Computing Logic-Stage Delays Using Circuit Simulation and Symbolic Elmore Analysis

Clayton B. McDonald Randal E. Bryant *
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213
{clayton,bryant}@ece.cmu.edu

ABSTRACT

The computation of logic-stage delays is a fundamental sub-problem for many EDA tasks. Although accurate delays can be obtained via circuit simulation, we must estimate the input assignments that will maximize the delay. With conventional methods, it is not feasible to estimate the delay for all input assignments on large sub-networks, so previous approaches have relied on heuristics. We present a symbolic algorithm that enables efficient computation of the Elmore delay under all input assignments and delay refinement using circuit-simulation. We analyze the Elmore estimate with three metrics using data extracted from symbolic timing simulations of industrial circuits.

1. INTRODUCTION

The computation of logic-stage delays in transistor networks is a fundamental sub-problem for a number of electronic design automation tasks. Examples include static timing analysis, timing simulation, transistor-sizing optimization, and library cell characterization. Determination of the sensitizing conditions for the maximum and minimum stage delays is extremely difficult in general, and exact solutions may be impossible for large networks.

Typically, stage delays are computed on channel-connected regions (CCRs), consisting of all nodes and transistors reachable from each other through transistor drain-source (channel) connections. Given a particular input transition or simultaneous set of transitions, we wish to determine the delay to the resulting transition on a designated output node. This delay value is generally dependent on the states of other inputs to the stage, as well as the initial conditions of the internal nodes. For example, Figure 1 shows a dynamic stage where the delay from *a* rising to *pc* falling is dependent on the values of *b*, *c*, and *x1*. Inputs *b* and *c* control the conductance of the discharge path, while the initial state of *x1* affects the amount of charge that must be removed. Determining the sensitizing conditions for the minimum or maximum delay is further complicated by potential logical relationships between the inputs.

*This research was supported by the SRC (contract DC-068)

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

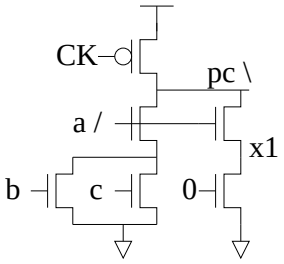

Figure 1: Effects of Side-Conditions on Delay

Previously published approaches have used a combination of formal methods and heuristics. Desai and Yen [8] implemented algorithms for sensitizing the maximum delay on a specified path through a multi-CCR transistor-level network, which they decomposed into sensitizing a series of maximum delays through single CCRs. Their method utilizes Boolean functions (stored as Binary Decision Diagrams, or BDDs) to compute the set of input assignments that enable the desired input to output transition. For small CCRs, they advocate explicit enumeration of these input assignments to determine which sensitizes the largest delay. For large CCRs, they first determine the maximum-resistance driving path, and then use a greedy algorithm that attempts to select the assignment which maximizes the capacitance connected to the output node. This approach assumes that the assignment that maximizes the Elmore delay will maximize the true delay, an assumption that is challenged by our data (Section 3). Furthermore, maximizing the resistance before considering capacitances may not even lead to a maximal Elmore delay, as there might be an assignment which sensitizes a low-resistance, high-capacitance driving path with a larger RC product. Burks and Main's approach [5] is quite similar, though they primarily focused on incorporating logical dependencies between inputs. To select the worst-case input assignment, they say only that they use a heuristic method.

Our approach also uses the Elmore delay as an estimate of the true delay. However, using symbolic techniques, we can exploit the regularity of large CCRs and compute the Elmore delay exactly for all input assignments while avoiding exponential blowup for all but the most pathological cases. The primary enabler for our methodology is the Multi-Terminal Binary Decision Diagram (MTBDD) data structure [2]. Using MTBDDs, we compute the Elmore delay for all possible input assignments. We can then select one assignment for each possible delay case, and use it as stimulus for a SPICE-like simulation.

