Performance-Oriented Synthesis of Large-Scale Domino CMOS Circuits

GIOVANNI DE MICHELI, MEMBER, IEEE

Abstract—**The quality of the design of large-scale integrated circuits is determined by such figures of merit as silicon area, power consumption, and switching-time performance. We address here the problem of the automatic synthesis of digital circuits with the goal of achieving high-performance designs. We assume we are given an intermediate circuit representation that optimizes area and/or power. We use timing optimization techniques to improve the circuit performance, possibly at the expense of the other figures of merit.**

We consider general classes of digital circuits, with a given partition into registers, combinational blocks, and I/O ports. Circuit performance is related to the worst-case propagation delay of signals between two register boundaries. In this context, circuit performance optimization is equivalent to minimizing the critical path delay through the combinational circuits. We assume a multiple-level implementation of the combinational logic, by means of an interconnection of logic gates implementing arbitrary multiple-input, single-output logic functions. We consider dynamic CMOS implementation of the logic gates, operating in the domino mode.

We present a global approach to timing performance optimization, which involves operations at the *logic, topological,* and *physical* level of abstraction of the circuit. In particular, at the logic level, we look for optimal structures of multiple-level combinational networks. At the topological level, we search for the optimal positions of gates or groups of gates. At the physical design level, we optimize MOS device sizes.

The algorithms are described, together with their implementation and the interface to the Yorktown Silicon Compiler system, which is an automated synthesis system described in [7]. The results of applying timing-performance optimization to a 32-bit microprocessor design are reported.

I. Introduction

THE RAPID EVOLUTION of electronic systems and the progress in performance/price ratios are related to the increasingly widespread use of very large scale integration (VLSI) circuits. Computer-aided design (CAD) tools are essential for designing integrated circuits, and their importance is increasing as designs become more complex. Moreover, the capability of quickly designing high-performance processors is key to commercial competitiveness. Fast-turnaround designs allow system-level tradeoff comparisons, by trying several architectures in the search for the best match between system structure and implementation technology. In this perspective, automated synthesis systems, or *silicon compilers*, are necessary for the development of integrated circuits now and in the future.

Manuscript received February 12, 1987; revised May 11, 1987.

The author is with the Computer System Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305.

IEEE Log Number 8715661.

In addition, circuit performance plays a key role in the commercial value of an integrated circuit. In the past, the first automated synthesis systems neglected this key design feature because of the complexity of achieving an automated design. Today, the automated synthesis systems technology has progressed to the point where performance-oriented synthesis has become a key issue [13], [1], [6], [7], [14].

We explore in this paper some of the problems of timing performance optimization of digital circuits in connection with automated synthesis. We present some algorithms that tackle the timing optimization problem at different levels: logic, topological, and physical. In particular, at the logic level, we look for optimal structures of multiple-level combinational networks. At the topological level, we search for the optimal positions of gates or groups of gates. At the physical design level, we optimize MOS device sizes. These techniques assume the existence of a circuit description that is generated by a synthesis system or by hand. The circuit representation, used as a starting point, is often determined by the criterion of optimizing some figures of merit of the circuit (e.g., area, power, total wiring length). In this paper we refer to timing optimization in a wide sense that includes tradeoffs among the above-mentioned figures of merit and timing performance. The algorithms have been specialized for use with the Yorktown Silicon Compiler (YSC) system [7] and they can be seen as the *code optimizer* part of the compiler, which can be invoked when compiling circuits with critical timing performances.

The Yorktown Silicon Compiler is an automated synthesis system that aims at generating circuit designs competitive with manual designs in silicon area, power, and switching-time performances. The circuit to be implemented is described in a behavioral language. The YSC transforms this representation, in a stage called structural synthesis, into a hierarchical interconnection of circuit blocks, called **modules.** The leaf modules are combinational logic units, registers, and library cells, e.g., I/O ports. Eventually, the YSC system generate the geometries of the masks of the chip. The quality of the "compiled" design is achieved by including several optimization procedures that modify the hierarchical structure and the module representations. By default, optimization at the logic level minimizes the silicon area. The floor plan design minimizes the total wiring length, which correlates

with minimal area and power. We refer the interested reader to [7] for the details.

Circuit performance can be optimized in the YSC system during the structural synthesis stage by allocating registers and by determining the system partition in terms of combinational logic modules, registers, and I/O ports. This global system structure is "frozen" after the structural synthesis stage. From this point on, circuit performance is related to the worst-case propagation delay of signals between two register boundaries, because the system clock has to be adjusted to allow the arrival of each signal to the destination registers within the clock cycle. In this context, the optimization of circuit performance is equivalent to the minimization of the critical path delay. We do not address here performance optimization in conjunction with the structural synthesis stage (see [7] and [10]). We present here techniques that apply to combinational logic circuits and, in particular, to the combinational modules that are connected between two register boundaries, as designed by the YSC system. We consider here only circuits designed in the dynamic CMOS technology and operating in the domino mode [20], [19].

Previous work on timing performance optimization addressed one particular level of the circuit representation. At the circuit level, Ruehli [22], Trimbereger [23], Fishburn [15], and Marple [21] proposed device-sizing optimization techniques. At the topological level, Burstein [9] (and, independently, Donath and Kurtzberg) studied and implemented placement and wiring strategies that optimize timing performances. At the logic design level, a fundamental contribution is due to Hitchock [16]. This is only a partial list of some significant contributions. Performance optimization was also achieved indirectly by methods that simplify the circuit complexity by reducing its area and/or number of gates (e.g., logic minimization) [17], [2] or by choosing a placement of the gates that minimizes the routine wire length [18].

We present here a global approach to timing optimization, which combines operations at different levels of the circuit representation, namely:

i) **resynthesis,** i.e., changing the multi-level structure of combinational logic;
ii) **resizing,** i.e., changing MOS device sizes;
iii) **repositioning,** i.e., changing the module positions.

Timing optimization can be seen as an iteration among these operations and an evaluation of the critical path delay until a satisfactory performance is obtained. A precise evaluation of the critical path delay requires the design to be complete. As an example, a complete geometric layout is needed to extract accurately the wiring capacitance, which plays an important role in the computation of the circuit delay. Since delay evaluations are needed before the circuit synthesis is completed, some assumptions and approximations are used for those circuit details that have not been determined yet. With these delay evaluations, a more detailed circuit description can be determined, which optimizes some timing-related criteria. In this perspective, synthesis and timing optimization can be seen as a stepwise refinement process.

In our approach, we use a worst-case estimate of the wiring capacitance computed on the basis of the gate/module positions. While this approximation provides circuit delay measurements that are less accurate than those based on capacitance extraction, the circuit model is more suitable for applying stepwise improvement techniques. The approximations in the circuit delay estimations are also related to the circuit synthesis flow. In particular, while the evaluation of the critical path delays requires a knowledge of the gate/module positions, the determination of the gate and module positions should also be driven by timing considerations. In our implementation, we estimate the delays early in the synthesis process, disregarding the gate positions and wiring length. The information about the critical nets in the circuit is then incorporated into the objective function of the floor-plan design stage of the synthesis. As a result, the module positions are influenced by timing considerations. These positions are then used to estimate the wiring. Based on a new critical path delay estimate, the positions of the circuit modules can then be readjusted. Then, resynthesis and resizing can be applied. Unfortunately, there is no guarantee that the module position previously determined is optimal for the new circuit structure. Therefore, repositioning, resynthesis, and resizing are iterated. With the circuit model and transformations presented in the following sections, we can claim an iterative improvement of the circuit performance. In practice, the number of iterations can be limited, because only a marginal gain is achieved after the first few iterations and to bound computation time.

It is a complicated problem to find optimal tradeoff points among the circuit figures of merit by using the techniques outlined above for general circuits. We do not attempt a complexity analysis of the problem, but rather concentrate on heuristic methods. The computer programs implementing the heuristic algorithms for resynthesis, resizing, and repositioning have been shown to yield good designs in most cases.

We present first, in Sections II and III, the domino gate model and the circuit model, respectively. In Section III, we show how critical path delays are estimated. Then, the resynthesis, resizing, and repositioning algorithms are presented in Sections IV, V, and VI, respectively. Eventually, in Section VII we describe the algorithm implementations and their interface to the YSC system, and we present some experimental results.

II. Domino Gate Model

In this paper we consider here only one family of circuits, in particular, dynamic CMOS circuits operating in the domino mode [20], [19]. Combinational circuits in this family are implemented by an interconnection of multiple-input, single-output gates. The gates consist of i) a series/parallel connection of transistors which provides a

discharging path to ground from a precharged node; ii) a **driver,** implemented by a static CMOS inverter; and iii) additional circuitry (some of which may be optional, e.g., a bleeder device [11]) that is irrelevant to the topics dealt with in this paper. An example of a domino gate is shown in Fig. 1. Each gate implements a Boolean function represented by a suitable[1] factored algebraic expression that has a straightforward mapping into the series/parallel transistor connection personalizing the gate. The output drive capability of the gate is determined by the physical dimensions of the devices implementing the driver. Each gate drives a capacitive load, consisting of the input capacitance of the following stages and of the parasitic capacitance to ground of the wires. The physical width of the two transistors implementing the driver (referred to as driver **size** in short) is a design parameter. We denote by w a parameter that is proportional to the gate width and therefore directly affects the gate output drive capability and the gate switching time. We assume that the physical length of the two transistors implementing the driver is kept constant. The physical dimensions of the transistors in the discharging path are not a design parameter, because the capacitance of the sense (precharged) node is small compared to the gate output capacitance and affects the gate switching time by a negligible amount.

