Generation of Performance Constraints for Layout

RAVI NAIR, SENIOR MEMBER, IEEE, C. LEONARD BERMAN, MEMBER, IEEE, PETER S. HAUGE, AND ELLEN J. YOFFA, MEMBER, IEEE

Abstract—**In this paper we present methods for generating bounds on interconnection delays in a combinational network having specified timing requirements at its input and output terminals. An automatic placement program which uses wirability as its primary objective could use these delay bounds to generate length or capacitance bounds for interconnection nets as secondary objectives. Thus, unlike previous timing-driven placement algorithms described in the literature, the desired performance of the circuit is guaranteed when a wirable placement meeting these objectives is found. We also provide fast algorithms which maximize the delay range, and hence the margin for error in layout, for various types of timing constraints.**

I. Introduction

THE PERFORMANCE of any digital system is limited by the delay in propagation of signals through the various paths from the input to the output of the system. In a synchronous sequential circuit, the time taken by the signals to traverse the path between two clocked gates should be no greater than the difference in the times of arrival of successive clock pulses at the two gates. In order to simplify analysis, this difference is often approximated as the period of the clock; however, in high-performance systems, the difference must account for other factors such as clock skew. In most digital systems, it is convenient, and fairly accurate, to lump the delay associated with the wire together with the delay associated with the logic components that comprise the system. With such an approximation, the delay of a path through the network may be calculated simply as the sum of the delays through the components along that path. In high-performance systems, however, in order to accurately estimate the delay associated with the wires, it is necessary to know the length, and sometimes the topology, of the interconnection nets driven by the component. This is particularly the case in very large scale integration (VLSI), where the delay associated with the interconnection network may be comparable to the delay due to circuit switching.

Since the length of interconnection nets is not known before actually laying out the components on the chip or board, it is common practice to use an estimate of the interconnection length to compute delays during logic design. After layout, exact lengths are used in order to verify that the system meets performance constraints. If the timing requirements are not met along some paths, an attempt must be made to alter the placement and/or the wiring to correct them. For example, in [1], correction is attempted by exchanging the locations of components along these paths with other components, or by inserting the components in favorable empty locations. Unfortunately, there is no guarantee that a local perturbation to the layout will correct timing errors, especially in high-performance systems where the limits of technology are being pushed by the design. In difficult cases, it is necessary to redesign the logic.

Because of the difficulty in altering a completed layout to improve timing, some researchers [2], [3] have investigated using timing requirements earlier in the physical design process. Since placement of components is generally done iteratively [4], one approach has been to use progressively more accurate timing information as placement proceeds. For example, in [3] timing information is used to evaluate potential placements, and the information is updated whenever components are perturbed during the placement process. One problem with this approach is that, in large circuits, it is not unusual to find that interchanging the positions of two components affects over a thousand paths. This could result in long computation times. Another possibility [2] is to pass timing and layout data periodically back and forth between the timing analysis and placement programs while the layout is being constructed.

Two problems arise in both approaches above. First, timing analysis must be done a number of times—sometimes a large number of times. Second, since the timing information used during layout is continually changing, the layout may converge rather slowly, or worse, may not converge.

A different approach to the problem uses performance requirements as additional direct constraints during layout. Since the length of interconnection, and hence the wirability of the circuit, is optimized by placing components on an interconnection net as close together as possible, it is possible to identify nets along "critical" paths for special treatment: either earlier consideration during the placement process, or assigning higher weight relative to other nets. For example, in [5], it is demonstrated that by using the criticality measure of nets to bias placement and routing programs, the performance of a chip may be brought significantly closer to that of an ideal layout. Similarly, in [6] a hierarchical layout approach is used, with timing computation done at each level of the hier-

Manuscript received September 20, 1988; revised February 17, 1989. The review of this paper was arranged by Associate Editor A. E. Dunlop.

The authors are with the IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

IEEE Log Number 8927950.

archy to determine a criticality class for a net. Each class is associated with a weight which is used to constrain placement at that level of the hierarchy.

The approach outlined in the last paragraph often leads to a layout which meets performance specifications while avoiding the problems mentioned earlier. The principal reason for errors in such an approach is that, in attempting to minimize the length of all nets along a critical path, the problem is often overconstrained to the point where nets which were not critical become excessively long. In this paper, we present a new method for using timing requirements in the placement process. Our method assigns length bounds to *all* wires. This relaxes the constraints to be met by the placement program by permitting it to increase the wire lengths on noncritical paths while ensuring that components on critical paths are close to each other. We introduce the *zero-slack algorithm* [7], which determines bounds for each net based on timing specifications. Our premise is that while the placement problem is a hard one, it is not more difficult to find a wirable placement which satisfies bounds in length for every net, than one which only minimizes total wire length. In fact, it has been shown in [6] that the imposition of such constraints on nets does not lead to less wirable solutions. This is possibly due to the fact that, in practice, when a solution exists, there are several placements which satisfy a given wirability constraint.

Since the set of net length bounds that guarantee performance requirements is not unique, we also investigate the problem of maximizing the range of permissible lengths for each net. We demonstrate the existence of a fast algorithm for maximizing the range when only upper bounds are important. When both lower and upper bounds are important we formulate the problem as a linear programming problem, for which polynomial time solutions have recently been demonstrated [8].

II. Preliminaries

Our model follows that used in [5], [6], and [9]. We assume that, as commonly practised, a given complex sequential circuit with feedback paths may be broken down into sets of clocked latches having a pure combinational circuit without feedback between the various latch stages. We can thus analyze each group of components forming the combinational circuit between latches independently. As in [6], for simplification, we associate with any component a single value for delay derived as a function of two delay values, one for a rising transition and one for a falling transition. The procedure to be developed generalizes to the case when the values have to be considered separately, as shown in [9]. We further assume that all combinations of logic values are possible at the inputs to any component. In practice, we could limit our attention to the subset of values permitted by the logic in order to tighten the timing bounds. However, as shown in [10], this does not affect the timing results significantly.

In a significant departure from [5] and [6] we consider both *late* and *early* modes. In the *late* mode, the required time of arrival of the signal at a specified point is the latest time by which the signal should arrive to ensure proper functioning of the circuit. Conversely, in the *early* mode, the time specified would be the earliest time at which the signal may arrive at that point. (Early mode constraints are needed to accommodate clock skew as well as complex clocking strategies common in high performance design.)

For a given combinational circuit comprised of a set of components $X = \{x_1, x_2, \cdots, x_n\}$ and a set of interconnection nets $E = \{e_1, e_2, \cdots, e_m\}$, we define a mapping

$$\pi: X \times E \rightarrow \{1, 0, -1\}$$

on every component–net pair as follows:

$$\pi(x, e) = \begin{cases} -1 & \text{iff } e \text{ is an input to } x \\ 1 & \text{iff } e \text{ is an output of } x \\ 0 & \text{otherwise } (e \text{ not connected to } x). \end{cases}$$

Each net e is assumed connected to at least two components. The r primary inputs to the circuit are treated as the outputs of r dummy components (I) in X. We will assume that each component has only one output. In the case where a component has more than one output, we simply replicate the component for the required number of outputs, with identical nets feeding each component. The components providing the q primary outputs of the circuit will be referred to as the *output components* (G). We also assume that "wired outputs" do not exist. These may be handled by introducing additional components at the "wired output" points. Thus we have the relationship $n = m + q$, since every component other than the output components is associated with exactly one net. Let $e(x)$ denote the output net of x.

We now define the set of *fan-in* components of a component $x \in X$ as

$$\pi^-(x) = \{x' \mid \pi(x, e(x')) = -1\}$$

and the set of *fan-out* components as

$$\pi^+(x) = \{x' \mid \pi(x', e(x)) = -1\}.$$

The mapping π may be represented compactly as the set of sets, $\Psi(X) = \{\psi(x_1), \psi(x_2), \cdots, \psi(x_n)\}$, where

$$\psi(x_i) = \{\pi^+(x_i) \cup x_i\}.$$

The set $\Psi(X)$ is often called the *net list* or the *interconnection list*. The size of the input is characterized by the size of the net list and is given by p, where

$$p = |X| + |\pi^+(X)|.$$

Each component x, as mentioned in the last section, is associated with a delay $d(x)$, which represents the time required for a change in signal at one of its inputs to result in a change at the inputs to components in $\pi^+(x)$. Thus $d(x)$, besides being a function of the circuitry of component x, is also a function of the layout characteristics of the net being driven by its output. We define the base

delay, $d_0(x)$, as the value of $d(x)$ when the components x and $\pi^+(x)$ are located ideally with respect to each other. One could then view the difference,

$$\delta(x) = d(x) - d_0(x)$$

as the contribution to delay due to the net $e(x)$. Thus, prior to physical design, the sequence of delays $D_0(X) = \langle d_0(x_1), d_0(x_2), \cdots, d_0(x_n) \rangle$ are given constants, while the incremental sequence $\Delta_0(X) = \langle \delta_0(x_1), \delta_0(x_2), \cdots, \delta_0(x_n) \rangle$ are unknown because the routes for segments of the net $e(x)$ are unknown.

