Layout-Driven Resource Sharing in High-Level Synthesis

Junhyung Um Jae-hoon Kim Taewhan Kim

Department of Electrical Engineering & Computer Science
and Advanced Information Technology Research Center(AITrc)
Korea Advanced Institute of Science and Technology,
Taejon, Korea

Abstract – In deep submicron (DSM) technology, the interconnects are equally as or more important than the logic gates. In particular, to achieve timing closure in DSM technology, it is very necessary and critical to consider the interconnect delay at an early stage of the synthesis process. It has been known that resource sharing in high-level synthesis is one of the major synthesis tasks which greatly affect the final synthesis/layout results. In this paper, we propose a new layout-driven resource sharing approach to overcome some of the limitations of the previous works in which the effects of layout on the synthesis have never been taken into account or considered in local and limited ways, or whose computation time is excessively large. The proposed approach consists of two steps: (Step 1) We relax the integrated resource sharing and placement into an efficient linear programming (LP) formulation based on the concept of *discretizing* placement space; (Step 2) We derive a feasible solution from the solution obtained in Step 1. Then, we employ an iterative mechanism based on the two steps to tightly integrate resource sharing and placement tasks so that the slack time violation due to interconnect delay (determined by placement) as well as logic delay (determined by resource sharing) should be minimized. From experiments using a set of benchmark designs, it is shown that the approach is effective, and efficient, completely removing the slack time violation produced by conventional methods.

1 Introduction

In deep-submicron (DSM) technologies, the interconnection delay becomes a dominating factor of system performance. Ignoring or inaccurately estimating interconnect delay at early stages of synthesis process, particularly in the high-level synthesis, could lead to serious timing closure problems at the final layout stage. Due to such an importance of interconnect delay in high-level synthesis, many researchers have addressed the problem of integrating layout into the several tasks of high-level synthesis [1, 2, 3, 4, 5, 6, 7, 8].

Weng and Alice [1] attempted to minimize interconnect delay by assigning operations to the modules that are close to the modules bound to their predecessors. The approach is greedy in that the operations on the critical-path are scheduled and bound first. Tarafdar *et al.* [2] incorporated layout into scheduling and binding of data transfers in high-level synthesis. The binding process is based on a branch-and-bound heuristic. Prabhakanran *et al.* [3] proposed a parallel algorithm to reduce the time complexity of simultaneous scheduling, binding and placement. Moshnyaga and Tamaru [4] defined eight basic moves (e.g., operation rebinding, tree-height reduction, $\cdots$) and repeatedly applied the moves to improve the current binding and placement solution, which is a very time consuming process. Kim *et al.* [5] performed operation binding, placement, and scheduling sequentially. The placement was driven by the inter-clock slack time information obtained from the binding results. Jang and Pangrle [6] proposed a layout estimation technique for binding, and used it to select the most effective binding. Further, Munch *et al.* [7] formulated and analyzed the simultaneous binding and placement problem into a Mixed Integer Linear Programming (MILP) model. However, the applications of both in [6] and [7] are confined to a 1-dimensional target architecture. Fang and Wong [8] estimated the layout cost for binding using a simulated annealing based floorplanner.

Note that in the previous approaches [1, 2, 3, 4, 5, 6, 7, 8] mentioned above, resource sharing has been considered one of the most important synthesis tasks which greatly affect the timing-driven placement results. In this paper, we propose a new layout-driven resource sharing approach in high-level synthesis to overcome some of the limitations of the previous works in which the effects of layout on the synthesis have never been taken into account or considered in local and limited ways, or whose computation time is excessively large. Specifically, our approach performs in two steps: (Step 1) We transform the integrated resource sharing and placement into a problem of rectangle packing with distance constraints. We then (globally) approximate the solution of the packing problem by using an efficient linear programming (LP) formulation; (Step 2) We derive a feasible solution from the solution obtained in Step 1. We then employ an iterative mechanism based on the two steps to tightly integrate resource sharing and placement tasks so that the slack time violation is minimized. Our key consideration is to explore the solution space fully and globally. Consequently, the proposed approach is designed to transform the combined resource sharing and placement problem into a 2-dimensional rectangle packing problem with distance constraints and approximate the solution of the transformed problem by using an efficient LP formulation. In experiments, a set of data is provided to show how much the proposed approach is useful in predicting the interconnect delay accurately in synthesis, and thus optimizing the timing of final circuit in layout afterwards.

