

# A Joint Gate Sizing and Buffer Insertion Method for Optimizing Delay and Power in CMOS and BiCMOS Combinational Logic

Kerry S. Lowe, *Member, IEEE*, and P. Glenn Gulak, *Senior Member, IEEE*

**Abstract**— This paper presents the first reported joint gate sizing and buffer insertion method for minimizing the delay of power constrained combinational logic networks that can incorporate a mixture of unbuffered and buffered gates (or mixture of CMOS and BiCMOS gates). In the method, buffered gates in a network are decided on by an iterative process that uses a sequence of sizing optimizations where after each sizing optimization an update to the selection of buffered gates is made. In this way, high drive capability buffered (i.e., BiCMOS) gates with sufficiently low fan-out are identified and replaced with a lower power unbuffered (i.e., CMOS) version. As well, the optimality of the final design is assessed based on a lower-bound delay value that is calculated. Experimental results have confirmed the efficiency and utility of the proposed method. In 8-b adder or  $8 \times 8$  b multiplier networks, just two iterations are sufficient to achieve a delay that is at worst within 0.6% of its final optimized value and at worst within 10% of the lower-bound value. In the design of BiCMOS networks, it is seen that a speed advantage (at equivalent power) can be systematically achieved by using a mix of CMOS and BiCMOS gates versus using all CMOS or all BiCMOS gates and that this advantage increases with the tightness of the power constraint and with load capacitance.

**Index Terms**— Buffer insertion, combinational logic circuits, gate sizing, optimization methods.

## I. INTRODUCTION

**G**ATE sizing and buffer insertion are two closely related techniques for optimizing the delay and power performance of a combinational logic network. *Gate sizing* optimizes performance by adjusting the *size* of each logic gate in the network (where the *size* of a logic gate is a scale factor that defines the dimensions of the transistors used in the gate) and uses the fact that increasing the size of a gate reduces its output resistance and hence delay but increases its power dissipation as well as its input capacitance (resulting in a higher delay for preceding gates). Conversely, *buffer insertion* optimizes

performance by adding an output buffer circuit to selected logic gates in the network and uses the fact that buffering a gate reduces its output resistance but increases its *internal delay* (i.e., unloaded delay) and its power dissipation. Gate sizing and buffer insertion are, thus, similar in that they both attempt to exploit a tradeoff between capacitive drive capability (i.e., output resistance) and power dissipation in a selected set of gates. As well, both techniques preserve the logic function and topology of a network and are applicable to design in a *standard cell* based methodology where logic gates and buffer circuits are selected from a predefined set of alternatives that comprise a *cell library*. The techniques complement the capabilities of logic synthesis tools and provide a means for refining technology mappings in both the prelayout and postlayout phases of design.

The close relationship between gate sizing and buffer insertion suggests that a method capable of joint optimization could eliminate the need for separate gate sizing and buffer insertion tools and identify superior designs. Moreover, in BiCMOS networks that can incorporate both BiCMOS and CMOS gates, a joint optimization method will tell designers where to use BiCMOS gates and where to use CMOS gates since a BiCMOS logic gate acts as a form of high-quality buffered CMOS logic gate. By comparing optimized BiCMOS and CMOS networks, designers will be able to calculate the performance advantage a BiCMOS technology achieves over CMOS for a given application.

The objective of this work is to develop a joint gate sizing and buffer insertion method and assess its utility in the design of typical CMOS and BiCMOS networks. We seek a method that is capable of constrained optimization and in particular the minimization of network delay subject to a constraint in network power dissipation. As well, the method is to be suitable for a continuous type cell library where gate and buffer sizes are selectable across a given continuous range. The allowed transformations in the joint optimization are, thus, as shown in Fig. 1. Namely, 1) add a buffer circuit to the output of an existing gate and optimize the size of the buffer circuit and gate and 2) just optimize the size of an existing gate without adding a buffer circuit. Note that we exclude the case of discrete sizing and assume that the given network topology has been optimized for logic depth and buffer tree expansions. Buffer tree optimization, unlike gate buffering, tries to resynthesize a critical path of a logic network so that no

Manuscript received September 20, 1994; revised February 11, 1997. This work was supported by Bell Northern Research Ltd. and the Natural Sciences and Engineering Research Council of Canada. This paper was recommended by Associate Editor M. Fujita.

K. S. Lowe was with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont. M5S 1A4 Canada. He is now with Nortel, Ottawa, Ont. K1Y 4H7 Canada.

P. G. Gulak is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont. M5S 1A4 Canada.

Publisher Item Identifier S 0278-0070(98)04632-6.

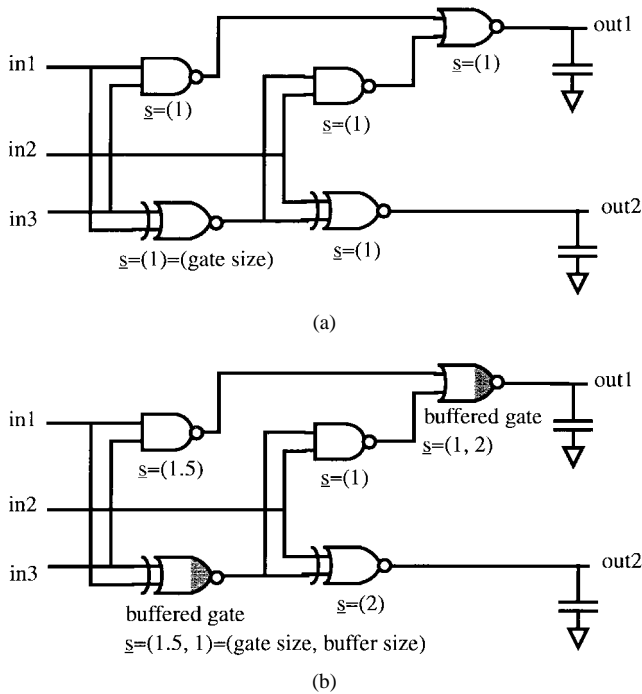


Fig. 1. Illustration of joint gate sizing and buffer insertion optimization showing (a) unoptimized network and (b) optimized network.

gate along this path drives a large number of other gates. This is generally done by identifying the *critical* and the *noncritical* signals emanating from each high fan-out gate. The critical signals are left as is (i.e., unbuffered). The noncritical signals are driven by a buffer that is added to the output of the high fan-out gate.

#### A. Prior Work

Unfortunately, previous methods concerning joint optimization or mixed CMOS/BiCMOS optimization appear to be inappropriate for our objective. In the method of Chen [1], [2], optimization of a mixed CMOS/BiCMOS network begins with an all CMOS implementation and proceeds in a *greedy* manner by iterating a three step procedure of 1) identifying critical paths and gates, 2) upward sizing transistors along the most critical path until no further improvement is obtained, and 3) switching the most *delay sensitive* CMOS gate along this path to BiCMOS if beneficial. In this way, Chen's method achieves CMOS versus BiCMOS gate selection capability without greatly increasing CPU time above that required for sizing optimization alone. However, the optimality of the method is not assessed and remains unclear. As well, the problem of delay minimization with respect to a power (or area) constraint is not considered. Conversely, previous methods by Duchene [3] and by Wissel [4] for mixed CMOS/BiCMOS networks have limited their scope to performance prediction and do not provide a means for finding an optimized design. Both methods are stochastic and estimate the fractional usage of BiCMOS and CMOS gates based on a distribution of node capacitances and, in [4], a distribution of node delay slack. The distributions are static and, as such, the methods do not incorporate the effects due to gate sizing.

In contrast to joint optimization, methods for continuous case gate sizing have been extensively investigated since the pioneering work of Ruehli [5], [6]. This work studied gate sizing to minimize power subject to a delay constraint and demonstrated the basic feasibility of exploiting nonlinear programming techniques. However, a crude gate delay model was used that did not consider input capacitance variation with gate size and the problem formulation that was used resulted in an objective function that had discontinuous first derivatives and singular points. Hedelund [7] improved the formulation by introducing lower and upper size constraints and modeled the dependence of capacitance on gate size. His and other methods [8], however, only handled a single delay path at a time making full network optimization tedious and suboptimal. Subsequently, Fishburn [9], Matson [10], and Marple [11] addressed multipath optimization. Fishburn, studying transistor sizing, showed that a multipath delay minimization problem could be formulated as a type of nonlinear program known as a posynomial program [12] that has the desirable property of being unimodal. Marple generalized this formulation to include the constrained case, identified an alternative posynomial formulation, and showed that gate sizing can be formulated as posynomial program provided the gate model is of posynomial form. Since a posynomial program is unimodal, the globally optimal solution for gate sizing can always be found using a gradient based numerical algorithm for constrained optimization such as the augmented Lagrangian or projected Lagrangian algorithms [13]. More recently, Sapatnekar [14] has used a novel convex programming algorithm to achieve global optimality with improved efficiency while Berkelaar [33] has investigated linear programming to improve efficiency.

#### B. Overview of this Work

To exploit the prior work in gate sizing, we investigate a joint gate sizing and buffer insertion method where buffered (i.e., BiCMOS) gates in a network are decided on by an iterative process that uses a sequence of sizing optimizations where after each sizing optimization an update to the selection of buffered gates can be made. In this way, high drive capability buffered (i.e., BiCMOS) gates with sufficiently low fan-out are identified and replaced with a lower power unbuffered (i.e., CMOS) version. As well, the optimality of the final design is assessed based on a lower-bound delay value that is calculated. Each required sizing optimization is formulated as a posynomial program and solved using a nonlinear programming routine. To assess the utility of the method, example adder, multiplier, parity generator, and decoder networks are optimized using a cell library for a 0.8- $\mu\text{m}$  BiCMOS process.

This paper is organized as follows. Section II provides basic definitions relevant to gate sizing and buffer insertion optimization while Section III describes network delay and power modeling. Section IV provides a formal description of the joint gate sizing and buffer insertion problem. Section V describes the optimization algorithm and its implementation. Example results are presented in Section VI. A discussion

of the algorithm and results follows in Section VII, while conclusions are presented in Section VIII.

## II. BASIC DEFINITIONS

### A. Network Topology

The *topology* of a combinational logic network to be optimized is represented as a directed acyclic graph following Definition 1. The graph specifies the logic operation and I/O connections for each gate in the network. The graph also specifies the wire capacitances. These capacitances can be estimated from a physical layout (or layout sketch) of the network (that gives the length of wire that a gate drives) and from the appropriate capacitance per length factor (that characterizes the wire interconnect).

*Definition 1:* The topology of a combinational logic network is a directed acyclic graph  $\Gamma = (\{\nu\}, \{e\}, \{C_w\})$ . The set of vertices is given by  $\{v\} = \{k\} \cup \{n\} \cup \{m\}$  where  $\{k\}$  is the set of  $K$  gates,  $\{n\}$  is the set of  $N$  network inputs, and  $\{m\}$  is the set of  $M$  network outputs. Each vertex  $k \in \{k\}$  implements a specified single-output Boolean logic primitive. There is an edge  $(k, j) \in \{e\}$  if and only if the output of vertex  $k$  is connected with an input of vertex  $j$ . For each vertex  $k$ , the term  $C_w^k$  is the net wire capacitance of the edges connected to the output of vertex  $k$ .  $\square$

In Definition 1, wire capacitance is a constant and wire resistance is not considered. Modeling wire capacitance as a constant is reasonable provided optimization does not force a change in the relative placement of gates in the layout. Setting an appropriate value for the maximum allowed gate size (see Definition 3) can ensure this is met. Neglecting resistance effects is reasonable provided wire lengths are less than about 1 mm [15].

### B. Gate Size and Logic Family

Given a network graph, the task of optimization seeks to find the best logic gate realization for each vertex  $k$ . Different realizations of a vertex  $k$  correspond to a different choice of *logic family* or *gate size*, or both, as defined below.

