iCOACH: A circuit optimization aid for CMOS high-performance circuits

H.Y. Chen

Texas Instruments, Inc., 12840 Hillcrest Dr., Dallas, TX 75230, USA

S.M. Kang

Dept. of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1101 W. Springfield Ave., Urbana, IL 61801, USA

Received 12 June 1990

Abstract. iCOACH is a two-pass iterative circuit optimizer which generates a polycell-based layout from a gate level description file and user-defined timing constraints. The first pass is to generate, place and route the cells and extract the interconnection parameters. The second pass optimizes the circuit at the transistor level and makes necessary layout adjustments including pitch-matchings. Although iCOACH has the layout style similar to the polycell approach, it is distinct in two important aspects. First, iCOACH does not rely on any fixed cell library. Instead iCOACH generated customized cells by invoking the circuit optimizer and performs the transistor-level optimization for both static and dynamic CMOS circuits and their layouts under realistic constraints. Secondly, although the cells in the same row are required to have the same height, different rows can have different heights to make circuit more compact. Dynamic circuits are used with a careful treatment on reliability issues related to charge sharing and noise margin, which has not been treated rigorously in the previous literature. An area-efficient polycell layout style is also introduced for dynamic CMOS circuits. A 4-bit ALU and a 32-bit adder examples are presented to demonstrate the capability of iCOACH.

Keywords. Circuit optimization, transistor sizing, dynamic CMOS, delay models, folding layout.

Elsevier
INTEGRATION, the VLSI journal 10 (1991) 185–212

1. Introduction

As the functionality and complexity of integrated circuits increase, sophisticated CAD tools become indispensable in reducing the considerable design time and cost. Many design automation tools like MINI [1] and ESPRESSO [2] have been used successfully for the two-level logic minimization appropriate for PLA-based implementation. Other tools like MIS [3], LSS [4], and SOCRATES [5] have been introduced for multilevel logic minimization. The minimized logic symbols can then be mapped to target technology through a variety of gate arrays or standard cells. Standard cell libraries have been used in the synthesis process thereby freeing the synthesis system from the details of cell layout. However, in the standard-cell based system, the size of the cell library have often caused database management problems. Also, the useful life of a particular library is relatively short, as dictated by the lifetime of the technology employed [6]. The prevalent use of complex gates like AOI and OAI further complicates the library issue. As mentioned in [7], when an individual gate is constrained to have at most

Hau-Yung Chen (S'85-M'88) received the B.S. degree from the National Chiao-Tung University in 1978, the M.S. degree from Southern Methodist University in 1982, and the Ph.D degree from the University of Illinois at Urbana-Champaign in 1988 all in electrical engineering.

From 1980 to 1981, he was a testing engineer at ERSO, Taiwan. From 1982 to 1984, he was an analog circuit designer at GE Intersil, Cupertino, CA. From 1985 to 1988, he was a research assistant in the Coordinated Science Laboratory at the University of Illinois. He joined Texas Instruments, Dallas TX in 1988 as a member of technical staff and is working on the next generation CAD systems. He received the best paper award from ICCD in 1987. His research interests include circuit modeling, VLSI CAD systems and VLSI architecture.

Sung-Mo (Steve) Kang (S'73-M'75-SM'80-F'90) received Ph.D degree in electrical engineering from the University of California at Berkey in 1975. Until 1985 he was with AT&T Bell Laboratories at Holmdel, Murray Hill, and also served as a faculty member of Rutgers University. At AT&T Bell Laboratories, he worked on large-scale telecommunications network planning, design and manufacturing of WE32000-series VLSI microprocessor chips and peripheral chips. In 1985, he joined the University of Illinois at Urbana-Champaign where he is Professor of Electrical and Computer Engineering and Research Professor of Coordinated Science Laboratory, and Associate Director of NSF Engineering Research Center for Compound Semiconductor Microelectronics. He has served on the editorial boards of IEEE Transactions on Circuits and Systems, IEEE Circuits and Devices Magazine, IEEE Design & Test of Computers, International Journal of Circuit Theory and Applications, and Circuits, Systems and Signal Processing Journal. He was special gest ditor for three special issues of IEEE Design & Test of Computers and International Journal of Circuit Theory and Applications. He also has served on the program committees of ICCAD, ICCD, ISCAS, MWSCAS and international workshops. His research interest includes modeling of semiconductor devices, VLSI design methodologies and optimization techniques for performance, reliability and manufacturability, and optoelectronic circuits and systems. He has been a consultant for semiconductor industry and delivered lectures at several international workshops on VLSI design and CAD. He has received best paper awards from ICCD (1987) and MWSCAS (1979). He has served as Administrative Vice President, Secretary and Treasurer, AdCom member and is the 1990 President-Elect of IEEE Circuits and Systems Society.

s transistors from its output to ground and p transistors from its output to power supply, as many as 3,503 different complex gates can be configured even for $(s, p) = (4, 4)$ gates. This number dramatically increases to 425,803 for $(s, p) = (5, 5)$ and 154,793,519 for $(s, p) = (6, 6)$. Moreover, most standard-cell libraries are not well suited for changing parameters to meet the special performance requirement, although the *flexible cell* concept has been known for some time. Therefore, most standard cell libraries would contain multiple versions of some cells with different driving powers.

These major deficiencies can be overcome by an adaptive cell generator approach. Instead of using the predefined cells, the cell generator dynamically generates cells for each job *on the fly* according to their circumstantial situation such as fan-in, fan-out, and input signal slew rate. Advantages of this approach include the ability to take logic circuits directly, without molding them into predefined patterns or manually adding custom cells. Furthermore, this approach can efficiently handle individual sizing of transistors, ranging from the minimum size to the maximum size.

Although the adaptive cell generator is more flexible in accommodating different sizes of transistors, it is not a simple task to determine the appropriate transistor sizes from the logic diagram and gate symbols. A sheer increase in transistor size does not always guarantee the decrease in delay time. Also, any dramatic increase in area is not justified when the speed improvement is only marginal. Moreover, important circuit reliability issues which have been neglected in the past need to be addressed.

This paper presents a rigorous system for optimizing the chip speed/area ratio. Section 2 presents an overall view of the iCOACH system. It should be noted that individual system modules such as the placement and routing tool, the critical path finder and others need not be hard bound and can be replaced with custom tools. Section 3 discusses the problem formulation and Section 4 discusses the delay–area tradeoff from the sensitivity point of view. Section 5 and 6 deal with the critical path identification and the timing allocation scheme. Sections 7 through 9 present our optimization method using circuit examples. Section 10 presents a unique folding layout style for dynamic CMOS complex gates. A 4-bit ALU and a 32-bit adder example are shown in Section 11 to demonstrate the capability of the iCOACH system. Finally a conclusion is drawn in Section 12.

2. The system structure

Figure 1 shows the structure of iCOACH. Given a functional description in the form of gate level representation and the user-defined timing constraint, iCOACH synthesizes a circuit which meets the timing constraints with a minimum active area. The specified gate representation is assumed already optimized by using logic minimization tools. A gate level description file for the 4-bit block carry look ahead (BCLA) adder is shown in Fig. 2.

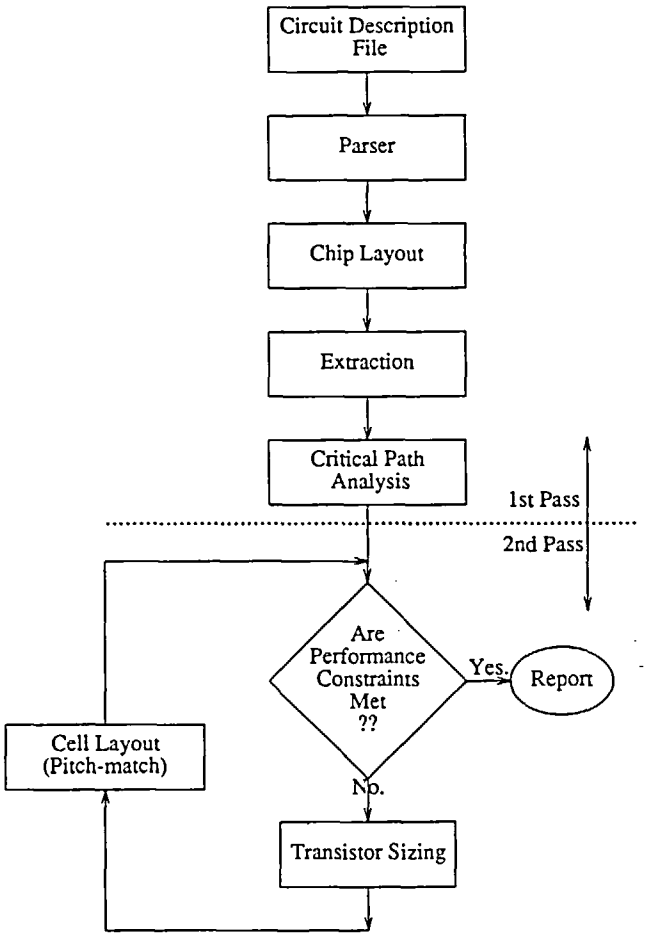

Fig. 1. System overview.

