Constructive Logic and Layout Synthesis: A Novel Incremental Approach

Yoonna Oh[†] **and Yuncheol Baek**[††],

CAE Team, Semiconductor R&D Center, Samsung Electronics, Hwasung 445-701, Korea[†]
Division of Computer Software, Sangmyung University, Seoul 110-743, Korea[††]

Summary
This paper examines the extension of constructive library-aware logic synthesis to the physical placement stage of integrated circuit design. Constructive logic synthesis differs from traditional synthesis approaches in that it builds a circuit netlist incrementally starting from the primary inputs and proceeding towards the primary outputs. In each iteration of this procedure, the semantic structure of the unsynthesized logic functions is utilized to identify and extract a small subcircuit that consists of library primitives reflecting that structure. The algorithm interleaves the steps of technology-independent decomposition and technology-dependent mapping into library cells in a way that mitigates its greedy nature. Conjecturing that adding a placement step to this methodology would further improve synthesis quality we developed a system which synthesizes circuits by incremental decomposition, mapping, and placement. We describe the algorithms used in this system and analyze the quality of the designs it generates under a variety of options for decomposition and placement. The empirical results we obtained, however, suggest that adding a placement step to constructive synthesis produces no noticeable improvement in design quality. This strongly suggests that our original conjecture is false, and we examine possible reasons for such a negative result.

Key words:
Logic synthesis, integrated circuit design, constructive library-aware logic decomposition

1. Introduction

In the traditional design flow of Very Large Scale Integration (VLSI) chips, the design process is serialized into a sequence of manageable steps of high level synthesis, logic synthesis, technology mapping, physical placement, global routing, and detailed routing. Over the past decade, the amount of integration onto a single chip, especially with the need for interconnecting sub-circuits to support networking infrastructure, has rapidly increased. Although Deep Submicron (DSM) technologies enable greater degrees of semiconductor integration, such integration makes the design, verification, and test more challenging. While gates are getting smaller and faster, designs are more constrained by the delays of the wires interconnecting the gates. The routability of a circuit after physical placement is another increasing concern. These issues often cause each design step to be iterated many times before all constraints are met.

Each design step involves optimizing some objective subject to certain constraints such as area, delay, and power consumption. In a serialized design flow, constraints imposed in earlier steps are often hard to satisfy in later steps. Repetition of the design steps does not necessarily guarantee that the constraints will be met, and there are often very limited chances to fix the problem in the later steps of a design in the serialized design paradigm.

In the 1990s, the increasing need for synthesizing larger blocks posed new challenges with the designs more constrained by the delays of the wires interconnecting the gates. There have been several related attempts to overcome these challenges by combining logic and layout synthesis. These efforts can be broadly divided into local netlist transformation, pre-layout prediction, and layout-driven optimization.

Local netlist transformation. In [15] [7] [17], local transformation approach is applied to a placed netlist after initial placement. Critical factors such as critical paths or congested regions of the circuit based on the initial placement are identified, then re-synthesized and re-placed. Liu et al. [15] proposed a technique that combines logic re-synthesis and linear placement in order to alleviate routing congestion in the most congested region of the routing area in order to reduce routing density, and accordingly shrink the chip height. Kannan et al. [7] begin with a synthesized netlist after initial placement and make incremental modifications to generate a final netlist and placement that meets the timing constraints after place-and-route. This is done by applying fanout buffering and gate resizing. The gate and wiring delays based on the initial placement are computed and used for selecting the most useful transformations. Lou et al. [17] iterate grouping cells along the critical paths, re-synthesizing and re-placing those cells until timing closure is achieved. These approaches work

Manuscript received October 5, 2007
Manuscript revised October 20, 2007

on accurate net length or congestion estimates and apply local transformations to preserve the initial placement as much as possible; this, however, tends to restrict the optimization potential.

Pre-layout prediction. Pre-layout prediction approaches such as [9] [12] [16] [26] do not employ an initial placement solution. Rather, logic optimizations are guided by heuristics such as distance [9], adhesion [12], mutual contraction [16], and fanout range [26], which are used to capture the routability of designs. These heuristics are applied during logic decomposition to obtain more routable designs with no knowledge about actual placement.

Layout-driven optimization. The majority of combining logic and layout synthesis is layout-driven optimization as appeared in [2] [4] [3] [13] [14] [20] [21] [22]. In these, they first create a companion placement of the technology-independent netlist for a given logic circuit and then use physical coordinate information to guide the logic synthesis process. This companion placement can be incrementally updated during logic synthesis. This method was initiated by Pedram and Bhat in their papers [20] [21] on improving timing closure by taking interconnect delays into account. Their approach is based on a point placement of a Boolean network. The placement solution is incrementally updated as intermediate Boolean nodes are extracted or eliminated during the decomposition and elimination procedures. Kutzschebauch and Stok [14] proposed congestion aware algorithms for layout driven decomposition and technology mapping to decrease wire length and improve congestion. In the decomposition algorithm, they include wiring delays based on estimated net lengths and geographical locations of the input signals in the companion placement. In their technology mapping step, the final implementation is chosen by identifying the fastest driver pin of each net. They choose the technology-dependent implementation as the minimum cost function subject to a timing constraint, thus they can reduce total wire length resulting in less global congestion. Another variation of this approach by Gosti [3] observes that the wire-load models in conventional logic synthesis underestimate the wire length of many nets, especially nets connecting a few pins. To address the timing closure problem, the technology decomposition and global placement steps are performed repeatedly. Placement is done using a quadratic programming placement tool, which places nets incrementally and spreads overlapping cells apart from the center of the placement area. However, the final placement of the layout-driven methods ends up being considerably different from the initial one, casting doubt on the accuracy of wire length estimates based on it. Indeed, these approaches did not exhibit much improvement in implementation quality beyond that achievable by logic synthesis followed by layout synthesis. Against these drawbacks, we embarked on yet another attempt to combine logic and layout synthesis. The *Constructive synthesis* approach, pioneered in [10] [8] [11], interleaves the technology-independent logic decomposition and technology mapping steps to more directly relate *functional* structure of a logic specification to the ultimate *physical* structure of its implementations. The method employs full Boolean decomposition and takes functional symmetries into account decomposition to tackle the problem of serialization in the circuit design process. For small synthesis problems, this approach produced superior netlist implementations with acceptable run times. More recently, Mishchenko *et al*. [19] proposed a new enhanced constructive algorithm which pre-computes the gate library in the form of *supergates*, and considers all support-reducing decompositions for a set of variables.

Noting the success of constructive library-aware logic synthesis in significantly improving the quality of netlist implementations, we conjecture that adding an incremental placement phase can only yield further improvement. The principal difference between our approach and previous approaches is that our synthesis flow is *constructive* and *incremental*. By interleaving functional decomposition and technology mapping, the constructive synthesis algorithm is able to uncover the natural "structure" of the logic functions being synthesized yielding much better netlist implementations than is possible by serializing these two phases. We reason that incrementally placing the gates as they are extracted from the yet-to-be decomposed logic would provide more accurate area and delay estimates that can inform future decomposition choices. Intuitively, we hypothesize such an enhancement can improve implementation quality further.

Figure 1 shows the design flow of the traditional approach and our constructive approach. Compared to Gosti's work [3], our design flow achieves a much tighter coupling between the logic synthesis and placement steps, and is aware of the technology library at the decomposition step. However, the implementation quality by this approach did not surpass that of traditional approach. This paper, thus, describes our attempt for testing this conjecture, and discusses the experimental results we obtained.

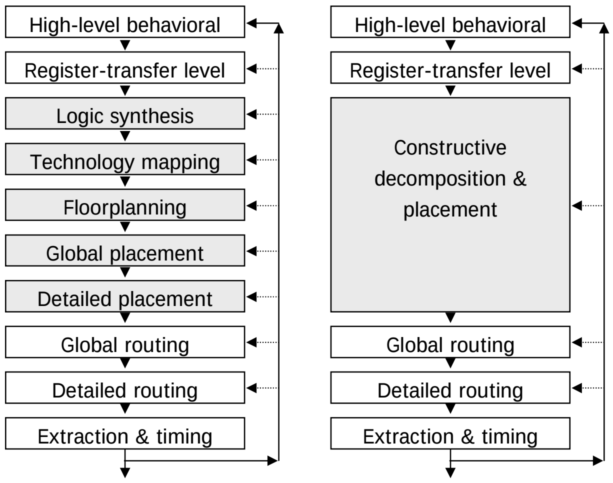

Fig. 1. Traditional design flow (left) vs. constructive design flow (right)

The remainder of this paper is organized as follows. Section 2 describes notations used in this paper. Section 3 describes the concept of the *constructive logic and layout synthesis* design approach, which extends the idea of *constructive logic synthesis* to the physical layout phase. This section describes a prototype implementation of this approach with a simple placement algorithm and delay model. In section 4, we explore the search space of the constructive logic and layout synthesis approach by trying another placement algorithm and delay model. In section 5, we examine possible reasons for the negative result and section 6 concludes this paper with its contributions and new directions for future work.

2. Notation and Preliminaries

An n -input *Boolean function* $f(x_1,\ldots,x_n)$ or $f(X)$ is defined as a mapping $f: B^n \to B$ where B is commonly restricted to the two-element set $B = \{0,1\}$. Each element in the domain B^n is a *minterm* of f. *Support*, whose size is denoted by $|X|$ or *n*, is the set of variables to the function f. A *multiple-output function* is defined as a vector of functions $F: B^n \to B^m$.
Multilevel synthesis can be viewed as a process of successive decompositions. Any Boolean function f can be expressed by the following decomposition template:

$$f(X) = h(g_1(X),\ldots,g_k(X))$$

where $g_1,\ldots,g_k$ and h are referred to the *decomposition functions*, and the *composition function*, respectively. This decomposition template encompasses all types of function decomposition. In particular, the *Ashenhurst decomposition* expresses the function f by the following template:

$$f(X) = h(g_1(X_1),\ldots,g_k(X_1),X_2)$$

where X_1 and X_2 are called the *bound set* and *free set*, respectively.
In particular, *support-reducing decomposition* [8] requires the composition function h to depend on fewer variables than the original function $f(X)$. Support-reducing decomposition is based on the decomposition template along with the restriction $k < s$ where $s = |X_1|$. Support-reducing decomposition is not necessarily a *disjoint decomposition* in which the two subsets of the support X_1 and X_2 are disjoint. This is because y_i, where $1 \le i \le k$, can be an identity function. In this paper, our scope is restricted to finding a support-reducing decomposition among many choices.