*Definition 2:* A logic family is a unique set of transistor circuit designs for the realization of a collection of logic primitives (such as  $n$ -input NOR, NAND, and XOR). A logic primitive is mapped to one and only one of these designs but all designs share a common set of operational characteristics. These characteristics are: 1) signal voltage level requirements, 2) supply voltage level requirements, 3) signal polarity requirements (i.e., single-ended or differential), and 4) clock requirements (i.e., static or dynamic). The transistor sizes (i.e., field-effect transistor (FET) gate widths and bipolar emitter lengths) that are specified in a design are the minimum allowed values consistent with proper operation and technology design rules.  $\square$

Typically, the characteristics of a logic family will conform to a popular logic standard (e.g., ECL, TTL, and 5-V CMOS). The choice of logic family and standard influences network performance including noise and process tolerance behavior and must be considered a critical design parameter.

In designing a network, more than one logic standard may be used but then logic signal translation circuitry and additional power supply levels may be needed [16]. Moreover, since the noise margins of the standards will in general differ, network optimization cannot be accomplished on the basis of delay and power metrics alone.

As stated in Definition 2, the specification of a logic family for gate  $k$  identifies the minimum transistor sizes that can be used. A scaled up version of this gate is identified by specifying its size relative to the minimum allowed.

*Definition 3:* The size of logic gate  $k$  is denoted by the vector  $\mathbf{s}^k$  and specifies its transistor sizes relative to the minimum allowed value. The dimension of  $\mathbf{s}^k$  is equal to the number of independent stages that are deemed to comprise gate  $k$ . Independent stage  $x$  is a subcircuit of gate  $k$  whose transistors may be uniformly scaled by a relative factor  $s_x^k$  such that all node voltage levels in gate  $k$  are preserved. In this scaling, only the FET widths and bipolar emitter lengths are scaled. FET lengths and emitter widths are kept constant. If gate  $k$  has  $X$  stages then  $\mathbf{s}^k = (s_1^k, s_2^k, \dots, s_X^k)$ , where the stages are numbered in increasing order from the input (stage 1) to the output (stage  $X$ ). For a stage  $x$  of gate  $k$ ,  $(s_x^k)_{\min} \leq s_x^k \leq (s_x^k)_{\max}$ , where  $(s_x^k)_{\min}$  is the minimum allowed size and  $(s_x^k)_{\max}$  is the maximum allowed size.  $\square$

Note that the independent stages of a gate are fully defined by a given cell library (see Definition 5) and by whether or not the gate is buffered. Typically, an unbuffered gate will be just one-stage and a buffered gate just two-stage or just 1-stage (see Definition 4).

### C. Buffered and Unbuffered Gate

A *buffered gate* and an *unbuffered gate* are logic gates that differ in logic family but share a close relationship. A buffered gate is defined relative to an unbuffered gate following Definition 4.

*Definition 4:* A buffered gate is formed by adding a buffer circuit to the output of a one-stage gate known as the corresponding unbuffered gate. The buffer circuit strives to improve drive capability but does not alter logic function. The resulting buffered gate may be either a one-stage gate or a two-stage gate. In a two-stage buffered gate, gate and buffer circuitry can be independently sized. The size of two-stage buffered gate  $k$  is written  $\mathbf{s}^k = (s_1^k, s_2^k)$  where  $s_1^k$  is the front-end gate size and  $s_2^k$  is the buffer size. In a one-stage buffered gate, gate and buffer circuitry cannot be independently sized. Instead, the ratio  $r^k = s_2^k/s_1^k$  of buffer size to front-end gate size is considered fixed. The size of one-stage buffered gate  $k$  is written  $\mathbf{s}^k = (s_1^k)$ . A one-stage buffered gate may directly correspond (in transistor circuit topology) to a front-end gate plus a buffer. Alternatively, a one-stage buffered gate may be fitted to correspond to a pseudo front-end gate plus a buffer. In this case,  $r^k$  is a fitted parameter. A two-stage buffered gate is physically realized as a double-cell gate. A one-stage buffered gate is physically realized as a single-cell gate.  $\square$

Examples of buffered gates and their corresponding unbuffered form are shown in Fig. 2. The buffered gate shown in Fig. 2(b) can be sized as a two-stage gate since the front-end

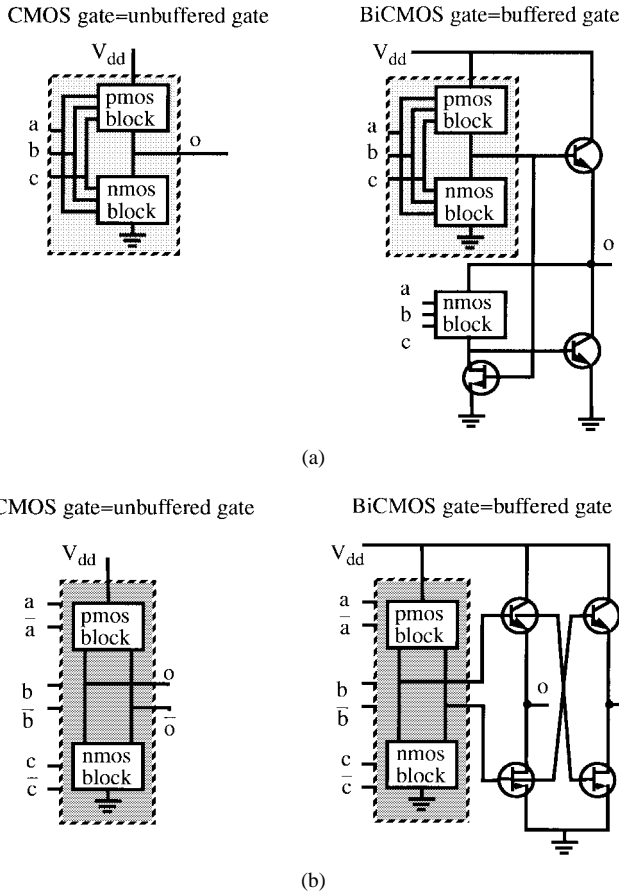


Fig. 2. Examples of BiCMOS gates that act as a form of buffered CMOS gate where (a) are single-ended gates and (b) are double-ended gates.

gate operation is independent of buffer circuit. The buffered gate of Fig. 2(b) could also be sized as a one-stage gate. This is done by specifying a fixed ratio for the buffer size to the front-end gate size. The buffered gate shown in Fig. 2(a) must be sized as a one-stage gate. This is because the buffered gate cannot be directly decomposed into an independently sizable front-end stage plus an independently sizable buffer stage.

#### D. Cell Library and Aggregate Cell Library

In optimizing a network, the choice of realizations for each gate is assumed to be restricted to a given set of logic families and sizes. This set of all available logic gates is called the *aggregate cell library*. The set of available logic gates belonging to a common logic family constitutes a *cell library*.

**Definition 5:** A cell library is the set of all logic gates (of different size and logic primitive) of a given logic family that have been characterized and are available for use in the implementation of a logic network. The set of gates in cell library  $\psi$  is denoted by  $\{\psi\}$ . If more than one cell library is available, each is labeled as  $\psi_1, \psi_2, \dots, \psi_B$ , where  $B$  is the number of cell libraries. Then the aggregate cell library is called  $\Psi$  and is defined by  $\{\Psi\} = \{\psi_1\} \cup \{\psi_2\} \cup \dots \cup \{\psi_B\}$ . The cell libraries that comprise an aggregate library are assumed to have intercompatible logic family characteristics

TABLE I  
VARIOUS AGGREGATE CELL LIBRARIES RELEVANT  
TO CMOS AND BiCMOS TECHNOLOGIES

Aggregate Library			Cell Libraries Included		
Name	Technology	Description	Buffered		
			CMOS $\Psi_a = \Psi_1$	CMOS $\Psi_b = \Psi_2$	BiCMOS $\Psi_b = \Psi_3$
$\Psi_1$	CMOS	CMOS	✓		
$\Psi_2$	CMOS	Buffered CMOS		✓	
$\Psi_{1,2}$	CMOS	mixed CMOS/buffered CMOS	✓	✓	
$\Psi_3$	BiCMOS	BiCMOS			✓
$\Psi_{1,3}$	BiCMOS	mixed CMOS/BiCMOS	✓		✓

(as listed in Definition 2). The cell library from which gate  $k$  is selected is denoted by  $\psi^k$ .  $\square$

#### E. Network and Cell Technology

The process *technology* of a network (or cell) is defined based on the types of transistors used to implement the network (or cell). A greater number of transistor types generally implies a more complex and costly process.

**Definition 6:** The technology of a network is defined by the set of transistor types used to implement the network. Different transistor types arise from differences in: 1) substrate type (silicon, gallium arsenide, etc.), 2) transistor action type (unipolar or bipolar), 3) majority carrier type (electrons or holes), and 4) gate/emitter junction type (MOS, PN, Schottky, etc.). In a similar manner, the technology of a cell (cell library) is defined by the set of transistor types used to implement the cell (cell library).  $\square$

Following Definition 6, a CMOS technology network (cell) is implemented using Si *N*-channel enhancement MOSFET (NMOS) and silicon *P*-channel enhancement MOSFET (PMOS) transistor types. A BiCMOS technology network (cell) is implemented using NMOS, PMOS, and Si NPN bipolar transistor types. A complementary BiCMOS technology network (cell) is implemented using NMOS, PMOS, NPN, and PNP transistor types.

Definitions 2–6, thus, provide the flexibility for describing the range of designs relevant to a BiCMOS technology. For example,  $\psi_1$  can be defined as a one-stage CMOS cell library containing unbuffered gates,  $\psi_2$  can be defined as a two-stage CMOS cell library containing buffered gates, and  $\psi_3$  can be defined as a BiCMOS cell library containing buffered gates. The aggregate library for mixed CMOS/BiCMOS networks is  $\{\psi_1\} \cup \{\psi_3\}$ , while the aggregate library for CMOS/buffered CMOS networks is  $\{\psi_1\} \cup \{\psi_2\}$ . Other relevant aggregate cell libraries are listed in Table I. In Table I,  $\psi_a$  denotes a library of unbuffered gates and  $\psi_b$  denotes a library of buffered gates.

### III. DELAY AND POWER MODELING

A set of selections for the cell library and size for each gate in a network is written as  $\{\Psi^k\}, \{s^k\}$  and defines a specific implementation of the network. For such an implementation, the resulting network delay and power are calculated as follows.

### A. Network Delay

*Definition 7:* The delay of a network is given by

$$T = \max_{\varphi} \left[ \sum_{k \in \{k\}_{\varphi}} \tau^k \right]. \quad (1)$$

The term  $\{k\}_{\varphi}$  denotes the set of gates in the  $\varphi$ th sensitizable delay path ( $\varphi = 1, 2, \dots, \Phi$ ) that starts at a network input and terminates at a network output. The term  $\tau^k$  denotes the contribution of the delay on path  $\varphi$  that is attributed to gate  $k$  and is modeled as

$$\tau^k = \alpha^k + \gamma^k (C_{\text{out}}^k / C_{\text{in}}^k) = \alpha^k + \gamma^k \xi^k \quad (2)$$

where  $\xi^k$  is called the fan-out of gate  $k$ . The term  $C_{\text{in}}^k$  is the capacitance of the relevant input of gate  $k$  for path  $\varphi$  and is given by

$$C_{\text{in}}^k = C_1^k s_1^k \quad (3)$$

where  $C_1^k$  is the relevant input capacitance of a unit size first stage of gate  $k$ . The term  $C_{\text{out}}^k$  in (2) is the output capacitance of gate  $k$  and is given by

$$C_{\text{out}}^k = (C_w^k + C_L) + \sum_{j \in \{j\}^k} C_{\text{in}}^j \quad (4)$$

where  $C_w^k$  is the wire capacitance,  $C_L$  is the external load capacitance (if gate  $k$  is connected to a network output), and the summation is the total input capacitance of the set of gates connected to the output of gate  $k$  (since  $\{j\}^k$  is the set of vertices connected to the output of  $k$ ). The term  $\gamma^k$  in (2) is the loaded delay factor for the relevant input of gate  $k$  and is given by

$$\gamma^k = \gamma_X^k \frac{C_1^k s_1^k}{C_X^k s_X^k} \quad (5)$$

where  $\gamma_X^k$  is the loaded delay factor of the final stage  $X$  of gate  $k$ . Finally, the term  $\alpha^k$  in (2) is the internal delay for the relevant input of gate  $k$  and is given by

$$\alpha^k = \sum_{x=1}^X \alpha_x^k + \sum_{x=1}^{X-1} \gamma_x^k \frac{C_{x+1}^k s_{x+1}^k}{C_1^k s_x^k} \quad (6)$$

where  $\alpha_x^k$  is the internal delay for the relevant input of stage  $x$  of gate  $k$ .  $\square$

$T$ , as defined, represents the delay of the longest path(s) of a network. It is assumed that each of the  $\Phi$  paths considered in (1) can be exercised by a primary input sequence and is, hence, not false [17]. Such a set of paths can be found by the method of Chen [18].