We consider in this paper circuit timing considerations that are related only to the *evaluation* phase of the domino cycle; i.e., we consider only the signal propagation through the circuit and we assume that the timing of the *precharge* phase is correct. Since we address here the delay evaluation and optimization of large-scale circuits, we have chosen a delay model which can be computed quickly and retains a reasonable accuracy. The **propagation delay** through a physical gate is modeled by an empirical delay equation. An empirical delay equation for a logic family, in particular dynamic domino CMOS gates, can be obtained by i) determining a set of characteristic parameters; ii) simulating the transient behavior of a large set of gates with different values of the parameters using a circuit simulator such as ASTAP or SPICE to compute the propagation delay; and iii) using regression analysis to tabulate the delays as a function of the characteristic parameters.

In the case of dynamic domino CMOS gates, the output transitions are always low-to-high. Interesting parameters that affect the gate propagation delay are i) the size w of the drivers; ii) the capacitive loading c at the output, which in turn depends on the fan-out of the gate and the wiring capacitance to ground; iii) the structure of the series/parallel transistor connection forming the discharging path of the sense node (in particular the maximum number of devices in series in discharging path l and the worst-case

[1]For some gate implementations, such as linear MOS transistor arrays used in the YSC system, the series/parallel graph, abstracting the interconnection, is required to be covered by an Euler path joining any two nodes and to be bounded in the number of series edges. In these cases, we assume that an algebraic expression in transformed into a form suitable for implementation, i.e., the corresponding graph has the required properties.

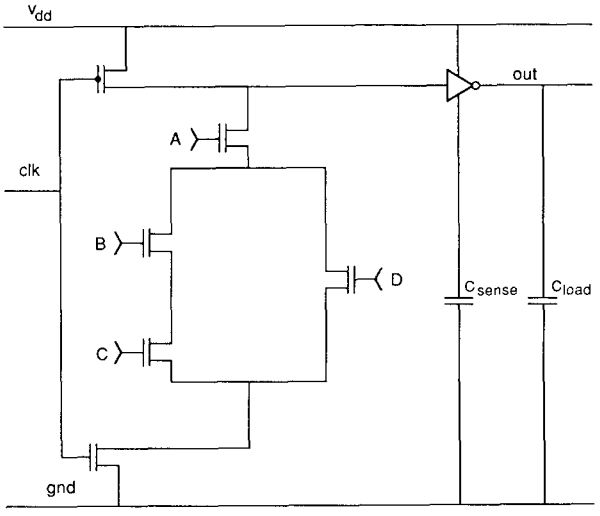

Fig. 1. Example of domino CMOS gate implementing $ABC + AD$.

charge redistribution factor); and iv) the parameters of the input signals, such as rise time and relative arrival times. Experimental results on gates with typical parameter values have shown a negligible dependency of the switching time as a function of worse-case charge redistribution and input signal parameters. Therefore, the gate delay has been tabulated as a function of the parameters (w, c, l). A convenient expression of the delay is

$$d(w, c, l) = \alpha(w) + \beta(w)c + \gamma(w)l.$$

The coefficients α, β, and γ are tabulated for a finite number of values of w by using the method outlined above. It is practical for automatic synthesis reasons to limit the choice of the driver device widths to a finite number. Therefore we assume that the parameter w can take a finite set of values $1, 2, \cdots, p$. In our delay model, the functions $\beta(w)$ and $\gamma(w)$ are monotonically decreasing with w, while $\alpha(w)$ is monotonically increasing with w. This is consistent with the fact that the current flowing through the MOS transistors forming the driver is directly proportional to the gate width (the higher the current, the lower the propagation delay) and that the driver gate capacitance increases as the size increases (the higher the gate capacitance, the higher the propagation delay). As a net result, for typical values of c and l, the propagation delay is monotonically decreasing with w.

The energy required by a gate in a domino cycle has three components: the first two are related to charging and discharging the sense and load nodes, and the third one is related to the energy dissipation within the driver. The last two terms are the most significant ones; and power dissipation can be modeled as $p = p(w, c)$, where the function p is increasing with both w and c. Power dissipation can be tabulated by a procedure similar to that used for the propagation delay.

The expression for silicon area occupied by a gate has also three terms, related to the discharging path, the driver, and the additional circuitry. The first term is proportional to the number of devices in the discharging path,

which is equal to the number of variables (counting repetitions)in a factored algebraic expression describing the gate. We denote this parameter by n. The second term is directly proportional to w, and the third term is a fixed overhead. In summary, the area taken by a gate can be expressed as $a = a(n, w)$. For some gate implementations, the dependence on w can be neglected and included in the overhead.

The critical reader may question the validity of this delay model that we assume as a basis of our optimization procedure. The characterization of the propagation delay by simulation and regression analysis has been carried out in the case of domino CMOS gates in a proprietary technology [12], as used by the YSC system. Therefore, the delay coefficients, as well as the domino gate parameters, are proprietary information. Nevertheless, it can be said that the computed gate delays are a conservative estimate of the simulated gate delays and that the error is at most 10 percent of the simulated delay value. We believe that our delay model, though simple, is very practical because it may be tuned to different technologies and because delays can be evaluated quickly as needed in the case of large-scale circuits.

III. Circuit Delay Model

We consider here the problem of optimizing the cycle time of synchronous sequential circuits. We assume that sequential circuits are implemented by connecting combinational circuits to synchronous registers. We assume also that we are given a global circuit structure in terms of an interconnection among input/output ports, a combinational circuit, and a set of registers and that this structure cannot be modified. Therefore the circuit cycle time is bounded from below by the speed of its combinational component. For this reason, we concentrate on the optimization of the timing performance of combinational circuits and we consider each signal stored in a register as both a primary input and output to the combinational subsystem. We assume that each primary input signal is available in both phases.[2]

We assume that combinational logic is implemented in a general multiple-level form. An abstraction, at the logic description level, of a digital circuit is a **Boolean network.** A Boolean network is described in terms of **Boolean variables** and **Boolean functions** that specify the value of each variable in terms of other variables; i.e., each function has multiple inputs and a single output. We assume that the Boolean network is **unidirectional,** i.e., that there are no cyclic dependences among the Boolean functions. With this assumption, we avoid the possibility of race conditions and we restrict Boolean networks to the domain of combinational circuits.

A Boolean network can be implemented by an interconnection of **logic gates:** each gate is the implementation of a Boolean function, and the interconnection is achieved by a set of **nets** carrying electrical **signals.** A network has a set of primary inputs and outputs, which are also interconnected to the logic gates by means of nets. Each logic gate is assumed to be unidirectional. Therefore, we can associate a source and one (ore more) sink gates to each net corresponding to the signal propagation direction along that net. The network can be modeled by a directed graph $G(V, A)$, whose node set $V = \{v\} = V^g \cup V^i \cup V^o = \{v^g\} \cup \{v^i\} \cup \{v^o\}$ is in one-to-one correspondence with the set of logic gates (V^g), primary inputs (V^i), and primary outputs (V^o). The ege set A is in one-to-one correspondence with the source–sink pairs of the nets. The graph $G(V, A)$ is acyclic, because the network is unidirectional. A node v_i is said to be a **predecessor (successor)** of node v_j if there is a directed path from v_i to v_j (from v_j to v_i) in $G(V, A)$. A predecessor (successor) is said to be **direct** if the path has length one.

Each node of the graph corresponding to a gate has a set of attributes. An **algebraic expression** is the attribute that specifies the Boolean function. An algebraic expression relates the algebraic input variables by means of the product and sum operators. Any level of parenthesis can be used. The input algebraic variables are the Boolean variables and their complements, i.e., Boolean **literals.** The value of an algebraic expression corresponds to the value of the corresponding Boolean variable with the positive phase. Each algebraic expression corresponds to a discharging path of the corresponding domino gate. Since domino circuits do not provide logic inversion, no complementation operator is permitted in the algebraic expressions. Output variables with negative phase (i.e., complemented) are specified by the algebraic expression of the complement of the Boolean function. To make this possible, we have assumed that each primary input is available as well as its complement.

We abstract a gate as a point on the chip surface. The x, y coordinates are other attributes of a node. The length of a net connected to a gate output can be estimated from a knowledge of the x, y attributes of a node and its direct successors (net end points). In particular, we use as a worst-case approximation the half-perimeter of the smallest rectangle containing the net end points. The gate capacitive loading (node attribute c) is estimated as a weighted sum of the output net length and of the gate fan-out (which depends on the number of transistor gates connected to the net). Assuming an average number of transistor gates per input of a logic gate, the gate fan-out may be approximated by the node out-degree. Other attributes of a node are the driver size (w), the maximum number of devices in series (l), and the total number of devices (n). The last two attributes depend on the algebraic expression. These parameters determine other node attributes, such as the gate propagation delay, power, and silicon area consumption, as shown in the previous section.

