Automatic Synthesis of Large Telescopic Units Based on Near-Minimum Timed Supersetting

L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino

Abstract—In high-performance systems, variable-latency units are often employed to improve the average throughput when the worst-case delay exceeds the cycle time. Traditionally, units of this type have been hand-designed. In this paper, we propose a technique for the automatic synthesis of variable-latency units that is applicable to large data-path modules. We define and study an optimization problem, *timed supersetting*, whose solution is at the kernel of the procedure for automatic generation of variable-latency units. We contribute a new algorithm for solving timed supersetting in the most difficult case, that is, when the timing behavior of the circuit is expressed through an accurate delay model. The proposed solution overcomes the computational limitations of previous approaches and its robustness is experimentally demonstrated by obtaining high-throughput, variable-latency implementations for all the largest circuits in the Iscas '85 and Iscas '89 benchmark suites, as well as for some realistic, high-performance arithmetic units.

Index Terms—Logic synthesis, timing analysis, throughput optimization.

1 Introduction

As performance constraints become tighter, it is increasingly difficult to speed up combinational units simply by reducing their critical path delays. Variable-latency units (i.e., circuits that take a variable, integer number of clock cycles to complete a computation) are frequently used in high-throughput systems to achieve good *common-case* performance even when the worst-case delay cannot be accommodated within the cycle time. Floating-point arithmetic units [1] are typical examples of circuits of this kind.

The hand-crafted design of variable-latency units is a difficult task. In this paper, we propose a method for automatically transforming a unit with a fixed one-cycle latency into a variable-latency unit with increased average throughput. We call *telescopic unit* the product of our transformation because the new circuit can "stretch" the number of cycles required for the completion of a computation, depending on the input values.

The throughput optimization paradigm based on telescopic units can be summarized as follows. We start from a single-cycle fixed-latency unit, defined as the logic between two sets of latches. The minimum allowable cycle time, T, of the unit, is equal to its longest delay. We specify a reduced target cycle time $T^* < T$. Then, the input patterns for which the propagation of the input values through the original logic takes longer than T^* are identified. Whenever the unit receives one of these patterns, it completes execution in two clock cycles, otherwise, it completes in one clock. As a last step, a combinational block is automatically synthesized and added to the original unit. The task of such block is to generate a handshaking signal, the *hold signal* f_h, whose value informs the environment when the final result is available at the outputs of the unit.

Average throughput is increased if the number of input patterns for which the unit requires two clock cycles to complete execution is small. If this is the case, the unit will almost always compute a new result in $T^* < T$. In the following, we will show how to determine a bound on the maximum probability of long-propagating input patterns. Another important condition for the applicability of the technique is to control the area, timing, and power overheads caused by the hold function.

A procedure for the automatic synthesis of telescopic units has been proposed in [2]. The main limitation of such procedure is its high computational cost. The algorithmic core which is at the basis of the transformation is a symbolic routine that exploits the expressiveness of algebraic decision diagrams (ADDs) [3] to perform exact circuit timing analysis [4]. Unfortunately, when a complex and realistic model is adopted to describe the gate delays, the algorithm becomes highly memory and time consuming. Therefore, it is usable only for small circuits, i.e., a few hundreds of gates. To partially alleviate this problem, one may resort to a simpler delay model, e.g., the unit delay model. Even in this case, however, the wall of a few thousand gates can hardly be broken, as demonstrated by the experimental data reported in [2]. This is not surprising, since the ADD-based method exactly solves the *false path* problem, which is known to be NP-complete [5], independently on the selected delay model.

We propose a new algorithm for the automatic construction of telescopic units that overcomes the computational bottleneck of the ADD-based synthesis procedure. We move from the observation that the automatic generation of telescopic units entails the solution of a general problem that we call *timed supersetting* (*TS* for brevity): *Find a set of input conditions that include all values propagating to the outputs with delay longer than a given* T^*. We study the

- *L. Benini and G. De Micheli are with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305.*
- *A. Lioy, E. Macii, G. Odasso, and M. Poncino are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy 10129. E-mail: enrico@athena.polito.it.*

Manuscript received 29 June 1998; revised 6 May 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 107084.

properties of TS and explore its relationship with classical results in the field of timing analysis.

The theoretical investigation leads to the implementation of a core optimization engine that replaces the computationally expensive ADD-based method and enables the synthesis of the hold function for large units (several thousand gates) even when a complex gate delay model is adopted. The technique of [2] computes the true propagation delay for each input pattern; hence, it allows us to determine the *minimum set of patterns* that solves TS. In contrast, the algorithm of this paper finds a nonminimum solution to TS. Such solution is *conservative*, that is, it always includes the minimum one.

The downside of the conservative solution is that the hold logic may be activated for patterns that do not actually violate the cycle time constraint. Thus, the telescopic unit may operate with an average throughput that is inferior to what could theoretically be achieved. Nevertheless, the advantages overcome the limitations since the new algorithms for the solution of TS are practical for much larger and more complex units, such as those that can be found in high-performance microprocessors or DSPs.

Throughout the paper, the knowledgeable reader may observe the relationship between TS and the timing analysis problem (i.e., finding the true longest delay of a circuit and a pattern that exercises it). Indeed, a minimum solution to TS and the true longest delay of a circuit can be found by the same ADD-based algorithm; on the other hand, approximate timing analysis methods cannot be directly used for solving TS.

The procedure for automatically synthesizing telescopic units which encompasses the TS solution algorithms of this paper has been benchmarked on the largest Iscas '85 [6] and Iscas '89 [7] examples. Results are satisfactory since an average throughput improvement of 14.1 percent has been achieved at the price of a 6.9 percent average area overhead. In addition, the viability of the presented throughput optimization paradigm has been demonstrated by applying it to real-life, high-performance arithmetic units.

The remainder of this manuscript is organized as follows: Section 2 provides the basic terminology related to timing analysis that will be used throughout the paper. It also recalls the definitions of throughput and latency of a unit, as well as those of some Boolean operators that will be exploited by the algorithms presented in subsequent sections. Section 3 introduces the telescopic units, and briefly summarizes the synthesis procedures proposed in [2]. In Section 4, we formally state the timed supersetting problem and we propose an approximate, yet accurate, algorithm for its solution that can be fruitfully applied for automatically synthesizing large telescopic units. Section 5 reports the experimental results and Section 6 closes the paper with some concluding remarks.

2 Background

2.1 Circuits and Delays

A *combinational circuit* is a feedback-free network of combinational logic gates. If the output of a gate, g_i, is connected to an input of a gate, g_j, then g_i is a *fanin* of g_j and gate g_j is a *fanout* of gate g_i. A *controlling value* at a gate input is the value that determines the value at the output of the gate independent of the other inputs, while a *noncontrolling value* at a gate input is the value whose presence is not sufficient to determine the value at the output of the gate.

Each gate, g, is associated with two delays, $d_r(g)$, rise delay, and $d_f(g)$, fall delay. The *delay function* of gate g is called $d(g, \mathbf{x})$. It equals $d_r(g)$ if g takes value 1 when input vector $\mathbf{x} = [x_1, x_2, \ldots, x_{ni}]$ is applied to the primary inputs of the circuit. Otherwise, $d(g, \mathbf{x}) = d_f(g)$.

