Algorithms for Synchronous Logic Synthesis

Giovanni De Micheli Thierry Klein
Computer Systems Laboratory
Stanford University

Abstract

This paper presents a new approach to logic synthesis of digital synchronous sequential circuits. We describe here algorithms for minimizing i) the area of synchronous combinational and/or sequential circuits under cycle time constraints and ii) the cycle time under area constraints. Previous approaches attacked this problem by separating the combinational logic from the registers and by applying circuit transformations to the combinational component only. We show in this paper instead how to optimize concurrently the circuit equations and the register position. This method is novel and can achieve results that are at least as good as those obtained by previous methods. A computer implementation of the algorithms in program Minerva is described.

1 Introduction

Logic synthesis has shown to be of pivotal importance in the computer-aided design of integrated circuits. Logic synthesis systems have been the object of extensive investigation and commercial implementations have shown to be practical for product-level design of digital circuits.

Most circuits of interest in digital design are **synchronous** logic circuits, that are interconnections of logic gates and registers with synchronous clocking. Feedback connections are restricted to be through synchronous registers, to guarantee race-free design. Semi-custom circuit implementations, such as standard-cells and sea-of-gates, have motivated the use of multiple-level (or multiple-stage) logic synthesis techniques. In particular, such implementations have shown to be more flexible and faster than two-level implementations, such as Programmable Logic Arrays. As a result, several techniques for multiple-level logic synthesis techniques have been investigated and clever algorithms for combinational logic synthesis have been reported in the literature [1] [2] [3] [4].

However, techniques for synthesizing synchronous logic circuits have been lagging behind, due to the additional complexity of handling registers and feedback connections. Most logic synthesis systems deal with such circuits by partitioning them into an interconnection of a combinational logic component and registers. The combinational portion of the circuit is optimized by combinational logic algorithms. Then registers are added back to the circuit. Needless to say, such optimization techniques are limited in their scope by this partitioning strategy.

We attempt in this paper to solve the synchronous logic synthesis problem by considering algorithms that operate on the entire sequential circuit, i.e. that do not separate registers from the combinational component. For this reason, we introduce the concept of synchronous Boolean network and we study transformations on this network that preserve I/O equivalence and that optimize i) the circuit area under cycle time constraints and ii) the cycle time under area constraints. Some of these transformations are a superset of those used in combinational logic synthesis and operate within and across the register boundaries. Therefore the potential quality of the optimized circuits is at least as good as that obtained by the previous techniques that were constrained to operate on the combinational component only.

The register position is determined as a by-product of these circuit transformations. It is important to remember that a technique to position the registers in a network, called **retiming**, was introduced by Leiserson and Saxe [5] in a different context, where logic synthesis transformations were not considered. This paper presents a model for synchronous logic synthesis that combines retiming with combinational logic synthesis techniques. Then algorithms that minimize the circuit area and cycle time are described. The algorithms are implemented in computer program Minerva, that performs combinational and sequential logic synthesis.

2 Basic concepts and definitions

We consider synchronous circuits that are interconnections of combinational logic gates and single-clock positive-edge-triggered registers with negligible setup times. We model synchronous circuits by **synchronous Boolean networks**. A synchronous Boolean network is described in terms of Boolean variables and Boolean functions. Each Boolean variable corresponds to either a primary input/output of the circuit or to the output of a combinational logic gate. A positive integer label on a variable (superscript) denotes the synchronous register delay, if any, of the corresponding signal with respect to the primary input or combinational logic gate that generates it. Zero-valued labels are omitted for the sake of simplicity. Each Boolean function specifies the value of a variable in terms of other variables, i.e. it is a multiple-input single-output combinational logic function. It is represented by an equation, whose left term is a variable with zero-valued label and whose right term is an expression, e.g. the equation at vertex v_i is represented by $i = \mathcal{I}$, where $\mathcal{I}$ is a Boolean expression in terms of other (labeled) variables.

The network is modeled by the **synchronous network graph**, that is a directed weighted multi-graph $G(V, E, W)$, whose vertex set $V = V^I \cup V^G \cup V^O = \{v\}$ is in one-to-one correspondence with the variables corresponding to the set of primary inputs, logic gates and primary outputs respectively. The edge set E and the edge weight set W are defined as follows. There is an edge between v_i and v_j with weight k when variable i appears in the expression $\mathcal{J}$ for vertex v_j with label k. Zero-valued weights are not indicated by convention. There is a (weighted) edge to each output vertex in V^O from the vertex in V^G corresponding to the gate generating that output signal. For each pair of vertices joined by a path in $G(V, E, W)$, the path weight is the sum of the weights along the path. We assume that each cycle (i.e. closed path) has strictly positive weight, to model the restriction of breaking combinational logic cycles by at least one register. An example of a synchronous Boolean network and its representation is shown in Fig. 1.