Section 2 discusses the computation of the symbolic Elmore delay and its refinement to high-accuracy delay values with a SPICE-

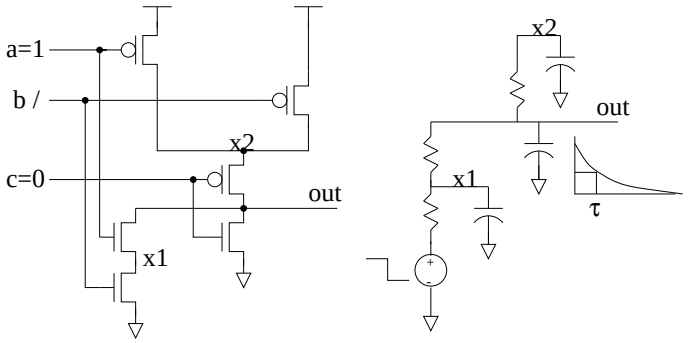

Figure 2: Elmore Approximation for Logic Stage

like simulator. We then present error-characterization data in Section 3, drawn from test-runs of our symbolic timing simulator on industrial circuit designs.

2. STAGE-DELAY CALCULATION

2.1 Elmore Estimate

Since our approach is based on the Elmore estimate of a logic-stage's delay, we will first present a quick review. The Elmore delay is an estimate of the dominant time constant of the step-response of an RC tree. As a result, it has been heavily utilized for estimating delays in interconnect networks. However, a number of researchers have adapted it to obtain delay estimates of single stages of MOSFET circuits [11, 6]. This is accomplished by removing non-conducting (OFF) transistors and replacing conducting (ON) transistors with simple linear resistors. At each internal node we compute a single grounded capacitance value, and then heuristically break loops of conducting transistors to complete the RC tree. This conversion process is depicted in Figure 2.

Besides the obviously risky approximation of transistors by linear resistances, the Elmore estimate has several deficiencies when used for logic stages. First, it assumes that the inputs switch instantaneously at time zero. To incorporate the effects of non-zero rise/fall times, we can effectively modify the resistance representing the turning-on transistor to reflect its reduced drive-capacity or calculate an empirical penalty to be added onto the final delay estimate. The second major assumption is that of a single driving voltage source. This can be a significant difficulty in analyzing stages where multiple pulldown paths can be activated simultaneously, or for ratioed circuits where pullup and pulldown paths are "fighting" each other. Again, empirical approximations can be made to incorporate these effects, as was done by Chu[6]. Since our symbolic Elmore analysis procedure is based on Chu's, it implements these enhancements.

2.2 MTBDDs

Previously published approaches to stage-delay computation have utilized symbolic techniques to handle logical restrictions on input patterns with considerable success. However, they have had to resort to heuristic methods for representing real-valued functions, such as resistance, capacitance and delay. The primary enabler for our approach, and the key to extending symbolic techniques to this real-valued domain, is the Multi-Terminal Binary Decision Diagram (MTBDD) [2].

MTBDDs are generalizations of BDDs that allow an arbitrary number of real-valued terminals. For example, the MTBDD in Figure 3 represents the function F having two inputs a and b. To determine the return value for any given input assignment, we work downwards from the root, following the solid arc from nodes assigned 1 and the dashed arcs from nodes assigned 0. We can see that $F \leftarrow 2.5$ when either a or b is 1, and $F \leftarrow 1.2$ otherwise.