2 Target Architecture and Motivation

2.1 Target Architecture

The target architecture we used is the distributed-register architecture [5], in which each functional unit performs computation by reading data from some location of the local register file that is dedicated to the functional unit and writing the outcome into some location of the register file. This allows an almost zero delay for the interconnect between the functional units and their local storage. However, for a data transfer between two operations bound to different functional units, a wire (with nontrivial delay) connecting the two functional units is used. The distributed-register architecture typically forms a regular (clustered) structure and is generally compatible with the nature of locality of computations.

2.2 Motivating Example

Before describing our main idea of solving the problem of resource sharing and placement, we first give an example to show how resource sharing results affect placement and conversely, placement results affects resource sharing.

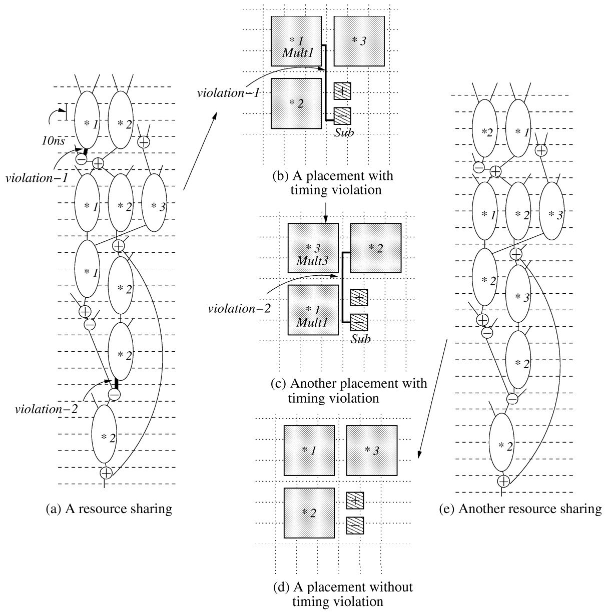


Figure 1: A resource sharing and its two possible placement instances, causing timing violations and no violation.

Suppose we want to perform module placement for the resource sharing in the scheduled data flow graph (CDFG) in Figure 1(a) where the clock period is 10ns. We assume that three multipliers, one adder and one subtractor modules are used, in which their execution delays are 38ns, 4ns and 4ns, respectively. The resource sharings among the multiplication operations are specified by the numbers labeled in the operations in Figure 1(a). Figure 1(b) shows an instance of placement.[1] (For simplicity, we assume that it takes 1ns to pass a signal through a distance of one grid-length in Figure 1(b).) The heavy (long) line in Figure 1(b) indicates 4ns interconnect delay. The heavy line in the upper side of Figure 1(a) indicates the data transfer from a register in functional module *mult1* to *sub*.

Since one subtraction and one addition are executed sequentially in a clock cycle, the the slack time for the data transfer is 10-(4+4) = 2ns. However, since the interconnect delay is more than the slack time, the data transfer and two operations (addition and subtraction) cannot be completed in the clock period, as indicated *violation-1* in Figures 1(a) and (b). To solve the timing violation, let us put functional modules *mult1* and *sub* side by side, as shown in Figure 1(c). Even the replacement removes *violation-1*, another timing violation, indicated as *violation-2* in the figure, happens due to the long wire between functional module *mult2* and *sub*. In fact, for the given resource sharing in Figure 1(a), we exhaustively searched all possible placements, but could not find a placement that has no timing violation. On the other hand, Figure 1(e) shows a resource sharing other than the one in Figure 1(a). One placement instance in Figure 1(d), which is exactly the same as that in Figure 1(b), causes no timing violation at all.

The examples in Figures 1 clearly show that the decision made in resource sharing greatly affects the degree of the layout complexity. Resource sharing without considering wire delay is more likely to spend an excessive run time in placement to meet the timing constraints, or to produce a less efficient layout (e.g., bigger chip area). Consequently, it is very necessary to perform the tasks of resource sharing and placement in an integrated fashion. However, most existing approaches perform the two tasks iteratively (i.e., one task first and another next, and repeat them) or locally, which is practically infeasible in terms of run time. In that respect, we propose

[1] We performed the placement according to the procedure in [5].

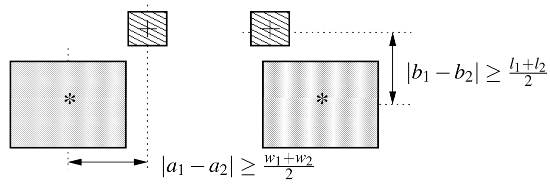


Figure 2: A simple example showing the disjoint constraint.

a new approach which solves the problem of integrating resource sharing and placement globally, but efficiently. The problem we want to solve is, for a given scheduled CDFG, module library, and placement area, to determine a resource sharing and placement such that the slack time violation is as minimal as possible.