In computing  $T$  by Definition 7, the values for the stage parameters  $\{\alpha_x^k, \gamma_x^k, C_x^k\}$  that are used to model gate  $k$  on

path  $\varphi$  can be specific to the input and state of gate  $k$ . Thus,  $T$  includes the fact that the delay through a gate can depend both on the particular input that is switched and on the state (i.e., low or high) of nonswitching inputs. Similarly, the delay dependence on the direction of input switching (i.e., low-to-high or high-to-low) can be included (however, this complication is eliminated by ensuring all gates have a symmetric output or have a differential logic structure).

Additionally, slew rate dependence on gate delay can be accommodated by noting Hedenstierna [19], and Hoppe [20] and adjusting internal delay according to

$$\alpha_x^k = \alpha_{x0}^k + \alpha_{x1}^k t_r \quad (7)$$

where  $t_r$  is the rise (or fall) time of the input signal and  $\alpha_{x0}^k$  and  $\alpha_{x1}^k$  are called the independent and dependent, respectively, internal delay parameter. Thus, gates connected to a network input can directly use (7) if the input rise time is specified. For other gates, rise time can be determined from an analysis of the gate  $i$  that drives the input to gate  $k$ . In particular

$$t_r = a + b \frac{\gamma_X^i C_{\text{out}}^i}{C_X^i s_X^i} \quad (8)$$

where  $a$  and  $b$  are constants for gate  $i$  [21].

### B. Network Power

*Definition 8:* The power dissipation of a network is given by

$$P = \sum_{k=1}^K \left( C_{\text{out}}^k \eta^k [V_{\text{out}}^k]^2 + \rho^k s_1^k \right) \quad (9)$$

where

$$\rho^k = \sum_{x=1}^{X^k} (\rho_x^k s_x^k / s_1^k) \quad (10)$$

and

$$\rho_x^k = p_x^k + \varepsilon_x^k \eta^k. \quad (11)$$

In (9), the term  $V_{\text{out}}^k$  is the output voltage swing of gate  $k$ . The term  $\eta^k$  is the average number of output transitions per second for gate  $k$  and is called the transition density [22] of gate  $k$  and is assumed to be independent of sizing. The term  $\rho^k s_1^k$  is the internal dissipation of gate  $k$  and is obtained by summing the internal dissipation of each stage of gate  $k$ . A unit size stage  $x$  of gate  $k$  has a internal static power dissipation of  $p_x^k$  and internal dynamic energy dissipation of  $\varepsilon_x^k$ . For  $x > 1$ ,  $\varepsilon_x^k$  includes the energy required to charge/discharge the input capacitance of the stage.  $\square$

Note that  $P$  as defined incorporates both static and dynamic power dissipation. Static (i.e., dc-leakage) dissipation is modeled by the  $p_x^k$  terms. Dynamic dissipation is modeled through: 1) the  $C_{\text{out}}^k \eta^k [V_{\text{out}}^k]^2$  terms that account for the power to charge/discharge gate input capacitances and interconnect capacitances and 2) the  $\varepsilon_x^k$  terms that account for short circuit dissipation [23] as well as the switching energy to

charge/discharge the internal nodes and (except for the first stage) input capacitance of the stages. For the first stage,  $\epsilon_1^k$  does not include the energy required to charge/discharge its input capacitance since this capacitance is driven by a preceding gate.

C. Model Parameters

The stage parameters  $\{\alpha_x^k, \gamma_x^k, C_x^k, \epsilon_x^k\}$  that are used to model gate delay and power are constants defined by the given cell library. They are independent of stage size and readily calculated by circuit simulation. The parameters  $\alpha_x^k$  and  $\gamma_x^k$  are found by measuring stage delay for different fan-out loadings and taking the intercept and slope, respectively, of a straight line fit. The parameter  $C_x^k$  is found by measuring the delay for a known capacitive load and using the straight line fit to determine the equivalent fan-out of this capacitive load. The parameter  $\epsilon_x^k$  is found by causing the output of the stage to change state and integrating over the switching period the product of the current supplied to the stage times the supply voltage. An illustration of the delay and power relationships for 1-stage unbuffered and 2-stage buffered gates are shown in Fig. 3. Note that delay and power relationships for a 2-stage buffered gate are defined for all choices of buffer size once the front-end parameters and buffer parameters are known.

IV. THE JOINT GATE SIZING AND BUFFER INSERTION PROBLEM

Using Definitions 1–8, a continuous case joint gate sizing and buffer insertion problem for the case of delay minimization subject to a power constraint can be formulated mathematically as Problem 1 below. The formulation for the case of power minimization subject to a delay constraint is similar except: 1) power constraint (12) is removed and 2)  $T$  in (13) is replaced by the specified delay constant  $T_{\max}$

Problem 1: Joint gate sizing and buffer insertion for delay minimization with power constraint  $P_{\max}$  and continuous case sizing.

- Given: 1) Network  $\Gamma$  with external capacitance  $C_L$  and  
 2) Library cells  $\{\psi_a\} \cup \{\psi_b\}$  where  $\{\psi_a\}$  are  $X = 1$  stage gates (and represent the *unbuffered* gates) and  $\{\psi_b\}$  are *unbuffered* versions of the gates in  $\{\Psi_a\}$  and are  $X = 2$  or  $X = 1$  stage.

Find: Network implementation that minimizes delay  $T$ .

Subject to: 1) Power constraint  $P/P_{\max} \leq 1$  (12)

2) Path constraints  $(1/T) \sum_{k \in \{k\}, \varphi} \tau^k \leq 1, \forall \varphi$  (13)

and

3) Size constraints  $(s_1^k)_{\min}/s_1^k \leq 1, s_1^k/(s_1^k)_{\max} \leq 1, \forall k$  (14)

$(s_2^k)_{\min}/s_2^k \leq 1, s_2^k/(s_2^k)_{\max} \leq 1, \forall k$  with  $X = 2$ . (15)

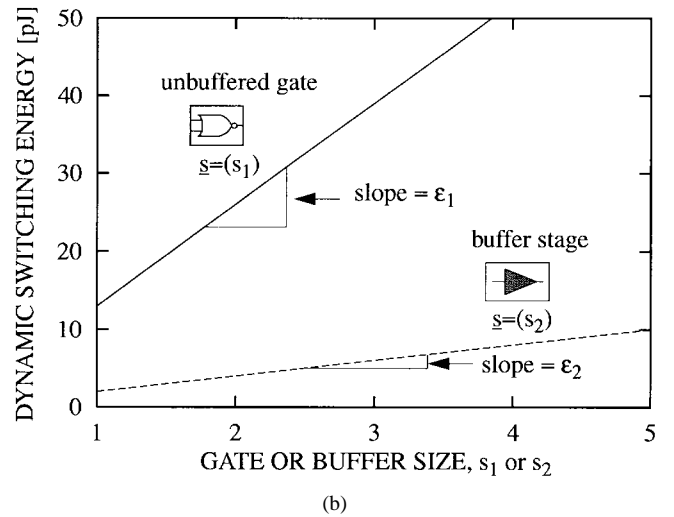
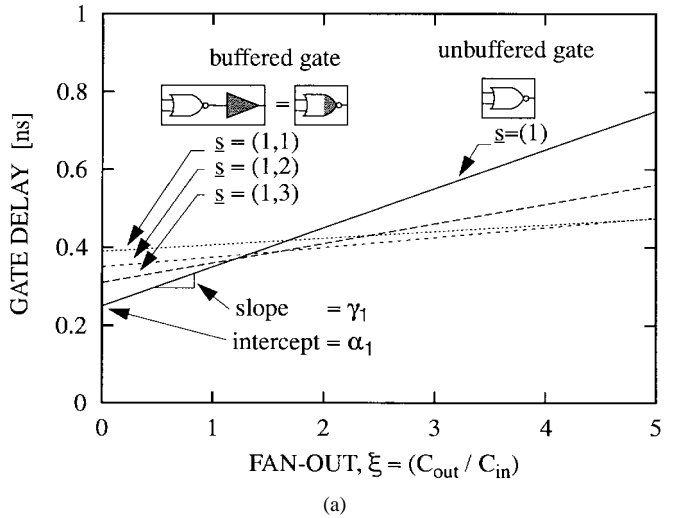


Fig. 3. Example of (a) delay modeling and (b) power modeling for 1-stage unbuffered and 2-stage buffered logic gate.

In Problem 1,  $C_L$  is a specified external load capacitance connected to each of the network outputs and represents the input capacitance of a succeeding latch or output cell. As indicated in (4),  $C_L$  contributes to the output capacitance  $C_{out}^k$  of each output gate. The size constraints (14)–(15) in Problem 1 are stated in a general form. Nodes of the same primitive will typically be specified to have the same allowed size range since they share a common aggregate cell library. Nodes connected to one of the network inputs will typically be specified to have a fixed size for its first stage since this fixes the input capacitance seen at a network input and ensures that the optimization will not alter the delay of the circuitry that drives the inputs of network  $\Gamma$ . If the input capacitance is the same for all input gates, a *network fan-out*  $\xi_\Gamma$  can be defined as the ratio of load to input capacitance.

As indicated in Problem 1,  $\psi_b$  is either a library of one-stage gates or a library of two-stage gates. If  $\psi_b$  is a library of two-stage gates, the first stage of a gate in  $\psi_b$  corresponds to a gate in library  $\psi_a$  while the second stage is a buffer (that can be sized independently of the first stage). In this case, Problem 1 is classified as a joint gate sizing and buffer insertion problem

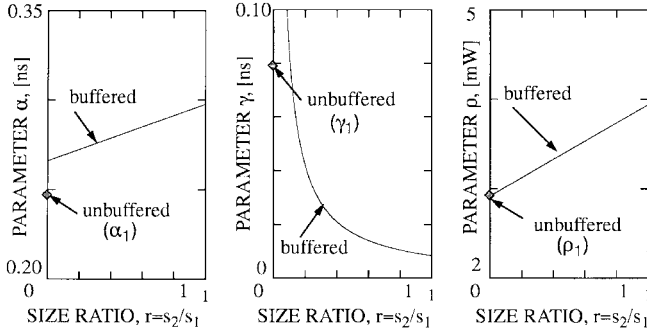


Fig. 4. A gate selection space that compares the characteristics of unbuffered gates (shown as the point at  $r = 0$ ) and buffered gates (shown as the solid curve).

with two-stage buffered gates. If  $(s_1^k)_{\min} = (s_1^k)_{\max}, \forall k$ , Problem 1 degenerates to a buffer insertion problem. If  $\{\psi_b\} = \emptyset$  or  $\{\psi_a\} = \emptyset$ , Problem 1 degenerates to a continuous case gate sizing problem. If  $\psi_b$  is a library of one-stage gates,  $\psi_b$  represents a library of buffered gates having a fixed buffer to gate size ratio  $r^k = s_2^k/s_1^k$ . In this case, Problem 1 is classified as a joint gate sizing and buffer insertion problem with one-stage buffered gates. Problems of this type can also be described as a *mixed-gate style* problem. In such a problem, a gate in  $\psi_b$  may directly correspond (in transistor circuit topology) to a gate in  $\psi_a$  plus a buffer. Alternatively, a gate in  $\psi_b$  may be fitted to correspond to a gate in  $\psi_a$  plus a buffer where  $r^k$  and  $\{\alpha_2^k, \gamma_2^k, \varepsilon_2^k, C_2^k\}$  are fitted parameters. For example, set  $r^k = 1$  then

$$\alpha_2^k = \frac{\alpha_{1b}^k - \alpha_1^k}{\gamma_1^k} \quad (16)$$

$$\gamma_2^k = \gamma_{1b}^k \quad (17)$$

$$\varepsilon_2^k = \varepsilon_{1b}^k - \varepsilon_1^k \quad (18)$$

$$C_2^k = C_1^k = C_{1b}^k \quad (19)$$

where  $\{\alpha_{1b}^k, \gamma_{1b}^k, \varepsilon_{1b}^k, C_{1b}^k\}$  are for the one-stage buffered gate  $k$  of library  $\psi_b$  and  $\{\alpha_1^k, \gamma_1^k, \varepsilon_1^k, C_1^k\}$  are the corresponding one-stage unbuffered gate of library  $\psi_a$ .