Given a gate g, the *arrival time*, $AT(g, \mathbf{x})$, is the time at which the output of g settles to its final value if the primary input vector $\mathbf{x}$ is applied at time 0. Given a maximum delay constraint, the *required time*, $RT(g, \mathbf{x})$, is the time at which the output of gate g is required to be stable when the primary input vector $\mathbf{x}$ is applied in order for the output to stabilize within the maximum allowed delay. The *slack*, $ST(g, \mathbf{x})$, of a gate g is the difference between its required time and its arrival time, i.e., $ST(g, \mathbf{x}) = RT(g, \mathbf{x}) - AT(g, \mathbf{x})$.

A *path* in a combinational circuit is a sequence of gates, $(g_1, \ldots, g_m)$, where gate g_i is in the fanin of gate g_{i+1}. The *length* of a path, $\mathcal{P} = (g_1, \ldots, g_m)$ is defined as:

$$d(\mathcal{P}, \mathbf{x}) = \sum_{i=1}^{m} d(g_i, \mathbf{x}).$$

An *event* is a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ at a gate. Given a sequence of events, $(e_1, \ldots, e_m)$, occurring at gates $(g_1, \ldots, g_m)$ along a path such that e_i occurs as a result of event e_{i-1}, the event e_0 is said to *propagate* along the path. Under a specified delay model, a path $\mathcal{P} = (g_1, \ldots, g_m)$ is said to be *sensitizable* if an event e_1 occurring at gate g_1 can propagate along $\mathcal{P}$. A *false path* is a nonsensitizable path. The *critical path* of a combinational circuit is the longest sensitizable path under a specified delay model: Its worst-case length, over all input conditions, is the delay, D, of the combinational circuit and it is a lower bound on the cycle time T, i.e., $D \leq T$. For the sake of simplicity, we neglect set-up and hold times, and propagation delays through registers. These factors can be easily incorporated into our analysis and synthesis technique.

Topological approximations to arrival times ($AT(g)$), required times ($RT(g)$), slacks ($ST(g)$), and path lengths ($d(\mathcal{P})$) can be computed through graph algorithms [8] whose complexity is linear in the number of gates involved. Such approximations have two properties: They are conservative and pattern-independent, that is, the following inequalities hold for all possible input vectors $\mathbf{x}$:

$$\begin{aligned} AT(g) &\geq AT(g, \mathbf{x}) \\ RT(g) &\leq RT(g, \mathbf{x}) \\ ST(g) &\leq ST(g, \mathbf{x}) \\ d(\mathcal{P}) &\geq d(\mathcal{P}, \mathbf{x}). \end{aligned}$$

The *topological critical path* of the circuit is the path with longest topological length.

2.2 Throughput and Latency

The *throughput* P of a unit is defined as the amount of computation (i.e., the number of times a new output value

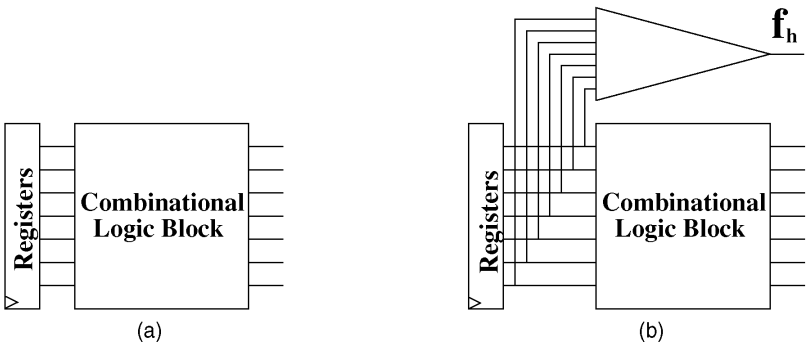


Fig. 1. (a) A combinational unit and (b) a telescopic unit.

is produced) carried out per time unit. The *latency*, L, of a digital system is defined as the number of clock cycles required for a computation to complete. A fixed-latency unit with latency L clocked with period T has constant throughput, given by:

$$P = \frac{1}{LT}.$$

For variable-latency units, we consider the *average* latency L_{ave} over a period of time, $T_{tot} >> T$. The average throughput is simply:

$$P_{ave} = \frac{1}{L_{ave}T}.$$

In the following sections, we use the shorthand notation L and P, as opposed to L_{ave} and P_{ave}, to denote average latency and throughput, respectively.

2.3 Boolean Functions and Operators

We assume the reader to be familiar with the basic concepts of Boolean functions. In this section, we only review two Boolean operators which are essential for our purposes. Let $\mathbf{x} = [x_1, x_2, \ldots, x_{ni}]$ be a vector of Boolean variables. Given a single-output Boolean function, $f(\mathbf{x})$, the *positive* and the *negative cofactors* of f, with respect to variable x_i, are defined as:

$$f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{ni})$$

and

$$f_{x_i'} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{ni})$$

The *existential abstraction* of f with respect to x_i is defined as:

$$\exists_{x_i} f(\mathbf{x}) = f_{x_i} + f_{x_i'}.$$

The *Boolean difference* of f with respect to x_i is defined as:

$$\frac{\partial f(x)}{\partial x_i} = f_{x_i} \oplus f_{x_i'}.$$

3 Telescopic Units

Consider the problem of increasing the throughput of a combinational unit, such as the one shown in Fig. 1a. This can be done by shortening the cycle time of the unit from its original value, T, to $T^* < T$. One possible way of ensuring functional correctness is to extend the unit to provide an additional output signal, f_h, which is asserted for all input patterns requiring more than T^* time units to propagate to the outputs of the block (see Fig. 1b).

We call *telescopic unit* the modified unit since it may require $L_{max} > 1$ cycles to complete its execution, depending on the specific patterns appearing at its primary inputs. We consider here the situation in which $L_{max} = 2$. In this case, the computation completes in T^* time units for patterns such that $f_h = 0$ and in $2T^*$ time units for patterns such that $f_h = 1$.

3.1 Conditions for Throughput Improvement

The average throughput of the original unit is given by:

$$P = \frac{1}{T}.$$

Conversely, for the telescopic unit, the lower the probability of the *hold* signal, f_h, to take on the value 1, the larger the overall throughput improvement. In fact, its average throughput, P^*, is given by:

$$P^* = \frac{Prob(f_h)}{2T^*} + \frac{1 - Prob(f_h)}{T^*},$$

where $Prob(f_h)$ is the probability of the *hold* signal being one.

The use of the telescopic unit is therefore advantageous only for some values of T^* and $Prob(f_h)$, i.e., when $P^* > P$. In particular, we have the following condition for throughput improvement:

$$Prob(f_h) < \frac{2(T - T^*)}{T}.$$

It should be noticed that the inequality above is valid only for $T^* \geq T/2$ since we have made the assumption that $L_{max} = 2$.

The extension to $L_{max} > 2$ is conceptually straightforward, but more complex to implement. This is because several hold signals $f_h^1, f_h^2, \ldots, f_h^k$ are required to make the unit work correctly. Function f_h^k is one for the input patterns that require $k + 1$ cycles to complete execution.