In general, a synchronous Boolean network may have cyclic dependencies, i.e. its corresponding graph be cyclic. A network is called **unidirectional** when the graph $G(V, E, W)$ is acyclic. It models a

ISCAS '89

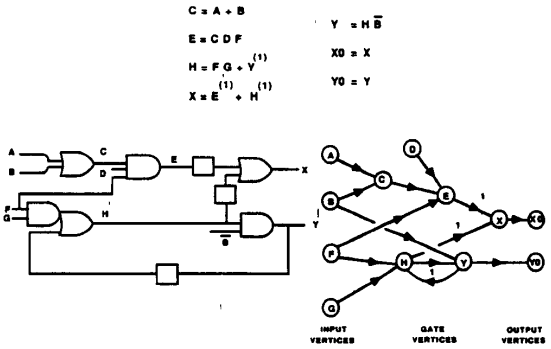

Figure 1: Synchronous Boolean Network and its representation.

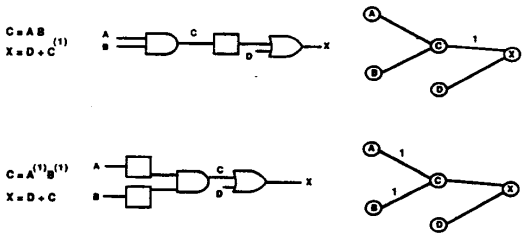

Figure 2: Retiming vertex v_c by +1.

pipelined combinational circuit. Note that the combinational Boolean network (without synchronous registers) introduced by Brayton [1] is just a special case of the synchronous Boolean network that is acyclic and whose labels are all zeroes.

The (direct) **fanin** set of a vertex v_i is the subset of vertices that are tail of an edge (with zero weight) incident to v_i and it is denoted by $FI(v_i)$ ($DFI(v_i)$). Similarly the (direct) **fanout** set of a vertex v_i is the subset of vertices that are head of an edge (with zero weight) incident to v_i and it is denoted by $FO(v_i)$ ($DFO(v_i)$). Each vertex of the graph $v_i \in V^G$ (i.e. corresponding to a gate) has as attributes an area estimate l_i in terms of literal count [1] and a positive gate delay d_i, which depends on the logic expression and which is a monotonically increasing function of l_i [1]. Each input and each output vertex has zero delay.

Each vertex v_i has a **data ready time** t_i, that is the time at which the signal generated by the corresponding gate is ready with respect to the clock edge [6]. We assume the primary inputs to be synchronized to the clock positive edge and therefore their data ready time is zero. For any other vertex v_i, the data ready time is the sum of its propagation delay d_i to the largest data ready time of its inputs that are not registers, i.e. $t_i = d_i + max_{v_j \in DFI(v_i)}(t_j)$. Since the subgraph representing the direct fanin relation is acyclic, the data ready time can be computed by topological sort.

Given a cycle time Φ, a synchronous network is a **timing-feasible** implementation if all the data ready times are bounded from above by the cycle time, i.e. $\Phi \geq max_{v_i \in V}(t_i)$. Each vertex v_i has a **slack** s_i representing the additional delay that the vertex can tolerate while preserving timing-feasibility of the network for a given Φ [6]. In a timing-feasible network a vertex is **critical** if its slack is null.

The area taken by a network implementation depends on the total number of literals and registers required. For each variable i, let m_i be the maximum of the labels that the variable takes in the network representation. Then m_i represents the number of synchronous registers that are connected in cascade at the output of the corresponding gate. An area estimate can be computed as: $A = \alpha \sum_{v_i \in V^G} l_i + \beta \sum_{v_i \in V} m_i$, where α and β are coefficients taking into account the relative area cost of a literal and a register. Given an area bound Ψ, a network is an **area-feasible** implementation if $\Psi \geq A$, and it is a **feasible** implementation if it is both area-feasible and timing-feasible.

[1] A better model of gate delay would include loading effects due to fanout [8]. We neglect this dependence here for the sake of simplicity.