3 The Combined Resource Sharing and Placement

This section contains two parts: (i) formulating and solving placement problem with slack time constraints (section 3.1) and (ii) integrating resource sharing problem into our solution of the placement problem (section 3.2).

3.1 The Formulation of Placement Problem

If we do not consider the resource sharing problem (i.e., temporarily assuming each operation uses a distinct functional module instance), satisfying the following two types of constraint leads to a feasible placement solution:

- *disjoint constraint*: Two functional module instances should not overlap.
- *slack time constraint*: The delay by the interconnect between functional modules should not cause any slack time violation.

• **The exact placement problem**: Let M_i and M_j denote two functional modules, and (a_i, b_i) and (a_j, b_j) be their (center) positions in a feasible placement, respectively. Further, let l_i and l_j be the lengths (i.e., heights) of M_i and M_j, and w_i and w_j be the widths of M_i and M_j, respectively. Then, the *disjoint constraint* is to satisfy (See Figure 2.):

$$|a_i - a_j| \geq \frac{w_i + w_j}{2}, \quad |b_i - b_j| \geq \frac{l_i + l_j}{2}. \tag{1}$$

Next, we consider a relationship between geometric distances between modules and slack times in CDFG. The *slack time* at a clock step is defined to be the maximum delay that can be used by interconnects without violating the clock period constraint. For example, suppose operation op_1 is scheduled at clock step i and bound to functional module M_1. Then, the slack time at clock step i can be depicted as Figure 3(a). Consequently, the *slack time constraint* is to place modules so that the total (maximum) data transfer delay chained through interconnects used at each clock step should not exceed the slack time at that clock step. (See Figure 3(b).) Another equivalent interpretation of the slack time constraint is to place modules so that the longest total length of chained interconnects used at each clock step should not larger than the distance whose delay amounts to the slack time at that clock step. We formally state the slack time constraint at clock step i as

$$D(I^i_{k_1}) + D(I^i_{k_2}) + \cdots + D(I^i_{k_i}) \leq DIST^i_{slack} \tag{2}$$

where $I^i_{k_1}, I^i_{k_2}, \cdots, I^i_{k_i}$ are the interconnects that are used in a sequence in clock step i. $D(\cdot)$ indicates the delay for the corresponding interconnect. $DIST^i_{slack}$ is the physical distance corresponding to the slack time at clock step i. Note that the quantity of $DIST^i_{slack}$ can be predetermined according to the wire delay model used.

An interesting placement problem is to find a rectangular space of minimum size into which all functional modules are placed while satisfying the distance and slack time constraints. The placement problem is then equivalent to a 2-dimensional packing problem, and there exist well-studied heuristics [11, 12] in the literature. However, it looks there is no easy way to effectively combine resource sharing problem into the packing heuristics. To tackle the integrated problem efficiently, we consider slightly different placement objective and assumption: We assume that we are given a rectangular space. Then, we want to place the modules in the space such that all the distance constraints should be satisfied, but the slack time violation due to interconnects is *to be minimized.* We have three reasons that validate our resource sharing and placement problem: (a) We found that the problem can be approximated by developing an efficient linear programming (LP) formulation that solves the problem in a global respect; (b) Designer mostly wants to know, at the (high-level) resource sharing stage, how much the slack time violation will be later; (c) By expanding the size of rectangular placement space incrementally in the iterative applications of our solution, it is possible eventually to solve the original problem as well. We now describe our approximation algorithm based on an LP formulation.

• **Approximation of the exact placement**: Let $B = [0,L] \times [0,W]$ be a rectangular space in which n rectangular modules in set $R = \{M_1, \cdots, M_n\}$ are to be placed. We denote w_i and l_i as the width and length (i.e., height) of module M_i, respectively. The placement objective is to place all the modules in R to B such that all rectangles in R do not overlap and the maximum value of $D(I^i_{k_1}) + D(I^i_{k_1}) + \cdots + D(I^i_{k_i}) - DIST^i_{slack}$ is minimized. We equally partition the space with L horizontal lines and W vertical lines to have approximately $(L)\cdot(W)$ grid points. (See Figure 5.) We restrict the placement problem in such a way that the lower-left corner of every module should be possibly positioned only at the grid points in B. We refer to the grid point at which the lower-left corner of the module is placed as the *anchor point* of the module in B. We introduce, for each functional module M_i and grid point (j,k), a real variable x^i_{jk}, $0 \le x^i_{jk} \le 1$, to indicate the likelyhood of positioning the lower-left corner of M_i to grid point (j,k). The larger the value of x^i_{jk} is, the more M_i is likely to be placed at position (j,k). Then, we approximate the solution of the placement problem by finding the values of x^i_{jk} with the following constraints.

