Modeling Uncertainty in RC Timing Analysis

Cheryl L. Harkness
Daniel P. Lopresti

Department of Computer Science
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-89-38
September 1989

Modeling Uncertainty in RC Timing Analysis †

Cheryl L. Harkness

Daniel P. Lopresti

Department of Computer Science

Brown University

Providence, RI 02912

Abstract

Even subtle disturbances in the manufacturing process used to fabricate a chip introduce variations in the values of circuit parameters (e.g., capacitances, resistances, transistor sizes). Although expected values can be obtained for all such parameters, the true values are likely to differ from those stated. Given the range of possible values for each input parameter, our goal is to determine bounds on the resulting timing behavior produced by these variations.

We propose representing uncertain parameters as intervals and present a theoretical framework based on interval algebra for manipulating these ranges. To illustrate this methodology, we modify an existing RC analysis algorithm (Crystal's PR-Slope model) to create one which computes worst-case delay bounds when given uncertain input parameters. We provide proofs of correctness for our approach and test its performance. Analysis shows that our algorithm is much more efficient than Monte Carlo simulation and produces bounds with similar accuracy.

†This research was supported in part by the Defense Advanced Research Projects Agency under ONR contract number N00014-83-K-0146. Daniel Lopresti is also supported by the National Science Foundation under contract MIP-87-10745.

1. Introduction

Timing verification is an important aspect of VLSI design. Its purpose is to predict the operating speed of a VLSI chip before it is fabricated. This speed is a function of three components: structure, technology, and the manufacturing process. The first two of these are well known to chip designers. Consider, for example, carry-propagate and carry-lookahead adders. Although both perform the same function, they have different structures and hence operating speeds. A comparison of CMOS and nMOS technologies provides an example of the effects of technology on performance. By replacing slow depletion mode pull-ups by complementary logic, CMOS designs achieve faster operating speeds than their nMOS counterparts. The third component, the manufacturing process, is also an important factor in chip performance. During fabrication, even slight variations in any of a multitude of processing parameters (exposure time, temperature, gas pressure, etc.) produce variations in the process and consequently in chip parameters. For example, over and under-etching, oxide growth rates, and ion implantation levels all affect the electrical characteristics of the resulting circuit. In particular, they affect the transistor sizes, threshold voltages, resistance factors, and capacitance factors used to perform RC analysis. As a result, the circuit specifications given for each technology are only approximations to the true values. These parameters actually have a range of possible values, typically varying between 1 and 20 percent from those stated. Currently, most timing verifiers use only the nominal values given to estimate circuit performance, but a more important issue is how the circuit will operate with other possible combinations of these values.

While there are many timing verifiers in use today [Hitc82, Joup83, McWi80, Oust83, Oust84, Oust86, Wall86], most do not address the problem of parameter uncertainty. A few [Hitc82, McWi80, Wall86], however, provide partial solutions to this problem. These logic-level timing verifiers use bounds on the delay of each macro-component (AND gates, OR gates, Flip-Flops, etc.) to compute bounds on the performance of the entire circuit, but none of them discusses how these component delays are derived initially. We present a method for RC analysis that works at the transistor-level and can be used to compute these macro-component delays.

The most widely-used approach for modeling parameter uncertainty within existing simulation systems is the Monte Carlo method [Sobo75] or the "Method of Statistical Trials." With this technique, analysis is repeated for random combinations of values chosen from within the acceptable range of each parameter. Once computed, these results are tabulated and returned. Unfortunately, accurately determining bounds on the behavior of a circuit requires a large number of trials; thus, while Monte Carlo simulation can be effective, it is also very time consuming. Our goal is to achieve similar accuracy in less time.

In this paper, we present a theoretical framework based on interval algebra for handling param-

eter uncertainty in RC analysis. We then illustrate our approach by modifying an existing timing analyzer, Crystal [Oust83, Oust84, Oust86], to create an RC analysis algorithm that incorporates this framework. Given a circuit description and bounds on all circuit parameters (transistor lengths, widths, threshold voltage, capacitances, and resistances), our algorithm computes best and worst-case bounds on the delay of the circuit. Although we employ Crystal for these experiments, the framework may be applied to other RC analysis algorithms.

In the next section, we briefly describe the RC analysis algorithm used in Crystal. Following that, we review interval algebra and its properties. In section 4, we incorporate interval algebra into the RC analysis algorithm to produce an interval version of it and present a proof of correctness. Finally, we conclude with an analysis of the performance of our method and a comparison with Monte Carlo simulation.

2. RC Analysis in Crystal

Crystal is a data-independent timing analyzer for sequential MOS VLSI circuits. Given a design description extracted from a mask layout, Crystal computes and returns a list of the *critical* paths through the circuit. This algorithm has three phases: breaking the circuit into pieces, estimating delay through the pieces (using RC analysis), and using these delays to locate critical paths.

In the first phase, Crystal decomposes the circuit into chains of transistors called *stages*. Each stage represents a path through one or more transistors originating at a strong signal source (Vdd or Gnd) and terminating at an output node or transistor gate (called the *target*). The stage is enabled by the last transistor in the path to switch on. This transistor is called the *trigger*. When analyzing pull-up paths in nMOS circuits, the trigger transistor is not on the path from source to target; therefore, the depletion load transistor closest to the target acts as the trigger transistor for the stage. The target transistor for one stage becomes the trigger transistor for the next stage. For an example of circuit decomposition, see Figure 1.