3 Logic transformations in synchronous logic synthesis

The problem of minimizing the area (cycle time) of a synchronous Boolean network implementation, possibly under cycle time (area) constraints, is difficult and no efficient exact solution method is known. Most techniques for multiple-level logic optimization are based on network transformations, that preserve the I/O equivalence of the network, and achieve area/time optimal solutions with respect to some local criterion. Transformations are classified as **local** and **global**. Transformations are said to be local when they modify the representation of a Boolean function at a network vertex at a time (e.g. factoring or Boolean simplification). Such transformations have been presented in [1] [2] for combinational logic synthesis and can be used (without significant extensions) in synchronous logic synthesis, because they do not depend on the network model. Global transformations target more than one vertex at a time and attempt to improve the network by restructuring the global interconnections (e.g. elimination, resubstitution and extraction). We consider here global transformations extended to synchronous logic synthesis in relation with network retiming.

Retiming [5] is a technique that determines a register assignment in a network (i.e. a set of weights in $G(V, E, W)$) so that it is a feasible implementation for a given cycle time Φ, if such an assignment exists. In our context, the retiming of variable i by an integer r corresponds to adding r to its label, and the retimed variable is denoted by $i^{(+r)}$. Similarly, the retiming of an expression $\mathcal{I}$ by an integer r corresponds to adding r to the labels of all its operands and it is represented by $\mathcal{I}^{(+r)}$. The positive (negative) retiming of a gate vertex v_i by r_i is the shift of r_i register delays from its outputs (inputs) to its inputs (outputs). It corresponds to retiming by r_i the expression $\mathcal{I}$ of v_i and to retiming by $-r_i$ the variable i in the expressions of the vertices of $FO(v_i)$. The retiming of an input vertex is just the retiming by $-r_i$ of the variable i in the expressions of the vertices of $FO(v_i)$. The retiming of an output vertex is just the retiming by r_i of the expression $\mathcal{I}$ of v_i. An example is shown in Fig. 2.

Since labels cannot be negative by definition, the retiming of a vertex is valid only for some restricted values of r_i. A retiming of the vertices of a Boolean network is **feasible** for a cycle time Φ, if the retimed network is a timing-feasible implementation with non-negative labels and I/O equivalent to the original network.

Leiserson and Saxe proposed a search technique that finds the minimum Φ for which such an assignment exists [5]. The corresponding network is said to be **optimal with respect to retiming**. If this technique were the only available to optimize the cycle time, then its

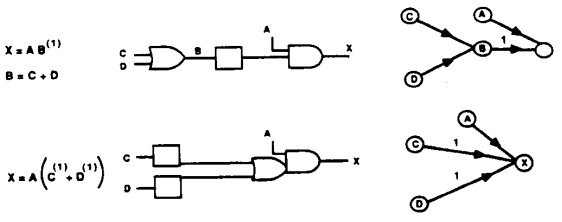

Figure 3: Elimination of vertex v_b.

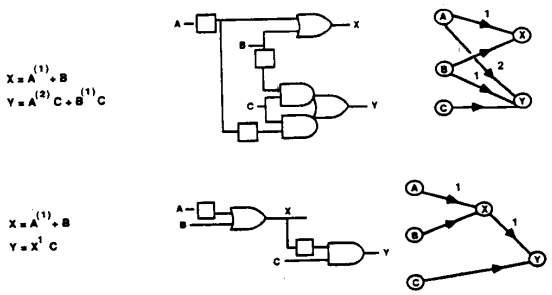

Figure 4: Resubstitution of v_x into v_y.

result would be a global optimum solution. However retiming does not change the structure of the network (i.e. the vertex and edge sets in $G(V, E, W)$), and therefore better results may be achieved by combining it with other transformations that modify the network structure. For this reason we consider here the following transformations.

The **elimination** of a variable with label k is the replacement of the variable by its corresponding expression retimed by k. Given two gate vertices v_i and $v_j \in FI(v_i)$, the elimination of v_j into v_i is the elimination of variable j in all its occurrences in the expression $\mathcal{I}$ for v_i (Fig. 3). The elimination of vertex v_j is its elimination into all the vertices in $FO(v_j)$. Note that the elimination of a variable with label zero is equivalent to the elimination used in combinational logic synthesis [1] [2]. The elimination of a variable with non-zero label corresponds to merging two logic gates that are separated by a register, by shifting the register to the inputs of the gate corresponding to the variable being eliminated.

Let $\mathcal{I}, \mathcal{J}, \mathcal{Q}$ and $\mathcal{R}$ be Boolean expressions. Then $\mathcal{J}$ is a **synchronous divisor** of $\mathcal{I}$ if $\exists r \geq 0$ such that $\mathcal{I} = \mathcal{J}^{(+r)}\mathcal{Q} + \mathcal{R}$ and $\mathcal{J}^{(+r)}\mathcal{Q} \neq 0$. Note that the product $\mathcal{J}^{(+r)}\mathcal{Q}$ may have the algebraic or Boolean flavor, as defined in [1]. Given two gate vertices v_i and v_j such that the expression $\mathcal{J}$ is a synchronous divisor of $\mathcal{I}$, the **resubstitution** of v_j into v_i is the factoring of $\mathcal{I}$ as $j^{(+r)}\mathcal{Q}+\mathcal{R}$. Note again that the divisors defined in [1] are a subset of the synchronous divisors and therefore resubstitution with null retiming (i.e. $r = 0$) is equivalent to resubstitution in combinational logic. The resubstitution of a variable with non-zero retiming corresponds to adding one (or more) register between two gates to simplify the latter (Fig 4).