The different problem types that Problem 1 incorporates are contrasted by a plot of the *selection space* for gate  $k$  as illustrated in Fig. 4. Such a plot shows how the gate parameters  $\{\alpha^k, \gamma^k, \rho^k\}$  as given by (6), (5), and (10) vary with the choice of library and size ratio  $r^k$ . (Note that these parameters are independent of size  $s_1^k$ .) A library choice of  $\psi_a$  (unbuffered gate) corresponds to the diamond point at  $r^k = 0$ . A library choice of  $\psi_b$  (buffered gate) corresponds to a point at  $r^k > 0$  along the solid curve shown. For a joint problem with two-stage buffered gates, the choice of  $r^k$  for a buffered gate is variable as  $r_{\min}^k \leq r^k \leq r_{\max}^k$ , where  $r_{\max}^k \leq (s_2^k)_{\max}/(s_1^k)_{\min}$  and  $r_{\min}^k \geq (s_2^k)_{\min}/(s_1^k)_{\max}$ . For a *mixed-gate style* problem, the choice of  $r^k$  for a buffered gate is fixed.

Note from Fig. 4 that a gate selection space is disjoint (for other than a sizing problem). Consequently, Problem 1 is classified as a type of discrete choice gate selection problem. Such problems, are known to be NP-complete (for even special case network topologies such as chains and trees) [24].

## V. JOINT GATE SIZING AND BUFFER INSERTION ALGORITHM

The proposed algorithm for solving Problem 1 is labeled Algorithm 1 and has the overall structure outlined in Fig. 5. First, a lower-bound delay calculation is performed. Next, an initial selection of buffered nodes is made based on this lower-bound result. A sizing optimization is then performed. Finally, if this sizing optimization reveals that some nodes are unnecessarily buffered, they are replaced with an unbuffered form (i.e.,  $\psi_b^k \rightarrow \psi_a^k$ ) and another sizing optimization is performed. The sizing optimization step solves a continuous case gate sizing problem.

In Algorithm 1, buffered gates are decided by an iterative process that uses a sequence of sizing optimizations where after each optimization an update to the selection of buffered gates can be made. Convergence of Algorithm 1 is ensured since each iteration can only remove buffered gates. Each iteration ensures an incremental performance improvement since a buffered gate is replaced with an unbuffered gate only if gate delay can be maintained (or improved). With minor modification, Algorithm 1 is also applicable to the case of power minimization subject to a delay constraint. In particular, the lower-bound optimization (step 2) and each sizing optimization (step 4) now refers to a power minimization optimization. The following subsections detail the steps of Algorithm 1.

### A. The Sizing Optimization Step

Algorithm 1 exploits the realization that once the set of cell library selections for each gate is set, the remaining task of sizing optimization can be efficiently and optimally solved using posynomial programming techniques [12]. A posynomial program consists of an objective function to be minimized and constraints that are all of posynomial form. An objective function  $g(\mathbf{z})$ ,  $\mathbf{z} = [z_1, z_2, \dots, z_m]$  is of posynomial form if it can be expressed as

$$g(\mathbf{z}) = \sum_i b_i (z_1)^{a_{i1}} (z_2)^{a_{i2}} \dots (z_m)^{a_{im}} \quad (20)$$

where  $z_i$  is a positive real design variable (i.e.,  $z_i > 0$ ), the exponents  $a_{ij}$  are arbitrary real numbers and the coefficients  $b_i$  are positive. A constraint is of posynomial form if it can be expressed as  $g(\mathbf{z}) \leq 1$ , where  $g(\mathbf{z})$  is of form (20). Thus, if  $\{\psi^k\}$  is set, Problem 1 reduces to a posynomial programming problem since the objective function and all the constraint equations indicated by (12)–(15) are of posynomial form in the design variables  $\{\mathbf{s}^k\}$  and  $T$  and these variables are continuous.

The resulting sizing optimization problem can be optimally solved since a posynomial program can be shown to be equivalent to a convex programming problem. That is, as a problem of minimizing a convex function over a feasible solution space that forms a convex set [12]. Consequently, for a posynomial program (as for a convex program), any local minimum that is found is guaranteed to also be a global minimum. Such a minimum can be found numerically by using a constrained nonlinear programming technique such as the augmented Lagrangian [13]. In Fig. 5,  $Posyl[go; \{g_c \leq$

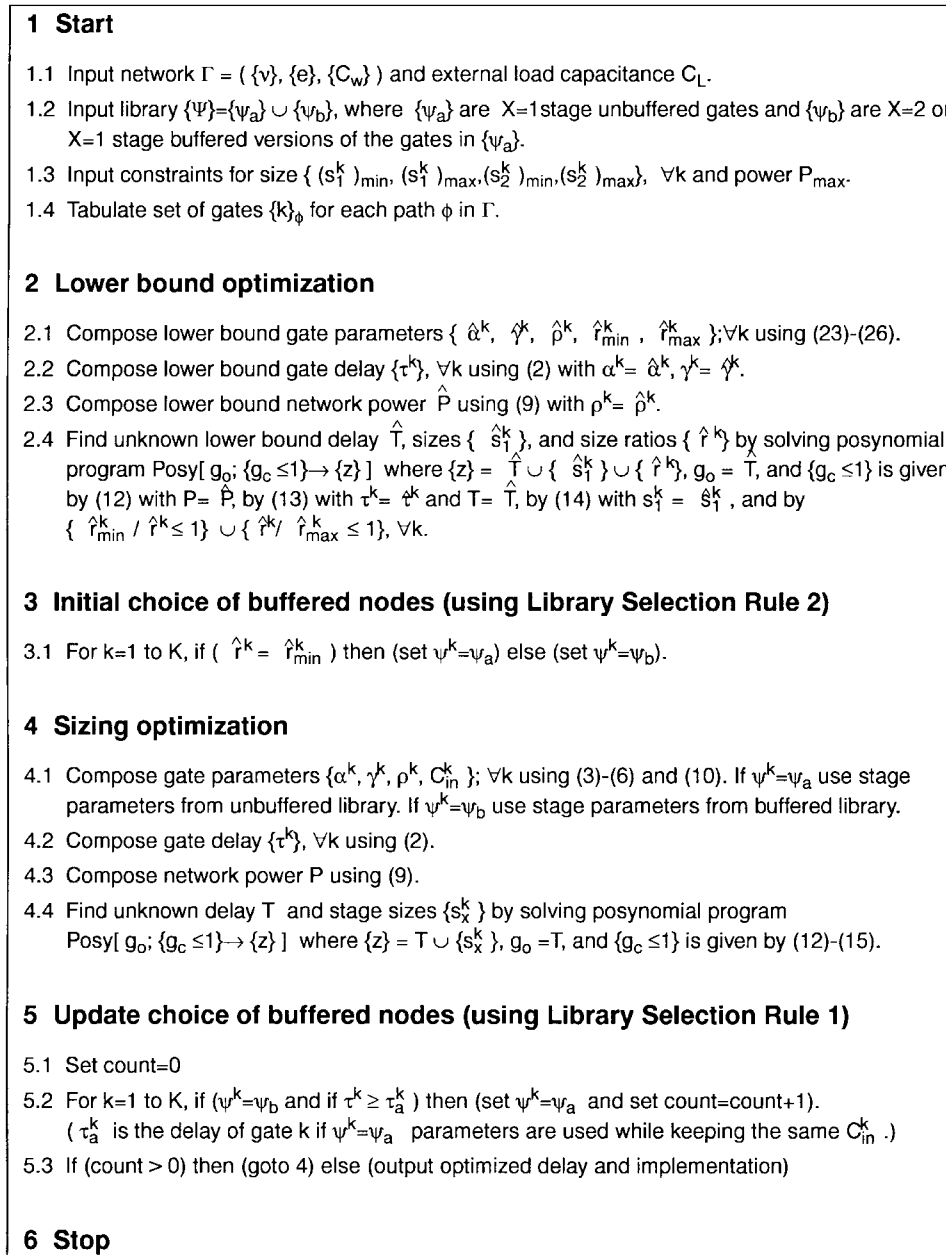


Fig. 5. Algorithm 1 for delay minimization of network  $\Gamma$  under power constraint  $P_{\max}$ .

$1\} \rightarrow \{z\}$  refers to this technique that finds the values for the design variables  $\{z\}$  that minimizes the posynomial objective function  $g_o$  subject to the posynomial constraints  $\{g_c \leq 1\}$ . The solution of a sizing optimization is denoted by  $\{s^{k*}\}$ . The resulting network implementation is written as  $\{\psi^{k*}\}, \{s^{k*}\}$ .

For networks that have a large number of delay paths to be optimized, the reformulation suggested by Marple [11] may be helpful in reducing the complexity of the posynomial program that needs to be solved. In this case, the path constraints of (13) are replaced by

$$\frac{z^i + \tau^k}{z^k} \leq 1, \quad \forall k, \quad \forall i \in \{i\}^k \quad (21)$$

and

$$\frac{z^k}{T} < 1, \quad \forall k \text{ connected to a network output.} \quad (22)$$

The term  $z^k$  ( $z^i$ ) represents the cumulative delay at the output of gate  $k$  (gate  $i$ ) measured from a network input and the term  $\{i\}^k$  represents the set of gates connected to the input of gate  $k$ . The resulting reformulated problem has  $K$  more unknowns to determine (i.e., the  $z^i$ 's) than the original problem but potentially significantly fewer constraints.

### B. The Buffer Insertion Update Step

Unless the set of library selections was optimal, the sizing optimized implementation  $\{\psi^{k*}\}, \{s^{k*}\}$  will not be a (globally) optimal solution to Problem 1. However, based on this implementation an improved set of selections can be systematically deduced. In particular, such an improved set is obtained by changing a buffered node in implementation  $\{\psi^{k*}\}, \{s^{k*}\}$  to an unbuffered node according to Library Selection Rule 1. This

rule is based on the observation that changing from a buffered gate to an unbuffered gate (with same input capacitance) not only results in a lower gate power (i.e., since the dissipation of the buffer circuit is eliminated) but can also result in a lower gate delay provided the fan-out loading is sufficiently low [e.g., see Fig. 3(a)].

**Library Selection Rule 1:** A node  $k$  that is buffered in implementation  $\{\psi^k\}, \{s^{k*}\}$  and has delay  $\tau^k$  is changed to an unbuffered form (i.e., changed from  $\psi_b$  to  $\psi_a$ ) if and only if  $\tau_a^k \leq \tau^k$ . The term  $\tau_a^k$  is the delay that node  $k$  would have if it is changed to an unbuffered form having the same input and output capacitance as in implementation  $\{\psi^k\}, \{s^{k*}\}$ . A node  $k$  that is unbuffered in implementation  $\{\psi^k\}, \{s^{k*}\}$  remains unbuffered.  $\square$

Note that Library Selection Rule 1 involves a simple algebraic calculation and comparison of gate delays for each buffered node and no computation for each unbuffered node. Since all the buffered nodes are tested, several of them may be replaced in one update step. Since application of Library Selection Rule 1 preserves gate input capacitance, the set of replaced nodes does not depend on the order that they were tested. As well, note that the subsequent sized optimized implementations are ensured to have a delay that is incrementally lower than implementation  $\{\psi^k\}, \{s^{k*}\}$ . This is evident since each replaced (and nonreplaced) gate can be initialized to a size that gives the same input capacitance as the corresponding gate in implementation  $\{\psi^k\}, \{s^{k*}\}$ , resulting in an implementation that has lower power and equivalent (or lower) delay than implementation  $\{\psi^k\}, \{s^{k*}\}$ .