Upon reading the circuit description file, the circuits and leaf cells are generated with default/minimum transistor sizes [1]. The placement tool Timber Wolf [9] and the routing tool YACR2 [10] are next executed to generate the

```
G .AO4432 G3 G2 G1 G0 P1 G3 G2 G1 P2 G3 G2 G3 P3
P .OR P3 P2 P1 P0
C(N+Z)B .AO4432 G2 G1 G0 CNB P0 G2 G1 G0 P1 G2 G1 G2 P2
C(N+Y)B .AO332 G1 G0 CNB P0.G1 G0 G1 P1
C(N+X)B .AO22 G0 CNB G0 P0
CNB .INV CN
C(N+Z) .INV C(N+Z)B
C(N+Y) .INV C(N+Y)B
C(N+X) .INV C(N+X)B
```

Fig. 2. Input file for block carry look ahead (BLCA) circuit.

[1] For instance, 4 μm for n-channel transistors and 8 μm for p-channel transistors in the 3 μm technology case.

layout [2]. The reason to have the physical design at this early stage is to extract the interconnect parasitics, in particular capacitances, for electrical optimization. The whole circuit is then analyzed with PERT technique [11] for critical path identification. The circuit optimizer is next invoked to perform electrical optimization starting from the cells in the critical path and followed by backtracking to other paths. After the electrical optimization, cell heights in each row are matched to the highest cell in that particular row. However, different rows may have different heights. Note that the relative cell locations as well as intra-cell connections are not altered during the electrical optimization. Therefore, the placement and the routing are only performed once before the electrical optimization.

3. Formulation of constrained circuit optimization problem

An often-discussed performance measure of a technology is the product of power dissipation and propagation delay of a typical gate. This product is regarded important because it represents the dissipated energy per switching operation [12–14]. However, once a technology is chosen, it becomes a major design goal to maximize on-chip functional events per unit time [8]. Since the throughput is proportional to both the chip speed and the packing density, which are inversely proportional to the delay and the area, we set the goal as to minimize the delay–area product with a realistic set of physical constraints on noise margins, charge sharing and minimum dimensions allowed by the technology. The circuit optimization problem can then be restated as follows:

$$\text{minimize}(t_d(W) \times A(W)) \tag{1}$$

subject to

$$W_{\min} \leqslant W \leqslant W_{\max},$$

$$t_d(W) \leqslant T,$$

$$\beta_{\mathrm{R(min)}} \leqslant \beta_{\mathrm{R}} \leqslant \beta_{\mathrm{R(max)}}.$$

where $W = (w_1, w_2, \ldots, w_n)$ is the transistor size vector and $W_{\min}$ is the minimum channel width imposed by the target technology, $t_d(W)$ is the critical path delay time which should be less than the timing constraint T. $A(W)$ is the total chip area and the beta ratio $\beta_{\mathrm{R}} = \beta_{\mathrm{n}}/\beta_{\mathrm{p}}$, defines a proper operation range for reliability issues such as noise margin and charge sharing, where $\beta_{\mathrm{n,p}}$ is the n- or p-device transconductance parameter, respectively. Although the transistor size variables include the channel length L, channel width W and drain/source length, all these variables other than transistor widths are dictated by the process technology, and therefore we concentrate on the user-specifiable variables, i.e., the channel width vector W.

[2] Other placement and routing tools can be adopted in lieu of TimberWolf and YACR2.

3.1. Noise margin constraint

Proper noise margins are important for both static and dynamic CMOS circuits. Normally, noise margins are set to be at least 25% of V_{DD} [15]. In general, noise margins are different in high and low logic states; they are $NM_H = V_{OH} - V_{IH}$ and $NM_L = V_{IL} - V_{OL}$, where subscripts O and I denote output and input signals, whereas H and L denote high and low logic signal levels. To ensure proper noise margins, the beta ratio of the inverter buffer should be adjusted within the upper and lower bounds. The upper bound, $\beta_{R(max)}$ is calculated by the following two equations [16,17].

$$V_{IL} = \frac{2V_{out} - V_{DD} + V_{Tp} + \beta_{R(max)} V_{Tn}}{1 + \beta_{R(max)}} \tag{2}$$

$$\beta_{R(max)}(V_{IL} - V_{Tn})^2 = \left[2(V_{IL} - V_{DD} - V_{Tp})(V_{out} - V_{DD}) - (V_{out} - V_{DD})^2\right] \tag{3}$$

Similarly, the lower bound, $\beta_{R(min)}$ can be also obtained from the following two equations

$$V_{IH} = \frac{\beta_{R(min)}(2V_{out} + V_{Tn}) + V_{DD} + V_{Tp}}{1 + \beta_{R(min)}}, \tag{4}$$

$$\beta_{R(min)}\left[2(V_{IH} - V_{Tn})V_{out} - V_{out}^2\right] = (V_{IH} - V_{DD} - V_{Tp})^2, \tag{5}$$

where V_{Tp}, V_{Tn} are the threshold voltages of p- and n- channel devices of the inverter buffer, respectively. With the requirement of $0.25V_{DD}$ for noise margin, β_R is bounded by 0.1 and 10 for $V_{Tn} = |V_{Tp}| = 1$, and $V_{DD} = 5$.

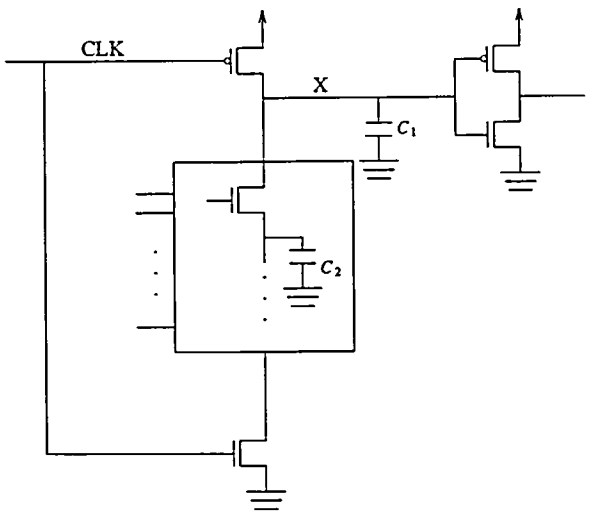

Fig. 3. Charge sharing phenomenon.

3.2. Charge sharing constraint

The other significant problem in dynamic circuits is the charge sharing problem. This phenomenon is elaborated in Fig. 3.

The capacitance C_1 is charged up to the power supply voltage V_{DD} and stores a total charge of $V_{DD} \times C_1$ during the precharge phase. During the evaluation phase, a path to C_2 may be on while the circuit output is still evaluated low. The charge originally stored in C_1 can be coupled to C_2. When the charge equilibrium is reached, the voltage at node X becomes

$$V_X = \frac{V_{DD} \times C_1}{C_1 + C_2}. \tag{6}$$

This charge redistribution may cause the voltage at node X to drop below the *logic threshold voltage* and erroneously flip the output state to high when it should be low. The *logic threshold voltage* V_{inv} can be expressed as [17]

$$V_{inv} = \frac{V_{DD} + V_{Tp} + V_{Tn}\sqrt{\beta_R}}{1 + \sqrt{\beta_R}} \tag{7}$$

V_{inv} is the *pivot voltage* for the output to flip the state. For the circuit to operate properly after charge redistribution, the voltage V_X must maintain its level at

$$V_X > V_{inv} = \frac{V_{DD} + V_{Tp} + V_{Tn}\sqrt{\beta_R}}{1 + \sqrt{\beta_R}} \tag{8}$$

Therefore, for the circuit not to suffer from the charge sharing problem, the following condition should hold:

$$\sqrt{\beta_R} > \frac{V_{DD} + V_{Tp} - V_X}{V_X - V_{Tn}} \tag{9}$$

β_R as a function of V_X for $V_{Tn} = |V_{Tp}| = 1$ is depicted in Fig. 4.

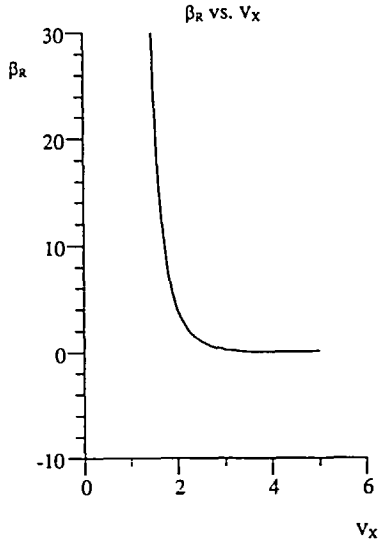

Fig. 4. β_R vs. internal voltage.

As the node voltage V_X at node X gets smaller, β_R has to be increased dramatically to prevent the false state. In the extreme case, V_X may drop to the threshold voltage V_{Tn} and the output will inevitably flip state erroneously no matter how big β_R is made.