In the second phase of critical path analysis, Crystal estimates the delay of each stage. Given the sizes and types of the transistors in the stage, the parasitic resistances and capacitances of the nodes, the trigger transistor, and the waveform at the gate of the trigger transistor, a delay modeler generates the resulting waveform at the target of the stage. Crystal supports three different models for computing stage delay: the linear RC model, the Slope model, and the PR-Slope model. Of the three models, the PR-Slope model is the most accurate. We will discuss this method in detail in the next section.

In the final phase of analysis, Crystal employs these stage delays to locate critical paths in the

design using depth-first search. For further details, see [Oust83].

2.1. The PR-Slope Model

To estimate delay, the PR-Slope model begins by determining a resistance and capacitance for each node and transistor along the stage. Node capacitances and resistances can either be provided by the circuit extractor or, in some older systems, computed by Crystal directly from the geometry of the node. In the latter case, Crystal is given an area and perimeter for each substrate layer in the node and computes the capacitance and resistance contribution of each. The respective substrate resistances and capacitances are then summed to compute the resistance and capacitance of the node. In addition to the above contributions, the gate-to-source capacitances of adjacent branch transistors are also included in the node calculations.

A transistor is modeled as a perfect switch in series with a resistor. This resistance is fixed for non-trigger transistors and is computed from the transistor type and geometry. Specifically, a resistance factor is determined by table lookup on transistor type, usage, and signal value. This factor is then multiplied by the length/width ratio of the transistor to generate an estimate of its effective resistance.

The effective resistance of the trigger transistor is more accurately modeled. This resistance is not a constant, but a function of the input waveform at the gate of the transistor. In particular, resistance is a function of the *rise time* of the gate signal, which is the rate at which the signal is changing when it crosses the threshold voltage. The faster the gate signal changes, the lower the effective resistance across the transistor. To model this effect, the timing analyzer combines the rise time at the transistor gate, the load being driven, and the transistor size to compute a *rise time ratio*. It then consults a table indexed by these rise time ratios to find the resistance factor of the trigger transistor. This factor is multiplied by the aspect ratio of the transistor to determine its resistance.

To calculate the effective capacitance of each transistor, Crystal determines the capacitance factor by table lookup on transistor type and multiplies this value by the area of the transistor, All transistor characterization tables used in these computations are generated by running SPICE simulations on sample circuits and compiling the results.

Having calculated the resistances and capacitances of each node and transistor in the path, Crystal uses these values to compute the total delay of the stage. The PR-Slope model employs the RC equations presented by Penfield and Rubinstein in [Rubi83]. Since a Crystal stage is just an RC

line, these delay equations simplify to the following sum:

$$D = \sum_{t<i\leq T} \sum_{S<j\leq i} C_i R_j \tag{1}$$

where t represents the trigger transistor, T the target node, S the source node, R_j the resistance of element j, and C_i the capacitance at element i. For example, the delay of Stage 1 in Figure 1 is given by the following equation:

$$D = C_B(R_1 + R_B) + C_3(R_1 + R_B + R_3) + C_D(R_1 + R_B + R_3 + R_D). \tag{2}$$

In this example, the node capacitances and resistances are denoted by alphabetic subscripts and the transistor capacitances and resistances are denoted by numeric subscripts.

3. Interval Algebra

When given the empirically-determined minimum and maximum values for an input parameter, it is natural to suppose that the parameter could assume any value within these bounds. We therefore represent uncertain values by bounding intervals and use interval analysis techniques to evaluate functions containing these uncertain values. In this section, we summarize some important aspects of interval algebra [Kuli81, Moor79].

An interval $[a, b]$ over the set of real numbers $\Re$ is defined as the set $\{x \mid x \in \Re, a \leq x \leq b\}$. A *degenerate* interval $[a, a]$ is an interval which contains only the one element a. Let I be the set of all intervals over $\Re$; thus, $I \equiv \{[a, b] \mid a, b \in \Re \text{ and } a \leq b\}$. The interval arithmetic operations $\oplus$ (addition), $\ominus$ (subtraction), $\odot$ (multiplication), and $\oslash$ (division) over the set I are defined as follows:

$$[a, b] \oplus [c, d] \equiv [a + c, b + d] \tag{3}$$

$$[a, b] \ominus [c, d] \equiv [a - d, b - c] \tag{4}$$

$$[a, b] \odot [c, d] \equiv [min(ac, ad, bc, bd), max(ac, ad, bc, bd)] \tag{5}$$

$$[a, b] \oslash [c, d] \equiv [a, b] \odot [1/c, 1/d] \quad \textit{provided that } 0 \notin [c, d] \tag{6}$$

Interval addition and multiplication are both commutative and associative. The degenerate intervals $[0, 0]$ and $[1, 1]$ form the additive and multiplicative identities respectively. The distributive property holds only in the following special case: $\hat{x} \odot (\hat{y} \oplus \hat{z}) = (\hat{x} \odot \hat{y}) \oplus (\hat{x} \odot \hat{z})$ for $\hat{x}, \hat{y}, \hat{z} \in I$, if either (1) $\hat{x}$ is a degenerate interval or (2) $\hat{y} \odot \hat{z} > [0, 0]$. However, the subdistributivity property always holds; therefore, $\hat{x} \odot (\hat{y} \oplus \hat{z}) \subseteq (\hat{x} \odot \hat{y}) \oplus (\hat{x} \odot \hat{z})$. Non-degenerate intervals have no additive or multiplicative inverses; in particular, $\hat{x} \oslash \hat{x} \neq [1, 1]$ unless $\hat{x}$ is degenerate. Another important

property of interval arithmetic is *monotonic inclusion*: if $\hat{w}, \hat{x}, \hat{y}, \hat{z} \in I$ and $\hat{w} \subset \hat{y}$ and $\hat{x} \subset \hat{z}$, then $\hat{w} * \hat{x} \subset \hat{y} * \hat{z}$ for any of the interval operators $*$ given above.