For the signal at each output component $x \in G$ we are given two types of arrival times: the required early time, $t_{re}(x)$, defined as the earliest time that a signal may arrive at the output of that component, and the required late time, $t_{rl}(x)$, defined as the latest time by which it must arrive, for proper processing. In addition, for every primary input component $x \in I$, we are given two more arrival times: the actual early time, $t_{ae}(x)$, as the earliest possible time that a change could occur, and the actual late time, $t_{al}(x)$, as the latest possible time that a change could occur at that primary input.

We could extend the definitions of actual late and actual early times to all components in the circuit. These values, $t_{ae}(X)$ and $t_{al}(X)$, may be computed easily, as shown in [9]. Basically, for each $x \in X$ such that $t_{ae}(z)$ and $t_{al}(z)$ are specified for all $z \in \pi^-(x)$, we set

$$t_{ae}(x) = d(x) + \min_{z \in \pi^-(x)} t_{ae}(z)$$

and

$$t_{al}(x) = d(x) + \max_{z \in \pi^-(x)} t_{al}(z).$$

An efficient way of implementing this procedure is to perform a *depth-first search* (see [11]) starting at the primary outputs. The complexity of this procedure is $O(p)$, where p is the size of the interconnection list. Thus, for a bounded fan-out circuit, the procedure is linear in the number of circuit components.

For each circuit output, $g \in G$, we can further compute two values, the early slack, $s_e(g)$, and the late slack, $s_l(g)$, as

$$s_e(g) = t_{ae}(g) - t_{re}(g)$$

and

$$s_l(g) = t_{rl}(g) - t_{al}(g).$$

Negative values for slack imply that the circuit does not meet the arrival time requirements for the specified values of $D(X)$. In particular, if the base delay values, $D_0(X)$, are used, then the wire contributions, $\Delta(X)$, are uniformly 0, and a negative late slack at some output implies that it is impossible for the circuit to meet the performance requirements. In such cases, the logic needs to be redesigned.

We illustrate the calculations in Fig. 1. For convenience, we have shown the calculations for each mode

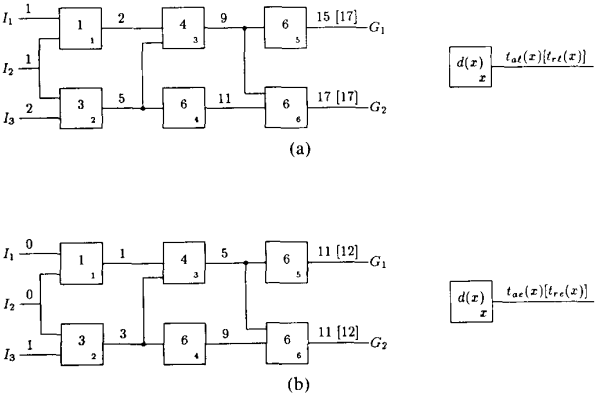

Fig. 1. Sample calculations of the various arrival times. (a) Late mode calculations. (b) Early mode calculations.

separately. We see that the late slacks are 2 and 0 for outputs G_1 and G_2 respectively. This implies that some path to G_2 is *critical* in the sense that no further delay can be tolerated along the path. A closer examination reveals that the critical path is the one starting at input I_3 and passing through components 2, 4, and 6 to output G_2. If the delay values indicated in the boxes are the base values of delay, this further implies that the layout must ensure that the lengths of nets along this critical path are at their shortest possible values. Any further delay on one of these nets would change the slack on G_2 to a negative value.

In Fig. 1(b) we see that the early slacks on both outputs are negative. This means that delay must be added to appropriate nets to make the slacks nonnegative. In adding these delays, it is necessary to ensure that the late slacks do not become negative at any output. For example, increasing the delays on each of components 5 and 6 by unit value would make the early slacks nonnegative, but would result in a negative slack on G_2. On the other hand, it is safe to add unit delay to either component 1 or component 3.

A circuit will be called *late-safe* if, for every output $g \in G$, $s_l(g) \geq 0$, and *early-safe* if $s_e(g) \geq 0$. An assignment of incremental delays $\Delta(X)$ will be termed *late-safe* if the resulting circuit after adding these delay values remains late-safe. A *safe circuit* is one which is both late-safe and early-safe. Finally, an assignment $\Delta(X)$ will be termed safe, if the circuit with delay values $D_0(X) + \Delta(X)$ is safe. In the example of Fig. 1(b), the assignment $\Delta(X) = \{0, 0, 0, 0, 1, 1\}$ is not late-safe, while the assignment $\{1, 0, 0, 0, 0, 0\}$ is safe, because the resulting circuit is both late-safe and early-safe.

III. Zero-Slack Algorithm

Consider the example in Fig. 1(a). We noted that the addition of any further delay to the critical path would result in an undesirable negative slack at output G_2. On the other hand, a layout for the circuit may add delays to other paths without causing a late path at any output. We present in this section an algorithm that determines a safe assignment $\Delta(X)$ which is *maximal* in the sense that any

additional delay on any component x would cause some output to have a negative slack. We call this the *zero-slack* algorithm. The maximal safe assignment allows us to specify a maximum value on the delay contribution of each net. This in turn may be used to guide the placement of the components. Such a set is not unique. In this section we demonstrate how one such set may be calculated. The problem of identifying the best set is addressed in Section VI.

First, the definitions for required early time and required late time which were given for the output components will be extended recursively to all components $x \in X$. If $t_{re}(z)$ and $t_{r\ell}(z)$ are specified for all $z \in \pi^+(x)$, set

$$t_{re}(x) = \max_{z \in \pi^+(x)} t_{re}(z) - d(z)$$

and

$$t_{r\ell}(x) = \min_{z \in \pi^+(x)} t_{r\ell}(z) - d(z).$$

Again, a depth-first search technique starting at the inputs provides a procedure to compute these values in $O(p)$ time, where p is the size of the interconnection list.

In addition, we can compute slack values for every component $x \in X$ as before:

$$s_e(x) = t_{ae}(x) - t_{re}(x)$$

and

$$s_\ell(x) = t_{r\ell}(x) - t_{a\ell}(x).$$

Let us define a *path* as a sequence $\langle x_1, \cdots, x_k \rangle$, $x_i \in I$ and $x_k \in G$, where $x_i \in \pi^+(x_{i-1})$ for $i = 2, \cdots, k$. We define early and late path slacks, $s_e(\rho)$ and $s_\ell(\rho)$ as

$$s_e(\rho) = t_{ae}(x_1) + \sum_{1 < i \le k} d(x_i) - t_{re}(x_k)$$

and

$$s_\ell(\rho) = t_{r\ell}(x_k) - t_{a\ell}(x_1) - \sum_{1 < i \le k} d(x_i).$$

Essentially, the path slacks reflect the slack at the output component assuming no other paths through the circuit are active.

Lemma 1: For any path $\rho = \langle x_1, x_2, \cdots, x_k \rangle$ through the circuit, $s_\ell(x_j) \le s_\ell(\rho)$, $1 \le j \le k$.

Proof: By definition,

$$t_{r\ell}(x_j) \le t_{r\ell}(x_k) - \sum_{i > j} d(x_i)$$

and

$$t_{a\ell}(x_j) \ge t_{a\ell}(x_1) + \sum_{i \le j} d(x_i).$$

Hence,

$$t_{r\ell}(x_j) - t_{a\ell}(a_j) \le t_{r\ell}(x_k) - t_{a\ell}(x_1) - \sum_{1 < i \le k} d(x_i)$$

leading to

$$s_\ell(x_j) \le s_\ell(\rho). \qquad \square$$

One of the implications of the above lemma is that if the circuit is late-safe, then no output component has a negative late slack and no path has a negative late slack. Another implication is that a circuit remains safe if a delay no greater than $s_\ell(x)$ is added to component x, because such a delay decreases the path slack $s_\ell(\rho)$ for any path ρ through x to $s_\ell(\rho) - s_\ell(x)$, which is nonnegative.

Lemma 2: For each component $x \in X$ there is some path ρ_x such that $s_\ell(x) = s_\ell(\rho_x)$.

Proof: Consider the path $\rho_x = \langle x_1, \cdots, x_j = x, \cdots, x_k \rangle$ such that $t_{r\ell}(x_i) = t_{r\ell}(x_{i+1}) - d(x_{x+1})$, for $i \ge j$, and $t_{a\ell}(x_i) = t_{a\ell}(x_{i-1}) + d(x_i)$, for $i \le j$. Such a path must exist by definition of $t_{a\ell}(x)$ and $t_{r\ell}(x)$. By induction,

$$t_{r\ell}(x) = t_{r\ell}(x_k) - \sum_{i \ge j} d(x_{i+1})$$

and

$$t_{a\ell}(x) = t_{a\ell}(x_1) + \sum_{i \le j} d(x_i).$$

Hence,

$$t_{r\ell}(x) - t_{a\ell}(x) = t_{r\ell}(x_k) - t_{a\ell}(x_1) - \sum_{1 < j \le k} d(x_i)$$

leading to

$$s_\ell(x) = s_\ell(\rho_x). \qquad \square$$

Lemma 2 implies that if a delay greater than $s_\ell(x)$ is added to component x, then there is some path ρ_x for which $s_\ell(\rho_x) < 0$. This path passes through the output component x_k. By Lemma 1 $s_\ell(x_k) \le s_\ell(\rho_x)$, leading us to conclude that the circuit is no longer late-safe. We also concluded from Lemma 1 that it is safe to add a delay no greater than $s_\ell(x)$ to component x. Hence we have the following corollary.