4. Critical path identification and timing allocation

4.1. Critical path identification

A topological representation of a 4-bit ALU circuit is shown in Fig. 5 with nodes representing gates and arcs representing connections between gates. The node number indicates the topological number of the gate. The critical paths of the circuit initially with default transistor sizes can be identified with PERT(Performance Evaluation and Review Technique) [11] and are shown in bold lines.
Critical Path I: $9 \rightarrow 11 \rightarrow 28 \rightarrow 29 \rightarrow 30$
Critical Path II: $13 \rightarrow 16 \rightarrow 28 \rightarrow 29 \rightarrow 30$

One of the critical paths, which has the longest delay time, is picked up as the *primary critical path*. If Path I is chosen to be the primary critical path and is

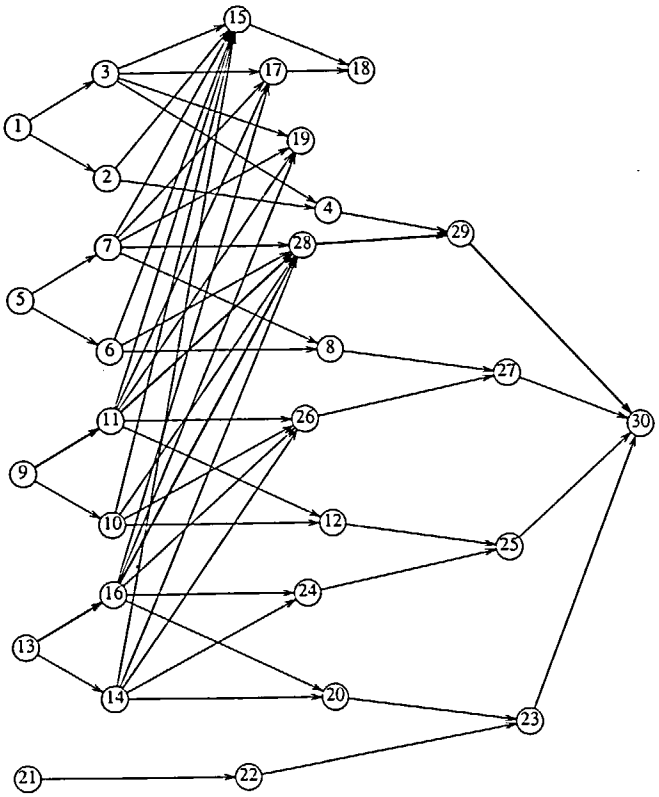

Fig. 5. Topological ordering graph for 4-bit ALU.

bound to the timing constraint t_{spec}, the following inequality constraint must be satisfied:

$$t_d = t_9 + t_{11} + t_{28} + t_{29} + t_{30} \leqslant t_{spec} = T \tag{10}$$

The notation $P_c(s, t)$ is used to represent a portion of a critical path starting from vertex s and ending at vertex t and $P(s, t)$ represents a portion of any path other than the critical one. If s is the primary input (PI) and t is the primary output (PO), $P_c(s, t)$ is the complete critical path and is abbreviated as P_c. For the graphical representation in Fig. 5, $P_c(11, 30)$ represents the path $V_{11} \rightarrow V_{28} \rightarrow V_{29} \rightarrow V_{30}$, and $P(11, 30)$ can be either $V_{11} \rightarrow V_{26} \rightarrow V_{27} \rightarrow V_{30}$ or $V_{11} \rightarrow V_{12} \rightarrow V_{25} \rightarrow V_{30}$. From the definition of a critical path, the following two relations should hold:

$$\sum_{v_i \in P_c} t_{di} = t_{max} \tag{11}$$

$$\sum_{v_i \in P(s,t)} t_{di} + t_{slack(i)} = \sum_{v_i \in P_c(s,t)} t_{di}, \tag{12}$$

where v_i is the ith vertex, t_{di} is its delay time, and $t_{slack(i)}$ is the associated slack time. Since the slack time $t_{slack(i)} \geqslant 0$, the following inequality is also true:

$$\sum_{v_i \in P(s,t)} t_{di} \leqslant \sum_{v_i \in P_c(s,t)} t_{di}. \tag{13}$$

This relation is useful to find the timing constraints for paths other than the critical ones and will be applied to the timing allocation scheme in the next section.

4.2. Gate-based optimization

To have a global optimal solution efficiently, all the variables in a circuit should be optimized simultaneously. However, since the total number of design variables is proportional to the gate or transistor counts in the circuit, such a global optimization approach is practical only for small circuits and is prohibitively expensive for big circuits with several hundreds or larger number of design variables. More discussion on optimization approaches will be presented in Section 5. In the case of polycell-based layout, extraordinarily large transistor widths can cause a significant consumption of silicon area when all cells are assembled together and should be avoided whenever possible; a large transistor has to be either split into smaller ones in parallel which increases the horizontal cell width and the layout complexity or other cells in the same row have to be stretched to match the largest transistor height.

In order to manage all the requirements of *large circuit, good performance* and *compact layout*, we propose a gate-based optimization scheme followed by a dynamic silicon resources adjustment for better performance and area tradeoff.

The algorithm reads as

```
for (i=1;i<=k;++i) {
      optimize objective function F(i);
}
silicon resource adjustment;
```

where k is the number of gates in a circuit and $F(i)$ is the objective function for the gate i. The gate-based optimization scheme makes the complexity of the optimization algorithm linear to the number of gates and therefore is able to handle large circuits. The dynamic adjustment scheme makes good balance between performance and silicon resources and therefore guarantees the layout quality while meeting the timing constraint.

4.3. Sensitivity and timing allocation scheme

In order to perform gate-based optimization, each gate needs its own timing constraint or budget. The total timing budget is allocated to individual gates according to

- Complexity of the gate, and
- Improvement factor of the gate.

The complexity of a gate is associated with *fan-in count, fan-out count* and *number of transistors in series*, i.e., s and p in Section 1. The *improvement factor* (IF), denoted by ζ is its capability to get speed improvement with the increase in transistor size and is proportional to its sensitivity of the delay time with respect to the transistor size.

Given a cell design, the cell delay time can be expressed qualitatively as an exponential function of its active area A.

$$t_d = a_1 \, e^{-b_1 A}$$

where a_1 and b_1 are positive constants. The differential or small change sensitivity of the delay time with respect to the area, denoted as S_A^t is shown as

$$S_A^t = \left| \frac{\Delta t}{\Delta A} \right|_{\Delta A \to 0} = \left| \frac{\partial t}{\partial A} \right|$$

Therefore, the sensitivity S_A^t [3] is also an exponential function of A and represents a monotonically decaying behavior. The delay sensitivity of a cell is usually large when the delay time is big, which corresponds to the case wherein transistor sizes or the area are small. As the sizes keep increasing and the delay time is brought down, it becomes more difficult to improve the performance, which means the sensitivity S becomes smaller. When the sensitivity S falls below a critical value S_c, it is not worthwhile to push performance any further by increasing the silicon

[3] Unless otherwise stated, the superscript and the subscript will be dropped from now on for simplicity.

area, since the drastic increase in area is not justified by the small improvement in performance.

From the electrical optimization point of view, a gate with larger S will have more potential to improve the whole circuit performance for the given amount of silicon resources. For a gate with small S, even a marginal improvement in delay performance will cost large silicon area. We therefore define the *improvement factor* (IF) to be

$$IF_i = \zeta_i = \frac{\Delta t_i}{t_i} = \frac{S_i \, \Delta A}{t_i} \tag{14}$$

With the same amount of area increase for each cell, the new delay time for the ith cell in the critical path will become $t_i \times (1 - \zeta_i)$ and the critical path delay will be $t_N = \Sigma_i t_i \times (1 - \zeta_i)$. If the total delay time is greater than the timing constraint, the total timing budget will be allocated to individual gates according to the following equation

$$t^i_{\text{alloc}} = t_{\text{spec}} \times \frac{t_i(1 - \zeta_i)}{t_N} \tag{15}$$

where t_{spec} is the timing constraint and t^i_{alloc} is the timing specification allocated to the ith cell. This timing allocation equation is applied to the critical path first and then to other paths iteratively with the path reduction equation of eqn. (13) until every cell is assigned a proper timing budget. The timing allocation algorithm is summarized as follows:

```
TimeAlloc( )                    /* timing allocation */
{
      Critical Path Identification(PERT);
      TimingSlice eqn. (15);
      while (not every cell has its own timing constraint)
{
            Path Reduction eqn. (13) to find sub-
            constraint;
            TimingSlice of eqn. (15);
      }
}
```

After every cell is allocated a timing constraint t_i, the optimization problem of eqn. (1) can be restated as

$$\text{minimize}(t_{di}(W) \times A_i(W)) \tag{16}$$

subject to

$$W_{\min} \leqslant W,$$

$$t_{di}(W) \leqslant t^i_{\text{alloc}},$$

$$\beta_{\text{R(min)}} \leqslant \beta_{\text{R}} \leqslant \beta_{\text{R(max)}}.$$

where $A_i(W)$ is the area of the cell i.

