# Crystal: A Timing Analyzer for nMOS VLSI Circuits

John K. Ousterhout
Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

**Abstract**

Crystal is a timing analyzer for nMOS circuits designed in the Mead-Conway style. Based on the circuit extracted from a mask set, Crystal determines the length of each clock phase and pinpoints the longest paths. The analysis is independent of specific data values and uses critical path techniques along with a simple RC model of delays. Several additional techniques are used to improve the speed and accuracy of the program, including separate up/down timing, static level assignment, flow control for pass transistors, and precharging.

This paper will be presented at the 3rd Caltech Conference on VLSI, March 21-23, 1983.

## 1. Introduction

As the density of integrated circuits increases into the hundreds of thousands of transistors, the intuitions of circuit designers become more and more fallible. The large number of components and their complex interrelationships make it almost impossible for designers to foresee all the consequences of each design decision. To decrease the likelihood of costly errors, designers depend on assistance from computer tools. Programs ranging from layout rule checkers to simulators are used to validate designers' intuitions and root out errors before they are cast in silicon.

This paper is concerned with one particular aspect of design validation: performance. Even if a circuit design is logically correct and free of layout errors, it will be of little or no use unless it can operate at an acceptable speed. For small designs, circuit simulators such as SPICE [6] can be used to verify performance. For large circuits, however, designers are left largely on their own, since circuit simulators become intolerably slow for even a few thousand transistors. Designers can extract pieces of a large design and simulate the pieces, but this depends on the designers' ability to select the right pieces. Experience with VLSI chips at U.C. Berkeley suggests that this is an error-prone approach [2,8]. The initial speed limitations were due not to major elements like ALU carry chains, but to seemingly trivial logic where small transistors were accidentally used to drive large loads. In one case [2] the errors were not discovered until after fabrication; in the other case [8] several such errors were discovered before fabrication by an early version of Crystal.

Crystal is a data-independent timing analysis program. Its input is a description of an nMOS circuit. The description is extracted from the mask layout and includes interconnect resistance and capacitance as well as transistor sizes and types. Crystal determines the length of each clock phase and identifies the slowest paths. The analysis is fast: one example circuit of 45000 transistors can be processed in about 20 minutes of CPU time on a VAX-11/780. However, since Crystal uses a very simple model of timing, it produces less accurate results than circuit simulators. Our goal is to achieve overall timing estimates within ±20% of the simulation results of SPICE.

Section 2 introduces the basic mechanism used by Crystal. In its simplest form, this approach produces grossly pessimistic timing estimates. Furthermore, the basic approach suffers from severe computational inefficiencies that make it impractical for any real circuits. Section 3 discusses these problems and describes several additional mechanisms used in Crystal to increase the efficiency of the program and the accuracy of its results. Section 4 presents the limits of the timing model, and Section 5 describes our experiences using the program.

## 2. The Basic Mechanism

The notion of timing analysis has existed for some time, and several timing analyzers have been described in the literature [1,4,5]. A timing analyzer can be thought of as a program that simulates a single clock cycle with all possible combinations of data values at the same time. In this way it computes worst case delays and verifies that the timing requirements of the circuit's various memory elements will be satisfied.

Most existing analyzers were designed for bipolar technologies such as ECL and TTL. MOS circuits, particularly those designed in the Mead-Conway style, differ in several important ways from bipolar circuits, and thus suggest a different approach to timing analysis. Whereas bipolar circuits tend to use complicated asynchronous clocking schemes, MOS circuits in the Mead-Conway style use simple, synchronous clocking mechanisms based on non-overlapping clock phases. Thus Crystal does not deal with minimum delay times or set-up and hold-times, which account for much of the complexity of bipolar timing anaylzers. Crystal's approach is simpler: for each clock phase, it merely determines how long it takes after the clock changes for all the effects of that change to propagate throughout the circuit. This results in a worst-case estimate for the length of the clock phase.

Crystal also differs from bipolar timing analyzers in its computation of the basic delays. In bipolar technologies the delay of a piece of logic is almost independent of the way the logic is used, and wire delays can be combined with logic delays in a simple fashion. The delay for each logic type is provided by the user. In MOS, the delay calculation is a complex function of the logic type, its geometry, and the way it is used in the circuit. The delay for one piece of logic may even be impacted by the logic that provides its inputs. This makes the basic delay calculations much more complex for MOS analyzers than for bipolar analyzers.

