Effective Coupling between Logic Synthesis and Layout Tools for Synthesis of Area and Speed-Efficient Circuits

MANDALAGIRI S. CHANDRASEKHAR^{a,*}, ROBERT H. McCHARLES^b and DAVID E. WALLACE^c

^aViewLogic Systems, 47211 Lake View Blvd. Fremont, CA 94538; ^bConsultant, 7 Skylonda Drive, Woodside, CA 94062;
^cMentor Graphics, 1001 Ridder Park Drive, San Jose, CA 95131

Traditionally logic synthesis and layout tools optimize designs without interaction between them. Lack of communication between the two tools often results in inferior post-layout circuit implementations. This paper presents three aspects of coupling synthesis with layout to minimize post-layout area and delay of circuits. It presents two new techniques for computing *net-weights* based on timing slacks, and shows how performance improvement with little overhead in area can be achieved. Secondly, it presents a novel idea of exploiting *logic equivalence* information in circuits to minimize circuit area and delay during layout. An algorithm for computing logic equivalence classes and performing net swapping using the equivalence classes during layout is described. Lastly, it shows the sensitivity of post-layout delays of circuits to wiring models used in synthesis and demonstrates how *resynthesis* techniques can be effectively used to generate good post-layout implementation. Significant reductions in post-layout area and delay on several industrial designs have been observed.

Keywords: Logic Equivalence, Net-Weighting, Logic Synthesis, Post-Layout Optimization, Resynthesis.

1. INTRODUCTION

In the quest for faster and denser circuits, designers attempt to optimize the speed and area of a circuit at several levels of the design process. Two such levels are:

1. The logic level, where the design is represented as an interconnection of logic gates.
2. The layout level, where the design has a geometric representation.

To perform optimization at the logic level, designers have been increasingly using logic synthesis tools. Automatic tools for generating physical layouts have been widely used by designers for a much longer period of time. Generally, optimizations at these levels are done independently of each other. A more global optimization process considering synthesis and layout together can lead to superior circuits. The main purpose of this paper is to show that a symbiotic relationship can be established between logic synthesis and layout tools, and to demonstrate that

*Corresponding author. Tel.: 510-659-4001. E-mail: mchandra@viewlogic.com.

†This work was performed while the authors were with Hewlett-Packard.

this relationship can be exploited to obtain faster and smaller circuits.

Designers use logic synthesis tools to generate logic implementations from the specifications of a design, subject to a set of constraints. The constraints are generally timing and/or area constraints. Synthesis tools may succeed in producing an implementation that satisfies the constraints. However, synthesis tools optimize the circuit at this stage with only rough estimates of the layout effects such as interconnect capacitance. The optimization at this level—called the *pre-layout* level—may not lead to optimal *post-layout* implementations. Often, the post-layout timing of these implementations fails to meet the specified timing constraints. Designers attempt to overcome timing violations by rerouting signals along critical paths to reduce the wiring capacitance, or by modifying the logic along the critical path. Without the aid of automatic tools, these tasks are tedious and time consuming.

Net-weighting has been used as a mechanism to speed up the critical paths in a circuit [1], [2], [3], [4]. The idea behind net-weighting is to assign heavy weights to nets on critical paths, and use the net-weights to guide the routing to reduce the wire lengths of nets. Dunlop *et al.* [1] use net-weighting to improve the speed of synchronous logic by assigning net-weights to signals in the clock path based on the clock skew. The weights on the combinational logic paths are determined by the deviation of the maximum frequency at which the circuit can operate from the desired frequency of operation. Tsay *et al.* [2] use an analytic technique for net-weighting using the zero-slack algorithm [5]. However, in all these papers, net-weighting is considered during the layout phase only.

More recently Pedram [6] has shown a method of coupling synthesis with layout using layout-driven technology mapping. In this technique, during the technology mapping step, the tool looks ahead to potential layout effects, and generates a mapped circuit which considers the layout area and wire delay.

The techniques presented here are different from the ones presented in the previous papers. We still use the synthesis tools in the conventional way for generating circuits, but interchange information in both the forward and backward paths between synthesis and layout to obtain good post-layout implementa-

tions. In this paper, we demonstrate two ways of coupling logic synthesis with layout tools—*net-weighting* and *logic equivalence*. We *forward annotate* the netlist from the synthesis tool with net-weights and logic equivalence classes. We approach the net-weighting technique from a different angle. We link net-weighting with the logic synthesis process, and let the synthesis tool generate a set of net-weights to start the place and route process, based on the critical paths observed in the synthesis tool. We propose two new strategies for automatic net-weighting of circuits. We demonstrate the effectiveness of net-weighting in reducing the delay of circuits over several points in the design space. Further, we show that net-weighting does not always increase the area of the layout as speculated in previous work [1].

We also show that logic equivalence inherent in many circuits can be effectively used to reduce the area and increase the performance. During the logic synthesis phase, the synthesis tool can recognize and mark signals that are logically equivalent. Such signals can be interchanged during the placement stage to achieve a significant reduction in the circuit area, and improvement in performance. This technique does not rely on the technology mapper generating netlists that potentially reduce wire lengths as LILY [6] does. It can be effectively used with any of the technology mappers in commercial synthesis tools.

Finally, we show how the synthesis-layout coupling can be used for resynthesis of circuits to obtain more efficient implementations. We show the potential sensitivity of the quality of circuits produced to the wiring models used in synthesis.

The synthesis-layout loop discussed in this paper has been fully automated. The results presented in this paper are on real industrial designs, and show significant improvements in area and delay with fairly small run times.