5. Optimization scheme

5.1. Previous approaches

There are basically two techniques in solving constrained optimization problems. One is the incremental method [19,20], in which every design parameter is optimized with pre-defined step until the timing specification is met. This procedure is time consuming and often the optimal solution is not guaranteed. The second approach is by applying the classical augmented Lagrangian multiplier method [21–23]. While this method is more widely used than the incremental method, it has some restrictions. Because this method requires taking derivatives with respect to each design parameter, the cost function has to be well behaved or differentiable. To guarantee the differentiability of the delay function, polynomial cost functions have been formed based on RC delay models [14,23]. Unfortunately, the accuracy was sacrificed for simplicity and hence the subsequent optimization process can produce inferior solutions. Furthermore such an approach is not well suited for handling important constraints on circuit reliability problems such as noise margins and charge sharing.

5.2. Optimization using non-RC delay models

Unlike the traditional approach using RC delay models, new analytical delay models are derived and used for both static and domino gates based on device parameters and I-V characteristics. The delay models have an accuracy of less than 10% error as compared to SPICE simulations. Due to space limitation, the delay models can not be elaborated here and readers are referred to [18,37]. The following optimization technique can also be used with other analytical models for CMOS gates, as long as the functions are continuous.

The general *two-phase method* concept is applied to our optimization problem. In phase I, a *feasible solution*, which satisfies the timing specification and all other constraints but not necessarily optimal, is obtained. The phase II searches for the optimal solution within the *feasible design space*. The optimization flowchart is shown in Fig. 6.

In phase I the cost function is simply set to be the delay time t_d, i.e., as long as the delay time becomes smaller than the previous value, the new set of design variables will be accepted. Phase I is continued until the delay time is brought down to within the timing specification t_{spec}. At this point, the feasible solution is found and the optimization process enters Phase II with the cost function switched to be the delay–area product.

The central optimization method is Rosenbrock's rotating coordinating scheme [25], which is a direct search method. Since the direct search method requires only the ability to evaluate the function at any given point and requires no derivatives, it can be used for all continuous functions. Since the derivatives are not required, the following advantages are obvious:

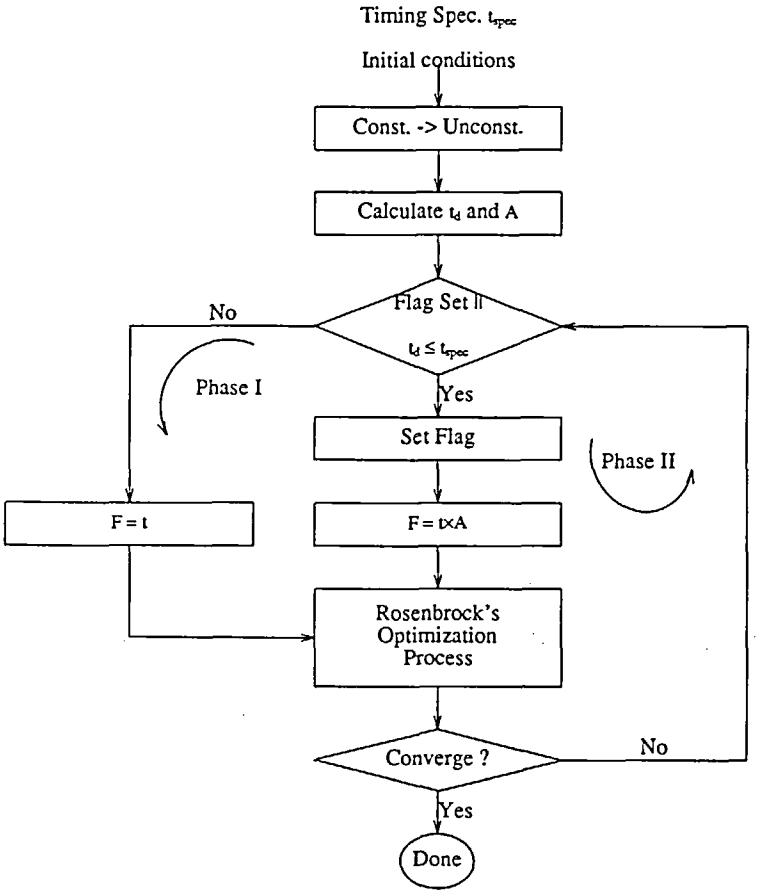

Fig. 6. Optimization flowchart.

- the cost function, as long as it's continuous, can be made general enough to account for all significant physical meanings and to use accurate delay models;
- the cost function is not required to be differentiable whereas other approaches require a special treatment for nondifferentiable functions [26]; and
- the technology update can be quickly achieved by easy change of the cost function.

5.3. *Transformation of independent variables*

Both the channel width and β ratio in the optimization problem are bounded by the upper and lower limits and have the general form:

$$l_i \leqslant x_i \leqslant u_i$$

Since the unconstrained optimization problem is easier to solve, the following transformation technique is applied to the constraints so that the constrained problem can be converted into an unconstrained one with new variables:

$$x_i = l_i + (u_i - l_i) \sin^2 x_i'$$

where x_i' is a new variable to be optimized. It is important to note that such transformations do not introduce additional and essentially distinct local optima [18].

5.4. Rosenbrock's rotating scheme

A straightforward method for the multivariable optimization problem is to try each variable in turn while all the others are kept constant, which is called the alternating variable method. The Rosenbrock's method [25] modifies this alternating variable method in two ways to produce one of the most robust optimization methods. The first modification is to avoid the single variable optimization for each direction in turn. Instead, a predefined step length is taken in each direction and these step lengths are modified after each calculation. If a step is taken in the x-direction and a better result is obtained, this is considered as a *good* direction and the step length will be lengthened by α times for the next exploration in the x-direction. If a *worse* result is obtained, the step length will be shortened by β times in the opposite direction for the next search in the x-direction. This procedure is followed as each of the variables is considered in turn.

The second modification is to recognize that the alternating variable method takes a large number of very small steps. To avoid this slowdown, the Rosenbrock's method realigns the axes when a good followed by a bad result has been obtained in *each* of the n dimensions for the n-variable problem. The axes are reoriented so that the first axis is along the most successful overall direction, the second axis along the next most successful direction and so on. The most successful overall direction in some sense is equivalent to the steepest direction in the gradient optimization methods. The real question, however, is to decide *when* and *how* to change the axes. Based on experience, it is found that the most useful criterion is to change axes when a good followed (not necessarily immediately) by a bad result has been recorded in *each* of the n dimensions. The requirements of waiting for at least one success in each direction has the effect that no direction is lost. Since coordinates are allowed to rotate such that one of the axes points along the gradient direction, this scheme is able to follow sharp valleys in the topology of the objective function. This property makes the Rosenbrock's scheme very robust [27]. The axis rotation can be achieved by performing Gram-Schmidt orthogonalization process as stated below.

Gram–Schmidt Orthogonalization Process [28]. Given N linearly independent vectors $e_1, e_2, \ldots, e_N$, N orthonormal non-zero vectors $q_1, q_2, \ldots, q_N$ can be constructed from them. The process is shown in the procedure:

$$u_i = e_i - \sum_{k=i}^{i-1} \langle q_k, e_i \rangle q_k$$

$$q_i = \frac{u_i}{|u_i|}, \qquad (i = 1, 2, \ldots, N)$$

Here $\langle x, y \rangle$ is the inner product of x and y.

5.5. Resource balance scheme

Although the individual timing budget is carefully allocated to each gate, well balanced usage of the silicon resource still can not be guaranteed without proper resource constraints and the post-optimization adjustments.

5.5.1. Resource constraint scheme – local constraint

During the optimization process, if the improvement factor of any individual cell is less than a critical value, a new set of variables will be searched for. If the timing budget of a cell can not be met, the excess amount of delay will be deducted from the timing budget of another cell in the path. This local constraint is to prevent extraordinarily large transistor sizes.

5.5.2. Resource redistribution scheme – global constraint

The global resource redistribution process is executed after the gate-based optimization to fix any big area skew. Beginning with the critical path, the process compares the largest and smallest transistor sizes in the path and confines the ratio by a user-specified limit, i.e.,

$$\frac{\text{Largest transistor size}}{\text{Smallest transistor size}} \leqslant \text{Upper limit}$$

If the above constraint is violated, the timing specification to those two cells will be reassigned. With this global area balancing scheme, extremely large transistors can be tuned down before layout.

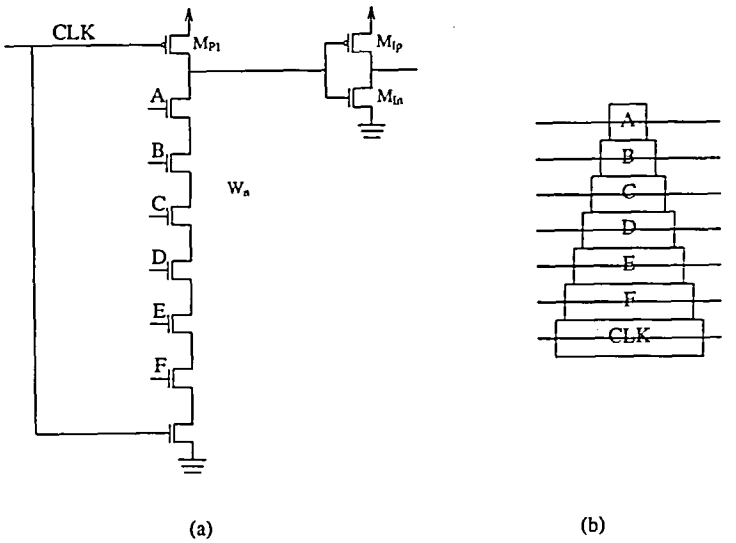

Fig. 7. (a) AND6 circuit implement with domino CMOS; (b) Tapered layout style for n-logic function.

