MINFLOTRANSIT: MIN-COST FLOW BASED TRANSISTOR SIZING TOOL *

Vijay Sundararajan, Sachin S. Sapatnekar, Keshab K. Parhi
Dept. of ECE, University of Minnesota
Minneapolis, MN 55455
E-mail: vijay@ece.umn.edu, sachin@ece.umn.edu, parhi@ece.umn.edu

ABSTRACT

This paper presents MINFLOTRANSIT, a new transistor sizing tool for fast sizing of combinational circuits with minimal cost. MINFLOTRANSIT is an iterative relaxation based tool that has two alternating phases. For a circuit with $|V|$ transistors and $|E|$ wires, the first phase *(D-phase)* is based on minimum cost network flow, which in our application, has a worst-case complexity of $O(|V||E|log(log(|V|)))$. The second phase *(W-phase)* has a worst case complexity of $O(|V||E|)$. In practice, during our simulations both the *D-phase* and *W-phase* show a near linear run-time dependence on the size of the circuit, comparable to TILOS. Simulation results show excellent run-time behavior for MINFLOTRANSIT on all the ISCAS85 benchmark circuits. For reasonable delay targets MINFLOTRANSIT shows up to 16.5% area savings over a circuit sized using a TILOS-like algorithm.

1. INTRODUCTION

As evidenced by the successive announcement of ever faster computer systems, increasing the speed of VLSI systems is one of the major requirements for VLSI system designers today. Faster integrated circuits are making possible newer applications that were traditionally considered difficult to implement in hardware. In this scenario of increasing circuit complexity, reduction of circuit delay in integrated circuits is an important design objective. Transistor sizing is one such task that has been employed for speeding up circuits for quite some time now [1]. Given the circuit topology, the delay of a combinational circuit can be controlled by varying the sizes of transistors in the circuit. Here, the size of a transistor is measured in terms of its channel width, since the channel lengths of MOS transistors in a digital circuit are generally uniform. In any case, what really matters is the ratio of channel width to channel length, and if channel lengths are not uniform, this ratio can be considered as the size. In coarse terms, the circuit delay can usually be reduced by increasing the sizes of certain transistors in the circuit from the minimum size. Hence, making the circuit faster usually entails the penalty of increased circuit area relative to a minimum sized circuit and the area-delay trade-off involved here is the problem of transistor size optimization. A related problem to transistor sizing is called gate sizing, where a logic gate in a circuit is modeled as an equivalent inverter and the sizing optimization is carried out on this modified circuit with equivalent inverters in place of more complex gates. There is, therefore, a reduction in the number of size parameters corresponding to every gate in the circuit. Needless to say, this is an easier problem to solve than the general transistor sizing problem.

There has been a large amount of work done on transistor sizing [1–8] that underlines the importance of this optimization technique. Starting from a minimum sized circuit, TILOS [1] uses a greedy strategy for transistor sizing by iteratively sizing transistors in the critical path. A sensitivity factor is calculated for every transistor in the critical path to quantify the gain in circuit speed achieved by a unit upsizing of the transistor. The most sensitive transistor is then bumped up in size by a small constant factor to speed up the circuit. This process is repeated iteratively until the timing requirements are met. The technique is extremely simple to implement and has run-time behavior proportional to the size of the circuit. Its chief drawback is that it does not have guaranteed convergence properties and hence is not an exact optimization technique. Among the past approaches, only [2] and [8] are exact optimization techniques and the other techniques do not have proven convergence properties. While [2] was the first ever polynomial time technique reported for addressing the problem exactly, it does not have very good run-time behavior. The technique presented in [8] has shown impressive run-time behavior for sizing large adders. The run-time behavior of this technique on more complex circuits such as multipliers and controllers was not demonstrated. Moreover, the approach appears to be amenable only to tackling sizing problems where the gate delay is expressed using Elmore delays. This paper presents a novel way of solving the transistor sizing problem exactly and in an extremely fast manner. The proposed approach has some similarity in form to [6, 7] which will be subsequently explained, but the similarity in content is minimal and the details of implementation are vastly different. In essence, the proposed technique and the techniques in [6, 7] are iterative relaxation approaches that involve a two-step optimization strategy. The first-step involves a delay budgeting step where optimal delays are computed for transistors/gates. The second step involves sizing transistors optimally to achieve these delay budgets. The two steps are iteratively alternated until the solution converges, i.e., until the delay budgets calculated in the first step are exactly satisfied by the transistor sizes determined by the second step. The primary features of the proposed approach are:

1. It is computationally fast and is comparable to TILOS in its run-time behavior.

2. It can be used for true transistor sizing as well as the relaxed problem of gate-sizing. Additionally, the approach can easily incorporate wire sizing, as outlined in section 2.1.

3. It can be adapted for more general delay models than the Elmore delay model.

To elaborate further on the last point, the proposed model only requires the transistor delay to be expressible as a sum of *simple monotonic functionals*, defined in section 2.1, of transistor sizes and hence admits more general delay models than just the Elmore delay model.