Corollary 1: Given a safe circuit, an assignment $\delta(x)$, $x \in X$, is safe iff $\delta(x) \le s_\ell(x)$.

This suggests a procedure to get a maximal set of values for $\Delta(X)$. We could arbitrarily choose a component x having $s_\ell(x) < 0$, augment the current delay value $d(x)$ by $\delta(x) = s_\ell(x)$, recompute the slacks, and repeat the process until no components having positive slack remain.

```
procedure greedy_zero_slack;      {late mode}
{Assume original circuit is safe}
begin
  repeat
    compute_slacks;
    s_min := ∞;
    {Find component with positive slack}
    i := 1;
    while i ≤ n and s_ℓ(x_i) = 0 do i := i + 1;

    if i ≤ n then d(x_i) := d(x_i) + s_ℓ(x_i);
        {Increase delay of i}
  until i > n;      {all nets have zero slack}
end ;
```

The above procedure often leads to situations where the slack on several components along a path become zero even though the delay is lumped with only one component on that path. In contrast to this "greedy" approach, the algorithm shown below distributes delays more uniformly over the components. At each iteration, a component having the least positive slack is selected. The slack is then distributed as delays over all components whose slacks get reduced to zero.

```
procedure zero_slack;      {late mode}
begin
  repeat
    compute_slacks;
    s_min := ∞;
    {Find minimum positive slack}
    for i := 1 to n do
      if (s_ℓ(x_i) < s_min) and (s_ℓ(x_i) > 0) then
        begin s_min := s_ℓ(x_i); x_min := x_i; end ;
    if s_min ≠ ∞ then
    begin     {Find forward path segment}
      a_0 := x_min; v := 0;
      repeat
        find x ∈ π⁺(a_v) | t_rℓ(x) = t_rℓ(a_v) + d(x) and t_aℓ(x) = t_aℓ(a_v) + d(x);
        if x exists then begin a_{v+1} := x; v := v + 1; end ;
      until no such x exists;
      {Find backward path segment}
      u := 0;
      repeat
        find x ∈ π⁻(a_u) | t_rℓ(x) = t_rℓ(a_u) − d(u) and t_aℓ(x) = t_aℓ(a_u) − d(u);
        if x exists then begin a_{u−1} := x; u := u − 1; end ;
      until no such x exists;

      {Distribute slacks}
      s := s_min;
      for i := u to v do begin
        ŝ := s/(v − i + 1);
        d(i) := d(i) + ŝ;      {Increase delay of i}
        s := s − ŝ;
      end ;
    end ;
  until s_min = ∞;      {all nets have zero slack}
end ;
```

While the algorithm is straightforward and easy to implement, a proof that it works is a little long. In order to prove that the algorithm works, we will first show that the process of selecting components and the assignment of delay increments to these components leave the circuit safe. Thereafter we show that at each step, all the selected components have their slacks reduced to zero. Finally we show that no component other than the selected ones can have its slack reduced to zero.

Lemma 3: Given a safe circuit, an assignment $\Delta(\chi)$, $\chi \subseteq X$, is safe if $\Sigma_{x_i \in \chi}\, \delta(x_i) \leq \min_{x_i \in \chi} s_\ell(x_i)$.

Proof: Consider any path ρ through the circuit. Let ϱ denote the set of components in ρ. Since the circuit is assumed safe originally, $s_\ell(\rho) \geq 0$. Let the new slack along path ρ after assignment $\Delta(\chi)$ be denoted by $s'_\ell(\rho)$. Now,

$$\sum_{x_j \in \varrho \cap \chi} \delta(x_j) \leq \sum_{x_i \in \chi} \delta(x_i) \leq \min_{x_i \in \chi} s_\ell(x_i) \leq \min_{x_j \in \varrho \cap \chi} s_\ell(x_j).$$

Hence,

$$s'_\ell(\rho) = s_\ell(\rho) - \sum_{x_j \in \varrho \cap \chi} \delta(x_j) \geq s_\ell(x_j) - \min_{x_j \in \varrho \cap \chi} s_\ell(x_j), \text{ by Lemma 1} \geq 0, \text{ since the circuit is originally safe.}$$

The circuit is safe since every path ρ is safe. □

Lemma 4: Let $\rho = \langle x_1, \cdots, x_u, \cdots, x_v, \cdots, x_k \rangle$ be a path through a circuit such that, for $u \leq i < v$, $t_{r\ell}(x_{i+1}) = t_{r\ell}(x_i) + d(x_{i+1})$ and $t_{a\ell}(x_{i+1}) = t_{a\ell}(x_i) + d(x_{i+1})$. Any assignment of delays to the components $\langle x_u, \cdots, x_v \rangle$ affects the slack on all the components $x_u, \cdots, x_v$ equally.

Proof: By choice of u and v, each of the components $x_u, \cdots, x_v$ must have the same slack. Let ρ_{uv} represent the path through the circuit that has slack $s_\ell(\rho_{uv}) = s_\ell(x_u) = \cdots = s_\ell(x_v)$. By using the construction in the proof to Lemma 2 such a path may be found as follows: First find ρ_u, a path which has slack $s_\ell(x_u)$ passing through x_u. Then find ρ_v, a path which has slack $s_\ell(x_v)$ passing through x_v. Now concatenate the portion up to x_u in ρ_u with portion between x_u and x_v in ρ, followed by the portion after x_v in ρ_v.

By Lemma 1 no other path through any of the nodes $x_u, \cdots, x_v$ has a smaller slack. Any assignment of delays, $\delta(x_u), \cdots, \delta(x_v)$, reduces the slack in path ρ_{uv} by $\Sigma_{u \le i \le v} \delta(x_i)$. Hence, by Lemmas 1 and 2 again, each of the components in $x_u, \cdots, x_v$ must have its slack reduced by $\Sigma_{u \le i \le v} \delta(x_i)$. □

Lemma 5: Let $\rho = \langle x_1, \cdots, x_u, \cdots, x_v, \cdots, x_k \rangle$, $s_\ell(\rho) > 0$, be a path through a circuit, satisfying the conditions, $t_{r\ell}(x_{i+1}) = t_{r\ell}(x_i) + d(x_{i+1})$ and $t_{a\ell}(x_{i+1}) = t_{a\ell}(x_i) + d(x_{i+1})$, for $u \le i < v$. Further, let u and v be extremal values in the sense that at least one of the conditions is not satisfied for $i = u - 1$ and $i = v$. Let x_a, $a \notin [u, v]$, be a component in path ρ such that $s_\ell(x_a) = s_\ell(x_u) = \cdots = s_\ell(x_v)$. For any safe assignment $\langle \delta(x_u), \cdots, \delta(x_v) \rangle$, the new slack, $s'_\ell(x_a)$, on component x_a must be positive.

Proof: Let $a < u$. By construction, either $t_{a\ell}(x_{u-1}) < t_{a\ell}(x_u) - d(x_u)$ or $t_{r\ell}(x_{u-1}) < t_{r\ell}(x_u) - d(x_u)$. This implies by definition that either $t_{a\ell}(x_a) < t_{a\ell}(x_u) - \Sigma_{i=a+1}^{u} d(x_i)$ or $t_{r\ell}(x_a) < t_{r\ell}(x_u) - \Sigma_{i=a+1}^{u} d(x_i)$. In fact, both these inequalities must hold because $s_\ell(x_a) = s_\ell(x_u)$. Denote the new required late time at x_u as $t'_{r\ell}(x_u)$, and that at x_a as $t'_{r\ell}(x_a)$. By definition,

$$t'_{r\ell}(x_a) = \min\left(t_{r\ell}(x_a), t'_{r\ell}(x_u) - \sum_{i=a+1}^{u} d(x_i)\right).$$

Since the incremental delay assignments are made only to those components x_i, $i > a$, the arrival time remains unchanged. Hence $t'_{a\ell}(x_a) = t_{a\ell}(x_a)$. If $t_{r\ell}(x_a) \le t'_{r\ell}(x_u) - \Sigma_{i=a+1}^{u} d(x_i)$, then $s'_\ell(x_a) = t'_{r\ell}(x_a) - t'_{a\ell}(x_a) = s_\ell(x_a) > 0$. Hence the interesting case is when $t_{r\ell}(x_a) > t'_{r\ell}(x_u) - \Sigma_{i=a+1}^{u} d(x_i)$. In this case,

$$\begin{aligned} t'_{r\ell}(x_a) &= t'_{r\ell}(x_u) - \sum_{i=a+1}^{u} d(x_i) \\ &= t_{r\ell}(x_u) - s_\ell(x_u) - \sum_{i=a+1}^{u} d(x_i) \\ &> t_{r\ell}(x_a) - s_\ell(x_u), \text{ as mentioned above} \\ &> t_{r\ell}(x_a) - s_\ell(x_a) \\ &> t_{a\ell}(x_a) \\ &> t'_{a\ell}(x_a). \end{aligned}$$

Hence the new slack, $s'_\ell(x_a) = t'_{r\ell}(x_a) - t'_{a\ell}(x_a) > 0$. A similar line of reasoning for the case when $a > v$ leads to the lemma. □

Theorem 1: For a given safe circuit, the zero-slack algorithm generates an assignment resulting in $s_\ell(x) = 0$ for all $x \in X$, in $O(np)$ time.