To compute how long it takes for a clock change to propagate, Crystal makes a recursive, depth-first pass over the circuit starting with the clock. At any given time, Crystal considers a change in value at a particular transistor gate, called the *driver* (see Figure 1). When the driver changes value, it may cause certain other transistor gates, called *targets*, to change value at later times. Crystal finds each target of the driver and computes the delay between driver and target. This information is then propagated recursively

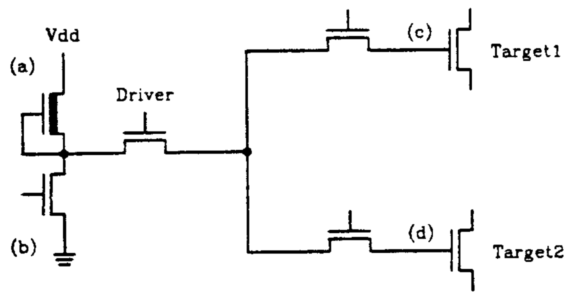

**Figure 1.** When it is discovered that the driver changes value at a particular time, Crystal firsts looks on one side of the driving transistor to find possible paths to Vdd and Ground at (a) and (b). Then it searches on the other side of the driving transistor for possible paths to other gates at (c) and (d). Delays are computed for each separate path between a target gate and Vdd or Ground (4 paths in this case).

by treating the target as driver and figuring out which other gates it can affect. If a target is reached from several different drivers, then it isn't necessary to perform the recursive step unless the target's new delay time is the latest one seen for it.

Crystal finds the targets of a particular driver in two steps, as illustrated in Figure 1. First, starting at the source terminal of the driving transistor, Crystal searches through sources and drains of other transistors to find all paths from the driver source to Vdd or Ground. Crystal assumes that all transistors could conceivably be on. For each such path, it then searches from the drain of the driving transistor through sources and drains to all possible gates. Each of these gates is considered a target, and the path from the target through the driver to Vdd or Ground is used to compute the delay from driver to target.

To compute the delay, Crystal approximates each transistor with a resistance value. The resistance values were chosen based on SPICE simulations

| Transistor Type | Ohms/square (pulling to 1) | Ohms/square (pulling to 0) |
|---|---|---|
| Enhancement | 30000 | 15000 |
| Depletion Load | 22000 | – |
| Super-Buffer (depletion, gate 1) | 5500 | 5500 |
| Depletion (gate 0 or unknown) | 50000 | 7000 |

**Table 1**. Resistance values used for transistors.

with expected processing parameters (see Table 1). Different resistance values are used depending on whether the target is being pulled to Vdd or Ground, and depending on whether the gate of the transistor is known to have a particular value (see Section 3). All the resistances and capacitances of transistors and nodes are summed along the path from target to Vdd or Ground, and the RC product is used as the delay from driver to target (load devices require special processing; see Section 3.1 for details). Only the resistance and capacitance directly along the path from target to supply rail is considered. If there are side paths separated from the main path by pass transistors, the capacitance from those side paths is ignored.

For each path from the driver source to Vdd or Ground, all possible targets are found on the drain side of the driver and delays are calculated. Then the drain side of the driver is examined for paths to Vdd or Ground, and for each of these the source side is examined for targets.

The algorithm described above is just a depth-first critical path analysis. A depth-first search is used rather than a breadth-first one because circuits such as static memory elements contain feedback loops; by marking the pending nodes during the search, infinite loops can be avoided. In a depth-first search, the processing time could in the worst possible case be exponential in the size of the circuit, whereas breadth-first search is linear.

Fortunately, the graphs describing integrated circuits tend to be shallow (only a few gate delays per clock period), and the loops tend to be short (usually just two gates). This results in running times that are almost linear in the size of the circuit.

Note that this analysis does not consider any specific data values at various nodes: all possible paths are assumed to be valid, and it is assumed that all transistors could conceivably be turned on. A data-independent analysis is more powerful than a simulation based on particular data values because it is certain to find the worst-case time; simulations will be effective only to the extent that the test cases are complete.

## 3. Improvements in Speed and Accuracy

In actual chips, not all transistors will always be turned on, and not all data values will be possible. A completely data-independent analysis, like that of Section 2, will chase many paths that can never be exercised in the real circuit, so it is likely to produce ridiculously pessimistic time estimates. Furthermore, the time required to chase all the paths will make the program too slow to be practical. Thus, although Crystal uses the simple mechanism as the core of its analyzer, it also employs several additional techniques that incorporate a few data dependencies in order to make its estimates more exact and its running times more reasonable.