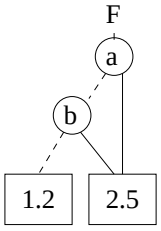

Figure 3: Example MTBDD

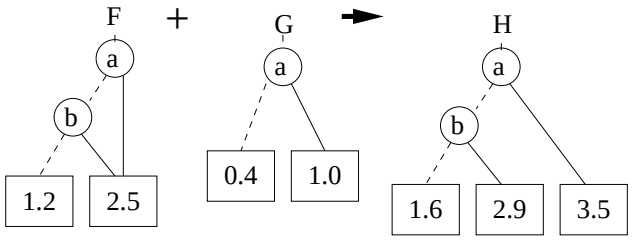

Figure 4: Example MTBDD Operation

Computation on MTBDDs can be accomplished using the function *MtbddApply*, which is virtually identical to the well-known BDD *Apply* function[3]. It takes as arguments an operator and two operands, and returns an MTBDD representing the result for all input assignments. For example, Figure 4 shows two input MTBDDs $\mathbf{F}$ and $\mathbf{G}$, and the MTBDD that would result from computing *MtbddApply*(+,$\mathbf{F}$,$\mathbf{G}$). *MtbddApply* has worst-case complexity $O(|\mathbf{F}| \times |\mathbf{G}|)$, where $|\mathbf{F}|$ represents the number of terminals in MTBDD $\mathbf{F}$.

Using *MtbddApply*, we can perform any algebraic operations necessary to compute series and parallel resistances, RC products, and other quantities needed for our analysis. For example, since Elmore delays in digital networks are computed by replacing transistors with switched resistors, we can represent its symbolic resistance by an MTBDD which returns infinity where the transistor is off, and its equivalent conducting resistance when it is on (Figure 5). Then, using *MtbddApply* calls, we can compute arbitrary parallel and series combinations of these symbolic resistors as shown in Figure 6.

Throughout this paper, we will denote BDDs and MTBDDs in boldface (i.e. $\mathbf{F}$), while scalar values will appear in normal type. We will also often utilize infix notation rather than explicit calls to *MtbddApply*, such that $\mathbf{F} + \mathbf{G} \equiv$ *MtbddApply*$(+, \mathbf{F}, \mathbf{G})$. In some cases it will be convenient to specify trivial MTBDDs, consisting only of a single terminal node, in brackets (i.e. $[1.5]$).

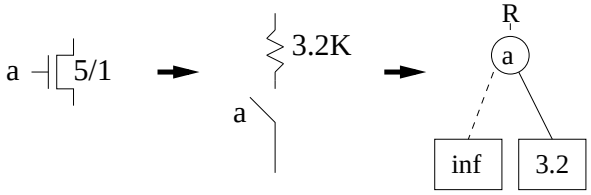

Figure 5: Representing FETs

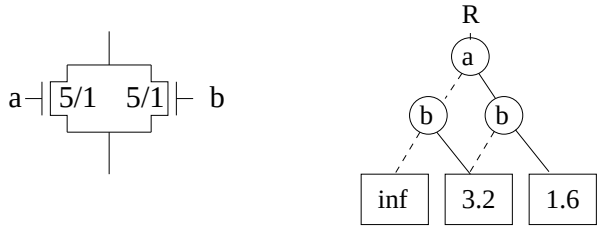

Figure 6: Symbolic FET Analysis

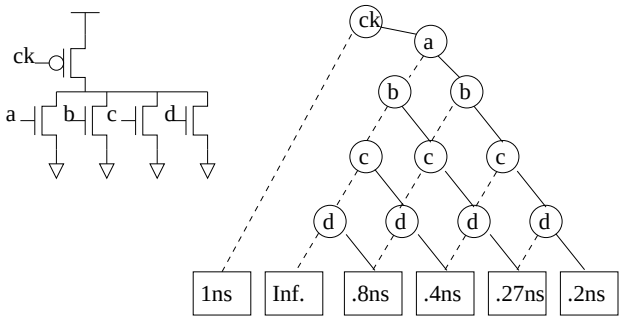

Figure 7: Example $\mathbf{T_{delay}}$ for dynamic NOR

2.3 Symbolic Elmore Analysis

The core of our symbolic Elmore analysis procedure is the pair of functions *SymbolicComputeDC* and *SymbolicComputeDelay*, which are described in detail in [9].