Proof: In each iteration of the loop of the zero-slack algorithm, a slack value which is nonzero, and hence positive, is chosen. Delays are distributed along components that have this chosen slack value. Further, the sum of incremental delays introduced on these components is exactly equal to the slack. By Lemma 3 the circuit remains safe at the end of the iteration. The components on which the slack is assigned satisfy the conditions of Lemma 4. By the same lemma, all chosen components have their slack reduced equally by an amount equal the sum of the incremental delays. Hence the slack on each of these components is reduced to zero. Finally, the procedure ensures that u and v are extremal in the sense of Lemma 5. By this lemma, no other component with the same slack value as the chosen components has its slack reduced to zero. All remaining components have either greater slack or zero slack. Components with greater slack cannot have their slack reduced to zero. Components with zero slack are unaffected because by Lemma 3 the circuit is still safe.

At each iteration, the most time consuming operation is the computation of slacks for the circuit. This takes $O(p)$ time, where p is the size of the interconnection list. We have proved that at least one component which has nonzero slack gets zero slack at the end of an iteration. By Lemma 1 no component having zero slack can have its slack increased by the introduction of delays. If n is the number of components, then n iterations, taking a total of $O(np)$ time, suffice to make all components have zero slack. □

The proof of the above theorem suggests the following corollary.

Corollary 2: Every component which has nonzero slack at the beginning is assigned a nonzero incremental delay by the algorithm.

This corollary essentially contrasts the *greedy_zero_slack* algorithm with the modified *zero_slack* algorithm. The latter essentially guarantees that if a net is not on an already-critical path, it will be provided with an upper bound on its length which is greater than the minimum possible length for it. In Section VI we will describe an algorithm which assigns a delay of at least c to every component, where c is the largest value which ensures that the circuit is safe.

We now provide an example to demonstrate how the algorithm works. Fig. 2(a) shows the initial actual arrival time and the required time at each component of the circuit. The path (2, 5, 8, 11) is critical. All the components on this path have zero slack. The minimum slack among the rest of the components is 2 and the path (I_1, 1, 4, 7, 10) with $x_u = 4$ and $x_v = 7$ satisfies the conditions of Lemma 5. At the end of this iteration, components 4 and 7 are each assigned an incremental delay of 1. This reduces the slack on these components to zero. Continuing

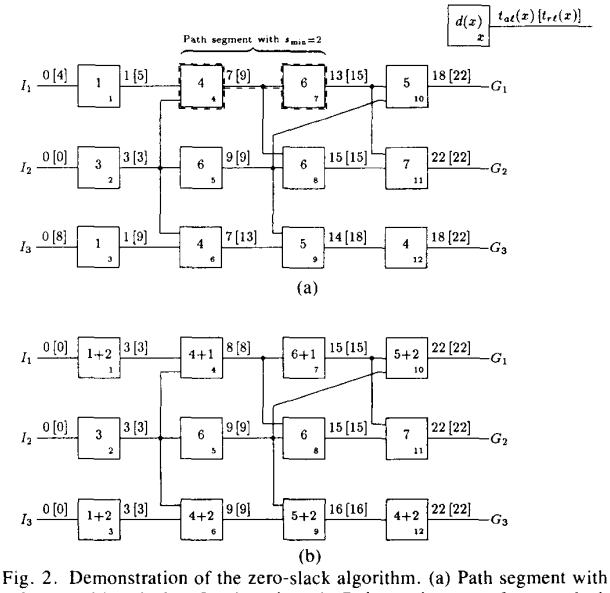

Fig. 2. Demonstration of the zero-slack algorithm. (a) Path segment with least positive slack at first iteration. (b) Delay assignments for zero slack at end of algorithm.

the process, at the second iteration, component 1 gets assigned an incremental slack of 2, since its slack is reduced to 2 by addition of delays to components 4 and 7. In three additional iterations, the algorithm assigns incremental delays to the remaining components as shown in Fig. 2(b).

IV. Adaptation to the Early Mode

In Section III we showed how delays may be assigned to a late-safe circuit in such a way that the actual late times are exactly equal to the required late times on every path through the circuit. The analogy for the early mode would be an early-safe circuit where *negative* incremental delays are assigned to the components such that the actual early times exactly equal the required early times. Such a procedure is uninteresting if the starting point is the set of base delay values, since the resulting delays lead to unimplementable net length values. However, in Section V we will see a situation where the starting delay values on the nets are higher than the base values, and reflect the constraints on net characteristics to be satisfied by the layout program. In such situations, the negative incremental delays essentially determine the tolerance to error during layout.

Analogous to the results for the late mode presented in Section III, we can prove the following:

Lemma 6: For any path ρ through the circuit, $s_e(x) \leq s_e(\rho)$ for $x \in \rho$.

Proof: Let $(x_1, \cdots, x = x_j, \cdots, x_k)$, $1 \leq j \leq k$, constitute path ρ. By definition,

$$t_{re}(x_j) \geq t_{re}(x_k) - \sum_{i>j} d(x_i)$$

and

$$t_{ae}(x_j) \leq t_{ae}(x_1) + \sum_{i \leq j} d(x_i).$$

Hence,

$$t_{ae}(x_j) - t_{re}(x_j) \leq t_{ae}(x_k) - t_{re}(x_1) + \sum_{1<i\leq k} d(x_i)$$

leading to

$$s_e(x_j) \leq s_e(\rho). \qquad \square$$

We will just state the remaining results without proofs. As seen above, the proofs are essentially identical to the late-mode case.

Lemma 7: For each component $x \in X$ there is some path ρ_x such that $s_e(x) = s_e(\rho_x)$.

Let $\delta_e(x)$ represent the amount by which the delay is reduced at component x. In other words the incremental delay at x is $-\delta_e(x)$.

Corollary 3: Given a safe circuit, an assignment $\delta_e(x)$, $x \in X$, is safe iff $\delta_e(x) \leq s_e(x)$.

Lemma 8: Given a safe circuit, an assignment $\Delta_e(\chi)$, $\chi \subseteq X$, is safe if $\Sigma_{x_i \in \chi} \delta_e(x_i) \leq \min_{x_i \in \chi} s_e(x_i)$.

Lemma 9: Let $\rho = \langle x_1, \cdots, x_u, \cdots, x_v, \cdots, x_k \rangle$ be a path through a circuit such that, for $u \leq i < v$, $t_{re}(x_{i+1}) = t_{re}(x_i) + d(x_{i+1})$ and $t_{ae}(x_{i+1}) = t_{ae}(x_i) + d(x_{i+1})$. Any assignment of delays to the components $x_u, \cdots, x_v$ affects the slack on all components $x_u, \cdots, x_v$ equally.

Lemma 10: Let $\rho = \langle x_1, \cdots, x_u, \cdots, x_v, \cdots, x_k \rangle$, $s_e(\rho) > 0$, be a path through a circuit satisfying the conditions $t_{re}(x_{i+1}) = t_{re}(x_i) + d(x_{i+1})$ and $t_{ae}(x_{i+1}) = t_{ae}(x_i) + d(x_{i+1})$, for $u \leq i < v$. Further, let u and v be extremal values in the sense that at least one of the conditions is not satisfied for $i = u - 1$ and $i = v$. Let x_a, $a \notin [u, v]$, be a component in ρ such that $s_e(x_a) = s_e(x_u) = \cdots = s_e(x_v)$. For any safe assignment $\langle \delta(x_u), \cdots, \delta(x_v) \rangle$, the new slack, $s'_e(x_a)$, on component x_a must be positive.

We could now write a zero-slack algorithm for the early mode in exactly the same way as the algorithm for the late mode, except that the distributed slacks along identified paths are *subtracted* instead of being added. However, the problem with a procedure implemented as above is that the delays on components at the end of the procedures could be less than their base values, and hence unimplementable.

Modification 1: At the end of such a "zero early slack" procedure, set delay values to be max $(d(x), d_0(x))$ for each x.

This ensures that the circuit is safe and that meaningful values are passed to the placement program. However there could be paths that have positive early slack. Moreover, it may have been possible to assign delays in a way that fewer paths have positive slack.