3. Constructive Logic and Layout Synthesis Approach

This section describes the constructive logic and layout synthesis algorithm and shows the experimental results of designs synthesized by this algorithm.

3.1 Motivating Example

Consider a small circuit of 12-bit parity to illustrate the idea of constructive logic and layout synthesis. The circuit is incrementally synthesized by repeatedly applying the steps of logic synthesis, technology mapping, and physical placement in a single step with 3-bit XOR and 2-bit XOR library primitives as its building blocks. When the circuit is decomposed in the constructive design flow, two types of information are available; logical and physical. *Logical* or *non-physical* type indicates information available without physical layout. This type of information includes functional symmetry, BDD size, logic level, area, and gate delay. *Physical* type information is provided from the physical placement, which includes wire-length, interconnection delay, I/O pin location, and cell coordinates.
Figure 2 shows the result of constructive synthesis of this circuit only using logical information. In this case, functional symmetry and logic level are used to select the support in performing support-reducing decomposition. Note that all variables are functionally symmetric in the parity circuit, and that any three variables chosen as

decomposition support are mapped to a 3-bit XOR gate. If there are more than one candidates evaluated as the highest in logical information, a “random” choice will be made. In the final circuit placed, there are 20 cells including primary input/output pins, and 19 nets.

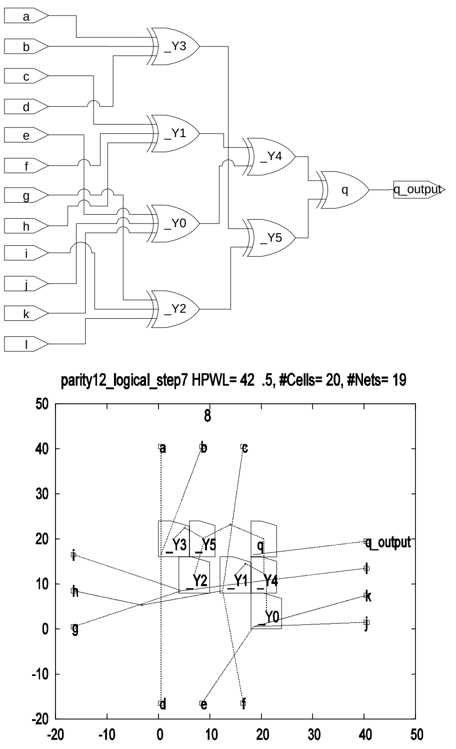

Fig. 2. Mapped (above) and placed netlist (below) of 12-bit parity circuit by constructive synthesis using logical information

In contrast to Figure 2, Figure 3 shows the result of constructive synthesis of the same circuit using physical information as well as logical information to select the support in decomposition. When a set of variables is to be selected for the support and there exist more than one candidate evaluated the highest in the logical information, cell coordinates are used to break the ties. Among the candidates, we select a group of cells that are the most closely located, assuming that origins of each cell are used for pin locations and cell coordinates as well. Initially, all inputs are functionally symmetric and have the same number of logic levels. Among those inputs, four groups of inputs are the most closely located to each other within the group; {*a, b, c*}, {*d, e, f*}, {*g, h, i*}, and {*j, k, l*}. By utilizing cell coordinates at each selection, we obtain the final circuit having 20 cells and 19 nets, which are the same numbers as in Figure 2.

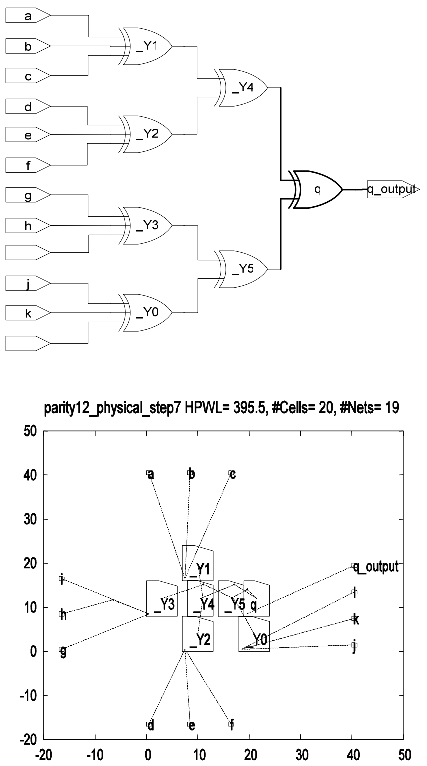

Fig. 3. Mapped (above) and placed netlist (below) of 12-bit parity circuit by constructive synthesis using physical information

However, if we compare half-perimeter wire-length (HPWL), which is the sum of half the perimeter of all bounding boxes of each net, to estimate the length of all interconnects, we notice that HPWL of Figure 3.8 is 428.5 and HPWL of Figure 3.15 is 395.5, which shows an improvement of 8%.

We apply the same technique to 32-bit and 64-bit parity circuits. In Figure 4, HPWL of the 32-bit parity circuit placed using logical information is 1282 whereas that using physical information is 1016, which has been improved by 21%. Moreover, HPWL of the 64-bit circuit placed using physical information, shown in Figure 5, has been improved by 38% over the result using logical information only.

From this observation, we conjecture that information gathered from the physical placement during constructive synthesis would be more beneficial to the final placement as the circuit size grows bigger. However, logic decomposition in the parity case was not a true meaning of decomposition, but a simple replacement with XOR gates.

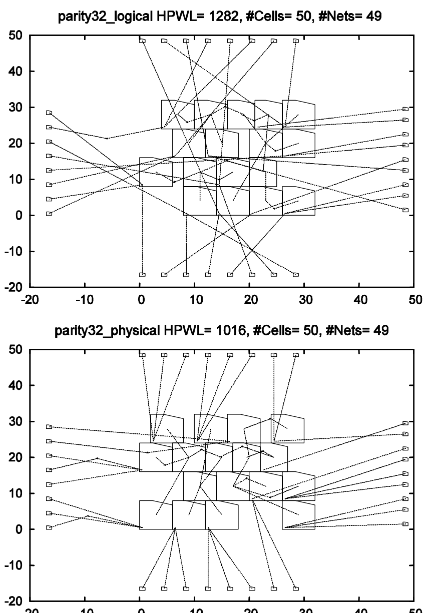

Fig. 4. Constructive synthesis of 32-bit parity using logical information (above) vs. using physical information (below)

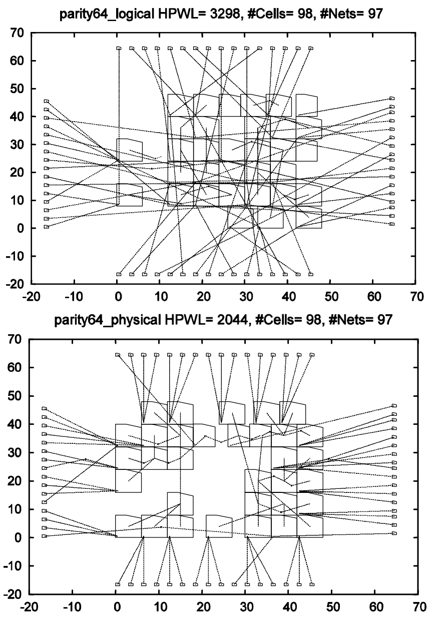

Fig. 5. Constructive synthesis of 64-bit parity using logical information (above) vs. using physical information (below)

3.2 Constructive Logic and Layout Synthesis Algorithm

In order to generalize this idea and test it, we modify the constructive logic decomposition algorithm as follows. Our objective is to minimize the delay of the critical path in the final mapped and placed netlist, including the interconnect delay. We assume the Elmore delay model [24] for the placed network. Our logic synthesis and technology mapping algorithm extends the constructive synthesis algorithm M31 [10] with the enhanced features of CDM [19] and bi-decomposition [18], a complete decomposition template. In our constructive synthesis and placement algorithm called 'COLOSSEUM', a circuit is synthesized incrementally in a single sweep by repeatedly applying the following steps:

Fig. 6. Constructive logic and layout synthesis flow (Appendix 1)

1) **Node selection**: choose an unimplemented logic node to decompose (F_1 in Figure 6(a))
2) **Support selection**: choose a set of fanin variables from the selected node's support (three input pins shown in bold in Figure 6(a))
3) **Decomposition**: choose a set of technology-dependent primitives (library cells) and introduce them as gates into the evolving network (NAND and XOR logic gates shown in Figure 6(b))
4) **Physical placement**: place the primitives chosen in step 3 in the layout plane and compute their coordinates (Two rectangles NAND and XOR2 in Figure 6(b))
5) **Logic re-expression**: re-express the forward unimplemented logic in terms of the newly-introduced gates (F_1 with fanins of logic gates NAND and XOR in Figure 6(b))