2. THE SYNTHESIS-LAYOUT COUPLING

Designers use logic synthesis tools to generate netlist implementations of designs under a set of timing and/or area constraints [7], [8]. The circuit produced by the synthesis tool is optimized at the *pre-layout*

level. At this level, the synthesis and optimization process cannot accurately incorporate the effect of the layout process on the area and delay of the circuit. The layout tools start with the netlist generated by the logic synthesis tools, and attempt to optimize the layout to meet the area and speed constraints. The real goal of the designer is to start with a functional specification of the design and produce a layout which meets area and timing constraints. The disjoint optimization steps, and lack of communication between the synthesis and layout tools often causes suboptimal post-layout circuits. To meet the real objective of producing better post-layout circuits, coupling between logic synthesis and layout tools is necessary.

The coupling between the logic synthesis tool and the layout tool is bi-directional. In the forward direction, the synthesis tool provides the layout tool with timing and functional information, which can be effectively used during the layout process. In the backward direction, the layout tool provides the synthesis tool with wiring capacitance extracted from the layout, which is used to generate improved netlists and more accurate timing information to be fed forward to the layout tool. The effectiveness of this coupling is demonstrated using the Palo Alto Synthesis System (PASS)—a logic synthesis tool used for synthesizing the circuits—and layout tools called HARP and LLAMA.

Fig. 1 is a block diagram that illustrates the coupling between the logic synthesis tool (PASS) and the layout tools. PASS takes a behavioral description of the design, a set of constraints and a wiring model. The wiring model is a mechanism for providing an estimate of the wiring capacitance likely to be encountered when the circuit is placed and routed.

FIGURE 1 Logic synthesis—Layout coupling

PASS generates netlist implementations of the design in EDIF and BDL formats. The netlist can be annotated with the information for forward coupling—net-weights, logic equivalence or wire widths. LLAMA is a standard cell placer which generates a layout using the BDL netlist produced by PASS. HARP is a collection of several layout programs. A channel router, called ROUTE, and a capacitance extractor available within HARP are used for detailed routing and capacitance extraction. The extracted wiring capacitance can be fed back to PASS to back-annotate the netlist. The static timing analyzer in PASS is used for timing analysis. An estimate of the extracted wiring capacitance is fed back to PASS to resynthesize the design using a more accurate wiring model.

3. NET-WEIGHTING USING PASS

DEFINITION 1: The *Net-weight* of a net is a number that defines the relative importance of that net with respect to other nets in the circuit for the purposes of reducing the wire length. It is a number that *multiplies* the actual length of a net in computing the effective total wire length of a placement.

Note that this technique uses net-weights to minimize the weighted wire length, unlike other techniques that use net-weights for determining the priority of routing. For example, if the net-weight of *A* is twice as large as the net-weight of *B*, then LLAMA will be willing to trade up to two units of additional length on net *B* to achieve one unit reduction in length of net *A*. The relative weights of the nets are what matter, not the absolute values. Thus, if all nets in a circuit have the same weight, they will be treated alike for placement and routing no matter what the actual value of the net-weight is.

DEFINITION 2: The *Slack* associated with a signal is defined as the difference between the time at which the signal is required to arrive, and the time at which it actually arrives.

A negative slack on a signal means that the signal actually arrives later than it should. In such cases there is a timing violation in the circuit. The designer would like to keep the slacks of all signals positive.

3.1 Computation of Net-weights in PASS

PASS has a built-in static timing analyzer. When PASS synthesizes a netlist, it uses estimates of the wiring capacitance of nets to determine the critical paths in the circuit, and compute the timing slacks of all signals in the circuit. This knowledge can be imparted to the layout tools by means of net-weights.

The net-weights of a circuit are computed based on the slacks of the signals. When the slacks are negative, the circuit has timing violations on the corresponding nets. Computation of net-weights and placement using the net-weights must be done so that the slacks are all positive. There are two alternatives for terminating the net-weighting process:

1. Stop the net-weighting as soon as all slacks are positive.
2. Continue net-weighting even when all nets have positive timing slacks until the maximum possible improvement in delay is obtained.

We use the latter approach to determine how much improvement net-weighting can provide. We can then use this information to resynthesize the design by relaxing the timing constraint appropriately. We will discuss more about this in Section 5 on resynthesis. It is, of course, possible to stop the iterative process as soon as the timing slacks become positive. Since the process is heuristic, we want to find out which net-weighting techniques are effective. The essence of our approach is the following:

1. Net-weights of a large number of nets are computed *automatically*.
2. Net-weights are a function of the slacks of each net, but each net is weighted not only based on the slack but also on a factor dependent on the ratio of the wiring delay to the intrinsic delay through the gate connected driving the net.
3. Intuitively, nets with less slack must be assigned large weights.
4. Net-weights are recomputed and updated based on the results of static timing analysis after placement and routing.

3.2 Strategies for Net-weighting in PASS

Given the desire to assign large net-weights to nets with small slacks, how should we do the actual assignment? We first recognize that an inverse relationship exists between net-weights and slacks. The nets are sorted by their slacks, and the slacks are normalized. To investigate effective approaches, we use the concept of a weighting function. A weighting function is a function that determines the weight of a net as a function of its slack. We investigate two weighting functions. The first weighting-function is the *inverse function*, $f(t_s) = 1/t_s$, where t_s is the normalized slack time of the net. The second weighting function is an *inverse-square function*, i.e., $f(t_s) = 1/t_s^2$. The motivation for considering the inverse-square function is that it produces a distribution in which net-weights taper off rapidly. That is, nets with very small slacks are weighted very heavily, while nets with larger slacks will be weighted much more lightly.

Another factor that affects the relative weights is the initial weights assigned to nets. Two different initial values are considered for the experimental results reported in this paper. The combination of two weighting functions and two initial weights gives four different weighting strategies as shown in Table I.