Given a channel-connected transistor network (CCR) whose input and internal node-values have been set to appropriate Boolean functions, *SymbolicComputeDC(z)* returns a BDD representing the function to which node *z* will settle. *SymbolicComputeDelay(z,***f***)* returns an MTBDD representing the time required for node *z* to settle to function $\mathbf{f}$. Both functions start from the output node and recur through channel connections until they reach power or ground. They then perform series and parallel computations as they return from the recursion, using symbolic algebra as outlined above. *SymbolicComputeDC* uses a voltage-divider or charge-sharing equation to compute an MTBDD representing the steady-state voltage at the output node under any input assignment. This voltage value is thresholded to obtain a Boolean function for the output node. In similar fashion, *SymbolicComputeDelay* computes symbolic resistance and capacitance MTBDDs that are combined to obtain the symbolic Elmore delay MTBDD, $\mathbf{T_{delay}}$.

Our procedure for computing the DC value of a CCR is more general than that presented by Desai [8], since it handles the intermediate voltage levels generated by ratioed logic, and differentiates automatically between the drive strengths of logic transistors and weak holders. If this generality is not required, a purely BDD-based approach such as Desai's may be substituted. Alternatively, the multi-strength approach used by COSMOS [4] has also been shown to work well for most digital circuits.

Since all computations are performed symbolically using MTBDDs and symbolic algebra, the final delay MTBDD $\mathbf{T_{delay}}$ encodes the correct Elmore delay under all input assignments. In general, $\mathbf{T_{delay}}$ can be of exponential size with respect to the number of inputs to the CCR. Typically, the CCRs being analyzed are quite small and this exponential possibility is not a concern. Fortunately, larger CCRs tend to contain regularities that can be captured by subgraph sharing in the MTBDD data structure. Figure 7 shows

```
1   Refine( Network N, T_delay )
2       T_refined ← [∞]
3       while (T_delay ≠ [∞])
4           d_min ← MtbddMinTerminal(T_delay)
5           Equal ← MtbddEqual(T_delay, d_min)
6           cube ← GetRandomCube(Equal)
7           ∀ nodes n ∈ N
8               n.v ← Evaluate(n.value, cube)
9           d_refined ← TETA(N)
10          T_refined ← MtbddITE(Equal, [d_refined], T_refined)
11          T_delay ← MtbddITE(Equal, [∞], T_delay)
12      return T_refined
```

Figure 8: Delay Refinement

the $\mathbf{T_{delay}}$ that results from computing the symbolic Elmore delay of a wide dynamic NOR gate. We see that we only need one terminal for each number of pulldowns that can be on simultaneously, and that there are a large number of reconverging paths in the MTBDD. Thus for circuits of this type, the delay MTBDD will only be of quadratic size, rather than exponential. In our experience, pathological cases are extremely rare. In fact, we fairly easily constructed the delay MTBDD for a 64-bit barrel-shifter containing more than 8000 transistors in a single CCR.

2.4 Refining the Delay Values

As the results in Section 3 will demonstrate, the Elmore delay is a fairly poor estimate of the stage delay. However, we have found it to be quite effective at separating input patterns into equivalent delay classes. Based on this observation, we have implemented a methodology that refines the symbolic Elmore delay by selecting a sensitizing input assignment from each delay class and recomputing the delay using a SPICE-like circuit simulator. In this way, the symbolic Elmore delay becomes a heuristic pre-processing routine for selecting input assignments.

The circuit simulator we have been working with is TETA [1, 7], from Carnegie Mellon. It is essentially a fast, callable circuit simulator with accuracy comparable to SPICE. By using a successive chord integration method and a table-lookup model for I_{ds} currents, TETA can re-use expensive LU factorization results across multiple timesteps and input stimuli. Thus, it is ideally suited to quickly evaluating sets of delays on a single network under multiple input assignments.

The algorithm for refining the symbolic Elmore delay MTBDD is shown in Figure 8. For each terminal of the delay MTBDD, we select a sensitizing assignment, evaluate the node values under that assignment, and compute the delay using TETA. The refined delay MTBDD is constructed with a series of *MtbddITE* operations, and we terminate once we have refined each terminal of the Elmore MTBDD.

