A systematic technique for verifying critical path delays in a 300MHz Alpha CPU design using circuit simulation

Madhav P. Desai
Y.T. Yen*
Digital Equipment Corp, Hudson MA

Abstract: A static timing verifier is an important tool in the design of a complex high performance VLSI chip such as an Alpha CPU. A timing verifier uses a simple and pessimistic delay model to identify critical failing paths in the design, which then need to be fixed. However, the pessimistic delay model results in a large number of correct paths being identified as failing paths, possibly leading to wasted design resources. Therefore, each critical path identified by the timing verifier needs to be analyzed using a circuit simulator such as SPICE in order to confirm that it is a real failure. Setting up such a simulation is complex, especially when the critical path consists of structures appearing in a datapath of the CPU. In this paper, we present algorithms for the construction of a model for simulating the maximum delay through a critical path. This technique has been used to analyze several critical paths during the design of a 300MHz Alpha CPU.

1 Introduction

A static timing verifier was an important and extensively used tool in . the design of the 300 MHz Alpha 21164 CPU[1]. This timing verifier uses path tracing and simple delay modeling techniques [4] to identify critical paths in the design. These techniques inevitably introduce two types of false path errors. The first type of error is due to a path that cannot be *logically* activated (logical false path) in the design, but is erroneously reported as a failing critical path by the timing verifier. Such paths are difficult to detect automatically, especially when they cross latches (sequential false paths)[5]. The designer is required to determine that such a path cannot be logically activated, and the timing verifier is then asked to ignore this path.

The second type of error is due to the inaccuracy of the simple delay model: the timing verifier may erroneously predict that a path fails, (or more seriously, miss a real failing path). A typical approach to avoid missing a real failing path is to use a pessimistic delay model in the timing verifier, and then fix all reported critical paths.

*Presently at TeraSystems Inc., 2105 S. Bascom Ave, Suite 158, Campbell CA 95008.

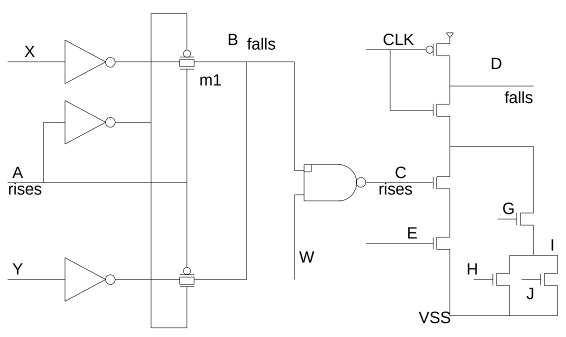

Figure 1: Fragment of a critical path

This results in the designer being forced to fix a path even when there is no need to. This waste of design resources is unacceptable in the course of a high-speed CPU design because the timing of most signal paths is extremely close to the margin, and a pessimistic delay model reports an astronomically large number of false critical paths. A better approach is to re-verify each reported critical path using a complete SPICE[2] model. This approach serves as an effective filter of false paths due to delay modeling errors, provided it can be automated. In this paper, we will describe a systematic technique to achieve this purpose.

2 Background

A simple delay model typically approximates the transistors and non-linear capacitances by linear or piecewise linear circuit elements. In addition, an approximation such as the *Elmore* delay model [10] is used to find the delay through the resultant simplified network. The combination of these approximations often results in large errors in the predicted delay, especially through complex circuit structures.

In order to verify the path timing using a circuit simulator such as SPICE[2], a circuit designer has to extract the devices needed to simulate the path correctly, and then select inputs to these devices so that a correct simulation of the worst case delay can be carried out. To illustrate this process, consider Figure 1, where we show a fragment of a critical path identified by a static timing verifier. Suppose the critical path was $A \uparrow \rightarrow B \downarrow \rightarrow C \uparrow \rightarrow D \downarrow$. In this case, it is clear that every device shown in the figure must be included in the simulation. To simulate the path correctly, the inputs must be chosen as follows: X must be set high to enable $A \uparrow \rightarrow B \downarrow$, and Y must be set low in order to initialize B correctly. The signal W must be set high to enable

$B \downarrow \rightarrow C \uparrow$. To enable $C \uparrow \rightarrow D \downarrow$, we must set E and CLK high. There is some choice in determining G, H and I. For the worst case delay simulation, we would need to pick G high and set H and I low (for typically sized devices).

