity. Instead of using *mincut* partitioning, some approaches create delay-driven partitions. In [14], a simple method for clustering (reduce_depth) was proposed: the nodes are clustered in DFS order and new clusters are created when a capacity constraint is exceeded. The method in [15] uses a similar approach, but enables the duplication of logic to improve the delay of the critical paths.

The method presented in [16] is able to find a delay-driven or area-driven partition. Initially, a good area-oriented partition under delay constraints is found and a refining step is done to obtain the final partition. However, delay results are not satisfactory, since the initial area partition has a high influence on the final solution.

To our knowledge, DEPART [17] is the best method to perform a delay-driven partitioning for large Boolean networks suggested so far. Only nodes in the transitive fanin of the critical outputs are selected for restructuring. The cones of logic of different outputs are clustered in the same block if their size is less than the capacity constraint. In this paper, DEPART is taken as a reference for comparison. The drawback of this method emerges when the cone of a critical output does not fit in one cluster. In this case, new clusters are greedily created until all clusters are small enough to fit in the size constraint.

Contributions. The main contribution of this paper is a novel method for network partitioning oriented to timing optimization. The method is based on the concept of *vertex dominator* and generates *topologically ordered clusters* that enable a correct propagation of delay information. With this strategy, tangible improvements can be obtained in delay, thus better exploring the area-delay trade-off of Boolean networks.

Based on the partitioning method, an algorithm for delay optimization is proposed. This algorithm is guided by the criticality of the nets and combines delay and area optimization in the same framework, using the dominator-based partitioning.

An overview of the approach is presented in Section 2. Section 3 presents the required background on vertex dominators. The new partition method is described in Section 4. Finally, the overall delay minimization strategy is explained in Section 5, and experimental results are reported in Section 6.

2. OVERVIEW

The method DBP (Dominator-based partitioning) presented in this paper aims at capturing fragments of critical paths that have small fanout to the rest of the circuit. Thus, the clusters tend to be deep (many levels) with internal nodes having little fanout to external nodes. This type of clusters offers more possibilities for restructuring towards delay minimization.

Clusters with little external fanout are sought by finding *dominators*. Intuitively, a dominator of a node *n cuts* all paths from *n* to the outputs [18]. This concept can be extended to multiple-vertex dominators [19] when the paths are cut by several nodes.

Figure 1 emphasizes the difference between hMetis [3] and DBP

[*]This work was supported by grants from Intel Corporation, CICYT TIN2004-07925-C03-01, European Social Fund, FI and a Distinction for Research by the Generalitat de Catalunya.

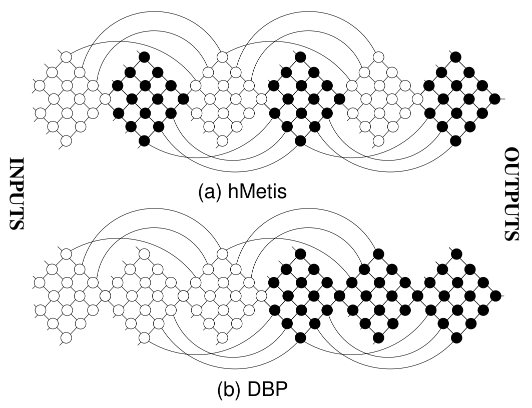

Figure 1: Comparison of hMetis and DBP.

in a particular example[1]. It is an "artificial" circuit with 2-input AND gates that has been created only for this comparison. In both cases, the graph is partitioned into two clusters, denoted by the $\circ$ and $\bullet$ nodes, respectively. The cut-size generated by hMetis is 5 (the output edges of the same node are assumed to belong to the same hyperedge), whereas the one generated by DBP is 7. However, the clusters generated by hMetis cannot be topologically ordered, since there are edges $\circ \rightarrow \bullet$ and $\bullet \rightarrow \circ$. The clusters generated by DBP can be topologically ordered. Finding a topological order is crucial to propagate the arrival times of the outputs of one cluster to the inputs of the successor clusters. The partition obtained by DBP has been determined by the output node, that is a single dominator of the whole graph. The leftmost cluster has been obtained by taking the nodes closer to the dominator without exceeding the pre-defined cluster size. The rightmost cluster has been obtained by gathering the remaining nodes.