This algorithm is perhaps deceptively simple, and we discovered a number of difficulties in implementing it in practice. We are effectively performing mixed-mode logic and circuit simulation, where the conversion between the two modes is performed at each CCR boundary. Since logic and circuit simulation operate at such widely separated levels of abstraction, there are bound to be substantial mismatches in results. While we found that the vast majority of CCR analyses completed flawlessly, significant special casing code was required to handle conflicts. Among other things, we were forced to handle mismatching DC values, especially those due to aggressively ratioed logic or charge-sharing glitches. In ad-

```
1   MinMaxTransitionDelay( Network N, ⟨in, out, in0, out0, R⟩ )
2       out.value ← out0
3       ∀ inputs i ∈ N
4           i.value ← boolean variable xi
5       in.value ← in0
6       f0 ← ComputeDC(out)
7       in.value ← ¬in0
8       f1 ← ComputeDC(out)
9       S ← f0 ⊕ f1
10      T ← R ∧ S
11
12      ∀ inputs i ∈ N
13          i.value ← i.value ⊕ T
14
15      ∀ internal nodes n ∈ N
16          n.value ← out0
17      M ← SymbolicComputeDelay(out, ¬out0)
18      MinDelay ← MtbddMinTerminal(M)
19
20      ∀ internal nodes n ≠ out ∈ N
21          n.value ← ¬out0
22      M ← SymbolicComputeDelay(out, ¬out0)
23      MaxDelay ← MtbddMaxTerminal(M)
24
25      return⟨MinDelay, MaxDelay⟩
```

Figure 9: Min/Max Stage Delay Computation

dition, circuit behavior ignored by the Elmore approximation (Section 3.4) also resulted in refined delay values that were negative, or transitions that occur outside the expected simulation time-window. Since our initial application is in a symbolic simulation environment, we chose to be conservative by utilizing X's wherever necessary to cover the uncertainties. However, this special-casing code will be highly dependent on the needs of the application in which this delay-calculation scheme is embedded.

2.5 Applications

Armed with these tools, we can perform stage-delay calculation for a number of applications. Each application will primarily differ in the manner of initializing the input and internal nodes. In this section we discuss how to apply these routines to symbolic timing simulation and to static timing analysis.

2.5.1 Static Timing Analysis

For static timing analysis, path tracing will identify a transition for which we need to calculate the minimum and maximum delay. We will assume that transition is specified as the following tuple:

$$\begin{aligned} T &= \langle in, out, in_0, out_0, \mathbf{R}\rangle \\ in &= \text{input node} \\ out &= \text{output node} \\ in_0 &= \text{initial value of input node } \in 0,1 \\ out_0 &= \text{initial value of output node } \in 0,1 \\ \mathbf{R} &= \text{logical restriction function} \end{aligned}$$

For example, $T = \langle a, z, 0, 1, b \oplus c\rangle$ would represent the transition from a rising input node *a* to a falling output node *z*, assuming that side inputs *b* and *c* are mutually exclusive.

The static timing analysis stage-delay algorithm is shown in Figure 9. To determine the setup conditions which enable a transition of this form, we first compute the DC values that result from the initial and final input stimuli. These are combined to determine the conditions **S** under which the output node will switch as desired. The switching constraint **S** is ANDed with the logical restriction function **R** to obtain the final transition condition **T**. Lastly, the inverse of **T** is XORed with all input node values to constrain the acceptable input patterns.

For max delay calculation, we initialize all internal nodes to the output initial value, and call *SymbolicComputeDelay*. The resultant MTBDD contains the delay under each input assignment, so we need only select the maximum terminal value. For min delay calculation, we initialize all internal nodes to the output final value, call *SymbolicComputeDelay* again, and select the minimum terminal.

2.5.2 Symbolic Timing Simulation

For symbolic timing simulation (STS), we require the full generality of these two routines. This is not surprising, since STS was the original motivation behind the development of our approach. Since symbolic timing simulation is event-driven, we repeatedly select an event from the event queue, update node state accordingly and compute the resultant effects. In this way, input and internal node state initialization is taken care of by the event-driven simulation engine. After each event, we merely call *SymbolicComputeDC* to determine output and internal node DC values, and *SymbolicComputeDelay* to obtain the symbolic delays. These delays are then scheduled as new events according to the algorithms in [10].