4. Bounds for RC Analysis

We now incorporate interval algebra into the PR-Slope model described in Section 2.1 to create an interval version of the RC analysis algorithm. We begin with a few definitions. Let $F(x_1, x_2, ..., x_n)$ be any real-valued, arithmetic function of n independent variables such that $F : \Re^n \rightarrow \Re$ and $x_i \in \Re$. Now assume that the value of each variable, x_i, is not precisely known. Instead, we are given a range of possible values for each x_i, $\hat{x}_i = [min_i, max_i]$. The tightest bounds on the solutions to F are found by evaluating the function over all possible combinations of input values and taking the union of the results. This is called the *United Extension of F* [Moor79, p. 18] and is defined by the following:

$$\hat{F}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \equiv \bigcup_{x_i \in \hat{x}_i} F(x_1, x_2, ..., x_n) \tag{7}$$

where $\hat{F} : I^n \rightarrow I$ and each $\hat{x}_i \in I$. For timing analysis, $F(x_1, x_2, ..., x_n)$ represents delay Equation 1 and the x_i represent the capacitances, resistances, and transistor sizes used to compute this delay. If the value of any of these variables is not precisely known, then the tightest bounds on the solution may be found by computing $\hat{F}$. Because the intervals are defined over $\Re$, each contains an infinite number of possible assignments; therefore, we cannot compute $\hat{F}$ directly. We can approximate $\hat{F}$ by dividing each interval into $k-1$ subintervals and evaluating F at the k endpoints. Since each variable x_i can assume any one of these k possible values, we must evaluate function F k^n times to compute its United Extension. Even for $k = 2$, this computation requires 2^n invocations of F. Since the number of variables n tends to be large even for small stages (e.g., the PR-Slope Model requires six variables per transistor and a minimum of two variables per node), this is not a feasible approach.

4.1. A First Approximation

Let $\mathcal{F} : I^n \rightarrow I$ be the interval equivalent of F derived by replacing the basic arithmetic operators $+$, $-$, $\cdot$, and $/$ by their respective interval counterparts $\oplus$, $\ominus$, $\odot$, and $\oslash$. We now show that $\mathcal{F}$ approximates $\hat{F}$.

Lemma 4.1 *If $\hat{x}_1, \hat{x}_2, ..., \hat{x}_n \in I$ are degenerate, then $F(x_1, x_2, ..., x_n) = \mathcal{F}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$.*

Proof. By induction on the number of operators (m) in F. By convention, if $\hat{x} = [x, x]$, then we write $\hat{x} = x$. Over the set of degenerate intervals, each interval operator is equivalent to its respective

discrete operator as shown by the following: for all degenerate $\hat{a}, \hat{b} \in I$

$$\hat{a} \oplus \hat{b} = [a, a] \oplus [b, b] = [a + b, a + b] = a + b$$

$$\hat{a} \ominus \hat{b} = [a, a] \ominus [b, b] = [a - b, a - b] = a - b$$

$$\hat{a} \odot \hat{b} = [a, a] \odot [b, b] = [ab, ab] = ab$$

$$\hat{a} \oslash \hat{b} = [a, a] \oslash [b, b] = [a/b, a/b] = a/b.$$

Note also that the set of degenerate intervals is closed over these interval operators; thus, when evaluated over this set, $\mathcal{F}$ always returns a degenerate interval. For the basis ($m = 1$), F must be one of the following four functions: $F = a + b$, $F = a - b$, $F = ab$, or $F = a/b$. We have just demonstrated that $F = \mathcal{F}$ over the set of degenerate intervals for each of these cases. Now we are given that $F = \mathcal{F}$ over the set of degenerate intervals for all functions F consisting of less than m operators. We must show that $F = \mathcal{F}$ for functions with m operators. Any arithmetic function can be written as a binary parse tree, where each parent node contains one of the four arithmetic operators and each child node contains either a value or a subtree representing a subexpression of the function. If we remove the root of the parse tree, we are left with two subtrees representing subfunctions. If the original function had m operators, each of these subfunctions must contain less than m operators, since we removed the operator at the root of the parse tree. Let F represent the original function, let F_1 and F_2 represent these subfunctions, and let $\star$ be the operator at the root of F. Clearly, $F = F_1 \star F_2$. By the definition of $\mathcal{F}$, we know that $\mathcal{F} = \mathcal{F}_1 * \mathcal{F}_2$ where $*$ is the interval counterpart of the $\star$ operator. Applying the assumption, we see that $\mathcal{F}_1 = F_1$ and $\mathcal{F}_2 = F_2$ over the set of degenerate intervals. Since $*$ is equivalent to $\star$ over the set of degenerate intervals, we conclude that $F = \mathcal{F}$ over this set. □