Another consequence of the dominator-driven partition is that the longest path is divided into two parts using DBP. However hMetis splits it into six subpaths, thus preventing the optimization across the cluster boundaries. DBP can optimize every cluster in topological order, from inputs to outputs, propagating the obtained arrival times. This propagation is essential to achieve a global restructuring of the network. In this example, the circuit originally has 36 levels. The final circuit after restructuring each cluster, produces a result with 26 levels with hMetis and only 10 with DBP.

The moderate size of the clusters enables the use of conventional delay optimization techniques and to iterate over the network several times to gradually reduce delay with different cluster boundaries. This strategy produces results with better quality, still keeping the method scalable for large networks.

3. PRELIMINARIES

We use the classical representation of Boolean networks as graphs.

DEFINITION 3.1. *A* Boolean network *is a directed acyclic graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes of the network and $\mathcal{E}$ the set of wires. A Boolean function is associated to each node of the network. The nodes with no fanin are the primary inputs, whereas the nodes with no fanout are the primary outputs.* □

A subset of nodes induces a *window* (cluster) in a Boolean network. The window contains all edges between nodes of the window. A window can be *extracted* from the Boolean network, transformed and inserted back into the network. For that, the set of inputs and outputs of the cluster must be identified to preserve the interface with the rest of network, as shown in Fig. 2. Note that, in general, a window need not be connected.

[1]For simplicity, the arrows of the edges are not shown and are implicitly assumed to go from the inputs to the outputs.

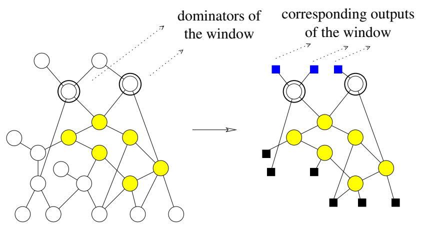

Figure 2: Window extraction in a Boolean network.

```
DominatorPartition ($G$, $S$)
{Input: Network's graph $G$. Size limit for a window, $S$}
{Output: Graph $G$ clustered into windows }
  repeat
    $Doms$ := FindDominators ($G$)
    while $Doms \neq \emptyset$ do
      $bestDominator$ := SelectBestDominator ($Doms$, $S$)
      $window$ := ClusterDominatedWindows ($G$, $bestDominator$)
      $Doms$ := $Doms$ - $window$
    endwhile
  until No changes
  ClusterSmallWindows($G$, $S$)
```

Figure 3: Algorithm for partition a network into windows.

The concept of *dominator* is a key in this paper. Dominators are widely used in several areas, such as code optimization in compilers [20] and test pattern generation techniques [21]. Recently, dominators have been used in logic synthesis for non-disjoint decomposition of Boolean functions [19].

DEFINITION 3.2 (DOMINATOR). *Given a network $G = (\mathcal{V}, \mathcal{E})$, a subset of nodes $X = \{x_1, x_2, \cdots, x_k\} \subset \mathcal{V}$ is a dominator of a node $u \in \mathcal{V} \setminus X$ if:*

- *Every path from u to a primary output contains some vertex $x_i \in X$, and*
- *no proper subset of X is a dominator of u.*

This definition can be naturally extended to sets of nodes, i.e. a subset of nodes being the dominator of another subset of nodes. □

In our work, we are interested in windows with few primary outputs. In the example of Fig. 2, the extracted window has three primary outputs, derived from the three pins connected from the two dominator nodes of the window to their fanouts.

Dominators with only one (two) nodes are called single (double)-vertex dominators. In general, a dominator with k nodes is called a k-vertex dominator. For the calculation of dominators we use the algorithms from [18, 19, 22].

EXAMPLE 3.1. *In Fig. 4(a), $\{b\}$ is a single dominator of $\{g, h, q\}$, but does not dominate r since there is a path from r to c that does not cross b. $\{r, s\}$ is a double-vertex dominator of $\{z, C\}$. Dominators of larger size can also be found, e.g. the 4-vertex dominator $\{h, i, j, k\}$ of $\{r, s, z, C\}$. Note that every node (or set of nodes) can have different sets of multiple-vertex dominators. For example, $\{C\}$ is also dominated by $\{r, s\}$ and $\{h, i, s\}$.* □

4. PARTITION METHOD

The method presented in this paper aims at partitioning the nodes of the Boolean network into a set of disjoint windows. This process is initiated by considering every node as a window and iteratively clustering them to build larger windows without exceeding some capacity constraint. In this iterative clustering, the concept of *dominator* plays an essential role. We next describe the details of the partitioning algorithm, presented in Fig. 3. The execution of the algorithm is later illustrated with the example shown in Fig. 4.