3. RESULTS

We have implemented this methodology in the symbolic timing simulator STEED [10], and run it on a number of substantial testcases. As a result, we have collected a large number of data points over a wide range of circuit types, including static, domino, DCVSL, pass-gate logic, and some bizarre custom topologies. We generated some of the test cases ourselves, but the majority were supplied by the Compaq Alpha microprocessor design team. The TETA device models are for a 0.18um ST-Microelectronics process.

In accordance with the potential applications of this methodology, we have collected data with respect to three performance metrics. These metrics measure the Elmore delay's absolute accuracy, its ability to detect the min/max delay cases, and its classification of delays into equivalence classes. In all three cases, the TETA delay values are assumed to give the true delays, while the Elmore delay is considered to be an approximation.

3.1 Absolute Accuracy

In terms of absolute accuracy, the Elmore delay does not perform particularly well. Figure 10 shows the percentage errors in Elmore delays vs TETA-based delays, over approximately 54000 simulation cases. Only 50% of all Elmore delay values were within 50% of the TETA-based value.

These inaccuracies are what we would obtain by simply utilizing the Elmore estimate as our delay value, without applying a refinement scheme based on a more accurate model.

3.2 Differentiation

The second performance metric we measured is *differentiation*. Here we are attempting to determine how well the Elmore delay groups different input assignments into equivalence classes. We measured this by generating two additional random input assignments within each equivalence class, and simulating them in TETA.