First, we translate the *disjoint constraint* into LP formulation:

$$\text{for each grid point } (s,t), \quad \sum_{i=1}^{n} \sum_{j=s-\ell_i}^{s} \sum_{k=t-w_i}^{t} x^i_{jk} \le 1. \tag{3}$$

Constraint 3 ensures that the sum of the fractions assigned to all rectangles covering a coordinate (s,t) does not exceed one. It implies that any pair of rectangles in a feasible placement does not overlap, because otherwise, at a grid point in the overlapping region the constraint will be violated. The term $\sum_{j=s-\ell_i}^{s}\sum_{k=t-w_i}^{t} x^i_{jk}$ in *Eq.(3)* represents the sum of the likelyhoods (i.e., fractions) of the placement of module M_i that covers coordinate (s,t), as shown in Figure 4(a) where in order for M_i to cover grid point (s,t), M_i should have its anchor point in the shaded area $[t-l_i, t] \times [s-w_i, s]$.

In addition, since every module should be placed, we have the constraint that

$$\text{for each module } M_i, \quad \sum_{j,k} x^i_{jk} = 1.$$

Next, we formulate the *slack time constraint* into LP by introducing nonnegative real variables $\delta^x_{ii'}$ and $\delta^y_{ii'}$ for each pair of modules M_i and $M_{i'}$:[2]

$$\text{for each pair of modules } (M_i, M_{i'}) \ (i \le i'),$$

$$-\delta^x_{ii'} \le \sum_{j=0}^{W}\sum_{k=0}^{L} j(x^i_{jk} - x^{i'}_{jk}) \le \delta^x_{ii'}, \tag{4}$$

$$\text{for each pair of modules } (M_i, M_{i'}) \ (i \le i'),$$

$$-\delta^y_{ii'} \le \sum_{j=0}^{W}\sum_{k=0}^{L} k(x^i_{jk} - x^{i'}_{jk}) \le \delta^y_{ii'}, \tag{5}$$

$$\text{for each pair of modules } (M_i, M_{i'}), \delta^x_{ii'} + \delta^y_{ii'} \le DIST^{i,i'}_{slack}. \tag{6}$$

where $DIST^{i,i'}_{slack}$ indicates the minimum of the physical distances corresponding to the slack times at the clock steps at which modules M_i and $M_{i'}$ are executed.

To simplify the LP expression, we measure the distance between modules by the L^1-distance between anchor points of modules rather than that between center points. To translate an absolute difference (i.e., distance) between two anchor points into LP constraints, we make use of the bounding box shown in Figure 4(b): we enclose two anchor points by a bounding box and try to reduce its size while satisfying the slack time constraints corresponding the modules of the anchor points. That is, in *Eq.(4)* $\sum_{j=0}^{L}\sum_{k=0}^{W} j \cdot x^i_{jk}$ represents the x-coordinate of M_i's anchor point and $\sum_{j=0}^{L}\sum_{k=0}^{W} j \cdot x^{i'}_{jk}$ represents M'_i's anchor point. Thus, *Eq.(4)* constrains that the absolute difference between x-coordinates should not be larger than $\delta^x_{ii'}$. Similarly, *Eq.(5)* constrains that the absolute difference between y-coordinates should not be larger than $\delta^y_{ii'}$. *Eq.(6)* indicates the slack time constraints.

Finally, we describe our LP formulation for layout-driven resource sharing. We assume that we are given a rectangular space $B = [0,W] \times [0,L]$. we equally partition B using W horizontal and L vertical lines, so that B is composed of $W \cdot L$ integral (internal) coordinates. Since our objective is to minimize the worst slack time violation, the formulation is:

$$Minimize \quad D_{violation} \quad \text{subject to}$$

$$\text{for each grid point } (s,t) \quad \sum_{i=1}^{n} \sum_{j=s-\ell_i}^{s} \sum_{k=t-w_i}^{t} x^i_{jk} \le 1 \tag{7}$$