### 3.1. Up/Down Times and Loads

In a purely data-independent analysis, Crystal would have to use the worst-case time for each driver-target delay. This will almost always be the low-to-high transition, since nMOS rise times are usually several times longer than fall times. However, real circuits consist almost exclusively of inverting

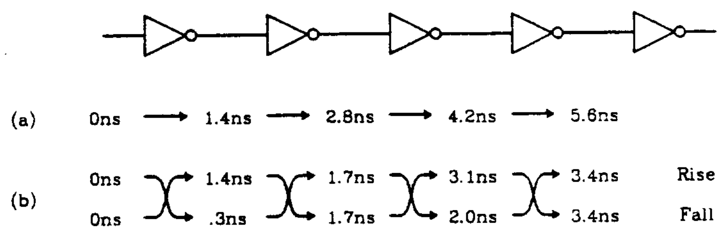
(a)
0ns → 1.4ns → 2.8ns → 4.2ns → 5.6ns
(b)
0ns 1.4ns 1.7ns 3.1ns 3.4ns Rise
0ns .3ns 1.7ns 2.0ns 3.4ns Fall

**Figure 2.** Crystal keeps separate rise and fall delay times for each node. If only the worst case times were used (the rising delay), then the pessimistic delay estimates in (a) would result. Instead, Crystal recognizes the inverters and combines rise delays for one stage with fall delays for the next, as shown in (b).

logic. For example, in Figure 2 it is unnecessarily pessimistic to assume that each of the several inverter stages is rising. Instead, Crystal keeps separate times for high-to-low and low-to-high transitions at each node and computes delays to reflect the level inversions that occur. Figure 2 shows how delay estimates are reduced by keeping separate up- and down-times.

There are three distinct cases that can occur in nMOS: pulldown, enhancement pullup, and depletion pullup. These are illustrated in Figure 3. Figure 3(a) shows the pulldown case. In this case a level inversion occurs: when the driver transistor turns on it pulls the target to ground. Only the high-to-low time of the target is affected, and it is determined by adding the low-to-high time of the driver to the delay through the path.

The enhancement pullup case is illustrated in Figure 3(b). No level inversion occurs here: when the pass transistor turns on, it pulls the target to a high voltage. For this path, only the low-to-high time of the target is affected. It is determined by adding the low-to-high time of the driver to the delay through the path. Because the target is being pulled high, different resistances are used for the transistors along the path than in the pulldown

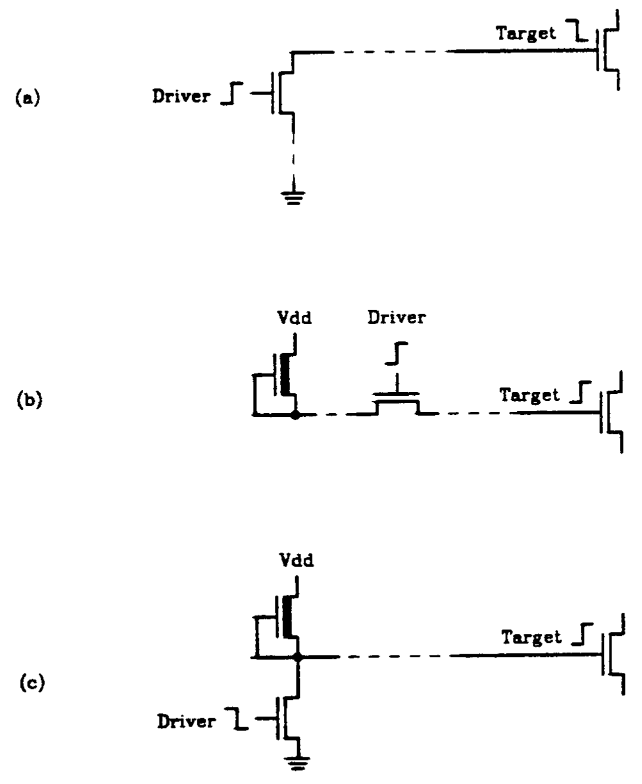

**Figure 3**. The three different ways that targets are driven in nMOS: a) a pull-down transistor turns on, causing the target to be pulled to 0; b) a pass transistor turns on, causing the target to be pulled to 1; c) a pulldown turns off, causing the target to be pulled to 1 through a depletion load.

case. In the case of Figure 3(b), it is likely that there is also a path through the pass transistor to ground; this path will be analyzed separately using the pulldown rules to compute a new high-to-low time for the target.

