PERFORMANCE-ORIENTED SYNTHESIS IN THE YORKTOWN SILICON COMPILER

Giovanni De Micheli

IBM - T.J.Watson Research Center
Yorktown Heights, NY 10598

Abstract: We present some algorithms for the optimization of the switching-time performances of synchronous systems designed by the Yorktown Silicon Compiler. Circuit performance is related to the worst-case propagation delay of signals between two register boundaries and the optimization of circuit performance is equivalent to the minimization of the critical path delay through combinational circuits.

We consider here a global approach to timing performance optimization which involves operations at the **logic, topological** and **physical** level of description of the circuit. We assume that the combinational portion of the circuit being designed is described at the logic level as an interconnection of gates implementing single-output logic functions. This description may be transformed, by a procedure called **logic re-synthesis**, by modifying the internal structure of the logic gates and their interconnection to minimize the propagation delay of those signals limiting the performance of the circuit. At the topological level, we perform a timing-oriented **re-positioning** of groups of logic gates to reduce the delay on the wires along the critical paths. At the physical design level, we perform **re-sizing** of the driver gate sizes to improve the switching speed. These operations are interlaced with the synthesis steps of the Yorktown Silicon Compiler and can be seen as the "code optimizer" part of the compiler that may be invoked when compiling circuits with critical timing performances.

The algorithms are described as well as their implementation and the interface to the Yorktown Silicon Compiler. The results of applying timing-performance optimization to a 32-bit microprocessor design are reported.

1. INTRODUCTION

The Yorktown Silicon Compiler (YSC) [BRAY85a] is an automated synthesis system that aims at generating circuit designs competitive with manual designs in silicon area and switching-time performances. The circuit to be implemented is described as a hierarchical interconnection of modules. The leaf-modules are combinational logic units, registers and library cells, *e.g.* off-chip drivers and receivers. The compiler generates the geometries of the masks of the chip from this high level description. The quality of the "compiled" design is achieved by including several optimization procedures during the logic and physical design phase. We refer the reader to [BRAY85a] for a description of the compiler and to [BRAY85c] for the description of the design of a 32-bit microprocessor using the YSC.

This paper presents some algorithms for optimizing the switching-time performances of the circuits designed by the YSC. We consider here the design of synchronous systems, designed in the SCVS technology [LUIS76] (dynamic CMOS circuits operating in the domino mode). 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.

Previous work on timing performance optimization addressed one particular level of the circuit representation. At the circuit level, Ruehli [RUEH77a], Trimbereger [TRIM85] and Fishburn [FISH85] proposed device-sizing optimization techniques. At the topological level, Burstein [BURS85] studied and implemented placement and wiring strategies that optimize timing performances. At the register-transfer level, Leiserson [LEIS83], addressed the problem of performance optimization by moving the latch boundaries. 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) [HONG74] [BRAY84b], or by choosing a placement of the gates that minimize the routing wire length [KIRK83].

We consider here a global approach to timing performance optimization which involves operations at the logic, topological and physical level of description of the circuit. In particular, at the logic level, we modify the internal structure of the logic gates and their interconnection inside each combinational module. At the topological level, we re-position the modules, and as a consequence their gates, to reduce the delay on the wires along the critical paths. At the physical design level, we optimize the gate sizes to improve the switching speed. These operations are interlaced with the synthesis steps of the Yorktown Silicon Compiler and can be seen as the "code optimizer" part of the compiler that may be invoked when compiling circuits with critical timing performances.

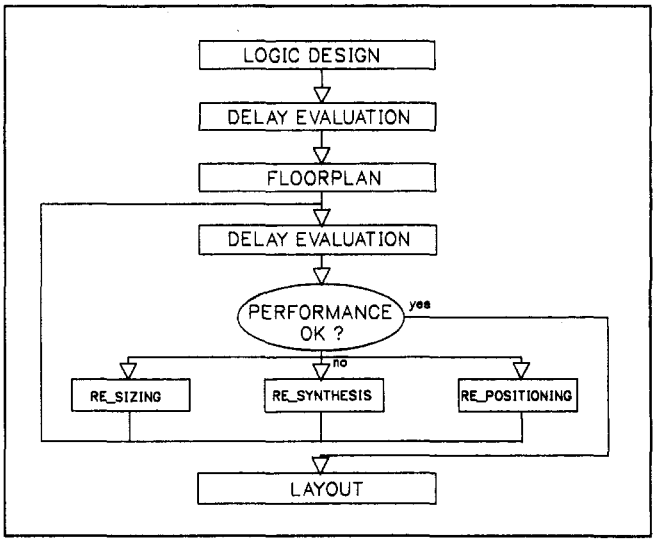
In principle, the strategy for timing optimization can be viewed as an iteration of the two steps:

i) evaluate the critical path delay;
ii) modify circuit appropriately;

until a satisfactory performance is obtained. The evaluation of the critical path delays departs from previous approaches [HITC82] [JOUP83] [OUST85] because the circuit to be optimized is being designed at the same time. In particular, the evaluation of the critical path delay requires the knowledge of the gate positions, because it is affected by the wire lengths. Therefore it is not possible to estimate precisely the delays before the physical design phase: however floorplanning [OTTE84] should be driven by timing considerations. For this reason, we estimate first the critical path delays after the logic design phase, by neglecting the capacitive load due to wiring. The results of this estimate are then used to drive the floorplanning. At this point, a more accurate estimate of the delays can be obtained, because the positions of the modules are known. If the performance is not satisfactory, the circuit may be redesigned for optimal performance by:

i) re-sizing the active devices;
ii) re-synthesis of the combinational modules;
iii) re-positioning the modules.

These operations are iterated until a satisfactory performance is achieved or no improvement is detected. Other figures of merit of the design, such as silicon area and wiring length, are also taken into account. The original logic representation of the circuit optimizes the number of gates and their internal complexity, which correlate to the optimization of the silicon area [BRAY84c]. The algorithms for performance-oriented re-design trade-off the silicon-area optimality for a faster timing performance. Due to the complexity of this approach and the interrelations among the effect of the changes at the logic, topological and physical level, it is not possible to guarantee an optimality of the procedure in rigorous terms. However, the heuristic procedures used for re-sizing, re-synthesis and re-positioning have shown to yield good designs in most cases.



2. CRITICAL PATH DELAY ESTIMATION

We consider here the design of synchronous systems using the YSC, where the registers are synchronized to one system clock. The goal of our technique is to minimize the maximum propagation delay between any two clocked registers. Therefore we model the signal propagation through the combinational part of the system. We consider each signal stored in a register as both a primary input and output to the combinational sub-system.

Combinational logic can be described as an interconnection of **logic gates** by means of **nets**, carrying logic **signals**. The primary inputs to this network are also interconnected to the logic gates by means of nets. The primary outputs are identified by the gates generating the corresponding signal. We assume each logic gate to be unidirectional. Therefore we can associate a direction to each net corresponding to the signal propagation direction along that net. The interconnection can be modeled by a directed graph $G(V, A)$, whose node set $V = \{v\} = V^g \cup V^i = \{v^g\} \cup \{v^i\}$ is in one to one correspondence with the set of logic gates (V^g) and primary inputs (V^i) and whose edge set A is in one to one correspondence with the nets. To avoid race conditions, the system design is constrained to be unidirectional, i.e. $G(V, A)$ is acyclic. A node v_i is said to be a **predecessor (successor)** of gate 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. The signal propagation is modeled by associating a **propagation delay** $d(v^g)$ to each logic gate or, equivalently, a weight to each node in the set V^g. The propagation delay through a physical gate can be modeled by simulating a transition due to a change in an input value. In the case of SCVS circuits [CHEN84a], the delay model of a gate depends on: i) the size w of the drivers (i.e. the devices implementing the static inverter driving the output wires of a gate); ii) the capacitive loading c at the output, which in turn depends on the fanout of the gate and the wiring capacitance to ground; iii) the structure of the path discharging the sense node and in particular the maximum number of devices in a discharging path l. It is convenient to express the propagation delay as $d = \alpha(w) + \beta(w)c + \gamma(w)l$. The coefficients α, β and γ are tabulated for a finite number of values of w and are obtained by regression analysis after circuit simulation [CHEN85]. We associate also to each logic gate a **data ready** time $t(v_i^g)$, $i = 1, 2, \ldots, |V^g|$. The data ready time of a gate is the time at which the signal generated by that gate is ready. Similarly, we associate to each primary input a data ready time $t(v_i^i)$, $i = 1, 2, \ldots, |V^i|$. 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 at the logic gates 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 corresponding to a gate in a sequence consistent with the partial order represented by the graph.