$$\text{for each pair of modules } (M_i, M_{i'}) \ (i \le i')$$

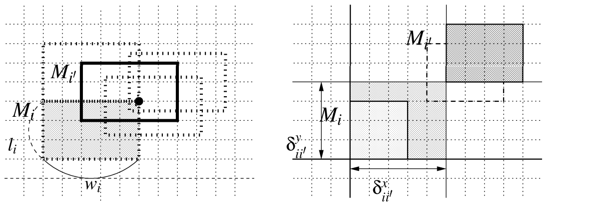
$$-\delta^x_{ii'} \le \sum_{j=0}^{W}\sum_{k=0}^{L} j(x^i_{jk} - x^{i'}_{jk}) \le \delta^x_{ii'} \tag{8}$$

$$\text{for each pair of modules } (M_i, M_{i'}) \ (i \le i')$$

$$-\delta^y_{ii'} \le \sum_{j=0}^{W}\sum_{k=0}^{L} k(x^i_{jk} - x^{i'}_{jk}) \le \delta^y_{ii'} \tag{9}$$

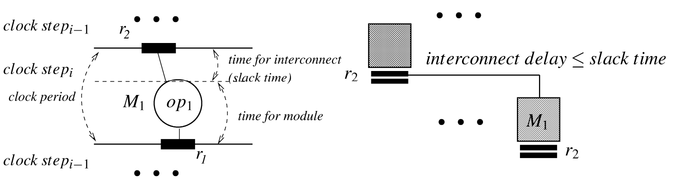
$$\text{for each pair of modules } (M_i, M_{i'})$$

$$\delta^x_{ii'} + \delta^y_{ii'} - DIST^{ii'}_{slack} \le D_{violation} \tag{10}$$



(a) Example for disjoint constraints (b) Bounding box for two module

Figure 4: An examples showing the constraints in our LP formulation.



(a) Slack time in CDFG (b) Relation between slack time and interconnect delay

Figure 3: A simple example showing the slack time constraint.

[2] We restrict the presentation of our LP formulation to the case that k_i=k_1 in $Eq.$ (2). Extending our formulation to the general case is rather straightforward.

$$\text{for each } M_i \quad \sum_{j=0}^{W} \sum_{k=0}^{L} x_{jk}^{i} = 1 \tag{11}$$

$$\text{for each pair of modules } (M_i, M_{i'}), \ \ \delta_{ii'}^{x} \geq 0, \ \ \delta_{ii'}^{y} \geq 0$$

$$D_{slack} \geq 0$$

$$\text{for each } M_i \text{and grid point } j,k, \quad 0 \leq x_{jk}^{i} \leq 1.$$

where N is the number of modules to be placed. *Eq. (7)* ensures that the sum of the fractions assigned to all modules that cover coordinate (s,t) does not exceed one. *Eqs. (8)-(10)* constrain that the worst slack time violation between modules M_i and $M_{i'}$ should not greater than $D_{violation}$, which is measured in term of physical distance rather than execution delay (i.e., time).

3.2 Integrating Resource Sharing into LP

Note that initially we assumed that all the operations in the CDFG were allocated to distinct functional modules. However, two functional modules with the same operation type can be combined into one if their assigned operations are not executed concurrently at all.

To support the module sharing among operations, we should allow the possibility of overlapping in placement between functional modules of the same operation type. We can model such an overlapping in LP by simply relaxing the disjoint constraints in *Eq.(7)*. In other words, *Eq.(7)* will be performed selectively only for the modules of different operation type. Assume that M_i was placed in area $[a_l, a_h] \times [b_l, b_h]$. In the LP formulation, to avoid the situation that another module M_j is to be placed in the area, we set x_{st}^{j} to 0 for any grid point (s,t) in $[a_l - w_j, a_h] \times [b_l - l_j, b_h]$ so that M_j does not overlap with M_i. But, if M_j is compatible with M_i, we ignore the overlapping constriaint so that M_j can be placed in overlapping with $[a_l, a_h] \times [b_l, b_h]$, which consequently, will be interpreted that operations performed by M_i and M_j share a functional module. Specifically, if two modules partially overalp, we make the modules completely overlap by moving the module with less total likelyhood at its anchor point to the location at which the other module is placed only if the move does not increase the worst slack time violation obtained in the LP. The proposed placement-driven resource sharing algorithm, called RS-pd, iteratively performs the LP formation combined with resource sharing. At each iteration we solve the LP formulation, based on which a partial solution of placement and resource sharing is determined. The LP formulation is then updated to reflect the partial solution. We repeat this process until a complete solution is obtained.