The example shown above illustrates the tasks involved in simulating a critical path for maximum delay. These tasks are:

1. The devices to be included in the simulation must be identified.
2. Inputs to these devices must be chosen so that the correct sequence of transitions (node switching) is activated, and the worst case delay is simulated (path sensitization for maximum delay). The inputs to the path must be chosen to support the correct initial conditions on simulation nodes.

Consider task 2 above: This seems to be superficially similar to the problem of worst case delay estimation through logic gates [6, 7]. In order to perform a correct simulation however, we also need to consider correct initialization of the circuit, and the effect of logical relationships between circuit inputs. Such logical dependencies have a significant influence on the worst case delay through the gate. In the earlier example of Figure 1, if nodes G and E are known to be complements, then the delay $C \uparrow \rightarrow D \downarrow$ would be smaller because the capacitance at node I would not need to be discharged. In more complex structures, ignoring logical dependencies between inputs can at best lead to overestimates of the delay, and at worst cause an incorrect simulation of the structure.

In a high performance VLSI CPU design, critical paths that span dozens of nodes need to be verified using detailed circuit simulations. Such paths often pass through complex logic structures, and as is hinted at by the example above, creating the simulation deck for a path is certainly a tedious, time consuming, and non-trivial process. In this paper, our primary goal is to present a set of techniques which when used, can perform this task correctly and efficiently. These techniques form the basis of a successful tool which has been used to verify the timing of thousands of paths during the design of the Alpha 21164 [1], a full-custom 300MHz CMOS microprocessor.

3 Preliminaries

In a CMOS digital circuit, a transistor m can be viewed as a switch (whose state is determined by the gate terminal $g(m)$) which controls the movement of charge between the source terminal $s(m)$ and drain terminal $d(m)$. We will assume that at the point of analysis, each transistor has a specified orientation, and may be viewed as a unidirectional switch. There are several effective algorithmic and heuristic techniques to determine such an orientation of the transistors [3, 4] in CMOS circuits. The transistor may then be considered as a labeled directed arc from its source to its drain.

A node x is said to drive (resp. to be driven by) a device m if there is a directed path from x to $s(m)$ (resp. from $d(m)$ to x). Similarly, a node x drives a node y if there is a directed path x to y. At the point of analysis, we will assume that an orientation of the devices is chosen so that there are no directed cycles. The circuit being simulated is assumed to have a power supply node V_{DD}, and a *ground* node V_{SS}. The power and ground nodes are the strongest nodes in the circuit, that is, there is no node which drives either of these nodes.

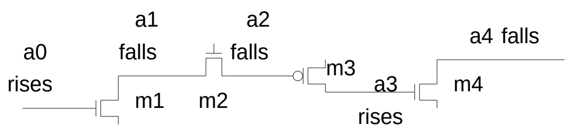

Figure 2: Critical Path

3.1 Devices needed for path simulation

The critical path to be verified is assumed to be a sequence of alternating nodes and devices, as illustrated in Figure 2. The nodes $a_0, a_1, \ldots a_n$ are termed *primary nodes*, and the devices $m_1, m_2, \ldots m_n$ are termed *primary devices*. Each primary node undergoes a *transition*, and the transition at primary node a_{j+1} is propagated to it from primary node a_j through the primary device m_{j+1}. We will assume that the path does not contain a cycle, that is, $a_i \neq a_j$ if $i \neq j$.

The first step is to identify the minimal set of devices which must be included in the simulation model for the path. We include the following sets of devices:

- All devices that can steer charge between the primary nodes on the path and the power supply nodes: the devices that drive the path are partitioned into n sets $S_1, S_2, \ldots S_n$, with S_i containing all the devices that drive primary node a_i but do not drive any primary node a_j for $j < i$. The set S_i will be termed as the *stage* corresponding to primary node i.
- All devices that contribute to the *channel loading* of the path: These devices are partitioned into n sets $T_1, T_2, \ldots T_n$. The set T_i contributes to the channel loading of node a_i in the path.
- All devices that are needed to model peripheral loading on the path (for worst case delays): We will also include devices which contribute to the *gate loading* of the path.