The expression for P^* can obviously be modified to account for values of $T^* < T/2$; in other words, when

$L_{max} > 2$, the formula that gives the value of the average throughput of the telescopic unit becomes:

$$P^* = \frac{Prob(f_h^k)}{(k+1)T^*} + \frac{Prob(f_h^{k-1})}{(k)T^*} + \ldots \frac{Prob(f_h^1)}{2T^*} + \frac{1 - Prob(f_h^1 + f_h^2 + \ldots f_h^k)}{T^*},$$

where $Prob(f_h^j)$ represents the probability that the unit completes the execution in a time frame between jT^* and $(j+1)T^*$. For clarity, in the remainder of this work, we focus on the case $L_{max} = 2$.

In order to automatically synthesize telescopic units, two problems must be solved. First, the hold function (that is, a combinational logic function that detects *all* input patterns that propagate to the outputs with delay larger than T^*) must be computed and synthesized. Second, the controller of the data-path where the telescopic unit is instantiated must be modified since it must be able to synchronize the environment with the telescopic unit by delaying subsequent computations when $f_h = 1$. The following section provides some details on the procedures we have proposed in [2] to synthesize function f_h. Controller redesign techniques are not discussed here since they are beyond the scope of this paper (the topic is extensively addressed in [2]).

3.2 Synthesis of the Hold Logic

The synthesis of the hold logic critically depends on the capability of finding all input patterns that propagate to the outputs with delay larger than T^*. Such patterns must be included in the ON-set of function f_h. In the next section we analyze this problem in detail. Here, we assume the availability of a black-box procedure, `ComputeF_h`(C, T^*) that returns the ON-set of f_h. The input parameters of such procedure are the initial specification of the unit, C, and the desired cycle time, T^*.

`ComputeF_h` solves the timed supersetting problem that was informally introduced in Section 1. In fact, the minimum solution of TS is the ON-set of the hold function f_h that contains *all and only those* input values that propagate to the outputs of the unit with a delay longer than T^*. Ideally, we would like to implement a hold logic that takes value 1 exactly for the input values corresponding to the ON-set of the hold function. In this way, the unit would require two cycles to complete only for patterns that do propagate to the output in a time longer than T^*. Conversely, we must guarantee that the implementation of the hold logic itself has a delay shorter than T^* and this may not be always possible. Thus, the target is to determine an *enlarged hold function*, $f_h^e \geq f_h$ such that the average performance of the unit is only marginally degraded, but the implementation of f_h^e meets the timing constraint, T^* and has a well-controlled area and power dissipation.

The heuristics devised in [2] for synthesizing the enlarged function f_h^e starts from the BDD representation of f_h. It generates the hold logic following an iterative paradigm. First, the BDD of f_h is mapped onto a multiplexor network. Then, the network is optimized through traditional synthesis techniques; finally, a check is made to find out if the timing constraint $T(f_h) < T^*$ is met. If this is not the case, the ON-set of f_h is enlarged, to obtain f_h^e, by properly removing some BDD nodes and the process is repeated.

4 The Timed Supersetting Problem

In this section, we formally state the timed supersetting (TS) problem and one important variation, called *minimum timed supersetting* (MTS). The practical relevance of TS and MTS for the synthesis of telescopic units has been outlined in the previous section. For the sake of comparison, we briefly describe the algorithm for the solution of MTS (and TS) presented in [2]. We then take a completely different approach and present the key contribution of this paper, namely a robust and widely applicable algorithm for the solution of TS.

Consider a combinational circuit C with primary inputs $\mathbf{x} = [x_1, \ldots, x_{ni}]$ and outputs $\mathbf{o} = [o_1, \ldots, o_{no}]$. The timed supersetting problem can be formally stated as follows:

Problem 1. *Find a set S of input values $\mathbf{x}$ that includes all values which propagate to the outputs $\mathbf{o}$ with a delay larger than or equal to a given T^*.*

Obviously, TS has always the trivial solution B^{ni}, i.e., the complete Boolean space is guaranteed to include all input values with propagation delay larger than T^*. We are interested in nontrivial solutions of TS. A theoretically relevant solution is the minimum one. The minimum timed supersetting problem consists of finding the smallest set of input values with propagation delay larger than T^*. Formally:

Problem 2. *Find the set S_{min} of all and only those input patterns $\mathbf{x}$ which propagate to the outputs $\mathbf{o}$ with a delay larger than, or equal to a given T^*.*

It is quite easy to prove the NP-completeness of MTS. Solving MTS when T^* is equal to the longest propagation delay of C is at least as hard as finding a single pattern with maximum propagation delay. This problem is NP-complete [5].

Observe that $S_{min} \subseteq S$, i.e., every solution of TS is guaranteed to contain the solution of MTS. Among the solutions of TS, we are interested in *near-minimum* solutions. In more detail, we are looking for approximations S of S_{min} that:

1. Include S_{min};
2. Are as close as possible to S_{min};
3. Can be computed in polynomial time and space (in *ni*).

Before discussing our approximation strategy, we review an algorithm for the exact solution of MTS.

4.1 Exact MTS Solution

An ADD-based algorithm for the exact solution of MTS has been presented in [2]. The arrival time ADD for each output o_i of the circuit is first computed using the algorithm of [4]. Such ADDs provide the propagation delay for any possible input vector. The logic function $f_h^{o_i}(\mathbf{x})$, which assumes the value 1 for all input vectors for which the arrival time of o_i

is greater than the desired cycle time T^*, is then obtained through symbolic ADD operations. Finally, function $f_h(\mathbf{x})$ collecting the set of input conditions for which *at least* one circuit output o_i has an arrival time greater than T^* is computed by OR-ing together all functions $f_h^{o_i}$.

The main limitation of the algorithm is its worst-case exponential time and space complexity. When a complex, load- and path-dependent delay model is used, it is impossible to build the arrival time ADDs even for the outputs of relatively small circuits. The memory requirements for such construction are simply excessive. Another shortcoming of this approach is that, when building the delay ADDs, complete delay information is computed, even for patterns that propagate much faster than T^*. The computation of unneeded information contributes to the memory blow-up problem.

4.2 Near-Minimum TS Solution

Since the exact solution of MTS is computationally infeasible for large circuits, we resort to algorithms that only solve TS but attempt to find solutions which are as close as possible to the minimum one. Notice the analogy with the approaches used in timing analysis. When exact delay computation is unaffordable, it is possible to resort to safe approximations with various degrees of tightness.

Consider a combinational circuit C with primary inputs $\mathbf{x} = [x_1, \ldots, x_{ni}]$. A gate, g_i, of the network is associated with a Boolean function $f_i(\mathbf{y})$, where $\mathbf{y} = [y_1, \ldots, y_{n_{g_i}}]$ is the *local support* of f_i. We call $F_i(\mathbf{x})$, the Boolean function associated with gate g_i expressed as a function of the primary inputs (*global support*).

Let us assume that topological timing analysis has been performed, and that the topological critical path $\mathcal{P} = (g_1, g_2, \ldots, g_m)$ has been determined. Let us assume also that the topological length of the critical path, T_c, violates the desired cycle time, namely $T_c > T^*$. Since we are relying only on topological delay analysis, we conservatively consider the path as a true one. Consequently, all input conditions that activate it must be in the ON-set of the hold function f_h.

To find such conditions, from the primary inputs, we move along the critical path toward the output. We call *critical input* y_c of a gate g_i on the critical path the input which connects it with gate g_{i-1}. For each gate in the path, we specify the *local sensitization function* s_i as the Boolean function that takes on the value 1 when gate g_i is sensitive to the value of y_c:

$$s_i(\mathbf{y}) = \frac{\partial f_i(\mathbf{y})}{\partial y_c}. \tag{1}$$

Note that, given $s_i(\mathbf{y})$ for a gate g_i, the *global sensitization function* S_i (that represents the sensitization conditions for gate g_i) can be expressed as a function of the primary inputs $\mathbf{x}$ through recursive backward substitution of the local support variables until the primary inputs are reached.