An important information about signal timing is the **slack** of the signals generated by each gate (called also slack at the gate). The slack of each primary output (or at the gates generating a primary output) is defined as the difference between a chosen time $\hat{t}(v_i)$ (*e.g.* the minimum system cycle time) and the computed data ready time $t(v_i)$. For the other gates, the slack $s(v_i)$ at node v_i is defined to be:

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

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

The slacks at each gate measure how much additional delay may each signal tolerate, while satisfying the relation $t(v_i) < \hat{t}(v_i)$ at each node v_i corresponding to an output signal. The slacks can be computed by tracing the signal propagation backward in the circuit, i.e. by computing the slack at each gate in a sequence consistent with the partial order represented by the graph obtained from $G(V, A)$ by reversing the direction of each edge.

Let $\varepsilon \geq 0$ be an arbitrary constant. The set of **critical nodes** $C \subseteq V$ is the set of nodes $C = \{v \text{ s.t. } s(v) \leq \varepsilon\}$. The **critical graph** $H(C, B)$ is the subgraph of $G(V, A)$ induced by C. A **critical path** is a directed path in the $H(C, B)$. The meaning of critical node, graph and path depends on the choice of $\hat{t}(v_i)$ at each node v_i corresponding to an output signal and on the parameter ε. If $\hat{t}(v_i)$ is the required circuit cycle time (or a required arrival time for circuit output signals) and ε is a negative number which takes into account safety margins and tolerances, then the critical nodes represent the gates whose data ready time need to be reduced to meet the timing specifications. With this choice, this formalism can be used to design the circuit in a performance feasibility region. A different design strategy may try to achieve the best timing performance of a given circuit and then adjust the system clock accordingly. In this case, let v^* be the node corresponding to the highest data ready time in the circuit, i.e. $v^* = \arg \max_{v \in V} t(v)$ Then, the system clock period is bounded from below by v^*. Timing performance optimization aims at reducing v^*. Let $\hat{t}(v_i) = v^*$ for each node v_i corresponding to an output signal. If we choose $\varepsilon = 0$, then the critical graph is the set of nodes having the property that any independent variation in the data ready times of any node separation set implies a variation in $t(v^*)$. (In particular, the critical graph may be a simple path and the separation set just any node.) Therefore the nodes (gates) along the critical graph are "critical" because the timing performance of the circuit (limited by $t(v^*)$) can be optimized by improving the data ready time at the critical nodes. This can be done by reducing the propagation delay at that node, or at a critical predecessor. Note also that a variation of the data ready time of a critical node v_i may influence $t(v^*)$ by very little if the direct successor of that node has a non-critical 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". The choice of positive values for parameter ε allows to widen the set of critical nodes.

Though our approach is fairly general, we consider in the sequel the problem of minimizing the maximum data ready time v^* and therefore we choose $\hat{t}(v_i) = v^*$ for each node v_i corresponding to an output signal and we set ε to a proper fraction of v^*. From the model of the circuit and the model of the propagation delay, it is clear that the timing performance can be improved by changing the sets of parameters $\{w\}$ (driver size), $\{c\}$ (load capacitance) and $\{l\}$ (gate structure) and/or the structure of the graph $G(V, A)$. Device re-sizing aims at optimizing the set of parameters $\{w\}$. Circuit re-synthesis improves the timing performance by changing the structure of the graph $G(V, A)$ and, as a by-product, the parameters $\{l\}$ and $\{c\}$. The module re-placement aims at speeding-up the circuit by modifying the parameters $\{c\}$ by changing the wire lengths.

An efficient design requires a trade-off between area and timing performances. In our model, a figure of merit $\mathcal{A}$ measures the required silicon area. Figure $\mathcal{A}$ is a function of the number of gates, the total number of devices inside each gate the number and size of the drivers used in each gate.

3. DEVICE RE-SIZING

The device sizing strategy used here is a heuristic descent technique. The size of the device implementing the drivers is adjusted (re-sized) by changing its width, which is a linear function of the parameter w ; the device length is kept constant. The sizes of the devices of the discharging path are kept constant, because the capacitance of the sense (pre-charged) node is negligible with regard to the gate and wire capacitance connected to the driver output. It is practical for layout design purposes 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 $\{1, 2, \ldots, p\}$. 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 drivers 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, i.e. $\Delta d(v) \equiv d(w + 1, c, l) - d(w, c, l)$ is always negative for each node $v \in V^g$. Note also that the SCVS 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 propagation delay through the corresponding gate and does not cause an increase of the data ready time at any other gate, i.e. the mapping $w \rightarrow v^*$ is monotonically decreasing.