The **extraction** of a common sub-expression of expressions $\mathcal{I}$ and $\mathcal{J}$ corresponding to two vertices v_i and v_j is the addition to the network of a vertex v_l corresponding to a common synchronous divisor of $\mathcal{I}$ and $\mathcal{J}$ and to the factoring of $\mathcal{I}$ and $\mathcal{J}$ in terms of the new variable l.

4 Algorithms for synchronous logic synthesis

4.1 Retiming

The following algorithm can be used to check whether a synchronous network implementation is feasible for a given Φ. It is derived from an algorithm described in [7] for networks without multiple I/O vertices, and it differs by having the subroutine *set-outputs*, that is not present in the original algorithm. In this paper we are concerned with networks with multiple I/Os, under the assumptions that all inputs are synchronous to the system clock. Such model better conforms to synchronous digital circuits that need to be interconnected among each other. It is important to note that a retiming of an output vertex increases all the path weights from the inputs to that vertex. In this case, if the graph $G(V, E, W)$ is connected, a necessary condition to preserve equivalence is to delay all the other outputs (to keep them in phase with the retimed output) and to recover the delay by subtracting a register delay from all the inputs.

```
retime {
  For (k = 1; k = |V|; k++) {
    Compute t_i for each vertex v_i ∈ V;
    M = {m | t_m > Φ};
    If (M = φ)
      return (TRUE);
    else {
      If (exit) return (FALSE);
      Retime by 1 all vertices in M;
      set-outputs;
    }
  }
}

Set-outputs {
  If ( ∃m ∈ M | v_m is a primary output) {
    Retime by 1 all primary output vertices not in M;
    S = {v ∈ V | ∃ a zero weight path from an input vertex to v};
    Retime by 1 all vertices in S;
  }
}
```

It is obvious that procedure *set-outputs* returns immediately in the case that the network has no primary output explicitly defined, as in [5] [7]. Assume that procedure *exit* returns true when $k = |V|$. The following theorem applies to such networks.

Theorem 1: [7] Given a cycle time Φ, algorithm *retime* returns TRUE iff a feasible retiming exists•

Let us consider now synchronous Boolean networks with multiple I/Os. Assume that procedure *exit* returns true when $k = |V|$. It can be easily noted that when the algorithm returns TRUE, a feasible retiming is constructed by the algorithm such that all the data ready times are bounded by the cycle time. Since every time that a primary output vertex is retimed, all the other outputs and all the inputs vertices are retimed, then the length of all the I/O path is preserved. Finally,

since $t_m > \Phi$ implies $t_i > \Phi\ \forall v_i \in DFO(v_m)$, then the retiming of a vertex implies the retiming of all the vertices on zero-weighted path originating from it as well. Therefore no negative weights (labels) can be introduced. Furthermore it can be shown that no feasible retiming exists if the algorithm returns FALSE.

Theorem 2: For any synchronous Boolean network described by $G(V, E, W)$ and a given cycle time Φ, algorithm *retime* returns TRUE iff a feasible retiming exists•

Proof: To prove the theorem, it is sufficient to note that running algorithm *retime* on any multiple I/O network $G(V, E, W)$ is equivalent to running the same algorithm on a modified network without I/Os. Consider a modified network obtained by merging the input and output vertices into a dummy vertex v_h, with $d_h = \Phi$, and by adding one to the weights of all edges incident to v_h. For any feasible retiming of both networks, the data ready time is the same for each pair of corresponding gate vertices. Indeed a retiming of the modified network cannot remove the synchronous register delays from the dummy vertex v_h to any vertex depending on a primary input and therefore the data ready time of these vertices is preserved. In addition, since any retiming of the modified network does not change the cycle weights in the corresponding graph [5], then all the I/O path weights are preserved in the original network. Therefore a feasible retiming of the modified network co-implies a feasible retiming of the original network. Consider now algorithm *retime*. The retiming of a primary output vertex in the original network corresponds to retiming v_h in the modified network and therefore to retiming all other primary output vertices. In turn, the retiming of v_h causes the retiming of all the vertices in the set S. Therefore running algorithm *retime* on any multiple I/O network is equivalent to running the same algorithm on the corresponding modified network and the claim follows from Theorem 1•

The theorem shows that a feasible retiming can be computed in $O(|V||A|))$ time for general synchronous Boolean networks, because each of the $|V|$ iterations involves the computation of the data ready times, which can be done by topological sort ($O(A)$). In some cases, the algorithm can terminate earlier.