The synthesis algorithm terminates when the network does not contain any unimplemented nodes. The final mapped and placed cells are shown in Figure 6(c). As a preprocessing step, we first compute supergates [19] and sort them according to the criteria sought, delay in this case. Initially, primary input pins are assumed to be located on the left border of the layout and primary output pins are assumed to be located on the right. Starting from an empty network and an empty placement, a node to decompose is chosen. Among the inputs of the chosen node, a set of inputs is selected for decomposition based on heuristics considering layout information. As the synthesis proceeds, actual physical information can be utilized. Different heuristics can be used in this support selection step, such as selecting inputs with minimum estimated delay or selecting inputs with the smallest half-perimeter bounding-box. Once the support is selected, the sorted supergate table is incrementally scanned to find a support-reducing decomposition. If such decomposition is not found, a bi-decomposition solution is attempted. In either case, the decomposed node is mapped to library cells and the optimal bounding box of each library cell is computed. Thereafter the cell location is legalized by using the simple Tetris algorithm [5], i.e., scanning leftmost legal site of each row from top to bottom and selecting a site with a minimum cost. Once the cells are placed, the logic of the chosen node is re-expressed in terms of the new nodes and the netlist is updated accordingly. In steps 1 and 2, physical as well as logical information can be used to guide the selection. Those heuristics are discussed in Section 3.5. The pseudo-code of the constructive design flow is shown in Figure 7. The layout information used is shown in bold in the pseudo-code.

```
Colosseum (function F, library { $f_i$ }) {
    N = empty network;
    P = empty placement;
    while (F $\notin$ { $f_i$ }){
       function f = Select_Node(F, P);
       X = Select_BoundSet(f, P);
      { $g_i$ } = DecompositionFtn(f, X, { $f_i$ });
      if({ $g_i$ } == $\varnothing$ )
         { $g_i$ } = BiDecompose(f, X, { $f_i$ });
      Place_Node({ $g_i$ }, P);
      h = CompositionFtn(f, X, { $g_i$ });
      AddToNetwork(N, { $g_i$ });
      f = h;
   }
   AddToNetwork(N, f);
   return N;
}
Place_Node({ $g_i$ }, P) {
          { $C_i$ }= Add_NewNode({ $g_i$ }, P);
Compute_Optimal_BoundingBox({ $C_i$ }, P);
Legalize_Node({ $C_i$ }, P);
}
```

Fig. 7. Constructive logic and layout synthesis algorithm

3.3 Delay Model

We model the delay of a path as the sum of the gate delays and the sum of the net delays on the path. Gate delays account for the "fixed" portion of a gate's delay, excluding the effect of fanout loads. Fanout contribution to delay is captured by the net delays using the Elmore delay model [24].

As the actual wiring paths are not known at the time of decomposition and placement (the algorithm does not perform detailed routing), a net model is needed to model the topology of the interconnection nets and to estimate the net delays. To estimate the optimal wiring paths for a net, rectilinear Steiner trees can be used. Considering the NP-completeness of the Steiner tree problem, we use, instead, the simpler-to-compute half-perimeter wire-length (HPWL):

$$HPWL = \sum_{\text{all nets}} (x\text{-span} + y\text{-span}) \text{, where}$$

$$x\text{-span} = \max(x_1, \ldots, x_n) - \min(x_1, \ldots, x_n)$$

HPWL is known to estimate the minimum Steiner tree exactly for 2- and 3-pin nets, and is considered a reasonable estimate for 4-pin nets. From the net model, a corresponding electrical model can be derived and used for net delay estimation.

The electrical model consists of the output resistance of the source R_s, the output capacitance of the source C_s, the branch resistance R_{l_k} of the destination sink *k*, a net capacitance C_{net}, a load capacitance C_{L_i} of each sink *i*. Let l_x and l_y denote the lengths of the *x*-span and *y*-span, and let l_{x_k} and l_{y_k} be the distances between the source and the destination sink in the *x*- and *y*- directions, respectively. Then the resistance R_{l_k} and capacitance C_{l_k} of the net are computed as follows:

$$R_{l_k} = r_x \cdot l_{x_k} + r_y \cdot l_{y_k}$$

$$C_{l_k} = c_x \cdot l_{x_k} + c_y \cdot l_{y_k}$$

where r_x, r_y, c_x, and c_y denote the resistances and capacitances per unit length in the *x*- and *y*- directions, respectively. The net capacitance is defined as

$$C_{\text{net}} = c_x \cdot l_x + c_y \cdot l_y$$

and the Elmore net delay from the source pin to the sink pin *k* is computed from:

$$D_{\text{net}} = R_S \cdot (C_S + C_{\text{net}} + \sum\nolimits_{\text{all sinks}} C_{L_i}) + R_k \cdot (C_{l_k} + C_{L_k})$$

During decomposition and placement, we maintain signal arrival times at the circuit nodes. An arrival time of zero is assigned to all primary inputs and the arrival times of all other pins are computed by traversing the network starting from the primary inputs towards the primary outputs recursively. The network is not complete during synthesis, however, and we need to approximate the interconnection and node delays of intermediate nodes. During constructive decomposition and placement, each node generated up to the point is dealt as the final netlist and the delay is computed each time in a greedy manner. Although this delay might not be accurate in the final netlist, this heuristic is chosen as to avoid selecting a critical part as a decomposition node or a decomposition support.

3.4 Constructive Placement Algorithm

In the constructive decomposition and placement flow, a cell that needs to be placed as a new node is created and a corresponding library cell is mapped. Our constructive placement algorithm consists of two phases: *optimal location computation* and *legalization*.

Optimal Location Computation. After a new node is extracted and mapped to a library cell, an optimal location of the mapped cell is computed for placement. This location may overlap other previously-placed cells and must be subsequently legalized during detailed placement. We are given a single movable cell which is newly mapped and a set of fixed cells connected to the movable cell. We seek a placement of the movable cell such that the bounding-box half-perimeter of all signal nets is minimized. This essentially reduces to two independent one-dimensional optimization problems that seek to minimize a) the total *x*-span, and b) the total *y*-span of all nets. All *x*-coordinates of cells must be placed on integer sites and *y*-coordinates must be placed on integer rows. The position of a cell C_i is defined as the *x*-coordinate of the left edge of the cell and the *y*-coordinate of the bottom edge of the cell. Hereafter we will consider *x*-coordinates only. We assume all pin positions to be the same as the cell position. We use the following notations:

- The position of the leftmost (respectively, rightmost) fixed pin of net *N* is denoted by $L(N)$ (respectively, $R(N)$).
- The position of the movable pin of net *N* is denoted by $M(N)$.

The cell cost function $\text{cost}_i(x)$ of a movable cell for a given position is

$$\text{cost}_i(x) = \sum_{j=1}^{m} \max(M(N_j) - R(N_j), 0) + \sum_{j=1}^{m} \max(L(N_j) - M(N_j), 0)$$

We compute the optimal bounding-box that minimizes the cost function as follows. Create a sorted list of $L(N)$ and $R(N)$ of all *m* nets. Since the cell cost function is piecewise linear and convex [6], the optimal location is the interval of the two middle points in the sorted list. By computing the interval for *y*-coordinates, we obtain an optimal bounding-box with minimum cost. This bounding-box is not always valid, thus need to be legalized.

Legalization. The global placer described in above attempts to improve the quality of the integrated circuit design by minimizing the wire lengths between connected cells. However, the initial bounding-box location computed is not always a legal location. In the actual row-based physical device, cells must be assigned to locations that align within a grid of discrete *x* (called *site*) and *y* (called *row*) coordinates without any cell overlaps. In general, the detailed placer slightly degrades wire length in an effort to find legal cell placements. We have modified Hill's Tetris algorithm [5], known to be simple and fast, for the detailed placement.

Our detailed placer works as follows. Cells have the same height but have variable width. When the detailed placer is invoked, all cells except one movable cell are already placed in legal locations. A movable cell is assigned its initial optimal bounding box coordinates. For a given cell, placement is performed by scanning through the rows of the substrate and selecting the leftmost vacant site of each row as a candidate site for placement of the given cell. A site that does not contain a previously placed cell is called *vacant*. A group of candidate sites is determined because each row returns a candidate site. A site that is used as the leftmost *x*-coordinate of the current cell is called the *left factor* and a penalty will be paid if the leftmost vacant site of a row is to the left of the left factor. The rightmost *x*-coordinate of the current cell fanins can be used as the cell left factor. Some candidate sites have high cost if they are located to the left of the cell left factor. Other sites might be invalid if they are too far to the left of the optimal location. Among the candidate sites, a site with the least cost is selected for the cell location. The cost function is defined as the Manhattan distance between the optimal bounding box and the candidate site in each row, augmented by left factor penalty if any. Figure 8 summarizes the pseudo-code of the legalization algorithm.

```
Legalize (optimal_BBox, new_Cell){
   lowestCost = MAX_POSSIBLE;

   for (row = bottom; row < top; row++){ // scan rows
       candidate_site = row.leftmost_site();
      if(candidate_site + new_Cell.width() >
         row.rightmost_site())
         continue;                 // cannot fit here
      currentCost = optimal_BBox.manhattan_distTo(row,
              candidate_site) + new_Cell.left_factor();

      if(currentCost < lowestCost){
           lowestCost = currentCost;
           new_Cell.set_coordinate(row,candidate_Site);
      }
   }
   if(lowestCost == MAX_POSSIBLE)
      not_enough_space();
   return;
}
```

Fig. 8. Legalization algorithm

3.5 Experimental results

The algorithm described in previous section has been implemented in C++ using the BDD package CUDD [25] and sub-modules of the placement package Capo 8.8 [1]. We conducted our experiments on a 2.2 GHz 2x AMD Opteron™ 248 machine with 8GB of RAM running the Linux operating system. We modified the mcnc.genlib library with parameters of 0.13 micron technologies similar to the TSMC 0.13 technology library. The evaluation was performed on the MCNC benchmarks [27]. In all experiments, the right-most cell boundary coordinates of the fanins are used as the cell left factor.

Table 1 Node selection heuristics (Appendix 2)

In the first experiment, shown in Table 1, we applied different node selection heuristics to a set of MCNC benchmarks. The purpose of this experiment was to quantify the relative benefits of using physical as well as functional information in the node selection heuristics during logic decomposition. Among all candidate nodes, the node chosen next for decomposition was determined according to the following six heuristics:

- **Most Complex ("MC"):** the node with the largest BDD of its forward (unimplemented) logic
- **Least Complex ("LC"):** the node with the smallest BDD of its forward logic
- **Smallest Bounding Box ("BB"):** the node with the smallest bounding box of its fanins in the layout
- **Least Delay ("DE"):** the node whose fanins have the least delay (recall that delay is computed with the partial netlist as described in Section 3.4.)
- **Least Logic Level ("LL"):** the node with the least (lowest) logic level
- **Random ("RD")**: the node selected randomly

Using the "MC" heuristic as a baseline, the columns in Table 1 show the ratios of area, delay, and HPWL for each of the remaining five heuristics compared to "MC". Note that "MC", "LC", and "LL" use logical information only, whereas "BB" and "DE" utilize physical information. The "RD" heuristic selects a random node among the candidates.