The algorithm for device sizing can be sketched as follows:

STEP 1: Compute the critical graph $H(C, B)$. Let $H(C', B')$ be the graph obtained from $H(C, B)$ by deleting the nodes corresponding to gates with maximal size, i.e. with $w = p$.

STEP 2: If no node in C' has non-positive slack, stop. Else set $\mathcal{G} = \phi$.

STEP 3: Let C'' be the subset of C' corresponding to the nodes with minimal slack. Select $o \in C''$ as the node that maximizes $|\Delta d(c)|$. Let $\mathcal{G} = \mathcal{G} \cup \{o\}$.[1]

STEP 4: Delete from $H(C', B')$ all the nodes that are predecessors or successors of o. If the node set C' is not empty, go to STEP 3.

STEP 5: Increment w for the gates corresponding to the nodes in $\mathcal{G}$. Go to STEP 1.

The detection of the critical nodes is achieved by computing the data ready times and the slacks, as described in Section 2. The computation of the critical nodes is done at each outer iteration of the algorithm, because the data ready times and the slacks are affected by the changes in the parameters $\{w\}$. At STEP 3 and STEP 4, the algorithm constructs a node separating set $\mathcal{G}$ of the graph $H(C', B')$. The selected node o is chosen among those with minimal slack in the set C'' as the one with maximal decrease in propagation delay for a unit increment in the driver gate size. The motivation of such a selection is as follows. The nodes with minimal slack (C'') correspond to the most critical gates in the circuit. To decrease the data ready time at one of these gate (target gate) we can decrease its propagation delay or the propagation delay of an appropriate predecessor. Predecessors with higher slacks are disregarded, because changes of their propagation delay would not affect the data ready time at the target gate. For this reason we choose as target gate one in the set C'' and in particular the one that maximizes $|\Delta d(v)|$, which in turn is an upper bound of $|\Delta d(v^*)|$. At STEP 5, the sizes of the gates represented by $\mathcal{G}$ are incremented by one. This has two consequences. First, the maximum data ready time $t(v^*)$ decreases, because $\mathcal{G}$ contains at least one critical node with non-positive slack. Second the data ready time at other "almost critical" output gates may decrease, in particular at some primary output gates whose data ready times differ from v^* by less than ε.

[1] There may be more than one node in C' with minimal slack. More precisely, let $\sigma = \min_{c \in C'} s(c)$. Then $C'' = \{c \in C' \text{ s.t. } s(c) = \sigma\}$, and $o = \arg \max_{c \in C''} |\Delta d(c)|$.

The rationale of the algorithm is to apply a global strategy for timing optimization. In fact, local optimization of only one output data ready time would be of little value if another output gate has a close data ready time. This other gate would probably generate the signal with highest data ready time at the next iteration of the algorithm and the algorithm would take many iterations to improve sensibly the circuit timing performance. Note that by choosing $\varepsilon = 0$, only the sizes of the gates that affect $t(v^*)$ are incremented. (Local strategy). The global strategy is achieved by choosing $\varepsilon > 0$. There is a trade-off in the choice of ε: by increasing ε fewer iterations are needed to achieve a given timing performance but possibly more devices are re-sized than those strictly needed.

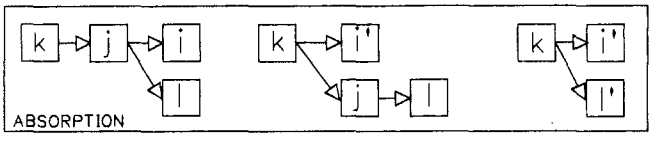
The algorithm described above terminates if there are no critical nodes whose size can be increased and affect $t(v^*)$, i.e. critical nodes with zero slack. Equivalently the algorithm terminates when the maximum data ready time $t(v^*)$ is determined by a sequence of gates with maximum sized drivers. Since v^* is monotonically decreasing with w, the descent algorithm terminates at a point of constrained optimality. To improve further the timing performance of the circuit with the given technological constraints on the maximum size of the drivers, it is necessary then to change the structure of the circuit by re-synthesis or to change the position of the gates, as described in the next two sections.

In the implementation of the algorithm, for practical reasons, two other stopping criteria are used at STEP 2. The algorithm terminates if the number of outer iterations reaches a predefined quantity or if the area estimate $\mathcal{A} = f(w)$, reaches a pre-defined bound $\mathcal{A}_{max}$. Note that by adding this 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. This solution is the starting point for the other optimization strategies described in the next sections.