3.3 Algorithm to Compute Net-weights

In this section we describe an algorithm used to compute the net-weights. This algorithm is used iteratively in the loop shown in Fig. 2. In the first iteration estimated wiring capacitances are used for determining the timing slacks. In the subsequent iterations capacitances extracted from the layout are back-annotated onto the circuit before computing the net-

TABLE I Strategies for net-weighting

Strategy	A	B	C	D
Initial weight	1	100	1	100
Weighting function	$1/t_s$	$1/t_s$	$1/t_s^2$	$1/t_s^2$

FIGURE 2 Net-weight computation loop.

weights. In the actual implementation the number of iterations can be controlled by the user.

Algorithm Compute-net-weights (circuit)

```

{
1. Initialize net-weights to selected value (1 or 100)
   in the first pass through this loop;
2. Perform static timing analysis on circuit and compute
   slacks;
3. Normalize the slacks;
4. Sort nets in increasing order of normalized slacks;
   /* Net with smallest slack needs to be weighted heavily
   and is considered first in the weighting process */
5. For the top half of the nets in the sorted list
6.   Net-wt := net-wt + (weight-function* (wire
   delay/intrinsic delay));
}
  
```

Initially all nets are assigned the same net-weight. For our experiments this value was 1 or 100. In steps 2 and 3 static timing analysis and slack computation are done. In step 4, nets are sorted in the increasing order of normalized slacks so that nets with the smallest slacks, i.e., nets that are most critical, are selected first for net-weighting. We believe that at most 50% of the nets can be improved with net weighting. So the net-weights of the top 50% of the nets are computed in steps 5 and 6. The weight of a net in the current iteration is determined by the weighting function and

also by the ratio of the wire delay to the intrinsic delay of the gate driving the net. This is done to ensure that the nets in which wire delay dominates are assigned larger weight. In step 6, the weight of the net in the current iteration is added to the current weight of the net so that the effect of net-weights in the previous iterations is carried forward.

3.4 Usage of Net-weights During Layout

The placement algorithm used in LLAMA is based on the force-directed placement technique [10]. But here, the forces on the nets (used to determine the relative positions of cells in the placement) are proportional to the net-weights. Thus, nets with large weights exert a large force on the cells they connect to, thereby placing the blocks closer. LLAMA does not use net-weights for determining the priority order for routing. Rather, it uses the net-weights to minimize the *weighted wire length*. LLAMA uses net-weights in attempting to minimize weighted wire length.

3.5 Experimental Results with Net-weighting

In this section, we describe some experiments performed to study the effect of applying the net-weighting strategies on the post-layout delay and area of circuits. For these experiments, we use two designs, CKT1 and CKT2. Several implementations of these designs were synthesized with several different timing constraints using PASS. A CMOS standard-cell library was used as the target technology. These implementations vary in complexity from about 1000 gates to 4000 gates.

The iteration loop used for net-weighting is shown in Fig. 2. A netlist is synthesized from the design specifications. Its net-weights are computed using one of the net-weighting strategies. The netlist is placed and routed. The circuit is back-annotated with the wiring capacitances extracted from the layout. The circuit delays and slacks are recomputed. The net-weighting computation is repeated using the new

values of the slacks. The area of the layout reported by HARP, and the circuit delay reported by PASS are used to assess the effect of net-weighting.

3.5.1 Comparison of Net-weighting Strategies

Table II shows the results of applying net-weighting using the four net-weighting strategies on one implementation each of CKT1 and CKT2, denoted by CKT1-1 and CKT2-1. Strategies A and B use the weighting function *inverse* with initial weights of 1 and 100, respectively, and strategies C and D use the weighting function *inverse-square* with initial weights of 1 and 100 respectively. The table shows that net-weighting reduces the delay for all four strategies. The strategies using the initial weight of 1 yield larger areas than strategies using the initial weight of 100 (strategies B and D). Strategy B yields layouts which are slightly smaller than the layouts without net-weighting. For CKT1-1, strategy D provides the greatest reduction in delay. For CKT2-1, strategy B results in the greatest reduction, but strategy D comes very close. Based on these observations, the strategies with initial weight of 100 were selected for further experimentation.

3.5.2 Impact of Net-weighting on Area-Delay Curves

The results presented in the previous section showed that net-weighting effectively reduced the delay for

one data point in the area-delay design space. To study the effectiveness of the net-weighting strategies at several data points in the design space, twelve implementations of CKT1 and CKT2 were synthesized by specifying six different timing constraints for each design. The net-weighting iteration loop was applied to each implementation using strategies B and D. Table III shows the post-layout area and delay without net-weighting, and with net-weighting using strategies B and D. The percentage change in area and delay is computed with respect to the un-weighted implementations. Figs. 3 and 4 show the area-delay curves for CKT1 using strategies B and D, respectively. These graphs and tables present some striking results. First, net-weighting effectively reduces the delay for every implementation. Secondly, strategy D is generally more effective than strategy B in reducing the delay. Thirdly, the increase in area due to net-weighting using strategy D is relatively small (about 7% on the average.) Strategy B generally produces smaller improvements in delay, but it does not increase the area as much. In fact, in ten out of twelve cases it decreases the area. CKT2 presents very interesting results. CKT2-1, an implementation synthesized with a timing constraint of 10 ns, failed to meet the timing as seen in Table III. However, implementations synthesized using looser timing constraints produced circuits that were both smaller and faster, even without net-weighting. This illustrates a case where the designer can, in fact, generate better circuits by relaxing the timing constraints used for synthesis. This behavior of CKT2 will be discussed in the section on resynthesis.