Modification 2: Each time the slack is distributed among components, allocate only as much incremental

delay to a component x such that its resulting delay is at least $d_0(x)$.

This ensures that no component gets assigned an unreasonable delay. However, we can no longer guarantee that at least one component gets zero slack at each iteration. The algorithm could stagnate with the same set of components being selected but unmodified at each iteration.

Modification 3: Perform the algorithm according to Modification 2. If the slack along the identified path in an iteration does not reduce to zero, add the components along the path to an initially empty *Done* list. At the beginning of each iteration, while identifying the component with the least positive slack, do not consider components in the *Done* list.

We now have a guarantee that some component is retired at each iteration. However, the procedure still does not guarantee the generation of a delay assignment in which all paths have zero slack when such an assignment exists. The distribution of slacks at any iteration is critical in obtaining a zero-slack safe assignment for the circuit when one exists. An efficient algorithm for obtaining a zero-slack safe assignment is currently not known. We conjecture the existence of an $O(np)$ algorithm for the problem. The distribution of slacks at any iteration is critical in such an algorithm. However, the only practical situation where we are interested in performing a zero-slack algorithm for the early mode is the one to be described in Section V, and in this situation an approximate algorithm, as in Modification 3, suffices. Presented below is Modification 3 as the *near_zero_early_slack* algorithm.

```
procedure near_zero_early_slack;    {early mode}
begin
  for i := 1 to n do Done(x_i) := false ;
  repeat
    compute_slacks;
    s_min := ∞;
    {Find minimum positive slack}
    for i := 1 to n do
      if (not Done(x_i)) and (s_e(x_i) < s_min) and (s_e(x_i) > 0) then
      begin s_min := s_e(x_i); x_min := x_i; end ;

    if s_min ≠ ∞ then
    begin {Find forward path segment}
      a_0 := x_min; v := 0;
      repeat
        find x ∈ π⁺(a_v) | t_re(x) = t_re(a_v) + d(x) and t_ae(x) = t_ae(a_v) + d(x);
        if x exists then begin a_{v+1} := x; v := v + 1; end ;
      until no such x exists;
      {Find backward path segment}
      u := 0;
      repeat
        find x ∈ π⁻(a_u) | t_re(x) = t_re(a_u) − d(u) and t_ae(x) = t_ae(a_u) − d(u);
        if x exists then begin a_{u−1} := x; u := u − 1; end ;
      until no such x exists;

      {Distribute slacks}
      s := s_min;
      for i := u to v do begin
        ŝ := min ( s/(v − i + 1), d(x_i) − d_0(x_i) );
        d(x_i) := d(x_i) − ŝ;      {Decrease delay of x_i}
        Done(x_i) := ( d(x_i) − d_0(x_i) = 0 );
        s := s − ŝ;
      end ;
    end ;
  until s_min = ∞;
end ;
```

The variable *Done* is a Boolean array variable whose element is set to true if the delay at the corresponding component has been reduced to its base value. It is easy to see in a manner analogous to Theorem 1 that this algorithm takes $O(np)$ time and, further, that it guarantees that every component having positive initial slack and delay greater than the base delay value has its slack reduced in the course of the algorithm.

We noted in Section III that if the initial circuit is not late-safe, it is necessary to redesign the logic. On the other

hand, if the circuit is not early-safe, it can be made early-safe by *addition* of delays to appropriate components. In the absence of late mode constraints, it is trivial to convert the circuit to a safe circuit and then apply procedure *near_zero_early_slack*. However in the presence of late-mode constraints, the resulting solution may not be late-safe if the additional delays are not chosen carefully. The problem is further complicated by the fact that there may be no safe assignment, let alone a zero-slack assignment for the circuit in the presence of mixed mode constraints. In Section V we will see a linear programming formulation to the problem which can handle this and other cases. The existence of a more efficient solution for these general cases is, at present, unknown.

V. The Mixed Mode Problem

In this section we will address the problem of simultaneously generating upper and lower bounds on the delays. We use the well-studied linear programming approach; several textbooks, e.g. [12], exist on the subject. It is now known [8] that a polynomial time solution exists for the linear programming problem. However, at present, it remains unclear whether the new methods are more effective in practice than such older techniques as the Simplex algorithm [13] and its variations, which could take exponential time in pathological cases, but are quite efficient for common examples.

Given a set of required times, $t_{rl}(x)$ and $t_{re}(x)$ at the output components, $x \in G$, and a set of actual times at the input components, $t_{al}(x)$ and $t_{ae}(x)$, $x \in I$, we can write the following sets of inequalities from the definitions of required times and actual times:

$$\begin{aligned}
t_{al}(x) &\geq t_{al}(z) + d(x), && x \in X, z \in \pi^-(x)\\
t_{ae}(x) &\leq t_{ae}(z) + d(x), && x \in X, z \in \pi^-(x)\\
t_{al}(x) &\leq t_{rl}(x), && x \in G\\
t_{ae}(x) &\geq t_{re}(x), && x \in G\\
d(x) &\geq d_0(x), && x \in X.
\end{aligned}$$

Each line above represents not just one inequality but a set of inequalities. Any feasible solution for the above sets of constraints results in a safe assignment of delay values $d(x)$, for $x \in X$. Such a solution is safe, even if the circuit with base delay values is not early-safe. Further, the last set of constraints ensures that if a solution exists then the delays must be greater than the base delay values, and hence implementable. (Because of this restriction, the initial circuit must be late-safe. No feasible solution can exist if it is not.) Finding a feasible solution or determining that none exists may be done in polynomial time by the ellipsoid algorithm [8]. A more practical approach may be to treat the constraints as those of a linear programming problem with a null objective function. Any feasible solution would then serve as an optimal solution to the resulting instance of the linear programming problem.

Solving the linear programming problem to obtain a feasible solution provides only one set of delay values for safe operation of the circuit. This set could be used to generate net length constraints for placement. Unfortunately, it would be impossible in practice to generate a placement in which every net has a specified length. It is more useful to generate a range of delay values for each component, such that any net lengths that correspond to delays within this range guarantee safe operation of the circuit. Further, in order to maximize the tolerance to error in placement, we wish to generate these delay bounds in such a way that the minimum difference between these bounds, calculated over all nets, is maximized.

We define a *minimum set* of safe delay values $\{d_1(x)\}$ as one in which, for each x, $d_1(x) \geq d_0(x)$ and reducing $d_1(x)$ causes the circuit to be no longer safe. Similarly, a *maximum set* $\{d_2(x)\}$ is defined as one in which, for each x, $d_2(x) \geq d_0(x)$ and increasing $d_2(x)$ causes the circuit to be no longer safe. We note that for a given component x, it may be possible to either reduce $d_1(x)$ or increase $d_2(x)$ in conjunction with changes in delays on other components. Hence there are several possible minimum and maximum sets. We observe here that the *zero_slack* algorithm generates a maximum set in the presence of late-mode constraints alone, while the *near_zero_early_slack* algorithm determines a minimum set in the presence of early-mode constraints alone.

Let us define a *compatible set* of delay values as a set of delay pairs $\{(d_1(x), d_2(x))\}$, where $\{d_1(x)\}$ is a minimum set and $\{d_2(x)\}$ is a maximum set, and for each x, $d_2(x) > d_1(x)$. Given a compatible set we may translate the delay values on each component to length bounds on their respective output pin nets. We are guaranteed that if these delay ranges are satisfied by a placement then the circuit is safe, i.e., the early and late timing constraints are satisfied. Let us define the *range* of a compatible set to be $\min_{x \in X}(d_2(x) - d_1(x))$. Shown below is a linear programming formulation, a solution to which determines the maximum possible range of any compatible set for a given set of timing constraints.

Maximize c, subject to the constraints:

$$\begin{aligned}
t_{al}(x) &\geq t_{al}(z) + d(x) + c, && x \in X, z \in \pi^-(x)\\
t_{ae}(x) &\leq t_{ae}(z) + d(x), && x \in X, z \in \pi^-(x)\\
t_{al}(x) &\leq t_{rl}(x), && x \in G\\
t_{ae}(x) &\geq t_{re}(x), && x \in G\\
d(x) &\geq d_0(x), && x \in X\\
c &\geq 0.
\end{aligned}$$

A solution to the above, if one exists, determines safe values $\tilde{d}(x)$ for each x in such a way that $\tilde{d}(x) + \tilde{c}$ is also safe, where $\tilde{c}$ is the optimal value of c. There can be no other set of safe delay values for which $c > \tilde{c}$. Typically these values will reduce the slack only on a few of the components to zero. Thus the layout will be overcon-

strained if every net is specified an upper bound on its delay contribution that is only $\tilde{c}$ units greater than its lower bound. In order to give a greater degree of freedom to other less critical nets, we apply the zero-slack procedure as described below, starting with the sets $\{\tilde{d}(x)\}$ and $\{\tilde{d}(x) + \tilde{c}\}$ just found.

Step 1) Solve the linear programming problem as mentioned above. Let $\tilde{c}$ be the maximum range determined. Let $\tilde{d}(x)$ be the values of $d(x)$ in the solution.