The sensitization conditions for the entire path, $\mathcal{P}$, can now be computed as the intersection of all sensitization conditions of the gates $g_1, \ldots, g_m$ in $\mathcal{P}$. In formula:

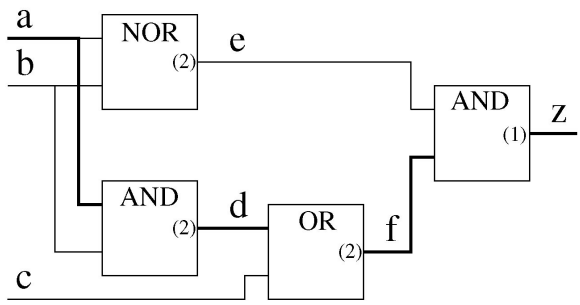


Fig. 2. Example of computation of the sensitization condition.

$$S_{crit}(\mathbf{x}) = \prod_{i=1}^{m} S_i(\mathbf{x}). \tag{2}$$

We call S_{crit} the *path sensitization conditions*. This formula holds because, for a signal to propagate along a path, all gates on the path must be sensitized. We call *partial path sensitization conditions* $S_{crit,j} = \prod_{i=1}^{j} S_i$ (with $j \leq m$) the path sensitization conditions for gates belonging to the path up to level j. Clearly, $S_{crit,m} = S_{crit}$. The partial sensitization conditions can be computed with the following recursion, for $j = 1, \ldots, m$:

$$\begin{array}{rcl} S_{crit,j}(\mathbf{x}) & = & S_{crit,(j-1)}(\mathbf{x}) \cdot S_j(\mathbf{x}) \\ S_{crit,1}(\mathbf{x}) & = & S_1(\mathbf{x}). \end{array} \tag{3}$$

A property of (3) is that $S_{crit,j} \leq S_{crit,k}$ for each $j > k$, that is, the $S_{crit,j}$s are *monotonically* decreasing (i.e., the ON-set of S_{crit} is monotonically shrinking) with increasing j. Notice that computing the complete S_{crit} is equivalent to testing the viability of path $\mathcal{P}$. Since this problem is NP-complete, there will be instances for which this computation requires an exponential amount of time or resources. However, the key observation is that we do not have to compute the complete S_{crit} to find a conservative set of input conditions for which the circuit delay T_c violates the timing constraint T^* (i.e., the hold function f_h). Any $S_{crit,j}$ is suitable for that purpose, because its ON-set contains the one of S_{crit}.

Example 1. Consider the circuit of Fig. 2 (taken from [5]). The delay of the gates are shown within parentheses. The topological length of path $\mathcal{P} = (a, d, f, z)$ is $d(\mathcal{P}) = 5$ and we compute $S_{crit}(\mathbf{x})$ for such path. For the first gate in the path, $S_{crit,1} = b$. For the second gate, $S_{crit,2} = b \cdot c'$, while, for the third gate, $S_{crit,3} = b \cdot c' \cdot e = b \cdot c' \cdot a' \cdot b' = 0$. Hence, path $\mathcal{P}$ is not sensitizable.

Although it may appear that (3) provides a viable procedure for finding a near-minimum solution of TS, two major problems need to be addressed:

1. The sensitization conditions of (1) are *static*, that is, it does not consider the dynamic propagation of the events along the paths. It is a well-known fact that the absence of static path sensitization conditions (i.e., $S_{crit} = 0$) is *not* sufficient to guarantee that a path does not propagate events with delays that violate the timing constraint T^*. This phenomenon is known as *dynamic* sensitization [5]. Notice that every valid solution to TS must include all patterns with

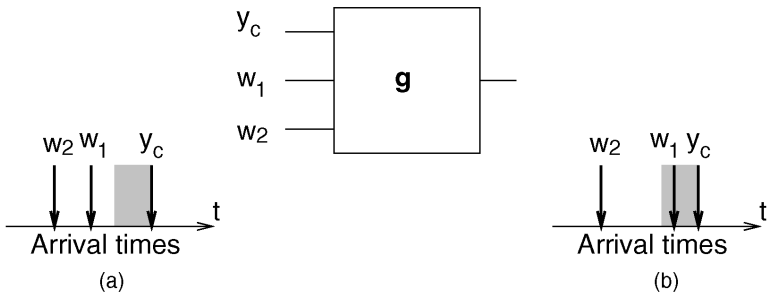


Fig. 3. Arrival times distribution for a gate on the critical path.

propagation longer than T^*. Hence, the approach based on simple static sensitization may lead to incorrect implementations and must be augmented by some form of dynamic sensitization test.

2. Equation (3) has been obtained under the assumption that there is only one path violating the timing constraint. This is not generally true. In almost all practical examples, there are multiple critical paths. Moreover, the number of such paths can be exponential in the number of gates in the network. The complexity explosion caused by the number of critical paths must be addressed in a conservative fashion.

In the next two sections, we analyze and solve the above problems. We then describe in detail our strategy for finding a near-minimum TS solution in an efficient and robust way.

4.2.1 Accounting for Dynamic Sensitization

In order to derive sensitization conditions which are correct and conservative, let us consider a gate g_i on a critical path $\mathcal{P}$ (i.e., a path whose topological length exceeds T^*). Let $AT(y_c)$ be the topological arrival time at the critical input and $ST(g_i)$ the (negative) slack of the gate. Let $W = \{w_1, \ldots, w_p\}$ denote the set of side inputs, and $AT(w_i)$, $i = 1, \ldots, p$ their topological arrival times. We present the following safe conditions for declaring a path as false.

Theorem 1. *Given the topological arrival times, required times, and slacks for all gates belonging to path $\mathcal{P}$, the static sensitization conditions of (1) are correct if, for all gates g_i of $\mathcal{P}$:*

$$(AT(y_c) + ST(g_i)) > AT(w_i), \quad \forall w_i \in W. \tag{4}$$

Proof. The topological arrival time is an upper bound to the actual arrival time; then, the topological slack is always more conservative (i.e., smaller) than the actual slack. Hence, all transitions on y_c that take place before time $T_{min} = AT(y_c) + ST(g_i)$ cannot arrive late at the outputs. If all side inputs w_i are early enough and stabilize before T_{min}, any transition on y_c that could arrive late at the outputs does find the side inputs already stable at their final value. Thus, static sensitization can be used to assess if the values of the side inputs filter out the propagation on the critical path. □

Theorem 1 states that, if the condition of (4) holds for all gates in the path and $s_i(\mathbf{y}) = \frac{\partial f_i(\mathbf{y})}{\partial y_c} = 0$, then the path $\mathcal{P}$ is surely false. If (4) does not hold for some side inputs of a gate g_i, the static sensitization conditions of the gate cannot be used for computing the path sensitization conditions. For a generic gate g, shown in Fig. 3, the arrival times of the inputs are shown with vertical arrow. The shaded area represents the (negative) slack of the critical input y_c. With the arrival times distribution of Fig. 3a, the static sensitization condition is valid for gate g. The condition is not valid for the arrival times in Fig. 3b. Noncritical input w_1 violates the constraint expressed by (4).