Two stages S_i and S_j are said to be *channel connected* if there is a node $x \notin \{V_{SS}, V_{DD}\}$ which is a channel terminal of a device $m_1 \in S_i$ and a device $m_2 \in S_j$. Consider the equivalence relation obtained by performing the transitive closure of channel connection. Each equivalence class of this closure relation is termed a *maximal channel connected region* (CCR). The path to be simulated can then be viewed as a sequence of CCR's.

4 Feasible input assignments

We are required to select logical values for the secondary inputs in the path, so that the worst case delay through the path is correctly simulated. If the logical dependencies between the secondary inputs to different CCR's are taken into account, then the problem as stated above is at least as difficult as the *dynamic path sensitization* problem, which is NP-hard [5]. We make the simplifying assumption of *ignoring* dependencies between secondary inputs of distinct CCR's. The price we pay for this simplification is a pessimistic estimate of the worst case delay through the path.

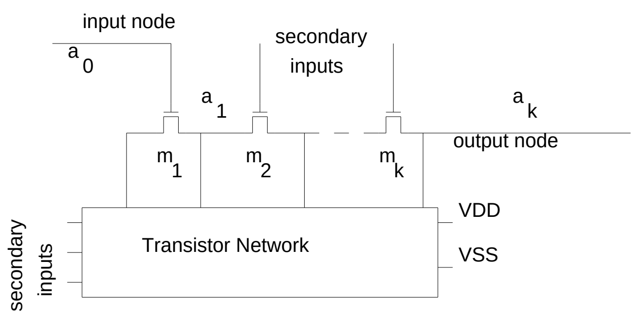

Figure 3: A Generic CCR

We have observed that in practice, the overestimation of the delay due to this decoupling is minor.

We will *not* ignore relationships between secondary inputs to a single CCR, because its behavior can be very sensitive to such logical dependencies. Some examples of CCR's with such dependencies are multiplexors, where only one out of a set of select lines can be asserted at any one time (*exclusivity*), or certain XOR gates, which have complementary inputs. The problem of sensitizing a single CCR in the presence of relationships between its inputs is also NP-hard, but the size of a typical CCR is usually quite small, and efficient heuristic algorithms for larger CCR's will be presented later in this section.

4.1 Sensitization conditions for a CCR

Consider the set of primary nodes and devices that fall inside a single CCR. The *sense* of the transition at each primary node in the interior of CCR is necessarily the same (there can be no inversions between nodes which are channel-connected). That is, each CCR may be further classified as either a *falling* CCR if all internal primary nodes undergo a falling transition or a *rising* CCR if all internal primary nodes undergo a rising transition. In the rest of this paper, we will only consider falling CCR's (rising CCR's may be handled in a similar fashion).

Such a falling CCR is shown in Figure 3. We will assume that a_0 switches from low to high at $t = 0$, causing a_1 to fall through device m_1, a_2 to fall through device m_2, and so on. In order to avoid complicating the discussion, we will only describe the case where the input node a_0 of the CCR switches at $t = 0$, and assume that the secondary input nodes are held to constant values throughout the simulation. The ideas in this paper may be easily extended to the case when multiple inputs to the CCR switch at $t = 0$.

A directed channel connected path is said to be an *ON-path* at time t if every device on the path is *ON* at time t. Otherwise, the path is an *OFF*-path. An *ON*-path connecting some node u to V_{SS} (resp. V_{DD}) is said to be a *discharging* (resp. *charging*) *ON*-path for node u. The following result characterizes the set of sensitizing input assignments for a falling CCR.

Theorem 4.1 *An assignment of logic values to the secondary inputs of a falling CCR will sensitize a correct simulation of the CCR if and only if the following restrictions are met.*

1. *At $t = 0^-$, there is* no ON-*path from any of a_1, a_2, $\ldots a_k$ to V_{SS}. This condition will be termed the* initial condition restriction.
2. *At $t = 0^+$, every* ON-*path connecting a_k to V_{SS} must contain* all *the devices $m_1, m_2, \ldots m_k$. This will be termed the* active path restriction.
3. *At $t = 0^+$, there is* no ON-*path connecting any of a_1, a_2, $\ldots a_k$ to V_{DD}. This will be termed the* passive path restriction.