Step 2) With initial delays of $\tilde{d}(x) + \tilde{c}$ on each component, perform the zero slack algorithm for the late mode. Let $\{d_2(x)\}$ represent the resulting maximum set.

Step 3) With initial delays of $\tilde{d}(x)$ on each component, perform the *near_zero_early_slack* algorithm for the early mode. Let $\{d_1(x)\}$ represent the resulting minimum set.

The resulting values represent a compatible set with optimum range. For any other compatible set $\{(d_1'(x), d_2'(x))\}$, it must be the case that $\min_{\text{all } x}(d_2'(x) - d_1'(x)) \leq \tilde{c}$.

The above procedure may be modified to obtain better delay ranges for noncritical nets. Notice that if there is a path through the circuit which is already critical in both the early and late modes, $\tilde{c}$ will be zero. Even when this is not the case $\tilde{c}$ is likely to be very small, determined by the near criticality of paths involving a rather small fraction of components. In such a case, it would be beneficial to run through the linear programming phase again, freezing the minimum and maximum delays on the components which have zero slack at the end of the first run of the linear programming problem. This procedure could be repeated until the linear programming formulation at some point has no feasible solution. Such an iterative procedure maximizes the assignment on at least one component at each iteration; however, it takes too long to run—in the worst case, as many n instances of the linear programming problem may have to be solved. To speed up the solution, at the end of each iteration one could retire from consideration components having small slack, in addition to components having zero slack.

Let us now return to the late-mode problem. In many cases, the initial circuit is already early-safe. In such cases, $d_1(x)$ is going to be uniformly zero, and early mode calculations are not needed. In the next section we provide a fast algorithm for finding the optimum range in such situations.

VI. Optimal Assignment for Late Mode

Let us reconsider the late mode zero-slack algorithm. In Section III we provided an algorithm that guarantees that all components have zero slack when such an assignment exists. The algorithm heuristically partitioned the slacks as delays among the nodes. On the other hand, in Section V we saw a linear programming formulation for the more general problem which guarantees an optimal range for the delays assigned to the components. As mentioned at the end of the section, we could remove some of the constraints to obtain an optimal solution when early mode constraints are absent. Such a formulation is shown below:

Maximize c, subject to the constraints:

$$t_{a\ell}(x) \geq t_{a\ell}(z) + d_0(x) + c, \qquad x \in X,\ z \in \pi^-(x)$$

$$t_{a\ell}(x) \leq t_{r\ell}(x), \qquad x \in G$$

$$c \geq 0.$$

Clearly, c is maximized when $d(x) = d_0(x)$; hence the absence of variables $\{d(x)\}$. Having obtained the optimal value $\tilde{c}$ it suffices to run the zero-slack algorithm as in Section III using $d_0(x) + \tilde{c}$ as the starting value for each x. The asymptotic complexity of such a procedure depends on the complexity of the procedure to solve the linear programming problem. Fortunately the above problem is structured in a way that allows an alternative efficient algorithm for its solution. In the rest of this section we will demonstrate such an algorithm.

Let R represent the nonnegative real number system. We define a Ξ *function* as a mapping $\Xi: \mathsf{R} \to \mathsf{R}$ satisfying the following properties:

1) It is *convex*, i.e., $\Xi((1-\lambda)p + \lambda q) \leq (1-\lambda)\Xi(p) + \lambda\Xi(q), \quad 0 < \lambda < 1.$
2) It is piecewise linear.
3) Each of the linear regions has a positive, integer slope.

An example of a Ξ function is shown in Fig. 3.

We can identify a Ξ function using the minimum set of lines of which it is the envelope. Thus the ordered set of pairs $\{(m_1, b_1), (m_2, b_2), \cdots, (m_n, b_n)\}$, $m_i < m_{i+1}$, characterizes such a function completely, with the component lines defined by equations $T = f(\tau) = m_i\tau + b_i$, $1 \leq i \leq n$. We will call b_i the *offset* of the line with slope m_i. For example, the Ξ function of Fig. 3 is characterized by $\{(1, 6), (2, 4), (3, 0)\}$ as shown in Fig. 4. We will define the steepness, ξ, of a Ξ function as its highest slope, m_n. Let Ξ_1, Ξ_2, and Ξ_3 represent three Ξ functions with steepness ξ_1, ξ_2, and ξ_3, respectively. The following properties, somewhat analagous to those of convex functions, follow from the definitions above:

Lemma 11: The sum, $\Xi_1 + \Xi_2$, of two Ξ functions, Ξ_1 and Ξ_2, is a Ξ function with steepness $\xi_1 + \xi_2$.

Lemma 12: The maximum, $\max(\Xi_1, \Xi_2)$, of two Ξ functions, Ξ_1 and Ξ_2, is a Ξ function with steepness $\max(\xi_1, \xi_2)$.

Lemma 13: $\max(\Xi_1, \Xi_2, \Xi_3) = \max(\max(\Xi_1, \Xi_2), \Xi_3)$ with steepness $\max(\xi_1, \xi_2, \xi_3)$.

Lemma 13 provides a bound on the steepness and hence the number of component lines for the maximum of a set of Ξ functions. We give below an efficient procedure to compute this maximum Ξ function. Assume that there are k Ξ functions, the ith function Ξ_i being characterized by

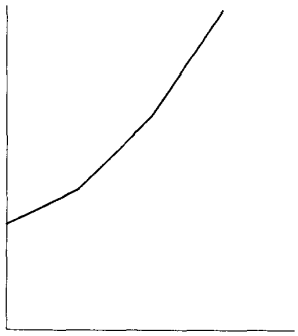

Fig. 3. Sample Ξ function.

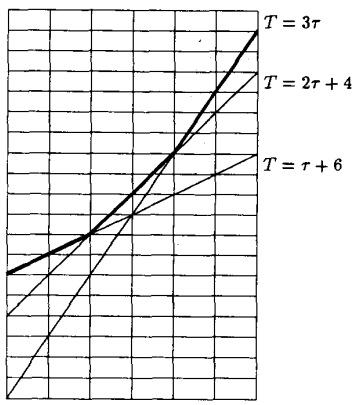

Fig. 4. Ξ function of Fig. 3 characterized by $\{(1, 6), (2, 4), (3, 0)\}$.

the pair set $\{(m_{i,1}, b_{i,1}), (m_{i,2}, b_{i,2}), \cdots, (m_{i,n_i}, b_{i,n_i})\}$.

At the end of the procedure, the set $\{(m'_1, b'_1), (m'_2, b'_2), \cdots, (m'_{n'}, b'_{n'})\}$ characterizes the maximum. The procedure works as follows: In line L1, the steepness of the maximum, and thereby the set of possible slopes, is determined. In L2 the procedure eliminates lines for which there are other lines with the same slope and a larger offset. Beginning at L3, the procedure generates the components of the maximum. It does this by iteratively generating the envelope of components up to a slope j from the envelope of components with slopes less than j. Specifically, in L4, the procedure eliminates those components of the current envelope which do not form part of the next envelope. At any iteration, s represents the number of components in the current envelope and a_s represents the abscissa (τ coordinate) of the point at which the line with slope j intersects the envelope of lines with slopes less than j. For two consecutive slopes j and k in the envelope, $j < k$, it must be the case that $a_j < a_k$.

For example, consider the set $\{(1, 6), (2, 4), (3, 0), (5, 2)\}$. The quantity a_1 is set to -6. The intersection of line $(1, 6)$ with $(2, 4)$ leads to $a_2 = 2$. Hence the envelope of lines with slope 2 or less is $\{(1, 6), (2, 4)\}$. For slope 3, we have $a_3 = 4$. Since $a_3 > a_2$, no component of the current envelope is discarded. The line $(5, 2)$ intersects line $(3, 0)$ at $x = 1$. So line $(3, 0)$ cannot be a component of the next envelope because $a_3 = 4$. Similarly the intersection of $(5, 2)$ with $(2, 4)$ occurs at $\tau = 2/3$. Since $a_2 = 2$, we discard the line $(2, 4)$ also. Finally, $(5, 2)$ is intersected with $(1, 6)$. This results in a_5

```
procedure maxΞ (Ξ_1, Ξ_2, ··· , Ξ_k);
begin {Compute steepness of maximum}
  ξ := 0;
L1. for i := 1 to k do ξ := max (ξ, m_{i,n_i});
  {Compute maximum offset for each slope}
  for j := 1 to ξ do bmax_j := −∞;
L2. for i := 1 to k do
      for j := 1 to n_i do
        bmax_{m_{i,j}} := max (bmax_{m_{i,j}}, b_{i,j});
  s := 0; a_0 := −∞; m'_0 := 0; b'_0 := 0;
L3. for j := 1 to ξ do
  if b_j ≠ −∞ then
  {line segment with slope j exists}
  begin
    s := s + 1; m'_s := j; b'_j := bmax_j;
    {intersect with line of lower slope}
    a_s := (c'_{s−1} − c'_s) / (m'_s − m'_{s−1});
L4.  while (a_s < a_{s−1}) do
      {Continue intersecting until a_s > a_{s−1}}
      begin
        s := s − 1; m'_s := m'_{s+1}; b'_s := b'_{s+1};
        a_s := (c'_{s−1} − c'_s) / (m'_s − m'_{s−1});
      end ;
  end ;
n' := s;
{Number of line components in the maximum}
end ;
```

$= 1$, which is greater than a_1. Thus there are two surviving lines $\{(1, 6), (5, 2)\}$ forming the final envelope.