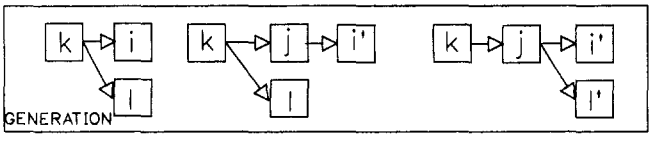
4. CIRCUIT RE-SYNTHESIS

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 nodes and/or edges. Note that a change in the graph structure affects both the gate fanout (and therefore the set of parameters $\{c\}$) and the gate fanins (and therefore the gate structure and the set of parameters $\{l\}$). We consider only implementable transformations, i.e. we assume that the internal parameters of the gates involved satisfy the technological constraints after the transformation. We consider two basic transformations: gate **absorption** and gate **generation**.

Let gate j be a direct predecessor of gate i. The gate absorption is the replacement of gate i by gate i'. The inputs to gate i' are all the signals that are input to gate i and to gate j, except for the output of gate j. Gates i and i' are equivalent in the sense that their outputs coincide for any valid combination of the circuit primary inputs. Note that gate j needs not to be implemented, if it has no successor and if its output is not a primary output of the circuit.



Gate generation is the transformation opposite to absorption. Gate i is replaced by gate i'. A new gate, j, is generated and it is a direct predecessor of gate i'. The inputs to gate i' are some of the signals that are input to gate i and the output of gate j; the inputs to gate j are the remaining inputs to i. Also in this case, gates i and i' are equivalent in the sense that their outputs coincide for any valid combination of the circuit primary inputs.



Gate transformations change the number of stages, or logic levels, needed to implement one (or more) primary output of the circuit. In general, for a gate absorption (generation) gate i' is more complex (simpler) than gate i and therefore its propagation delay may get larger (smaller). The data ready times of all the gates that are successors of gate i are affected by the transformation. Moreover, the fanout of the direct predecessor of gate i' and j may change, and, as a result, the data ready times at some other circuit outputs. Let $t'(v)$ denote the data ready time after the transformation and let $\Delta u(v) \equiv t'(v) - t(v) - s(v)$. A transformation of gate i is said to be **locally favorable** if in the modified circuit: i) $\Delta u(v_i) + s(v_i) < 0$, i.e. the data ready time decreases; ii) $\Delta u(v_j) < 0 \ \forall j \neq i$, i.e. any increase in the data ready times is less than the slack. Since a gate transformation affects the entire circuit, it is important to measure its global impact on the timing of the circuit. Let $\Delta \underline{u} \equiv [\Delta u(v_1), \Delta u(v_2), \ldots, \Delta u(v_{|V|})]$. Then a figure of merit of the global impact of the transformation is measured by $\underline{\eta}^T \cdot \Delta \underline{u}$, where $\underline{\eta}$ is a vector of coefficients. Gate transformations affect the silicon area (estimated by $\mathcal{A}$) of the circuit implementation, because of the change in the number of gates and in their internal structure. Therefore a comprehensive figure of merit is $\Delta e \equiv \underline{\eta}^T \cdot \Delta \underline{u} + \theta \Delta \mathcal{A}$, where θ is a scalar coefficient.

Circuit re-synthesis can be applied directly to the entire combinational portion of the circuit represented by $G(V, A)$, by using an algorithm with a strategy similar to that presented in Section 3:

STEP 1: Compute the critical graph $H(C, B)$. Let $H(C', B')$ be the graph obtained from $H(C, B)$ by deleting the nodes corresponding to the gates that cannot be transformed due to technological constraints.
STEP 2: If no node in C' has non-positive slack, stop. Else set $\mathcal{E} = \phi$.
STEP 3: Let C'' be the subset of C' corresponding to the nodes with minimal slack. Select a node $e = \epsilon\ C''$ and a locally favorable transformation that maximizes $|\Delta e|$. Let $\mathcal{E} = \mathcal{E} \cup \{e\}$.
STEP 4: Delete from $H(C', B')$ all the nodes that are predecessors or successors of e. If the node set C' is not empty, go to STEP 3.
STEP 5: Perform the chosen transformations at the gates corresponding to the nodes in $\mathcal{E}$. Go to STEP 1.