The attributes of the node are figures of merit of the corresponding gate. Additive figures of merit of the network, such as power and area, can be computed by sum-

[2]Registers have complementary outputs, and both output phases are available simultaneously. Input ports with negative (or both phases) are assumed to be available. The delay through the corresponding circuit may be taken into account by the input signal arrival time.

ming the corresponding attributes of the nodes. (In the case of area, the routing space must be added too). The switching-time performance of a Boolean network is evaluated by attributing to each node a **data ready** time $t(v_i)$, $i = 1, 2, \cdots, |V|$. The data ready time of a node is the time at which the signal generated by the corresponding gate (or input) is ready. For our purposes, we synchronize the computation of the data ready times to the system clock. Therefore, we assume the data ready time to be zero for each primary input corresponding to a register. The data ready times of the remaining inputs are set to the delay of the corresponding input signal with regard to the system clock. The data ready times can be computed by tracing forward the signal propagation, i.e., by computing

$$t(v_i) = d(v_i) + \max_{k \in K} t(v_k) \qquad K = \{k \text{ s.t. } (v_k, v_i) \in A\}$$

for each node v_i in a sequence consistent with the partial order represented by graph $G(V, A)$. Here, the propagation delay through node v_i is denoted by $d(v_i)$. The propagation delay of the nodes corresponding to gates ($v_i \in V^g$) is computed as shown in Section II; the propagation delay of the other nodes ($v_i \in V^i \cup V^o$) is zero.

Another node attribute to the node **slack.** Let $\hat{T} = \{\hat{t}(v) \text{ s.t. } v \in V^o\}$ be a set of required arrival times for the primary circuit outputs (e.g., the minimum system cycle time). The slack of each primary output node ($v_i \in V^o$) is defined as the difference between the required time and the computed data ready time:

$$s(v_i, \hat{T}) \equiv \hat{t}(v_i) - t(v_i).$$

The slack at the other nodes ($v_i \in V^i \cup V^g$) is defined to be

$$s(v_i, \hat{T}) = \min_{j \in J} \{s(v_j, \hat{T}) + \max_{k \in K} t(v_k) - t(v_i)\}$$

where

$$J = \{j \text{ s.t. } (v_i, v_j) \in A\} \qquad K = \{k \text{ s.t. } (v_k, v_j) \in A\}.$$

The slack at an internal node measures how much additional delay the corresponding signal may tolerate, while the relation $t(v_i) \leq \hat{t}(v_i)$ is satisfied for each node $v_i \in V^o$. The slacks are denoted in short by $s(v)$, by assuming as implicit their dependence on $\hat{T}$. The slacks can be computed by tracing the signal propagation backward in the circuit, i.e., by computing the slack at each node in a reverse sequence consistent with the partial order represented by graph $G(V, A)$.

Let ϵ be an arbitrary constant. The set of **critical nodes** $C(\epsilon, \hat{T}) \subseteq V$ is the set of nodes $C(\epsilon, \hat{T}) = \{v \text{ s.t. } s(v, \hat{T}) \leq \epsilon\}$. The set of critical nodes is denoted in short by C. The **critical graph** $H(C, B)$ is the subgraph of G induced by C. Note that in general $H(C, B)$ is not a connected graph. A **critical section** (of the critical graph) $S \subseteq C \cap V^g$ is a maximal subset of critical nodes corresponding to gates which are not mutually reachable (i.e., not connected by a directed path). A critical section is a node separating set when $H(C, B)$ is a connected graph. A **critical path** $P \subseteq C$ is a maximal directed path in $H(C, B)$.

The critical graph can be used in different contexts according to the attributes of the problem we want to solve, which can be specified by the choice of ϵ and $\hat{T}$. Suppose we require that the output data ready times of a circuit satisfy a set of upper bounds, represented by $\hat{T}$, and that the parameter ϵ takes into account safety margins and tolerances. Then, the circuit meets the timing specification if and only if the critical graph is empty. If not, the data ready time of the critical nodes must be reduced by a timing optimization procedure to meet the timing specifications. Reducing the data ready time at the noncritical nodes does not help in meeting the circuit timing specification.

It is common in circuit design that the output nodes of interest represent the input to synchronous registers. A design strategy may try to minimize the data ready times at these nodes and then adjust the system clock accordingly. Suppose no upper bound $\hat{T}$ is specified. In this case, the critical nodes and graph may be used to detect those gates that limit the timing performance of a circuit. Let t^* be the largest data ready time in the circuit, i.e., $t^* = \max_{v \in V} t(v)$. The system clock period is bounded from below by t^*, and timing performance optimization aims at reducing t^*. For this problem, we set $\hat{T} = \hat{T}^*$ where $\hat{T}^*$ is a set whose elements are all t^*. Then the set of critical nodes $C(0, \hat{T}^*)$ has the following obvious properties: i) the corresponding critical graph is connected; ii) any critical section is a node separating set for the graph and intersects any critical path (in particular, a critical graph may be a simple path and a critical section just a node); and iii) any independent variation in the data ready time of any critical section implies a varition of t^*. Therefore the nodes (gates) along the critical graph are "critical" because the timing performance of the circuit (limited by t^*) can be improved by decreasing the data ready time at the critical nodes. This can be done, for example, by reducing the propagation delays at a critical section.

Output timing constraints can be considered in conjunction with the problem of minimizing the maximum data ready time by setting the elements of $\hat{T}$ to the minimum of t^* and the corresponding data ready upper bound. In the sequel, we will refer to the problem of minimizing t^* as the timing optimization problem. The value of t^* can be regarded as a figure of merit of the circuit.

It is important to remark that a variation of the data ready time of a critical node v_i may influence t^* by very little if the direct successor of that node has a noncritical direct predecessor v_j such that $|t(v_i) - t(v_j)|$ is a small quantity. In other words, node v_j is "almost critical." Considering "almost critical" nodes is important in a timing optimization procedure, as far as computational efficiency is concerned. For this reason, "almost critical" nodes are made "critical" by choosing $\epsilon \geq 0$. The choice of positive values for parameter ϵ widens the set of critical nodes.

From the model of the gates and of the circuit, it is clear that the timing performance can be improved by changing the structure of the graph $G(V, A)$ and/or the node attributes. Circuit resynthesis operates on the structure of the graph, device resizing on the set of sizes w, and repositioning on the coordinate attributes x, y. In all cases, we assume the existence of an initial circuit structure and values of the node attributes. Though the structure and the attributes may be arbitrary, we assume that the circuit has been synthesized with the goal of optimizing a given figure of merit. In particular, in the YSC system, the graph $G(V, A)$ is constructed with the goal of minimizing the estimated circuit area. The attributes w take minimal values to optimize the worst-case power dissipation and area. The coordinates x, y are determined with the objective of minimizing the total wiring length, which correlates positive with minimal values of power (because it minimizes the total capacitance Σc) and area (because it reduces the wiring space).

A general algorithm for timing optimization (minimizing t^*) and for exploring the area/power/timing tradeoffs has the following frame:

Algorithm Model

data $(\epsilon, \hat{T})$;
do{
    $C =$ **critical** $(\epsilon, \hat{T})$;
    $S =$ **select** (C);
    if $(S = \phi)$ **stop;**
    transform (S);
}

The algorithm iterates the steps: i) compute the critical nodes C; ii) select a section S of the critical graph; iii) perform a transformation that modifies the attributes of the nodes in S and possibly those of its predecessors and successors. The transformation may change locally the structure of $G(V, A)$. The three procedures, **critical**, **select**, and **transform**, depend on the particular technique used and are detailed in the following sections. The transformations of the attributes of a node may affect the node propagation delay and that of its predecessors (e.g., a change of the out-degree of a direct predecessor, which affects the parameter c). As a result, in general a transformation may affect the data ready time of other nodes in the graph and their being critical or not. For this reason, a transformation may or may not be useful to improve the circuit switching-time performance according to the context. A transformation of node v_i is said to be **favorable** if in the modified circuit i) the data ready time at node v_i decreases; and ii) any increase of the data ready time at any other node is less than the corresponding slack. To be more specific, let $t(v_i)$ and $t'(v_i)$ denote the data ready time before and after the transformation at node v_i and let $\Delta u(v_i) \equiv t'(v_i) - t(v_i) - s(v_i)$. A transformation of node v_i is favorable if

$$\Delta u(v_i) + s(v_i) < 0$$

$$\Delta u(v_j) \leq 0 \qquad \text{for each } j \neq 1.$$

For a given transformation at a given node, the set of values $\Delta u(v_i)$, $i = 1, 2, \cdots, |V^g|$, gives a measure to evaluate the desirability of the transformation. Lowering the entries of $\Delta u(v)$ corresponds to increasing the slacks at the nodes and moving them far from their region of criticality. Therefore, transformations are graded on the basis of the values of $\Delta u(v)$ and the variation in area Δa and in power Δp.