4.1 Core of the algorithm

Initially, every node of the network is a window. The function `FindDominators` calculates all the single and double-vertex dominators of the network. The complexity of finding dominators is $O(n^k)$, where n is the number of graph nodes, and k is the number of nodes in the dominator. The Lengauer-Tarjan algorithm [18] and the algorithms described in [19, 22] are used to search for single- and double-vertex dominators, respectively [2].

The innermost loop merges windows using the calculated dominators. In the clustered graph, the weight of each edge between a pair of windows indicates the number of wires between nodes of the two windows. The algorithms receives a parameter S (size) that defines the maximum capacity for a window.

The selection of the best dominator is performed according to the characteristics of the window of dominated nodes (including the dominator itself). The criteria to select the best dominator are the following (in priority order):

- Smallest number of output nodes in the window. The output nodes are those having fanout outside the window. Initially, the output nodes are the dominators themselves. In a partially-clustered network, every window acting as dominator can have several output nodes.
- Smallest weight of the outgoing edges from the window.
- Largest size of the window

This selection is based on the observation that a node (or a set of nodes) that dominates many other nodes is a good candidate to become the root of the window, since all the dominated nodes do not have transitive fanout outside of the window.

When the size of the window induced by a dominator exceeds S, some of the nodes are excluded from the window. In the case of delay optimization, these nodes correspond to the least critical nodes in the window (largest slack between the arrival and the required time). In our experiments, the unit delay model has been used to calculate the delay information, after having decomposed the network into 2-input nodes.

The procedure `ClusterDominatedWindows` merges the dominated windows into a single one. The new window may contain other dominators of the network (a dominator can be dominated by another dominator). For this reason, all the nodes from the window are removed from the set of dominators ($Doms := Doms - window$). The clustering proceeds until no more dominators exist.

The outermost loop executes the clustering loop to build larger windows. After the first iteration, most of the windows are not single nodes any longer and the new windows are built by merging windows from the previous iteration. The process continues until no more clustering is possible.

At the end of the main loop, some small windows or individual nodes might remain *orphan*, out from the large windows generated by clustering. The procedure `ClusterSmallWindows` still gives an opportunity for further clustering. Here is where S is taken as a soft constraint and a moderate growth of the windows is tolerated to incorporate neighbouring small windows. In the experiments done in this paper, a 25% increase from S is tolerated in the final phase of partitioning.

4.2 Example of a partitioning

Figure 4 shows an example of applying the `DominatorPartition` algorithm to a small graph. The size limit for a window is defined as $S = 8$. The circles represent simple nodes, the boxes represent windows obtained after clustering. The numbers in boldface inside every window indicate the number of nodes contained in the window. The two other small numbers indicate the number of output and input nodes on the interface of the window. The connectivity cost is indicated by the weights in the edges (omitted if the cost is 1).

Using the original network graph (Figure 4.a) the single vertex dominators are first selected, since they produce windows with the least number of outputs. These dominators are used to cluster simple nodes into windows as indicated by the shadow clouds. If the size limit is exceeded, like in the case of the dominator c that dominates 9 nodes $\{c, i, j, t, A, B, D, E, F\}$, some nodes are excluded from the window using delay criticality information. Assuming, in this example, that all arrival times on the primary inputs are the same, node A is excluded from the window since it has a delay slack larger than other nodes dominated by c.

When the clustered object is a window with multiple nodes, the delay slack is calculated as the minimum slack among all the outputs of the window. Therefore the most critical path captured inside the window is the one that determines the criticality of the window.

After the first iteration, the partially-clustered graph shown in Fig. 4(b) is obtained. At this point, two windows (labelled with 4 and 2) are selected as a double-vertex dominator for node y. The clustering of the three nodes derives another window with 7 nodes and leads to the graph shown in Fig. 4(c).