6. Optimization examples

First, a domino CMOS AND6 circuit, shown in Fig. 7(a), although rather simple, is presented to illustrate the optimization principle and the procedure.

In this example, only W_n, W_{ip} and W_{in} are considered as the design variables to be optimized. The size of the precharge p-channel transistor is determined by the global clock rate. Although a uniform size W_n has been used for transistors in the logic evaluation tree, our optimization results already outperform the tapered layout style of Fig. 7(b) [29]. However, if desired, the tapering can also be applied for fine-tuning. This case will be elaborated in Section 6.2. The second example illustrate the optimization of a NORA one-bit full adder circuit in the latter part of Section 6.2.

6.1. Minimization of the delay with area limit

For the first example, with the area A in eqn. (1) set to unity, we can calculate the lowest achievable delay as long as the silicon area is not of major concern. The delay time is monotonically decreasing with channel width W_n of the n-channel devices in the logic block up to about two thousand microns, which is too big for non-I/O cells. Therefore, it is necessary to set a feasible upper bound for those n-channel devices in the logic block. On the other hand, it is unnecessary to have a upper bound for W_{In} since the delay tends to creep up beyond some point of diminishing return. Table 1 shows the results with three different initial conditions. They essentially converge to the same point. The upper limit for transistor size is set to 100 μm in this case.

6.2. Minimization of the area with delay limit

Table 2 shows the results with the delay specification imposed to the first circuit example. The first two rows show the result with 10 ns timing specification and the last two rows show the result with 8 ns specification. It can be noted that the last column SPICE delays are slightly bigger than the spec values due to the error of analytical delay models. This problem is easily fixed by lowering the spec values with proper error margins.

Table 1
Best performance table – Minimum delay time

Initial value			Optimal value				SPICE
W_n^0(μm)	W_{In}^0	β_R	W_n(μm)	W_{In}	W_{Ip}	t_d	t_d
4	4	0.5	100	4.0	80	6.83	7.2
10	50	3	100	4.0	80	6.83	7.2
10	30	5	100	4.0	80	6.83	7.2

Table 2
Optimal solutions with specified delay time – Minimum silicon area

Initial value				Optimal value			SPICE
$W_n^0(\mu m)$	W_{In}^0	β_R	spec.(ns)	$W_n(\mu m)$	W_{In}	W_{Ip}	t_d
4	4	0.5	10.0	40.27	4.18	43.8	10.7
10	50	3	10.0	40.5	4.8	44.7	10.5
4	4	0.5	8.0	66.22	4.0	59.16	8.7
10	50	3	8.0	66.27	4.1	59.26	8.7

Recently, Shoji [29] has introduced a transistor scaling technique to increase the speed of gates consisting of series combinations of transistors. The size of each transistor is scaled according to its position in the series structure when the following condition is met:

$$C_{1C} < \frac{N-1}{2} C_{1D}$$

where N is the number of series-connected transistors, C_{1D} is the capacitance associated with the topmost transistor and C_{1C} is the remaining capacitances contributed to the precharge node. $C_1 = C_{1C} + C_{1D}$ is the total precharge node capacitance. This tapered sizing style is shown in Fig. 7(b) for the AND6 example. The following formula is used to determine the size of each transistor in the n-logic block.

$$W_n(i) = W_n(3)[1 - \alpha(i-3)]$$

where $W_n(3) = 44$ μm is the channel width of the third transistor with $n = 0$ at the bottom. α is the scaling factor and has been shown to give the best delay performance with $\alpha = 0.29$ for this example. Note that the value of $[1 - \alpha(i-3)]$ has to be greater than zero. Once the value of α is chosen, the maximum number of series transistors is also set. In this approach, the inverter buffer is treated only as a load and no attempt is made to adjust the size, although it is in fact an important design parameter. With $W_{In} = 4$ μm, we vary W_{Ip} from 10 μm to 80 μm and tabulate the simulated delay time as well as the total active area in Table 3 [4]. All the cases were run with MOSIS 3 μm process parameters. Table 3 shows that row entry 6 with $W_{Ip} = 60$ μm has a slightly better speed performance over other cases. Row entry 4 with $W_{Ip} = 40$ μm has the best delay–area product value.

To compare with the best performance and the best delay–area product cases in Table 3, we ran our optimization program with specification of 11.2 ns and 10.7 ns for arbitrary feasible initial conditions for W_n^0, W_{In}^0, β_R and obtained optimal values in Table 4. The slight discrepancies between the SPICE delays and spec values are again due to the error in the delay model.

[4] The inverter transistor sizes W_{In} and W_{Ip} chosen in Ref. [29] are 5 μm and 9 μm, respectively, which make the delay time worse than any of the entries in the table while the active area remains the same as that in the first entry.

Table 3
Delay–area product of Shoji's case – AND6 example (Shoji's scheme)

W_{In}	W_{Ip}	t_d(ns)(SPICE)	Area(μm^2)	Delay × Area ($ns \cdot \mu m^2$)
4	10	18.4	2300.4	42327.4
4	20	13.3	2400.4	31925.3
4	30	12.0	2500.4	30004.8
4	40	11.2	2600.4	29124.5
4	50	11.0	2700.4	29704.4
4	60	10.7	2800.4	29964.3
4	70	10.8	2900.4	31324.3
4	80	11.1	3000.4	33304.4

Table 4
Delay–area product of our optimization scheme – AND6 example (new optimization scheme)

Initial value				Optimal value			Actual Cost	
$W_n^0(\mu m)$	W_{In}^0	β_R	spec.(ns)	$W_n(\mu m)$	W_{In}	W_{Ip}	SPICE (t_d)	Area
10	30	5	11.2	32.68	4.30	38.91	11.8	2139.3
4	4	0.5	10.7	35.25	4.46	41.06	10.8	2265.2

The delay–area product of row 1 is 25243.7 $ns \cdot \mu m^2$ and that of row 2 is 24464.2 $ns \cdot \mu m^2$. It can be observed from the above results that our optimization scheme yields 18% saving in active area with equal delay time performance [30]. Both results are also shown in Fig. 8.

This example demonstrates that the inverter buffer of the domino module does play an important role in the circuit performance and non-inclusion of the inverter buffer as part of the domino module design optimization often results in sub-optimal designs. In our case, after sizing W_{Ip}, W_{In} and W_n, we found no

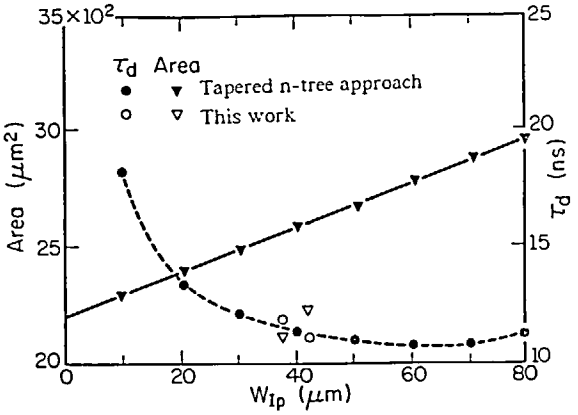

Fig. 8. Delay–Area comparison of AND6 circuit example.