The algorithm allows fairly general circuit transformations. However, it is inefficient to apply it to large circuits because for each node there are several transformations to be evaluated. In particular, each selected node may absorb one or more predecessors. Similarly, for each selected node, several gate generations may be possible by considering the possible decompositions of the logic function implemented at the gate. Each transformation requires a recomputation of the data ready times. Unfortunately, in the case of large circuits, the computing-time requirements make this approach unattractive.

The strategy for circuit re-synthesis which has been implemented exploits the structure of the circuit and the nature of the transformations. A partition of the the circuit into functional units with specific tasks (e.g. ALU, rotator, ...) is used. This partition corresponds to the one used by the YSC for synthesis purposes, i.e. the blocks of the partition are the combinational modules that are leaves of the hierarchical circuit description. Transformations are applied only inside the combinational modules. Each combinational module is described by a sub-graph $G^m(V^m, A^m)$ of $G(V, A)$. A **critical module** is a module whose subgraph has critical nodes.

At first, the critical graph for the entire circuit is computed. Then, the critical modules (i.e. the modules along the critical graph) are redesigned one at a time. For this purpose, each graph corresponding to a critical module is considered along with its boundary conditions, which are the data ready times at the module inputs and the slacks at its outputs. While doing re-synthesis of a module, the goal is to reduce the data ready times at the critical nodes. Note that since the module is "extracted" from the whole circuit, the slacks cannot be recomputed after each transformation. Therefore the critical nodes are computed only once and the transformations are applied to reduce the data ready times at these nodes.

The overall goal of the algorithm is an area-time-effective synthesis. An absorption leads to a more compact implementation, if the absorbed node is not implemented. Therefore, if the absorbed node had more than one direct successor, it is convenient to try to have it absorbed by its direct successors, even if these absorptions would not improve the timing performances; they would improve though the silicon area. The generation of a gate requires the implementation of an additional gate: the cost of the additional area can be (partially) offset by the internal simplification of the successor gate. For this reason, it is convenient to consider the "common sub-expressions" [BRAY84c] of the logic functions implemented by two (or more) gates as candidates for the nodes to be generated. In this case, any node generation would lead to the simplification of the two (or more) direct successors of the generated node, and eventually to a reduced loss (or possibly a saving) in silicon area.

STEP 1: Compute the data ready times $t(v)$ and the slacks $s(v)$ for all nodes in $G(V, A)$. Compute the critical graph $H(C, B)$. Select a critical module. Let $G^m(V^m, A^m)$ be its graph representation and C^m be the critical nodes within the module. Let $\mathcal{A}_{max}$ be an upper bound on the area of the module.
STEP 2: For each critical node $e \in C^m$ and for each direct predecessor $f \in C^m$ of e, let e absorb f if the absorption is locally favorable and $\mathcal{A} < \mathcal{A}_{max}$.
STEP 3: For the logic functions implemented by the gates corresponding to V^m, compute all common subexpressions. For each sub-expression common to at least one node in C^m, generate the corresponding gate if the generation is locally favorable and $\mathcal{A} < \mathcal{A}_{max}$.
STEP 4: Perform area/delay trade-off by moving subexpressions across gate boundaries to reduce gate and device count.
STEP 5: Disregard the module from further considerations. If all critical modules have been considered, stop. Else, go to STEP 1.

The algorithm selects at STEP 1 as critical module a sub-circuit whose nodes have no predecessor in any other critical module which has not been re-synthesized. The reason is not to change the data ready times at the input gates of a module after its re-synthesis, since the circuit structure has been tuned to this particular input signal arrival-time distribution. For the selected module, the number $\mathcal{N}_{max}$ is an upper bound on the module area after re-synthesis, because node absorption and generation lead in general to an increase in silicon area. The common sub-expressions are computed as described in [BRAY84c]. After having re-synthesized the module, the data ready times and the slacks are recomputed for the whole circuit. The algorithm terminates when all critical modules have been re-synthesized.

5. CIRCUIT RE-POSITIONING

The re-positioning algorithm aims at improving the timing performances by reducing the capacitive load $\{c\}$ of the critical gates. This is done by shortening the length of the nets carrying critical signals. In the YSC, each module is designed as a rectangular macro-cell consisting of an interconnection of cells implementing each a logic gate. Gates are interconnected by wires that run inside and outside the macro-cell. We assume that the capacitive loading on a gate depend primarily on the length of the wires that go across the module boundary. Therefore we restrict our attention to the inter-module wiring and we estimate the wire lengths by the macro-cell positions.