This criterion may be extremely conservative in some cases because it prevents us from using the sensitization conditions for a gate g_i if any of its side inputs do not satisfy (4). In the vast majority of cases, only some of the inputs violate the inequality. When the inputs that satisfy the inequality have controlling value, the gate still filters out events on the critical input. Therefore, we can relax the conditions stated by (1). This can be done by exploiting some of the results available in the literature on timing analysis.

A well-known criterion which is particularly suitable for a BDD-based symbolic implementation is the one introduced by Brand and Iyengar [11]. In that work, the sensitization conditions (1) are overestimated by *abstracting* a set of the local gate inputs.

The key point with this approach is selecting which and how many inputs should be abstracted. Inequality (4) provides the criterion to do that. If we call $\widetilde{W} = \{\widetilde{w}_1, \ldots, \widetilde{w}_k\} \subseteq W$ the set of side inputs that do not satisfy (4), the comprehensive criterion for robustly and correctly detecting a false path, at a gate g_i, becomes:

$$\sigma_i(\mathbf{y}) = \exists_{\widetilde{W}} \left(\frac{\partial f_i(\mathbf{y})}{\partial y_c} \right). \tag{5}$$

Similarly to the static sensitization conditions, we can extend $\sigma_i(\mathbf{y})$ to the global support of the circuit, and compute $\Sigma_i(\mathbf{x})$ as a function of the primary inputs $\mathbf{x}$. The sensitization conditions for the entire path are then given by the intersection of all sensitization conditions of the gates $g_1, \ldots, g_m$. In formula:

```
procedure Compute_Sigma (P) {
  Σ_crit(x) = 1;
  foreach (gate g_i ∈ P) {
    s_i(y) = ∂f_i(y)/∂y_c;
    foreach (side input w_i of g_i)
      if ((AT(y_c) + ST(g_i)) ≤ AT(w_i))
        s_i(y) = ∃_{w_i} s_i(y);
    σ_i(y) = s_i(y);
    Σ_i(x) = Extend(σ_i(y), x);
    Σ_crit(x) = Σ_crit(x) · Σ_i(x);
  }
  return (Σ_crit(x));
}
```

Fig. 4. Algorithm for Computing $\Sigma_{crit}(\mathbf{x})$.

$$\Sigma_{crit}(\mathbf{x}) = \prod_{i=1}^{m} \Sigma_i(\mathbf{x}). \tag{6}$$

In summary, the correct and conservative sensitization conditions $\Sigma_{crit}(\mathbf{x})$ for a single path $\mathcal{P}$ are obtained through the pseudocode of Fig. 4.

Procedure `Extend` expresses the sensitization conditions in terms of the global support of the circuit, $\mathbf{x}$. Inequality (4) is used to decide which side inputs arrive too late and should be quantified out from the sensitization conditions. Clearly, the procedure does not solve MTS exactly since conservative and pattern-independent topological estimates of the arrival times and slacks are used. In other words, (4) is a sufficient, but not necessary, condition for deciding whether a side input stabilizes before the critical input arrives.

Example 2. Consider again the circuit of Fig. 2. In Example 1, we have found that the static sensitization conditions for critical path $\mathcal{P} = (a, d, f, z)$ is null (i.e., $S_{crit} = 0$). However, the path is not false. This can be verified by inspection of the circuit: The output stabilizes after $T = 5$ time units when the inputs have the following transitions: $a : 1 \rightarrow 0$, $b : 1 \rightarrow 0$, and $c : 0 \rightarrow 0$. Hence, the circuit is a counter-example that proves the insufficiency of static sensitization to solve TS.

We set the required time to $T^* = 4$. The arrival time of the inputs is 0. The arrival times of the outputs of the gates are: $AT(d) = 2$, $AT(f) = 4$, $AT(e) = 2$, and $AT(z) = 5$. The slacks on the critical inputs are: $ST(f) = -1$, $ST(d) = -1$, and $ST(a) = -1$.

Now, we apply our technique for computing the path sensitization conditions Σ_{crit}. We start with the NOR gate with output d. The critical input is a and $\partial d/\partial a = b$. Inequality (4) is not satisfied for side input b because $AT(a) + ST(a) = -1$ and $AT(b) = 0$. Thus, we must quantify out b from the sensitization conditions of the gate: $\exists_b(\partial d/\partial a) = 1$. Inequality (4) is satisfied for the remaining two gates on the critical path. They contribute to the path sensitization conditions with $\partial f/\partial d = c'$ and $\partial z/\partial f = e$. The final path sensitization conditions are $\Sigma_{crit} = 1ec' = a'b'c'$, which are obviously not null.

4.2.2 Dealing with Multiple Paths

So far, we have described a robust, yet simple, algorithm for finding a near-minimum TS solution which is applicable to the cases where a single critical path is present in the circuit. In this section, we present an algorithm (see the pseudocode in Fig. 5) to find a near-minimal solution to TS (i.e., the hold function f_h for a telescopic unit) in the case of multiple critical paths.

The procedure receives, as inputs, circuit C and the desired cycle time T^*, given as an absolute time value or as a percentage of the actual critical delay. It initially performs (Line 1) static timing analysis, computing arrival times, required times, and slacks for each gate. Then, the network is levelized (Line 2), that is, the gates are grouped into the list $Levels[]$ according to their topological level, starting from the primary inputs, which are assumed to be at level 0. Starting from level 0, the critical gates (i.e., gates with negative slack) at each level are processed (Line 4), and a Boolean function $PAF(\mathbf{x})$ (*Path Activation Function*) is computed as follows: At each gate, the function is obtained by summing, over its critical fanins, the product of two quantities: 1) The path activation function of the ith fanin ($PAF_i(\mathbf{x})$ in Line 7); 2) The sensitization conditions $\Sigma_i(\mathbf{x})$, computed with (6). Clearly, the *PAF* for each primary input is assumed to be 1. The output of the procedure is a near-minimum solution of TS or, equivalently, the hold function f_h of a telescopic unit. It is computed in Lines 8 and 9, by accumulating the *PAF*s of all critical gates that are connected to an output.

The rationale of the algorithm is that every critical gate filters the activation conditions of a critical input i by ANDing the Σ_i of the input to the conditions for which an event propagates up to input i (i.e., PAF_i). If a gate has more than one critical input, its *PAF* is the sum of the filtered *PAF*s of its critical inputs.

An important feature of the algorithm is that it is based on a traversal of the critical gates and not of the critical paths. In fact, the number of (critical) paths can be exponential in the number of gates in the network, whereas the number of critical gates is guaranteed to be smaller than the number of gates.