The starting point for the proposed approach is a fast guess solution. This could be obtained, for example, from a circuit that has been optimized using TILOS to meet the given delay requirements. The proposed approach, as outlined earlier, is an iterative relaxation procedure that involves an alternating two-phase relaxed optimization sequence that is repeated iteratively until convergence is achieved. The two-phases in the proposed approach are:

- The *D-phase* where transistor sizes are assumed fixed and transistor delays are regarded as variable parameters. Irrespective of the delay model employed, this phase can be formulated as the dual of a min-cost network flow problem. Using $|V|$ to denote the number of transistors and $|E|$ the number of wires in the circuit, this step in our application has worst-case complexity of $O(|V||E|log(log|V|))$ [9].

- The *W-phase* where transistor/gate delays are assumed fixed and their sizes are regarded as variable parameters. As long as

*This research has been supported in part by the ARO under grant number DA/DAAG55-98-1-0315 and by SRC under grant number 99-TJ-692.

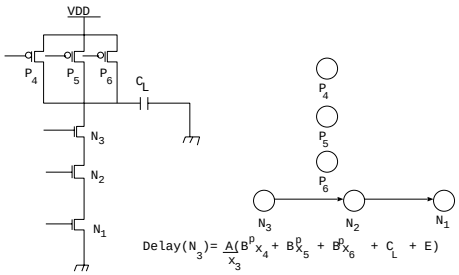

Figure 1. The DAG corresponding to a 3-input static CMOS nand gate.

the gate delay can be expressed as a separable function of the transistor sizes, this step can be solved as a Simple Monotonic Program (SMP) [10]. The complexity of SMP is similar to an all-pairs shortest path algorithm in a directed graph, [10, 11], i.e., $O(|V||E|)$.

The objective function for the problem is minimization of circuit area. In the W-phase, this objective is addressed directly, and in the D-phase the objective is chosen to facilitate a move in the solution space in a direction that is known to lead to a reduction in the circuit area.

2. PROPOSED APPROACH

The transistor size optimization problem can be stated as,

$$\begin{aligned} &minimize \quad \sum_{\text{all transistors, i}} x_i, \\ &subject\ to: Delay(Circuit) < T, \\ &minsize \leq x_i \leq maxsize, \end{aligned} \tag{1}$$

where x_i refers to the size of transistor i, T is timing requirement specified as an input to the optimization and $minsize$, $maxsize$ are, respectively, minimum and maximum bounds on the sizes of transistors in the circuit.

2.1. Equivalent DAG for Transistors/Gates

The proposed approach requires transistor delay to be expressible as a sum of *simple monotonic functionals* of transistor sizes, which are defined as follows:

Definition 1 *A function $D_i(x_1, ..., x_n)$ is a simple monotonic functional if it can be rewritten as $D_i = g(x_i)q(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$, where $g(x_i)$ is a monotonic decreasing function of x_i and $q(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ is a monotonic increasing function of each x_j $j \in \{1, ..., n\}$ $j \neq i$.*

Definition 2 *A function $D(x_1, ..., x_n)$ is termed decomposable into simple monotonic functionals if $D = \sum_{i \in \{1,...,n\}} D_i$ where each D_i, as in definition (1), is a simple monotonic functional. D_i is termed the simple monotonic projection of D on x_i.*

In order to mathematically model the transistor size optimization problem, every static CMOS gate in the circuit is first converted in to an equivalent Directed Acyclic Graph (DAG) model, shown in figure 1, as follows. There is a vertex in the DAG corresponding to every transistor. An edge is drawn between an NMOS (PMOS) transistor and another NMOS (PMOS) transistor provided there is a discharging (charging) path consisting of the two transistors. The edge is always directed from the transistor higher up in the discharging (charging) path to the transistor lower down in the discharging (charging) path. Every vertex of this DAG has a delay attribute associated with it. This delay attribute is given by the *simple monotonic projection* of the worst case discharging (charging) path delay through the transistor corresponding to the vertex on to the size of this transistor. Note that, as will be evident soon, the rise and fall delays are implicitly distinguished due to the fact that the DAG corresponding to every gate has separate components for pullup and pulldown networks.

For ease of exposition we will henceforth consider the commonly used Elmore delay model that can be decomposed in to *simple monotonic functionals*. Assuming x_i to be the size of transistor $N_i(P_i)$ in the pulldown(pullup) network of the 3-input NAND gate shown in figure 1, it can be shown [1] that the pulldown Elmore delay can be expressed as,

$$\begin{aligned} delay^{pulldown} = &(\frac{A}{x_1})(Bx_1 + Cx_2) + \\ &(\frac{A}{x_1} + \frac{A}{x_2})(Bx_2 + Cx_3 + D) + \\ &(\frac{A}{x_1} + \frac{A}{x_2} + \frac{A}{x_3})(Bx_3 + \\ &B^p x_4 + B^p x_5 + B^p x_6 + C_L + E), \end{aligned} \tag{2}$$

where A, B and C are constant coefficients, the resistance, drain and source capacitance, respectively, of a unit NMOS transistor. D and E are related to the wire capacitances. Similarly, B^p is the drain capacitance of a unit sized PMOS transistor and C_L is the load capacitance. Under this model if wire sizes were considered to be variables also then the form of (2) would remain similar. Rewriting the expression in (2) we get,

$$\begin{aligned} delay^{pulldown} = &\frac{A}{x_1}(Bx_2 + Bx_3 + Cx_2 + Cx_3 + D + E + \\ &B^p x_4 + B^p x_5 + B^p x_6 + C_L) + \\ &\frac{A}{x_2}(Bx_3 + Cx_3 + D + E + \\ &B^p x_4 + B^p x_5 + B^p x_6 + C_L) + \\ &\frac{A}{x_3}(B^p x_4 + B^p x_5 + B^p x_6 + C_L + E) + \\ &3AB. \end{aligned} \tag{3}$$