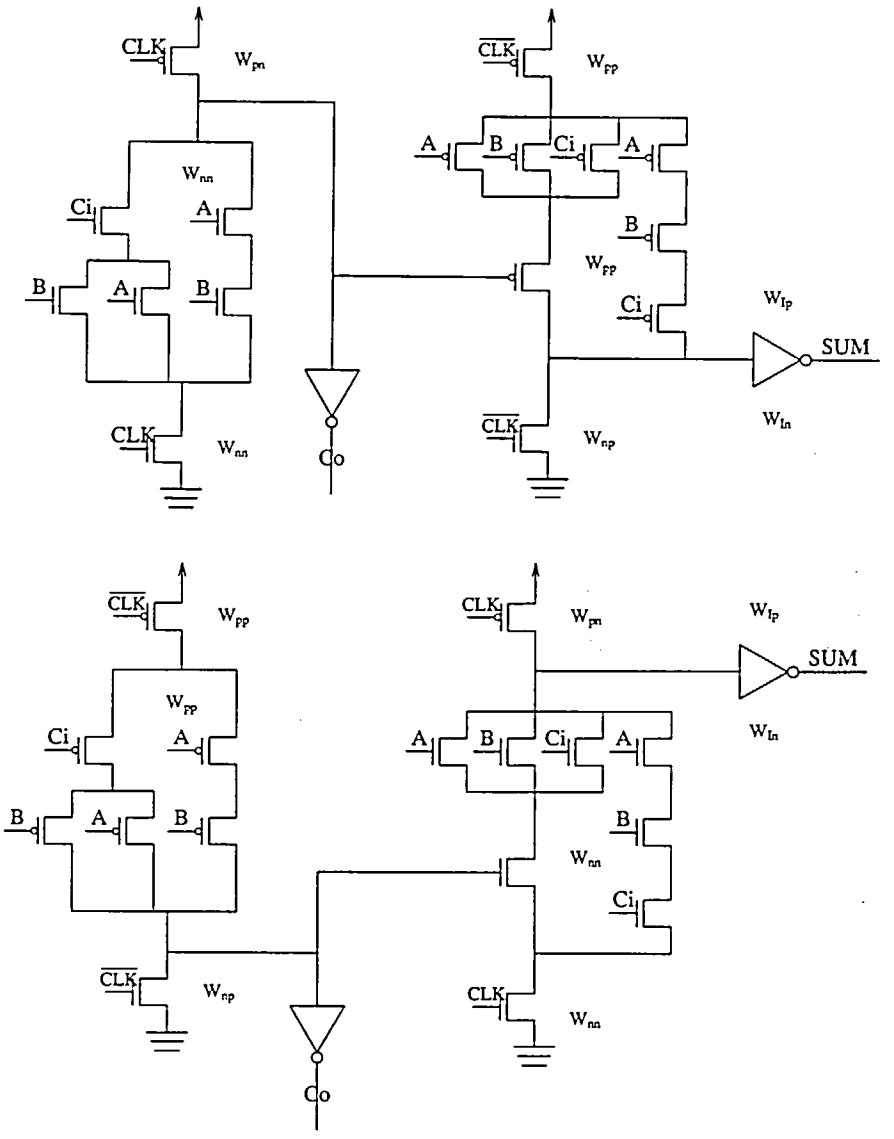

Fig. 9. (a) One-bit full adder (Type I); (b) One-bit full adder (Type II).

improvement in delay time even when we tried to taper the n-channel transistors [34]. However, the transistor sizes can be still gradually reduced (tapered), if beneficial, after determining W_n.

The second example is a one-bit full adder implemented with a NORA CMOS circuit. Figure 9(a) shows the implementation with CARRY bit generated by nMOS logic and SUM bit generated by pMOS logic. Figure 9(b) shows another implementation with CARRY bit generated by pMOS and SUM bit generated by nMOS. We call the circuit in Fig. 9(a) Type I and the circuit in Fig. 9(b) Type II. Table 5 shows the optimized transistor sizes with and without timing constraints.

Table 5
One-bit full adder circuit implemented with NORA CMOS – Optimization results

	Without timing constraint		With timing constraint	
	Type I	Type II	Type I	Type II
$W_{pn}(\mu m)$	30	30	30	30
$W_{np}(\mu m)$	30	30	30	30
$W_{nn}(\mu m)$	14.72	16.08	22.78	26.32
$W_{pp}(\mu m)$	21.93	21.04	34.20	37.41
$W_{In}(\mu m)$	23.6	4.0	30.15	4.07
$W_{Ip}(\mu m)$	4.0	29.5	4.01	43.61
t_d (ns) (Model)	12.67	13.4	10	10
t_d (ns) (SPICE)	13.1	15.4	10.7	11.2
Area(μm^2)	1717.2	1757.5	2252.4	2482.3

Again, all the precharge transistor sizes are predefined and are not optimized in the program.

7. Folding layout for unbalanced CMOS cells

iCOACH supports both static and dynamic gates. Static CMOS is used to implement primitive gates and dynamic CMOS to implement complex gates. For static gates, the number of n- and p-channel transistors are equal and can be paired and stacked vertically underneath common poly runners. For dynamic CMOS functional cells, which have a highly unbalanced circuit structure, i.e., the number of p-channel transistors is much less than that of n-channel transistors, we use the *folding layout* [30] which almost equally divides the n-channel transistors into two rails by using area-saving criteria and the delayed binding concept [31]. This layout style can also be applied to other unbalanced circuit structures like nMOS circuits which use a depletion mode transistor as load and enhancement mode transistors for logic implementation.

Two examples of this folding layout style for dynamic circuits are shown in Fig. 10 and Fig. 11. A portion of a 4-bit carry look ahead circuit (74182) is shown in Fig. 10(a). Its circuit implementation after local optimization (factorization in this case) is shown in Fig. 10(b). Here all input signals are distinct. The layouts before and after electrical optimization are shown in Figs. 10(c) and (d). The second example in Fig. 11 was taken from DAC '86 F2 [32] by ignoring all the inverters to complement the input signals. The logic and circuit diagram after local optimization are shown in Figs. 11(a) and (b). Here, not all input signals are distinct and permutations are made for a more compact layout as shown in Fig. 11(c). In these layouts, no attempt has been made to replace long diffusion runners with metal runners to reduce RC parasitics. In many cases, such fine-tuning can be achieved as long as no horizontal metal runners are present. Also the use of silicided shallow source and drain technology [35] can significantly reduce

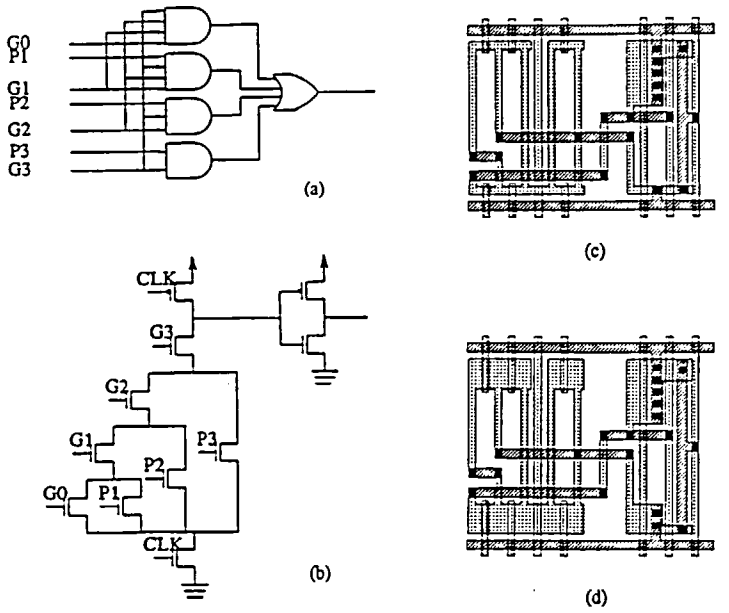

Fig. 10. (a) Logic diagram; (b) Circuit schematic; (c,d) Corresponding folding layouts.

the parasitics. In domino complex gates, the RC parasitics in the internal nodes are not so detrimental as in the case of static gates when the pull-up transistor size W_{Ip} is properly chosen [36].

Since the diffusions are folded for dynamic gates, some input signals are accessible only from top or bottom as shown in Figs. 10 and 11. In the worst case, all input signals can be distinct and every poly gate signal would run only half of the cell height. An interesting question is then whether the area saving achieved

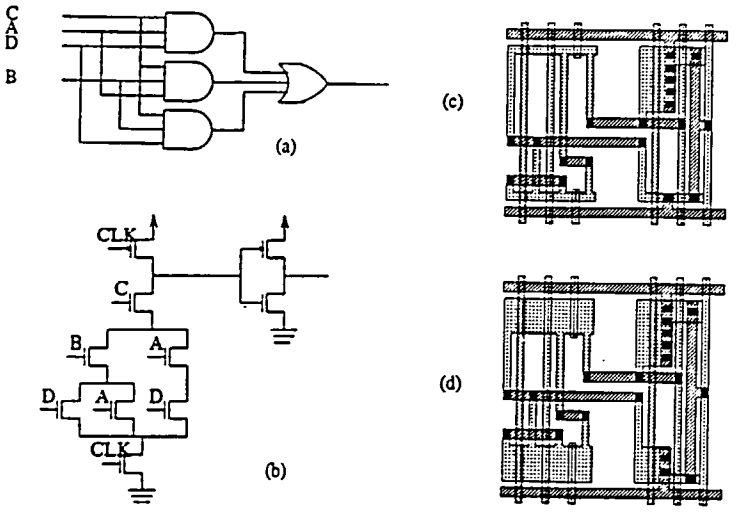

Fig. 11. (a) Logic diagram; (b) Circuit schematic; (c,d) Corresponding folding layouts.

by folding the diffusion rail would be overshadowed by the extra area introduced by feedthroughs. The following analysis shows that judicious folding layout does achieve area savings. We first define the following notations:

A_t = total chip area in conventional polycell layout
A'_t = total chip area in the folding layout
A_{ch} = total channel area in conventional polycell layout
A_{ce} = total cell area in conventional polycell layout
A'_{ch} = total channel area in the folding layout
A'_{ce} = total cell area in the folding layout
N_{ft} = number of total feedthroughs used with the folding layout
N_c = number of feedthroughs which are necessary in conventional polycell approach
N_{dg} = number of dynamic gates
N_t = average number of transistors folded
K_1 = an empirical ratio of channel area vs. cell area in conventional polycell approach A_{ch}/A_{ce}
K_2 = an empirical ratio of channel area vs. cell area in folding layout A'_{ch}/A'_{ce}
A_{da} = cell area difference between two layout styles $A_{ce} - A'_{ce}$
A_d = total area difference between two layout styles $A_t - A'_t$
P = horizontal pitch for poly columns
H = vertical height of cell