Lemma 14: Procedure max Ξ determines the maximum of k Ξ functions each with steepness $\leq \xi$ in $O(k\xi)$ time.

Proof: The initial operation of uniquely identifying one line for each slope takes $O(k\xi)$ time as implemented above. We now need to bound the number of times a line intersection is calculated. Observe that whenever an intersection is made for a selected slope, either the line that it is intersected with is discarded, or the intersection is retained. In the latter case, the next higher slope is selected. Since each slope is selected once, at most ξ intersections are retained. Since a line once discarded is never intersected with any other line, at most ξ intersections result in discarded lines. Thus at most 2ξ intersections are calculated by the procedure. □

Let us now consider a circuit C with specified actual late arrival times $t_{a\ell}(I)$ at the inputs and required late arrival times $t_{r\ell}(G)$ at the outputs. The basic delays for the components is $\{d_0(x)\}$, $x \in X$. Assume that a variable delay τ is added to every component, resulting in the delay set $\{d_0(x) + \tau\}$. If the circuit is originally safe, then as τ is increased from 0, a value $\tilde{\tau}$ is reached beyond which the circuit is never safe. Let $t_{a\ell}(x, \tau)$ represent the actual arrival time at component x as a function of τ.

Lemma 15: For any component $x \in X$, if $t_{a\ell}(z, \tau)$, $z \in \pi^-(x)$ are Ξ functions with respect to τ with steepness ξ_z, then $t_{a\ell}(x, \tau)$ is also a Ξ function with respect to τ. Further, the steepness of $t_{a\ell}(x, \tau)$ is $1 + \max_{z\in\pi^-(x)} \xi_z$.

Proof: By definition, $t_{a\ell}(x, \tau) = \max_{z\in\pi^-(x)} t_{a\ell}(z, \tau) + d(x)$. By Lemma 13, $\max_{z\in\pi^-(x)} t_{a\ell}(z, \tau)$ is a Ξ function with steepness $\max_{z\in\pi^-(x)} \xi_z$. Now $d(x) = d_0(x) + \tau$, and is hence a Ξ function with steepness 1. Hence by Lemma 11 we have the desired result. □

A simple procedure to compute the actual late time as a function of τ for every component is shown below. We assume that two n-dimensional arrays $\{m_x\}$ and $\{b_x\}$ contain the information characterizing the n Ξ functions.

```
procedure compute_function (x, Ξ_x);
begin
  for z ∈ π⁻(x) do
    if not visited(z) then compute_function(z, Ξ_z);
  Ξ_x := max Ξ(Ξ_z | z ∈ π⁻(x));
  visited(x) := true ;
end ;
```

Define the *depth* of a component x in a circuit as follows:

$$depth(x) = \begin{cases} 0, & \text{if } x \in I \\ 1 + \max\limits_{z\in\pi^-(x)} depth(x), & \text{otherwise.} \end{cases}$$

The depth of a component also indicates the maximum number of components that are in any path from the primary inputs to that component. Define the *depth of a circuit* as the maximum depth of any component in the circuit. Computing the depth of a circuit is essentially a *topological sorting* of the components, an operation which can be done in a depth-first manner in $O(p)$ time, where p, as usual, is the size of the interconnection list. Since the primary input components $x \in I$ have given constant values for $t_{a\ell}(x)$, we can prove the following by induction:

Lemma 16: For a component $x \in X$ in a circuit, $t_{a\ell}(x, \tau)$, expressed as a function of a uniform delay parameter τ, is a Ξ function with steepness equal to the depth of the component.

The maximum value of τ for which the circuit is still safe is given by

$$\tilde{\tau} = \min_{\omega\in G} \tau_\omega \,|\, t_{a\ell}(\omega, \tau = \tau_w) = t_{r\ell}(\omega).$$

It is easy to see that any $\tau > \tilde{\tau}$ will violate the late timing requirement at some primary output.

The code below illustrates the computation of the optimum value for τ. We assume that when $t_{a\ell}(x)$ is computed for each x in procedure *compute_function*, the discontinuity points of the corresponding Ξ function are also saved in the vector $\{a_{x,i}\}$, $1 \leq i \leq k_x$, where k_x is the number of component line segments in $t_{a\ell}(x)$.

```
function τ̃(G: output_component_set): real;
begin
  for x ∈ X do visited(x) := false ;
  τ̃ := ∞;      {Initialization}
  for ω ∈ G do
  begin
    compute_function(ω, Ξ_ω);
    {Solve for τ_ω}
    for i := 1 to k_ω do
    begin      {Check intersection with ith segment}
      τ_ω := (t_rℓ(ω) − b_ω,i) / m_ω,i ;
      if τ_ω < a_ω,i+1 then leave ;      {Intersection found}
    end ;
    if τ̃ > τ_ω then τ̃ := τ_ω;
  end
end ;
```

Lemma 17: Computation of $\{t_{a\ell}(G, \tau)\}$ at the outputs to a circuit Ξ as a function of a uniform delay parameter τ takes $O(p \cdot \text{depth}(C))$ time, where p is the size of interconnection list.

Proof: To compute the values of $t_{a\ell}(G, \tau)$ we successively compute the $t_{a\ell}(x, \tau)$ functions for the components starting at the primary inputs, in increasing depth order. This is essentially a depth-first traversal starting at the outputs and is implemented recursively as shown in the procedure above. From Lemmas 14 and 16 we note that the computation of $t_{a\ell}(x, \tau)$ for any x takes $O(fanin(x) \cdot depth(x))$ time, where $fanin(x) = |\pi^{-}(x)|$. Since $depth(x)$ is bounded by $depth(C)$, and since the size of the interconnection list, p, is simply $\Sigma_x (1 + fanin(x))$, we have the required result. □

Theorem 2: Computing the optimum value of τ takes $O(p \cdot depth(C))$ time, where p is the size of the interconnection list for circuit.

Proof: We refer to the function $\tilde{\tau}$ above, which computes the optimum value for τ. By Lemma 17 the total computation involved with the *compute_function* procedure is $O(p \cdot depth(C))$. As implemented, solving for τ_ω takes $O(\xi_\omega)$ time. In any case, the total computation of this part is bounded by $O(q \cdot depth(C))$, where $q = |G|$. Since $q < p$, we have the desired result. □

As in Section V the *zero-slack* algorithm may now be invoked, assuming that the delay on each component x is $d_0(x) + \tilde{\tau}$. The resulting incremental delay values when added to $\tilde{\tau}$ give zero-slack delay assignments for the original circuit: any value of τ greater than $\tilde{\tau}$, when uniformly assigned to all components in the circuit would violate the late timing requirements at at least one of the primary outputs.

As mentioned at the end of Section V, it may be the case that $\tilde{\tau}$ as determined by the above procedure is zero or very small. In these cases, not much is gained by computing the value of $\tilde{\tau}$ by the above procedure. However, as mentioned there, it would still be possible to optimize the incremental delay assignments on the remaining components by freezing the delay assignments on the components with zero or small slack. The $t_{a\ell}(\omega)$ values at the outputs remain Ξ functions, possibly with lower steepness, because "frozen" delays are themselves Ξ functions with steepness 0. The resulting value of $\tilde{\tau}$ provides the additional delay that may be safely added to the remaining components. This process may then be continued or the *zero-slack* algorithm invoked to get upper bounds on delays for the remaining components.

VII. Concluding Remarks

We have presented a new approach to the problem of physical layout of circuits with specific performance constraints. The idea behind the approach has been embodied in an efficient algorithm called the *zero-slack* algorithm. In contrast to previous approaches, our technique guarantees that satisfying the constraints on each net individually satisfies the performance requirements on the circuit as a whole. The *zero-slack* algorithm has been implemented and is being used in the design of circuits within IBM.

The approach has been exercised on a number of CMOS parts, ranging from a macro with 291 components (358 nets) to a 9 mm semicustom chip having over 5000 components (>8000 nets). We performed a series of experiments on one of the parts to demonstrate the effectiveness of the approach. This circuit involved 1325 components and 1599 nets. The timing analysis and *zero-slack* algorithm were done within a logic synthesis system environment (LSS [14]). The *zero-slack* algorithm generated delay increments for each component, which were then transformed to upper bounds on net capacitances using delay equations for the cell library. The CPU time needed to read the design into the LSS data base, run timing analysis, and perform the *zero-slack* algorithm was 9 min on an IBM 3090. Of this, the time taken to calculate the maximum delays for nets and convert these delays to maximum capacitance values was 200 s. The placement algorithm used was a simulated annealing [15] program, widely used within the company. The program could be set up to optimize weighted functions of several parameters, one of which was the maximum permitted capacitance of each net.