Theorem 4.1 $\hat{F}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \subset \mathcal{F}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$

Proof. Let $y_1, y_2, ..., y_n$ be any legal assignment of values to the variables $x_1, x_2, ..., x_n$. For each y_i, $y_i = \hat{y}_i = [y_i, y_i] \in \hat{x}_i = [min_i, max_i]$; therefore, $\hat{y}_i \subset \hat{x}_i$. Hence, by the principle of monotonic inclusion, $\mathcal{F}(\hat{y}_1, \hat{y}_2, ..., \hat{y}_n) \subset \mathcal{F}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$. Applying Lemma 4.1, we note that $F(y_1, y_2, ..., y_n) = \mathcal{F}(\hat{y}_1, \hat{y}_2, ..., \hat{y}_n)$. Since our choice of values was not special, we conclude that $F(y_1, y_2, ..., y_n) \subset \mathcal{F}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$ for all legal assignments $y_1, y_2, ..., y_n$. Since $\hat{F}$ is just the union of the values $F(y_1, y_2, ..., y_n)$, we know that $\hat{F} \subset \mathcal{F}$. □

According to the above theorem, we can approximate the United Extension of our delay function by replacing the discrete mathematical operators in the function with the corresponding interval ones. This substitution yields an interval version of the algorithm that computes bounds on the delay of the circuit much more efficiently than exhaustive enumeration.

4.2. Improving these Bounds

Up to this point, our discussion has been general and our proofs apply to any real-valued, arithmetic function F. Now, we discuss a particular example: delay equation 1. Let D denote this function, let $\hat{D}$ represent its united extension, and let $\mathcal{D}$ be the corresponding interval delay function. While $\mathcal{D}$ does indeed yeild correct bounds on the delay of the circuit, they tend to be overly conservative. One reason for this is the lack of multiplicative inverses in the interval algebra. For example, let $\hat{x} = [a, b]$, then $\hat{x} \oslash \hat{x} = [a/b, b/a]$. This yields the expected result $[1, 1]$ only when $a = b$; however, $[1, 1] \subset \hat{x} \oslash \hat{x}$ for all $\hat{x}$. Crystal's delay equation contains several such instances. One occurs when we compute the RC contribution of transistor j as in the following:

$$\begin{aligned} C_j \odot R_j &= CperArea_j \odot Area_j \odot RperSquare_j \odot Aspect_j \\ &= CperArea_j \odot L_j \odot W_j \odot RperSquare_j \odot L_j \oslash W_j \\ &= CperArea_j \odot RperSquare_j \odot L_j^2 \odot W_j \oslash W_j \\ &\supset CperArea_j \odot RperSquare_j \odot L_j^2 \end{aligned} \tag{8}$$

To alleviate this problem, we must carefully tune the delay equations to eliminate any unnecessary divisions. Specifically, we rewrite function D to create an equivalent version D_2 by expanding the RC terms and eliminating as many of these divisions as possible. For example, the RC terms for transistor j are transformed as follows:

$$\begin{aligned} C_j(R_1 + ... + R_j) &= C_j(R_1 + ... + R_{j-1}) + C_j \cdot R_j \\ &= C_j(R_1 + ... + R_{j-1}) + CperArea_j \cdot Area_j \cdot RperSquare_j \cdot Aspect_j \\ &= C_j(R_1 + ... + R_{j-1}) + CperArea_j \cdot RperSquare_j \cdot L_j^2 \end{aligned} \tag{9}$$

Similarly, the RC terms for node k are rewritten as follows:

$$\begin{aligned} C_k(R_1 + ... + R_k) &= C_k(R_1 + ... + R_{k-2}) + C_k \cdot R_{k-1} + C_k \cdot R_k \\ &= C_k(R_1 + ... + R_{k-2}) + \\ &\quad (CRest_k + CperWidth_{k-1} \cdot W_{k-1})R_{k-1} + C_k \cdot R_k \\ &= C_k(R_1 + ... + R_{k-2}) + CRest_k \cdot R_{k-1} + \\ &\quad CperWidth_{k-1} \cdot W_{k-1} \cdot R_{k-1} + C_k \cdot R_k \\ &= C_k(R_1 + ... + R_{k-2}) + CRest_k \cdot R_{k-1} + \\ &\quad CperWidth_{k-1} \cdot W_{k-1} \cdot RperSquare_{k-1} \cdot Aspect_{k-1} + C_k \cdot R_k \\ &= C_k(R_1 + ... + R_{k-2}) + CRest_k \cdot R_{k-1} + \\ &\quad CperWidth_{k-1} \cdot RperSquare_{k-1} \cdot L_{k-1} + C_k \cdot R_k \end{aligned} \tag{10}$$

where $C_k = CRest_k + CperWidth_{k-1} \cdot W_{k-1}$. In other words, $CperWidth_{k-1} \cdot W_{k-1}$ is the portion of node k's capacitance contributed by transistor $k-1$ and $CRest_k$ is the remainder.

Once we have D_2, we can replace the arithmetic operators with the corresponding interval operators to create an interval version, $\mathcal{D}_2$. Although the delay functions D and D_2 are equivalent, their interval counterparts $\mathcal{D}$ and $\mathcal{D}_2$ are not. Since $\mathcal{D}_2$ contains fewer interval divisions, the bounds that it produces are always as good as those produced by $\mathcal{D}$. Often they are tighter. In our next theorem, we show that $\mathcal{D}_2$ approximates $\hat{D}$, producing bounds that are at least as good as $\mathcal{D}$.