TABLE II Post-layout area and delay with net-weighting

Design	No weight	Net-weighting using			
		Strategy A	Strategy B	Strategy C	Strategy D
CKT1-1					
Area	3.48	4.40	3.45	5.04	4.14
(sq.mm)					
Delay	8.78	8.40	8.61	8.77	7.75
(ns)					
CKT2-1					
Area	9.74	11.29	9.33	14.78	10.38
(sq. mm)					
Delay	18.36	15.08	14.26	15.77	14.82

TABLE III Effect of net-weighting using strategies B and D on CKT1 and CKT2

Design	Constr. ns	No Net-weight		Net-weight—Strategy B				Net-weight—Strategy D			
		Area mm ²	Delay ns	Area mm ²	Delay ns	Area % Change	Delay %	Area mm ²	Delay ns	Area % Change	Delay %
CKT1-2	7	4.11	10.12	4.14	8.53	0.0	-15.7	4.25	7.87	3.6	-22.2
CKT1-1	8	3.48	8.78	3.15	8.61	-9.3	-1.9	4.14	7.75	18.8	-11.7
CKT1-3	10	3.18	9.18	3.25	8.88	2.3	-3.6	3.58	8.47	12.7	-9.5
CKT1-4	11	3.20	10.57	3.27	9.81	2.2	-7.2	3.27	8.91	2.1	-15.7
CKT1-5	13	3.11	11.54	3.09	11.20	0.0	-2.9	3.34	9.83	7.6	-14.8
CKT1-6	20	1.72	13.19	1.73	12.23	0.0	-7.3	1.77	11.28	3.1	-14.5
CKT2-1	10	9.74	18.36	9.33	14.26	-4.1	-22.3	10.38	14.82	6.6	-19.3
CKT2-2	11	4.65	12.55	4.50	11.62	-3.2	-7.4	4.63	11.80	0.0	-5.8
CKT2-3	12	2.54	11.41	2.53	10.80	0.0	-5.2	2.66	10.46	4.7	-8.3
CKT2-4	13	1.55	10.57	1.55	10.20	0.0	-3.5	1.75	10.00	13.1	-4.5
CKT2-5	14	1.55	11.16	1.51	10.69	-2.3	-4.2	1.64	10.25	5.8	-8.1
CKT2-6	15	1.57	10.96	1.50	10.66	-4.0	-2.7	1.71	10.25	9.3	-6.5

3.5.3 Convergence of the Iteration Process

Improvement of circuit delay by using net-weighting is an iterative process. Designers would like to know how many iterations are necessary to get a reasonable improvement. Fig. 5 is a plot of the best circuit delay as a function of the number of iterations. It can be noticed that good improvement can be obtained within the first few iterations. The net-weighting loop is quite fast. For an implementation of CKT1 with

more than 2000 cells, 8 iterations through the loop took 5 hours and 33 minutes of *elapsed* time in a multi-user environment. This includes the time to synthesize, compute net-weights, and analyze circuits which were done on an HP9000 Series 370 (a 68030-based machine), and the time to place, route, and extract capacitance from the layout which was done on an HP9000 Series 845. The largest run time observed was 15 hours (*elapsed time*) to run 16 iterations on an implementation of CKT1.

FIGURE 3 CKT1: Net-weighting using strategy B.

FIGURE 4 CKT1: Net-weighting using strategy D.

FIGURE 5 Delay improvement with number of iterations for CKT1-1.

4. EXPLOITING LOGIC EQUIVALENCE IN CIRCUITS

In this section, we introduce the idea of logic equivalence in circuits, and show how logic equivalence can be effectively used to reduce the post-layout area and delay of circuits. This technique is general enough to be used on circuits synthesized using any technology mapper, including layout-driven technology mappers such as LILY. Using this technique, the synthesis tool can generate and forward-annotate the netlist with the logic equivalence information. This process is one of marking certain signals as equivalent. This information can be used by the place and route tool to interchange equivalent signals, which results in significant reduction in circuit area and improvement in circuit performance.

4.1 Existence of Logic Equivalence

Two types of equivalence, collectively called *logic equivalence*, can be identified in circuits—*output equivalence* and *input equivalence*. Many circuits, whether synthesized by automatic logic synthesis tools or by other methods, contain many internal nodes (outputs of logic gates) that implement the same boolean function. Nodes that implement the same logic function are considered *output equivalent*. At first sight, it appears that the presence of logic equivalence in a circuit is an indication that logic minimization was not properly carried out during logic synthesis. But this is not necessarily the case.

Often, implementations based on the minimized boolean functions result in some gates with large fanouts, which result in large capacitive loads at the gate. Large capacitive loads cause two problems—(i) large delays, and (ii) large current densities, which can give rise to electromigration problems. Synthesis systems often attempt to limit the fanouts of gates to reasonably small values by replicating logic. Fig. 6(a) shows a situation in which gate G_1 is required to drive 15 other gates. Fig. 6(b) is a modified implementation in which three copies of the gate G_1 are created by limiting the fanout of each gate to 5. The outputs of the gates G_{11} , G_{12} and G_{13} are output equivalent.

Output equivalence can also be found in buffer trees. In many large circuits, some primary inputs are heavily loaded, i.e., their fanouts are very large. To limit the fanout of each primary input to a reasonable value, a tree of inverters rooted at the primary input is constructed. One such tree is shown in Fig. 7. It can be seen that the outputs of G_1 , G_2 , G_7 , G_8 , G_9 and G_{10} are output equivalent. Equivalence in buffer trees has been exploited independently by Brasen [9].

FIGURE 6(A) Circuit with large fanout.

FIGURE 6(B) Example of output-equivalence.

Existence of input equivalence in a circuit is illustrated with the example shown in Fig. 8. The inputs of gates G_1 are driven by A and B, and the inputs of G_2 are driven by C and D. The signals A, B, C and D are said to be input equivalent since the output of gate G_3 does not change if the circuit configuration is changed by interchanging the drivers connected to the gates. For example, we could drive G_2 with A and G_1 with C. The logic function at the output of gate G_3 remains the same. There are several other circuit structures that exhibit input equivalence.