The Table I shows the variation of the minimum slack at the outputs with the number of nets for which an upper bound was specified for capacitance. (The chosen nets were ones which were on the most critical paths after timing analysis.) Several interesting points emerge from this table. First, no significant change was observed in total wire length due to capacitance constraints on the nets. This was not particularly surprising for this example because the utilization of channels was below 50 percent. However, even for the 5000 component example, the additional constraint did not affect wirability.

The effect of controlling all nets in the circuit, rather than a subset, is noticeable. In this example, controlling all the nets makes it possible for the circuit to meet timing requirements. Controlling 178 or fewer nets on the most critical paths did not lead to successful placements.

It does not seem to be prudent to select a subset of nets to be constrained during placement because of yet another reason: the size of the subset does not appear to have a bearing on the suitability of the solution. For example, increasing the number of nets controlled from 0 to 23 to 63 successively generated placements which were worse in performance. In fact, the top 63 nets were the ones identified by the timing analysis program to be on paths with negative slack, assuming a "nominal" delay value on all nets. On the other hand, when an additional set of 115 nets on less critical (positive slack) paths were chosen, the placement generated was almost acceptable. With all nets controlled, an acceptable solution with positive slack on all paths was generated. The above phenomenon can only be explained by the fact that in attempting to make nets on the most critical paths short, those on the

TABLE I
VARIATION OF CIRCUIT PERFORMANCE WITH NUMBER OF NETS CONTROLLED

# nets controlled	1570	178	63	23	0	Ideal*
Worst slack (ns.)	0.543	−0.355	−4.024	−3.510	−0.953	5.506
Total length (wire units)	476	467	462	457	451	0

* Ideal implies zero delay on interconnections

TABLE II
AVERAGE DEVIATION FOR NETS THAT DO NOT MEET SPECIFIED CAPACITANCE BOUNDS

Capacitance upper bound (pf.) from *zero-slack* algorithm		0 – 0.5	0.5 – 1.0	1.0 – 1.5	1.5 – 2.0	2.0 – 2.5	2.5 – 3.0
Capacitance weighting =	0	0.140	0.451	1.148	1.953	0	0.997
	1	0.125	0.266	0.391	0.671	0.097	0.422
	2	0.110	0.173	0.249	0.454	0	0.103
		Average deviation (pf.) of violating nets					

TABLE III
DISTRIBUTION OF SLACK ON OUTPUTS

Output slack (ns.) less than		−5	0	5	10	15
Capacitance weighting =	0	5	25	44	85	139
	1	0	18	49	65	126
	2	0	0	39	57	110
		Number of outputs				

less critical paths are being elongated to the point where they become critical.

Table II shows the distribution of deviation of nets from the maximum capacitance computed by the *zero-slack* algorithm for various cases: (a) when the capacitance component in the placement objective function is completely ignored, (b) when the capacitance component is lightly weighted, and (c) when it is moderately weighted. The reduction in the deviation from specified bounds with increased capacitance weighting is evident. In fact, in the third case, timing requirements for the circuit were met in spite of the few violations. This is because a typical path may have as many as ten components, and there is a good chance that the *sum* of the delays along a path does not exceed bounds even though individual delays may exceed bounds.

Table III demonstrates the reduction in the number of outputs that have negative slack as the weighting factor for the capacitance component in the placement objective function is increased. As the weighting factor is increased, the net length component plays a smaller role, allowing alternate routes for nets to be explored to meet capacitance constraints. Wirability is ensured by controlling the parameters in the placement objective function corresponding to the peak and average channel utilizations.

The *zero-slack* algorithm does not attempt to maximize the bounds on the delay assignments. In Section VI we separately showed the existence of an efficient algorithm to compute the optimum delay assignment. The asymptotic time complexity of the algorithm was shown to increase as the product of circuit depth and net list size.

For the case when both early and late timing constraints are placed on signals through a circuit, the currently known algorithms employ linear programming. Whether more efficient algorithms exist is an interesting open question.

ACKNOWLEDGMENT

The authors wish to express their sincere thanks to A. Holm-Hansen and T. Masterson for providing the experimental results quoted in Section VII.

REFERENCES

[1] P. K. Wolff, A. E. Ruehli, B. J. Agule, J. D. Lesser, and G. Goertzel, "Power/timing: Optimization and layout techniques for LSI chips," *J. Design Automat. and Fault-Tolerant Comput.*, pp. 145–164, 1978.
[2] S. Teig, R. L. Smith, and J. Seaton, "Timing-driven layout of cell-based ICs," *VLSI Syst. Design*, pp. 63–73, May 1986.
[3] W. E. Donath, "Timing-driven placement," unpublished.
[4] M. Hanan and J. M. Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems: Theory and Techniques*, M. A. Breuer, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1972, ch. 5, pp. 213–282.
[5] A. E. Dunlop *et al.*, "Chip layout optimization using critical path weighting," in *Proc. 21st Design Automat. Conf.*, 1984, pp. 133–136.
[6] M. Burstein and M. Youssef, "Chip layout optimization using critical path weighting," in *Proc. 22nd Design Automat. Conf.*, 1985, pp. 124–130.
[7] P. S. Hauge, R. Nair, and E. J. Yoffa, "Circuit placement for predictable performance," in *Proc. Int. Conf. Computer-Aided Design*, 1987, pp. 88–91.
[8] L. G. Khachian, "A polynomial algorithm for linear programming," *Dokl. Akad. Nauk SSSR*, vol. 244, no. 5, pp. 1093–1096, 1979. Translated in *Sov. Math.*, vol. 20, pp. 191–194.
[9] R. B. Hitchcock, G. L. Smith, and D. D. Cheng, "Timing analysis of computer hardware," *IBM J. Res. Develop.*, vol. 26, pp. 100–105, Jan. 1983.
[10] D. Brand and V. S. Iyengar, "Timing analysis using functional relationships," in *Proc. Int. Conf. Computer-Aided Design*, 1986, pp. 126–129.
[11] R. Sedgewick, *Algorithms*. Reading, MA: Addison-Wesley, 1988, pp. 423–430.
[12] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[13] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press, 1963.
[14] J. A. Darringer, D. Brand, J. V. Gerbi, W. H. Joyner, and L. Trevillyan, "LSS: A system for production logic synthesis," *IBM J. Res. Develop.*, vol. 28, pp. 272–280, Sept. 1984.
[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

*

Ravi Nair (M'82–SM'87) received the B. Tech. degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, India, in 1974. He received the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1976 and 1978, respectively.

Since 1978, he has been with IBM at the Thomas J. Watson Research Center, in Yorktown Heights, NY. During 1987–1988 he was on sabbatical leave at Princeton University in the Department of Computer Science. He has worked on computer synthesis and analysis of VLSI layout, parallel machines for physical design, fault-tolerant systems, and testing. He is interested in algorithms and systems for the design of digital integrated circuits.

C. Leonard Berman (M'88) was born in Jersey City, NJ, in October 1949. He received the B.S. degree in mathematics (with honors) from the California Institute of Technology, Pasadena, CA, in 1971, the M.A. degree in mathematics from Cornell University, Ithaca, NY, in 1973, and the Ph.D. degree in computer science in 1977.

He has been at IBM Research since then, except for 1984–1985, when he held a visiting appointment at the Massachusetts Institute of Technology, Cambridge, MA. His main research area is automatic logic design.

*

Peter S. Hauge received the B.S., M.S., and Ph.D. degrees in electrical engineering in 1961, 1963, and 1967, respectively, from the University of Minnesota, Minneapolis.

He is manager of a group concerned with performance-driven VLSI physical design at the Thomas J. Watson Research Center, where has been since he joined IBM in 1968. He has developed physical design systems for cascode voltage switch macros and chips, and is currently exploring fast, timing-related heuristics for physical design automation tools. Until 1980, when he joined the Computer Science Department, he studied the interaction of microwave, infrared and visible radiation, with semiconducting materials and films. He is the coinventor of two automated instruments for on-line semiconductor process control.

Dr. Hauge is a member of the Optical Society of America.

*

Ellen J. Yoffa (M'86) received the B.S. and Ph.D. degrees in physics from the Massachusetts Institute of Technology, where her area of study was theoretical solid-state physics.

In 1978, she joined the IBM Thomas J. Watson Research Center for postdoctoral work in the Semiconductor Science and Technology Department, where she investigated ballistic conduction in semiconductor devices and the physics of the laser annealing process. Since 1980, she has been a member of the research staff in the Computer Science Department and is currently senior manager of the System Design and Verification Department, where she is responsible for managing research in tools for computer-aided design of VLSI chips, including system description and early design tools, logic synthesis, and hardware and software logic simulation. She is a member of the editorial board of *IEEE Design and Test of Computers*. Her research has involved the development of tools for VLSI physical design automation.

Dr. Yoffa is a member of Phi Beta Kappa and Sigma Xi.