Theorem 3: If at any iteration of the algorithm, $\exists v_m \in M \cap S$ and v_m is a primary output, then no feasible retiming exists•

Proof: In this case, there is a zero weighted path from some input vertex to v_m and $t_m > \Phi$. Since the path weight must be preserved, then t_m cannot be reduced•

This theorem provides an early exit condition which is incorporated into procedure *exit* of algorithm *retime*.

Algorithm *retime* has several advantages over the original retiming algorithm [5]. First, the description of a synchronous Boolean network structure in terms of a (sparse) graph suffices to implement the algorithm. This contrast the requirements for the algorithm in [5], that needs two full square matrices of dimension $|V|$. Second, *retime* is an incremental algorithm, and so it can be applied in connection with network transformations that make small modifications to the network to check feasibility. Circuit transformations affecting the structure of the graph may require local rippling of the registers around the modified area, and in many cases it is likely that the algorithm completes in a number of iterations much smaller than $|V|$.

The algorithm requires the update of the data ready times at each iteration. Note that not all the data ready times need to be recomputed at each iteration. Therefore, the algorithm can be made more computationally efficient by scheduling the set of vertices that are target of the transformation (e.g. M and S). Then the following steps are iterated until the schedule is empty: i) selecting the subset of the scheduled nodes whose direct fanin set is not scheduled; ii) updating their data ready time; iii) scheduling their direct fanout set if the data ready time has changed; iv) deleting them from the list of scheduled vertices.

4.2 Elimination

The *elimination* algorithm follows the outline of that presented in [1] and [2]. Candidate vertices are selected according to some criterion and the elimination takes place if some constraints are satisfied. Elimination terminates when no candidate vertices can be found. We concentrate here on the selection and acceptance criteria for synchronous networks.

Let us consider first the area cost (or value) of an elimination. An elimination changes the total number of literals in a network, say by δ_l. This number can be computed by the formulae given in [1] [2]. When elimination is performed across a register boundary, then it is important to compare the saving in terms of literals with the possible increase of registers. This can be computed as follows. Assume that we are eliminating vertex v_j into vertex v_i. Then, for each vertex $v_k \in FI(v_j)$, let $m_k(\mathcal{I})$ be the maximum label of variable k in the expression $\mathcal{I}$ after the elimination. Similarly, let $m_k(\overline{\mathcal{I}})$ be the maximum label of variable k in all other expressions. Note that $m_k(\overline{\mathcal{I}})$ is not affected by the elimination. Then additional registers are needed to delay variable k if $m_k(\mathcal{I}) > m_k(\overline{\mathcal{I}})$. The total additional registers are: $\delta_r = \sum_{v_k \in FI(v_j)} max(0, m_k(\mathcal{I}) - m_k(\overline{\mathcal{I}}))$. When vertex v_j is eliminated in all $v_i \in FO(v_j)$, then m_j registers are saved and m_j must be subtracted from δ_r. The area cost of an elimination is then $\delta = \alpha\delta_l + \beta\delta_r$. Then, for unconstrained area minimization, candidates are selected to minimize δ, which is required to be less than a threshold usually set to zero.

In the case of area minimization under cycle time constraints, we assume that the network is timing-feasible before the transformation and that it is required to remain such thereafter. In previous work [6], a necessary and sufficient condition for preserving timing-feasibility was shown to be that any increase of the data ready time of any vertex be bounded by its slack. Indeed an elimination of vertex v_j into vertex v_i increases the literal count l_i and it is likely to increase the propagation delay d_i and data ready time t_i. While the sufficiency of the above condition still holds in this setting, its necessity no longer does, because a feasible retiming of the network may exist after the elimination. Therefore any elimination that does not satisfy the above sufficient condition is followed by retiming. If a feasible retiming is not found, then the transformation is rejected. The candidate selection is based on the previous criterion.

Let us consider now the problem of minimizing the cycle time Φ. Let us assume that the network is optimal with respect to retiming (by using the *retime* algorithm for decreasing values of Φ as described in [5] and in [7]) and with respect to elimination within register boundaries (as described in [6] and in [2]). We assume that Φ is the minimum cycle time achieved by these techniques and we address the problem of reducing it by attempting elimination across register boundaries.