4.1.1 Encoding feasible assignments

We will generate a boolean expression which encodes the set of allowable assignments for the inputs to the CCR. Represent the logical state of an input node u at time t by $\mathbf{u}(t)$. Then, $\mathbf{a_0}(0^-) = \mathbf{0}$, and $\mathbf{a_0}(0^+) = \mathbf{1}$. An n-channel device m is said to be *ON* (resp. *OFF*) at time t if $\mathbf{g}(m)(t) = \mathbf{1}$ (resp. $\mathbf{0}$). A p-channel device m is *OFF* (resp. *ON*) at time t if $\mathbf{g}(m)(t) = \mathbf{1}$ (resp. $\mathbf{0}$). The set of inputs to the CCR is

$$\mathbf{X} \;=\; (\mathbf{x_0}, \mathbf{x_1}, \ldots \mathbf{x_p})$$

Assume that $x_0 = a_0$, and $x_i = g(m_i)$ for $i = 1, 2, \ldots k$. All boolean functions described in this section are assumed to be functions of $\mathbf{X}$, and this dependence will not always be explicitly denoted.

Let m be a device in the CCR whose gate terminal is the input node x_j, where $0 \le j \le k$. Then the ON-function of m is defined to be $\mathbf{1}$ if and only if the device m is ON for input assignment $\mathbf{X}$, and is denoted by $ON_m(\mathbf{X})$. For any node u in the CCR, the *driven high* function D_u^1 of u is defined to be $\mathbf{1}$ if and only if u has a charging ON-path for input assignment $\mathbf{X}$, and is denoted by $D_u^1(\mathbf{X})$. The *driven low* function of u is defined to be $\mathbf{1}$ if and only if u has a discharging ON-path for input assignment $\mathbf{X}$, and is denoted by $D_u^0(\mathbf{X})$. For the power supply nodes, we define $D_{V_{SS}}^0 = D_{V_{DD}}^1 = \mathbf{1}$, and $D_{V_{SS}}^1 = D_{V_{DD}}^0 = \mathbf{0}$.

For every device m in the CCR, the *device driven functions* G_m^0 and G_m^1 are defined as follows.

$$G_m^0 \;=\; ON_m \wedge D_{s(m)}^0 \tag{1}$$

$$G_m^1 \;=\; ON_m \wedge D_{s(m)}^1 \tag{2}$$

These functions encode the presence of charging and discharging ON-paths through device m to $d(m)$, the drain of m.

The D and G functions are related. For $j = 0, 1$,

$$D_u^j \;=\; \bigvee_{m:w\rightarrow u} G_m^j \tag{3}$$

This relation, combined with (1) and (2) serves as the basis of a simple recursive algorithm to compute D_u^j.

It may be easily checked that the following function encodes the initial condition restriction.

$$Initial(\mathbf{X}) = \left[\bar{D}_{a_1}^0(\mathbf{X}) \wedge \bar{D}_{a_2}^0(\mathbf{X}) \ldots \wedge \bar{D}_{a_k}^0(\mathbf{X})\right]_{\mathbf{x_0}=\mathbf{0}} \tag{4}$$

Similarly, the active path restriction is encoded by the function

$$Active(\mathbf{X}) \;=\; \bigwedge_{j=1}^{k} \left(G_{m_j}^0 \wedge \left(\bigwedge_{m:u\rightarrow a_j, m\neq m_j} \bar{G}_m^0 \right) \right) \tag{5}$$

Finally, the passive path restriction is encoded by the function

$$Passive \;=\; \bigwedge_{j=1}^{k} \bar{D}_{a_j}^1 \tag{6}$$

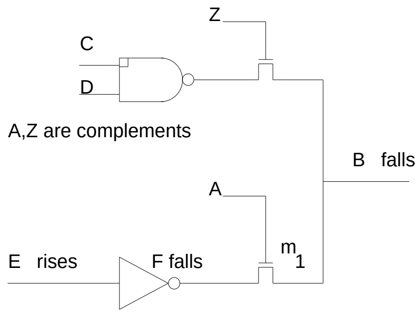

Figure 4: Example to illustrate *Constraint* derivation

Summarizing the discussion of the last few paragraphs, we obtain the following result.

Theorem 4.2 *Let Constraint define the function*

$$Constraint \;=\; Active \;\wedge\; Passive \;\wedge\; Initial \qquad (7)$$

where Active is defined in (5), Passive is defined in (6), and Initial is defined in (4). Then, an assignment of values to the CCR inputs sensitizes a correct simulation of the CCR if and only if Constraint is true.