Examination of the average improvement over the "MC" heuristic (last row of Table 1) we note no particular advantage to the use of physical instead of logical information in choosing the next node to decompose. Furthermore, we note that a *random* selection seems to be no worse than a selection based on logical or physical information.

Table 2 Area using different support selection heuristics (Appendix 2)

In the second experiment, shown in Table 2 through Table 4, we applied different support selection heuristics to the same MCNC benchmarks. In the proposed algorithm, once a node is selected for decomposition, a subset of its input variables should be selected as the decomposition support. This experiment compared the relative benefits of using different support selection heuristics utilizing physical and/or logical information. The shaded cells in these tables indicate that bi-decomposition had to be invoked after failing to find a support-reducing decomposition. In all tables, the column labeled "baseline" computes the symmetry group of the selected node and selects as many variables as the support size from the largest symmetry group. The column labeled "base+" shows the "improvement" over the baseline when the support is chosen from closely-located pins within the same symmetry group or among many symmetry groups of smaller sizes. The "mdv" heuristic selects the minimum delay variables from the fanin list without utilizing logical information. Those three heuristics take computation time $O(n)$ where *n* is the number of primary input variables.

Since we did not find any benefit of utilizing physical information over purely logical information from the experimental results, we attempted a rather expensive computation by trying all possible combinations of sizes 3 and 4 among fanins. In heuristics D through LGH, we try to compute information from all combinations of the fanin list and select the combination with the best result. We use the following notations for logic or physical information used to guide the support selection:

Table 3 Delay using different support selection heuristics (Appendix 2)

- **"L":** the smallest logic level of the input combination
- **"B":** the number of broken symmetry groups of the fanins. A broken symmetry group is a symmetry group from which some members are selected and some members are not. For example, suppose the symmetry groups computed are $G_1 = \{x_1, x_2, x_3\}$, $G_2 = \{x_4, x_5\}$, $G_3 = \{x_6\}$, and $G_4 = \{x_7\}$. If x_1, x_2, x_4, x_5 are selected as the support, G_1 is broken, but G_2 is not. In this case, *B* = 1.
- **"G":** the number of symmetry groups selected. In the previous example, *G* = 2.
- **"D":** delay of the primitive cell to be mapped
- **"H":** bounding box half-perimeter of the input combination
- **"T":** ready time of the input signal including interconnect delay

Table 4 HPWL using different support selection heuristics (Appendix 2)

Note that the "L", "B", "G", and "D" heuristics are based on purely logical information whereas the "H" and "T" heuristics are based on physical information. Heuristics named with more than one letter use the heuristic corresponding to the first letter as their main selection criterion, and those of subsequent letters to break ties. In the tables, column headings are shaded to indicate that physical information is utilized. The last column in each table shows the result of selecting the support randomly and is much worse than any of the other heuristics.

In this experiment, we do find some cases in which the delay is reduced by utilizing physical information. For example, in the case of benchmark x4 heuristic “BH” helped achieve a delay that is 85% of the baseline delay. However, heuristic “LB” which is based on purely functional information was able to achieve a similar result (87% of baseline). We conclude, thus, that the use of physical information in choosing the support for decomposition is not superior to the use of purely logical information in this setting of experiment.

4. Exploring the Search Space of Constructive Approach

In previous section, we observed that our choice of delay model and placement algorithm did not guide our constructive flow appropriately so as to yield more “natural” and “efficient” implementations of the function. We explore the search space of our constructive logic and layout synthesis approach by applying different delay models and different placement methods in this section.

4.1 Delay Models

The constructive logic and layout synthesis flow is independent of the choice of delay model. The delay model described in section 3.3 was simple to compute; however, the resulting delay estimations can become quite inaccurate for placements where wire delays are dominant compared with gate delays and for nets with large numbers of pins. In this chapter, we use the Elmore delay based on the star model proposed for analytical timing-driven placement [23]. The main advantage of this model is that it enables the calculation of individual delays between the source pin and each sink pin of a net.
To estimate the circuit’s timing behavior, arrival times of all primary inputs are assigned to 0 and those of all other pins are computed by traversing the netlist staring from the primary inputs towards the primary outputs in a breadth-first-search manner. Tracing back from the primary outputs with the largest arrival times to the primary inputs, the critical paths of the circuit are identified. This star model is assumed for the experimental work in this section.

4.2 Placement Algorithms

Our initial attempt at obtaining a legalized placement uses the method patented by Hill [5]. This method is a fast and simple greedy approach, but may produce results that are far from those produced by state-of-the-art standard cell placement algorithms such as simulated annealing, analytic methods, or partitioning-based placement. If interconnect delays dominate gate delays, slight modifications in the arrangement of the cells can cause large changes in the overall performance of the resulting circuit. In order to test the influence of different placement algorithms on logic decompositions in the constructive design flow, we employ in this chapter the recent placement technique *Capo* [1] which is based on recursive multi-level partitioning.
Consider the small 13-input circuit generated by extracting the output named “v” from the MCNC benchmark “cu.blif”. Figure 9 shows five different layouts for this circuit obtained as follows:

(a) “Baseline”: a layout generated by our constructive flow using the “Tetris” placement algorithm without placement information
(b) “Baseline & Capo”: a re-placed layout generated by *Capo* on the result of “Baseline”
(c) “Base+”: a layout generated by the constructive flow and the “Tetris” placement algorithm with placement information
(d) “Base+ & Capo”: a re-placed layout using *Capo* on the result of “Base+”.
(e) “each-Capo”: a layout generated by the constructive flow utilizing the “base+” heuristic and by running *Capo* after each cell created

Fig. 9 Different layouts obtained with the constructive flow and different placement options: (a) Baseline (b) Baseline & Capo (c) Base+ (d) Base+ & Capo (e) each-Capo (Appendix 1)

The decomposition support selection heuristic “base+” discussed in section 3.5 is used in (c), (d), and (e). This heuristic is similar to the *Candidate cube divisor selection method* proposed by Kutzschebauch and Stok [14]. As we have seen in the parity circuit example in section 3.1, this method selects closely-located pins within the same symmetry group or among many symmetry groups of smaller sizes.
Table 5 shows the results of applying the different placement methods for circuit “v”. In this table, option (b) shows improvements in terms of delay and HPWL compared to option (5 to (a) by the utilizing physical information during decomposition. Option (d) shows improvement compared to (c) in terms of delay, but not in HPWL, and option (e) shows the best result in terms of area and HPWL; however its delay is worse than that of (b). This is due to the fact that we have added more than 80% of white space for illustration purposes, and (b) results in more library cells but shorter interconnections among them. From this example we can see that different placement methods might result in different logic decompositions which are different in area, delay, and/or HPWL.

Table 5 Results of applying different placement algorithms for circuit “v”

	#cells	area	delay	HPWL
(a) Baseline	23	180	2.90	666
(b) Baseline & Capo	23	180	1.99	553
(c) Base+	17	129	2.49	554
(d) Base+ & Capo	17	129	2.61	498
(e) each-Capo	15	124	2.38	496
(e) each-Capo	15	124	2.38	496

4.3 Experimental Results

We implemented the Elmore delay based on the star model within the framework of COLOSSEUM. We also integrated a version of *Capo* with COLOSSEUM for placement. Other experimental setups are same as is section 3.
In this experiment, shown in Table 6, we applied different placement algorithms along with different support selection heuristics as described in section 4.2. Using option (a) as a baseline, the columns in Table 6 show the ratios of area, delay, and HPWL for each of the remaining four placement options compared to (a). Note that options (a) and (b) use logical information only, whereas options (c), (d), and (e) utilize physical information (i.e. actual locations of already-placed cells). Since options (b) and (d) use *Capo* to re-place the netlists generated by options (a) and (c), respectively, the area does not change after the re-placement. The last row shows the averages of the ratios compared to the baseline option (a). Examining these ratios, we note that HPWL improved by 14% with the application of *Capo* in options (b), (d), and (e). However, there was no improvement in delay when physical information was utilized during logic decomposition; the small enhancement in the average delay is mostly likely due to the improvement in HPWL.

Table 6 Results of applying different placement options (Appendix 2)

In this experiment we use different delay models and different placement options in order to test the effectiveness of utilizing physical information in guiding logic decomposition. The goal of this investigation was to gain a deeper understanding of the constructive logic and layout synthesis flow, in order to better explain the observed insensitivity of its logic decomposition procedure to the physical information created but its layout procedure. Based on the experiments reported in this section, we can state that; under the constructive synthesis paradigm, the choice of node and support is always “local” and “greedy” with the expectation that local optimization would guide the process towards a global minimum solution. Obviously, this is not guaranteed for general combinational optimization, and seems to be almost always ineffective when layout is interleaved with logic decomposition. A plausible explanation for this outcome is that placement is an inherently global optimization problem which we restrict in the constructive flow to operate more “locally” by placing the partial netlist of implemented nodes, completely ignoring the possibly huge impact of the cells to be extracted form the unimplemented logic. The constructive approach, thus poses on inherent mismatch between the requirements of constructive decomposition and those of global placement.

5 . Further Investigation into the Results

In previous sections, we have examined that physical information of the placed library primitives did not improve the design quality of logic synthesis under the constructive synthesis flow. This outcome is contrary to our intuition that adding physical information would improve the design quality of constructive synthesis. In this section, we inspect reasons of this result more in detail.

5.1 Possible logic decompositions Experimental Results

In this section, we first raise a question and answer to it in the remaining of the section; are there no alternatives in logic decomposition of a selected node at all? Before answering to this question, we limit the scope of our concern to constructive synthesis flow using pre-computed and sorted supergates.
In order to enumerate the number of logic decompositions, we analyze the functional symmetry profile for the selected node of a circuit at each decomposition step. If there are more than one symmetry groups, we examine the one with the largest number of fanins only. If the largest symmetry group has fanins no more than the fanin limit (a fixed number 3 is used), we do not have alternative decompositions using the largest symmetry group. Otherwise, there are more than one decomposition choices. For example, if the largest symmetry group has 5 fanins, there exist ${}_5C_3 = 10$ decompositions. Using the same fanin limit 3, there are ${}_nC_3$ different decompositions for $n\ {}^{3}\ 4$. At each step of logic decompositions, we count the number of different decompositions as shown in Table 7.