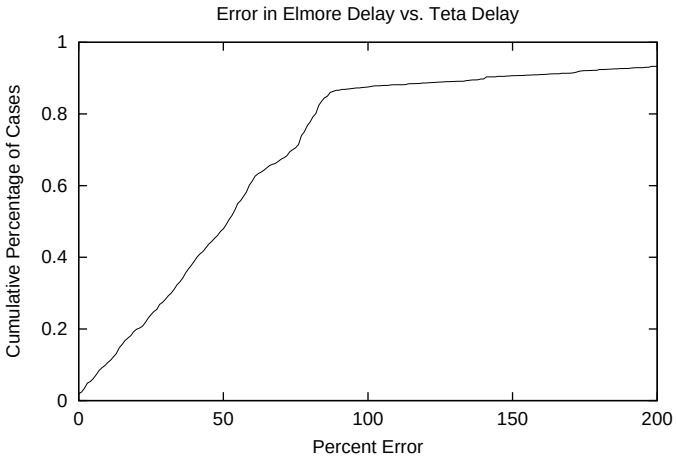

Figure 10: Elmore Delay Accuracy

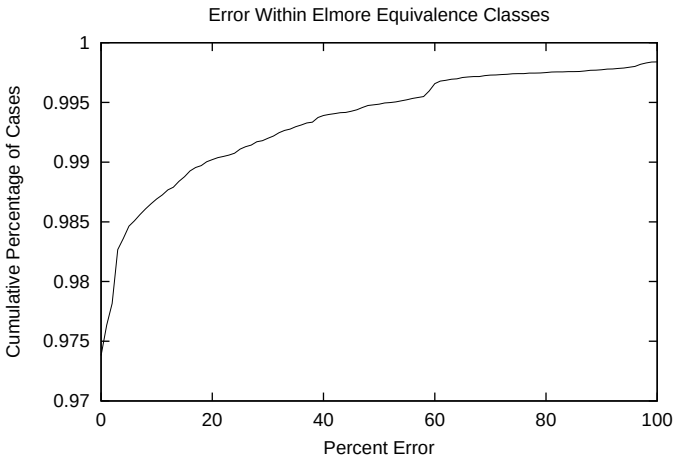

Figure 11: Spread Within Elmore Equivalence Classes

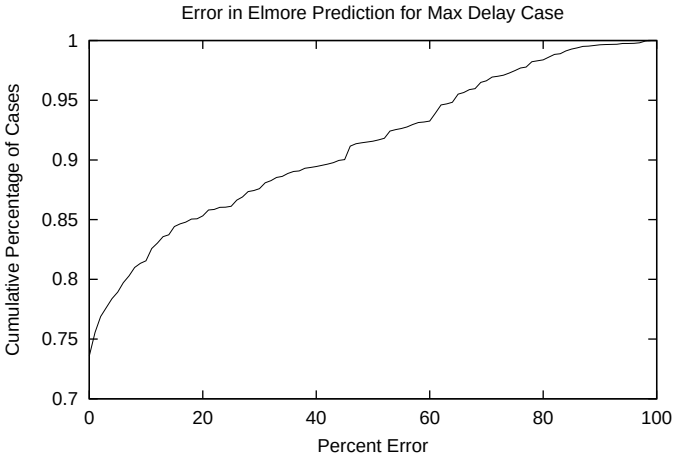

Figure 12: Errors From Maximizing Elmore Delay

The additional data points were compared with the one selected by our algorithm, and the percentage error was computed.

Figure 11 shows a cumulative plot of these errors. Approximately 97% of the the additional data points obtained the same delay value as the original representatives of their equivalence classes, and 99% were within 20%.

This metric demonstrates the errors that can be expected when applying our approach. For the vast majority of cases, the Elmore estimate appears to correctly partition the input assignments into equivalence classes. While there remain circuit effects that are not accounted for by the Elmore estimate (see Section 3.4), they are relatively rare and the frequency of their associated errors tend to drop off quickly as their magnitudes increase.

3.3 Min/Max Selection

The final metric is *min/max selection*, which we measured by comparing the true maximum delay over all input assignments with the refined value of the maximum Elmore delay. This reflects the way in which the Elmore delay is used to identify the maximum delay case in previously published techniques.

Figure 12 shows the relative error due to assuming the maximum Elmore delay case will yield the maximum true delay. Only in about 74% of the cases did the maximum Elmore delay case give the true maximum. Furthermore, in nearly 10% of the cases, the maximum Elmore delay case produced a delay with more than 50% error relative to the true maximum.

This metric highlights the errors associated with previously published approaches, which attempt to maximize the Elmore estimate and then refine that case alone. Since the absolute accuracy of the Elmore estimate is so poor, it is not surprising that we would incur errors by assuming the maximum Elmore delay will lead to the maximum true delay.

3.4 Limitations of the Elmore Estimate

The previous section gives some quantitative information on the potential inaccuracies associated with using the Elmore estimate. Since large amounts of information are lost in constructing the Elmore equivalent circuit, we should expect some circuit-effects to be ignored which could lead to significant errors.

In general, the circuit behavior on nodes that are reachable through conducting transistors from the output node is modeled sufficiently to allow the Elmore delay to identify equivalent assignments. From analysis of a large number of error cases from our experiments, we

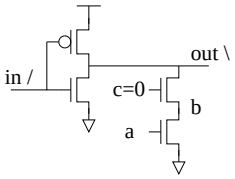

Figure 13: Elmore Delay with S-D Coupling

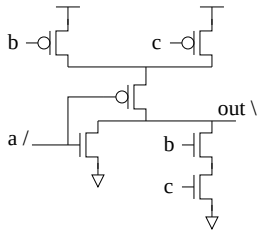

Figure 14: Elmore Delay with Cross-Over Current

have determined that the largest sources of errors are due to circuit behavior on the far side of off or turning-off transistors. By far the largest source of errors are capacitive coupling effects through non-conducting source-drain connections and cross-over current flowing through turning-off transistors. Both effects are inherently excluded by the nature of the Elmore estimate.