The third case, depletion pullup, is illustrated in Figure 3(c). After the driver transistor turns off, the load will pull the target to a high level. When chasing out the paths, Crystal watches for load devices along the path, and remembers the closest load to each target. The low-to-high time for the target is computed by adding the high-to-low time for the driver to the delay to Vdd through the load. Only the closest load to the target is considered.

Furthermore, if a depletion load is seen along a particular path, then enhancement pullup times are ignored for the path: Crystal assumes that the depletion load will be responsible for all low-to-high transitions at the target.

## 3.2. Fixed Values

In real circuits, certain nodes will have fixed values during each clock phase. These fixed values will prevent some delay paths from occurring when the chip runs. The most obvious examples are the clock signals themselves, as illustrated in the dynamic shift register example of Figure 4. Without any knowledge that Phase2 is zero when Phase1 is on, Crystal will attempt to propagate the input signal through all the shift register stages during Phase1, and will thus produce a very long worst-case time for Phase1.

Before invoking the delay analysis for a particular clock phase, users can indicate that certain signals (most notably the other clock phases) have fixed values. Crystal performs a static logic simulation to fix as many other node values as possible. If one input of a NAND structure is fixed at 0, Crystal will infer that the output must be fixed at 1. If one input of a NOR structure is fixed at 1, Crystal will infer that the output must be fixed at 0, and so on.

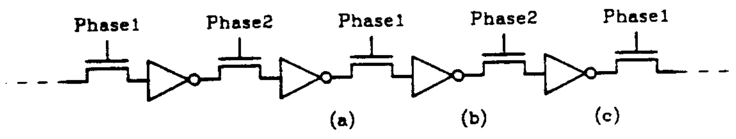

**Figure 4.** In analyzing the delays for Phase1 it is important to realize that Phase2 is zero. Otherwise, delays will be computed under the assumption that data could flow from end to end in a single clock phase. For the Phase1 analysis, the user indicates that Phase2 is fixed at 0; Crystal will evaluate the path from (a) to (b) but not from (b) to (c).

When searching for delay paths, Crystal assumes that there is a zero delay to all nodes with fixed values. It also refuses to propagate delays through enhancement transistors whose gates are fixed at zero. In the case of Figure 4, this means that only a few short paths will be considered during Phase1, and only a (different) few short paths during Phase2.

## 3.3. Flow Control

Pass transistors cause special problems, especially when the pass transistors are bi-directional. An example is in Figure 5. Without any extra information, Crystal will try a path where Input1 drives Output2 by passing forwards and backwards through the multiplexor structure. In practice, only one of the mux pass transistors will be turned on at a time, so the path is impossible. A more severe problem arises for barrel shifters and other

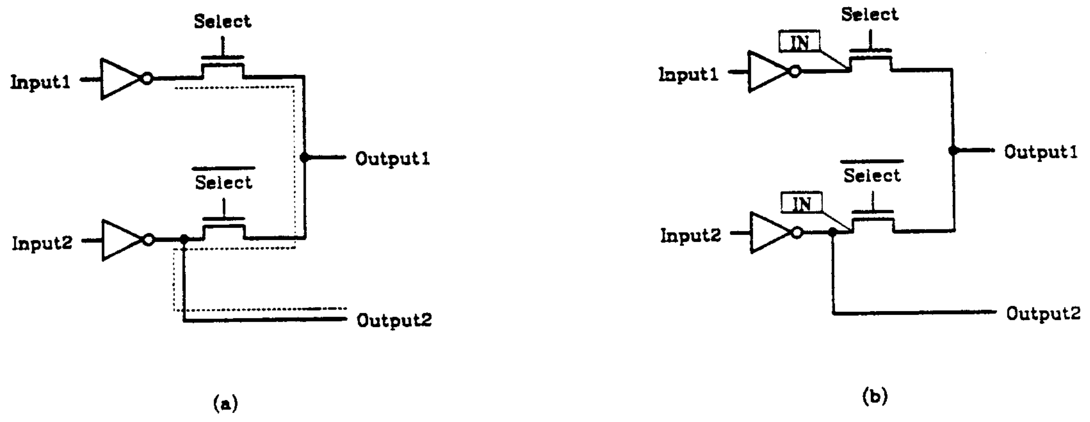

**Figure 5.** Without any information about how information flows through pass transistors, Crystal will examine the dotted path in (a), which can never occur in practice. If the user indicates that 0/1 signals always flow into the two mux pass transistors at their left sides, as in (b), then Crystal will not examine the bogus path.