In general, a transformation may be a single modification or a set of modifications of the attributes of its target nodes. In the second case, a transformation may still be favorable even though some of the corresponding circuit modifications may be uphill moves. The algorithm model guarantees a descent of the maximum data ready time t^* when all the transformations are favorable. We use this property to combine different types of transformations (e.g., resynthesis and repositioning) to minimize t^*. While in principle each call to procedure **transform** may invoke a different type of transformation (e.g., the "local best" among resynthesis, resizing, and repositioning), an efficient program implementation requires alternating sequences of transformations of a given type. In practice, circuit resynthesis, resizing, and repositioning (i.e., sequences of transformations affecting the structure of $G(V, A)$, the attributes w and x, y) are iterated.

IV. Circuit Resynthesis

We consider in this section the problem of improving the circuit performance by changing the gate interconnection. According to our model, we attempt to optimize the circuit by modifying the graph $G(V, A)$ by adding/deleting edges/nodes and/or changing the node attributes (e.g., the algebraic expressions). Note that these changes may affect both the parameters $\{c\}$ and $\{l\}$ at some nodes, because of the variation of the corresponding node out-degree and in-degree. We assume in this section that the timing performance of the network is not limited by the driver size (parameter w). Indeed, we assume that all the driver sizes are maximal (i.e., $w = p$) while searching for an optimal structure of the graph $G(V, A)$. Resizing, described in the next section, is then applied to select the optimal size for the drivers.

A. Circuit Transformations

Circuit resynthesis is achieved by a sequence of transformations. Circuit transformations for synthesis have been presented elsewhere [3], [7], [8]. We report here only on the basic transformations that are used to improve timing performance and the relations among these transformations and the timing/area figures of merit. We refer to the reader to [3], [7], and [8] for the details of the algorithms. We describe three basic transformations: **elimination, decomposition** and **simplification.** We consider the first two transformations in their algebraic flavor [7] for the sake of simplicity.

1) Elimination: The elimination of a node, say v_j, into a direct successor node, say v_i, is the substitution of the algebraic expression attributed to v_j into the one of v_i. The

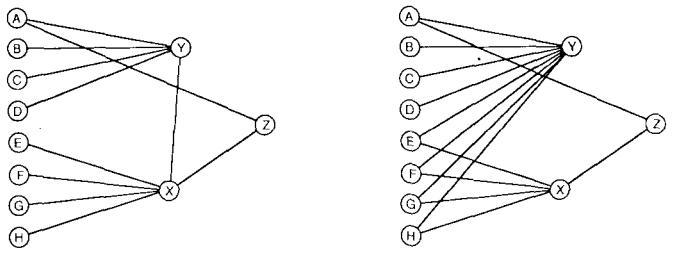

Fig. 2. Network graph before and after elimination.

edge (v_j, v_i) is removed from the edge set A; the edges (v_k, v_i) are added to A when v_k is a direct predecessor of v_j and not of v_i before the transformation. A node $v \in V^g$ with no successors can be deleted from the graph.

Example 1: Let us consider the graph of Fig. 2(a). Let

$$X = (E + F)G + H$$

$$Y = (A + B)X + CD$$

$$Z = AX.$$

Then the algebraic expression for X can be substituted into the one for Y, yielding $Y = (A + B)((E + F)G + H) + CD$. The corresponding graph is shown in Fig. 2(b). ■

By carrying on node elimination, the network will eventually collapse into one having a single gate for each primary output. In principle, this can always be achieved since each primary output function can be expressed in a *sum of product* form. Moreover, the network has a minimal number of nodes: $|V^i| + 2 \times |V^o|$ because $|V^g| = |V^o|$. (We assume that no two primary output functions coincide.) Collapsing a network into a minimal number of gates may not be practical for some circuits, due to the explosion in complexity of the corresponding algebraic expression.

2) Decomposition: We call here decomposition the transformation opposite to elimination. In its general form, an algebraic expression f is replaced by $g \cdot h + r$, where g, h, and r are also algebraic expressions, called subexpressions of f. Since we consider here algebraic transformations, the subexpressions g and h are required not to share variables that are identical or represent Boolean complements. One, two, or all three subexpressions g, h, and r may be replaced by intermediate variables: if a node has an algebraic expression attribute equivalent to the subexpression, its output variable replaces the subexpression; otherwise, a new node is added to the graph (with default values for the attributes x, y, and w). The edges incident to the node which is the object of the decomposition are modified according to the dependency on the input variables. Note that decomposition may have several flavors, and different algorithms may be used to achieve different goals. In particular, the literature [3], [6]–[8] refers to "resubstitution" for the transformation in which g is replaced by an existing variable and "extraction" for the addition of a new node implementing a subexpression common to the expression attributes of two nodes. Decomposition is referred to in the literature as the transformation used to break down an expression into smaller subexpressions to satisfy technology requirements. In particular, note that g or h may be a Boolean TRUE (and therefore the decomposition yields the sum of two intermediate subexpressions) or r may be a Boolean FALSE (and therefore the decomposition yields the product of two intermediate subexpressions).

Example 2: Let us consider the graph of Fig. 3(a). Let

$$X = A + PQR$$

$$Y = AC + AD + PQRC + PQRD + EF$$

$$Z = CX.$$

Then the algebraic expression for Y can be decomposed as: $Y = (A + PQR)(C + D) + EF$ and the first factor can be substituted by X, i.e., $Y = X(C + D) + EF$. The corresponding graph is shown in Fig. 3(b). ■

By carrying on decomposition, the network can be decomposed into an interconnection of nodes implementing two-input OR's and AND's. By combining elimination and decomposition, a network may be transformed into an interconnection of nodes implementing arbitrary expressions (e.g., gates from a given library).

3) Simplification: The algebraic expression of a node is replaced by an equivalent one having a simpler form. The goal of this transformation is to minimize area, represented by the parameter n.[3] This transformation does not change the number of nodes in the graph representation. However, some edges incident to the node may be added and/or deleted as a consequence of a change in the set of support variables.

Example 3: Let us consider the graph of Fig. 4(a). Let

$$X = A\bar{B} + BC$$

$$Y = AD$$

$$Z = X + \bar{C}Y.$$

Then a simplification at node v_z leads to $Z = X + Y$ and to the graph of Fig. 3(b).[4] (See [7], [8] for details.) ■

Gate transformations change the attributes (l, c, n) of one or more nodes and as a consequence change the circuit figures of merit representing switching time, area, and power. As far as timing is concerned, a node elimination (decomposition) corresponds to reducing (increasing) the number of stages needed to implement a primary output of the circuit. However, by node elimination (decomposition), the attribute l, corresponding to the modified expression, may increase (decrease) as well as the node propagation delay. The data ready time at the node and at its successors is affected by the transformation. Moreover, the capacitive load c of some direct predecessors of

[3]Different objectives may be used for simplification [7], [8]. We consider here the minimization of algebraic support variables, which represent Boolean literals. Minimization of Boolean support variables is also possible. Minimizing the support variables correlates to reducing n.

[4]Let dc be the don't care set. Then, $X = A\bar{B} + BC \rightarrow dc \supset \bar{X}(A\bar{B} + BC) \rightarrow dc \supset \bar{X}ABCD + \bar{X}A\bar{B}CD = \bar{X}CAD = \bar{X}CY$. Then, $Z = X + \bar{C}Y = X + \bar{X}\bar{C}Y + \bar{X}CY = X + \bar{X}Y = X + Y$.

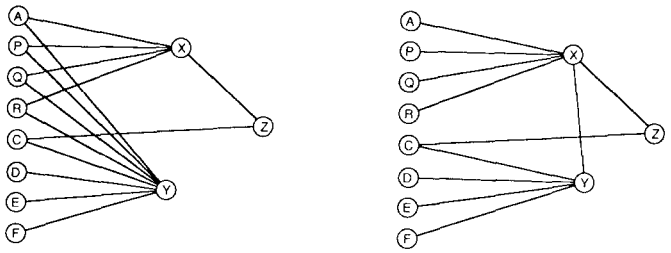

Fig. 3. Network graph before and after decomposition (resubstitution).

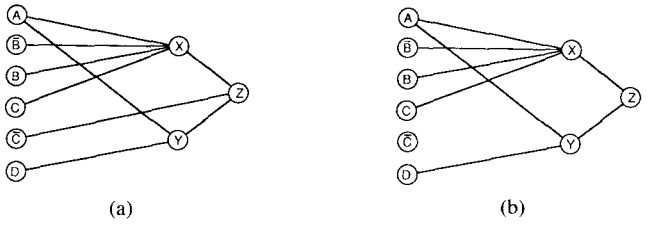

Fig. 4. Network graph before and after simplification.

the transformed node may vary, as well as their propagation delay. For example, a node elimination without a deletion of the eliminated node may lead to an increase in the out-degree of some direct predecessors of the node. As a by-product of the transformation, the data ready time at some other circuit outputs may be affected. Similar considerations apply to node simplification. In particular, the parameter l may vary as well as the parameter c of the nodes that are direct predecessors of the node under consideration before and after the simplification.

In summary, a node transformation may be found to be favorable or not. This is the criterion for measuring the usefulness of a transformation as far as timing is concerned. The values of $\Delta u(v)$ can be used to give a quantitative grade to the transformation.