It is important to realize that the partitioning algorithm never explicitly looks for dominators with more than two nodes. However, multiple-vertex dominators are implicitly used for clustering by iteratively applying the clustering with single- and double-vertex dominators. This property enables an efficient use of multiple-

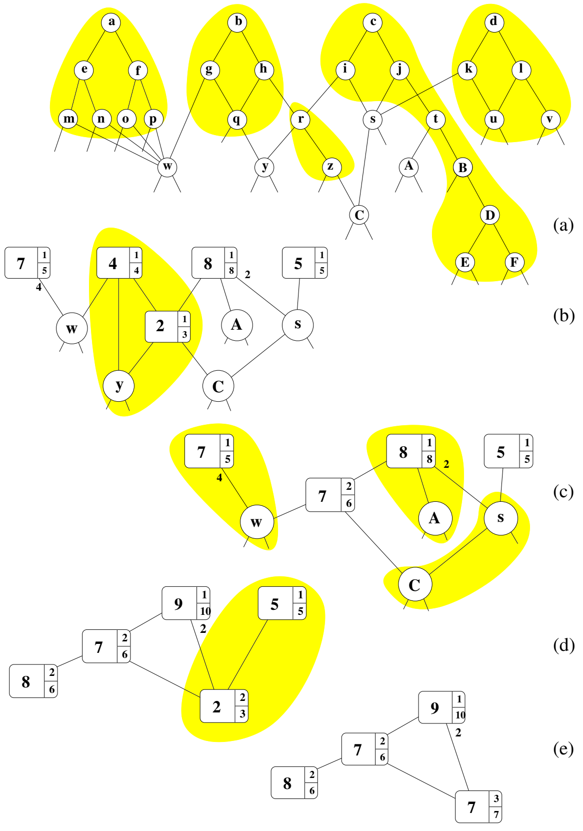

Figure 4: Example of a dominator-based partitioning.

[2]Experimentally it has been observed that calculating larger multiple-vertex dominators has a negligible impact on the results, while increasing the computational complexity significantly.

vertex dominators without an excessive computational complexity.

The main loop of the algorithm completes and delivers the clustering in Fig. 4(c). We can observe that there are still some *orphan* nodes in the graph. Here is where the procedure `ClusterSmallWindows` does the rest of the work. Rather than explaining the details of this procedure, we explain it by means of the intuitive example. The node w is clustered with the leftmost neighboring window with 7 nodes, since the connectivity cost with it (equal to 4) is larger than the one with the neighboring window at the right. Similarly, node A is clustered with its neighbour at the top. Node C is clustered with s (both are small). This leads to the graph in Fig. 4(d).

Finally, the window with two nodes in Fig. 4(d) is clustered with the window with 5 nodes, thus deriving the clustering shown in Fig. 4(e), since the combined window still satisfies the capacity constraint S=8.

4.3 Preserving topological order

In Fig. 4(d), the partitioning algorithm could have also considered clustering the window with label 2 with the window with label 9 (even though the size limit would have been slightly exceeded). However, clustering window 2 with window 9 would create a cycle in the graph due to the three edges $2 \rightarrow 7$, $2 \rightarrow 9$ and $7 \rightarrow 9$ that form a cycle. Graphs with cycles cannot be topologically ordered, which makes easy propagation of the delay information between windows impossible.

The DBP algorithm prevents cycles by imposing the following constraint:

A window cannot be created if one of its output nodes belongs to the transitive fanin of one of its input nodes in the graph before clustering.

5. TIMING-DRIVEN OPTIMIZATION

Figure 5 presents the algorithm for timing-driven optimization of large Boolean networks. Initially, a pre-processing step with low computational cost can be applied to the whole network. In our case, we implemented the algorithms in SIS [23] and used some of the typical scripts for logic synthesis. The network was finally decomposed into 2-input gates.

The algorithm has two main loops: one for delay optimization and another for area optimization. These loops use different optimization scripts (DelayScript or AreaScript in Fig. 5).

The structure of each loop is similar. It is a set of procedures that are repeated until no further improvement is observed. This iterative process is captured by the outermost `repeat-until` loops.

We next describe the details inside each optimization loop. Initially the delay information is calculated: arrival times, required times, and slacks based on the unit delay model. Next, the critical region of the network is extracted. The critical region is defined as the set of nodes with slack smaller than a pre-defined slack (parameter *Slack* in Fig. 5). In case of the area optimization loop, we select a non-critical region according to the same slack information.