FIGURE 8 Example of input-equivalence.

4.2. Use of Logic Equivalence

We illustrate how to effectively exploit the existence of logic equivalence in a circuit using the in-house tools, PASS and LLAMA. Designs implemented using other logic synthesis tools can be fed into PASS for computing the logic equivalence.

Consider the circuit of Fig. 6(b) in which the outputs of gates G_{11} , G_{12} and G_{13} are logically equivalent. The output signals of these gates can be grouped into an equivalence class $\{G_{11}, G_{12}, G_{13}\}$. PASS generates and supplies all such equivalence classes in the circuit to LLAMA with the understanding that LLAMA can interchange connections within an equivalence class if such an interchange can reduce wire length. Suppose, for this circuit, LLAMA generates the placement topology as shown in Fig. 9(a). In particular, note the long wires from G_{11} to G_2 , and from G_{12} to G_3 . A placement such as this can easily

result because of other placement constraints on the blocks not shown in this figure. Since the signals G_{11} , G_{12} , and G_{13} are known to be in the same equivalence class, any of them can be used to drive any of the gates G_2 , G_3 and G_4 . The effect of interchanging nets driving G_2 and G_3 is shown in Fig. 9(b). The decrease in wire length, and the potential decrease in the number of tracks in the channel can be noticed. This shows that net swapping based on logic equivalence is a promising technique to reduce the wiring capacitance and the circuit area.

4.3 Avoiding Cycles During Net-Swapping

While identifying equivalence classes, care must be exercised to ensure that the modified netlist remains acyclic, i.e., signals which could make the circuit cyclic after net-swapping are not included in the equivalence class. For example, consider the buffer tree shown in Fig. 7. The outputs of gates G_1 and G_7 implement the same function and hence they are logically equivalent. But we do not want to put them in the same equivalence class, as swapping these nets could create a combinational loop between gates G_3 and G_7 . Our algorithm for determining logic equivalence in circuits ensures that the equivalence classes do not contain any nets that can cause cycles. Further, the algorithm allows net swapping only between nets at the same logic level, to ensure that the original timing-driven structuring introduced by the synthesis

FIGURE 7 Buffer tree.

FIGURE 9(A) Original layout.

FIGURE 9(B) Layout after net-swapping.

tool is preserved. This is important for timing-driven designs, but if minimization of the area is the only criterion this restriction could be relaxed.

4.4 Algorithm to Find Logic Equivalence Classes

The logic equivalence algorithm implemented in PASS finds a useful subset of the output equivalence relationships in a combinational circuit. This subset includes the logic equivalence arising from logic replication or buffering in circuits generated by PASS. The algorithm considers only those output nodes that are both logically and structurally equivalent to one another, allowing for the possibility of input pin swaps that may have been performed on individual gates. Two nodes are considered *structurally equivalent* if

1. they are both driven by gates with the same logic function, and
2. some legal set of pin swaps can be performed on one of the gates so that corresponding input pins are driven by nodes that are themselves structurally equivalent.

This algorithm finds nodes that were originally identical, irrespective of any pin swapping or gate resizing that may have occurred since the nodes were

split. The algorithm does not examine input equivalence relationships such as those shown in Figure 8.

The function of a logic gate in PASS is represented symbolically with nested lists using prefix notation, using the operators ! for NOT, * for AND, and + for OR. Thus an AOI33 cell (which is a 3-wide and-or-invert cell) would have a symbolic function:

$$(!(+(* a b c) (* d e f)))$$

All occurrences of gates with the same logic function have the same symbolic functions, regardless of drive strength.

Each equivalence class has associated with it a numerical index and a symbolic function, which expresses the logic function of the driving gate in terms of the equivalence classes of its input nets. Each net is annotated with the index of its equivalence class as the latter is computed. Thus the equivalence classes are derived indirectly, consisting of all nets with the same index. Permutable terms of the symbolic function are sorted to put all structurally equivalent functions into a canonical form. For example, for an instance of the AOI33 above, if the input pins *a* through *f* belong to equivalence classes 17, 5, 12, 8, 3, and 7, respectively, the symbolic function after substitution and sorting would be:

$$(!(+(* 3 7 8) (* 5 12 17)))$$

The sorting is performed “inside-out”—operands of inner lists are sorted first, then operands of outer lists are sorted in lexicographical order.

Figure 10 shows a simple circuit with five equivalent

FIGURE 10 Illustration of logic equivalence algorithm

lence classes. These are shown in Table IV. The symbolic function for primary input nets is a singleton list consisting of the index of the equivalence class of that input (in our implementation every input belongs to a unique equivalence class, but this could be easily generalized to cover logically equivalent primary inputs). An outline of the algorithm follows.

The algorithm partitions the nets of the circuit into equivalence classes. Initially, each input net is placed in its own equivalence class. These nets are the initial members of set S . The set T consists of nets which have not yet been placed in an equivalence class. Initially T consists of all nets (except primary inputs) in the circuit. The algorithm proceeds via a topological sort of the circuit, moving nets from set T (those nets whose equivalence class has not yet been determined) to set S as the equivalence classes are found.