Example 4: Consider the elimination of Example 1. Suppose $t(v_h)$ and $t(v_x)$ are larger than the data ready time of the other inputs. Then, $t(v_y) = t(v_h) + d(v_x) + d(v_y)$ before the elimination. After the elimination, $t'(v_y) = t'(v_h) + d'(v_y)$, where $t'(v_h) \geq t(v_h)$ because the parameter c of v_h increased due to larger out-degree and $d'(v_y) > d(v_y)$ because the parameter l of v_y increased due to the variable elimination. However, the propagation delay through v_x is eliminated. It is possible that $t'(v_y) < t(v_y)$ according to the actual values of the parameters. This is not the only condition for a favorable transformation. Since $t'(v_h) > t(v_h)$, then $t'(v_x) > t(v_x)\, t(v_a)$ and $t'(v_z) > t(v_z)$. To be a favorable transformation, we also need $t'(v_z) - t(v_z) \leq s(v_z)$. ■

Example 5: Consider the decomposition of Example 2. Suppose $t(v_f)$ is larger than the data ready time of the other inputs and larger than $t(v_x)$. Then, $t(v_y) = t(v_f) + d(v_y)$ before the decomposition. After the decomposition, if $t(v_f) > t'(v_x)$ holds, then $t'(v_y) = t(v_f) + d'(v_y)$. Note that the parameter l of v_y decreases, $d'(v_y) < d(v_y)$, and $t'(v_y) < t(v_y)$. The other condition to have a favorable transformation is that $t'(v_z) - t(v_z) \leq s(v_z)$. Note that $d(v_x)$ will increase due to the larger out-degree and, as a consequence, $t(v_z)$ may also increase. ■

The silicon area is affected by the circuit transformations. Let us consider first simplification. This transformation targets the reduction of the parameter n. Therefore, simplification may only be beneficial as far as silicon area is concerned.

The elimination of node v_j into v_i affects the area of gate v_i, which increases, because its total number of variables (parameter n) increases. The total silicon area of the circuit may increase or not, according to the circumstances. For example, an elimination leads to a savings of silicon area when node v_j is deleted from the circuit and its corresponding variable occurs only once in the algebraic expression for v_i. In this case, the number of variables in the expression for v_i after the transformation is equal to the sum of the number of variables in v_i and v_j minus one. Moreover, we are saving the area overhead of the node being deleted. In general, a node elimination opens a potential simplification of the corresponding algebraic expression. For this reason, the effectiveness of node elimination and simplification should be evaluated together. We refer the interested reader to [7] for details.

A gate decomposition may or may not be advantageous in terms of the total silicon area. Clearly, a decomposition is a loss of area when a new node is introduced in the graph to implement one subexpression; this node has only one direct successor and the variable related to the new node is used only once in the algebraic expression of its direct successor. In this case, the silicon area increase comes from the overhead of the additional gate and the additional variable introduced. On the other hand, resubstitution and extraction may be used efficiently to save silicon area. Algorithms have been devised to detect when these transformations can be used to reduce the area penalty. We refer the interested reader to [7] for details.

Finally, these circuit transformations affect the power consumption by changing the capacitive load (c) by varying the node out-degree. Changes may be positive or negative and must be computed along with the transformation.

B. Resynthesis Algorithms

Circuit resynthesis consists of applying a sequence of transformations to the original circuit description. Though the starting point may be arbitrary, we consider in this section methods for devising area/speed tradeoffs. Therefore we assume that the circuit structure used as a starting point has been optimized as far as area is concerned. Circuit transformation modify this structure by trading off silicon area for improved switching-time performances.

We consider different strategies for resynthesis. They can be classified as global or local. Global strategies apply a sequence of transformations to nodes of the graph of the entire combinational circuit. Local strategies assume a partition of the circuit into blocks; transformations

are applied to the nodes of the subgraph representing each block of the partition one at a time. Clearly, global strategies can achieve results that are at least as good as those obtained by applying local strategies. However, the computational cost of algorithms grows with the node set cardinality, and it may be impractical to apply a global strategy to a large graph. (For example, we consider "large" the graph representing all the combinational logic of a 32-bit microprocessor.) Local strategies exploit the *divide and conquer* paradigm. Local strategies yield good results if the circuit partition is appropriate. Since in this case circuit transformations do not consider the interactions among nodes in different blocks of the partition, a measure of the appropriateness of a circuit partition is the separation of independent or loosely coupled circuit blocks. In the actual implementation of these algorithms, performance optimization has been coupled to the YSC synthesis system. A circuit partition is achieved at the level of the structural synthesis of a circuit, and this partition has been shown to be adequate for our purposes.

We consider first an algorithm with a global strategy. The algorithm is based on the model presented in Section III and iterates three steps: i) compute the set of critical nodes; ii) determine a particular critical section; if none is found, stop; and iii) apply a transformation to the nodes in the section. Procedure **critical** computes the propagation delays, the data ready times, and the slacks, as shown in Section III. Then it determines the set of critical nodes, as a function of the parameter ϵ and $\hat{T}$. The parameter ϵ can be set to a positive value (usually a fraction of t^*) to widen the set of the critical nodes. This is done for computational efficiency, to avoid fluctuations in the critical node set which may add/drop iteratively group of nodes with similar interconnections and similar timing behavior. (This effect may be common in circuits with wide bus structures, e.g. 32-bit processors.) Procedure **select** returns a critical section and the transformation to be applied to each node in the section. The selection is based on grading each transformation for each node by a weighted sum of the elements of $\Delta u(v)$ and the variation in area Δa. The selected section must satisfy two requirements: i) include a critical section of $H(C(0, \hat{T}), B)$; and ii) the transformations chosen for the nodes in $H(C(0, \hat{T}), B)$ are favorable. In the case where no section satisfying the above requirements exists or the set of critical nodes is empty, the procedure returns an empty set ϕ. Procedure *transform* applies the chosen transformations to the chosen nodes in subset S.

The algorithm has the property that at each iteration there exists a subset of critical nodes, which includes $C(0,\hat{T})$, whose data ready time is decreased. As a consequence, the data ready time at some critical outputs decreases as well; in particular, the maximum data ready time t^* decreases. The algorithm is fairly general, and procedure **select** may choose among a wide set of transformations. A specific example of a resynthesis algorithm with global strategy was detailed in [5]. A fixed stage delay model was assumed (i.e., $d(v) = 1$). Procedure **transform** used a set of eliminations first, followed by other transformations. Elimination was used to reduce the number of stages and was always favorable, because of the delay model. An area/delay tradeoff was achieved by associating a weight to each node in the graph corresponding to the increase in silicon area Δa due to the elimination. Procedure **select** returned a minimum weight section. A similar resynthesis technique was used in the system described in [6].

Resynthesis with global strategies may be inefficient for circuits described by large graphs. In this case, due to the large number of possible transformations for each node, time-effective implementations of procedure **select** must include heuristics to limit the search of candidate sections and transformations. This limitation has been found to affect the quality of the results. On the other hand, resynthesis with a local strategy applies transformations to smaller graphs. A more careful search for circuit transformations is then possible.

We consider now an algorithm with a local strategy, as was finally implemented. The circuit is partitioned into blocks called **modules.** Each module is described by a graph $G^M(V^M, A^M)$, which is obtained by adding to the subgraph of $G(V, A)$ induced by the nodes corresponding to the gates in the partition block, the set of the corresponding input and output nodes, and the related edges. Each module is considered along with its boundary conditions, which are the data ready times at the module inputs and the slacks and external capacitive load at its outputs. The interconnections among modules are represented by graph $G^I(V^I, A^I)$, which is obtained from $G(V, A)$ by merging into a single node all the nodes in each circuit block. Since the partition may be arbitrary, we assume that this graph is acyclic. (The YSC system constructs an acyclic graph $G^I(V^I, A^I)$.) A module is said to be critical if its graph $G^M(V^M, A^M)$ has at least one critical node. A module critical section in $G^I(V^I, A^I)$ is a maximal subset of critical modules which are not mutually reachable.

The main algorithm, based on the local strategy, may be described in terms of the algorithm model (Section III) as an iteration among the steps: i) compute the set of critical modules; ii) determine a particular subset of critical modules; if none is found, stop; and iii) apply resynthesis to the chosen modules. In this case, procedure **critical** computes the set of critical nodes and returns the set of critical modules. Procedure **select** determines the set of critical modules to be resynthesized by procedure **transform**. By analogy with the previous algorithm, procedure **select** may return a critical section of $G^I(V^I, A^I)$. The size (i.e., cardinality) of this section reflects some tradeoffs in the circuit partition. Fine-grain partitions correlate to small modules and large section sizes, while coarse partitions lead to large modules and small section sizes, often just one module. The experience of running the resynthesis algorithms with different environments has shown that procedure **transform** cannot achieve its goal if the modules are too small, because of a limited search transformation space. Therefore, a coarse partition has been

used. For implementation reasons, it is sometimes convenient to resynthesize several modules at a time. Therefore, procedure **select** has a switch that makes it possible, if requested, to return a subset of critical modules which includes a section and other modules. In this case, the modules are sorted according to the partial order represented by the subgraph induced by the selected set. The rationale is not to change the input data ready times of a module after its resynthesis, since the circuit structure has been tuned to this particular input signal arrival-time distribution.