Theorem 4.2 $\hat{D}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \subset \mathcal{D}_2(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \subset \mathcal{D}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$

Proof. We begin by proving that $\hat{D}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \subset \mathcal{D}_2(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$. Applying Theorem 4.1, we note that $\hat{D}_2 \subset \mathcal{D}_2$. But $D = D_2$; therefore, $\hat{D} \subset \mathcal{D}_2$. Now we must show that $\mathcal{D}_2(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n) \subset \mathcal{D}(\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$. Since all our variables (capacitance factors, resistance factors, transistor lengths and widths) are positive real numbers, the special case of the distributive property holds. Specifically,

$$\begin{aligned} C_k \odot (R_1 \oplus R_2 \oplus ... \oplus R_k) &= C_k \odot (R_1 \oplus R_2 \oplus ... \oplus R_{k-1}) \oplus C_k \odot R_k \\ &= C_k \odot (R_1 \oplus R_2 \oplus ... \oplus R_{k-2}) \oplus C_k \odot R_{k-1} \oplus C_k \odot R_k. \end{aligned}$$

Hence $\mathcal{D}$ only differs from $\mathcal{D}_2$ in the simplified RC terms. We noted above that $[1,1] \subset \hat{x} \oslash \hat{x}$; therefore, by monotonic inclusion,

$$CperArea_k \odot RperSquare_k \odot L_k^2 \subset CperArea_k \odot Area_k \odot RperSquare_k \odot Aspect_k.$$

$$CperWidth_k \odot RperSquare_k \odot L_k \subset CperWidth_k \odot W_k \odot RperSquare_k \odot Aspect_k$$

Applying the principle of monotonic inclusion again, we conclude that $\mathcal{D}_2 \subset \mathcal{D}$. □

5. Performance Analysis

We have implemented an interval version of the RC analysis algorithm using the revised interval delay function $\mathcal{D}_2$. We also implemented a Monte Carlo version of D for comparison. To select values from the range of each variable, the Monte Carlo program employs a random number generator with a uniform distribution. It then uses the original delay function D to compute the delay for each trial and returns the minimum, maximum, and average delays calculated. Both implementations were written in the C programming language and run on an Encore Multimax. We now present an analysis of the performance of our interval algorithm as compared to Monte Carlo simulation.

Algorithm	Number of Operations			
	+	−	·	/
D (original)	$10N+1$	4	$9N+3$	$N+2$
D_2 (revised)	$18N-2$	4	$23N-10$	$2N+3$
$\mathcal{D}_2$ (interval)	$36N-4$	8	$46N-20$	$4N+6$

Table 1: Worst-Case Operations Count for a Single Stage with N Transistors

5.1. Speed of the Algorithm

We used two measures to compare the speed of each implementation. First, we counted the occurrences of each arithmetic operation in each implementation. Table 1 contains a breakdown of operations for each of the three functions tested – the original delay function D used by Crystal and the Monte Carlo program, the revised delay function D_2, and the interval version, $\mathcal{D}_2$. In the analysis, N represents the number of transistors in a stage. The table shows that the interval version of the PR-Slope model contains approximately four times as many operations in each category as the original delay function; therefore, we expect that this algorithm will require roughly four times longer to compute the stage delay.

To confirm this hypothesis, we ran all three versions on identical circuits and recorded their response times. Our experiments support this conjecture, showing that the interval version is roughly four times slower than the original. Asymptotically, both versions have the same time complexity, $O(N)$.

Since the Monte Carlo program uses the original function D, we note that our interval algorithm computes delay bounds for a stage in about the time required to perform four Monte Carlo trials. Our experiments, however, show that 50,000 Monte Carlo trials are often required to produce comparable delay bounds. Our interval algorithm, therefore, offers a vast improvement in speed.

5.2. Quality of the Bounds

To determine the quality of the bounds produced by both the Monte Carlo (50,000 trials) and interval methods, we conducted a number of experiments comparing their bounds to the theoretical bounds ($\hat{D}$) for the given circuit. For these experiments, the theoretical bounds were determined by fixing as many of the variables as possible and computing the United Extension. We also tested nominal delay (i.e., the delay computed using the mean values of the input parameters). For all methods, delay was calculated in two stages. First, we determined the resistance and capacitance

Stage	Interval		Monte Carlo		Nominal	
Length	lower	upper	lower	upper	lower	upper
1	0	0	-4.42	3.91	-18.98	16.08
3	0.26	-0.24	-4.87	4.67	-16.23	13.49

Table 2: Average % Error for Upper and Lower Delay Bounds

factors for all transistors by table lookup. Then we used the values obtained to compute delay.

Figures 2 and 3 illustrate the relationship between delay and stage length (the number of transistors in a stage). As Figure 2 demonstrates, the Monte Carlo and nominal delay methods always underestimate the bounds, while the interval approach always overestimates them. As shown in Figure 3, both the Monte Carlo and interval methods produce bounds with similar accuracy, coming within 10% of the theoretical bounds for all stage lengths tested. The graph further shows that the error of both methods grows with the length of the stage. Fortunately, stages encountered in most circuits are typically small, containing fewer than five transistors. As the graph demonstrates, nominal delay, the method currently employed in most timing analyzers, provides the poorest estimate of possible timing behaviors.