Table 7 Number of different decompositions with fanin limit 3 (Appendix 2)

One exception used in this enumeration is the case that a node has exactly 4 fanins regardless of the symmetry information. In that case, all 4 fanins are selected for support and are replaced with a supergate pre-computed.

Using these numbers obtained at each step, we compute the average number of decompositions of the MCNC benchmark. Table 8 shows an example of the symmetry profiles of the selected node at each decomposition step of circuits. Column 'Symmetry information' is read as follows: fanin names within a bracket represent a group of functionally symmetric fanins, and the number within parentheses right after a fanin name represents logic depth of the fanin. 'Fanins selected' shows the selected support using the baseline support selection heuristic. Column '3-to-t' is checked with 'No' in case we could not find support-reducing decomposition with the selected support, but have found one with the increased fanin limit 4.

Table 8 Symmetry information of the selected node at each decomposition step: 9symml.blif (Appendix 2)

When we could not find support-reducing decomposition after increasing the fanin limit to 4, the bi-decomposition module was invoked. BeΣnchmark cm151a and cm152a are such instances. However, there are cases that we could find a supergate for mapping by choosing a different set of support. We do not take this situation into account for our enumeration.
In some benchmarks such as cm162a and cm85a, the size of the largest symmetry group at each step is less than or equals to the fanin limit (in other words, we do not have any alternatives using symmetry group only), but could find a better decomposition by choosing a different set of support. Neither is the case of our consideration.

Table 9. Average number of decompositions and number of decomposition steps (Appendix 2)

Table 9 summarizes the average number of decompositions and the total number of decomposition steps. (*) indicates that the circuit could not find support-reducing decomposition and have used bi-decomposition to finish decomposition. From this table, we notice that many circuits do not exhibit highly-symmetric functionality and there are only a few alternative decompositions in those cases. A few benchmarks such as cmb, count, and parity are highly symmetric circuits and we have considerable number of decomposition choices.

5.2 Cases with decomposition choices

Since we have observed that the benchmarks used for our experimentation do not have highly-symmetric functionality and there are limited choices of logic decompositions, we want to try all possible choices and analyze the result for benchmarks which have more than one decomposition choices. Even though there are a few choices at each step, total number of possible decompositions will blow up rapidly, without any restriction. For this reason, we limit our decomposition by trying different choices at a specific step but selecting exactly one branch at all other steps. In this experiment, interconnection delay is included for delay estimation using star model.

Fig. 10 Example: A search process for a circuit with decomposition choices (Appendix 1)

Figure 10 illustrates the search process of 9symml circuit with this restriction. In this example, there is one symmetry group, {1, 2, 3, 4, 5, 6, 7, 8, 9}. Since there are 9 fanins in this group, ${}_9C_3 = 84$ choices are possible at the first step. For 84 possible branches, we perform exactly one decomposition each and do not expand their children. Similarly, we try 20 choices at the second step, and obtain only one decomposition from each branch.
Based on this assumption, we can plot the result for each benchmark. Based on the plotted chart (although we do not include in this paper due to page limitation), we were able to observe that it is hard to find any specific decomposition that has superior quality in any criteria, especially in delay.
We also compute the correlation coefficient between delay and HPWL, which is a measure of linear association between variables. The formula for the correlation *r* is:
,

$$r = \frac{N\sum xy - \sum x \sum y}{\sqrt{N\sum x^2 - (\sum x)^2}\sqrt{N\sum y^2 - (\sum y)^2}}$$

where N = number of pairs of scores

Σxy = sum of the products of paired scores

Σx = sum of *x* scores

Σy = sum of y scores

Σx^2= sum of squared *x* scores.

The correlation coefficient between delay and HPWL are shown in Table 10. From this table, it is hard to say that there is any correlation between those two criteria. This might tell us that HPWL did not help in reducing delay overall, because HPWL of the whole circuit is not a direct measure of the critical path delay.

Table 10 Correlation coefficient between delay and HPWL

Benchmark	corr. coef.	Logarithmic corr. coef.
9symml	0.052	0.049
c8	0.017	0.022
cc	-0.089	-0.091
cm138a	0.147	0.151
cm163a	0.262	0.256
cmb	-0.205	-0.214
count	0.163	0.159
cu	0.485	0.482

decod	0.323	0.325
i1	-0.110	-0.111
lal	0.235	0.232
majority	0.193	0.205
parity	0.120	0.119
pcle	-0.183	-0.189
pm1	0.720	0.721
x4	-0.006	-0.006
Avg of absolute val.	0.207	0.208

5.3 Summary

In this section we have analyzed two main reasons of the negative result of our experiment. One is that the average number of decompositions of circuits for our experiment is quite low in most cases. This means that there are not very many choices in decomposition and it is hard to improve the design quality by utilizing physical location of placed cells. The other reason is that the criteria of our interest vary relatively small amount and the correlation of the targeting delay and HPWL is very low, thus HPWL did not guide very well to produce a fast circuit in this design flow. This observation would provide an answer to the question raised in this work.

6. Conclusions and Future Work

This paper has focused on extending constructive library-aware logic decomposition approach by including incremental physical placement phase and explored hypothesis that this extension can only yield further improvement in design quality. To test effectiveness of the hypothesis, we developed and implemented the COLOSSEUM system. The experimental result with COLOSSEUM was not able to show any benefit to our attempt at simultaneous logic and layout synthesis. Although we have explored more possible options to test our hypothesis on top of the COLOSSEUM framework, we obtained another set of negative results. We also address possible reasons of this outcome; mismatch of local and greedy nature of constructive paradigm with a need of global information in physical layout phase, and lack of utilizing physical information in supergate pre-computation step. We also analyzed two main reasons of the negative result of our experiment in section 5.3.
This work introduces, develops and elaborates constructive logic and layout synthesis method in the CAD flow. The primary contribution of this work is that we extended the original constructive library-aware logic decomposition paradigm by adding an incremental physical placement phase. This algorithm performs logic decomposition, technology mapping, and physical placement as one, utilizing physical layout information as well as logical information.
The work presented in this paper does not provide a complete answer to whether the constructive paradigm can be improved further by its extensions. Some possible extensions of this work are listed as follows:

- **Further exploration of physical layout phase**: This work can be extended further to include physical routing phase in the design flow. The final design solution might be improved by utilizing routing information at the time of logic decomposition.
- **Libraries with flexibilities:** In this work, all library supergates are pre-computed and sorted according to the criteria sought. The supergates computation depends largely on library cell properties, such as gate delay and area. One possible extension is to develop a method that computes a set of library primitives that can be mapped into a pattern by utilizing physical information. Another possible extension is to consider sharing a subset of a supergate when we select the support for decomposition.
- **Resynthesis by constructive logic and layout synthesis:** Instead of starting from scratch, we might get a better solution by identifying a critical part of a placed netlist and resynthesizing a subset of the network using our constructive logic and layout framework.
- **Power-driven logic synthesis and physical design**: One of important optimization goals in DSM technologies is power, which is not considered in this work. Support selection heuristics, for example, can be extended to include power optimization, since interconnect power is directly related to wire length.

References

[1] A. E. Caldwell, A. B. Kahng and I. L. Markov, “Can Recursive Bisection Alone Produce Routable Placements?” Proc. of DAC, pp. 477 – 482, 2000. Software available at http://vlsicad.eecs.umich.edu/BK/PDtools/

[2] S. Chatterjee and R. Brayton, “A New Incremental Placement Algorithm and its Application to Congestion-Aware Divisor Extraction,” Proc. of ICCAD, pp. 541-548, 2004.

[3] W. Gosti, Layout Aware Synthesis, Ph. D. Thesis, Univ. of California, Berkeley, 2000.

[4] W. Gosti, S. Khatri, and A. Sangiovanni-Vincentelli, “Addressing the Timing Closure Problem by Integrating Logic Optimization and Placement,” Proc. of ICCAD, pp. 224-231, 2001.

[5] D. Hill, “Method and Systems for High Speed Detailed Placement of Cells within an Integrated Circuit Design,” US Patent 6370673, April 2002.

[6] A. B. Kahng, P. Tucker, and A. Zelikovsky, “Optimization of Linear Placements for Wirelength Minimization with Free Sites,” Proc. of ASPDAC, pp. 241-244, 1999

[7] L. N. Kannan, P. R. Suaris, and H.-G. Fang, "A Methodology and Algorithms for Post-placement Delay Optimization," Proc. of DAC, pp. 327-332, 1994.

[8] V. Kravets, Constructive Multi-level Synthesis by Way of Functional Properties, Ph. D. Thesis, Univ. of Michigan, 2001.

[9] V. Kravets and P. Kudva, "Understanding Metrics in Logic Synthesis for Routability Enhancement," Proc. of SLIP, pp. 3-6, 2003.

[10] V. Kravets and K. Sakallah, "Constructive Library-Aware Synthesis Using Symmetries," Proc. of DATE, pp. 208-216, 2000.

[11] V. Kravets and K. Sakallah, "Resynthesis of Multi-level Circuits Under Tight Constraints Using Symbolic Optimization," Proc. of ICCAD, pp. 687-693, 2002.

[12] P. Kudva, A. Sullivan, and W. Bougherty, "Metrics for Structural Logic Synthesis," Proc. of ICCAD, pp. 551-556, 2002.

[13] T. Kutzschebauch and L. Stok, "Congestion Aware Layout Driven Logic Synthesis," Proc. of ICCAD, pp. 216-223, 2001.

[14] T. Kutzschebauch, and L. Stok, "Layout Driven Decomposition with Congestion Consideration," Proc. of DATE, pp. 672-676, 2002.

[15] S. Liu, K. Pan, and M. Pedram, "Alleviating Routing Congestion by Combining Logic Resynthesis and Linear Placement," Proc. of EuroDAC, pp. 578-582, 1993.

[16] Q. Liu and M. Marek-Sadowska, "Technology Mapping with Pre-layout Wire Length Prediction," IWLS, 2004.

[17] J. Lou, W. Chen, and M. Pedram, "Concurrent Logic Restructuring and Placement for Timing Closure," Proc. of ICCAD, pp. 31-36, 1999.

[18] A. Mishchenko, B. Steinbach, and M. Perkowski, "An Algorithm for Bi-Decomposition of Logic Functions," Proc. of DAC, pp. 103-108, 2001.