Procedure **transform** performs the resynthesis of a module with the goal of reducing the data ready times at its critical nodes. Note that since the module is "extracted" from the whole circuit, the slacks cannot be recomputed after each transformation. Therefore, the slacks are computed only once, at the beginning. The corresponding critical nodes are labeled. Circuit transformations are applied to reduce the data ready times at these nodes. In the end, the set of transformation is tested for being favorable using the initial values of the slacks.

Circuits transformations can be applied in different sequences and they can be specified by a "program." We have experimented with different programs and we have found that resynthesis can be effectively done by stepping through these tasks: i) collapse (partially) the network by using node elimination; ii) apply simplification to the collapsed network; iii) extract common subexpressions and introduce new nodes; iv) perform resubstitution; and v) adjust the gate and network structure according to the target technology requirements. Procedure **transform** can be synthetically described by this program:

critical; collapse; simplify; distill; resub; design.

Procedure **critical** computes the data ready times and slacks and detects the critical nodes. These nodes are labeled and they are the target of the transformations. Procedure **collapse** detects first the pairs of critical nodes and critical direct predecessors. These pairs are sorted by choosing first the nodes with lowest slack. In case of tie, pairs with the lowest total slack are chosen first. Node elimination is then tried on all pairs in the computed sequence. An elimination is performed if favorable. Procedure **collapse** attempts then to delete some of the nodes that have been eliminated into their direct successors, to balance the possible increase in silicon area. To do this, these nodes are eliminated into their direct successors, even if these are noncritical and even if their data ready time increases. The criterion to perform these eliminations is based on the value of $\Delta u(v)$ and Δa. In particular, all the entries in $\Delta u(v)$ must be negative (any possible increase in the data ready time is bounded by the slack) and there should be a saving in area. Procedure *simplify* is applied after the node elimination to reduce the total number of variables n in the algebraic expressions and therefore save area.

Procedure **distill** extracts the algebraic subexpressions that are common to pairs of nodes, one of which is critical. Then, a node is introduced into the network if the decomposition of the expression at the critical node (nodes) is favorable. Procedure **resub** attempts to use node decomposition differently. For each critical node, we search for algebraic divisors of the corresponding expression among the expression of the other nodes. Only favorable transformations are performed.

Procedure **design** factors the node algebraic expressions and makes sure that each signal is generated with the positive phase and that the parameter l is lower than a given threshold if required by the implementation technology. This procedure is a set of standard tasks in the YLE program [7].

V. Device Resizing

Before introducing the device sizing strategy, it is important to consider the particular delay model for domino CMOS circuits and how the device sizing affects the propagation delay. As pointed out in Section II, the propagation delay $d(w,c,l)$ of a gate is monotonically decreasing with w. The variation of the propagation delay as a result of a variation in the driver size is denoted by $\Delta^m d(v) \equiv d(w+m,c,l) - d(w,c,l)$. Then $\Delta^m d(v)$ is always negative for each node v and $m > 0$. Note that the domino gate input capacitance does not change with w, because only the driver size changes. Therefore an increase in the parameter w is always beneficial in reducing the node propagation delay and does not increase the data ready time of any other node; i.e., the gate resizing transformation is always favorable.

With this model, it is clear that a minimum value of t^* can be achieved by selecting the maximal value for w (i.e., $w = p$) at each node. However, the drawback of this trivial size assignment is the cost in silicon area and power consumption, which are both nondecreasing functions of the parameter w. Conversely, assigning a minimal value to the size w (i.e., $w = 1$) at each node would optimize area and power but would also correspond to a maximum value of t^*. Therefore, the problem of device resizing can be seen as finding an assignment of the size parameters w that yields a good area/speed tradeoff. Here we consider the problem of finding a size assignment corresponding to a minimum value for t^* with minimal area/power consumption.

In this context, as far as power estimation is concerned, we consider just the energy dissipated through the drivers. The gate power dissipation can be approximated as a linear function of w, and the total power consumption as a linear function of $W = \Sigma w$. As Fishburn [15] pointed out, the quantity W correlates positively with the total area and other interesting figures of merit of the circuit. Therefore our problem specializes to finding an assignment of the sizes w that minimizes W and corresponds to a minimum value of t^*.

We consider two heuristic approaches to the solution of the problem. A first algorithm starts by setting the size parameter of each node to its minimal size and then raising the parameter w of the critical nodes in an iterative

way. The algorithm is based on the model of Section III. Procedure **critical** computes the data ready times and slacks and returns the critical nodes. Procedure **select** checks first if there exists a critical path in $H(C(0, \hat{T}), B)$ where the size attribute w is maximal (i.e., $w = p$) at each node. In this case, procedure **select** returns an empty set and the algorithm terminates. Otherwise, the procedure deletes from $H(C, B)$ all the nodes with maximal w and selects a critical section in the reduced graph $H'(C', B')$. Procedure **transform** increases the size attribute w by one for each node in S.

The selection of the critical section is done by setting $S = \phi$ and iterating the steps: i) select a node from C' and add it to S; and ii) delete from $H'(C', B')$ all the predecessors and successors of the selected node. The iteration stops when C' is empty. The node selection is done using a greedy strategy by choosing the nodes with largest $|\Delta^1 d(v)|$. The rationale is to maximize the minimum value of $|\Delta^1 d(v)|$, $v \in S \cap C(0, \hat{T})$, which in turn is an upper bound of the improvement of the maximum data ready time $|\Delta t^*|$.

The algorithm terminates when the maximum data ready time t^* is determined by a critical path in $H(C(0, \hat{T}), B)$ consisting of nodes with maximum size attribute w. At this point, the maximum data ready time is insensitive to any other increase of any node attribute w. Unfortunately, the procedure does not guarantee any minimality of W.

A second algorithm uses an opposite heuristic strategy. The attribute w of each node is set initially to its maximum value. Then we decrease the parameter w at some noncritical nodes that would not become critical after w is decreased. These nodes are characterized by the property that $s(v) - \Delta^{-1} d(v) \geq \epsilon$ and are called informally "oversized" because their corresponding driver size is larger than required. Also this algorithm is based on the model of Section III. Here, procedure **critical** returns the subset of C of oversized nodes with nonminimal parameter w (i.e., $w > 1$). Procedure **select** returns a maximal subset of C, whose data ready time attribute can be decreased independently. This corresponds to a maximal subset of nodes of C that are not mutually reachable. Procedure **transform** decrements the parameter w of the nodes in S.

The algorithm terminates when there are no more "oversized" gates, i.e., at a local minimum of W. Note that the value of t^* is not affected by the algorithm, because the size attribute of no critical node is changed. In the implementation of the sizing procedure, the first algorithm is used to determine a size assignment corresponding to a minimum value of t^*. Then, this assignment is used as a starting point for the second algorithm, which ensures a local minimality of W. In the actual implementation, for practical reasons, two other stopping criteria are used in both algorithms. The algorithms terminate if the number of outer iterations reaches a predefined quantity. The first algorithm terminates also if the area/power estimate W reaches a predefined bound $W_{\max}$. Note that by adding these two other stopping criteria, the optimality of the solution cannot be claimed. However, a "good" solution can be found with limited computing time and/or silicon area and power requirements.

In the first algorithm, to reduce the number of iterations needed to reach a minimum t^*, the parameter ϵ can be set to a positive value, usually a fraction of t^*. Widening the set of critical nodes allows us to resize, in the same step, nodes affecting output nodes with data ready time within ϵ from t^*. This may avoid fluctuations in the set of critical nodes, as we mentioned in the case of the resynthesis algorithm. On the other hand, with the choice $\epsilon = 0$, the algorithm resizes only nodes that cause a decrease of the local value of t^*. There is a tradeoff in the choice of ϵ: by increasing ϵ fewer iterations are needed by possibly more nodes are resized and eventually W is higher. In the second algorithm, the choice of $\epsilon = 0$ corresponds to the widest set of "oversized" nodes and therefore is preferable.

VI. Gate and Module Repositioning

We consider here how to improve the circuit timing performance by operating on the node capacitance attribute c and in particular on the component of c which depends on the wiring length. A worse-case estimate of the wiring capacitance of the net interconnecting some gates is computed using the node coordinate attributes, as shown in Section III. As we mentioned previously, the node coordinate attributes are often determined with the goal of minimizing the total wiring length. This figure of merit correlates positively with the silicon area and worst-case power dissipation. Unfortunately, it does not correlate with the switching-time performance measure t^*. The objective of the repositioning algorithm is to improve the timing performance by a set of transformations, called moves, that modify the coordinate attributes of the nodes. As in the case of resynthesis, we assume here that all drivers have maximal size. Resizing is applied after repositioning to select the optimal driver size.