Algorithm logic-equivalence(circuit);

- ```
{
1. $S \leftarrow$ the primary input nets of circuit;
2. Each primary input net is placed in its own
 equivalence class;
3. $T \leftarrow$ all other nets of circuit;
4. while (T is not empty) {
5. Find a cell C with all its inputs in S and its
 output, N , in T ;
6. If no such C exists, then exit /* all remaining
 nets in T are either part of a combinational
 loop or are driven by such a loop */
 Else {
7. Get the symbolic function of cell C ;
8. Substitute the equivalence classes of the in-
 put nets for the symbolic variables of the
 function;
9. Sort the permutable terms of the function;
```

- ```
10.    If (the resulting symbolic function is associ-
        ated with an existing equivalence class  $E$ ),
        put  $N$  in  $E$ ;
11.    Else put  $N$  in a new equivalence class,  $E_1$ ;
12.    Move  $N$  from  $T$  to  $S$ , as its equivalence class
        has now been found;
    }
  }
}
```

4.5 Wire Length Reduction Using Equivalence Classes

The logic equivalence classes are annotated on the netlist supplied to LLAMA. Each element in an equivalence class is essentially an output of a gate or a primary input that drives other gate inputs or primary outputs of the circuit. For a 2-pin net there is one driver and one driven signal (load). It is a straight forward extension of the algorithm to handle multiple loads by replicating the driver as many times as it has loads. Recall the description in Section 4.2 that the wire length is reduced by swapping the driven signals between drivers in the same equivalence class. LLAMA uses a recursive slicing algorithm described below. The algorithm performs pair-wise interchange between two loads driven by nets in the same equivalence class to reduce the wire length.

Algorithm *reduce-wire-length* (equiv-class, N)

- ```
{
/* Let N = number of 2-pin nets in the equivalence
class. Then there are N loads and N drivers;
*/
1. Determine the bounding box enclosing loads and
 drivers. Let L_x and L_y be the dimensions of the
 bounding box around the loads, and D_x and D_y
 around the drivers;
2. If $N = 1$ done. Just connect the driver and the load
 Else {
 /* Partition the loads and drivers into two sub-
 sets such that no nets from the class cross be-
 tween the subsets. Each subset contains $N/2$
 drivers and $N/2$ loads; */
```

TABLE IV Equivalence classes of nets in figure 10

| Index | Symbolic Function |
|-------|-------------------|
| 1     | (1)               |
| 2     | (2)               |
| 3     | (! 1)             |
| 4     | (! 3)             |
| 5     | (! (* 3 2))       |

3. If  $L_x + D_x > L_y + D_y$  then
  4. Sort the load and drivers by  $x$ ; Else
  5. Sort the load and drivers by  $y$ ;
  6. Form a sub problem taking the first most  $N/2$  drivers and  $N/2$  loads;
  7. Form a second sub problem by taking the remaining drivers and loads;
  8. Recursively call *reduce-wire-length* on each of the sub problems.
- }
- }

#### 4.6 Results with Logic Equivalence

In this section, we show the effectiveness of logic equivalence in reducing the circuit area and delay. A CMOS standard-cell library is used to generate circuit implementations for a number of designs using PASS. These are large control blocks used in real chips. The circuits have various complexities. The number of inputs varies up to 316, the number of outputs up to 404. The circuits were synthesized using PASS. The output equivalence classes for the circuits are also determined by PASS, and the netlist is annotated with these classes. LLAMA places the netlist and swaps the signals in each equivalence class. After the layout, the wiring capacitance is extracted, and static timing analysis is done using PASS. Table V shows the post-layout area and delay of each circuit with and without using logic equivalence. In this table, the column *Design* lists one implementation each of seven different designs. The second and third columns are post-layout area and

delay of the circuit without exploiting the logic equivalence information in the circuit. The fourth and fifth columns are the post-layout area and delay of the circuit after performing net-swapping based on the equivalence classes. The last two columns show the percentage change in the area and delay by using logic equivalence. The negative entries mean that the corresponding area or delay decreased. It can be seen that both the area and delay decrease significantly for all the implementations considered as a result of using the logic equivalence information inherent in the circuit during layout. Fig. 11 shows two layouts with and without using logic equivalence for net-swapping.

### 5. RESYNTHESIS TECHNIQUES

Automatic coupling of logic synthesis with layout provides a mechanism for guiding the synthesis process to obtain efficient circuits. There are two ways of approaching resynthesis—(i) relaxing timing constraints, (ii) improving the wiring model used for synthesis. These are illustrated in this section using CKT2 as an example.

#### 5.1 Relaxation of Timing Constraints

Table VI shows the pre-layout and post-layout area and delay of two implementations of CKT2. CKT2-1 and CKT2-4 are implementations synthesized with timing constraints of 10 ns and 13 ns respectively. For the constraints specified, CKT2-1 is really the

TABLE V Effect of logic equivalence coupling between PASS and LLAMA

| Design | Without Logic Equivalence |             | With Logic Equivalence |             |                  |       |
|--------|---------------------------|-------------|------------------------|-------------|------------------|-------|
|        | Area<br>sq.mm             | Delay<br>ns | Area<br>sq.mm          | Delay<br>ns | Area<br>% Change | Delay |
| CKT1-1 | 3.26                      | 8.76        | 2.94                   | 7.62        | <b>-15.0</b>     | -13.0 |
| CKT2-3 | 2.49                      | 11.24       | 1.99                   | 10.23       | <b>-20.0</b>     | -9.0  |
| CKT3   | 1.55                      | 9.99        | 1.27                   | 9.42        | <b>-18.1</b>     | -5.7  |
| CKT4   | 1.10                      | 8.40        | 0.77                   | 7.83        | <b>-30.0</b>     | -6.8  |
| CKT5   | 5.41                      | 13.74       | 3.14                   | 11.03       | <b>-41.9</b>     | -19.7 |
| CKT6   | 5.15                      | 8.67        | 4.76                   | 8.37        | <b>-7.6</b>      | -3.5  |
| CKT7   | 5.33                      | 11.68       | 4.82                   | 10.94       | <b>-9.6</b>      | -6.4  |

FIGURE 11 Layout of CKT1: (a) Without logic-equivalence, (b) With logic equivalence

faster circuit at the pre-layout stage. However, CKT2-4 is faster when the delay is measured after layout. Not only is it faster, CKT2-4 also meets the timing constraint of 10 ns, while CKT2-1 does not. The reason for this behavior is that the synthesis tool is working too hard to meet the timing constraint at pre-layout stage. Generally, synthesis tools produce larger circuits when the timing constraints are aggressive. In this case CKT2-1 has far more cells than CKT2-4. Consequently, the layout of CKT2-1 is much bigger, which leads to large wiring capacitance and larger delay. Thus, trying to be greedy to meet the timing constraints at the pre-layout level is not always a good idea. Some pre-layout timing violations can be tolerated, particularly since the synthesis-layout coupling can provide some improvement in delay. Discarding circuit implementations solely based on their pre-layout delays can be undesirable.