We also analyzed a portion of the Brown Systolic Array (B-SYS)[Hugh89], a 2-μ CMOS design. In particular, we surveyed 33 stages extracted from the ALU of the chip (See Figure 4). The stages ranged in length from one to three transistors, with an average length of 2.45 transistors. The average percent error for each method is displayed in Table 2. As shown, the interval method produced the tightest bounds, coming within 0.3% of the theoretical bounds on average. The Monte Carlo method (10,000 trials) produced an average error of about 4.6%, while the nominal delay had an average error of 15.6%.

6. Summary

In this paper, we presented a method for modeling the effects of uncertainty in RC analysis. Representing uncertain parameters as intervals, we used interval algebra to create a mathematical framework for manipulating these uncertain values. We then modified an existing RC analysis algorithm (Crystal's PR-Slope model) to test our approach. Although Crystal was chosen for our experiments, the same techniques may be applied to other timing analysis algorithms.

Given a circuit description with uncertain input parameters, our algorithm computes bounds on the delay. We claimed that the delay estimates produced by our algorithm approximate the

theoretical delay bounds for the circuit and provided proofs to support our hypothesis. We then implemented our algorithm and tested its performance. Analysis shows that its operating speed is not significantly slower than that of the non-interval RC analysis algorithm.

When compared to Monte Carlo simulation, our algorithm is much more efficient, operating several thousand times faster for the same quality results. For stages with few transistors, the accuracy of both methods is similar: both come within 10% of the theoretical bounds. We are currently working on methods to further improve the accuracy of these bounds. We also plan to extend these results to create an interval timing verifier that combines uncertain stage delays to compute bounds on the total delay of the circuit.