In particular, we consider as candidates for elimination the critical vertices whose gate is connected to a register, i.e. at the head of a critical path. Let us assume, for the sake of simplicity that there is only one such candidate, say v_j and that it is critical (i.e. its slack $s_i = 0$ or equivalently its data ready time $t_i = \Phi$). The elimination of such a vertex shortens the critical path and it is beneficial if no other longer critical path is introduced in the circuit. Therefore, to verify the feasibility of the elimination of a candidate vertex v_j, we must consider the increase of data ready time of each vertex $v_i \in FO(v_j)$. If such increases are all strictly bound by the corresponding slack, then the elimination is accepted because there is a cycle time $\Phi' < \Phi$ for which the network is a feasible implementation after the elimination. If the increase of the data ready time at some vertex is not bound by its slack, then the elimination is accepted under the condition that a feasible retiming is found.

Since we would like to speed up the computation time of the elimination algorithm as much as possible, we seek conditions to avoid to retime a network to check feasibility. Consider first the case that the inputs of the gate corresponding to v_i are registers (i.e. $DFI(v_i)$ is empty). Since $t_i = d_i$, its variation is much easier to compute. If, in addition, $t'_i > d_i + d_j$, then the elimination can be rejected outright. Indeed no feasible retiming can be found for $\Phi' < \Phi$ because, if it were so, vertex v_j could be retimed by -1, and then the network would not

(217) 333-4789

be optimal with respect to retiming, as assumed before.

Consider now the case when an elimination is accepted in this context. Then the circuit cycle time can be reduced to the new maximum data ready time Φ'. It is important to know if the network is still optimal with respect to retiming for this new cycle time.

Theorem 4: Given a synchronous network that is optimal with respect to retiming for a cycle time Φ, assume that a vertex v_j with $t_j = \Phi$ is eliminated. If after the elimination $\exists v_k \in FI(v_j)$ such that the maximum data ready time is $t_k = \Phi' < \Phi$, then the network is optimal with respect to retiming for cycle time Φ'•

Proof: Before the elimination, the network is optimal with respect to retiming for a cycle time Φ implies that no feasible retiming exists for a smaller cycle time and that the cycle time is bounded from below by the data ready time of a vertex which is the head of a critical path. Such node was necessarily v_j, because $t_j = \Phi$ and the vertex was unique because after its elimination the maximum data ready time decreases. Since $v_k \in FI(v_j)$, then v_k was on the critical path before the elimination and it becomes the head of the critical path thereafter. Then the critical path does not change, but for v_j. After the elimination, suppose that a feasible retiming exists for a cycle time $\Phi'' < \Phi'$. This would correspond to shortening the path whose head is v_k. But then, such a retiming could have been applied before the elimination, contradicting the assumption of optimality•

The problem of minimizing the cycle time by elimination under area constraints can be approached in a similar way. Candidates for eliminations are selected on the basis of a possible reduction of the cycle time Φ and the acceptance of the transformation is based on the above considerations, as far as timing is concerned, and on the check that the area bound Ψ is not violated. Therefore, an area cost for each elimination is computed and added to the current value of A. If the result is larger that Ψ the candidate elimination is rejected. The area cost may also be useful as a tie rule to chose among candidates yielding the same cycle time reduction.

4.3 Resubstitution

The *resubstitution* algorithm follows the outline of that presented in [1] and [2]. Candidate vertices are selected in pairs, say v_i, v_j, so that the expression $\mathcal{J}$ is a synchronous divisor of $\mathcal{I}$. The resubstitution of v_j into v_i is performed if some constraints are satisfied. The algorithm terminates when no candidate pair can be found. We concentrate here on the selection of synchronous divisors and on the acceptance criteria.

We consider here only algebraic division [1]. Synchronous divisors are searched by iterating algebraic divisions, performed by procedure *alg-div*, described in [1] [2].

```
synchronous-divisors {
   QR = o;
   II = expand(I);
   For (r = 0; ; r++) {
      JJ = expand(J^(l+r));
      If(exit(I, J^(l+r)))return
      QR = alg-div(II, JJ);
   }
}
```

The algorithm stores the quotient and the remainder of the division in $\mathcal{QR}$, which is initialized empty, when non trivial ones are found. Procedure *expand* replaces every variable with non zero label by a new variable. Procedure *exit* returns true if any variable in $\mathcal{J}^{(l+r)}$ has a label larger than the maximum of the labels that the corresponding variable takes in $\mathcal{I}$. If both expressions $\mathcal{I}$ and $\mathcal{J}$ have no labels, the algorithm returns after one iteration and performs just the algebraic division as in [1] [2].