Logical dependencies between the CCR inputs can be accounted for as follows. Suppose that these logical dependencies are captured by the condition $Logical(\mathbf{X}) = \mathbf{1}$, where *Logical* is a Boolean function. Then, we will modify the constraint function as

$$Constraint \leftarrow Constraint \;\wedge\; Logical \qquad (8)$$

For example, suppose inputs x_i, x_j are known to be complements of each other. Then we will have

$$Logical(\mathbf{X}) \;=\; \mathbf{x_i} \;\oplus\; \mathbf{x_j}$$

Any combinational relationship between the inputs may be captured in this manner.

To illustrate the derivation of the *Constraint* function, consider the CCR shown in Figure 4. The critical path is from $E \uparrow$ to $F \downarrow$ to $B \downarrow$. In this case, $Logical = A \oplus Z$, $Initial = \overline{\mathbf{D} \wedge \mathbf{C} \wedge \mathbf{Z}}$, $Active = \mathbf{A} \wedge \mathbf{E}$, and

$$Passive \;=\; \overline{(\mathbf{A} \wedge \bar{\mathbf{E}}) \;\vee\; (\mathbf{Z} \wedge \overline{\mathbf{C} \wedge \mathbf{D}})}$$

Combining these equations using (7) and (8), we obtain $Constraint \;=\; \mathbf{A} \wedge \bar{\mathbf{Z}} \wedge \mathbf{E}$. Thus, to obtain a correct simulation in this case, we need to set secondary input $\mathbf{A} = \mathbf{1}$, $\mathbf{Z} = \mathbf{0}$. We have complete freedom in selecting inputs $\mathbf{C}, \mathbf{D}$.

Remark: The *Constraint* function is generated from the D and G functions, which themselves are computed for each node and device in the CCR by using a recursive algorithm on the basis of Equations (3), (2) and (1). The manipulation of boolean functions is performed using a BDD-package [9].

5 Input Selection for a worst case delay simulation

We need to find a feasible input assignment which satisfies the constraint equation (7), and in addition, causes the maximum delay in the CCR. For a CCR with a small number of inputs, the following enumerative algorithm can be used: For each feasible input assignment, simulate the CCR and find the delay through it. Keep track of the input assignment which caused the worst case delay, and use this in the final simulation of the path. This naive algorithm will yield the desired input assignment, but is practical only for a CCR with a small number of inputs and a small number of devices. A slightly more efficient algorithm can be obtained by using a delay estimate [10] (instead of an actual simulation of the CCR) to grade each candidate input assignment. An input assignment chosen using such an exhaustive enumeration algorithm has the advantage that the delay obtained using this assignment will be close to the actual worst case delay.

For a CCR with a large number of inputs, we will use the following heuristic which is often used in worst case delay estimation [8, 6]: To find the worst-case delay through a gate, find the most resistive path which can discharge the output node of the gate. Then maximize the loading on this path. In our context, this heuristic can be modified as:

1. For the output node of the CCR, find the *most resistive discharging path* (MRDP) which can be turned ON without violating *Constraint* (find a feasible MRDP).
2. Maximize the capacitive loading on the MRDP selected in step (1) above.

The MRDP heuristic gives priority to finding the most resistive path for the discharge of the output node capacitor, and is especially effective when the output node capacitance dominates the internal node capacitances in the CCR.

5.1 An Algorithm to find a feasible MRDP

Using Dijkstra's algorithm [11], we can compute an unconstrained MRDP for any node u efficiently (because the CCR is acyclic). Denote the resistance of this MRDP by $\mathbf{R_{max}}(u)$. Suppose we want to find a constraint feasible MRDP for node u. Let $\mathbf{f}$ denote the constraint function at this point. We will try to enumerate all possible $\mathbf{f}$-feasible discharge paths for u, using $\mathbf{R_{max}}$ values to bound this search.

The enumeration of these paths proceeds as follows. We try to extend discharge paths from u such that the constraint function is not violated. Denote by $\mathbf{R_{best}}$ the value of the most resistive feasible discharge path found so far in the algorithm. Initially, $\mathbf{R_{best}} = 0$. Suppose w is immediately upstream of u (that is, there is a device $m : w \to u$ in the the CCR). Let x denote the gate of device m, and assume without loss of generality that m is an n-channel device. Suppose the resistance of device m is $\mathbf{R_m}$.