Figure 5 shows how the proposed algorithm produces a placement and resource sharing step by step. We assume that the input CDFG contains two multiplication, two addition and two subtraction operations. Further, initially we assume that one multiplication is executed by a slow multiplier (*Mult1*), another by a fast multiplier (*Mult2*) while two additions and subtractions are executed by the same types (size) of adders (*Add1, Add2*) and subtractor (*Sub1, Sub2*) modules, respectively. We assume that the solution of the LP formulation at the first iteration is shown in Figure 5(a). The anchor point with the highest likelyhood value (i.e. x_{jk}^{i} values) is the placement of *Mult1* with likelyhood of 0.7. Consequently, we place *Mult1* at the anchor position. In the second iteration, we update the LP formulation under the restriction that module *Mult1* should be placed only at the shaded area in Figure 5(a). Figure 5(b) shows the values of likelyhood of placement for the remaining modules. We then choose *Mult2* and place it at the shaded position because it has the highest likelyhood, which is 0.4. We repeat this process until all the remaining modules are placed. Figures 5(c)-(g) shows the placement and resource sharing results after the third, fourth, fifth and sixth iterations, respectively. Note that at the fifth iteration, *Add2* is overlapped with *Add1*, indicating resource sharing among the operations assigned initially by *Add1* and *Add2*, and at the sixth iteration, *Sub2* is overlapped with *Sub1*.

The overall flow of the proposed algorithm is summarized in Figure 6. It should be noted that reformulating LP, at each *repeat-*

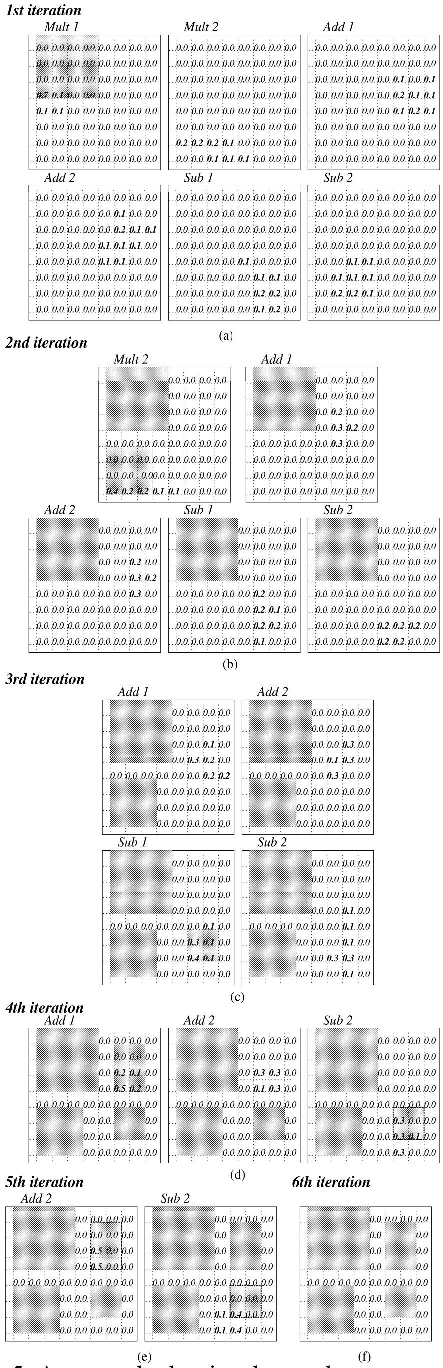


Figure 5: An example showing the step by step procedure

Algorithm RS-pd: *Placement-driven Resource Sharing*
Inputs: ($CDFG$, B, $\mathcal{L}$) /* B: space, $\mathcal{L}$: module library */
• Bind each operation in CDFG to a distinct module instance using $\mathcal{L}$;
repeat {
  • Formulate an LP and solve it; /* section 3.1 */
  • Choose module M_i with the highest x_{jk}^{i} value of its anchor point;
  • Locate M_i to its anchor position (j,k); /* placement */
  if (the area covered by M_i overalps with the area
    by another compatible module M_j) {
    • Merge M_i and M_j to completely /* resource sharing */
    overlap if it does not cause an increase of the worst slack time violation;
  } **endif**
} **until** (all modules are placed)

Figure 6: A summary of the proposed layout-driven resource sharing algorithm.