[19] A. Mishchenko, X. Wang, and T. Kam, "A New Enhanced Constructive Decomposition and Mapping Algorithm," Proc. of DAC, pp. 143 – 148, 2003.

[20] M. Pedram and N. Bhat, "Layout Driven Technology Mapping," Proc. of DAC, pp. 99-105, 1991.

[21] M. Pedram and N. Bhat, "Layout Driven Logic Restructuring/Decomposition," Proc. of ICCAD, pp. 134-137, 1991.

[22] D. Pandini, L. Pileggi, and A. Strojwas, "Congestion-Aware Logic Synthesis," Proc. of DATE, pp. 664-671, 2002.

[23] B. M. Riess and G. G. Ettelt, "Speed: Fast and Efficient Timing Driven Placement," IEEE International Symposium on Circuits and Systems (ISCAS), pp. 377 – 380, 1995.

[24] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks," IEEE Trans. on CAD, 2(3), pp. 202-211, 1983.

[25] F. Somenzi, "CUDD: CU Decision Diagram Package," 1997, http://vlsi.colorado.edu/~fabio/CUDD

[26] H. Vaishnav and M. Pedram, "Minimizing the Routing Cost During Logic Extraction," Proc. of DAC, pp. 70-75, 1995.

[27] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," Tech. Report, Microelectronics Center of North Carolina (MCNC), 1991.

Dr. Yoonna Oh received the B.S. and M.S. degree in Computer Science from Seoul National University and Ilinois Institute of Technology in 1992, and 1999, respectively, and Ph.D. degree in Computer Science and Engineering from the Univerisity of Michigan, Ann Arbor, in 2007. She is now working with Semiconductor R&D Center, Samsung Electronics.

Dr. Yuncheol Baek received the B.S., M.S. and Ph.D. degree in Computer Science from Seoul National University. During 2005-2007, he stayed at the Department of Electrical Engineering in Princeton University, New Jersey, USA, as a research fellow. He is an Associate Professor at the Division of Computer Software, Sangmyung University, Seoul, Korea.

Appendix 1 Large Figures

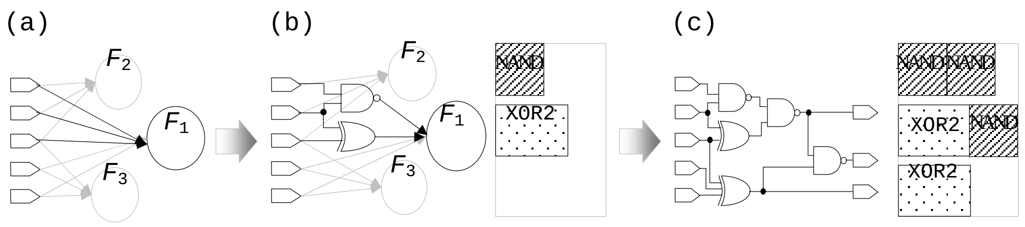

Fig. 6 Constructive logic and layout synthesis flow

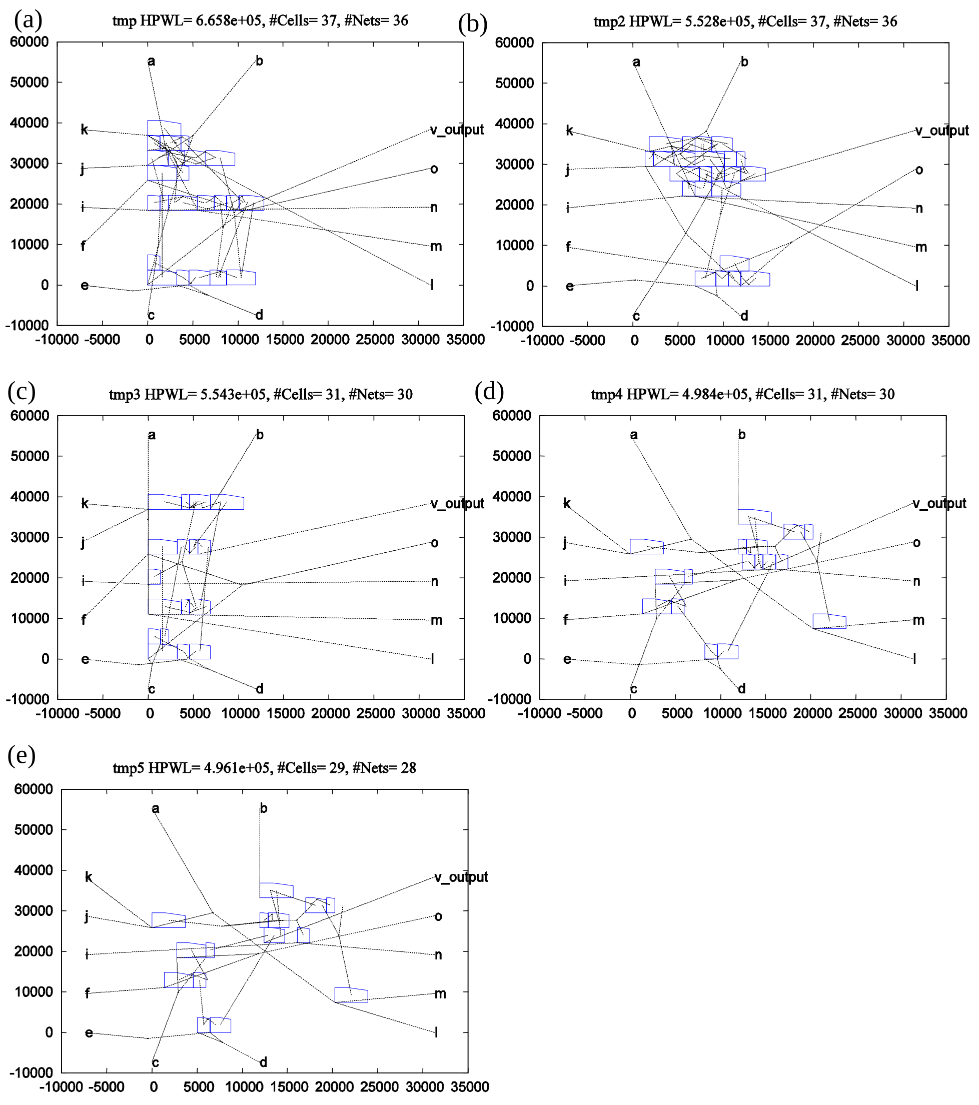

Fig. 9 Different layouts obtained with the constructive flow and different placement options: (a) Baseline (b) Baseline & Capo (c) Base+ (d) Base+ & Capo (e) each-Capo