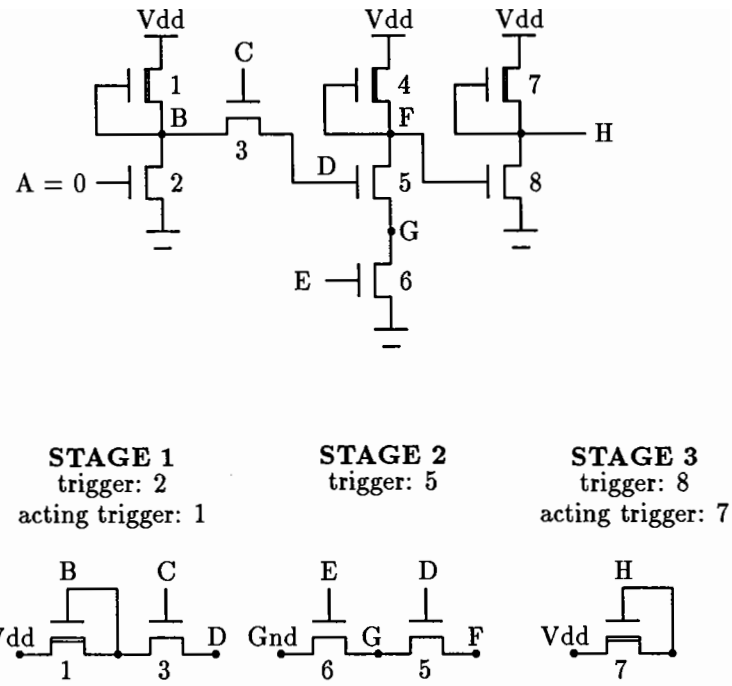

Figure 1: Decomposing a circuit into stages

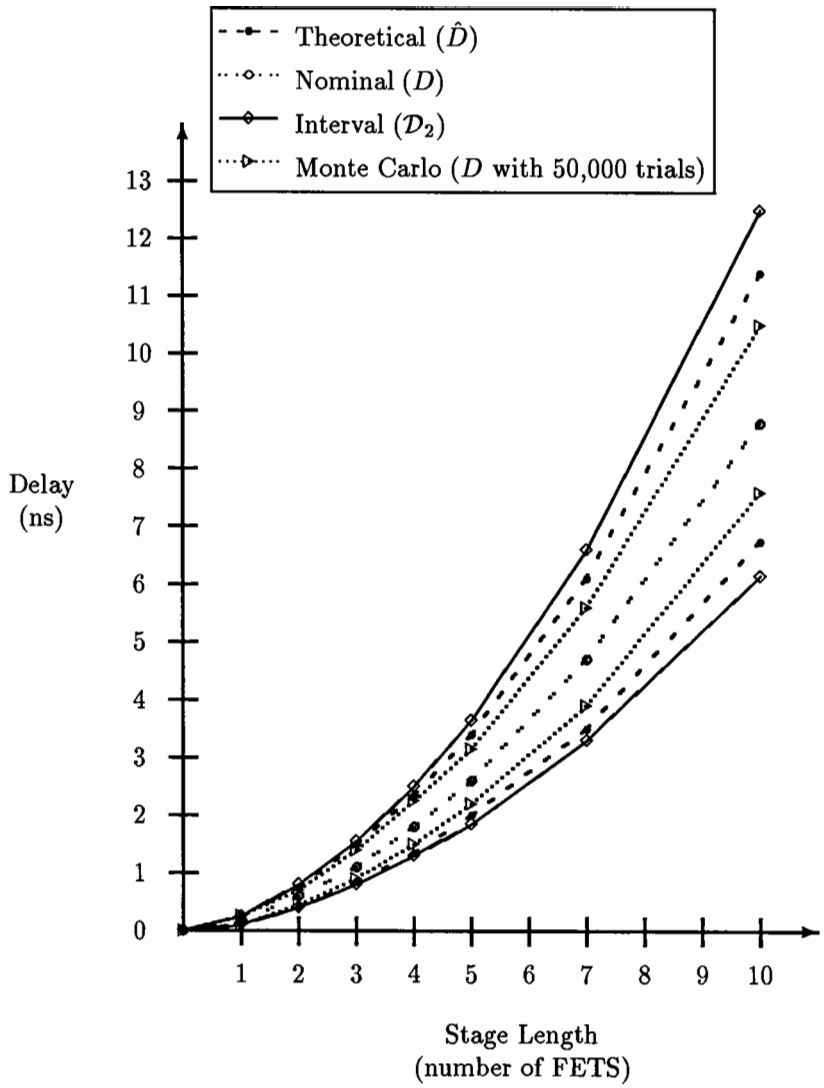

Figure 2: Delay vs. Stage Length

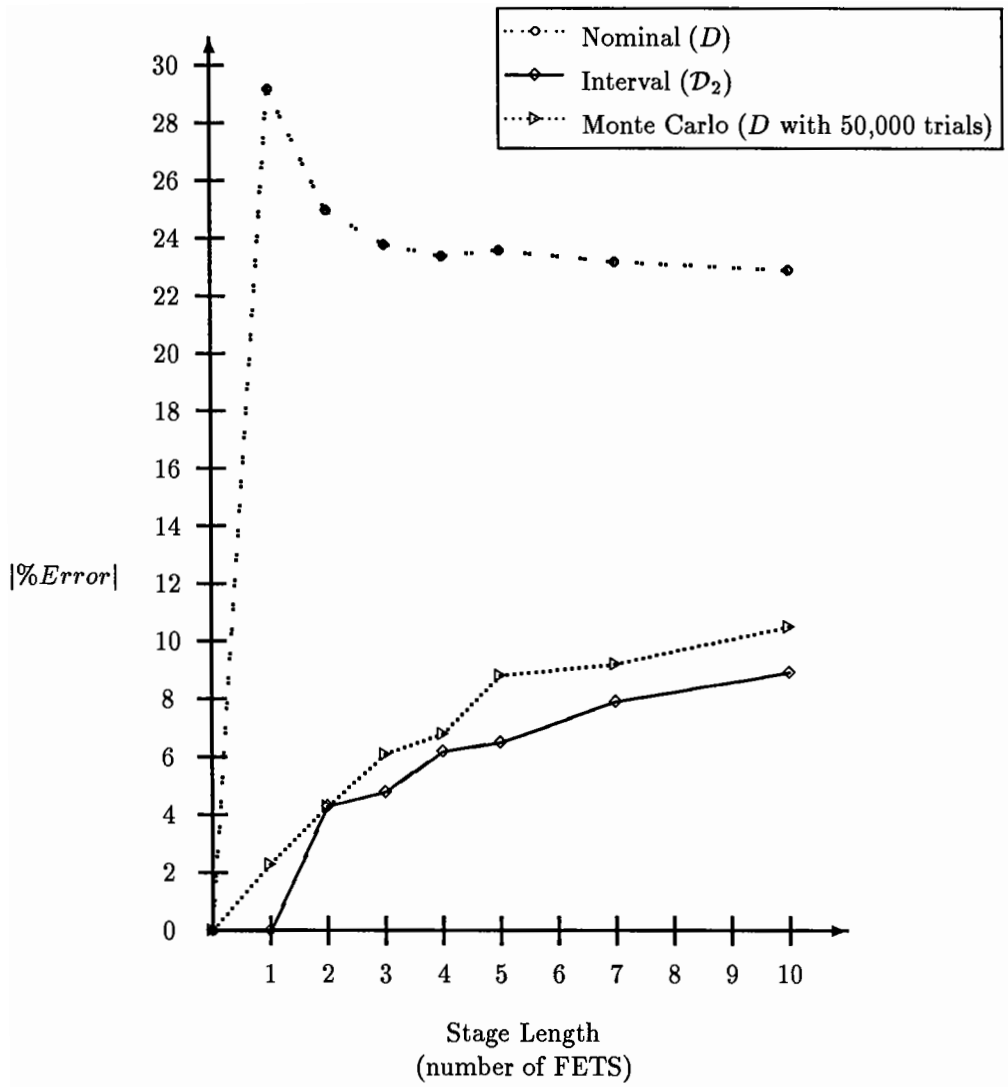

Figure 3: % Error vs. Stage Length

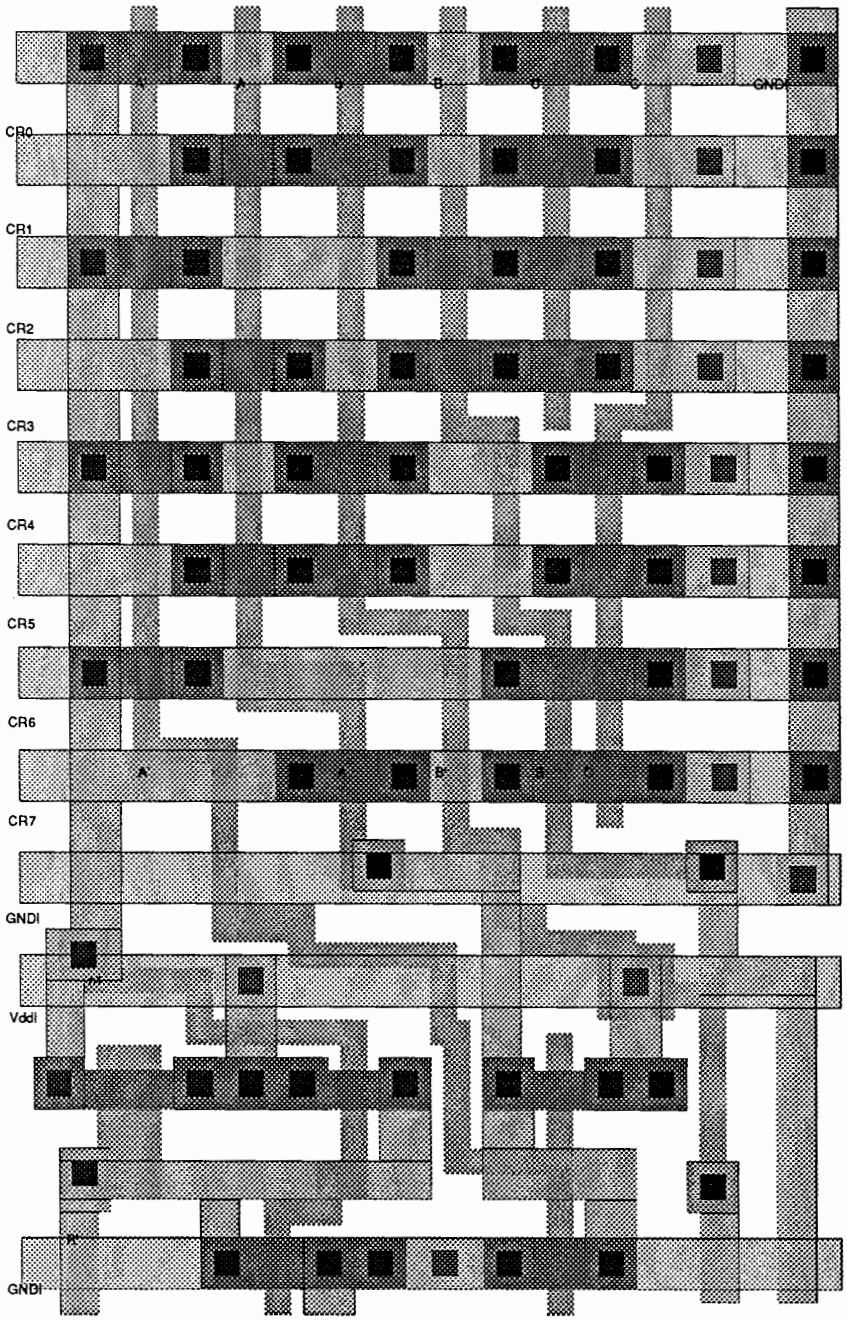

Figure 4: A portion of the B-SYS ALU

References

[Hitc82] Hitchcock Sr., Robert B. "Timing Verification and the Timing Analysis Program." *19th IEEE Design Automation Conference.* 1982, pp. 594-603.

[Hugh89] Hughey, Richard and Daniel Lopresti. "An Architecture for Programmable Systolic Arrays." Dept. of Computer Science, Brown University, Report No. CS-89-08, 1989.

[Joup83] Jouppi, N. P. "Timing Analysis for nMOS VLSI." *20th IEEE Design Automation Conference.* 1983, pp. 411-418.

[Kuli81] Kulisch, Ulrich and Willard Miranker. *Computer Arithmatic in Theory and Practice.* New York: Academic Press, 1981.

[McWi80] McWilliams, Thomas M. "Verification of Timing Constraints on Large Digital Systems." *17th IEEE Design Automation Conference.* 1980, pp. 139-147.

[Moor79] Moore, Ramon E. *Methods and Application of Interval Analysis.* Philadelphia: SIAM, 1979.