structures where arrays of pass transistors are used to re-arrange data, as in Figure 6. In searching for all possible paths between a driver on the left and a target at the bottom, Crystal will analyze tortuous long paths through the pass transistor array. This has two consequences. First, a large number of pass transistors will be examined in series, leading to unrealistic long delay estimates (in reality only a few pass transistors are enabled at any one time). Second, the number of possible paths through such a structure grows exponentially with its size. For even a 16-bit barrel shifter the analysis takes too long to be practical.

One approach to the problem is to use fixed values to limit the possible paths. For example, in the case of the 2-input mux, 2 separate timing analyses could be performed, one with *Select* fixed to 0 and $\overline{\textit{Select}}$ fixed to 1, and one with the values reversed. This is the approach taken by the SCALD system. However, this approach has two disadvantages. First, it is tedious and expensive to make separate analyses with different fixed values, espe-

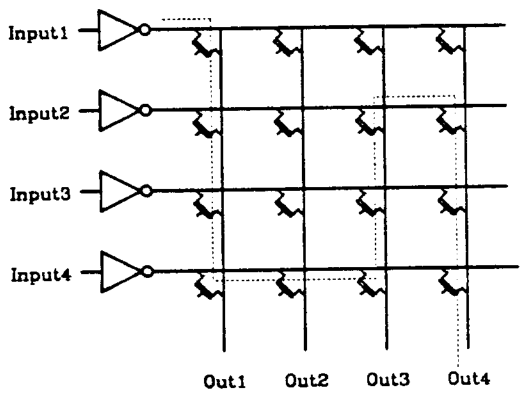

**Figure 6**. Another example where bogus paths, like the dotted one, will be examined unless information is provided about how the structure is used.

cially for more complex structures where there are many valid combinations of control lines. The second problem is that no delays are calculated through fixed nodes. Thus, if the critical path involves the $\overline{Select}$ signal, Crystal will never detect that fact.

Crystal's solution to the problem is to allow designers to indicate the direction of signal flow through transistors. This feature is used in two ways. For the mux case, where information always flows in one direction, designers indicate which side of each pass transistor is the source of the zero or one signal, as shown in Figure 5(b). The indication is made directly in the mask layout and passed through to Crystal by the circuit extractor. Crystal will then ignore any paths through the pass transistor with the zero or one source on the wrong side.

A similar but more powerful technique is used for bidirectional structures, such as the one in Figure 7. For bidirectional structures, designers tag one side of each transistor in the structure with a particular name other than "In" or "Out". When a potential path contains transistors tagged this way, Crystal will consider the path only if information flow in the path is always unidirectional with respect to the tags. This permits separate paths

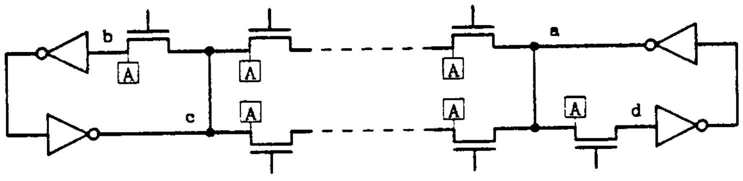

**Figure 7**. To handle bidirectional structures, the designer indicates a direction of flow by tagging one side of each transistor in the structure. Crystal permits paths in either direction, e.g. from *a* to *b* or from *c* to *d* or from *c* to *b*, but not paths that pass back and forth with respect to the tags, as from *a* to *c* to *d*.

passing in opposite directions across the structure, but ignores paths that go back and forth. Each different structure in the circuit uses a separate tag for its pass transistors, so Crystal handles them independently.

Although it might seem that this kind of flow control would require the designer to spend a large amount of time tagging his design, in practice the work for this is small. In the RISC II Cache [8], which has several large bidirectional structures, approximately 12000 transistors have tags out of 46000 total transistors. But because the design uses arrays extensively, only 103 individual tags had to be entered by the designers, compared to about 900 other distinct node labels. If all pass transistors are unidirectional, it may be possible for the timing analyzer to infer the directionality and eliminate flow tagging. This is the approach taken by the TV program [3].