The selected region of the network (either for delay or area optimization) is partitioned using the dominator-based algorithm described in Section 4. After this step, a set of disjoint windows that cover the region is defined. The size of the windows (S_d or S_a) is defined according to the computational complexity of the optimization script that is used in the innermost loop. For synthesis scripts with large run-time, the size of the windows must be smaller than the one used for simpler scripts.

The innermost loop visits the windows in topological order, from inputs to outputs. For each window, the synthesis script is applied to optimize either for delay or area. This script transforms the window W without changing its interface to other windows. Next, the delay information is propagated and re-calculated incrementally.

```
DelayOptimization (N, DelayScript, AreaScript, Slack, S_d, S_a)
Inputs:
  N: Boolean network;
  DelayScript: Script for delay optimization;
  AreaScript: Script for area optimization;
  Slack: slack to select the critical region;
  S_d: size limit for windows during delay optimization;
  S_a: size limit for windows during area optimization;
Output: An optimized network N

  Preprocess(N)

  {Delay optimization}
  repeat
    CalculateDelayInformation(N)
    Critical := SelectCriticalRegion(N, Slack)
    DominatorPartition(Critical, S_d)
    for each window W in Critical visited
              in Topological Order from PIs do
      DelayScript(W)
      PropagateDelays (W)
    endfor
  until No delay improvement

  {Area optimization}
  repeat
    CalculateDelayInformation(N);
    NonCritical := SelectNoncriticalRegion(N, Slack)
    DominatorPartition(NonCritical, S_a)
    for each window W in Noncritical visited
              in Topological Order from PIs do
      AreaScript(W)
      PropagateDelays (W)
    endfor
  until No area improvement

  return N
```

Figure 5: Algorithm for optimizing large networks.

Once the innermost loop is completed, another iteration of the outermost loop is executed, until no improvements are produced.

5.1 Optimization of sequential networks

The optimization scheme presented in this paper can be also applied to sequential networks. In this case any window of the network can contain latches. Within a window, combinational and sequential don't cares can be computed. The sequential don't cares can be computed by assuming that the window has a free environment. Thus, a conservative underestimation of the actual don't care set is obtained.

In the current version of our optimization prototype, only combinational optimization has been implemented. Future efforts will be devoted to sequential optimization.

6. EXPERIMENTAL RESULTS

To evaluate the efficiency of the optimization algorithms presented in this paper we have conducted three types of experiments: (1) comparison with DEPART [17] on medium-size examples, (2) comparison with mincut partitioning (hMetis) on large examples, and (3) evaluation of the trade-off between area and delay depending on the size of the windows. When running DBP, the delay script used for optimization was simply the *speed_up* [9] command in SIS, whereas the area script was the *algebraic*.

For the first two experiments the size of the windows for DBP have been selected to be $S_d = 50$ for delay optimization and $S_a = 100$ for area optimization with a *Slack* of 2 units.

Bench.	Levels			Literals			CPU (sec)	
	sp_up	DEP.	DBP	sp_up	DEP.	DBP	sp_up	DBP
C880	21	21	20	834	913	873	8	8
alu4	26	26	25	1214	1917	1398	25	19
C2670	16	17	15	1450	1482	1442	50	46
apex5	14	14	14	1465	1580	1504	2	3
table3	40	44	40	1746	2750	1746	120	37
C3540	33	33	33	2275	2382	2359	33	14
apex3	13	12	13	2569	2810	2575	21	14
seq	14	15	15	2895	3003	2905	19	25
C5315	22	25	22	3081	3099	3152	29	23
pair	18	20	15	3105	3158	3522	16	6
C7552	23	22	21	4127	4577	4547	61	37
des	19	20	19	6083	6223	6385	52	51
C6288	69	75	66	6627	6367	7655	464	236
Norm	1.00	1.05	0.97	1.00	1.08	1.07	1.00	0.58

Table 1: Comparison of speed_up, DEPART and DBP.

6.1 Comparison with DEPART and speed_up

The results for the first experiment are shown in Table 1. The normalized average results are shown in the last row of the table. The goal of this experiment is twofold: to compare our method with DEPART [17] and with *speed_up* applied to the flat netlist. The experiment has been performed on the same MCNC benchmarks used in [17], using the same pre-processing script (twice *script.rugged* followed by *eliminate -1; speed_up -i*) and the same measurement units for delay and area: levels of 2-input gates and literals.