## 5.2 Sensitivity to Wiring Model Used for Synthesis

The pre-layout wiring model used by synthesis tools is a crude approximation for the actual wiring capacitance in the circuit. This is necessarily so because the first time a circuit is synthesized, there is no layout

TABLE VI Comparison of pre-layout and post-layout data for CKT2

|        | Cells | Pre-layout |       | Post-layout |       |
|--------|-------|------------|-------|-------------|-------|
|        |       | Area       | Delay | Area        | Delay |
| CKT2-1 | 3845  | 1.10       | 11.04 | 9.33        | 14.26 |
| CKT2-4 | 901   | 0.30       | 12.21 | 1.76        | 10.0  |

associated with it. PASS uses a model in which the estimated wiring capacitance increases linearly with fanout. Many decisions made during synthesis and optimization are dependent on the circuit delay which is influenced by the estimate of the wiring capacitance used during synthesis. The wiring capacitance is not only a function of the technology but also of the size of the circuit and the layout tools used. Table VII shows the sensitivity of post-layout area and delay when the wiring capacitance per fanout used is changed from 0.1 to 0.05, which is a revised estimate obtained from the layout. For both CKT1-2 and CKT2-1, it can be seen that resynthesis with a smaller estimated wiring capacitance results in a circuit that is smaller, and also almost meets the timing constraints. This demonstrates that over constraining a design for synthesis is not desirable. Being somewhat conservative in choosing the constraints can help meeting the desired goals.

While the resynthesis process built into the PASS-HARP/LLAMA loop is general, the following strategy is recommended. For initial synthesis, the estimated wiring capacitance is set to zero. The process is carried out all the way through routing and capacitance extraction. Based on the values of the extracted capacitance, the wiring model is revised, and the circuit is resynthesized using this model.

## 6. CONCLUSIONS

In this paper, we have demonstrated that automatic coupling between logic synthesis and layout tools is effective in reducing the delay and area of circuits. We have presented two new strategies for net-weight-

TABLE VII Sensitivity of post-layout delay and area to the wiring model used for synthesis

|                         | Wiring Capacitance per fanout |             |                         |             |
|-------------------------|-------------------------------|-------------|-------------------------|-------------|
|                         | 0.1 pf                        |             | 0.05 pf                 |             |
|                         | Area<br>mm <sup>2</sup>       | Delay<br>ns | Area<br>mm <sup>2</sup> | Delay<br>ns |
| CKT1-2 (Constr. = 8ns)  | 3.87                          | 7.96        | 3.14                    | 8.01        |
| CKT2-1 (Constr. = 10ns) | 10.38                         | 14.82       | 2.81                    | 10.26       |

ing, which are both effective and efficient. The strategy based on the inverse weighting function decreases the delay (by 7% on the average), and also decreases the area (by about 3%). The strategy based on the inverse-square weighting function produces larger improvements in delay (12% average) accompanied by a small increase in area (7% average). Delay improvements as large as 22% have been observed with net-weighting. Significant improvements in delay have been observed using only a few iterations of the net-weighting loop.

We have introduced the concept of logic equivalence in netlists, and developed an algorithm that identifies a useful subset of the output equivalence as defined in this paper. The algorithm has been incorporated into the logic synthesis system PASS, which synthesizes netlists, and annotates them with the logic equivalence classes. We have shown that by interchanging nets in the equivalence classes, it is possible to reduce the wire length, which reduces the area and delay of circuits. The reduction in area is about 20%, and the average improvement in delay is about 9%. The amount of improvement in delay and area that can be obtained depends on the number of equivalence classes in the circuit, and on the number of elements in each class. The smaller improvements in area and delay in circuits CKT6 and CKT7 are attributable to the smaller size of the equivalence classes.

While net swapping using logic equivalence reduces the total wire length, which generally reduces both area and delay, there may be situations in which an individual wire length may increase. If such a wire happens to be on the critical path, then the delay may actually increase, but it is straightforward to prevent it from happening. We can simply annotate some nets (e.g., critical nets) to prevent them from being swapped even if they are equivalent to some other nets, or alternatively, we can delete such nets from the equivalence classes. It is also possible that net swapping may reduce the wire lengths on the critical paths also, but the delay may increase because now there may be much lower-drive cells on the critical path. Rather than preventing net swapping in such a case, it is more appropriate to overcome this problem

using a post-layout cell substitution technique, where low-drive cells are traded for high drive cells as needed. Determination of the input equivalence classes of the type shown in Fig. 8 is similar to the problem of determining symmetric (or partially symmetric) boolean functions. We are looking into extending our algorithm to capture the equivalence classes of this type also.

In our experiments we used a two-layer routing model. Because of this, wire length reductions could result in significant area reductions, since they directly reduce the amount of space devoted to routing between the cells. In technologies with three or more layers of routing, the area reduction is likely to be less dramatic. However, the technique described in this paper should still be useful to reduce the wiring capacitance, and hence power and delay.