Since the wiring length and capacitance depend on the node coordinates, any move affecting a node changes the parameter c of the node itself and of all its direct predecessors. For this reason, also in this case, we use the concept of favorable transformations to denote those moves that reduce locally the data ready time and do not introduce new critical nodes. Incidentally, favorable moves for this problem can be described by considering the slacks in the geometrical domain. The **geometrical slack** $g(v_i)$ of a node v_i represents the additional wiring length that can be added to the gate output before the corresponding node becomes critical. The geometrical slack $g(v_i)$ is derived from the slack $s(v_i)$ as $g(v_i) = s(v_i)/\delta\beta(w)$, where δ is the capacitance per unit length. It is straightforward to see that a move of node v_i is favorable when: i) the length of the net, which has source node v_i, decreases; and ii) any increase in the length of a net, which has any other node as source, is bounded by the corresponding geometrical slack.

A repositioning algorithm can be implemented by using

the algorithm model, where now procedure **transform** performs favorable moves of the selected nodes. The move set depends on the layout style. For example, in a standard-cell approach, a move may be a displacement of a cell or a pairwise cell interchange.

The repositioning algorithm we implemented was geared to the layout style of the YSC. The YSC system partitions a circuit hierarchically into modules; each module is implemented by a rectangular cell. The leaf cells corresponding to combinational logic blocks are designed as rectangular macrocells consisting of an interconnection of domino gates [7]. Gates are interconnected by wires that run inside and outside the macrocell. In general, gates are highly connected within the module and weakly connected across module boundaries. However, the wiring capacitance is due primarily to the interconnections across the module boundary. Moreover, we decided, for the sake of convenience, not to change the module hierarchy, i.e., not to move gates across the module boundary. For these reasons, we restricted our attention to the intermodule wiring and to the moves that attempt to reduct this quantity. Therefore we considered the modules as the object of the moves. This was done by assigning as coordinate attributes of a node the coordinates of the bottom left corner of the corresponding module.

The algorithm described above can be adapted to the search for module candidates and module moves, by considering pairwise interchanges of modules. The definition of favorable moves is modified accordingly. Since modules may have different aspect ratios and areas, a compatibility relation (represented by a binary matrix) associates to each module those whose differences in area and aspect ratio satisfy a given upper bound. The compatibility relation is used to determine the modules that may be interchanged with a given one. For implementation purposes, as in the case of the resynthesis algorithm, it has been shown to be convenient to detect the critical modules C and then perform several moves of the elements of C, regardless of whether we perform transformations on a critical section or on a larger set. The repositioning algorithm that has been implemented replaces procedure **select** and **transform** by a combined routine, called **interchange**. Procedure **interchange** performs a favorable sequence of pairwise interchanges. The procedure attempts to minimize an objective function which is a weighted sum of the length of the nets whose sources are critical nodes. The procedure uses a greedy strategy in determining the sequence of moves. The local "best" move is the one corresponding to a maximal decrease of the objective function.

Procedure **interchange** performs the following operations. A candidate module is selected from C by using a weighted sum of the parameters of the critical nets connecting it to other modules of compatible shape. Then the set of favorable moves for the candidate module is determined. If no move is possible, the candidate is rejected. Otherwise, the local best interchange is performed and another interchange for that module is searched for. A flag signals that at least one interchange has been performed. When all the critical modules have been examined as candidates, another pass is done if the flag is set. Otherwise the procedure returns.

When the original module positions minimize the total wiring length, this figure of merit is monitored during the interchange. The algorithm trades off the total wiring length for the wiring length of the critical nets. An additional (optional) termination criterion is reaching a bound on the total wiring length.

VII. System Implementation and Results

The major goal of this research project is to combine timing optimization strategies at different circuit abstraction levels. The problems related to the interactions of the different techniques were outlined in the introduction. We present here the actual system implementation. From an algorithmic standpoint, the entire optimization process can be explained in terms of the algorithm model. The circuit is optimized by a sequence of favorable transformations, which guarantee a descent of the timing figure of merit t^*. The iteration stops when either the output data ready times satisfy the requested upper bounds or when area (or other technolgical) bounds are met or when no acceptable transformation is available. The overall iteration is actually an iteration among resynthesis, resizing, and repositioning, i.e., transformations of the same kind are clustered together for the sake of efficiency. In turn, each of these tasks is implemented by subsequences of transformations. Resizing is always applied after resynthesis or repositioning to tune the driver size to the new logic and geometric structure of the circuit. Therefore, the overall iteration alternates the pairs (resynthesis, resizing) and (repositioning, resizing). A sufficient condition for termination is the failure of resynthesis or repositioning to find a favorable transformation.

The program implementation of the algorithms described in the previous sections was influenced by the programming environment of the Yorktown Silicon Compiler system and its data base. The YSC is written mainly in APL, except for a few computational-intensive programs. The high-level interface of the YSC system is written in Rexx, an interpretive language for VM/CMS systems. Data are stored in CMS files, one for each node in the hierarchy. Different views of the circuit being designed (e.g., at the logic or physical abstraction levels) are represented by files with different types on the CMS file system. As far as the implementation of the timing algorithms is concerned, a circuit is represented by a "geometric" hierarchy. Each file, describing an internal node, stores the interconnection of its children as well as their position and points to the files describing them. Each file, related to a combinational leaf module, described the macrocell to be implemented. This representation can be thought of in terms of the graph model and the set of attributes.

The programs that needed a "global view" of the circuit had to access both the hierarchy and the detailed leaf

representation. This is the case of the entire resizing program and the main routines of the resynthesis and repositioning algorithms. The module resynthesis program needed just to access the leaf module representation. The module repositioning program needed just to operate on the representation of the hierarchy, which contains the module position information. The way in which programs view the required data affected their implementation and the selection of the targets of the transformations. The programs were implemented as three APL workspaces. The first one implemented those programs that needed the "global view" of the circuit. The second, implementing resynthesis, is just an "overlay" to the YLE program [7] that performs the logic synthesis stage of the YSC. The last one implements procedure **interchange** of the repositioning algorithm. The workspaces access the data base and communicate through it. The loading of the workspaces, as well as the APL function execution, is determined by short Rexx programs. To optimize execution time, the implementation of the algorithms tends to avoid change workspaces. In particular, the resynthesis algorithm selects at each iteration a set of modules that includes a critical section; therefore, the resynthesis workspace operates on a larger set of modules each time it is loaded. Procedure **interchange** (repositioning) performs a sequence of moves every time the corresponding workspace is loaded.

The first workspace maps the design hierarchy on the file system into an internal hierarchy. Here, the leaf modules are represented by APL character variables. The resynthesis of a module is seen as a variable update, and the change in the internal data structure can be done efficiently. Time stamps are stored to keep track of the evolution in time of the data structure, to avoid duplicate information and keep consistency.

The programs have been tested on some test cases. However, a global evaluation of the capability of the system can be done only on a large-scale example, so that all the different techniques could be exercised appropriately. We report here on the application of the timing optimization algorithms to the design of a 32-bit microprocessor [4]. The circuit, with a total of about 55 000 devices, had a combinational logic component (data path and control) consisting of 58 combinational modules, 1415 domino gates, 2698 nets, and 17 660 devices. The overall computing time for loading the workspaces, creating the internal data structure, and running the algorithms was of the order of a few minutes on an IBM 3090 computer.

The maximum data ready time in the circuit, t^*, was 72.7 ns before optimization. A pass of a resynthesis and a resizing step could reduce the maximum data ready time to 53.2 ns, i.e., by 26.9 percent. Table I summarizes the results of resynthesis. The critical path of the processor was related to the generation of condition codes as a result of a trap condition based on the outputs of the processor ALU. Therefore, the path affected nine critical modules, i.e., 15 percent of the total. In the table, the delays are given while assuming maximal sizes for all drivers. Note

TABLE I
RESULTS OF RESYNTHESIS (EXCLUDING RESIZING)

Module	#Gates/ #Lits	#Elim/ #Extr/ #Resub	Total I/O Delay	Delay Reduction	Percent Delay Reduction
CBFMT	9/50	0/0/3	4.4	<0.8>	18.1%
BFMT	38/275	6/2/0	2.7	<0.9>	33.3%
ALUI	112/499	3/2/12	11.1	<1.9>	17.1%
ALUII	87/961	18/64/20	10.4	<2.9>	27.8%
TRAPGN	5/54	0/0/0	2.7	<0>	0%
IVSSET	25/104	0/0/0	3.3	<0>	0%
RUNSET	6/157	0/0/0	14.4	<0>	0%
KILSET	8/48	0/0/0	4.4	<0>	0%
CRENAB	8/27	3/2/0	6.8	<1.6>	23.5%
total.	298/2175	30/76/35	60.2	8.1 †	13.4%†

†Total delay reduction is lower than or at most equal to the sum of the local delay reductions.

first that module resynthesis was effective on the largest modules, where "large" is related to the number of gates and literals. Smaller modules (e.g. TRAPGN) had a simple structure which could not be modified effectively by the program, or were found already optimally designed by the first logic synthesis stage. In the five modules in which resynthesis was effective, critical signals were speeded up by 0.8 to 2.9 ns. Note that the total delay reduction figure, 8.1 ns, is an upper bound on the actual delay reduction, because data ready times are not additive quantities. Indeed, an effective speedup of about 7 ns (11.6 percent) was achieved by resynthesis only (excluding resizing). After resynthesis, the device sizing algorithm took 46 iterations with $\epsilon = 0$ and 17 iterations with $\epsilon = 2.5$ ns. The increase in $W = \Sigma w$ over the original value (i.e., all drivers with minimal size) was 5.3 percent in the first case and 9.8 percent in the second.