### C. The Lower-Bound Optimization Step

A lower bound for network delay consistent with the given power constraint is found by deriving a lower-bound model for gate delay and power and optimizing the network using this model. To guarantee the delay found is a lower bound, the lower-bound model is defined such that: 1) for each choice in the selection space of gate  $k$ , there exists a choice in the lower-bound selection space that is equivalent or superior in all gate characteristics, 2) the lower-bound selection space is continuous (rather than disjoint as previous), and 3) the resulting lower-bound optimization is a posynomial program. To enable the delay found to be a tight lower bound, the lower-bound model is defined such that the lower-bound selection space closely approximates the gate selection space. The lower-bound parameters for gate  $k$  are denoted  $\{\hat{\alpha}^k, \hat{\gamma}^k, \hat{\rho}^k\}$  to distinguish them from the gate parameters  $\{\alpha^k, \gamma^k, \rho^k\}$ .

As shown in Fig. 6, each lower-bound parameter is derived from its respective gate parameter and is a function of the lower-bound size ratio  $\hat{r}^k$  where  $0 < \hat{r}_{\min}^k \leq \hat{r}^k \leq \hat{r}_{\max}^k$ . The value for  $\hat{r}_{\min}^k$  is chosen such that at  $\hat{r}^k = \hat{r}_{\min}^k$ , the loaded delay factor of the corresponding buffered gate equals the loaded delay factor of the unbuffered gate. Thus,  $\hat{r}_{\min}^k = C_1 \gamma_2^k / C_2 \gamma_1^k$  using (5). The value for  $\hat{r}_{\max}^k$  is chosen such that  $\hat{r}_{\max}^k \geq (s_2^k)_{\max} / (s_1^k)_{\min}$ . With  $\hat{r}_{\min}^k$  and  $\hat{r}_{\max}^k$  fixed, the lower-bound relationships as a function of  $\hat{r}^k$  are defined such that at  $\hat{r}^k = \hat{r}_{\min}^k$  they match the characteristics of the unbuffered gate while at  $\hat{r}^k = \hat{r}_{\max}^k$  they match characteristics

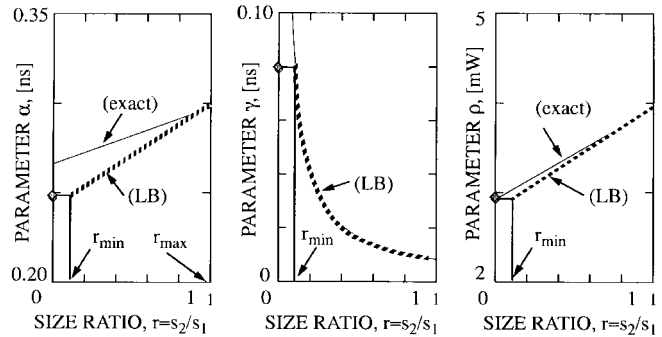


Fig. 6. Comparison of lower-bound parameters (dotted line marked LB) and exact parameters (solid line and diamond point at  $r = 0$ ). LB characteristics equal the exact (unbuffered) gate at  $r = r_{\min}$  and the exact (buffered) gate at  $r = r_{\max}$ .

of the buffered gate. In particular, the relationships for  $\hat{\alpha}^k$  and  $\hat{\rho}^k$  are given by the straight line drawn between the two endpoints while the relationship for  $\hat{\gamma}^k$  simply follows the buffered gate characteristics. Since at  $\hat{r}^k = \hat{r}_{\min}^k$ ,  $\hat{\alpha}^k = \alpha_1^k$  and  $\hat{\rho}^k = \rho_1^k$  while at  $\hat{r}^k = \hat{r}_{\max}^k$ ,  $\hat{\alpha}^k = \alpha_1^k + \alpha_2^k + \hat{r}_{\max}^k \gamma_1^k C_2 / C_1$  and  $\hat{\rho}^k = \rho_1^k + \hat{r}_{\max}^k \rho_2^k$  and since the equation for a straight line defined by two endpoints  $(x_i, y_i)$  and  $(x_j, y_j)$  is  $(y - y_i) / (x - x_i) = (y_j - y_i) / (x_j - x_i)$ , it follows that the lower-bound parameters for gate  $k$  are defined as follows:

$$\hat{\alpha}^k = \left( \frac{\alpha_2^k \hat{r}_{\min}^k + \gamma_2^k \hat{r}_{\max}^k}{\hat{r}_{\max}^k - \hat{r}_{\min}^k} \right) + \left( \frac{\alpha_1^k \hat{r}_{\min}^k + \gamma_1^k \hat{r}_{\max}^k}{[\hat{r}_{\max}^k - \hat{r}_{\min}^k] \hat{r}_{\min}^k} \right) \hat{r}^k \quad (23)$$

$$\hat{\rho}^k = \left( \rho_1^k - \frac{\rho_2^k \hat{r}_{\max}^k \hat{r}_{\min}^k}{\hat{r}_{\max}^k - \hat{r}_{\min}^k} \right) + \left( \frac{\rho_2^k \hat{r}_{\max}^k}{\hat{r}_{\max}^k - \hat{r}_{\min}^k} \right) \hat{r}^k \quad (24)$$

$$\hat{\gamma}^k = \frac{\gamma_1^k \hat{r}_{\min}^k}{\hat{r}^k} \quad (25)$$

where  $\hat{r}_{\min}^k \leq \hat{r}_{\max}^k$  and  $\hat{r}_{\min}^k, \hat{r}_{\max}^k$  are constants such that

$$\hat{r}_{\min}^k = \frac{C_1 \gamma_2^k}{C_2 \gamma_1^k} \quad \text{and} \quad \hat{r}_{\max}^k \geq \frac{(s_2^k)_{\max}}{(s_1^k)_{\min}}. \quad (26)$$

Note that if  $\psi_b$  is a one-stage buffered library, fitted parameters for  $\alpha_2^k, \gamma_2^k$ , and  $\rho_2^k$  (see Section IV) are used in (23)–(26).

In the subsequent lower-bound optimization, (23) and (25) replace (6) and (5), respectively, in gate delay equation (2), while (24) replaces (10) in power equation (9). The input capacitance of a gate is calculated using (3) as previously. The solution of the lower-bound optimization is denoted  $\{\hat{r}^k\}$  where  $\hat{r}^k$  is the size ratio of node  $k$ . The corresponding lower-bound network delay is denoted  $\hat{T}$ .

### D. The Buffer Insertion Initialization Step

An additional feature of the lower-bound solution is that it directly provides a good initial selection of buffered nodes following Library Selection Rule 2.

**Library Selection Rule 2:** Following a lower-bound optimization: if  $\hat{r}^k = \hat{r}_{\min}^k$ , then set  $\psi^k = \psi_a$ ; else set  $\psi^k = \psi_b$  where  $\hat{r}_{\min}^k$  is given by (26). For the case of a single cell buffered gate library with a fixed buffer size ratio  $r_f^k$ , a

gate can be initially unbuffered if  $\hat{r}_{\min}^k \leq \hat{r}^k \leq r_c^k$  where  $\hat{r}_{\min}^k \leq r_c^k \leq r_f^k$ .  $\square$

Library selection rule 2 is based on the observation that the lower-bound selection space of each gate  $k$  is a selection space that contains the unbuffered gate choice (i.e., for  $\hat{r}^k = \hat{r}_{\min}^k$ ) plus a set of lower-bound buffered gate choices (i.e., for  $\hat{r}_{\min}^k < \hat{r}^k \leq \hat{r}_{\max}^k$ ) such that for each choice of buffer gate in the actual gate selection space (e.g., Fig. 4) there is a corresponding choice of lower-bound buffer gate that is equivalent or superior in all gate characteristics. Consequently, if the lower-bound optimization determines a gate should be unbuffered, it is unlikely that by changing to an inferior set of buffers, such as offered by the actual selection space, there will be a benefit to now making this gate buffered. Therefore, in Algorithm 1 the lower-bound optimization step is used to set an upper limit on the number of buffered gates while subsequent steps need only focus on reducing this initial number of buffered gates.

### E. Implementation

Algorithm 1 is implemented in a program called technology optimization program (TOP). TOP consists of: 1) a main program that sets up a sizing iteration by calculating an initial or updated choice of buffered nodes and 2) a subroutine that solves this sizing (or lower-bound sizing) problem and realizes  $Posy[ ]$  in Fig. 5. Under user-defined control is the choice of: 1) power constraint value or list of values, 2) size range constraint values, 3) external load, and 4) including or not including wiring capacitance. The subroutine used in TOP is called automated design synthesis (ADS) and is described by Vanderplaats in [25]. ADS provides a choice of numerical methods for constrained nonlinear programming.

## VI. EXPERIMENTAL RESULTS

Example networks that were optimized using Algorithm 1 and the TOP program are listed in Table II. Shown as well are the number of distinct gates, paths, and primitives in each network as well as the unoptimized network power  $P_u$  and delay  $T_u$  corresponding to the case where all gates are minimum size and unbuffered. Networks *par16a* and *par16b* are 16-b parity generators constructed using three- and two-input XOR gates, respectively. Networks *dec16a* and *dec16b* are 4–16 b decoders constructed using two-input gates. Network *dec16a* is a balanced decoder. Network *dec16b* is a tree style decoder. Networks *add8-d* are 8-b adders constructed using two-input gates. Network *add8a* corresponds to the lookahead carry adder of Fishburn [26]. Network *add8d* is a ripple-style adder while networks *add8b* and *add8c* are intermediate complexity designs. Network *mult8a* is a  $8 \times 8$  b Braun-type parallel multiplier [27]. In this multiplier, only the 162 longest paths and array of 56 SUM-3 and 56 CARRY-3 gates are explicitly considered. Each of the 64 partial product generation gates in the multiplier is taken to be fixed at minimum size and unbuffered.

In optimizing the networks, certain conditions were assumed constant. The load capacitance present on each network output is 100 fF. The transition density for each gate is 0.5

TABLE II  
PROPERTIES OF EXAMPLE NETWORKS FOR OPTIMIZATION

Network Name $\Gamma$	Network Description	No. of Gates K	No. of Paths $\Phi$	No. of Prim.	Unoptimized	
					Power $P_u$ [pJ]	Delay $T_u$ [ns]
par16a	16-bit parity (3-input gates)	9	6	1	19.6	2.26
par16b	16-bit parity (2-input gates)	15	8	1	23.8	1.45
dec16a	4-16 bit decoder (balanced)	32	64	1	44.2	1.32
dec16b	4-16 bit decoder (tree)	36	64	1	50.2	1.56
add8d	8-bit adder (ripple)	37	72	2	56.1	6.25
add8c	8-bit adder (intermediate)	40	74	2	60.7	5.54
add8b	8-bit adder (intermediate)	43	77	2	65.2	4.83
add8a	8-bit adder (lookahead)	46	82	2	69.6	4.13
mult8a	8x8 bit multiplier (Braun)	112	162	2	224.9	11.66

transitions/clock-cycle. The wire capacitance is 10 fF per outdegree. That is, there is 10 fF for each for each node connected to the output of a gate. Each gate connected to a network input has an input capacitance fixed at 50 fF. All optimizations were run on a SUN 4/40 SPARCstation IPC with 20-MB MM. The augmented Lagrangian method was used as the constrained nonlinear programming method where each unconstrained minimization step used the Broyden-Fletcher-Goldfarb-Shanno (BFGS) technique and each line search used the Golden Section method followed by polynomial interpolation [13].