Since A, B, C, D, E, B^p, C_L are non-negative quantities, the Elmore delay model admits a *simple monotonic decomposition*. The above fact is illustrated in figure 1 where the delay corresponding to NMOS transistor N_3 is explicitly shown. The constant terms used in this expression have the same connotation as in (3). We claim that such a DAG model can always be developed for any complex static CMOS gate consisting of a series/parallel network of transistors as long as the underlying delay model admits a *simple monotonic decomposition*. This is a reasonable requirement since the reduction in the gate delay with an increase in its size can be modeled by the function g in Definition 1, and the increase in the gate delay with increasing fanout gate sizes can be modeled by the function q.

In addition, if wire sizing were also to be performed together with transistor sizing, then we could model the problem by augmenting the DAG corresponding to a gate by adding vertices corresponding to each wire. The edges emanating from and incident on these wires will be similarly constructed as for transistors. The delay attribute of a vertex corresponding to any wire can also be similarly defined as for that of a vertex corresponding to a transistor. We conclude that modeling the problem of wire sizing along with transistor sizing may use the same framework as transistor sizing alone, and the approach developed in this paper can simultaneously handle both. For ease of exposition, from here onwards, wire sizing will not be considered for the remainder of the paper.

Note that the DAG corresponding to a static CMOS gate has at least two disjoint connected components, as shown in figure 1, corresponding to the pulldown network of NMOS transistors (i.e., vertices N_1, N_2, N_3) and the pullup network of PMOS transistors (i.e., vertices P_1, P_2, P_3) corresponding to the gate. The portion of the DAG representing the NMOS pulldown networks corresponds to falling transitions and the portion of the DAG representing the PMOS pullup network is related to rising transitions at the output of the gate. Note that there are several vertices in the DAG of a gate that only have edges emanating from them and have no edges terminating on them; we refer to these vertices as the **root vertices** of the gate DAG. Also, note that there are several vertices in the DAG of a given gate that have only edges terminating on them and no edges emanating from them; these vertices constitute the **leaf vertices** of the gate DAG.

2.2. A DAG for the Circuit

The entire circuit consisting of static CMOS gates can be represented with an equivalent DAG, $G = (V, E)$, by connecting the

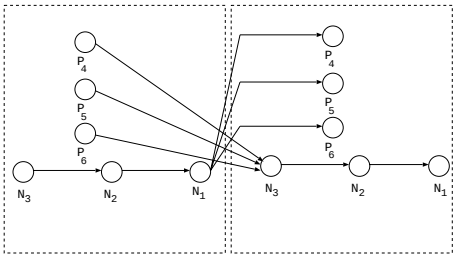

Figure 2. The DAG corresponding to a circuit consisting of two 3-input static CMOS nand gates in series.

component DAG's of individual gates. The construction of the circuit DAG is as follows, the vertex set V of the circuit DAG is simply the union of the vertex sets of DAG's corresponding to the gates in the circuit. The edge set E of the circuit DAG is constructed as follows. For every wire connecting the output of one gate to the input of another there will be a set of edges in the circuit DAG that go from the NMOS (PMOS) DAG components of the first gate to the PMOS (NMOS) DAG components of the second gate. So corresponding to every wire connecting the output of the first gate to a given NMOS (PMOS) transistor in the second gate, there will be edges emanating from all the leaf vertices of the PMOS (NMOS) DAG of the first gate. These edges terminate on all the root vertices of the NMOS (PMOS) DAG component of the second gate that are connected to the given transistor in the second gate. Figure 2 illustrates the construction of the circuit DAG for a circuit consisting of two 3-input nand gates in series.

2.3. Two-Phase Optimization

Note that the delay corresponding to a vertex, i, in the circuit DAG, whose corresponding transistor has a size x_i, can always be expressed as,

$$delay(i) = \frac{\sum_{j \in S(V(G))} a_{ij} x_j + b_i}{x_i}, \tag{4}$$

where $S(V(G))$ denotes some subset of $V(G)$ that is located in the neighborhood of the vertex i. In particular, this subset consists of the vertices corresponding to all those transistors whose sizes directly affect the delay of the transistor corresponding to vertex i. Also, note that all the coefficients a_{ij}, b_i in (4) are non-negative. Rearranging (4), we have,

$$delay(i) \cdot x_i - \sum_{j \in S(V(G))} a_{ij} x_j = b_i. \tag{5}$$

Denoting a diagonal matrix whose $(i,i)^{th}$ entry is $delay(i)$ by D, a matrix whose $(i,j)^{th}$ entry is a_{ij} by A, a column vector whose i^{th} component is b_i by $\mathbf{B}$ and a column vector whose i^{th} component is x_i by $\mathbf{X}$ we can rewrite (5) as

$$(D - A)\mathbf{X} = \mathbf{B}. \tag{6}$$

This formulation can be written as long as the delay model admits a simple monotonic decomposition. It can be shown that for strict gate sizing the matrix $(D-A)$ can be written as an upper triangular matrix. This is due to the fact that the adjacency matrix of the circuit DAG can be always written as an upper triangular matrix [12]. On the other hand it can be shown that for transistor sizing the matrix $(D - A)$ can be written as a block upper triangular matrix (proof not included due to space restrictions). We will henceforth assume that $(D-A)$ can always be represented in an upper triangular form or block upper triangular form.