When $\mathcal{QR}$ is empty, the candidates are rejected, because no non-trivial synchronous divisor is found. Otherwise the area cost is computed. Note that the number of registers in the network is affected only for resubstitution across register boundaries (i.e. when $r > 0$). In this case, resubstitution may increase or decrease the number of registers according to the circumstances. For example, when resubstituting v_j into v_i as $i = j^{(l+r)}\mathcal{Q} + \mathcal{R}$, we require r register delays for variable j and r fewer register delays on some inputs to v_i. The total variation in register count δ_r, can be computed from the local variation as follows. Let $V^{JI} \subset V^G$ be the local subset of vertices affected by resubstitution, i.e. $V^{JI} = FI(v_i) \cup v_j$, where the fanin set is computed before the resubstitution. For each vertex $v_k \in V^{JI}$ let $m'_k(\mathcal{I})$ ($m_k(\mathcal{I})$) be the maximum label of variable k in the expression $\mathcal{I}$ after (before) the resubstitution. Similarly, let $m_k(\overline{\mathcal{I}})$ be the maximum label of variable k in the other expressions. Then $\delta_r = \sum_{k \in V} w_k(max(m_k(\overline{\mathcal{I}}), m'_k(\mathcal{I})) - m_k(\mathcal{I}))$. For unconstrained area minimization, the candidates are selected to minimize $\delta = \alpha\delta_l + \beta\delta_r$, as for elimination.

In the case of area minimization under cycle time constraints, the same assumptions and considerations used for elimination apply. Note that a resubstitution of vertex v_j into vertex v_i decreases the literal count l_i and it is likely to decrease its propagation delay d_i. However, the data ready time t_i may depend now on t_j, if $v_j \in DFI(v_i)$ after the resubstitution. In this case ($v_j \in DFI(v_i)$), the transformation can be accepted if the increase in t_i is bounded by the slack s_i. Otherwise, a feasible retiming must be searched for. On the other hand, when a register delay is inserted between v_j and v_j ($v_j \notin DFI(v_i)$), then t_i cannot increase and it is likely to decrease. Then the transformation can be accepted without further checks.

The problem of minimizing the cycle time Φ is analyzed under the same assumptions used for elimination. We assume that the network is optimal with respect to retiming and with respect to resubstitution within register boundaries. We also assume that Φ is the minimum cycle time and we address the problem of reducing it by attempting resubstitution of two vertices, say v_j into v_i across register boundaries. In this case, the data ready time t_i cannot but decrease and t_j remains constant. Then a critical vertex v_i (i.e. with slack $s_i = 0$) and vertices $v_j \in FI(v_i)$ are candidates for resubstitution of v_j into v_i. Candidates are selected to minimize locally the cycle time Φ. Since an upper bound on the decrease of Φ is the variation in propagation delay d_i, this can be used as a quick way of choosing a candidate.

Eventually, the problem of minimizing the cycle time by resubstitution under area constraints can be approached by the same candidate search strategy and by weeding out those candidates that would cause a violation of the area bound Ψ.

5 Minerva

Algorithms *retime*, *elimination* and *resubstitution* have been implemented as a part of program Minerva. Minerva is a computer program to support logic design and optimization of large scale synchronous digital circuits. Circuit specifications can be entered to the program by specifying the circuit equations and interconnection in a hierarchical way. An appropriate format, called SLIF, is used to support the circuit description. Alternatively the circuit can be specified in a Hardware Description Language, HardwareC, that can be compiled into the circuit equations by program Hercules [8]. The output of Minerva is an optimized circuit description in the SLIF format. Minerva can also generate an output compatible with the netlist format of the LSI Logic tools and with the OCT logic view. Minerva is interfaced to the MIS-II program [2], that provides an excellent set of routines for optimizing and mapping combinational sub-components of the circuit being designed. Minerva can isolate these components and interface them with MIS-II in a bidirectional way. Minerva is programmed in C and consists of approximatively 12000 lines of code. Minerva in a part of the Olympus synthesis system developed at Stanford University (Fig 5).