The gate and buffer designs used for network optimizations are shown in Fig. 7. Note that gate cells and buffer cells are independent (i.e., a buffered gate is implemented as a double cell) and that both a CMOS buffer design and a BiCMOS buffer design are included. The gate and buffer designs used are similar to those of Song [28]. Namely, differential CMOS designs are used for the first stage while differential CMOS and BiNMOS [29] designs are used for the second stage. The parameter values shown in Fig. 7 are for the a cell of minimum (i.e., unity) size. The maximum allowed size is five for all gates and buffers. Note that since all designs have negligible static dissipation, only a value for the (internal) dynamic dissipation parameter is listed. All the parameter values in Fig. 7 were deduced by simulation program with integrated circuit emphasis (SPICE) simulation using transistor models obtained from a BiCMOS process that has a 0.8- $\mu\text{m}$  minimum gate length and a 0.8- $\mu\text{m}$  minimum emitter width [30].

For each network, joint gate sizing and buffer insertion optimizations were run to deduce mixed CMOS/BiCMOS (i.e., library type  $\Psi_{1,3}$ ) implementations for three different power constraints. For comparison, gate sizing optimizations were run to deduce optimal all CMOS (i.e.,  $\Psi_1$ ) and all BiCMOS (i.e.,  $\Psi_3$ ) implementations. Optimization results for the various networks are summarized in Table III showing the 1) power constraint setting, 2) CPU time required per iteration, 3) number of iterations (including lower-bound iteration), 4) lower-bound delay  $\hat{T}$  (calculated as described in Section V-C) expressed in relation to the final iteration delay  $T^*$ , 5) first iteration delay  $T_{(1)}$  normalized with respect to  $T^*$ , 6) second iteration delay  $T_2$  normalized with respect to  $T^*$ , 7) final delay  $T^*$  normalized with respect to the unoptimized delay  $T_u$ , 8) fraction of buffered (i.e., BiCMOS) gates in the final design,

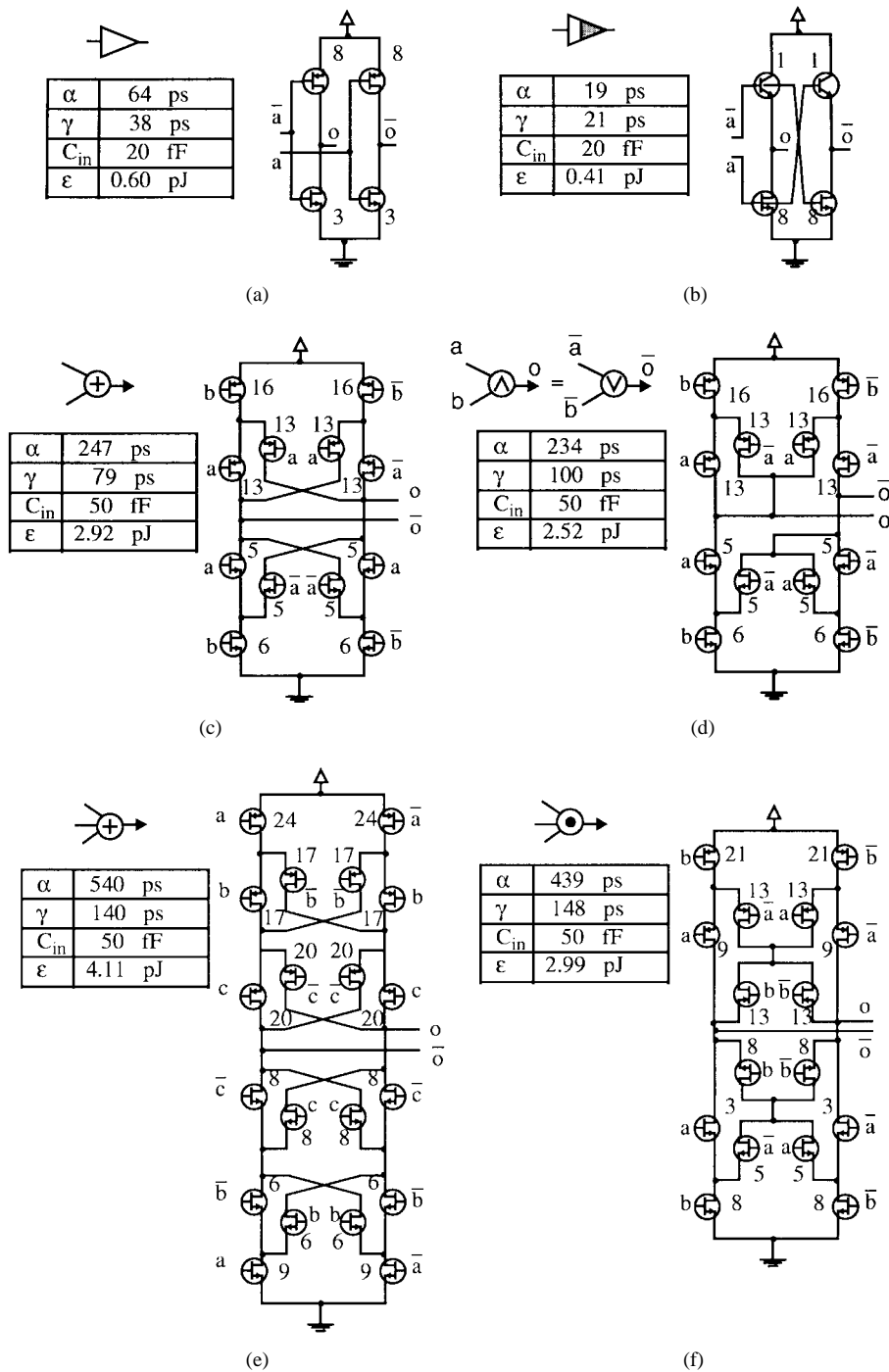


Fig. 7. Example (differential logic) designs for (a) CMOS buffer, (b) BiCMOS buffer (BiNMOS style), (c) XOR-2 gate, (d) AND-2 and OR-2 gate, (e) SUM-3 gate, and (f) CARRY-3 gate. Parameter values shown are for  $V_{dd} = 5$  V.

9) delay advantage (expressed as a speed-up ratio  $\mathfrak{R}$ ) of the final design compared with an optimized CMOS design, and 10) delay advantage of the final design compared with an optimized all BiCMOS design (where a “>” sign means that the given power constraint cannot be met in an all BiCMOS design so the delay advantage listed is conservative and based on the minimum power all BiCMOS design).

Typical optimization iteration results generated by Algorithm 1 are shown in Tables IV and V. Table IV list results for a mixed CMOS/BiCMOS optimization of *add8a* with

a power constraint of  $P_{max} = 1.1P_u$ . As indicated, three optimization iterations were required. The first iteration is the lower-bound optimization that finds a lower-bound delay 3.13 ns. The lower-bound iteration also suggests that 14 gates be buffered. (Note that the lower-bound optimization does select only a fraction 14/46 of the nodes to be buffered.) The second iteration optimizes the network (with these 14 gates as buffered gates) using exact (rather than lower bound) gate parameters. A delay of 3.43 ns is computed and three of the 14 buffered gates are determined to be unnecessary by Library Selection Rule 1.

TABLE III  
JOINT GATE SIZING AND BUFFER INSERTION OPTIMIZATION RESULTS FOR DIFFERENT NETWORKS

Network	Max power $\frac{P_{max}}{P_u}$	Execution		Delay				Fraction Bi-CMOS gates	Speed-up, $\mathfrak{R}$	
		CPU [s] / iter.	Total iter.	LB iter.	1st iter.	2nd iter.	Final		vs. all CMOS S $\Psi_1$	vs. all BiC-MOS $\Psi_3$
				$\frac{T^*}{\hat{T}}$	$\frac{T_{(1)}}{T^*}$	$\frac{T_{(2)}}{T^*}$	$\frac{T^*}{T_u}$			
par16a	1.1	2.3	3	1.056	1.000	1.000	0.920	9 / 9	1.068	1.000
	1.2	2.1	2	1.056	1.000	-	0.916	9 / 9	1.073	1.000
	1.5	2.0	2	1.051	1.000	-	0.912	9 / 9	1.072	1.000
par16b	1.1	3.1	3	1.068	1.071	1.000	0.973	1 / 16	1.007	1.057
	1.2	2.7	4	1.069	1.035	1.014	0.965	2 / 16	1.008	1.036
	1.5	2.6	3	1.061	1.036	1.000	0.959	8 / 16	1.015	1.036
dec16a	1.1	52.1	3	1.104	1.283	1.000	0.803	8 / 32	1.141	>1.113
	1.2	28.6	3	1.075	1.060	1.000	0.758	24 / 32	1.170	1.060
	1.5	20.3	2	1.088	1.000	-	0.750	32 / 32	1.151	1.000
dec16b	1.1	28.7	3	1.112	1.108	1.000	0.819	12 / 36	1.079	>1.058
	1.2	34.5	3	1.081	1.023	1.000	0.853	28 / 36	1.113	1.030
	1.5	27.3	2	1.090	1.000	-	0.853	36 / 36	1.096	1.000
add8d	1.1	69.1	3	1.085	1.000	1.000	0.899	16 / 37	1.060	>1.039
	1.2	52.5	3	1.093	1.007	1.000	0.882	14 / 37	1.056	1.020
	1.5	67.5	5	1.083	1.021	1.006	0.855	14 / 37	1.052	1.022
add8c	1.1	65.4	3	1.090	1.006	1.000	0.877	13 / 40	1.074	>1.056
	1.2	60.2	3	1.082	1.019	1.000	0.854	12 / 40	1.071	1.034
	1.5	56.4	3	1.072	1.026	1.000	0.831	13 / 40	1.065	1.033
add8b	1.1	64.8	3	1.095	1.005	1.000	0.859	12 / 43	1.075	>1.063
	1.2	64.0	3	1.097	1.005	1.000	0.845	13 / 43	1.059	1.022
	1.5	52.6	4	1.072	1.031	1.000	0.805	12 / 43	1.074	1.033
add8a	1.1	69.5	3	1.083	1.012	1.000	0.821	11 / 46	1.088	>1.091
	1.2	60.2	4	1.075	1.018	1.000	0.799	11 / 46	1.082	1.039
	1.5	57.8	4	1.070	1.031	1.000	0.772	13 / 46	1.095	1.034
mult8a	1.1	1067.1	4	1.056	1.003	1.000	0.815	63 / 112	1.128	>1.019
	1.2	634.7	3	1.054	1.000	1.000	0.800	67 / 112	1.121	1.006
	1.5	544.5	4	1.051	1.004	1.000	0.778	67 / 112	1.128	1.004

The third iteration reoptimizes the network with the remaining 11 buffered gates and finds that delay is improved to 3.39 ns. Moreover, since none of the buffered gates were determined to be unnecessary and all buffer sizes are minimum or greater, iteration three is the final iteration. (If one or more of the buffer sizes was below minimum size, an additional iteration would be run with the buffer size constraint of  $s_2^k \geq (s_2^k)_{\min}$  imposed.) The user CPU time required by TOP for the three iterations was 208.5 s (or 69.5 s per iteration). The final iteration *add8a* design is shown in Fig. 8.