Then, m may be included in a constraint feasible discharge path for u if and only if setting m ON does not violate the constraint $\mathbf{f}$. That is, if $\mathbf{f}\,|_{\mathbf{x}=\mathbf{1}} \neq \mathbf{0}$. Suppose this condition is satisfied. Then, it is logically possible to extend the path from w. We use the bound to prune the search here. If

$$\mathbf{R_{best}} \;>\; \mathbf{R_m} \;+\; \mathbf{R_{max}}(w)$$

then any extension from w will not be part of a feasible MRDP, and there is no need to continue extending the path beyond w.

In the algorithm, we carry this argument a step further. Anytime we visit a node w, we carry the resistance of the current path that is being extended. This parameter will be termed $\mathbf{R_{sofar}}$. In the paragraph above, $\mathbf{R_{sofar}} = \mathbf{R_m}$

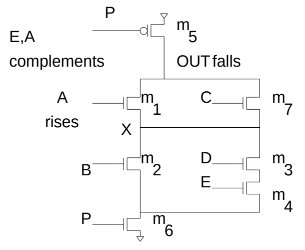

Figure 5: Example of MRDP calculation

when we visited w. We will try to extend the path from w only if

$$\mathbf{R_{best}} < \mathbf{R_{sofar}} + \mathbf{Rmax}(w)$$

If the already computed unconstrained MRDP for w is feasible, we report it as the best continuation from w. Otherwise, the search is continued from w. Below, we present the details of the algorithm. We assume that $\mathbf{Rmax}(u)$ has already been computed for each node in the CCR. We will also assume that for each node u, the unconstrained MRDP is available to be checked by the algorithm.

```
Procedure FeasibleMRDP($u$, $\mathbf{f}$)
    $\mathbf{R_{sofar}} \leftarrow 0.0$
    $\mathbf{R_{best}} \leftarrow 0.0$
    LongPath($u, \mathbf{f}, \mathbf{R_{sofar}}, \mathbf{R_{best}}, \mathbf{R_{ret}}$)
    return($\mathbf{R_{ret}}$)

Procedure LongPath($u, \mathbf{f}, \mathbf{R_{sofar}}, \mathbf{R_{best}}, \mathbf{R_{ret}}$)
    if ($\mathbf{R_{sofar}} + \mathbf{Rmax}(u) < \mathbf{R_{best}}$) return(FALSE)
    $\mathbf{R_{ret}} \leftarrow \mathbf{0.0}$
    if (unconstrained MRDP for $u$ does not violate $\mathbf{f}$)
      $\mathbf{R_{ret}} = \mathbf{Rmax}(u)$;
      $\mathbf{R_{best}} = \mathbf{Max}(\mathbf{R_{best}}, \mathbf{NextR_{sofar}} + \mathbf{NextR_{ret}})$
      return(TRUE);
    foreach $m : w \rightarrow u$
      $x \leftarrow g(m)$
      $\mathbf{g} = \mathbf{f}\,|_{\mathbf{x=1}}$ /* assume m is an n-device */
      if ($\mathbf{g} \neq \mathbf{0}$)
        $\mathbf{NextR_{sofar}} \leftarrow \mathbf{R_{sofar}} + \mathbf{Rm}$
        if (LongPath($w, \mathbf{g}, \mathbf{NextR_{sofar}}, \mathbf{R_{best}}, \mathbf{NextR_{ret}}$))
          $\mathbf{R_{ret}} = \mathbf{Max}(\mathbf{R_{ret}}, \mathbf{NextR_{ret}} + \mathbf{Rm})$
        else return(FALSE)
        $\mathbf{R_{best}} = \mathbf{Max}(\mathbf{R_{best}}, \mathbf{NextR_{sofar}} + \mathbf{NextR_{ret}})$
    return(TRUE)
```

An example will be helpful here. Consider the circuit shown in Figure 5. The constraint function for the entire CCR is

$$\mathbf{P} \wedge \mathbf{A} \wedge \bar{\mathbf{C}} \wedge \ (\mathbf{B} \vee \ \mathbf{D} \wedge \mathbf{E}) \ \wedge \ (\mathbf{A} \oplus \mathbf{E})$$

