Efficient Calculation of All-Pairs Input-to-Output Delays in Synchronous Sequential Circuits

Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
201 Coover Hall, Iowa State University,
Ames, IA 50011, USA.

Abstract

In this paper, we consider the problem of finding all-pairs input-to-output delays for combinational circuits. This method is of practical utility in several design situations and CAD algorithms, for example in [1, 2]. An algorithm for solving this problem was proposed in [1]; however, this can be computationally expensive. We take advantage of some properties of large realistic circuits to present an algorithm that is two orders of magnitude faster than the method in [1] for large circuits. Experimental results on ISCAS benchmark circuits prove the efficacy of this approach.

I. Introduction

Particularly in the design of high-speed circuits that push the limits of technology, the importance of obtaining the circuit timing information cannot be understated. Various methods for estimating the timing behavior of digital VLSI circuits have been proposed in the literature. Most fast timing estimation methods proceed by calculating the delay of each gate individually and then using a traversal technique to propagate the delays through the circuit. In considering circuit delays, one is primarily concerned with combinational blocks; a typical objective, for example, is to ensure that all combinational blocks in a sequential circuit have delays that satisfy the clock period.

Any combinational circuit can be conceived as being composed of a set of gates that lie between a set of input flip-flops or primary inputs and a set of output flip-flops or primary outputs. This work deals with the efficient calculation of all *pairwise* input-to-output delays in a combinational circuit, and presents an efficient algorithm for the purpose.

The motivation for the work is as follows. While in some timing analysis situations, it is adequate to identify the single worst-case input-output path and its delay, in many cases, it is also required that one find the worst-case delay from every single input of a combinational circuit to every single output. We refer to this as the *all-pairs* input-output delay calculation problem. An example of such a situation arises in [1], where clock skew optimization is used as a means of improving the timing behaviour of a sequential circuit. In this situation, it is required that the all-pairs input-output delay calculation problem be solved for every combinational block of the sequential circuit. Another example where this procedure is used in an algorithm for retiming sequential circuits is described in [2].

A simple algorithm for solving the all-pairs delay calculation problem has been proposed in [1]. In the absence of a careful implementation, we show that it is computationally expensive, and that we may take advantage of the structure of real circuits to make the procedure much more efficient.

The paper is organized as follows. In Section II, the PERT method for critical path identification, which forms the basis for the algorithms described in this paper is described. Section III outlines the adaptation of this method in [1], and illustrates our method that performs the all-pairs delay computations efficiently. Experimental results are presented in Section IV, followed by concluding remarks in Section V.

II. Finding the Critical Path in a Circuit

The Program Evaluation and Review Technique (PERT) was first proposed for use in timing analysis in [3], and dates back to a classical technique used in management science. This method forms the heart of many static timing analysis algorithms. Although in many timing analysis contexts, as pointed out in [4], the procedure that is referred to as PERT is actually an execution of the critical path method (CPM) in management science, we will persist in referring to this method as PERT for consistency with the literature. PERT may be used to find the worst-case path to each output of a combinational block from *any* input (note that this differs from the all-pairs problem that we tackle, where one is required to find the delay from *each* input to each output).

The procedure is best illustrated by means of a simple example in which the worst-case (longest path) delay to each output (from any input) is calculated. Consider the circuit in Figure 1. Each box represents a gate, and the number within the box represents the delay associated with it. We assume that the worst-case arrival time for a transition at any input, i.e., at the inputs to boxes A, B, C, and D, is 0.

A component is said to be *ready for processing* when the signal arrival time information is available for all of its inputs. Initially, since signal arrival times are known only at the primary inputs, only those components that are fed solely by primary inputs are ready for processing. In the example, these components would be A, B, C, and D. These are placed in a queue and are scheduled for processing.

In the iterative process, the component at the head of the queue is scheduled for processing. Each processing step consists of

(a) finding the latest arriving input to the component,

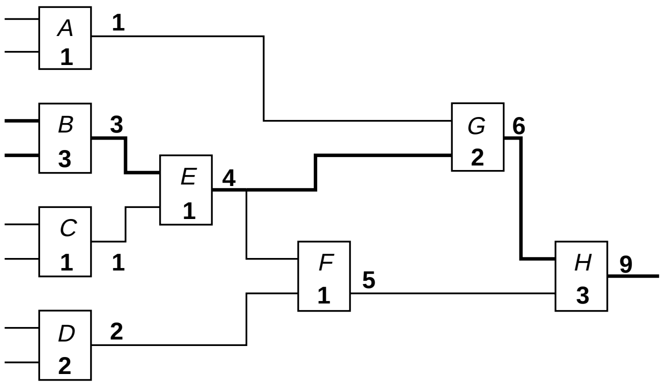

Figure 1: The PERT technique.

which triggers the output transition,

(b) adding the delay of the component to the latest arriving input time to obtain the worst-case transition time at the output, and

(c) checking all of the components that the current component fans out to, to find out whether they are ready for processing. If so, the component is added to the tail of the queue.