loop, reflecting previous placement and resource sharing results can be done by setting some x^i_{st} as zero: Suppose that the area of newly placed module M_j at the previous iteration is $[a_l, a_h] \times [b_l, b_h]$. We set $x^i_{st} = 0$ for $(s,t) \in [a_l - w_i, a_h] \times [b_l - l_i, b_h]$ for M_i that is not compatible with M_j. Moreover, since only the modules compatibles with M_j can be positioned at $[a_l, a_h] \times [b_l, b_h]$, the probability that the compatible modules can be positioned at $[a_l, a_h] \times [b_l, b_h]$ becomes much higher in the succeeding iterations. That is, if we assume that M_i is compatible with M_j and M_j has already been placed in the previous iteration, the value of $x^i_{s,t}((s,t) \in [a_l, a_h] \times [b_l, b_h])$ will be relatively high in the current and next iterations, possibly causing a very active resource sharing with M_i.

Arithmetic Expressions	#violation / long_violation / #module	
	Conventional	RS-pd
Exp.1	2 / 4 ns / +8	0 / 0 ns / +8
Exp.2	2 / 3 ns / +3, *5	0 / 0 ns / +5, *5
Exp.3	1 / 3 ns / +3, *5	0 / 0 ns / +3, *5
Exp.4	3 / 4 ns / +5, *3, ÷3	0 / 0 ns / +7, *3, ÷3
Exp.5	2 / 4 ns / +3, *3, ÷2	0 / 0 ns / +5, *3, ÷2

Table 1: Result for arithmetic expressions. Exp.1 : $X_1 + \cdots + X_{20}$; Exp.2 : $X_1 \cdot X_2 + X_3 \cdot X_4 + \cdots + X_{19} \cdot X_{20}$; Exp.3 : $(X_1 \cdot X_2 \cdot X_3 + X_4 \cdot X_5 - X_6 - X_7 \cdot X_8) \cdot X_9 + X_{10} \cdot X_{11}$; Exp.4 : $X_1/X_2 + X_3/X_4 + X_5/X_6 + X_7 \cdot X_8 \cdot X_9 + (X_{10} + X_{11}) \cdot X_{12} \cdot X_{13} + X_{14} 14 \cdot X_{15} + X_{16} - X_{17}$; Exp.5 : $X_1 \cdot X_2 \cdot X_3 + X_4 \cdot X_5 + X_6 \cdot X_7 + X_8 - X_9 + X_{10} \cdot X_{11} + X_{12} \cdot X_{13} + X_{14}/X_{15} + X_{16}/X_{17} + X_{18} \cdot X_{19} + X_{20}$

4 Experimental Results

We conducted a set of experiments to verify the effectiveness of the proposed placement-driven resource sharing algorithm RS-pd. We tested RS-pd on a set of arbitrary arithmetic expressions in Table 1 as well as high-level synthesis benchmark designs [13, 14]. All experiments were performed using a Pentium-3 500MHz Linux machine with 1 GB Memory. We used ILOG CPLEX 7.0[17] to solve LP formulations. We assumed that the functional modules for addition, multiplication, and division operations take 4ns, 38ns, and 75ns for execution, respectively. We used a simple resource driven list scheduling heuristic for each testcase to get a scheduled CDFG. We equally partition a given placement space, and assume that the physical size of the rectangular functional modules of addition/subtraction, multiplication, division are $(1,1)$, $(3,3)$, and $(7,7)$, respectively. Finally, we assume that the delay for one unit-length interconnect is 1ns, the delay for one control step is 10ns, and the interconnect delay is linearly proportional to its length. In fact, if the given placement area is sufficiently large, our algorithm always produces resource sharing and placement result that satisfies the slack time constraint using the same number of modules as the operations in the CDFG.

• **Synthesizing arithmetic designs:** We tested our algorithm using a set of various arithmetic expressions, and compared our results with that produced by the conventional approach in which resource sharing based on a maximal-clique finding heuristic was applied, and subsequently, placement based on Fiduccia and Mattheyses's partitioning heuristic [15] was applied. The results are summarized in Table 1. *#violation* indicates the number of clock steps having slack time violation, and *long_violation* indicates the worst slack time beyond clock period, i.e., $D_{violation}$ in section 3.1. We assume that the size of addition modules used in Exp.1 is (3,3). The comparisons indicate that our resource sharing combined with a global estimation of placement based on LP formulation significantly reduces the slack time violations over the conventional method.