Table V lists iteration results for a mixed CMOS/BiCMOS optimization of *mult8a* with a power constraint of  $P_{\max} = 1.1P_u$ . As indicated, four optimization iterations were required. The lower-bound optimization finds a lower-bound delay 9.00 ns. The final design has a delay of 9.50 ns and uses 63 buffered gates. The user CPU time required by TOP for the three iterations was 4268.4 s (or 1067.1 s per iteration). The final iteration *mult8a* design is shown in Fig. 9 along with designs corresponding to other power constraint settings. (Although the reformulation given by (21)–(22) can be used to implicitly enumerate all paths of *mult8a*, the resulting sizing optimization problem has twice the number of unknowns and approximately twice the number of constraints

TABLE IV  
JOINT GATE SIZING AND BUFFER INSERTION OPTIMIZATION  
ITERATION RESULTS FOR NETWORK *add8a* WITH  $P_{\max} = 1.1P_u$

Optimization Result	Iteration #		
	1 (LB)	2	3 (final)
Delay T [ns]	3.13	3.43	3.39
Delay T [units of $T_u$ ]	0.757	0.831	0.821
Power P [units of $P_u$ ]	1.099	1.098	1.100
No. of buffered gates	-	14	11
No. of sized gates	-	6	5
Smallest size of $s_1$	1.00	1.00	1.00
Largest size of $s_1$	1.87	1.61	1.92
Smallest size of $s_2$	-	0.25	1.04
Largest size of $s_2$	-	3.44	3.87
Smallest size ratio r	0.53	0.25	1.04
Largest size ratio r	2.23	2.48	2.35

to consider. Moreover, no improvement in optimized performance is achieved. This is because an upper-bound delay for the ignored paths can be calculated knowing the maximum allowed gate size and shown to be less than the optimized delay found by considering just the 162 longest paths.)

TABLE V  
JOINT GATE SIZING AND BUFFER INSERTION OPTIMIZATION ITERATION  
RESULTS FOR NETWORK *mult8a* WITH  $P_{max} = 1.1P_u$

Optimization Result	Iteration #			
	1 (L.B)	2	3	4 (final)
Delay T [ns]	9.00	9.53	9.50	9.50
Delay T [units of $T_u$ ]	0.772	0.817	0.815	0.815
Power P [units of $P_u$ ]	1.097	1.095	1.095	1.096
No. of buffered gates	-	64	63	63
No. of sized gates	-	9	7	9
Smallest size of $s_1$	1.00	1.00	1.00	1.00
Largest size of $s_1$	2.16	1.72	1.74	1.55
Smallest size of $s_2$	-	0.38	0.43	1.00
Largest size of $s_2$	-	3.48	3.63	3.23
Smallest size ratio r	0.35	0.38	0.43	1.00
Largest size ratio r	2.58	2.13	2.43	2.13

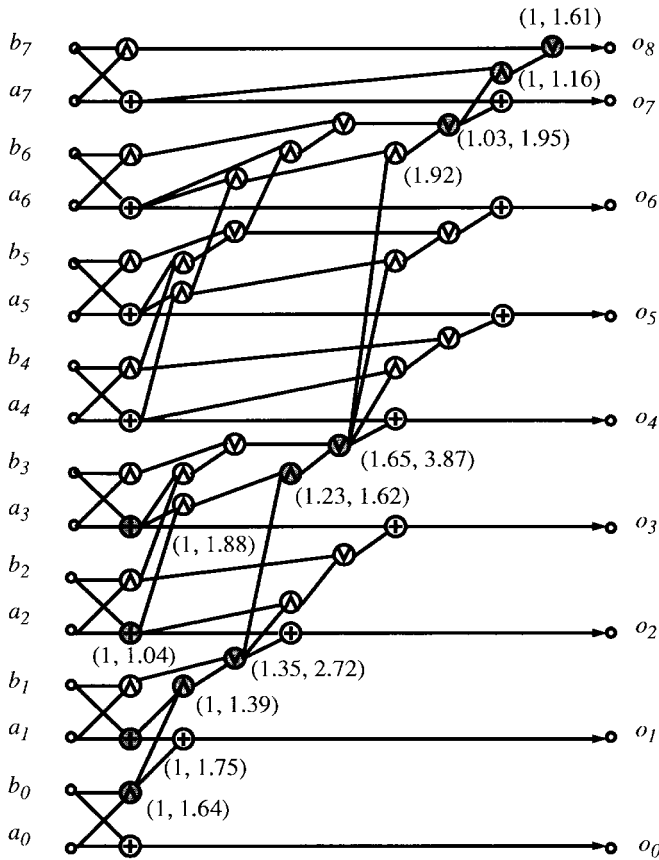


Fig. 8. Final optimized implementation for network *add8a* with  $P_{max} = 1.1P_u$ . BiCMOS (two-stage) gates are shown shaded. Optimized gate sizes are minimum unless indicated in brackets.

Typical relationships between optimized delay and power constraint setting are plotted in Fig. 10. Fig. 10(a) compares the optimized characteristics for network *add8a* for different choices of aggregate cell libraries as defined in Table I. Fig. 10(b) compares the optimized characteristics for network *add8a* for different categories of optimization. Shown also in Fig. 10(b) is a comparison of optimization results obtained using alternative approaches that contrast the nonlinear programming approach of Algorithm 1. Point G (solid square) is for an alternative gate sizing approach similar to the repower

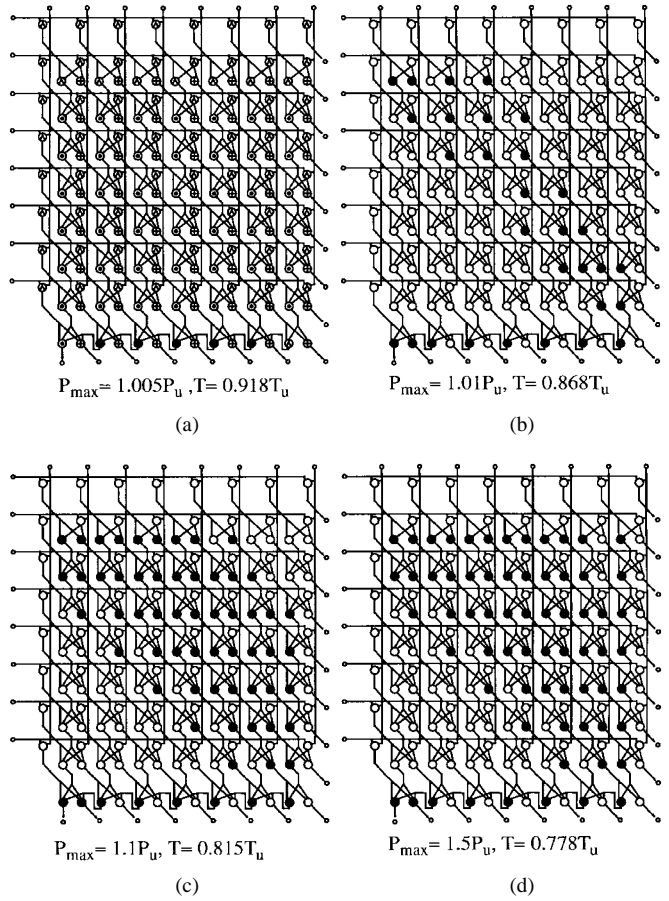
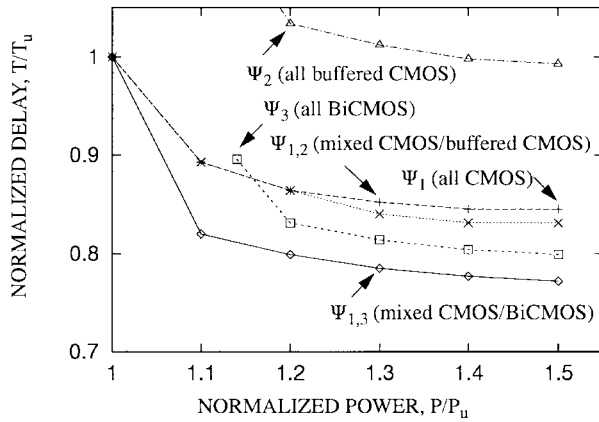


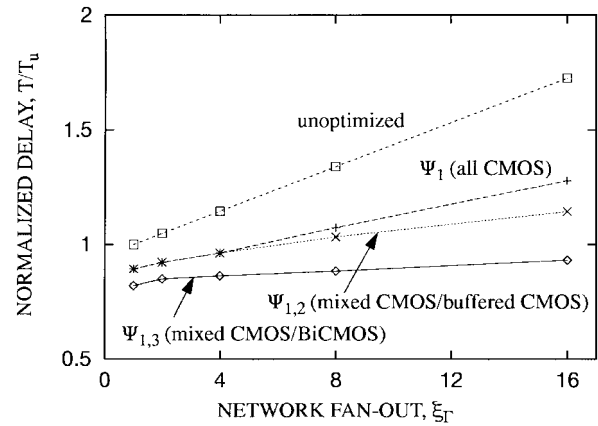
Fig. 9. Final optimized implementation and delay  $T$  for network *mult8a* at various settings of power constraint  $P_{max}$ . BiCMOS (2-stage) gates are shown shaded. All other gates are CMOS (one-stage).

method of Singh [31] but modified to handle continuous case sizing. In this optimization, the target network delay was set to the minimum value found by Algorithm 1 and a critical network defined based on analysis of node timing slacks. Nodes for resizing were identified from a min-cutset of the critical network after assigning node weights based on fan-out. Point J (solid diamond) is for an alternative joint gate sizing and buffer insertion approach. This approach is similar to the alternative gate sizing approach except that nodes can now be buffered as well as sized to improve their delay. Point T (solid circle) is for the full optimization method of Singh [31] that allows both gate sizing and critical path resynthesis through buffer tree transformations. The points G, J, and T appear consistent with the locus of values generated by Algorithm 1. A direct comparison with Algorithm 1 is, however, limited by the fact that the alternative approaches do not presently handle the case of constrained optimization.

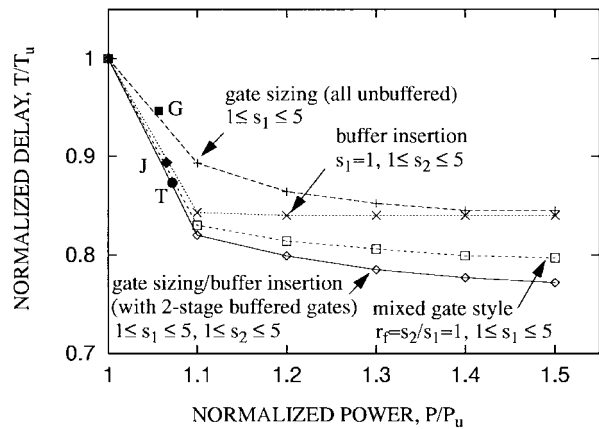
Finally, typical relationships between optimized delay and external load (i.e., network fan-out) are plotted in Fig. 11. (A fan-out greater than unity occurs, for example, if the network has broadcasted outputs that provide the inputs to a parallel multiplier or other array structure.) Fig. 11(a) shows optimized delay versus external load for network *add8a* for different choices of aggregate cell libraries. Fig. 11(b) shows the performance advantage of BiCMOS over CMOS versus



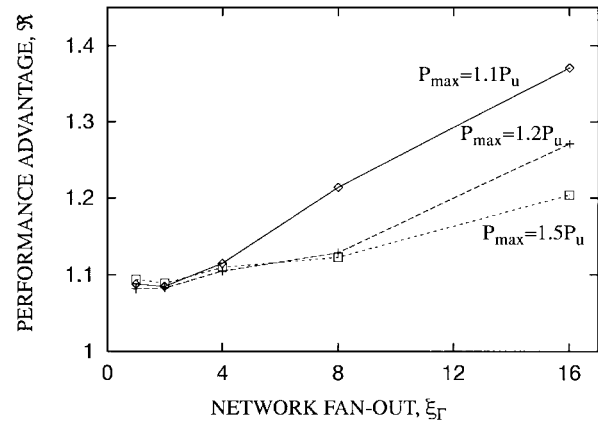
(a)



(a)



(b)



(b)

Fig. 10. Optimization results for network *addsa* with (a) different aggregate libraries and (b) different problems categories. Points G, J, and T were obtained using alternative approaches to Algorithm 1.

external load for network *addsa* for different choices of power constraint.