The iterations end when the queue is empty.

In the example, the algorithm is executed as follows:
(1) In the initial step, A, B, C, and D are placed in the queue.
(2) A is scheduled. The latest input transition for A is 0, the delay of A is 1; hence, the latest output transition for A is at time (0+1) = 1. No additional elements can be added to the queue.
(3) B is scheduled. The latest output transition is found to be at time (0+3) = 3. No additional elements can be placed in the queue.
(4) C is scheduled, and the worst-case output transition occurs at time (0+1) = 1. At this point, all input information for component E is available, and it is placed at the tail of the queue.
(5) D is scheduled; the output transition time is found to be (0+2) = 2.
(6) E is scheduled; the latest arriving input comes in at time 3; hence, the worst-case output transition occurs at time (3+1) = 4. At this point, F and G are added to the tail of the queue.
(**Step 7.** F is scheduled; its latest output transition is at time (4+1) = 5.
(8) G is scheduled; its latest output transition occurs at time (4+2) = 6. H is added to the tail of the queue.
(9) H is scheduled; its worst-case output transition is at time (6+3) = 9. The queue is now empty and the algorithm terminates.

The worst-case delays at the output of each component are shown in Figure 1.

The *critical path*, defined as the path between an input and an output with the maximum delay, can now easily be found by using a traceback method. We begin with the component whose output is the primary output with the latest transition time. This is the last component on the critical path. Next, the latest arriving input to this component is identified. The component that causes this transition is the preceding component on the critical path. The process is repeated recursively until a primary input is reached.

In the example, we begin with component H at the output. The latest arriving input to this component is caused by component G, which thus precedes H on the critical path. Similarly, the transition at G is caused by the latest arriving input, which comes in from component E, and so on. By continuing this process, the critical path from the input to the output is identified as B-E-G-H.

In the case of CMOS circuits, the rise and fall delay transitions may be calculated separately. For inverting CMOS gates, the latest arriving input rise (fall) transition triggers off a fall (rise) transition at the output. This can easily be incorporated into the PERT method.

One may also use this procedure to find the the best-case (shortest path) delays to all outputs of a combinational block from each input. The only difference is that the earliest arrival time at the output of each gate is maintained, and the earliest arrival times at the inputs to each gate are added to the delay of a gate to obtain the earliest arrival time at its output.

III. Finding All Latch-to-Latch Critical Paths

A. Existing Method

In [1], a procedure for adapting PERT to find the worst-case delay path for *every* input-output pair, i.e., from every input latch to every output latch, was described. The procedure is a simple adaptation of PERT.

The procedure for finding the worst-case delay from a specific input latch, i, to all output latches is described below; this procedure is repeated for all input latches to obtain all pairwise input latch-to-output latch delays.

(1) Initialize the latest arrival time at each input latch except i to $-\infty$. Set the latest arrival time at latch i to 0.

(2) Propagate all arrival times to the output using PERT to find the longest paths to the outputs.

(3) The delays obtained at the output latch j represents the maximum delay from input latch i to output latch j. If this delay is $-\infty$, it implies that there is no path from i to j.

The procedure works because in taking the maximum of all arrival times at each gate in the PERT algorithm, all paths that do not originate at i have a latest arrival time of $-\infty$. If all paths to the inputs of a gate have a latest arrival time of $-\infty$, the latest arrival time at the output of the gate is also calculated as $-\infty$, consistent with the fact that there is no path from i to the output of that gate. If, however, there are one or more paths from input i to the gate, then only the delays along these paths (which are positive numbers) are effectively considered in calculating the latest arrival time at the output of the gate, as all other paths have an arrival time of $-\infty$.

In case of finding the minimum delay for all input-output combinations, the procedure is similar, except that all inputs other than the input of interest are awarded arrival times of ∞, and PERT is carried out to find out the shortest path to the outputs.