We first calculate the cell area difference A_{da} between two layout styles. In comparison to the conventional layout style, the area savings due to the folding layout is

$$A_s = (N_t \times N_{dg}) \times P \times H \tag{17}$$

and the area increase due to additional feedthroughs is

$$A_i = (N_{ft} - N_c) \times P \times H \tag{18}$$

The net area saving can be achieved so long as $A_s > A_i$ or

$$(N_t \times N_{dg}) \times P \times H > (N_{ft} - N_c) \times P \times H \tag{19}$$

Assume that no feedthroughs are necessary in conventional polycell layout approach (although not true), i.e., $N_c = 0$, the above expression becomes

$$N_t > \frac{N_{ft}}{N_{dg}} \tag{20}$$

Table 6
Results of folding layout style – Experimental data

Example	No. of Trans.	No. of Cells	No. of Runs	No. of Feedthrus	No. of Nets	No. of Blocks	No. of Dynamic cells
4-bit ALU	209	30	5	7 to 13	41	4	12
32-Bit Adder	978	148	5	37 to 50	167	6	42

Normally N_t is greater than 2 and therefore cell area saving is achieved as long as the number of feedthroughs is less than twice the number of dynamic gates. Table 6 shows two circuit examples both of which have the worst situations, i.e., after local optimizations (mainly factorization) no common signals exist in the cell except the precharge transistor pair and the inverter buffer.

From the table, it can be observed that the folding layout does achieve cell area savings even in worst cases. The reason is that although it is desirable for the signals to be accessible from both the top and bottom, in many cases the signals terminate in the cells and need not travel any further. Many nets can usually be confined within a single channel when the minimum net length is used as the cost functions in the layout system.

Next we consider the effect of folding layout on the whole chip area. Note that $A_t = A_{ch} + A_{ce}$, and $A'_t = A'_{ch} + A'_{ce}$, and also $A_{ch} = K_1 \times A_{ce}$. Therefore, $A_t = (1 + K_1)A_{ce}$, and $A'_t = (1 + K_2)A'_{ce}$, which can be also expressed as $A'_t = (1 + K_2)(A_{ce} - A_{da})$. The total area difference is then $A_d = A_{ce}(K_1 - K_2) + A_{da}(1 + K_2)$.

Case 1. $K_1 \geqslant K_2$. Both terms are greater than zero and the area saving is obvious.

Case 2. $K_1 < K_2$. The following inequality should hold to achieve the total area saving:

$$A_{ce}(K_2 - K_1) \leqslant A_{da}(1 + K_2).$$

Normally, K_1 and K_2 are about same and range from 1 to 3, which shows that the inequality holds.

8. iCOACH applications

iCOACH is coded in C and runs on SUN 3/50. It currently supports MOSIS 3 μm CMOS technology, but the technology update can be achieved easily by updating delay models and layout parameters. Tables 7 and 8 summarize iCOACH results for a 4-bit ALU and a 32-bit carry-look-ahead adder which are depicted in Fig. 12 and Fig. 13. The active area increase is steeper than that of total chip area

Table 7
Area versus delay for 4-bit ALU – 4-Bit ALU (209 transistors)

Delay time t_d (ns)	Total chip area (μm^2)	Total active area (μm^2)	Optimization time (s)	Total run time (s)
63	3.4×10^5	1.3×10^4	–	9
38.3	3.78×10^5	1.97×10^4	162	173
30	3.9×10^5	2.81×10^4	277	286
20	5.6×10^5	7.34×10^4	371	381

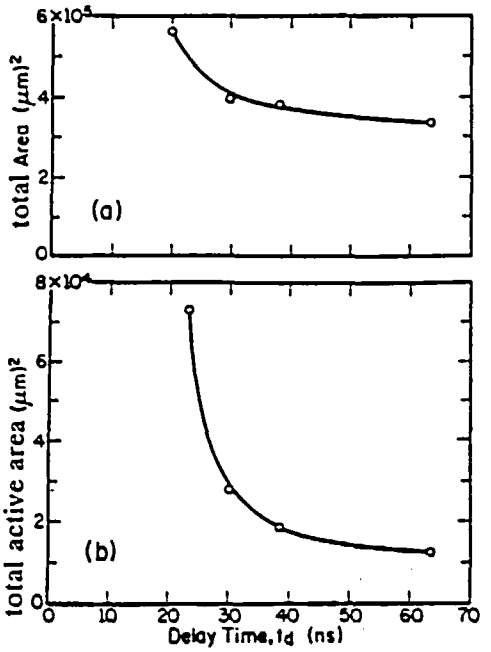

Fig. 12. 4-bit ALU: (a) Delay vs. total chip area; (b) Delay vs. total active area.

and follows the trend of single cells. The slower increase in total chip area is expected as the area increase in single cell may not affect the total area as much.

A full 32-bit carry-look-ahead adder has been optimized and laid out by

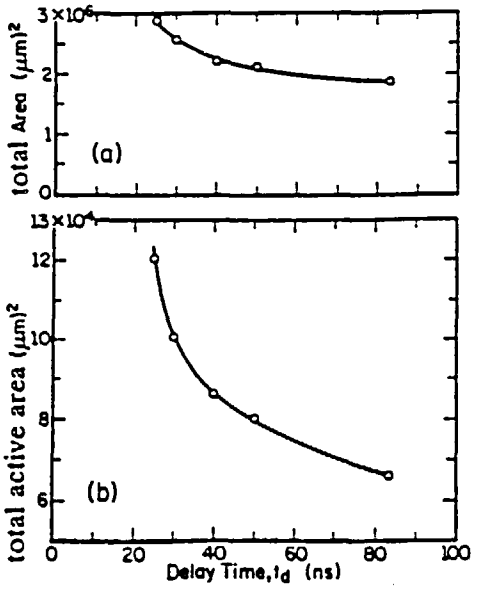

Fig. 13. 32-bit adder: (a) Delay vs. total chip area; (b) Delay vs. total active area.

iCOACH. Figure 14(a) shows the layout before optimization which has 83 ns delay time in the critical path. When the delay constraint is set to 30 ns, iCOACH reduces the critical path delay time to 30 ns with 37% increase in area as shown in Fig. 14(b).

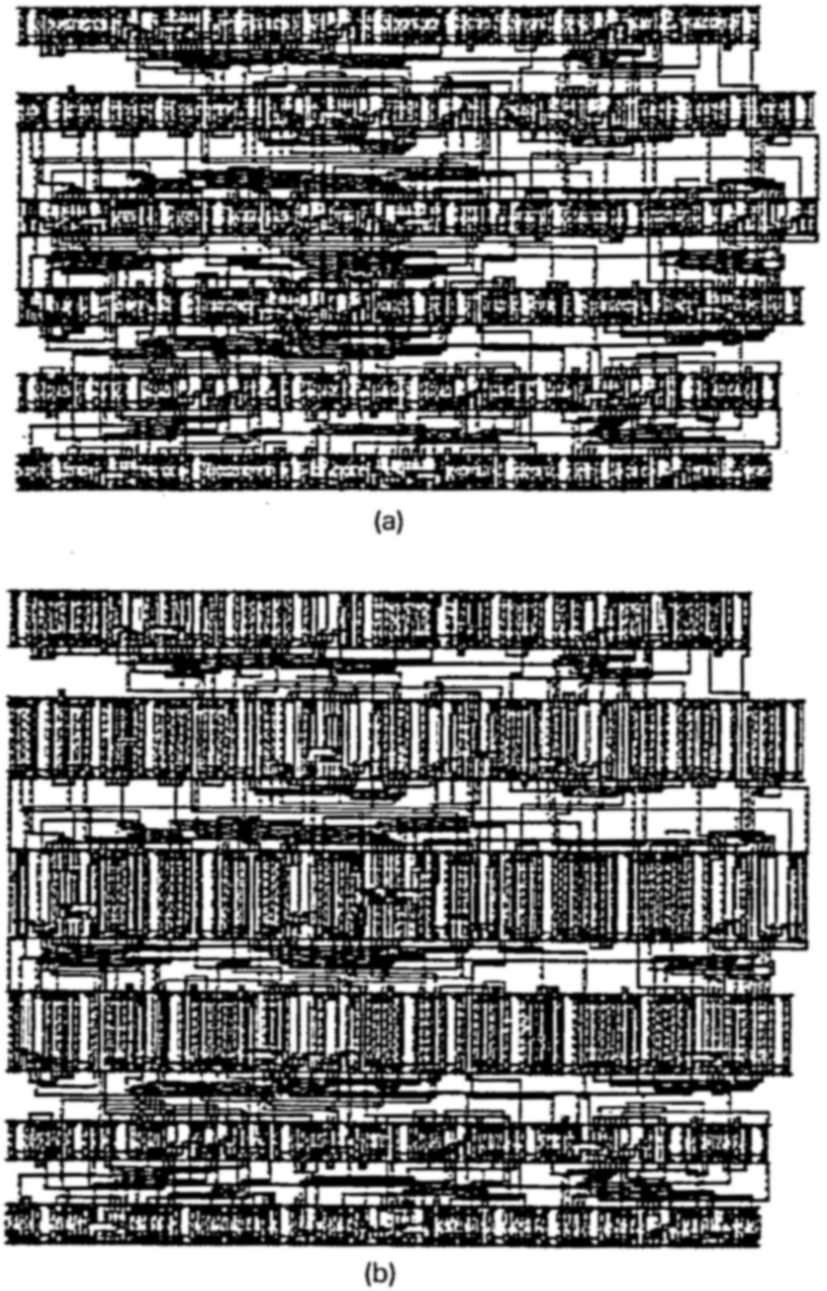

Fig. 14. (a) Layout of 32-bit carry-look-ahead adder before optimization; (b) Layout of 32-bit carry-look-ahead adder after optimization.