With this assumption we can state if D is a constant matrix, then as long as $(D - A)$ is invertible, a system of equations of the form $(D - A)\mathbf{X} = \mathbf{B}$ can be solved for the variables $\mathbf{X}$ by a backward substitution process beginning from the bottom row and proceeding upwards and progressively solving for all x_i. From a circuit point of view, this process proceeds in a backward breadth-first manner beginning with the primary outputs and proceeding backwards in order of decreasing levels of logic of the circuit. This elimination process has $O(|F|N)$ computational complexity, where N is the number of components in the vector X and $|F|$ is bounded (in gate sizing) by the maximum fanout of any gate in the circuit. Note that as long as all vertices of the circuit DAG have a non-zero delay, which is always the case, the (block) upper triangular matrix $(D - A)$ for (transistor) gate sizing will always be invertible.

Now assume that we start with some initial sizing solution $\mathbf{X_0}$ and some delay matrix D_0 satisfying (6). We now resize the transistors slightly so that the new delay matrix is $D_0 + \delta D$ and the new size vector is $\mathbf{X_0} + \delta\mathbf{X}$, so that we then have,

$$\begin{aligned}
(D_0 + \delta D - A)(\mathbf{X_0} + \delta\mathbf{X}) &= \mathbf{B}, \\
(D_0 - A)(\mathbf{X_0} + \delta\mathbf{X}) + \delta D(\mathbf{X_0} + \delta\mathbf{X}) &= \mathbf{B}, \\
\mathbf{B} + (D_0 - A)\delta\mathbf{X} + \delta D\mathbf{X_0} + \delta D\delta\mathbf{X} &= \mathbf{B}, \\
(D_0 - A)\delta\mathbf{X} &\approx -\delta D\mathbf{X_0}, \\
\delta\mathbf{X} &\approx -(D_0 - A)^{-1}\delta D\mathbf{X_0},
\end{aligned} \tag{7}$$

where the term $\delta D\delta\mathbf{X}$ has been ignored, assuming small perturbations in δD and $\delta\mathbf{X}$. In other words, we make the following observations:

- For an infinitesimal resizing of the transistors corresponding to the vertices in the circuit DAG, the infinitesimal changes in transistor sizes can be represented as a linear function of the infinitesimal changes in transistor delays.

- As a result of (1), we see that the sum of all the components of $\delta\mathbf{X}$, which represents the sum of the change in sizes of the transistors corresponding to all the vertices in the circuit, can be expressed as a linear function of diagonal entries of the matrix δD.

It can be shown that since all components of $\mathbf{X_0}$ are positive, all components of $-(D_0 - A)^{-1}\mathbf{X_0}$ will be negative. Hence, the sum of all the components of $\delta\mathbf{X}$, can be expressed as a linear function of diagonal entries of the matrix δD, where the coefficient corresponding to each diagonal element of δD is negative.

This motivates a two-phase strategy for solving the transistor size optimization problem. In the D-phase, as above, we assume fixed transistor sizes and redistribute the delay budgets in such a manner as to minimize the resultant change in transistor sizes. In the W-phase, on the other hand, we try to find the minimal-sized circuit that satisfies the modified delay budget obtained after the D-phase. The two-phases are alternated till convergence is achieved and the delay budgets output by the D-phase are exactly satisfied by the transistor sizes calculated by the W-phase.

2.3.1. D-phase

First assume that the circuit has been sized to meet delay requirements using an algorithm such as TILOS. We now define three attributes for every vertex in the circuit DAG *G*. For a vertex i, these are the arrival time $AT(i)$, the required time $RT(i)$ and the slack, $sl(i)$. Additionally, every edge $e_{ij} \in$ E has the attribute *edge-slack*, $esl(e_{ij})$. The entire circuit graph G has an additional attribute $CP(G)$ that refers to the delay of the critical path of the corresponding circuit. We will now define all of these attributes formally.

$$\begin{aligned}
&\begin{cases} AT(i) &= external\ time\ of\ arrival,\ u \in PI, \\ &= max_{v \in fanin(i)}(AT(j) + delay(j)), else \end{cases} \\
&\{\ CP(G) \quad = \quad max_{u \in V}\,(AT(i) + delay(i)), \\
&\begin{cases} RT(i) &= CP(G) - delay(i),\ u \in PO, \\ RT(i) &= min_{v \in fanout(i)} RT(j) - delay(i), else \end{cases} \\
&\quad \begin{cases} sl(i) &= RT(i) - AT(i), \\ esl(e_{ij}) &= RT(j) - AT(i) - delay(i). \end{cases}
\end{aligned} \tag{8}$$

where PI and PO denote respectively the primary inputs and primary outputs of the circuit.

We call a circuit safe when all vertices $i \in$ V have $sl(i) \geq 0$ and all edges have $esl(e_{ij}) \geq 0$.