The repositioning algorithm was applied only to a part of the chip consisting of the glue logic circuits implementing the interrupt and global processor control. (The other combinational modules were placed in a stack and their positions were determined to optimize a bus structure interconnection.) One pass of repositioning and resizing reduced the maximum data ready time by an additional 2.1 ns; i.e., the total timing improvement in one pass was about 30 percent. This corresponded to a decrease of the total estimated length of the critical nets of about 4.9 percent.

After repositioning (and the corresponding resizing), the set of critical modules included only two modules (CAFMT and AFMT), which were not critical before and which replaced CBFMT and BFMT. The other seven critical modules were the same as those detected in the first pass. Resynthesis could reduce the delay through CAFMT and AFMT by about 1 ns, which corresponded to an equivalent reduction of t^*. For this circuit example, no other timing improvement could be achieved; i.e., the maximum data ready time was 49.9 ns and the total reduction of t^* was 31.3 percent. At this point, the performance of the circuit was limited by the structural decomposition on which the present techniques have no control. (For example, the speed of module RUNSET was limited by an excessive fan out.) This suggests that further improvements of these methods can be achieved by migrat-

ing the timing optimization techniques to the structural synthesis stage.

VIII. Conclusions and Future Directions

Automated synthesis systems need to incorporate area/power/timing tradeoff strategies to be competitive. Here we have presented some algorithms which improve the timing performance of combinational circuits at the expense of area/power optimality. The algorithms apply circuit transformations to a given circuit representation, which may be a point of optimality for some figures of merit of the circuit. Transformations are applied at different circuit abstraction levels. In particular, resynthesis modifies the structure of multiple-level combination logic blocks, resizing affects the physical device sizes, and repositioning changes the position of gates or groups of gates. Heuristic algorithms implementing these techniques have been programmed and interfaced to a complete automatic synthesis system, the Yorktown Silicon Compiler.

Experience in developing and running the algorithms has taught us that the difficult problem of achieving an automated design that is balanced in the value of its figures of merit can be solved by a stepwise refinement of an initial circuit "draft." Early timing estimation can be based on preliminary (partial) circuit geometric layouts. Indeed, progress has led to faster computer systems and better synthesis algorithms. It is conceivable to use a synthesis system to generate draft layouts using appropriate approximation for the sake of estimating figures of merit. Circuit drafts are then improved iteratively. This process, which contrasts the "one pass compilation" of the earlier systems, mimics better the thinking and action of a human designer.

The interaction among different techniques, such as resynthesis, resizing, and repositioning, has to be studied further in connection with more accurate circuit models that would take the geometric layout information into account. The definition of a circuit partition, by means of structural synthesis, has to be incorporated into this frame and linked to the other techniques. These techniques should also be developed for general classes of circuits, not restricted to the domino family.

Acknowledgment

This research was done at the IBM T. J. Watson Research Center. The author would like to thank Dr. R. Brayton, Dr. C. L. Chen, and Dr. R. Otten for many helpful discussions and ideas.

References

[1] K. Bartlett, W. Cohen, A. De Geus, and G. Hachtel, "Synthesis and optimization of multilevel logic under timing constraints," *IEEE Trans. Computer-Aided Design*, vol CAD-5 no. 4, pp. 582-596, Oct. 1986.

[2] R. Brayton, G. D. Hachtel, C. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA: Kluwer Academic Publishers, 1984.

[3] R. Brayton and C. McMullen "Synthesis and optimization of logic circuits," in *Proc. Int. Conf. Circuit Comput. Design* (Rye, NY), Sept. 1984, pp. 23-28.

[4] R. Brayton, C. Chen, G. De Micheli, J. Katzenelson, C. McMullen, R. Otten, and R. Rudell, "A microprocessor design using the Yorktown Silicon Compiler" in *Proc. Int. Conf. Circuit Comput. Design* (Rye, NY), Oct. 1985, pp. 225-230.

[5] R. Brayton and G. De Micheli "A method for minimizing critical timing in digital systems," *IBM Technical Disclosure Bulletin*, 1985.

[6] R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L. Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung, and A. Sangiovanni-Vincentelli, "Multiple-level logic optimization system," in *Proc. Int. Conf. Computer-Aided Design* (Santa Clara), Nov. 1986, pp. 356-359.

[7] R. Brayton, R. Camposano, G. DeMicheli, R. Otten, and J. van Eijndhoven, "The Yorktown Silicon Compiler system," in *Silicon Compilation*, D. Gaiskj, Ed. Reading, MA: Addison Wesley, 1987.

[8] R. Brayton, "Algorithm for multilevel synthesis and optimization" in *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, Norwell, MA: Martinus Nijhoff, 1987.

[9] M. Burstein and M. Youssef,, "Timing influenced layout design," in *Proc. 22nd Design Automat. Conf.* (Las Vegas), June 1985, pp. 124-130.

[10] R. Camposano and J. van Eijndhoven, "Partitioning a design in structural synthesis," to be published in *Proc. Int. Conf. Circuit Comput. Design* (Rye), 1987.

[11] C. L. Chen and R. Otten "Considerations for implementing CMOS processors," in *Proc. Int. Conf. Circuit Comput. Design* (Rye, NY), Sept. 1984, pp. 48-53.

[12] C. L. Chen, private communication.

[13] J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L. Trevillyan, "LSS: A system for production logic synthesis," *IBM J. Res. Develop.*, vol. 28, no. 5, Sept. 1984.

[14] G. De Micheli, A. Sangiovanni-Vincentelli, and P. Antognetti, Ed., *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, Norwell, MA: Martinus Nijhoff, 1987.

[15] J. Fishburn and A. Dunlop, "TILOS: A polynomial programming approach to transistor sizing," in *Proc. Int. Conf. Computer-Aided Design*, (Santa Clara), Nov. 1985, pp. 326-328.

[16] R. Hitchcock, G. Smith, and D. Cheng, "Timing analysis of computer hardware," *IBM J. Res. Develop.* vol. 26, no. 1, pp. 100-105, Jan. 1982.

[17] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, pp. 443-458, Sept. 1974.

[18] S. Kirkpatric, D. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, May 1983.

[19] R. Krambeck, C. Lee, and H. Law, "High speed compact circuits with CMOS," *IEEE J. Solid-State Circuits*, vol. SC-17, no. 3, pp. 614-619.

[20] J. Luisi, "High-speed low-cost clock-controlled CMOS logic implementation," *U.S. Patent* 3 982 138, Sept. 21, 1976.

[21] D. Marple and A. El Gamal, "Optimal selection of transistor sizes in digital VLSI circuits," in *Advances in Research in VLSI*, P. Losleben, Ed. Cambridge, MA: MIT Press, 1987.

[22] A. E. Ruehli, P. K. Wolff, Sr., and G. Goertzel "Analytical power/timing optimization technique for digital systems," in *Proc. 14th Design Automat. Conf.*, 1977, pp. 147-152.

[23] S. Trimberger, "Automated performance optimization of custom integrated circuits," in *Advances in Computer-Aided Engineering Design*, A. Sangiovanni, Ed. Greenwich, CT: Jai Press, 1985.

*

Giovanni De Micheli (S'79-M'83) was born in Milan, Italy, in 1955. He received the Dr. Eng. degree (*summa cum laude*) in nuclear engineering from the Politecnico di Milano, Italy, in 1979, and the M.S. and Ph.D. degrees, in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

He spent the fall of 1981 as Consultant in Residence at Harris Semiconductor, Melbourne, FL. In 1983, he was Assistant Professor in the Department of Electronics of the Politecnico di Milano. In 1984, he joined the technical staff of IBM T.J. Watson Research Center, Yorktown Heights, NY, where he held the position of Project Leader of the Design Automation

Workstation group. In 1987, he became Assistant Professor in the Department of Electrical Engineering, Stanford University. He was codirector of the NATO Advanced Study Institute on Logic Synthesis and Silicon Compilation, held in L'Aquila (Italy) in 1986. He was granted a Fulbright Scholarship in 1980, a Rotary International Fellowship in 1981, and an IBM Fellowship for VLSI in 1982 and 1983.

Dr. De Micheli received a Best Paper Award at the 20th Design Automation Conference, in June 1983. He received the 1987 Best Paper Award for a paper published in the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. His research interests include several aspects of the computer-aided design of integrated circuits, with particular emphasis on automated synthesis, optimization, and verification of VLSI circuits. He has published several papers in the field. He is coeditor of the book *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, Martinus Nijhoff Publishers, 1987.

Dr. De Micheli has been an active member of IEEE in both the Computer and Circuit & System societies. He is member of the editorial board of the *IEEE Deisgn & Test* magazine. He has served as Group Vice-Chairman and Chairman of the ICCD Conference in 1986 and 1987, respectively, and as a member of the technical committee of the ICCAD Conference. He also served as a member of the executive committee of the New York Chapter of the Computer Society in 1985 and 1986.