Table 8
Area versus delay for 32-bit adder – 31-bit carry-look-ahead adder (978 transistors)

Delay time t_d (ns)	Total chip area (μm^2)	Total active area (μm^2)	Optimization time (s)	Total run time (s)
83	1.9×10^6	6.57×10^4	–	41
49.7	2.13×10^6	7.98×10^4	86	130
40	2.2×10^6	8.62×10^4	231	274
30	2.6×10^6	10.07×10^4	618	662
25	2.95×10^6	12.05×10^4	892	933

9. Conclusions

An integrated design optimization system iCOACH which performs the circuit optimization at the transistor level is presented. The timing specification is accommodated as a design constraint and reliability issues on the charge sharing and noise margins are also modeled and embedded into the design constraints, which has not been incorporated in previous optimization tools. The optimization scheme does not require derivatives and therefore can be used for general continuous cost functions. Special attention is also given to the most effective usage of silicon area resource. Excessively large transistor sizes are avoided by investigating the sensitivity of delay versus area which leads to the timing allocation and area balancing schemes. Because of the timing allocation scheme, the computation time is approximately linearly proportional to the number of gates. Two examples demonstrate that iCOACH results are competitive with fine-tuned manual designs. It can be also noted from Table 7 and Table 8 that the total run time of iCOACH increased sublinearly from 381 seconds for 209 transistors to 933 seconds for 978 transistors.

The folding layout scheme further improves the designs by increasing the packing density of unbalanced CMOS cells such as domino CMOS and NORA circuits. Detailed analysis and experimental results have been presented to show the area savings using realistic circuit examples.

The underlying principles of iCOACH should allow incorporation of different modules, either timing analyzer or cell generator, without much difficulty.

10. Acknowledgements

The authors wish to thank reviewers for providing many constructive and helpful comments. This work was supported in part by the Semiconductor Research Corporation contract RSCH 86-12-109 and the Joint Services Electronics Program (JSEP) contract N00014-85-C-0149.

References

[1] Hong, S.J., R.G. Cain and D.L. Ostapko, MINI: A heuristic approach for logic minimization, *IBM J. Res. Develop.* **18** (9) (September 1974) 443–453.

[2] Brayton, R.K., G.D. Hachtel, C.T. McMullen and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis* (Kluwer Academic Publishers, Boston, 1984).

[3] Brayton, R.K., R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, MIS: A multiple-level logic optimization system, *IEEE Trans. CAD* **CAD-6** (November 1987) 1062–1081.

[4] Darringer, J., D. Brand, J. Gerbi, W. Joyner and L. Trevillyan, LSS: A system for production logic synthesis, *IBM J. Res. Develop.* **28** (September 1984) 537–545.

[5] de Geus, A.J. and W. Cohen, A rule-based system for optimizing combinational logic, *IEEE Design and Test of Computers* **2** (August 1986) 22–32.

[6] Keutzer, K., K. Kolwicz and M. Lega, Impact of library size on the quality of automated synthesis, *Proc. Int. Conf. CAD (ICCAD)* (November 1987) 120–123.

[7] Detjens, E. and G. Gannot, Technology mapping in MIS, *Proc. Int. Conf. CAD (ICCAD)* (November 1987) 116–119.

[8] Murphy, B.T., VLSI-direction and impact, *Proc. Eur. Solid-State Circuits Conf.*, Southampton, England, September 1979 pp. 50–53.

[9] Sechen, C. and A. Sangiovanni-Vincentelli, The TimberWolf placement and routing package, *IEEE J. Solid-State Circuits* **SC-20** (April 1985) 510–522.

[10] Reed, J., A. Sangiovanni-Vincentelli and A. Santamauro, A new symbolic channel router: YACR2, *IEEE Trans. CAD* **CAD-4** (July 1985) 208–219.

[11] Kirkpatrick, T.I. and N.R. Clark, PERT as an aid to logic design, *IBM J. Res. Develop.*, **10** (March 1966) 135–141.

[12] Keyes, R.W., The evolution of digital electronics toward VLSI, *IEEE J. Solid-State Circuits* **SC-14** (April 1979) 193–201.

[13] Glasser, L.A. and L.P.J. Hoyte, Delay and power optimization in VLSI circuits, *Proc. 21st Design Automation Conference*, June 1984, pp. 529–535.

[14] Matson, M., *Macromodeling and Optimization of Digital MOS VLSI Circuits*, Ph.D. dissertation, MIT, February 1985.

[15] Kang, S.M., A design of CMOS polycells for LSI circuits, *IEEE Trans. Circuits and Systems* **CAS-28** (August 1981) 838–843.

[16] Uyemura, J.P., *Fundamentals of MOS Digital Integrated Circuits*, Chapter 3 (Addison-Wesley Publishing, 1988).

[17] Weste, N. and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective* (Addison-Wesley Publishing, 1985).

[18] Chen, H.Y., *Design automation for high performance CMOS VLSI circuits*, Ph.D. dissertation, University of Illinois at Urbana-Champaign, UILU-ENG88-2258, December 1988.

[19] Kao, W., N. Fathi and C. Lee, Algorithms for automatic transistor sizing in CMOS digital circuits, *Proc. 22nd Design Automation Conference*, June 1985, pp .781–784.

[20] Lai, F.P., *Rule-based circuit optimization for CMOS VLSI*, Ph. D. dissertation, Univ. of Illinois at Urbana-Champaign, UILU-ENG-87-2244, July 1987.

[21] Hedlund, K.S., Aesop: A tool for automated transistor sizing, *Proc. 24th Design Automation Conference*, June 1987, pp. 114–120.

[22] Cirit, M.A., Transistor sizing in CMOS circuit, *Proc. 24th Design Automation Conference*, June 1987, pp. 121–124.

[23] Fishburn, J.P. and A.E. Dunlop, TILOS: A posynomial programming approach to transistor sizing, *Proc. Int. Conf. Computer Aided Design*, November 1985, pp. 326–328.

[24] Kang, S.M. and H.Y. Chen, Modeling of propagation delay for a class of domino CMOS circuits, *29th Midwest Symp. Circuits and Systems*, August 1986, pp. 907–910.

[25] Rosenbrock, H.H., An automatic method for finding the greatest or least value of a function, *Comput. J.* **3** (October 1960) 175–184.

[26] Shyu, J.M., A. Sangiovanni-Vincentelli, J.P. Fishburn and A.E. Dunlop, Optimization-based transistor sizing, *IEEE J. Solid-State Circuits* **23** (April 1988) 400–409.
[27] Schwefel, H.P., *Numerical Optimization of Computer Models* (John Wiley & Sons, 1981).
[28] Chen, C.T., *Linear System Theory and Design* (Holt, Rinehart and Winston, 1984).
[29] Shoji, M., FET scaling in domino CMOS gate, *IEEE J. Solid-State Circuits*, **SC-20** (October 1985) 1067–1071.
[30] Chen, H.Y. and S.M. Kang, An efficient layout style for dynamic functional cells, *Proc. 31st Midwest Symp. on Circuits and Systems*, Aug. 1988, pp. 866–869.
[31] Hwang, D.K., W.K. Fuchs and S.M. Kang, An efficient approach to gate matrix layout, *IEEE Trans. CAD* **CAD-6** (September 1987) 802–809.
[32] de Geus, A.J., Logic synthesis and optimization benchmarks for the 1986 design automation conference, *Proc. 23th Design Automation Conference*, June 1986 pp. 78–79.
[33] Mukherjee, A., *Introduction to nMOS and CMOS VLSI Systems Design* (Englewood Cliffs, NJ, Prentice-Hall, 1986).
[34] Chen, H.Y. and S.M. Kang, Performance optimization for domino CMOS circuit modules, *Proc. Int. Conf. Comp. Des. (ICCD)*, October 1987, pp. 522–525.
[35] Butler, A.L. and D.J. Foster, The formation of shallow low-resistance source-drain regions for VLSI CMOS technologies, *IEEE Trans. on Electron Devices* **ED-32** (2) (Feb. 1985) 150–155.
[36] Kang, S.M., Domino-CMOS barrel switch for 32-bit VLSI Processors, *IEEE Circuits and Devices Magazine* **3** (3) (May 1987) 3–8.
[37] Kang, S.M. and H.Y. Chen, A global delay model for domino circuits, *Int. J. Circuit Theory and Applications*, 18 (3) (1989) 289–306.