The D-phase involves minimally altering the delay budgets of transistors in the circuit to move towards a feasible minimum area

solution. For this to be possible, we need to capture the slack (available delay budget) for every transistor and also present a strategy to alter/redistribute these delay budgets. In the next section, we will first prsent an approach to capture the slack in a circuit in terms of fictitious buffer-like entities called *Fictitious Specific Delay Units (FSDU's)*. Next, an approach called *FSDU-displacement* will be presented, which redistributes the delay budgets for every transistor in such a manner that a lower area solution (from the present solution) is achieved that is also timing feasible.

Delay Balancing

A given circuit DAG *G* can be transformed to a functionally equivalent circuit DAG *G'* by introducing dummy units of appropriate delay on to each edge in the circuit DAG in such a manner that for every $e_{ij} \in E$, $esl(e_{ij}) = 0$ and $CP(G') = CP(G)$ [13]. This process is known as delay balancing. For our purposes, we do not explicitly insert physical delays. We instead, use the *concept* of delay balancing as a tool to capture all the slack in the circuit DAG. This captured slack is then used for the D-phase optimization. The delay units used for delay balancing are, therefore, *fictitious* entities whose only purpose is to model the slack present in the circuit DAG. We refer to these fictitious delay units as FSDUs (Fictitious Specific Delay Units). Figure 3 shows a circuit DAG and figure 4 shows its delay balanced counterpart; the "square boxes" on the edges of the circuit in figure 4 represent the FSDUs on that edge.

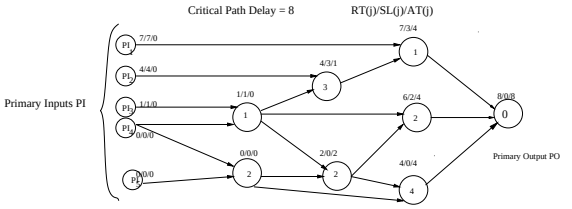

Figure 3. An example of a circuit DAG the integer numbers within each vertex represent its delay and each vertex i has the triplet (RT/SL/AT) above it.

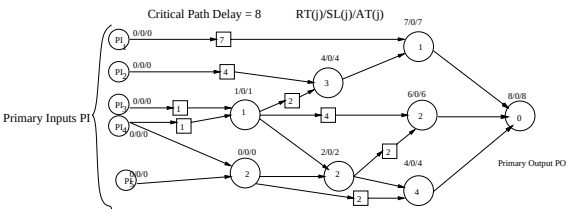

Figure 4. The circuit DAG in figure 3 after delay balancing. The square boxed integers on edges represent the FSDUs added to the edges for delay balancing.

Starting with a given circuit DAG there are several possible ways to produce a delay balanced graph. Any such delay balanced graph will from now on be referred to as a delay balanced configuration.

FSDU-Displacement

We define *FSDU-Displacement*, a circuit DAG transformation technique, as a mapping *r:V*$\longrightarrow$**Z**, {**Z***: the set of integers*} such that the delay of the FSDU on the edge e_{ij} after FSDU-Displacement, $FSDU^r(e_{ij})$, is related to the delay of the FSDU before FSDU-Displacement, $FSDU(e_{ij})$, by,

$$FSDU^r(e_{ij}) = FSDU(e_{ij}) + r(j) - r(i). \quad (9)$$

We state the following without proof, due to space limitations.

Theorem 1 *All legal delay balanced configurations for a given circuit-graph G are* FSDU-Displaced *versions of each other.*

Theorem 2 *The net change in the delay of any structural path from a vertex i to another vertex j after* FSDU-Displacement *is always* $r(j) - r(i)$.

The above theorem gives rise to the following corollary.

Corollary 1 *If we connect all the leaf vertices corresponding to primary output nodes of a given circuit DAG to a common dummy vertex O through dummy edges and if we restrict $r(O)$ to be exactly 0 and also restrict $r(I)$ for every input vertex I ∈ PI to be exactly 0, then the critical path of the transformed circuit DAG after FSDU-displacement remains unaltered.*

Before we develop a formal mathematical model, we first modify the circuit DAG by adding a $dummy$ vertex $Dmy(i)$ of delay 0 units at the output of every vertex i in the circuit DAG. A dummy edge connects vertex i to its corresponding dummy vertex $Dmy(i)$. All fanout edges which initially originated from vertex i now originate from $Dmy(i)$. Figure 5 illustrates this circuit DAG transformation with an example. Now we can summarize the D-phase as follows:

D-phase

(1) Produce any valid delay balanced configuration of the given circuit DAG. We use a depth first FSDU insertion heuristic for this purpose, [13].