Note that the algorithm relies on topological timing analysis. It is a well-known fact that such estimates can be very conservative. In the limiting case, if the topological delay is longer than the true delay and the true delay is shorter than T^*, we may actually synthesize useless hold logic. This is due to the fact that our procedure is conservative and it may actually flag as belonging to f_h some input conditions that do not propagate any perturbation to the output. Observe, however, that the accuracy of the procedure can be improved if more powerful algorithms for the computation of the arrival times are used (see, for example, [12], [13]). The modification of the pseudo-code in Fig. 5 is straightforward: It is sufficient to replace the `StaticTimingAnalysis` call with the call to an advanced timing analysis procedure. On the other hand, the computational burden of obtaining accurate delay information for all gates in the network may be substantially higher than that required by simple static timing analysis. In summary, the `StaticTimingAnalysis`

```
   procedure ComputeF_h(C,T*) {
1    StaticTimingAnalysis(C);
2    Levels[] = Levelize(C);
3    foreach (level l = 0,...,L) {
4      foreach (critical gate n ∈ Level[l]) {
5        PAF_n(x) = 0;
6        foreach (critical fanin i of gate n)
7          PAF_n(x) = PAF_n(x) + (PAF_i(x) · Σ_i(x)) ;
       }
     }
     f_h(x) = 0;
8    foreach (critical gate n ∈ Output_gates)
9      f_h(x) = f_h(x) + PAF_n(x);
     return (f_h(x));
   }
```

Fig. 5. Algorithm for computing the hold function.

should be replaced by the procedure that is used for timing analysis in the design flow.

4.2.3 Cutting Heuristics

Although the algorithm of Fig. 5 does not suffer the computational bottleneck of the exact method of [2], there may be circuits for which constructing the BDDs for the sensitization function is still not feasible. In these cases, an approximate solution is required that allows us to compute *partial* timing information.

A simple solution may be that of stopping procedure `ComputeF_h` after a desired number of levels or, alternatively, when the sizes of the BDDs grow beyond a given threshold. Unfortunately, this would result in incomplete timing information since some critical paths could be incorrectly left out of the computation. In fact, computing the hold function by levels does not necessarily take into account all critical paths unless we guarantee that last level (i.e., the primary outputs) is reached.

The observation above suggests a criterion for computing the timing information *incrementally*. The key for such criterion is to progressively select sets of critical gates, hereafter called *cuts*, such that the gates in a set cut all critical paths. If we can compute the BDDs (in the global support) of the path activation functions of all gates in a cut G, a solution of TS (i.e., a valid f_h) is simply:

$$f_h = \sum_{n \in G} PAF_n(\mathbf{x}). \tag{7}$$

A good cutting heuristics is obviously essential for an effective realization of the f_h computation algorithm. The one we propose starts from the critical inputs (cut G_0) and consists of the repeated application of three phases until no gate in the combinational circuit is left:

1. From a cut G_i, we reach the critical gates in the fanout of any gate in G_i. Only critical connections (i.e., connections from the output of a critical gate to the critical input of another critical gate) are explored. A newly reached gate is marked as belonging to the new cut G_{i+1} only if all its critical fanins belong to a previous cut G_j, $j = 0, 1, \ldots, i$ or to cut G_{i+1} itself.
2. If at least one gate has been marked, we check if all critical fanins of some additional gates reached from G_i have been reached. If this is the case, such gates are marked as belonging to G_{i+1}. This step is repeated until no new gate is marked. In other words, all gates for which all the critical fanins belong to G_j, $j \leq i+1$ are marked.
3. The remaining critical gates reachable from G_i do not belong to G_{i+1} and are *discarded*. However, to guarantee that all critical paths are cut, we insert in G_{i+1} all critical fanins of the discarded nodes which belong to previous cuts (or to cut G_{i+1} itself).

The set of gates G_{i+1} is the new cut. Notice that if an output is reached during traversal at cut G_j, such output is inserted in all successive cuts G_k, $k > j$. In addition, it can be easily observed that, in general, successive cuts are not disjoint.

After the computation of G_{i+1}, the path activation functions of its gates and f_h are computed. The termination conditions of the traversal algorithm are the following:

- If the BDD of a *PAF* for a gate in G_{i+1} blows up, the computation is aborted and the BDD of the f_h of the previous cut is returned.
- Once all *PAF*s have been computed, f_h is obtained by taking the Boolean sum of all *PAF*s. If the BDD of f_h blows up during the Boolean sum, the computation is aborted and the BDD of the f_h of the previous cut is returned.
- If the computation of f_h in the global support succeeds and the cut is the last one, f_h is returned. Conversely, if the cut is not the last one, f_h is stored and the next cut is generated.

The f_h for G_0 is obviously the most conservative TS solution, that is, $f_h = 1$. In the worst case, if *PAF* or f_h computation fails at the first cut, the value of f_h returned is the tautology. Hence, the procedure is guaranteed to return a valid solution to TS, but it may return the trivial one.

Example 3. Assume that all gates in Fig. 6 are critical. Initially, $G_0 = \{a, b, c, d\}$. For generating G_1, the fanouts of gates in G_0 are explored (notice that here all connections are critical because all gates are critical), i.e., gates $\{e, f, h\}$. First, only e is marked because all its critical fanins belong to G_0. Then, f is marked because c belongs to G_0 and e was previously marked. Gate h is discarded. The new cut is then: $G_1 = \{e, f, d\}$. Gate d is included in G_1 to guarantee that all critical paths are cut.

Notice that $G_0 \cap G_1 \neq \emptyset$. The f_h for G_1 is $f_h = PAF_e + PAF_f + PAF_d$. The third and last cut, G_2, is finally computed and it consists of gates g, h, and f. Gate f is included in G_2 because it is a primary output.

The algorithm for near-minimum TS solution described in Section 4.2.2 is modified by replacing a level-based traversal with a cut-based traversal. In this way, `ComputeF_h` is guaranteed to always return a valid solution to TS, even in the case of BDD blow-up.

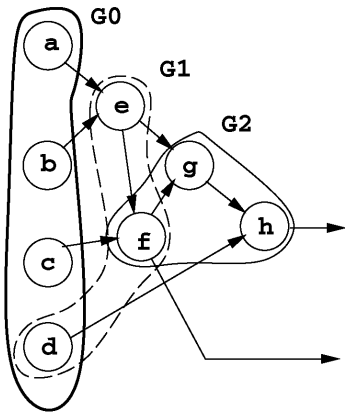


Fig. 6. An example of cut calculation.

5 Experimental Results

We have implemented the algorithms for TS solution described in Section 4 within the tool for telescopic units synthesis of [2]. The logic synthesis framework we have exploited is SIS [14] which uses CUDD [15] as the underlying BDD package. Experiments have been run on a DEC AXP 1000/400 with 256 MB of main memory. We applied our technique to standard benchmarks as well as realistic high-performance arithmetic units. The results of our experiments are summarized in the next two sections.

5.1 Standard Benchmarks

We have considered *all* circuits in the Iscas '85 [6] suite with more than 1,000 gates. Since only six examples were available, we have also experimented with the combinational logic of the 12 largest Iscas '89 [7] (addendum included) benchmarks.

The library used for mapping consisted of 2- to 4-input NAND and NOR gates, plus inverters and buffers, each of which had five different driving strengths. The gates are nonsymmetric, that is, they have different pin-to-pin delays, as well as different rise and fall delays. The delay model used is the SIS *real delay*.

The original circuits have been first optimized for speed using a modified version of `script.delay`, where the `full_simplify` and sometimes the `rr` commands have been removed to allow the optimization to complete on the large examples and then mapped for speed with either `map -m1` or `map -n1 -AFG`.