B. Complexity of the Method

Let the combinational circuit be represented by a directed graph in which each gate corresponds to a node. A fanout of gate i that goes to gate j is represented as a directed edge from node i to node j in the directed graph. Since the circuit is combinational, it cannot contain any cycles, and therefore the graph is a directed acyclic graph (DAG).

The complexity of performing each PERT operation is $O(|E|)$, where $|E|$ is the number of edges in the graph. If the number of input latches is $|I|$, then the complexity of finding the maximum delays for all input-output pairs is $O(|E| \cdot |I|)$.

C. Efficient All-Pairs Delay Computation on Real Circuits

The procedure described above may be used to calculate all pairs input-to-output delays of a combinational block. However, if it were to be performed directly, it would lead to large computation times, as will be shown in our experimental results.

We use a key attribute of realistic circuits in making the procedure more efficient, observed during the symbolic propagation of constraints in [5]: that most inputs to a combinational block exercise only a small fraction of the total number of input-output paths in the combinational block. In such a situation, performing the procedure above results in excessive computation in propagating signals along paths that are not influenced by the input under consideration. Consider, for example, the situation in Figure 2, where only the darkened edges contribute to useful operations during PERT. However, if one applied the approach outlined above, one would perform a large number of wasteful computations in propagating arrival times of $-\infty$ from the other primary inputs to the gates along this path. We therefore see that if we could reduce the number of wasteful propagations of $-\infty$ delays, the complexity of the procedure could be brought down considerably from the $|E|$ operations required for each input. In the example shown, it can be seen that the only useful computations for input i are the ones corresponding to the darkened edges, which is a small fraction of the total number of edges.

Based on this observation, we develop an efficient event-driven procedure for calculating the values of the worst-case delay from a given input to all outputs; if this procedure is repeated for all inputs to a combinational block, all input-output worst case delays may be computed. It was found that the use of this procedure gave run-time improvements of several orders of magnitudes.

Definition: The level number, level(k), of a gate k in the combinational block is defined as the largest number of gates from a primary input to the gate, inclusive of the gate.

The level numbers of all gates may be computed by a single PERT run in which all gate delays are set to 1; the worst-case delay of each gate is equal to its level as defined

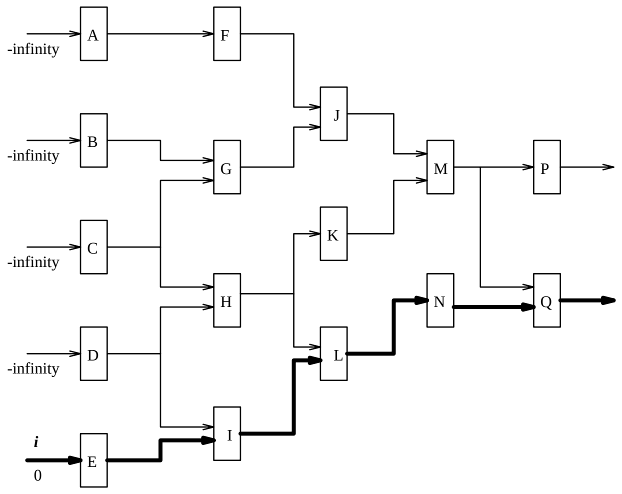

Figure 2: Computing the delay from i to all outputs.

above. To find $\overline{d}(i, o)$, the largest delay from primary input i to all $o \in$ the set of primary outputs, we conduct an event-driven PERT-like exercise starting at flip-flop i, as described in the following piece of pseudocode. During the process, we maintain a set of queues, known as level queues. The level queue indexed by k contains all gates at level k that have had one or more inputs processed.

```
Initialize all level queues to be empty;
currentlevel = 0;
currentgate = flip-flop $i$;
while (currentgate != nil) {
    if (currentgate $\neq i$)
        for (all fanins $j$ of currentgate)
            $\overline{d}(i,$currentgate) =
             max($\overline{d}(i,j)$) + delay(currentgate);
    else
            $\overline{d}(i,i)$ = 0;
    for (all fanouts $k$ of currentgate)
        Append gate $k$ to the level queue indexed by
            level($k$), if it does not already lie
            on the queue;
    if (all level queues are not empty) {
        currentlevel = lowest level number whose
            level queue is nonempty;
        currentgate = head of the level queue
            for currentlevel;
        delete currentgate from its level queue;
    }
    else
        currentgate = nil;
}
```

At each step, an element from the lowest unprocessed level is plucked from its level queue, and the worst-case delay from flip-flop i to its output is computed. All of its fanouts are then placed on their corresponding level queues, unless they have already been placed on these queues. Note that by construction, no gate is processed until the delay to all of its inputs that are affected by flip-flop i have been computed, since such inputs must necessarily have a lower level number.