While Minerva can handle a hierarchical circuit description, including specific circuit macros (such as bus drivers, tristate elements, etc.), the algorithms for synchronous logic synthesis are applied, to date, on a flat circuit description. Four different optimization goals, constrained and unconstrained minimization of area and cycle time, correspond to four different strategies of the *elimination* and *resubstitution* algorithms. In the program implementation, care has been taken in avoiding to

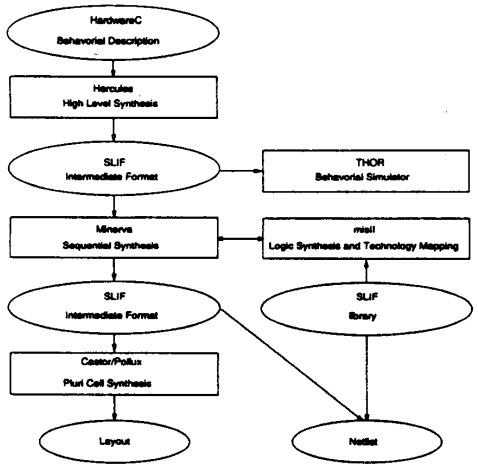

Figure 5: The Olympus synthesis system.

retime a circuit whenever not necessary, to reduce computation time. However, when the desired goal is to minimize the cycle time, then a frequent use of retiming has shown to be beneficial. Algorithm *retime* generally returns true, when it does, in few iterations. Computing time is in the order of few seconds to few minutes on a 1-MIP workstation.

Experimental results on the MCNC benchmarks have shown that these techniques are useful in avoiding some low-valued local optimal solutions that are found by other logic synthesis algorithms applied to the combinational component of the circuit (without registers). Results are heavily dependent on the delay model, and therefore comparisons are useless in the absence of a standardization of the delay parameters. To date Minerva has been used successfully to support the synthesis of three large scale digital designs at Stanford University [9] [10].

6 Concluding remarks and future work

This paper has presented a new approach to the optimal logic synthesis of digital synchronous sequential circuits, based on the concurrent optimization of the circuit equations and the register positions. This method, which combines retiming techniques with network restructuring operations, can achieve results that are at least as good as those obtained by other logic synthesis approaches that separate the combinational logic from the registers. Three algorithms have been studied and implemented in program Minerva.

This research as shown the feasibility of approaching sequential logic design from a global perspective that considers synchronous register delays and gate propagation delays. The choice of realistic delay models for both registers and combinational gates is of pivotal importance. Fanout considerations in the delay model complicate the algorithms for retiming, elimination and resubstitution. An extension of the algorithms to cope with fanout-dependent gate delay is under progress. We are also currently researching the application of other logic synthesis transformations in synchronous logic synthesis, the analysis of multiple clocking phases and the use of different register types.

7 Acknowledgements

This research has been sponsored by NSF and ARPA, under contracts MIP-8710748, MIP-8719546 and N00014-87-K-0828. We would like to acknowledge the stimulating discussions with Andrew Fox and Michiel Ligthart. Frederic Maithot developed the front and back ends of program Minerva.

References

[1] R.Brayton, "Algorithm for Multilevel Synthesis and Optimization" in G.De Micheli, A.Sangiovanni-Vincentelli and P.Antognetti, Editors, *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, Martinus Nijhoff, 1987.

[2] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang "MIS: A Multiple-Level Logic Optimization System", *IEEE Transactions on CAD/ICAS*, Vol. CAD-6, No. 6, November 1987, pp. 1062-1081.

[3] J.Darringer, D.Brand, J.Gerbi, W.Joyner and L.Trevillyan, "LSS: A System for Production Logic Synthesis", *IBM Journal of Res. and Dev.*, Vol 28, No 5, pp. 537-545, Sep 1984.

[4] K.Bartlett, W.Cohen, A.De Geus and G.Hachtel, "Synthesis and Optimization of Multilevel Logic under Timing Constraints" *IEEE Transactions on CAD/ICAS*, Vol CAD-5 No. 4, pp.582-596, Oct. 1986.

[5] 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.

[6] G. De Micheli, 'Performance-oriented synthesis in the Yorktown Silicon Compiler', *IEEE Trans on CAD/ICAS*, Vol CAD-6, NO 5, Sept' 1987, pp.751-765.

[7] J.Saxe "Decomposable Searching Problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures" *Ph. D. Dissertation*, Department of Computer Science, Carnegie Mellon University, 1985.

[8] G. De Micheli, D. Ku "HERCULES - A system for High- Level Synthesis", *Proceedings of 25th Design Automation Conference*, Anaheim, pp. 483-488, 1988.

[9] M.Ligthart, A.Bechtelstein, G.De Micheli and A.El Gamal "Design of a Digital Audio Input Output chip", *Proceedings of the Custom Integrated Circuit Conference*, San Diego, 1989.

[10] V.Rampa and G.De Micheli, "Computer Aided Synthesis of a Discrete Cosine Transform Chip", *Proceedings of the International Symposium on Circuits and Systems*, Portland, 1989.

(217) 333-4789

Further information can be obtained from **Dr. W. Kenneth Jenkins.**