Table 1 reports the experimental data. Columns *Circuit*, I, O, G, T, and P give the name, number of inputs, outputs, and gates, the static delay (in *nsec*), and the throughput of the original circuit. Column $Prob(f_h)$ shows the probability of f_h, column G^* gives the total number of gates of the telescopic unit, column T^* reports the cycle time (in *nsec*) at which the telescopic unit is clocked to achieve the increased throughput of column P^*, and column $T(f_h)$ tells the (static) arrival time of the hold signal (in *nsec*). Columns ΔP and ΔG give the throughput improvement and the area overhead (in terms of gates) of the telescopic unit. Obviously, the area overhead only refers to the additional logic implementing f_h, while it does not consider the circuitry required to control the operations of the telescopic unit, the latter being dependent on the specific context in which the unit will be instantiated. Finally, column *Time* reports the CPU time (in *sec*) required to perform the automatic synthesis of f_h for a given T^*. A symbol * beside the circuit name indicates that the heuristics of Section 4.2.3 were required to complete the calculation of f_h. This has happened only on example `s38417`, where the computation of f_h stopped after 21 cuts (out of 28).

Only two benchmarks are missing from the table: `c6288` and `s38584`. The former is a 32-bit multiplier for which it is well-known that the computation of the BDDs for all outputs is infeasible [16]. The application of the algorithm of Fig. 5 therefore failed; we thus resorted to the heuristic

TABLE 1
Results: Standard Benchmarks

Circuit	I	O	G	T	P	$Prob(f_h)$	G^*	T^*	P^*	$T(f_h)$	ΔP	ΔG	*Time*
c1908	33	25	1339	23.51	0.04253	0.05761	1410	20.08	0.04836	17.46	13.7%	5.2%	63.2
c2670	233	140	1617	52.43	0.01907	0.15902	1723	44.50	0.02068	41.30	8.4%	6.5%	29.2
c3540	50	22	1876	29.28	0.03414	0.10192	2040	23.42	0.04053	23.30	18.7%	8.7%	541.5
c5315	178	123	3286	29.49	0.03391	0.26033	3448	23.02	0.03778	22.90	11.4%	4.9%	91.6
c7552	207	108	5439	23.16	0.04317	0.37850	5831	16.30	0.04973	16.08	15.2%	7.2%	2188.4
s3271	142	130	1393	41.63	0.02402	0.09462	1493	35.86	0.02656	35.44	10.6%	7.1%	8.6
s3330	172	205	1434	16.36	0.06112	0.02085	1540	13.89	0.07124	13.84	16.5%	7.3%	84.1
s3384	226	209	2307	32.91	0.03038	0.01305	2481	28.66	0.03466	28.17	14.1%	7.5%	193.5
s4863	153	120	3684	34.83	0.02871	0.06831	3828	27.63	0.03495	27.06	21.7%	3.8%	472.4
s5378	199	213	2078	36.80	0.02771	0.07885	2229	30.60	0.03139	28.89	13.2%	7.2%	138.5
s6669	322	294	4975	52.54	0.01903	0.04002	5210	45.93	0.02133	42.98	12.1%	4.7%	302.0
s9234	247	250	2821	19.32	0.05175	0.10112	3119	15.43	0.06153	15.34	18.9%	10.5%	2248.3
s13207	700	790	3554	22.61	0.04422	0.38551	3968	15.83	0.05099	15.05	15.3%	11.6%	76.7
s15850	611	684	4850	165.25	0.00605	0.08029	5377	132.20	0.00726	91.70	20.1%	10.8%	219.2
s35932	1763	2048	12944	874.98	0.00114	0.00036	13396	833.45	0.00120	133.11	5.3%	3.4%	13.9
s38417*	1664	1742	10326	60.31	0.01658	0.46151	10704	42.13	0.01825	40.89	10.1%	3.6%	1670.4
Average											14.1%	6.9%	

TABLE 2
Results: Arithmetic Units

Circuit	I	O	G	T	P	$Prob(f_h)$	G^*	T^*	P^*	$T(f_h)$	ΔP	ΔG	Time
DW02_prod_sum1.a	65	32	3448	164.11	0.00609	0.03301	3801	137.44	0.00715	120.19	17.5%	10.2%	605.0
DW02_prod_sum1.d	65	32	5201	131.57	0.00760	0.01843	5436	108.95	0.00909	105.48	19.6%	4.5%	442.8
DW02_sin.a	16	16	1916	152.87	0.00654	0.04645	1988	138.13	0.00707	71.52	8.1%	3.7%	102.0
DW02_sin.d	16	16	2642	108.74	0.00919	0.02112	2790	93.18	0.01061	91.04	15.5%	5.6%	572.8

method discussed in Section 4.2.3; also, in this case, however, the result was negative since the computation of f_h stopped after 41 cuts (out of 103) with $T^* = 0.95T = 70.40$ *nsec* and $Prob(f_h) = 0.99999$. The application of our algorithm to the latter example, on the other hand, has not been tried since a mapped version of it could not be obtained for the selected gate library.

The results are quite satisfactory since an average throughput improvement of 14.1 percent has been achieved with an average area penalty of 6.9 percent. The proposed approach thus demonstrates its scalability and applicability to the largest available benchmarks. Needless to say, most of the circuits examined here are well beyond the capability of the exact MTS solution algorithm of [2], which, for this library and delay model, fails for circuits larger than a few hundreds of gates. On small circuits, for which exact MTS solution is possible, our tool still achieves improvements around 10-15 percent, while the exact minimum solution allows average improvements around 27 percent [2].

5.2 Arithmetic Units

In this section, we study the application of our technique to two complex arithmetic units. The purpose of this analysis is to show that our automatic transformation is applicable and useful not only on standard benchmarks, whose functionalities and architectures are uncertain, but also for carefully designed units that are used in real-life systems. We have considered two units belonging to the advanced mathematical library of Synopsys' `DesignWare` components, namely, a combinational multiplier-adder module (called `DW02_prod_sum1`) and a combinational sine function module (called `DW02_sin`). These components are hand-coded in synthesizable HDL (Verilog or VHDL) by expert designers and can be instantiated as black boxes in register-transfer level descriptions. Clearly, such library components are specifically designed for high performance, hence, they represent a good test for assessing the effectiveness of our paradigm in pushing throughput beyond the possibilities of standard synthesis techniques.

The multiplier-adder module implements the function $S = A * B + C$, where the width of the operands can be chosen at instantiation time. Furthermore, the unit has a control input TC, whose function is to select two's complement versus sign-magnitude representation for the data. The internal architecture of the multiplier is based on a fast carry-save array. For our experiment, we selected a 16-bit width for operands A and B and a 32-bit width for operand C and output S.

The sine module implements the function $C = sin(A)$. Input angle A is treated as a binary fixed point number which is converted to radiants when multiplied by π. When A is interpreted as unsigned, the input angle A is a binary subdivision of the range $0 \leq A < 2$. When A is interpreted as signed (two's complement), the range is $-1 \leq A < 1$. The value of the sine function is computed with either a linear, or quadratic, or cubic interpolation scheme, depending on the value of A. The bit width of both the angle and the sine values are parameterized and are chosen at instantiation time. For our experiment, we set both widths to 16 bits.

The gate-level netlists of the two arithmetic units have been generated from the corresponding HDL descriptions using Synopsys `DesignCompiler` and then translated into the `blif` format. Two technology-dependent implementations for each unit have been obtained through logic optimization and technology mapping using SIS. In particular, circuits `DW02_prod_sum1.a` and `DW02_sin.a` are minimum area realizations, while circuits `DW02_prod_sum1.d` and `DW02_sin.d` are obtained from the min-area descriptions by applying delay optimization under area constraints (we allowed a 2X area increase).