## VII. DISCUSSION OF RESULTS

The results of Section VI confirm the utility of the proposed joint gate sizing and buffer insertion algorithm to designers of CMOS and BiCMOS networks. From Fig. 10 and from examining the speed-up values  $\mathcal{R}$  in Table III, it is evident that joint optimization yields an advantage over just gate sizing or just buffer insertion optimization. In particular, it is seen that for a BiCMOS technology, a mixed CMOS/BiCMOS approach achieves a delay advantage (at equivalent power) versus any other CMOS approach (i.e.,  $\Psi_1$ ,  $\Psi_2$ , or  $\Psi_{1,2}$  in Table I) or the all BiCMOS approach (i.e.,  $\Psi_3$  in Table I). In large fan-out networks, the delay advantage increases proportionally with external load on the network at a rate dependent on the power constraint (Fig. 11). The fact that the performance advantage and optimized design (e.g., Fig. 9) depends strongly on the power constraint provides justification for seeking a method that is capable of constrained optimization. Note that it is possible to include area constraints into Algorithm 1 provided a posynomial area model is used such as identified by Hoppe [20].

The results of Table III also demonstrate that Algorithm 1 can find both a tight lower bound and a near-optimal final

Fig. 11. Variation with network fan-out of (a) optimized delay and (b) performance advantage  $\mathcal{R}$  versus all-CMOS (network = *addsa*).

design. From the ratios of final to lower-bound delay  $T^*/\hat{T}$ , it is seen that in all cases  $T^*/\hat{T} \leq 1.11$ . Moreover, from Fig. 10(b), it is seen that Algorithm 1 is superior to each of the alternative methods examined in determining minimum delay designs and may reflect that fact that Algorithm 1 considers sufficient global consequences of each gate choice that it is less prone to converge on a local minimum. As well, Algorithm 1 appears to have a rapid rate of convergence in terms of the number of iterations needed. From the intermediate iteration delay ratios  $T_{(i)}/T^*$  in Table III, it is seen that in all cases  $T_{(2)}/T \leq 1.014$ . Hence, just two iterations are sufficient to achieve a delay that is at worst within 1.4% of its final optimized value in and at worst within 11% of the lower-bound value.

At present the CPU requirement of Algorithm 1 permits networks of a few hundred gates,  $O(K = 10^2)$ , to be optimized. The computational complexity of an augmented Lagrangian algorithm that uses the BFGS method for unconstrained minimization is approximately  $O(K^3 c_{\max})$  flops where  $K$  is the number of unknowns (i.e., number of gates in the network) and  $c_{\max}$  is the number of constraints (i.e., number of delay paths in the network). This assumes that the Lagrange multipliers are updated  $O(c_{\max})$  times, each BFGS minimization needs  $O(K)$  line searches, and a line search direction calculation requires  $O(K^2)$  flops [34]. The mem-

ory required by such an augmented Lagrangian algorithm is approximately  $O(K^2 + Kc_{\max})$  since the Hessian and gradient of the augmented Lagrangian function must be accessible.

A more efficient gate sizing approach can be easily incorporated into Algorithm 1 and will enable larger networks to be optimized (with possibly some sacrifice in accuracy). For example, the method of feasible directions algorithm [32] for constrained nonlinear programming can be selected in TOP instead of the augmented Lagrangian algorithm. (Doing this for *add8a* with  $P_{\max} = 1.02P_u$  resulted in a CPU time reduction of  $1.87\times$ , but the optimized delay was 1.05 times higher.) Future incorporation of the convex programming-based sizing approach of Sapatnekar [14] or the linear programming-based sizing approach of Berkelaar [33] may also yield efficiency improvements. Their work suggests that continuous case sizing optimization of networks containing  $O(K = 10^3)$  gates should be feasible assuming the same SPARCstation used in this work.

The absolute accuracy of the network delay and power optimization values reported in Section VI are limited by two simplifying assumptions that were made but that are not an inherent limitation of Algorithm 1. First, the gate parameter values used (Fig. 7) were for the slowest input of each gate. Second, the transition density value used was the same for all gates in a network. Assigning unique parameters values for each gate input will improve the delay accuracy of Algorithm 1 without increasing CPU time. Precalculating [21] and assigning a distinct transition density to each node will improve dynamic power accuracy without increasing CPU time.

## VIII. CONCLUSION

In conclusion, this paper has presented the first reported joint gate sizing and buffer insertion method for minimizing the delay of power constrained combinational logic networks that can incorporate a mixture of unbuffered and buffered gates (or mixture of CMOS and BiCMOS gates). The method is versatile and solves mixed CMOS/BiCMOS optimization problems or any other optimization problem where the network uses a mixture of compatible logic families and continuous sizing. Experimental results have shown that the approach of the algorithm (where logic family choices are decided by an iterative process that uses a sequence of sizing optimizations) results in a rapid convergence to a near-optimal design. As well, this approach enables the algorithm to be implemented from an existing gate sizing capability and to benefit from future developments in gate sizing. Experimental results have also highlighted the need for the proposed algorithm by demonstrating that: 1) a mixed logic family design (e.g., CMOS/BiCMOS design) can achieve a significant performance advantage over a corresponding single logic family design (e.g., CMOS only design), 2) the advantage is strongly dependent on the given application and must be systematically calculated, and 3) a simpler algorithm (not using iteration) yields inferior designs. The proposed algorithm, thus, represents an effective tool for applying and designing logic circuits implemented in BiCMOS or other mixed technologies.

## ACKNOWLEDGMENT

The authors would like to thank R. Malm of IBM as well the anonymous referees for their helpful comments on this work.

## REFERENCES

- [1] D. Chen and C. Zukowski, "CMOS optimization including logic family mixing," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1991, pp. 2240–2243.
- [2] D. Chen, C. Zukowski, and M. Banu, "Macromodeling BiCMOS gates for circuit optimization," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1993, pp. 8.1.1–8.1.4.
- [3] P. Duchene and M. Declercq, "Strategies for CMOS/BiCMOS gate usage on sea-of gates arrays," in *Proc. IEEE 1991 Custom Integrated Circuits Conf.*, 1991, pp. 14.2.1–14.2.4.
- [4] L. Wissel and E. Gould, "Optimal usage of CMOS within a BiCMOS technology," *IEEE J. Solid-State Circuits*, vol. 27, pp. 300–306, Mar. 1992.
- [5] A. Ruehli, P. Wolff, and G. Goertzel, "Power and timing optimization of large digital systems," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1976, pp. 402–405.
- [6] ———, "Analytical power/timing optimization technique for digital system," in *Proc. IEEE Design Automation Conf.*, 1977, pp. 142–146.
- [7] K. Hedlund, "Models and algorithms for transistor sizing in MOS circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1984, pp. 12–14.
- [8] L. Glasser and L. Hoyte, "Delay and power optimization in VLSI circuits," in *Proc. IEEE Design Automation Conf.*, 1984, pp. 529–535.
- [9] J. Fishburn and A. Dunlop, "TILOS: a posynomial programming approach to transistor sizing," in *Proc. IEEE Int. Conf. Computer Aided Design*, 1985, pp. 326–328.
- [10] M. Matson and L. Glasser, "Macromodeling and optimization of digital MOS VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 659–678, Apr. 1986.
- [11] D. Marple, Performance optimization of digital VLSI circuits, Computer Systems Laboratory, Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-86-308, Oct. 1986.
- [12] J. Ecker, "Geometric programming: methods, computations and applications," *SIAM Rev.*, vol. 22, no. 3, pp. 338–362, July 1980.
- [13] G. Vanderplaats, *Numerical Optimization Techniques for Engineering Design with Applications*. New York: McGraw Hill, 1984.
- [14] S. Sapatnekar, V. Rao, R. Vaidya, and S. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1621–1634, Nov. 1993.
- [15] T. Schreyer, "The effects of interconnect parasitics on VLSI circuit performance," Integrated Circuits Laboratory, Stanford Univ., Stanford, CA, Tech. Rep. G815-3, Mar. 1989.
- [16] F. Omerod, D. Schucker, K. Deierling, and N. Salamina, "A mixed technology gate array with ECL and BiCMOS logic on a single chip," in *Proc. Symp. VLSI Circuits*, 1987, pp. 31–32.
- [17] S. Devadas, K. Keutzer, and S. Makik, "Delay computation in combinational logic circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 176–179.
- [18] H. Chen, D. Du, and L. Liu, "Critical path selection for performance optimization," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 185–195, Feb. 1993.
- [19] N. Hedenstierna and K. Jeppson, "CMOS circuit speed and buffer optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 270–281, Mar. 1987.
- [20] B. Hoppe, G. Neuendorf, D. Schmitt-Landsiedel, and W. Specks, "Optimization of high-speed CMOS logic circuits with analytical models signal delay, chip area, and dynamic power dissipation," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 236–247, Mar. 1990.
- [21] L. Gal, "Reply to comments on the optimum tapered buffer problem," *IEEE J. Solid-State Circuits*, vol. 29, pp. 158–159, Feb. 1994.
- [22] F. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 310–323, Feb. 1993.
- [23] H. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 468–473, Aug. 1984.
- [24] W. Li, A. Lim, P. Agrawal, and S. Sahni, "On the circuit implementation problem," in *Proc. 29th Design Automation Conf.*, 1992, pp. 478–483.
- [25] G. Vanderplaats and H. Sugimoto, "A general-purpose optimization program for engineering design," *Comput. & Structures*, vol. 24, no. 1, pp. 13–21, 1986.

- [26] J. Fishburn, "A depth-decreasing heuristic for combinational logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between," in *Proc. 27th Design Automation Conf.*, 1990, pp. 361–364.
- [27] E. Braun, *Digital Computer Design*. New York: Academic, 1963.
- [28] P. Song and G. De Micheli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1184–1198, Sept. 1991.
- [29] A. Gamal, J. Kouloheris, D. How, and M. Morf, "BiNMOS: a basic cell for BiCMOS sea-of-gates," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1989, pp. 8.3.1–8.3.4.
- [30] R. Hadaway, P. Kempf, P. Schvan, M. Rowlandson, V. Ho, J. Kolk, B. Tait, D. Sutherland, G. Jolly, and I. Emesh, "A sub-micron BiCMOS technology for telecommunications," in *Microelectron. Eng.*, Elsevier, vol. 15, pp. 513–516, 1991.
- [31] K. Singh and A. Sangiovanni-Vincentelli, "A Heuristic algorithm for the fanout problem," in *Proc. 27th Design Automation Conf.*, 1990, pp. 357–360.
- [32] G. Zoutendijk, *Method of Feasible Directions*. Amsterdam, the Netherlands: Elsevier, 1960.
- [33] M. Berkelaar, P. Buurman, and J. Jess, "Computing the entire active area/power consumption versus delay tradeoff curve for gate sizing with a piecewise linear simulator," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1424–1434, Nov. 1996.
- [34] R. Fletcher, "An overview of unconstrained optimization," in *Proc. Algorithms for Continuous Optimization*, 1994, pp. 109–143.



**Kerry S. Lowe** (S'82–M'83) was born in Vancouver, BC, Canada. He received the B.A.Sc. (Honors) degree in engineering physics in 1981 and the M.A.Sc. degree in electrical engineering in 1983, both from the University of British Columbia, Canada, and the Ph.D. degree in electrical engineering in 1995 from the University of Toronto, Canada.

From 1983 to 1990 and from 1994 to present he has been with Nortel in Ottawa, Ont, Canada. His areas of interest include the design and optimization of high speed digital integrated circuits for telecommunications transmission, switching, and signal processing applications.

Dr. Lowe is a licenced member of the Professional Engineers of Ontario.



**P. Glenn Gulak** (S'82–M'83–SM'96) received the Ph.D. degree in electrical engineering from the University of Manitoba, Canada, in 1984, while holding a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship.

From 1985 to 1988 he was a Research Associate in the Information Systems Laboratory and the Computer Systems Laboratory at Stanford University, Stanford, CA. He is now a Professor in the Department of Electrical and Computer Engineering at the University of Toronto, Canada. His research

interests are in the area of digital communications, digital microelectronics, and computer architecture.

Dr. Gulak has received several teaching awards for undergraduate courses taught in both the Department of Computer Science and the Department of Electrical and Computer Engineering at the University of Toronto. He is a registered Professional Engineer in the Province of Ontario, Canada. He has served on the ISSCC Signal Processing Technical Subcommittee since 1990 and currently serves as Program Secretary for ISSCC.