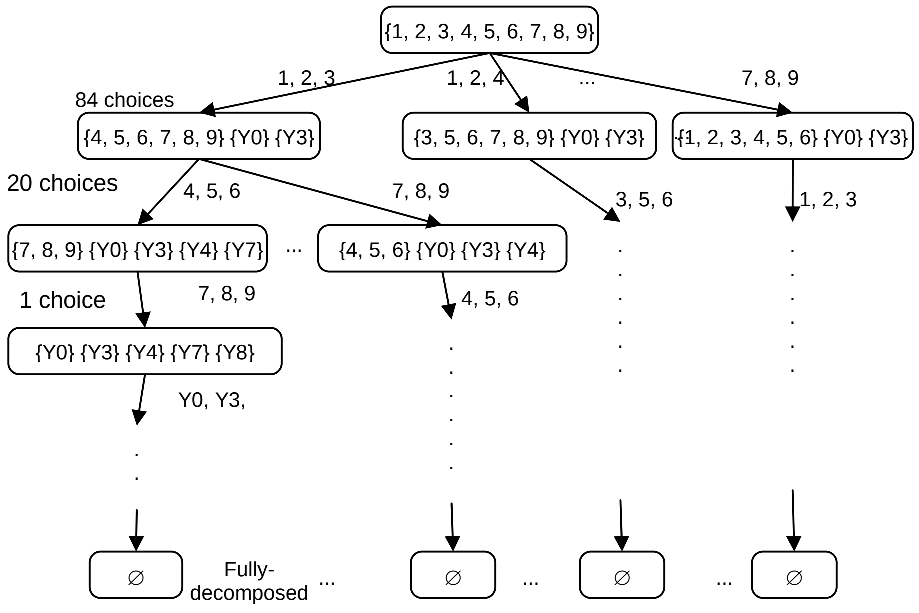

Fig. 10 Example: A search process for a circuit with decomposition choices

Appendix 2 Large Tables

Table 1 Node selection heuristics

Name	#I/#O	Area						Delay						HPWL					
		MC	LC	BB	DE	LL	RD	MC	LC	BB	DE	LL	RD	MC	LC	BB	DE	LL	RD
9symml	9/1	**290.3**	100%	100%	100%	100%	100%	**1.94**	100%	100%	100%	100%	100%	**430.6**	100%	100%	100%	100%	100%
b1	3/4	**45.8**	100%	100%	100%	100%	100%	**1.27**	100%	100%	100%	100%	100%	**110.9**	106%	107%	100%	100%	105%
c8	28/18	**646.7**	100%	100%	100%	100%	100%	**1.88**	101%	100%	100%	100%	101%	**3629.3**	97%	102%	100%	100%	109%
cc	21/20	**324.2**	100%	100%	100%	100%	100%	**1.65**	99%	98%	97%	100%	100%	**2126.1**	103%	103%	103%	100%	96%
cht	47/36	**947.1**	100%	100%	100%	100%	100%	**1.98**	104%	101%	107%	100%	98%	**10111.2**	101%	103%	96%	100%	99%
cm138a	6/8	**183.3**	100%	100%	100%	100%	100%	**1.57**	100%	100%	100%	100%	100%	**634.6**	89%	97%	95%	100%	110%
cm151a	12/2	**414.2**	100%	100%	100%	100%	100%	**1.86**	100%	100%	100%	100%	100%	**1009.3**	105%	100%	100%	100%	100%
cm152a	11/1	**174.8**	100%	100%	100%	100%	100%	**1.75**	100%	100%	100%	100%	100%	**453.1**	100%	100%	100%	100%	100%
cm162a	14/5	**312.3**	99%	108%	100%	100%	101%	**1.91**	100%	102%	100%	100%	97%	**923.5**	103%	102%	95%	100%	109%
cm163a	16/5	**176.5**	100%	100%	100%	100%	100%	**1.89**	100%	100%	99%	100%	100%	**792**	97%	96%	101%	100%	108%
cm42a	4/10	**105.2**	100%	100%	100%	100%	100%	**1.28**	100%	100%	100%	100%	100%	**510.5**	94%	100%	100%	100%	106%
cm82a	5/3	**101.8**	100%	100%	100%	100%	100%	**1.54**	100%	100%	100%	100%	100%	**172.6**	124%	100%	100%	100%	100%
cm85a	11/3	**339.5**	100%	100%	100%	100%	100%	**2.07**	100%	101%	101%	100%	100%	**683.4**	119%	119%	119%	100%	106%
cmb	16/4	**129**	100%	100%	100%	100%	100%	**1.47**	100%	100%	100%	100%	101%	**604.7**	98%	99%	99%	100%	103%
count	35/16	**750.3**	102%	101%	104%	101%	100%	**2.29**	103%	102%	103%	92%	102%	**5163.7**	110%	101%	118%	99%	106%
cu	14/11	**410.8**	90%	96%	100%	100%	107%	**2.16**	89%	87%	99%	100%	99%	**1435.9**	80%	84%	88%	100%	109%
decod	5/16	**232.5**	100%	100%	100%	100%	100%	**1.6**	101%	100%	100%	100%	100%	**1149.2**	92%	100%	97%	100%	108%
i1	25/16	**217.3**	100%	100%	100%	100%	100%	**1.77**	100%	100%	98%	100%	99%	**1837.4**	101%	98%	99%	100%	102%
lal	26/19	**509.2**	97%	98%	97%	97%	103%	**2.06**	97%	98%	102%	100%	101%	**2680.8**	108%	104%	117%	116%	99%
majority	5/1	**59.4**	100%	100%	100%	100%	100%	**1.49**	100%	100%	100%	100%	100%	**140.1**	100%	100%	100%	100%	100%
parity	16/1	**208.8**	100%	100%	100%	100%	100%	**1.76**	100%	100%	100%	100%	100%	**476.8**	100%	100%	100%	100%	100%
pcle	19/9	**337.8**	100%	104%	98%	100%	99%	**1.91**	99%	100%	100%	100%	100%	**1529.3**	92%	94%	89%	100%	101%
pm1	16/13	**264.8**	100%	92%	100%	100%	96%	**1.63**	100%	98%	99%	100%	100%	**1273.9**	101%	104%	111%	100%	104%
sct	19/15	**412.5**	99%	105%	97%	100%	100%	**2.01**	95%	110%	95%	100%	100%	**1872**	97%	102%	93%	100%	99%

t481	16/1	**181.6**	100%	100%	100%	100%	100%	**1.74**	100%	100%	100%	100%	100%	**451.5**	100%	100%	100%	100%	100%
tcon	17/16	**122.2**	100%	100%	100%	100%	100%	**1.19**	100%	100%	100%	100%	100%	**1075.2**	100%	100%	100%	100%	100%
unreg	36/16	**775.7**	100%	100%	99%	99%	100%	**1.97**	96%	94%	94%	100%	96%	**4680.2**	95%	97%	98%	100%	98%
x4	94/71	**2627.6**	94%	99%	99%	98%	103%	**4.33**	105%	132%	108%	103%	123%	**32926.4**	102%	113%	100%	101%	106%
AVG(%)			**99%**	**100%**	**100%**	**100%**	**100%**		**100%**	**101%**	**100%**	**100%**	**101%**		**100%**	**101%**	**101%**	**101%**	**103%**

Table 2 Area using different support selection heuristics

AREA	**baseline**	**base+**	**mdv**	***D***	***LD***	***LB***	***LGD***	***H***	***BH***	***BLH***	***BT***	***GLDT***	***LH***	***LBH***	***LBT***	***LGH***	**Random**
9symml	290	97%	97%	100%	100%	97%	100%	101%	101%	107%	97%	100%	107%	107%	97%	107%	120%
b1	46	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
c8	647	96%	117%	109%	119%	118%	110%	149%	120%	121%	108%	104%	143%	118%	118%	123%	151%
cc	324	94%	100%	98%	104%	87%	92%	141%	94%	93%	87%	92%	141%	93%	87%	103%	181%
cht	947	99%	100%	100%	100%	115%	100%	126%	122%	122%	115%	100%	125%	122%	115%	122%	109%
cm138a	183	100%	100%	83%	83%	91%	83%	91%	91%	91%	91%	83%	91%	91%	91%	91%	90%
cm151a	414	100%	100%	34%	34%	34%	34%	69%	69%	55%	34%	34%	59%	55%	34%	59%	101%
cm152a	175	100%	100%	54%	54%	72%	54%	99%	99%	91%	72%	54%	91%	91%	72%	91%	91%
cm162a	312	108%	102%	75%	107%	92%	103%	140%	96%	90%	75%	85%	114%	90%	75%	104%	143%
cm163a	177	101%	137%	110%	151%	150%	148%	142%	152%	143%	135%	122%	138%	152%	150%	156%	210%
cm42a	105	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
cm82a	102	100%	100%	132%	132%	132%	132%	132%	132%	132%	132%	132%	132%	132%	132%	132%	137%
cm85a	339	83%	103%	71%	113%	75%	85%	78%	78%	78%	85%	85%	78%	78%	85%	72%	129%
cmb	129	100%	100%	72%	72%	72%	72%	72%	72%	72%	72%	72%	72%	72%	72%	72%	155%
count	750	95%	145%	110%	126%	114%	114%	118%	110%	105%	112%	109%	120%	109%	116%	117%	214%
cu	411	95%	104%	82%	90%	107%	91%	126%	107%	122%	114%	91%	132%	122%	114%	105%	148%
decod	233	100%	100%	101%	101%	101%	101%	101%	101%	101%	101%	101%	101%	101%	101%	101%	136%
i1	217	95%	128%	109%	133%	116%	108%	92%	97%	97%	116%	108%	97%	103%	116%	97%	160%
lal	509	96%	139%	100%	146%	112%	98%	118%	98%	101%	112%	98%	116%	106%	114%	103%	205%
majority	59	100%	100%	71%	71%	71%	71%	80%	71%	71%	71%	71%	80%	71%	71%	71%	106%
parity	209	100%	100%	85%	85%	85%	85%	85%	85%	85%	85%	85%	85%	85%	85%	85%	100%
pcle	338	112%	136%	123%	152%	137%	120%	135%	125%	122%	133%	122%	140%	129%	137%	133%	150%
pm1	265	100%	100%	98%	91%	87%	87%	104%	87%	87%	87%	87%	104%	87%	87%	80%	136%
sct	412	105%	130%	99%	130%	117%	120%	128%	115%	114%	100%	107%	117%	127%	117%	116%	173%
t481	182	100%	100%	103%	100%	103%	107%	114%	100%	100%	103%	107%	120%	100%	103%	100%	101%
tcon	122	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
unreg	776	102%	100%	77%	87%	108%	87%	98%	98%	98%	108%	87%	98%	98%	108%	98%	99%
x4	2628	100%	153%	89%	149%	128%	106%	180%	96%	103%	111%	92%	175%	126%	131%	119%	176%
AVG(%)		99%	110%	92%	105%	101%	97%	111%	101%	100%	98%	94%	110%	102%	101%	102%	136%

Table 3 Delay using different support selection heuristics

DELAY	**baseline**	**base+**	**mdv**	***D***	***LD***	***LB***	***LGD***	***H***	***BH***	***BLH***	***BT***	***GLDT***	***LH***	***LBH***	***LBT***	***LGH***	**Random**
9symml	1.94	99%	99%	100%	100%	99%	100%	113%	113%	110%	99%	100%	110%	110%	99%	110%	117%
b1	1.27	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
c8	1.88	102%	97%	103%	103%	94%	96%	118%	99%	98%	96%	96%	101%	96%	94%	98%	116%
cc	1.65	101%	100%	100%	106%	99%	100%	111%	105%	99%	99%	100%	111%	99%	99%	108%	110%
cht	1.98	96%	100%	100%	100%	97%	100%	101%	103%	95%	97%	100%	96%	95%	97%	95%	103%
cm138a	1.57	100%	100%	97%	97%	97%	97%	97%	97%	97%	97%	97%	97%	97%	97%	97%	99%
cm151a	1.86	100%	100%	89%	89%	97%	89%	163%	163%	105%	97%	89%	105%	105%	97%	105%	116%
cm152a	1.75	100%	100%	87%	87%	103%	87%	122%	122%	107%	103%	87%	107%	107%	103%	107%	100%
cm162a	1.91	98%	94%	101%	94%	102%	94%	111%	103%	91%	90%	91%	97%	91%	90%	91%	115%
cm163a	1.89	97%	87%	104%	93%	87%	85%	99%	97%	95%	98%	95%	87%	86%	87%	86%	142%
cm42a	1.28	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
cm82a	1.54	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	102%
cm85a	2.07	101%	92%	113%	100%	94%	94%	90%	90%	90%	96%	94%	90%	90%	96%	91%	131%
cmb	1.47	100%	100%	105%	105%	105%	105%	105%	105%	105%	105%	105%	105%	105%	105%	105%	132%
count	2.29	96%	95%	130%	90%	85%	85%	109%	94%	81%	84%	84%	96%	82%	82%	93%	126%
cu	2.16	111%	101%	94%	88%	84%	87%	129%	94%	103%	96%	87%	97%	103%	96%	87%	100%
decod	1.6	100%	100%	89%	89%	91%	89%	91%	91%	91%	91%	89%	91%	91%	91%	91%	99%
i1	1.77	100%	110%	106%	107%	98%	98%	102%	95%	95%	98%	98%	98%	98%	98%	98%	133%
lal	2.06	91%	94%	99%	98%	102%	86%	101%	94%	87%	93%	86%	86%	94%	93%	90%	140%
majority	1.49	100%	100%	97%	97%	97%	97%	105%	97%	97%	97%	97%	105%	97%	97%	97%	108%
parity	1.76	98%	100%	102%	89%	89%	89%	89%	89%	89%	89%	89%	89%	89%	89%	89%	116%
pcle	1.91	102%	97%	109%	98%	95%	95%	103%	92%	92%	93%	97%	102%	92%	95%	95%	115%
pm1	1.63	99%	100%	108%	108%	101%	100%	105%	100%	100%	101%	100%	105%	100%	101%	106%	129%
sct	2.01	97%	93%	100%	99%	92%	92%	117%	101%	105%	91%	91%	92%	97%	92%	94%	113%
t481	1.74	100%	100%	123%	100%	108%	94%	120%	100%	100%	108%	94%	110%	100%	108%	100%	107%
tcon	1.19	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
unreg	1.97	108%	100%	97%	100%	107%	100%	100%	100%	100%	107%	100%	100%	100%	107%	100%	100%
x4	4.33	101%	99%	111%	101%	87%	89%	108%	85%	86%	87%	87%	87%	87%	93%	94%	112%