The feasible MRDP until the node X must include the primary device m_1. The constraint function for continuing the feasible path beyond X is obtained by making the substitution $\mathbf{A} = \mathbf{1}$ in the equation above to yield the modified constraint equation

$$\mathbf{P} \wedge \mathbf{B} \wedge \bar{\mathbf{C}} \wedge \bar{\mathbf{E}}$$

The algorithm will be unable to extend the discharge path through devices m_3, m_4 because the constraint above does not allow us to set $\mathbf{E} = \mathbf{1}$. Thus it will find the feasible MRDP to be $m_1\ m_2\ m_6$, which corresponds to the input selection $\mathbf{A} = \mathbf{1}$, $\mathbf{B} = \mathbf{1}$, $\mathbf{P} = \mathbf{1}$. The residual constraint function at this point is $\bar{\mathbf{C}} \wedge \bar{\mathbf{E}}$. This implies that both C and E must be set low. The input D is not determined at this point, and should be selected to maximize the loading, and hence the delay through the CCR.

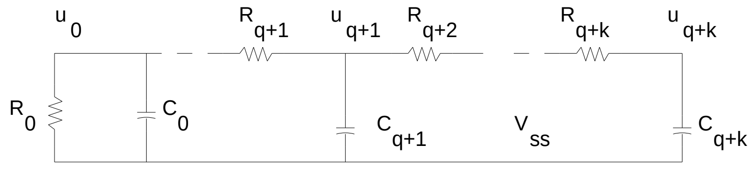

Figure 6: RC approximation of CCR

5.2 Maximizing capacitive loading on the MRDP

The inputs which are not set in the feasible MRDP must be selected so that they maximize the loading on the MRDP. More precisely, consider an RC approximation of the CCR, with the feasible MRDP being modeled as in Figure 6. The number C_i represents the total capacitance seen by node u_i. The node u_{q+i} is the same as the primary node a_i for $i = 1, 2, \ldots k$. The first order *Elmore* estimate [10] of the delay to node u_{q+k} in Figure 6 is given by

$$\tau = \sum_{i=0}^{q+k} C_i \left(\sum_{j=0}^{i} \mathbf{R_j} \right) \tag{9}$$

In other words, we should select the remaining inputs (which control the capacitances seen by the nodes u_i) so that the weighted sum in (9) is maximized. The constraint function used to guide this selection is obtained from (7) by setting the feasible MRDP devices ON.

We use a greedy heuristic algorithm which attempts to maximize (9) by causing the charge from capacitances in the CCR to be injected into the feasible MRDP as close to the output node a_k as possible. Starting at the output node a_k, we attempt to connect as much capacitance to node a_k as possible without violating the constraint equation. This is achieved by successively removing maximum capacitance paths from an acyclic directed graph. Then, we proceed to node a_{k-1} and repeat this process, until all primary nodes have been visited.

We describe the algorithm below. For a node u in the CCR, we define a residual network $N_u = (V_u, A_u)$ by including in V_u every node which is in the CCR, but not connected to the MRDP by an ON-path. The set A_u will contain only the undetermined devices (whose gate states have not been determined) whose channel terminals are in V_u. A path in N_u is $\mathbf{f}$-feasible if setting the corresponding path in the CCR ON does not violate the constraint $\mathbf{f}$. The greedy algorithm proceeds by stripping off the maximum capacitance $\mathbf{f}$-feasible directed path with one end at u in the residual network. The algorithm used to find such a path is a simple modification of the procedure *LongPath* described earlier. After an $\mathbf{f}$-feasible path has been stripped off, we modify $\mathbf{f}$ by setting the input variables labeling the arcs in the path,

Path	length	crude delay	spice delay
eb1	17	3.12ns	1.85ns
eb2	14	2.65ns	1.34ns
ib1	14	2.70ns	1.74ns
ib2	23	3.66ns	2.21ns
ib3	11	2.56ns	2.02ns

Figure 7: Comparison of path delays

and we modify the residual network N_u by removing all the determined arcs in it.

```
Procedure WorstCaseLoading($u$, $\mathbf{f}$)
    Find residual network $N_u = (V_u, A_u)$ for $u$
    while(undetermined arc incident on $u$ exists)
        Find max. cap. feasible path for $u$ in $N_u$
        Update $\mathbf{f}$ and $N_u$
    foreach $w \to u$
      WorstCaseLoading(w,$\mathbf{f}$)
```

If we refer to the example in Figure 5, we can verify that the algorithm above chooses $\mathbf{D} = \mathbf{1}$ to maximize the delay through the CCR.