(2) Now starting from the delay balanced configuration in (1) above, let $\delta\mathbf{X} = -(D_0 - A)^{-1}\delta D\mathbf{X} = -\mathbf{C^T} diag(\delta D)$ *where* $\mathbf{C^T} = -(D_0 - A)^{-1}\mathbf{X}$*, all other symbols are as defined earlier and diag(δD) is a column vector consisting of the diagonal elements of δD. Note that minimizing* $\sum \delta\mathbf{X_i}$ ≡ *minimizing* $\sum \mathbf{X_i}$*. Now, for every vertex i let* $\delta\mathbf{D_i} = r(Dmy(i)) - r(i)$*, which means that the delay of the FSDU at the output of a vertex is the change in its delay after the D-phase.*

(3) To maintain the requirement that δD will be small, introduce the following constraints for every vertex.

$$FSDU(i \rightarrow Dmy(i)) + r(Dmy(i)) - r(i) \geq MIN\Delta D(i),$$
$$FSDU(i \rightarrow Dmy(i)) + r(Dmy(i)) - r(i) \leq MAX\Delta D(i),$$

where $MIN\Delta D(i)$ and $MAX\Delta D(i)$ bound the change in delay of vertex i from both sides, i.e., decrease or increase of vertex delay.

(4) For every edge $e(Dmy(i) \rightarrow j)$ introduce the causality constraint that states that the slack for all edges in the original DAG will be non-negative after the D-phase.

$$FSDU^r(Dmy(i) \rightarrow j) = FSDU(Dmy(i) \rightarrow j) + r(j) - r(Dmy(i)) \geq 0.$$

(5) Now solve the following optimization problem, whose dual is a min-cost network flow problem [14],

$$minimize \sum_{over\ vertices\ i} \mathbf{X}_i$$
$$\equiv maximize \sum_{over\ vertices\ i} C_i \cdot (r(Dmy(i)) - r(i))$$

subject to :

$$FSDU(i \rightarrow Dmy(i)) + r(Dmy(i)) - r(i) \geq MIN\Delta D(i),$$
$$FSDU(i \rightarrow Dmy(i)) + r(Dmy(i)) - r(i) \leq MAX\Delta D(i),$$

For all edges Dmy(i) → j,

$$FSDU^r(Dmy(i) \rightarrow j) = FSDU(Dmy(i) \rightarrow j) + r(j) - r(Dmy(i)) \geq 0. \quad (10)$$

Note that the D-phase optimization is in the form of the dual of a minimum cost network flow problem, [9]. Also the constant terms in the RHS of the contraints in the D-phase can be integerized by ap-

propriate scaling, i.e., by multiplying every constant term by some power of 10 and then rounding off the product. By choosing appropriate powers of 10 arbitrary accuracy canbe maintained with almost no penalty in computational requirements. In this way, fast methods devised for integerized minimum cost network flow approaches [9] can be fruitfully employed in solving the D-phase optimization problem.

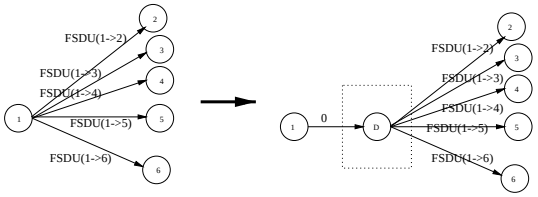

Figure 5. Illustration of circuit DAG transformation related with the addition of a dummy vertex at the output of every vertex.

2.3.2. W-phase

Once the D-phase has computed new delays (delay budgets) for all the vertices in the circuit DAG, we need to find feasible sizes for the transistors corresponding to every vertex in the circuit DAG to satisfy the delay requirements while using up minimal area. In effect we have to solve the following problem,

$$\begin{aligned} minimize \quad & \sum_{over\ vertices\ i} x_i, \\ \text{subject to :} \quad & \frac{\sum_{j \in S(V(G))} a_{ij} x_j + b_i}{x_i} \leq delay(i), \\ \equiv \quad & \sum_{j \in S(V(G))} a_{ij} x_j + b_i \leq delay(i) \cdot x_i, \\ & minsize \leq x_i \leq maxsize. \end{aligned} \tag{11}$$

It turns out that due to the non-negativity of a_{ij}, $delay(i)$ and the coefficients of x_i in the objective function, this optimization problem can be modeled as a Simple Monotonic Program (SMP) [10]. This kind of problem can be solved by a constraint relaxation procedure with worst case complexity of $O(|V||E|)$ where $|E|$ is the number of constraints and $|V|$ is the number of variables. The detail of this relaxation procedure are being omitted for lack of space, but can be found in [10]. In the W-phase, due to the restrictions on the magnitude of the change in delay budgets computed in the D-phase, the magnitude of the change in x_i, i.e., δx_i will be small. To sum up, the W-phase finds a set of sizes for the transistors in the circuit that is a minimum area solution for satisfying the delay requirements calculated by the D-phase.

2.4. Putting it All Together

Having, defined the D-phase and W-phase of the optimization strategy, we are now in a position to finally describe $MINFLOTRANSIT$, our Min-cost Flow based Transistor sizing Tool.

MINFLOTRANSIT
1.Size the circuit to meet delay requirements using TILOS.
2.Iteratively perform alternately the D-phase and W-phase optimizations, solving the problems formulated in (10) and (11) respectively.
3.Stop the iterations when the area improvement after the W-phase is negligible.

We now present an example that qualitatively illustrates the improvements provided by MINFLOTRANSIT over TILOS.