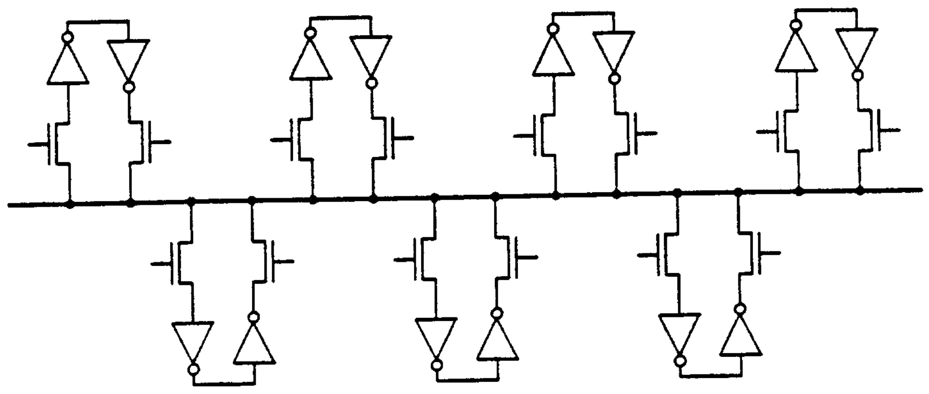

**Figure 8.** To avoid examining separate paths from each memory cell onto the bus then out to each other memory cell, Crystal computes separate delays from cells to the bus and from the bus to cells.

### 3.4. Busses and Precharging

Consider the structure of Figure 8, where several memory cells connect to a single bus and each cell can be written from or read out onto the bus. When analyzing this circuit, Crystal will consider paths from each memory cell out onto the bus and into each other memory cell. Thus, for N cells on the bus, Crystal will consider $N^2$ paths. For large memory arrays this is both expensive and unnecessary.

As long as the capacitance of the bus is much greater than the internal capacitance of any of the memory cells, the delay from one cell to another can be approximated by two separate delays: one from a cell onto the bus, and a second "independent" delay from the bus to each other cell. Thus, instead of $N^2$ paths, Crystal need only examine 2N paths: N paths from cells onto the bus, and N paths from the bus back into individual cells. There are two ways that Crystal finds out about busses. First, users may indicate this explicitly. Second, there is a user-settable capacitance threshold above which Crystal automatically considers a node to be a bus. The default threshold is 1pf.

Crystal also allows users to specify that certain nodes are precharged before certain clock phases. When this happens, Crystal ignores all low-to-high transitions for the precharged nodes.

### 3.5. Cross-Phase Signals

It is quite common for a particular piece of combinational logic to stabilize across several clock phases. For example, the input to an ALU might be loaded at the beginning of Phase 1 and the output latched at the end of Phase 2. In this case, Crystal will "bill" the entire delay of the ALU to Phase 1: it assumes that anything that could change in a clock phase must change during that clock phase. Work is currently underway to relax this restriction. Instead of requiring all nodes to settle during each clock phase, only static and dynamic memory nodes loaded by that clock must settle. Other nodes can continue settling during the next clock phase. This mechanism has only recently been implemented and is still undergoing testing. It appears to be important, though: all of the designs that have used Crystal so far contain such cross-phase signals.

## 4. Weakness of the Delay Model

The simple resistive approximation for transistors can occasionally produce large errors in delay estimates. Consider the situation of Figure 9. In (b) the inverter will take much longer to drive its output to zero than in (a)