[Oust83] Ousterhout, John K. "Crystal: A Timing Analyzer for nMOS VLSI Circuits." *Third Caltech Conference on Very Large Scale Integration.* Ed. Randal Bryant. Computer Science Press, 1983, pp. 57-70.

[Oust84] Ousterhout, John K. "Switch-Level Delay Models for Digital MOS VLSI." *21st IEEE Design Automation Conference.* 1984.

[Oust86] Ousterhout, John. "Using Crystal for Timing Analysis." *1986 VLSI Tools.* Report No. UCB/CSD 86/272. University of California at Berkeley. 1986.

[Rubi83] Rubinstein, Jorge, Paul Penfield Jr. and Mark A. Horowitz. "Signal Delay in RC Tree Networks." *IEEE Transactions on Computer-Aided Design.* Vol. CAD-2, No. 3, July 1983, pp. 202-211.

[Sobo75] Sobol, I.M. *The Monte Carlo Method.* Trans. by V.I. Kisin. Mir Publishers: Moscow, 1975.

[Wall86] Wallace, David E. and Carlo H. Sequin. "Plug-in Timing Models for an Abstract Timing Verifier." *23th IEEE Design Automation Conference.* 1986, Paper 39.2, pp. 683-689.

[Zuko86] Zukowski, Charles A. *The Bounding Approach to VLSI Circuit Simulation.* Kluwer Academic Publishers: Boston, 1986.