We have demonstrated that over-constraining a design, either by specifying very aggressive timing constraints or by using overly conservative wiring models for synthesis, can result in implementations with inferior post-layout area and delay. In fact, since net-weighting and logic equivalence can provide some improvement in delay, the designer can specify looser timing constraints for synthesis, which generally results in smaller circuits. We have also shown that the post-layout area and delay can be very sensitive to the wiring model used for synthesis. Resynthesis by calibrating the wiring model in the synthesis-layout loop can generate smaller and faster circuits.

The synthesis-layout coupling using PASS and HARP/LLAMA has been fully automated to carry out synthesis, layout and resynthesis with or without using net-weighting and logic equivalence. Since both net-weighting and logic equivalence are aimed at reducing the wire length, we expect some interaction between these techniques when used together. We have done some tests in which logic equivalence is followed by net-weighting, and observed that the overall delay reduction is better than using either technique alone, and that the layout area is smaller than the area using un-weighted implementations.

Another application for net swapping using logic equivalence is clock tree synthesis. Structurally, clock trees are similar to the buffer trees generated during

logic synthesis as in Fig. 7. We can mark the equivalence classes in the clock tree and perform net swapping to minimize the skew. Hooking up scan chains to minimize the layout area is another possible application. When scan chains are hooked up (in the pre-layout stage), the order of the connections does not take into account the potential impact on the layout. Since all the scan flip-flops are *equivalent* in a sense, this technique can be investigated to minimize the layout area.

### Acknowledgment

The authors would like to thank Marc Clarke who provided invaluable support in automating the synthesis-layout loop. The authors would also like to thank the other members of the PASS team, Ron Crough, Kathy Hsu, John Privitera and Gregory Weeks for their contributions. Finally, they want to thank Jim Duley and Ken van Bree for providing the support and encouragement during this work.

### References

- [1] Dunlop, A. E., Agrawal, V. D., Deutsch, D. N., Jukl, M. F., Kozak, P., and Wiesel, M., "Chip Layout Optimization Using Critical Path Weighting," Proc. of 21st Design Automation Conference, 1984, pp. 133–136.
- [2] Tsay, R. S., and Koehl, J., "An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement," Proc. of 28th Design Automation Conference, 1991, pp. 620–625.
- [3] Burstein, M., and Youssef, M. N., "Timing Influenced Layout Design," Proc. of 22nd Design Automation Conference, 1985, pp. 124–130.
- [4] Hauge, P. S., Nair, R., and Yoffa, E. J., "Circuit Placement for Predictable Performance," International Conference on Computer-Aided Design, Digest of Technical Papers, 1987, pp. 88–91.
- [5] Nair, R., Berman, C. L., Hauge, P. S., and Yoffa, E. J., "Generation of Performance Constraints for Layout," IEEE Transactions on Computer-Aided Design, vol. 8, no. 8, August, 1989, pp. 860–874.
- [6] Pedram, M. and Bhat, N., "Layout Driven Technology Mapping," Proc. of 28th Design Automation Conference, 1991, pp. 99–105.
- [7] Brayton, R.K., Rudell, R., Sangiovanni-Vincentelli, A., and Wang, A.R., "MIS: A Multiple-Level Logic Optimization System," IEEE Transactions on Computer-Aided Design, vol. 6, no. 6, November, 1987, pp. 1062–1081.
- [8] Bartlett, K., Cohen, W., De Geus, A., and Hachtel, G., "Synthesis and Optimization of Multilevel Logic under Timing

Constraints," IEEE Transactions on Computer-Aided Design, vol. 5, no. 4, October 1986.

- [9] Brasen, D., Schaefer, T., Ginetti, A., and Chu, S., "Post-layout Buffer Optimization," Euro ASIC Conference, 1992, pp. 156–161.
- [10] Hanan, M., Wolff, P.K., and Agule, B. J., "Some Experimental Results in Placement Techniques," Proc. of 14th Design Automation Conference, 1976, pp. 214–219.

### Authors' Biographies

Mandalagiri S. Chandrasekhar received the B.E degree in electronics from Bangalore University, Bangalore, India, in 1971, the M.E. degree in electrical communication engineering (with distinction) from the Indian Institute of Science in 1973, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles in 1984.

From 1973 to 1979 he was with the Hindustan Aeronautics Limited and the Indian Space research Organization in India, where he was involved in the design of avionics systems. From 1984 to 1992 he was with Hewlett-Packard, where he worked in formal verification, and development of a logic synthesis system. From 1992 to 1996 he was with Compass Design Automation, San Jose working in logic and high-level synthesis, and Floor Planning. Since June 1996 he has been with ViewLogic Systems, Fremont, CA. His current research interests include high-level synthesis, logic synthesis, formal verification and physical design.

Dr. Chandrasekhar is a member of the IEEE.

Robert H. McCharles attended the University of California, Los Angeles where he graduated summa cum laude in 1972. He attended graduate school at University of California, Berkeley, where he earned his Ph.D. degree in electrical engineering in 1981. From 1978 to 1992 he was with Hewlett-Packard in Palo Alto. He is an experienced chip designer and author of IC design tools. He is currently an independent consultant.

David E. Wallace (S' 82-M' 84) received the Sc.B. degree in mathematics and computer science from Brown University in 1978, and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley in 1984 and 1988.

He is currently with Mentor Graphics Corporation in San Jose, CA, working in the logic synthesis group. From 1978 to 1981, he worked on automated placement and routing of gate arrays for IBM in East Fishkill, NY. From 1988 to 1993 he worked on logic

synthesis development at Hewlett-Packard. His research interests include logic synthesis, timing verification and physical design.

Dr. Wallace is member of the ACM and Sigma Xi.