Figure 13 shows an example of the error induced by source-drain capacitive coupling. While somewhat contrived, it serves to demonstrate the circuit issues involved. Here we are attempting to determine the delay from *in* rising to *out* falling, given that node *c* is low. Since *c* is low and its associated transistor is non-conducting, the values of *a* and *b* will have no effect on the Elmore delay estimate. However, using a circuit simulator such as SPICE or TETA, we might see a considerable difference between the cases $\langle a = 1, b = 1 \rangle$ and $\langle a = 0, b{=}0 \rangle$. In the former, node *b* will be falling, and through source-drain coupling, will accelerate the fall of *out*. In the latter, node *b* will be stable and *out* will behave as the Elmore delay predicts. In general, the effects of source-drain coupling can be unbounded, and are heavily dependent on process and transistor-sizing.

Figure 14 shows a case in which cross-over current through the turning-off pFET connected to *a* will affect the stage delay. In computing the Elmore delay, we set all inputs to their final values ($a \leftarrow 0$ in this case), replace conducting transistors with resistors, and determine the settling time of the resulting RC network. Therefore, any circuit behavior occurring in the top portion of the pFET chain cannot affect the Elmore computation. However, under circuit simulation, we will see a delay difference between the cases $\langle b = 0, c = 0 \rangle$ and $\langle b = 0, c{=}1 \rangle$. The two cases differ only in the resistance of the top portion of the pFET pullup chain, and thus in the amount of cross-over current that will flow into node *out* while *a* is rising. If *a* is rising slowly enough, and the difference in cross-over current is large, we could see substantial variations in the true delay of these sub-cases despite having the same Elmore estimate.

4. CONCLUSION

We have presented a new technique for computing logic-stage delays in CMOS transistor networks. Our technique leverages MTBDDs to enable the computation of an Elmore delay estimate for all possible input assignments, which effectively groups these assignments in Elmore-equivalent classes. We can extract a representative from each of the classes for refinement of the delay value using circuit simulation. This approach is applicable to a wide range of EDA problems, and demonstrates the power of symbolic methods even in dealing with largely real-valued domains such as timing.

Our approach represents an improvement in accuracy over previously published methods of delay calculation for static timing analysis in that enables direct computation of the Elmore estimate for all input assignments. Given this capability, we can avoid a major source of errors by maximizing the refined delay values directly, rather than assuming the maximal-Elmore case will maximize the true delay.

5. ACKNOWLEDGEMENTS

We would like to thank Emrah Acar for his help in integrating the TETA circuit simulator, and the Compaq Alpha development team and ST Microelectronics for allowing us access to sensitive information for our experiments.

6. REFERENCES

[1] E. Acar and L. T. Pileggi. TETA: Transistor-Level Engine for Timing Analysis. *CMU Internal Report*, 2000.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and Their Applications. *ACM/IEEE International Conference on Computer Aided Design*, pages 188–191, November 1993.

[3] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):79–85, August 1986.

[4] R. E. Bryant. Boolean Analysis of MOS Circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, CAD-6(4):634–639, July 1987.

[5] T. M. Burks and R. E. Mains. Incorporating Signal Dependencies into Static Transistor-Level Delay Calculation. *TAU: ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 110 – 119, 1997.

[6] C. Y. Chu. *Improved Models for Switch-Level Simulation*. PhD thesis, Stanford University, October 1988.

[7] F. Dartu and L. T. Pileggi. TETA: Transistor-Level Engine for Timing Analysis. *Proceedings of the Design Automation Conference*, pages 595–598, June 1998.

[8] M. P. Desai and Y. T. Yen. A Systematic Technique for Verifying Critical Path Delays in a 300MHz Alpha CPU Design Using Circuit Simulation . *Proceedings of the Design Automation Conference*, pages 125–130, June 1996.

[9] C. B. McDonald and R. E. Bryant. Symbolic Functional and Timing Verification of Transistor Level Circuits. *ACM/IEEE International Conference on Computer Aided Design*, pages 526–530, November 1999.

[10] C. B. McDonald and R. E. Bryant. Symbolic Timing Simulation Using Cluster Scheduling. *Proceedings of the Design Automation Conference*, June 2000.

[11] C. J. Terman. RSIM - A Logic-Level Timing Simulator. *International Conference on Computer Design*, pages 437–440, October 1983.