AVG(%)		100%	99%	102%	98%	97%	95%	107%	101%	97%	97%	95%	99%	97%	97%	97%	114%

Table 4 HPWL using different support selection heuristics

HPWL	**baseline**	**base+**	**mdv**	***D***	***LD***	***LB***	***LGD***	***H***	***BH***	***BLH***	***BT***	***GLDT***	***LH***	***LBH***	***LBT***	***LGH***	**Random**
9symml	431	99%	90%	100%	99%	89%	99%	107%	107%	108%	89%	99%	108%	108%	89%	108%	114%
b1	111	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
c8	3629	99%	101%	101%	102%	102%	97%	117%	104%	105%	98%	94%	108%	107%	102%	106%	108%
cc	2126	98%	100%	102%	104%	97%	103%	110%	100%	99%	97%	103%	110%	99%	97%	100%	129%
cht	10111	94%	100%	94%	94%	100%	94%	100%	98%	98%	100%	94%	100%	98%	100%	98%	104%
cm138a	635	100%	100%	70%	70%	77%	70%	77%	77%	77%	77%	70%	77%	77%	77%	77%	76%
cm151a	1009	100%	100%	44%	44%	44%	44%	77%	77%	60%	44%	44%	61%	60%	44%	61%	88%
cm152a	453	100%	100%	63%	63%	77%	63%	111%	111%	84%	77%	63%	84%	84%	77%	84%	90%
cm162a	924	93%	105%	91%	89%	103%	84%	150%	90%	99%	96%	81%	101%	99%	96%	95%	137%
cm163a	792	96%	95%	101%	105%	118%	106%	101%	102%	97%	107%	103%	94%	100%	118%	101%	148%
cm42a	511	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
cm82a	173	100%	100%	136%	136%	136%	136%	136%	136%	136%	136%	136%	136%	136%	136%	136%	140%
cm85a	683	102%	117%	95%	115%	104%	121%	92%	92%	92%	110%	121%	92%	92%	110%	83%	126%
cmb	605	98%	100%	94%	94%	94%	94%	94%	94%	94%	94%	94%	94%	94%	94%	94%	139%
count	5164	100%	128%	127%	119%	89%	89%	101%	97%	96%	99%	98%	98%	92%	89%	96%	158%
cu	1436	89%	105%	72%	76%	92%	80%	116%	86%	100%	96%	80%	100%	100%	96%	85%	120%
decod	1149	100%	100%	96%	96%	91%	96%	91%	91%	91%	91%	96%	91%	91%	91%	91%	107%
i1	1837	100%	104%	103%	106%	105%	105%	99%	99%	99%	105%	105%	100%	102%	105%	100%	106%
lal	2681	97%	140%	109%	153%	116%	98%	118%	98%	105%	109%	99%	113%	109%	111%	104%	166%
majority	140	100%	100%	99%	99%	99%	99%	95%	99%	99%	99%	99%	95%	99%	99%	99%	96%
parity	477	88%	100%	89%	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%	80%	142%
pcle	1529	90%	104%	103%	111%	94%	107%	110%	92%	95%	89%	98%	109%	92%	94%	99%	143%
pm1	1274	101%	100%	120%	102%	115%	116%	118%	111%	111%	115%	116%	118%	111%	115%	99%	123%
sct	1872	96%	111%	101%	118%	108%	106%	106%	104%	102%	103%	109%	106%	105%	108%	117%	139%
t481	452	100%	100%	110%	100%	106%	106%	115%	100%	100%	106%	106%	135%	100%	106%	100%	121%
tcon	1075	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
unreg	4680	100%	100%	94%	96%	105%	96%	107%	107%	107%	105%	96%	107%	107%	105%	107%	124%
x4	32926	95%	130%	101%	115%	118%	97%	127%	100%	101%	109%	103%	112%	108%	115%	108%	132%
AVG(%)		98%	105%	97%	100%	99%	96%	106%	98%	98%	98%	96%	101%	98%	98%	97%	121%

Table 6. Results of applying different placement options

	Area			Delay					HPWL				
	(a)(b)	(c)(d)	(e)	**(a)**	(b)	(c)	(d)	(e)	**(a)**	(b)	(c)	(d)	(e)
9symml	**282**	103%	111%	**3.06**	93%	113%	139%	161%	**415**	92%	86%	87%	91%
b1	**46**	100%	100%	**0.51**	82%	100%	82%	82%	**102**	89%	100%	89%	89%
c8	**638**	98%	100%	**2.26**	88%	122%	72%	81%	**1908**	82%	99%	80%	80%
cc	**324**	93%	93%	**1.53**	111%	88%	82%	82%	**1152**	91%	98%	89%	89%
cht	**947**	101%	101%	**2.58**	70%	132%	88%	81%	**4035**	81%	100%	80%	80%
cm152a	**175**	100%	100%	**1.31**	95%	100%	95%	95%	**321**	90%	100%	90%	90%
cm162a	**311**	104%	117%	**1.86**	91%	109%	86%	96%	**757**	85%	96%	84%	92%
cm163a	**177**	107%	107%	**1.56**	95%	82%	78%	78%	**516**	87%	107%	92%	92%
cm42a	**105**	100%	100%	**0.42**	99%	100%	99%	99%	**313**	86%	100%	86%	86%
cm82a	**102**	100%	100%	**0.96**	94%	100%	94%	96%	**163**	97%	100%	97%	97%
cm85a	**339**	135%	133%	**2.52**	89%	117%	109%	118%	**613**	93%	127%	115%	112%
cmb	**129**	95%	95%	**0.72**	109%	114%	104%	104%	**466**	89%	94%	82%	82%
count	**750**	95%	92%	**3.14**	86%	113%	98%	99%	**2306**	85%	95%	80%	81%
cu	**394**	98%	98%	**3.39**	69%	94%	78%	88%	**1172**	71%	93%	75%	72%
decod	**233**	100%	100%	**1.01**	102%	100%	95%	95%	**657**	71%	100%	72%	72%
lal	**519**	94%	91%	**2.65**	81%	84%	80%	83%	**1654**	84%	101%	79%	78%
parity	**209**	100%	100%	**3.00**	91%	86%	83%	89%	**359**	91%	94%	84%	84%
pcle	**336**	113%	113%	**1.78**	98%	125%	144%	118%	**951**	85%	101%	83%	83%
pm1	**253**	103%	107%	**1.06**	107%	125%	98%	95%	**849**	81%	95%	82%	84%
sct	**412**	113%	113%	**2.21**	84%	136%	103%	95%	**1216**	86%	119%	88%	88%
T481	**182**	100%	100%	**1.70**	109%	99%	100%	109%	**401**	85%	103%	84%	85%
tcon	**122**	100%	100%	**0.17**	100%	100%	100%	100%	**582**	93%	100%	93%	93%
unreg	**771**	103%	103%	**2.15**	69%	128%	131%	138%	**2249**	82%	90%	79%	79%
AVG(%)		102%	103%		92%	107%	97%	99%		86%	100%	86%	86%

Table 7. Number of different decompositions with fanin limit 3

# OF FANINS (n)	4	5	6	7	8	9	10	11	12	13	14	15	16
# of decomp. (${}_nC_3$)	4	10	20	35	56	84	120	165	220	286	364	455	560

Table 8. Symmetry information of the selected node at each decomposition step: 9symml.blif

9SYMML.BLIF					
Step	Node	Symmetry information	Fanins selected	3-to-t	#decomp
1	52	{1(0), 2(0), 3(0), 4(0), 5(0), 6(0), 7(0), 8(0), 9(0)}	1 2 3		84
2	52	{4(0), 5(0), 6(0), 7(0), 8(0), 9(0)} {_Y0(1)} {_Y3(2)}	4 5 6		20
3	52	{7(0), 8(0), 9(0)} {_Y4(1)} {_Y0(1)} {_Y7(2)} {_Y3(2)}	7 8 9		1
4	52	{_Y4(1)} {_Y0(1)} {_Y8(1)} {_Y7(2)} {_Y11(2)} {_Y3(2)}	_Y4 _Y0 _Y8		1
5	52	{_Y7(2)} {_Y11(2)} {_Y12(2)} {_Y13(2)} {_Y3(2)}	_Y7 _Y11 _Y12 _Y13	No	1
6	52	{_Y3(2), _Y21(4)} {_Y18(4)}	_Y3 _Y21 _Y18		1

Table 9. Average number of decompositions and number of decomposition steps

BENCHMARK	AVERAGE # OF DECOMP.	# OF STEPS	BENCHMARK	AVERAGE # OF DECOMP.	# OF STEPS
9symml	18.0	6	count	53.7	93
b1	1.0	4	cu	7.0	30
c8	5.4	52	decod	5.5	32
cc	1.3	33	I1	9.8	24
cht	1.0	46	lal	2.1	60
cm138a	10.5	16	majority	2.5	2
cm151a	1.0	(*) 2	parity	191.6	7
cm152a	1.0	(*) 1	pcle	10.4	39
cm162a	1.0	25	pm1	3.7	27
cm163a	1.8	16	sct	2.7	49
cm42a	1.0	10	t481	1.0	13
cm82a	1.0	5	tcon	1.0	16
cm85a	1.0	16	unreg	1.0	47
cmb	83.4	20	x4	2.0	310