The goal of the algorithm is to reduce the length of the critical nets by changing the mutual positions of the modules. The rationale of the re-positioning algorithm is to "bring closer" the critical nodes (gates) to their direct successors in other (not necessarily combinational) modules. Since any change in a module position affects the length of all the nets connected to it, we require that any change in the module position correspond to an improvement in timing performance. For this reason we introduce the **geometrical slack** $g(v_i)$ for each net connected to the output of the gate denoted by v_i. The geometrical slack $g(v_i)$ is derived from the slack $s(v_i)$ and represents the additional length that each net can tolerate while satisfying the relation $t'(v_i) \geq t(v_i)$ at each output node v_i.

The re-positioning algorithm is based on pair-wise module interchange. The moves that are allowed are position interchanges between modules with compatible shape and such that the increase in wire-length for each net connected to each gate v_i of the modules under consideration is less than $g(v_i)$. The objective function is a weighted sum of the net-lengths connected to the critical nodes. The algorithm uses a greedy strategy in determining a sequence of allowed moves corresponding to a maximal decrease of the objective function. The algorithm can be sketched as follows:

STEP 1: Compute the critical graph $H(C, B)$, the set of critical modules M, the slacks $s(v)$ and the geometrical slacks $g(v)$.
STEP 2: Let $M' = M$. Reset the flag.
STEP 3: If $M' = \phi$ and the flag is set go to STEP 2. If $M' = \phi$ and the flag is reset stop.
STEP 4: Select $m \in M'$. Determine the set of allowed moves for m. If this set is empty, remove m from M'. Else, choose the interchange that maximizes the decrease of the objective function and set the flag. Go to STEP 3.

The algorithm determines first the set of critical modules. Then a candidate module for interchange is selected by using a weighted sum of the parameters of the critical nets connecting it to other modules of compatible shape. Then the set of allowable moves is determined by considering the geometrical slacks. If no move is possible, the candidate is rejected. Else, 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 (STEP 3 and STEP 4) is done if the flag is set. Else the algorithm terminates. If the original placement was obtained by minimizing the total wire length, this number is monitored during the interchange. The algorithm trades off the total wire length for the wire length of the critical nets. An additional termination criterion that may be used is reaching a bound on the total wire-length.

6. IMPLEMENTATION AND RESULTS

The algorithms have been implemented in three APL workspaces. The program that evaluates the delay represents each combinational module by an APL variable and constructs a pointer structure that allows a fast evaluation of the data ready times and the slacks. The re-synthesis program is implemented as an overlay of the YLE program [BRAY84c]. The re-synthesis of a module corresponds to the update of a variable in the delay workspace, and therefore the delay and slack evaluation can be executed efficiently without re-linking the entire circuit data-structure. Similarly, the re-positioning algorithm is implemented by an APL workspace that examines the compound (i.e. not leaf) modules of the hierarchical chip description. The description of these modules contains the relative position of the corresponding child-modules. This information is used in the re-positioning algorithm.

The program has been used to optimize the performance of a 32-bit microprocessor design having 1415 SCVS gates corresponding to 17660 devices and a total of 55066 transistors including latches, a register file and off-chip drivers and receivers [BRAY84c]. The combinational portion of the circuit which has been optimized had 1565 equivalent logic gates ($|V^g|$) and 2698 nets ($|A|$).

The device sizing algorithm was able to reduce the maximum data ready time from 72.7 ns to 57.8 ns i.e. by 20.5%. It took 46 iterations with $\varepsilon = 0$ and the increase in driver active area corresponding the the driver sizing was 5.3%. With $\varepsilon = 2.5$, it stopped after 17 iterations and the increase in the driver active area was 9.8%. The computing time of a program that reads in the chip data description, forms the internal data structure, runs the algorithm and updates the chip description took 202 and 178 seconds on an IBM 3081 computer in the two cases. Note that only a minor fraction of the computing time was spent by the sizing algorithm.

The delay estimation algorithm detected 9 critical modules along the critical path. Circuit re-synthesis was effective only for 5 of these modules, primarily because the overall circuit partition had large and small modules (in terms of number of gates) and re-synthesis could not speed-up further the small modules, which were optimally designed by YLE in the original synthesis. Critical signals were improved, after re-synthesis, by .5 to 1.6 ns, with an average speed-up of 9%. This figure is computed as the average of the ratio of the difference in data ready time to the total propagation delay through the module.