• **Synthesizing filter designs:** We tested RS-pd using high-level synthesis benchmark designs [13, 14], and the results are summarized in Table 2. We assume that the size of adder module in ELLITPIC is $(3,3)$. Comparisons indicate that, for the case of design SPLINE, the worst slack time violation is too large, causing a very tight timing budget in the placement and routing stage later. Overall, the proposed high-level resource sharing approach using global placement information produces resource sharing results with little or no slack time violation, which considerably helps to achieve timing closures between high-level synthesis and physical layout.

Filter Designs	#violation / long_violation / #module	
	Conventional	RS-pd
Complex	0 / 0 ns / +2, *2	0 / 0 ns / +2, *2
DIFF.2	0 / 0 ns / +3, *3	0 / 0 ns / +3, *3
IDCT	0 / 0 ns / +6, *4	0 / 0 ns / +6, *4
FFT.2	4 / 3 ns / +4, *6	0 / 0 ns / +6, *7
Wavelet	0 / 0 ns / +3, *4	0 / 0 ns / +3, *4
SPLINE	3 / 7 ns / +6, *3, ÷3	0 / 0 ns / +7, *3, ÷3
ELLIPTIC	3 / 1 ns / +7	0 / 0 ns / +7
FFT for Real.	4 / 3 ns / +4, *6	0 / 0 ns / +6, *6

Table 2: Result for filter benchmark designs

5 Conclusions

In this paper, we presented a new global layout-driven resource sharing approach in high-level synthesis. As the interaction between architectural/logical (synthesis) and physical (layout) domains becomes much important issue in DSM technology, considering layout effects at an early stage of the synthesis process is crucial. In that respect, the proposed approach, which (Step 1) relaxed the integrated resource sharing and placement into an efficient linear programming formulation, and (Step 2) derived a feasible solution from the (global) solution obtained in (Step 1), can be used effectively in earlier stage of the synthesis process. From a set of benchmark designs, we confirmed that the proposed approach produced much shorter slack time violations, by taking into account the potential interconnect delays, over conventional method, which leads to significantly reliable (relaxed) timing budgets in placement and routing stage.

Acknowledgment This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center(AITrc).

REFERENCES

[1] J. Weng and A. Parker, "3D Scheduling: High-Level Synthesis with Floorplanning," *Proc. of DAC*, 1991.

[2] S. Tarafdar, M. Leeser, and Z. Yin, "Integrating Floorplanning in Data Transfer Based High-Level Synthesis," *Proc. of ICCAD*, 1998.

[3] P. Prabhakaran and P. Banerjee, "Parallel Algorithm for Simultaneous Scheduling, Binding and Floorplanning in High-Level Synthesis," *Proc. of ISCS*, 1998.

[4] V. Moshnyaga and K. Tamaru "A Placement Driven Methodology for High-Level Synthesis of Sub-Micron ASIC's," *Proc. of ISCS*, 1996.

[5] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-Placed RTL Synthesis with Performance-Driven Placement," *Proc. of ICCAD*, 2001.

[6] H. Jang and B. Pangrle, "A Grid-Based Approach for Connectivity Binding with Geometric Costs," *Proc. of ICCAD*, 1993.

[7] M. Munch, N. Wehn, and M. Cleasner, "Optimum Simultaneous Placement and Binding for Bit-Slice Architecture," *Proc. of DAC*, 1995.

[8] Y. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning," *Proc. of ICCAD*, 1994.

[9] A. Bar-Noy, S. Guha, J. Naor ,and B. Schieber, "Approximating the throughput of multiple machines under real-time scheduling," *Proc. of ACM Symposium on Theory of Computing*, 1999.

[10] M. Garey and D. Johnson, *Computers and Intractability*, Freeman Pub.Comp., San Francisco, 1979.

[11] B. Baker, E. Coffman, and R. Rivest, "Orthogonal Packing in Two Dimensions," *SIAM J. Computing*, pp. 846-855, 1980.

[12] E. Coffman, M. Garey, D. Johnson, and R. Tarjan, "Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms," *SIAM J. Computing*, pp. 808-826, 1980.

[13] N. D. Dutt, "High-Level Synthesis Design Repositories", *http://www.ics.uci.edu/ dutt*.

[14] W. Press, S. Teukolsiy, W. Vetterling, and B. Flannery, "Numerical Recipes in C", *Cambridge University Press*.

[15] C. Fiduccia, R. Matteyses, "A Linear-time Heuristic for Improving Network Partitions", *Proc. of DAC*, 1992.

[16] C. Lee, "An Algorithm for Path Connections and its Applications", *IRE Trans. on Electronic Computers*, VEC-10. pp. 346-365, 1961.

[17] http://www.ilog.com