The results for DEPART are the ones reported in [17] in which the windows were constrained to have 200 nodes at most. We observed that this size was excessive for *speed_up* in the optimization of some windows and, for this reason, we chose a smaller size ($S_d = 50$) for running DBP.

Even using smaller windows, DBP is on average superior to DEPART in delay (0.97 vs. 1.05), while similar in area. Since the networks have moderate size, *speed_up* was also executed on the whole network to compare with the same command applied within a window using the DBP-based partitioning. Surprisingly, the application of *speed_up* to the DBP-windows was superior in delay than applying it to the whole network (3% improvement in delay at the expense 7% increase in area). One of the reasons for that is that the restructuring obtained by *speed_up* depends on the order in which the transformations are performed (e.g. see Fig. 2 in [13]). The dominator-based clusters offer a better guidance for *speed_up* and prevent transformations that can later result in worse delays. Additionally, the runtime time between the flat method and DBP is compared, and a clear reduction for DBP is observed. The runtime for DEPART is not shown, since the CPU and the windows sizes used for the experiments were different.

This experiment gives us the confidence that our partitioning method does not incur in large penalties with regard to the flat method, even on relatively small examples.

6.2 Comparison with hMetis

The results of the second experiment are presented in Table 2. They have been executed on the largest ISCASS'99 benchmarks, selecting only those that were larger than 9000 literals in factored form after applying the pre-processing script. In this experiment, a lighter script is applied as the pre-processing step (*algebraic* script and *speed_up -i*) instead of the *script.rugged,* since don't care calculation is infeasible on large examples.

The experiments were also run using *reduce_depth* [14]. This command was fast on small examples, but ran out of time for large networks. The obtained results were worse than the hMetis-based algorithm and, for this reason, are not shown in the paper.

The comparison with hMetis was done as follows:

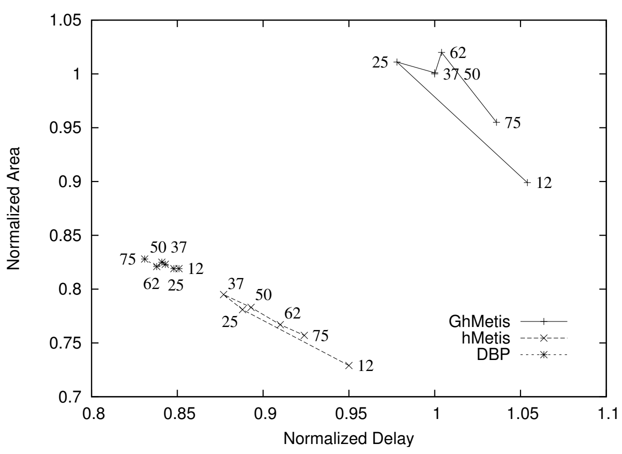

Figure 6: Trade-off area-delay depending on window sizes.

- In the algorithm of Fig. 5, the `DominatorPartition` procedure was substituted by hMetis, with the same size of windows.
- Since hMetis does not guarantee a topologically-ordered partition, a minimum set of feedback edges was calculated and ignored to obtain an close to topological order of the windows [24]. The delay information in the feedback edges was obtained from the one calculated in the previous iteration (since it was not possible to propagate it in topological order).
- Two variations with hMetis were run.
  - The first one using $Slack = \infty$ to capture the whole network in the critical region and to only apply the delay optimization script (labeled GhM in the table).
  - The second one using $Slack = 2$, as in DBP (labeled hM in the table).

  In both cases, the weights of the edges between nodes on the critical region were multiplied by a constant factor to bias hMetis towards avoiding cuts in the critical paths (without weights the results of hMetis partitioning are much worse). With this experiment, we estimated the contribution of running different optimization scripts for the critical and non-critical regions of the network.