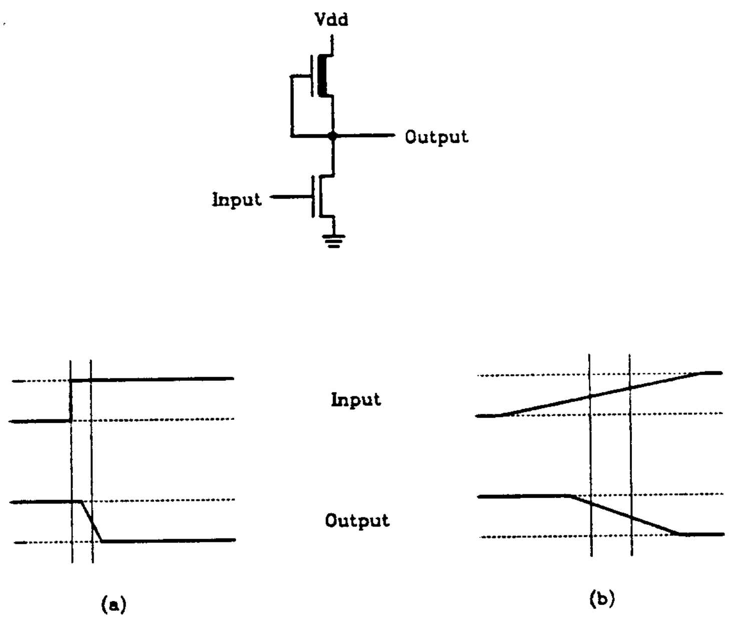

**Figure 9.** If the input to an inverter rises quickly to 1, as in (a), the output is driven quickly to 0. If the input rises very slowly as in (b), the pulldown transistor is driven by a lower gate voltage, so the output falls more slowly.

because the gate voltage is much lower than Vdd when the output is being driven in (b). In general, the effective drive power of a transistor is a function both of the input waveform and of the load being driven, whereas Crystal considers only the load being driven. If the characteristic resistance for a transistor is chosen based on fast inputs, the delay for a given device may be underestimated by as much as an order of magnitude. In the case of Figure 9(b), the delay of the input signal is much greater than the delay of the inverter itself, so the underestimate for the inverter's delay will cause only a small percentage error in the overall delay estimate for the circuit. However, there are many situations where overall errors of as much as 50% or more will occur if a single resistance value is used for each transistor type (see Section 5).

The solution to this problem is to include the input waveform in the delay calculation. An approach that we are currently exploring is to include slopes in delay calculations. In the slope model, the low-to-high and high-to-low times for each node will be accompanied by slopes. The resistance values of transistors will be specified as functions of the transistor type, input slope, and load being driven.

## 5. Experiences Using Crystal

It was fortunate, and not entirely coincidental, that the RISC II Cache chip was undergoing final pre-fabrication validation at the same time that Crystal was developed. The cache has been used throughout the development of Crystal as a large test case. It contains about 46000 transistors, over half of which are in a large memory array. It uses a four-phase non-overlapping clocking scheme. Crystal uncovered a half-dozen performance bugs that would have forced the clock to run 40% slower than planned. The most important bug found was a small transistor accidentally left to drive very long line, resulting in a 60ns delay for that one wire. In virtually all cases, the long delays were due to single isolated transistors; in each case there was plenty of space to increase the driving size. Thus, the performance tuning was easy to do, once the slow transistors were identified.

Approximately 20 minutes of CPU time are required to process the four clock phases of the cache on a VAX-11/780. Of this time, only 6 minutes was spent in delay analysis (only one of the four clock phases required more than 1 minute of analysis time); 7 minutes were required just to read in the 2.5 megabyte circuit description file, 4 minutes were used to propagate fixed node values, and the remaining 3 minutes were used in scanning the database to print out results. Although Crystal is relatively fast, it is not small: the program requires nearly 5 megabytes of virtual address space to process the cache chip, or about 100 bytes per transistor.

The RISC II microprocessor has also been analyzed using Crystal. RISC II contains over 40000 transistors, and the design has been carried out with relatively ambitious performance goals. The designers did not initially expect Crystal to be ready in time for their use, so they made extensive timing checks using SPICE and hand calculations. The Crystal analysis of this circuit pinpointed only one performance bug, which would have slowed the clock time by 40%. Crystal also uncovered a functional error that was not detected during simulation because none of the test cases triggered it. The timing analysis for the RISC II microprocessor requires over an hour of CPU time.

The third Crystal experience to date is a small circuit (2100 transistors) provided by the VLSI Systems Area at Xerox PARC. This circuit had been fabricated and tested before running it through Crystal. The initial version of the circuit had required 200ns for Phase 1, instead of the hoped-for time of 50ns. The designers found and fixed a performance bug, and the second fabrication of the circuit ran at 50ns. We ran both versions of the circuit through Crystal. In the first version, Crystal pinpointed the performance bug and estimated a clock time of around 190ns. In the second version of the circuit, Crystal estimated a clock time of about 55ns. The entire Crystal analysis of both versions of the circuit, including modifying the mask layout to produce the second version and extracting the circuit, took less than one day of real time. Crystal's analysis required less than a minute of CPU time.