Each description has been transformed into a telescopic unit, and Table 2 collects the results of the experiments. Throughput improvements range from 8.1 percent to 19.6 percent, while the area overheads are between 3.7 percent and 10.2 percent.

By direct inspection of the data in the table, it can be evinced that telescopic units provide an area-effective way of improving system's throughput. As an example, consider the telescopic version of circuit `DW02_prod_sum1.a`. Its throughput is approximately the same as that of the reference circuit `DW02_prod_sum1.d`(0.00715 versus 0.00760), but its area is substantially smaller (3,801 versus 5,201 gates).

6 Conclusions

We have addressed the timed supersetting problem and we have contributed an algorithm for its solution which is well-suited for the automatic synthesis of large telescopic units in the cases where complex and realistic gate delay models are adopted. Results obtained on the largest benchmarks available in the literature (i.e., the Iscas '85 and the Iscas '89 circuits), as well as on realistic, high-performance arithmetic units are quite satisfactory and confirm that the use of telescopic units represents a robust and flexible alternative for improving the performance of delay-critical digital applications.

REFERENCES

[1] S.F. Oberman and M.J. Flynn, "Design Issues in Division and Other Floating-Point Operations," *IEEE Trans. Computers,* vol. 46, no. 2, pp. 154-161, Feb. 1997.

[2] L. Benini, G. De Micheli, E. Macii, and M. Poncino, "Telescopic Units: A New Paradigm for Performance Optimization of VLSI Designs," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 17, no. 3, pp. 220-232, Mar. 1998.

[3] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and Their Applications," *Formal Methods in System Design,* vol. 10, pp. 171-206, 1997.

[4] R.I. Bahar, H. Cho, G.D. Hachtel, E. Macii, F. Somenzi, "Symbolic Timing Analysis and Re-Synthesis for Low Power of Combinational Circuits Containing False Paths," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 16, no. 10, pp. 1,101-1,115 Oct. 1997.

[5] P.C. McGeer and R.K. Brayton, *Integrating Functional and Temporal Domains in Logic Synthesis.* Boston, Mass.: Kluwer Academic, 1991.

[6] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *Proc. ISCAS-85: IEEE Int'l Symp. Circuits and Systems,* pp. 785-794, Kyoto, Japan, June 1985.

[7] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. ISCAS-89: IEEE Int'l Symp. Circuits and Systems,* pp. 1,929-1,934, Portland, Ore., May 1989.

[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *An Introduction to Algorithms.* New York: McGraw-Hill, 1990.

[9] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers,* vol. 35, no. 8, pp. 79-85, Aug. 1986.

[10] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys,* vol. 24, no. 3, pp. 293-318, Sept. 1992.

[11] D. Brand and V.S. Iyengar, "Timing Analysis Using Functional Analysis," *Proc. ICCAD-86: Int'l Conf. Computer-Aided Design,* pp. 126-129, Santa Clara, Calif., Nov. 1986.

[12] H.C. Chen and D.H.C. Du, "Path Sensitization in Critical Paths," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 12, no. 2, pp. 196-207, Feb. 1993

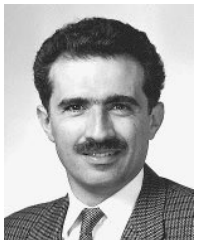
[13] S. Devadas, K. Keutzer, and S. Malik, "Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 12, no. 12, pp. 1,913-1,923, Dec. 1993.

[14] E.M. Sentovich, K.J. Singh, C.W. Moon, H. Savoj, R.K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," *Proc. ICCD-92: IEEE Int'l Conf. Computer Design,* pp. 328-333, Cambridge, Mass., Oct. 1992.

[15] F. Somenzi, *CUDD: University of Colorado Decision Diagram Package,* Release 2.1.2, technical report, Dept. of Electrical and Computer Eng., Univ. of Colorado, Boulder, Apr. 1997.

[16] R.E. Bryant, "On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication," *IEEE Trans. Computers,* vol. 40, no. 2, pp. 205-213, Feb. 1991.

Luca Benini received the DrEng degree in electrical engineering from the Università di Bologna, Italy, in 1991, and the MS and PhD degrees in electrical engineering from Stanford University in 1994 and 1997, respectively. Since 1998, he has been an assistant professor in the Department of Electronics and Computer Science at the University of Bologna. He also holds visiting researcher positions at Stanford University and the Hewlett-Packard Laboratories, Palo Alto, California. Dr. Benini's research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications. He has been a member of the technical program committee for the Design and Test in Europe Conference and the International Symposium on Low Power Design.

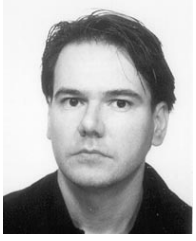


Giovanni De Micheli is a professor of electrical engineering and, by courtesy, of computer science at Stanford University. His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on automated synthesis, optimization and validation. He is the author of *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, 1994), co-author of *Dynamic Power Management: Circuit Techniques and CAD Tools* (Kluwer, 1998), and of three other books. He received the 1987 IEEE Transactions on CAD/ICAS Best Paper Award and two Best Paper Awards at the Design Automation Conference in 1983 and in 1993. He was program and general chair of the International Conference on Computer Design in 1988 and 1989, respectively, and the program chair (for design tools) of the Design Automation Conference in 1996 and 1997. He is the Editor-in-Chief of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Antonio Lioy received the DrEng degree in electrical engineering from the Politecnico di Torino, Italy, in 1982, and the PhD degree in computer engineering from the Politecnico di Torino in 1987. From 1987 through 1989, he was a research assistant at the Politecnico di Torino, from 1990 through 1992, he was an assistant professor at the same institution, and, in 1993, he became an associate professor at the Università di Parma, Italy. Currently, he is an associate professor at the Politecnico di Torino. His research interests include simulation and testing of digital circuits and systems, as well as advanced networking technologies and computer security.

Enrico Macii received the DrEng degree in electrical engineering from the Politecnico di Torino, Italy, in 1990, the DrSc degree in computer science from the Università di Torino in 1991, and the PhD degree in computer engineering from the Politecnico di Torino in 1995. From 1991 through 1994, he was an adjunct faculty member at the University of Colorado at Boulder. Currently, he is an associate professor at the Politecnico di Torino. His research interests include synthesis, verification, simulation, and testing of digital circuits and systems. He received the Best Paper Award at the European Design Automation Conference in 1996. He was the technical program co-chair of the 1999 IEEE Alessandro Volta Memorial Workshop on Low Power Design. He is an associate editor of the *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Giuseppe Odasso received the DrEng degree in electrical engineering from the Politecnico di Torino, Italy, in 1994. Currently, he is working toward his PhD in computer engineering at the Politecnico di Torino. His research interests include synthesis, verification, simulation, and testing of digital circuits, with special emphasis on low-power and high-performance systems.



Massimo Poncino received the DrEng degree in electrical engineering in 1989 and the PhD degree in computer engineering in 1993, both from the Politecnico di Torino, Italy. From 1993 through 1994, he was a visiting faculty member at the University of Colorado at Boulder. Currently, he is an assistant professor at the Politecnico di Torino. His research interests include synthesis, verification, simulation, and testing of digital circuits and systems.