Table 2 reports the results of the technology independent optimization after the initialization script (Orig.), and for all three methods. It also reports the results after technology mapping using *map* with the *lib2* library. DBP improves delay by 22% and area by 18% when compared to global hMetis (GhM). After technology mapping the delay improvement is reduced to 16%. Global hMetis produces networks with larger area, since non-critical regions are also optimized for delay. After technology mapping, DBP offers a 5% delay improvement with a cost of 4% in area comparing with hMetis focused on the critical path. DBP creates windows where the delay minimization script performs a better optimization.The runtime of DBP is approximately 2x higher than hMetis, since the delay script spends more time on the minimization of individual windows (that have on average larger depth in case of DBP) and more iterations are executed to reach a network with no further improvements.

The CPU time for partitioning is negligible compared to the time for the logic optimization within the windows.

6.3 Trade-off between area and delay

The objective of this experiment is to evaluate the impact of the window size on the results obtained by the previously described

Bench.	Literals				Levels				Area			Delay			CPU (sec)		
	Orig.	GhM	hM	DBP	Orig.	GhM	hM	DBP	GhM	hM	DBP	GhM	hM	DBP	GhM	hM	DBP
b14	11122	15972	14443	14515	92	44	32	32	11017	9893	9809	44.4	37.7	36.7	167	180	404
b14_1	9424	14701	11871	12393	70	40	30	30	10058	8222	8706	40.5	34.3	33.6	191	117	329
b15	17829	22962	20359	20229	91	63	54	49	14734	12811	12814	61.9	56.4	53.4	298	140	686
b15_1	17116	21422	18766	18829	71	38	31	30	13636	11947	12127	40.5	35.4	35.1	590	157	528
b17	56405	77962	62990	63332	128	62	55	53	50048	38704	38669	68.6	60.3	58.0	4266	617	1613
b17_1	53188	66455	57323	57889	71	37	33	32	41832	35550	36630	43.1	38.0	35.2	2697	415	1274
b20	22727	34644	30969	32634	108	54	37	36	23123	20617	22072	57.3	41.5	40.1	668	349	488
b20_1	19663	36471	27408	26008	105	46	36	36	25101	18425	17512	46.3	40.7	40.5	1111	385	515
b21	23654	37815	30127	31364	109	51	38	36	25859	20040	20988	50.0	41.6	40.6	893	538	523
b21_1	19652	33726	27465	28544	99	46	35	35	23288	18607	19531	48.2	40.0	40.2	751	338	675
b22	34441	59030	39316	44592	101	43	51	38	40543	24946	29228	45.2	51.3	42.0	2246	166	1078
b22_1	29799	54271	34535	40969	102	47	48	36	37291	22201	27111	48.9	47.6	40.1	1711	229	892
s35932	16304	19376	19352	19500	22	10	9	9	13657	12742	12787	14.0	13.6	13.8	267	620	104
s38417	22172	25785	23184	23694	31	25	24	23	17349	15067	15462	26.6	26.7	24.5	546	69	84
s38584	20086	21470	20146	20222	28	22	21	20	13480	12787	12810	22.5	23.2	21.5	71	184	327
Norm		1.00	0.80	0.82		1.00	0.85	0.78	1.00	0.78	0.82	1.00	0.89	0.84	1.00	0.27	0.57

Table 2: Comparison between hMetis and DBP for large networks.

methods (Figure 6). The experiment is conducted using different sizes for S (12, 25, 37, 50, 62, and 75 nodes) and, as a reference point, the method *GhM* with $S = 50$.

The main conclusion is that DBP always obtains better results in delay regardless the size of the windows. Moreover, as the plot shows, the impact of the window size on the results of the DBP is much smaller than the impact on the results of the mincut-based methods.

The mincut-based methods (hMetis and global hMetis) perform better (delay-wise) results for windows with size of 20-40 nodes. For smaller sizes there is a significant degradation, which is easily explained by the restructuring limitations imposed by the size of the window. However, a similar degradation in delay is also observed when the window size grows. We studied this strange phenomenon in more detailed and we observed that the topological order of the clusters was violated more often when the window size grew, i.e. the larger the windows, the smaller the probability to obtain a topologically ordered partition. For this reason, the suboptimal propagation of delay information has a negative influence on the resulting delay.

The CPU time is also affected by the size of the windows. For DBP, the runtime increases with the size of the windows, whereas it decreases for hMetis. The dependency is the opposite with larger windows, hMetis stops on local minima much earlier and executes fewer iterations of the algorithm.

7. CONCLUSIONS AND FUTURE WORK

Scalability is a crucial aspect for the applicability of logic synthesis techniques on large networks. A partitioning technique based on the calculation of dominators has been proposed to tackle the complexity of delay optimization.