Referring back to the example in Figure 2, it can be seen that the application of this procedure results in considering only the gates E, I, L, N and Q during the PERT process, thereby considerably reducing the computation by considering just 6 of the 25 edges for input i, and only 49 of the 125 ($= 25 \times 5$)edges while considering the all-pairs problem. In practice, we have observed that the fraction of useful computations grows smaller for larger combinational circuits.

It is worth noting that in the worst case, this procedure performs PERT $|I| + 1$ times on the circuit; once for finding the level numbers, and once for each input. However, each PERT operation after the first one typically traverses a very small subset of the total number of gates in the circuit, and therefore the amount of computation is significantly reduced for real circuits. This also illustrates that in the worst case this approach is only marginally worse than the method in [1]. This worst case was never exercised in any of the large circuits that we ran the algorithm on, and in reality, large computational savings were seen.

IV. Experimental Results

The procedure described above was applied to circuits in the ISCAS89 benchmark suite. The circuits in this suite are sequential circuits, and each of these is decomposed into a set of combinational subcircuits that lie between latches. The algorithm above is applied to each combinational subcircuit to find the maximum all-pairs delays from each input of the subcircuit to each output.

Table 1: Experimental Results

Circuit	$\|G\|$	$\|F\|$	CPU Time (in [1])	CPU Time (ours)	Speedup
s4863	2342	104	3.9s	0.17s	22.9
s5378	2779	179	6.8s	0.16s	42.5
s6669	3080	239	12.1s	0.25s	48.4
s9234.1	5597	211	18.5s	0.60s	30.8
s13207.1	7951	638	76.3s	0.94s	81.2
s15850.1	9772	534	82.5s	3.13s	26.4
s35932	16065	1728	418.5s	3.36s	124.6
s38417	22179	1636	529.9	4.01s	132.1
s38584.1	19253	1426	498.2	5.48s	90.9

The results on all of the large circuits in the ISCAS89 benchmark suite are shown in Table 1, where the first column lists the name of the circuit, followed by a listing of the number of gates and flip-flops in the circuit, to give an idea of the circuit size. The procedure described in this paper was used to solve the all-pairs maximum delay problem, and was compared to the approach in [1]. The delays calculated in each case were identical, testifying to the correctness of the new approach, with difference that our approach took a significantly smaller amount of time. A comparison of the CPU times for the all-pairs problem is shown in the next two columns of Table 1. The speedup, shown in the last column, is calculated as the ratio of the CPU time required by the method in [1] divided by the CPU time required by our approach. The CPU times correspond to an execution of the program on a DEC 3000/300L AXP machine. It is seen that the speedup can be as high as 132, and is generally larger for larger circuits. This is consistent with the intuitive observation that an input is likely to excite a smaller fraction of the total number of paths in a larger circuit.

We also ran the algorithm on all of the circuits in the ISCAS85 benchmark suite. However, these circuits are sufficiently small so that the method in [1] can handle them easily, and the additional gain from using our approach is not appreciable. The strength of the proposed approach is its speedup when it is applied to large circuits, and this has been illustrated in Table 1.

This approach has been embedded into the retiming algorithm in [2] and has contributed to making the procedure significantly more computationally efficient.

V. Conclusion

The problem of finding all-pairs input-to-output delays for combinational circuits is tackled in this paper. An efficient algorithm that is a hybrid of the event-driven and compiled approaches has been presented for the purpose, taking advantage of the observed properties of real circuits. Experimental results on a set of ISCAS benchmark circuits illustrate that this approach provides a tremendous benefit.

References

[1] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, pp. 945–951, July 1990.

[2] R. B. Deokar and S. S. Sapatnekar, "A fresh look at retiming via clock skew optimization," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 310–315, 1995.

[3] T. Kirkpatrick and N. Clark, "PERT as an aid to logic design," *IBM Journal of Research and Development*, vol. 10, pp. 135–141, Mar. 1966.

[4] T. M. Burks, K. A. Sakallah, and T. N. Mudge, "Critical paths in circuits with level-sensitive latches," *IEEE Transactions on VLSI Systems*, vol. 3, pp. 273–291, June 1995.

[5] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "Timing and area optimization for standard cell VLSI circuit design," *IEEE Transactions on Computer-Aided Design*, pp. 308–320, Mar. 1995.