Example 1 *Figure 6 shows a simple three gate circuit to be sized. TILOS is a sensitivity based greedy heuristic that proceeds by bumping up in each pass the size of that transistor/gate that leads to maximal benefit in speed for a unit increase in area. Such a transistor/gate is called the transistor/gate with the highest sensitivity. Assume that in figure 6, both B and C are gates with identical sensitivity and A has a lower sensitivity. Therefore the paths $A \rightarrow B$ and $A \rightarrow C$ are both critical. TILOS, due to its greedy nature, will bump up the sizes of B and C in alternate passes, whereas it should be intuitively clear that sizing up A, even though it has lower sensitivity may be the better option as it speeds up both paths $A \rightarrow B$ and $A \rightarrow C$ simultaneously. In the D-phase, MINFLOTRANSIT explicitly includes constraints to evaluate the benefits of altering the sizes of gates A, B and C. It is therefore able to identify whether sizing gate A, in spite of its lower sensitivity, will be advantageous.*

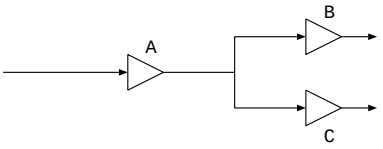

Figure 6. An example illustrating the global perspective taken by MINFLOTRANSIT which TILOS tends to overlook.

Table 1. The area savings in % of MINFLOTRANSIT over TILOS is listed. The CPU time required by TILOS and the extra time required by MINFLOTRANSIT over and above that required by TILOS are listed. The critical path of a minimum sized circuit is denoted by D_{min}.

Circuit	# Gates	Area savings over TILOS	Delay Specs.	CPU (TIME) (TILOS)	CPU (TIME) (OURS)
	# Gates				
adder32	480	$\leq$ 1%	$0.5D_{min}$	2.2s	5s
adder256	3840	$\leq$ 1%	$0.5D_{min}$	262s	608s
c432	160	9.4%	$0.4D_{min}$	0.5s	4.8s
c499	202	7.2%	$0.57D_{min}$	1.47s	11.26s
c880	383	4%%	$0.4D_{min}$	2.7s	8.2s
c1355	546	9.5%	$0.4D_{min}$	29s	76s
c1908	880	4.6%	$0.4D_{min}$	36s	84s
c2670	1193	9.1%	$0.4D_{min}$	27s	69s
c3540	1669	7.7%	$0.4D_{minsize}$	226s	335s
c5315	2307	2%	$0.4D_{minsize}$	90s	111s
c6288	2416	16.5%	$0.4D_{minsize}$	1677s	2461s
c7552	3512	3.3%	$0.4D_{minsize}$	320s	363s

Theorem 3 *$MINFLOTRANSIT$ produces minimum transistor sizing for any delay constraints.*

Proof: Let us assume that we are in some intermediate iteration at a non-optimal point. We iteratively apply the D-phase, followed by the W-phase, and it is sufficient to show that the application of each of these steps causes the objective function to reduce, while maintaining feasibility. The D-phase uses a Taylor series approximation to the constraint in Equation (6) to represent the objective function entirely in terms of the delay variables. This approximation is valid within a radius of convergence of ϵ around the current point, $\mathbf{X}$, corresponding to some radius of δ around the vector of delays. Therefore, a solution to the D-phase is a valid solution to the original problem as long as the allowable delay change is bounded by a quantity that lies within a δ-ball of the current delays. This is achieved forcing $MAX\Delta D(i)$ and $MIN\Delta D(i)$ to be small for all i.

If the current solution is not optimal, due to the convexity of the problem [1, 2], there must be another feasible point in the neighborhood of the current point that has a smaller objective function value, and this point will be found by the D-phase. In the W-phase that follows, the solution found in the D-phase is a feasible solution. Since the W-phase does not limit the change in the delay or the transistor sizes as greatly as the D-phase, its solution must have an objective function value that is no larger than the solution of the W-phase.

Therefore, since the objective function value decreases in each phase, the procedure is guaranteed to find an optimal solution to the problem.

3. SIMULATION RESULTS

Simulation results were obtained on all the combinational circuits in the ISCAS85 benchmark suite and also on ripple carry

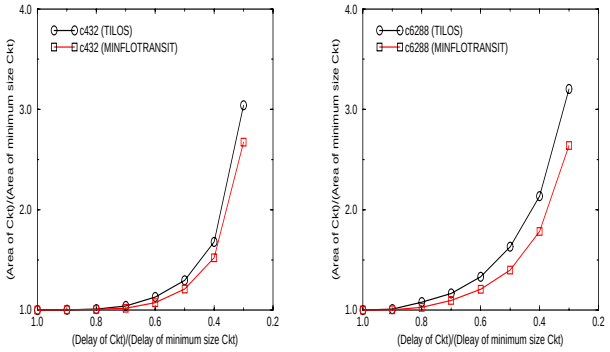

Figure 7. Comparative area-delay curves for gate sizing of two ISCAS85 benchmark circuits. The total device area of the circuits after transistor sizing with TILOS and MINFLOTRANSIT is plotted against delay, normalized with respect to the delay of a minimum sized circuit. Even though the curves look close the area benefits are actually significant. For example in the case of c6288, for a circuit with 0.5 times the delay of the minimum sized circuit, the area savings of MINFLOTRANSIT over TILOS is 14.2%.

adders of 32-256 bits. The results shown in this section are for gate sizing which as mentioned before is a special case of true transistor sizing. We implemented the TILOS algorithm as described in [15]. Starting with all transistors at minimum size, a bumpsize = 1.1 was used for the initial sizing. Iterative application of D-phase and W-phase optimization was then carried out for optimal transistor sizing. Figure 7 shows the area delay curve for representative benchmark circuits for both TILOS and MINFLOTRANSIT. The technology parameters used for simulation were obtained from [16] for 0.13μ technology. As can be seen a clear gain in performance is seen when using MINFLOTRANSIT as opposed to using TILOS.