In DSM technologies, the combination of logic and physical synthesis seems to be essential to meet the demand of today's designers regarding delay and power optimization. We believe that the proposed partitioning strategy, enhanced with layout information, could be a valid approach for integrating and exploring logic and physical parameters of the design.

8. REFERENCES

[1] C. I. Park and Y. B. Park, "An efficient algorithm for vlsi network partitioning problem using a cost function with balancing factor," *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 12, no. 11, pp. 1686–1694, 1993.

[2] D. I. Cheng, C. C. Lin, and M. Marek-Sadowska, "Circuit partitioning with logic perturbation," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1995, pp. 650–655.

[3] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in vlsi domain," *Proc. ACM/IEEE Design Automation Conference*, 1997.

[4] J. Cong and C. Wu, "Global clustering-based performance-driven circuit partitioning," in *Proc. of the Int. Symp. on Physical Design*, 2002, pp. 149–154.

[5] Y. Cheon and D. F. Wong, "Design hierarchy-guided multilevel circuit partitioning," *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 22, no. 4, pp. 420–427, 2003.

[6] C. Ababei, N. Selvakkumaran, K. Bazargan, and G. Karypis, "Multi-objective circuit partitioning for cutsize and path-based delay minimization," in *ICCAD*, 2002, pp. 181–185.

[7] D. Brasen and G. Saucier, "FPGA partitioning for critical paths," in *EDAC-ETC-EUROASIC*, 1994, pp. 99–103.

[8] C. N. Sze, T. C. Yang, and L. C. Yang, "Multilevel circuit clustering for delay minimization," *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 23, no. 7, pp. 1073–1085, 2004.

[9] K. Singh, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1988, pp. 282–285.

[10] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "On clustering for minimum delay/area," *Proc. ACM/IEEE Design Automation Conference*, pp. 6–9, 1991.

[11] K. C. Chen and S. Muroga, "Timing optimization for multi-level combinational networks," in *Proc. ACM/IEEE Design Automation Conference*, 1990, pp. 361–364.

[12] H. Ajuha and P. R. Menon, "Delay reduction by segment substitution," *Proc. European Conference on Design Automation (EDAC)*, pp. 82–86, 1994.

[13] J. Cortadella, "Timing-driven logic bi-decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 675–685, June 2003.

[14] H. Touati, H. Savoj, and R. Brayton, "Delay optimization of combinational circuits by clustering and partial collapsing," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1991, pp. 188–191.

[15] R. Rajaraman and D. F. Wong, "Optimal clustering for delay minimization," *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 14, no. 12, pp. 1490–1405, 1995.

[16] H. Yang and D. Wong, "Circuit clustering for delay minimization under area and pin constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 9, pp. 976–986, 1997.

[17] R. Aggarwal, R. Murgai, and M. Fujita, "Speeding up technology-independent timing optimization by network partitioning," *Proc. ACM/IEEE Design Automation Conference*, pp. 83–90, Nov. 1997.

[18] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Transsactions of Programing Languages and Systems*, vol. 1, no. 1, pp. 121–141, 1979.

[19] E. Dubrova, M. Teslenko, and A. Martinelli, "On relation between non-disjoint decomposition and multiple-vertex dominators," in *Proc. International Symposium on Circuits and Systems*, 2004, pp. 493–496.

[20] W. A. Havanki, S. Banerjia, and T. M. Conte, "Treegion scheduling for wide issue processors," in *Proc. Intl. Symp. on High-Performance Computer Architecture*, 1998, pp. 266–276.

[21] M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana, "Fault equivalence identification using redundancy information and static and dynamic extraction," *Proc. of the Asian Test Symposium*, pp. 124–130, Apr. 2001.

[22] M. Teslenko and E. Dubrova, "An efficient algorithm for finding double-vertex dominators in circuit graphs," in *Proc. Design, Automation and Test in Europe (DATE)*, 2005, pp. 406–411.

[23] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," U.C. Berkeley, Tech. Rep., May 1992.

[24] P. Eades, X. Lin, and W. F. Smyth, "A fast and effective heuristic for the feedback arc set problem," *Information Processing Letters*, vol. 47, no. 6, pp. 319–323, 1993.