Before we conclude this section, we will briefly touch upon the setting of initial conditions in the CCR. For every node u in the CCR, we check if there is some discharging ON-path for u at time $t = 0^-$. If not, the initial condition at this node should be set to a high value. Otherwise, the initial condition at u must be set to a low value. This will ensure that the largest possible amount of charge is present in the CCR at $t = 0^-$. Note that the actual physical voltage to which the initial condition at a node is set is either equal to, or a threshold voltage drop off a power supply level.

6 Results

All algorithms described in this paper have been implemented in a CAD tool which works in conjunction with a static timing verifier. The tool has been successfully used during the design of the Alpha 21164, which is a 300MHz high performance microprocessor with nearly 10 million devices[1]. As part of the design process, it was decided to reverify each critical path in the design using this CAD tool, and thousands of signal paths were successfully simulated.

In Figure 7, we present a comparison of path delays obtained using a crude delay model and by circuit simulation. The paths eb1, eb2 are critical in a 64bit adder, and ib1, ib2, ib3 are critical paths in the instruction datapath. The results illustrate the need for path delay verification by circuit simulation. The crude delay model can be too pessimistic when compared to the SPICE simulation, and often overestimates the path delays by more than 50%.

In all examples that we have encountered, the time needed to create a path model was much smaller than the time needed to run SPICE on the model. For example, in case of the path eb1 in Figure 7, the (wall clock) time needed to prepare a path model (by collecting devices, setting inputs and initial conditions) was less than 1 second, whereas the time needed to run SPICE on the path model was 187 seconds (measured on a DEC 3000-700 workstation).

7 Conclusions

We have demonstrated a systematic technique for automatic verification of critical path delays using circuit simulation. We have developed an efficient mechanism for encoding the feasible input assignments for simulating a CCR, and have presented algorithms to select the feasible input assignment for the worst case delays. A simple enumerative algorithm will quickly find such an input assignment for a small CCR, but an efficient heuristic is needed in general. We have demonstrated an efficient implementation of such a heuristic, namely the feasible most resistive discharge path (MRDP) algorithm.

Acknowledgement: The authors would like to thank Bill Grundmann, Bill Wheeler, Nick Rethman and Emily Shriver for their helpful suggestions.

References

[1] J. Edmondson et. al., “Internal Organization of the Alpha 21164, a 300-MHz, 64-bit, Quad-Issue, CMOS RISC Microprocessor,” *Digital Technical Journal,* vol. 7, no. 1, 1995.

[2] L.W. Nagel, “SPICE2: A Computer Program to Simulate Semiconductor Circuits,” Electronics Research Laboratory *Rep. No. ERL-520, University of California, Berkeley,* May 1975.

[3] J. Ousterhout, “A Switch-Level Timing Verifier for Digital MOS VLSI,” *IEEE Transactions on Computer-Aided Design,* vol. CAD-4, no. 3, pp. 336-349, July 1985.

[4] J.J. Grodstein, J. Pan, W. Grundman, B. Gieseke, and Y.T. Yen, “Constraint Identification for Timing Verification,” *Proceedings of International Conference on Computer-Aided Design*, pp. 16-19, November 1990.

[5] P.C. McGeer and R.K. Brayton, *Integrating Functional and Temporal Domains in Logic Design: The False Path Problem and its Implications*, Boston, MA: Kluwer Academic Publishers, 1991

[6] S. S. Sapatnekar and S. M. Kang, *Design Automation for Timing-Driven Layout Synthesis,* Boston, MA: Kluwer Academic Publishers, 1993.

[7] M.R. Dagenais, S. Gaiotti and N. C. Rumin, “Transistor-Level Estimation of Worst-Case Delays in MOS VLSI Circuits,” *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 384-394, March 1992.

[8] L. A. Glasser and D.W. Dobberpuhl, *The Design and Analysis of VLSI Circuits,* Reading, MA: Addison-Wesley Publishing Company, 1985.

[9] K.S. Brace, R.L. Rudell, and R.E. Bryant, “Efficient Implementation of a BDD Package,” in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 40-45, July 1990.

[10] T-M Lin and C.A. Mead, “Signal Delay in General RC Networks,” *IEEE Transactions on Computer-Aided Design,* vol. CAD-3, pp 331-349, October 1984.

[11] S. Even, *Graph Algorithms*, Rockville, MD: Computer Science Press, 1979.