The re-positioning algorithm was applied to the compound module consisting of the "random logic" units corresponding to the control and interrupt portion of the microprocessor. The re-positioning algorithm reduced the critical net length inside the module by about 5%. This corresponded to an improvement of the maximum data ready time of about 2.1 ns, i.e. 3.7%.

7. CONCLUSIONS

Automated synthesis systems need to incorporate area-timing optimization procedures to be effective. Here, a set of procedures for optimizing the area-timing performances of the circuits designed by the Yorktown Silicon Compiler has been presented. Optimization is achieved at the logic level by re-synthesis, at the topological level by re-positioning and at the device level by re-sizing. Experimental results on a micro-processor design example have shown the applicability and effectiveness of the optimization methods presented here for large designs.

8. ACKNOWLEDGEMENTS

These ideas have been shared with Dr. R.Brayton and Dr. R. Otten of the T.J.Watson Research Center.

9. REFERENCES

[BRAY84b] R.Brayton, G.D.Hachtel, C.McMullen and A.L.Sangiovanni- Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[BRAY84c] R.Brayton and C.McMullen "Synthesis and Optimization of Logic Circuits", Int. Conf. on Circ. and Comp. Des., Rye, NY, pp.23-28, Sep 1984.

[BRAY85a] R.Brayton, N.Brenner, C.Chen, G.De Micheli, C.McMullen and R. Otten "The Yorktown Silicon Compiler" Proc. Int. Symp. on Circuit and Systems, Kioto, Japan, pp. 391-394, Jun 1985.

[BRAY85c] R.Brayton, C.Chen, G.De Micheli, J.Katzenelson C.McMullen R. Otten and R.Rudell "A Microprocessor Design Using the Yorktown Silicon Compiler" Proc. Int. Conf. on Circuit and Comput. Des., Rye, N.Y., pp. 225-230, Oct 1985.

[BURS85] M.Burstein and M.Youssef, "Timing Influenced Layout Design" Proc. 22th Des. Autom Conf. pp. 124-130, Las Vegas, Jun 85.

[CHEN84a] C.L.Chen and R.Otten "Considerations for Implementing CMOS Processors", Int. Conf. on Circ. and Comp. Des., Rye, NY, pp.48-53, Sep 1984.

[CHEN85] C.L.Chen, Private Communication.

[FISH85] J.Fishburn and A.Dunlop "TILOS: A Posynomial Programming Approach to Transistor Sizing" Int. Conf. on Comp. Aid. Des., Santa Clara, pp. 326-328, Nov. 1985.

[HITC82] R.Hitchcock, G. Smith and D.Cheng, "Timing Analysis of Computer Hardware", IBM Journal of Research and Development, Vol. 26, No.1 pp.100-105, Jan. 1982.

[HONG74] S.J.Hong,R.G.Cain and D.L.Ostapko, "MINI: a Heuristic Approach for Logic Minimization", IBM Jour. of Res. and Dev., vol. 18, pp. 443-458, Sep. 1974.

[KIRK83] S.Kirkpatric, D.Gelatt and M. Vecchi, "Optimization by Simulated Annealing", Science, May 1983.

[JOUP83] N.Jouppi, "TV: an nMOS Timing Analyzer", in R.Bryant, Editor Third Caltech Conference on VLSI, Computer Science Press, 1983.

[LEIS83] C.Leiserson, F.Rose and J.Saxe "Optimizing Synchronous Circuitry by Retiming", in R.Bryant, Editor Third Caltech Conference on VLSI, Computer Science Press, 1983.

[LUIS76] J.Luisi, "High-speed low-cost clock-controlled CMOS logic implementation", U.S. Patent 3982138, Sept 21, 1976.

[OTTE84] R.Otten and L.Van Ginneken "Stepwise Layout Refinement", Int. Conf. on Circ. and Comp. Des., Rye, NY, pp.30-36, Sep 1984.

[OUST85] J.Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI" IEEE Trans. on CAD/ICAS, Vol. CAD-4, No 3, pp. 336-348, July 1985.

[RUEH77a] A.E. Ruehli P.K. Wolff Sr. and g.Goertzel "Analytical power/timing optimization technique for digital systems" Proc. 14th Des. Autom Conf.

[TRIM85] S.Trimberger "Automated Performance Optimization of Custom Integrated Circuits" in A.Sangiovanni, Editor Advances in Computer-Aided Engineering Design, Jai Press, 1985.