Crystal provides several kinds of output. It identifies the longest path for each clock phase, and can also print out the longest paths. If desired, Crystal will identify single-stage delays that are longer than a given value, so designers can identify all the slow drivers. Crystal will also pinpoint nodes with large capacitance or resistance values. Output is provided both textually and graphically. Graphical output is provided by generating a command file for the Caesar layout editor [7]: when Caesar processes the command file it identifies the critical path with patches of a special layer along with text giving the delay to each point. Caesar commands can then be used to view each transistor in the path in detail, from beginning to end.

Crystal also generates a SPICE deck for the critical path that it finds. The SPICE deck includes all the transistors along the path as well as parasitic capacitance and resistance for the interconnect. No information is output for side paths. Table 2 compares Crystal's and SPICE's estimates for several

| Circuit | Crystal | Spice | Error |
|---|---|---|---|
| Test | 20ns | 21ns | 5% |
| PLA | 17ns | 18ns | 6% |
| Cache Phase 1 | 79ns | 111ns | 41% |
| Cache Phase 2 | 176ns | 270ns | 53% |
| Cache Phase 3 | 61ns | 83ns | 37% |
| Cache Phase 4 | 104ns | 170ns | 64% |

**Table 2.** Comparison of Crystal and SPICE delay calculations for several sample circuits. Test consists of 4 AND gates and inverters in series. PLA is a small programmed logic array with 4 inputs, 6 minterms, 4 outputs. The last four circuits are the critical paths (as determined by Crystal) for the four clock phases of the RISC II Cache chip.

sample circuits. For small circuits Crystal's estimates are very close to SPICE's more detailed calculations. For the larger cache circuits, Crystal's error relative to SPICE is 40-60%. The deviation between SPICE and Crystal is almost entirely due to problems of the sort discussed in Section 4.

## 6. Summary

The following list contains the approximations made by Crystal in computing delays:

[1] Each transistor type is characterized by an effective resistance when pulling to Vdd and another effective resistance when pulling to Ground.

[2] Capacitance from side paths is not included: Crystal considers only the direct path from a particular gate to Vdd or Ground.

[3] When examining a path, the closest depletion load device to the target gate is assumed to be responsible for pulling the gate high.

[4] For nodes with large capacitance, delays through the node are broken down into separate delays to the node and from the node.

In spite of these approximations, I believe Crystal will be a useful tool for designers, particularly if the model can be upgraded to include slope information. In any case, even the simplest model has enabled designers to find performance bugs and to pinpoint potential trouble spots for more detailed analysis with SPICE.

## 7. Acknowledgements

Berkeley's VLSI community provides a fertile environment for experimentation with new design tools. The ambitious design projects led by Prof. Dave Patterson motivate much of the tools work and validate the results. Lynn Conway, Alan Bell, and Gaetano Borriello of the VLSI Design Area at Xerox PARC provided early advice to us as well as the test case discussed above. Dimitris Lioupis, Bob Sherburne, and Gaetano Borriello were early users and critics of Crystal and have provided important feedback. Dan Fitzpatrick modified our circuit extractor to provide flow information for transistors. Dave Patterson and Carlo Séquin have provided helpful comments on this paper.

The work described here was supported in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3803, monitored by the Naval Electronic System Command under Contract No. N00039-81-K-0251.

## 8. References

[1] Bening, L.C., et al. "Developments in Logic Network Path Delay Analysis." *Proc. 19th Design Automation Conference*, 1982, pp. 605-615.

[2] Foderaro, J.K., Van Dyke, K.S., and Patterson, D.A. "Running RISCs." *VLSI Design*, Vol. III, No. 5, Sept./Oct. 1982.

[3] Jouppi, N.P. "TV: An NMOS Timing Verifier." *3rd Caltech Conference on VLSI*, 1983.

[4] Kirkpatrick, T.I. and Clark, N.R. "PERT as an Aid to Logic Design." *IBM Journal of Research and Development*, Vol. 10, March 1966, pp. 135-141.

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

[6] Nagel, L.W. "SPICE2: A Computer Program to Simulate Semiconductor Circuits." ERL Memo ERL-M520, Univ. of California, Berkeley, May 1975.

[7] Ousterhout, J.K. "Caesar: An Interactive Editor for VLSI." *VLSI Design*, Vol. II, No. 4, Fourth Quarter 1981, pp. 34-38.

[8] Patterson, D.A., et al. "Architecture of a VLSI Instruction Cache." To appear, *10th Int'l Symposium on Computer Architecture*, 1983.