Table 1 lists the area savings of MINFLOTRANSIT over TILOS and the CPU times required in an Ultrasparc 10 Sun workstation for sizing the ISCAS85 benchmark circuits. The tabulated results are for sizing solutions where the area penalty is within $1.5 - 1.75$ times that of a minimum sized circuit and all these correspond to points where the area penalty of sizing is reasonable. For the adders the improvement in area over TILOS is marginal thereby suggesting that adders can be easily sized by using greedy heuristics. This is not surprising since ripple carry adders have a single dominant critical path which can, possibly, be sized optimally using a heuristic like TILOS. On the other hand, however, as can be clearly seen for the ISCAS85 benchmark circuits, the area savings vary from $2\% - 16.5\%$. The overall time required by MINFLOTRANSIT is almost always (except in the small circuits c432, c499) within $2 - 4$ times of TILOS. The circuit c6288 shows an unusually high time requirement for sizing, possibly due to the fact that this circuit (a type of multiplier) has a large number of paths, many of them reconvergent. Therefore, a number of competing paths can become critical at any instance and sizing this circuit is consequently harder. It is also notable that TILOS performs poorly as compared to MINFLOTRANSIT for this particular circuit. In all our simulations only a few tens of iterations were required by MINFLOTRANSIT (except the steepest portions of the area delay curve where no more than 100 iterations were required).

4. CONCLUSIONS

We presented a new transistor sizing tool MINFLOTRANSIT that has shown impressive run-time behavior over various benchmarks in the ISCAS85 benchmark suite. MINFLOTRANSIT is a two-phase iterative-relaxation based technique. The first, D-phase has a worst case complexity of $O(|V||E|log(log|V|))$, the second, W-phase has a worst case complexity of $O(|V||E|)$. The run-time behavior of this tool is comparable to TILOS but it is guaranteed to produce optimal transistor sizes for meeting the delay constraints. Although Elmore delay models were used in this paper for illustration, MINFLOTRANSIT is valid for a larger class of delay models characterized by the *monotonic decomposition* property in Definition 2.

REFERENCES

[1] J. P. Fishburn and A. E. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *Proceedings of the 1985 International Conference on Computer-Aided Design*, pp. 326–328, November 1985.

[2] S. Sapatnekar, V. Rao, P. Vaidya, and S. Kang, "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization.," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1621–1634, November 1993.

[3] J.-M. Shyu, A. L. Sangiovanni-Vincentelli, J. Fishburn, and A. Dunlop, "Optimization-based Transistor Sizing," *IEEE Journal on Solid State Circuits*, vol. 23, no. 2, pp. 400–409, 1988.

[4] D. Marple, "Performance Optimization of Digital VLSI Circuits," *Technical Report CSL-TR-86-308, Stanford University*, October 1986.

[5] D. Marple, "Transistor Size Optimization in the Tailor Layout System," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 43–48, June 1989.

[6] H. Y. Chen and S. M. Kang, "iCoach: A Circuit Optimization Aid for CMOS High-Performance Circuits," *Intergration, the VLSI Journal*, vol. 10, pp. 185–212, January 1991.

[7] Z. Dai and K. Asada, "MOSIZ: A Two-Step Transistor Sizing Algorithm based on Optimal Timing Assignment Method for Multi-Stage Complex Gates," in *Proceedings of the 1989 Custom Integrated Circuits Conference*, pp. 17.3.1–17.3.4, May 1989.

[8] C. Chen, C. N. Chu, and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," in *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*, pp. 617–624, November. 1998.

[9] A. V. Goldberg, M. D. Grigoriadis, and R. E. Tarjan, "Use of Dynamic Trees in a Network Simplex Algorithm for the Maximum Flow Problem," *Mathematical Programming*, vol. 50, pp. 277–290, June 1991.

[10] M. C. Papaefthymiou, "Asymptotically Efficient Retiming under Setup and Hold Constraints," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 288–295, Nov. 1998.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. McGraw-Hill New York, NY, 1990.

[12] G. Strang, *Linear Algebra and its Applications*. Harcourt Brace Jovanovich, Publishers, San Diego, CA, 1988.

[13] V. Sundararajan and K. K. Parhi, "Low Power Gate Resizing Using Buffer-Redistribution," in *Proceedings of the Twentieth Anniversary Conference on Advanced Research in VLSI*, (Atlanta, GA), pp. 170–184, March 1999.

[14] V. Chvatal, *Linear Programming*. W. H. Freeman and Company, New York, NY, 1983.

[15] A. E. Dunlop, J. P. Fishburn, D. D. Hill, and D. D. Shugard, "Experiments Using Automatic Physical Design Techniques for Optimizing Circuit Performance," in *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, (Urbana, IL), pp. 216–220, August 1989.

[16] J. Cong, "Challenges and Opportunities for Design Innovations in Nanometer Technologies," *Technical Report, Semiconductor Research Corporation*, 1997.