Timing Verification and the Timing Analysis Program

Robert B. Hitchcock, Sr.

IBM General Technology Division
Endicott, New York 13760

ABSTRACT

Timing Verification consists of validating the path delays (primary input or storage element to primary output or storage element) to be sure they are not too long or too short and checking the clock pulses to be sure they are not too wide or too narrow. The programs addressing these problems neither produce input patterns like test pattern generators nor require input patterns like traditional simulators. Several programs (described here) operate by tracing paths [PI73, WO78, SA81, KA81]. One program [MC80] extends simulation into a pessimistic analyzer not dependent on test patterns.

Timing Analysis, a program described recently in [HI82a], is designed to analyze the timing of large digital computers and is based, in part, on the concepts disclosed in a patented method [DO81] for determining the extreme characteristics of logic block diagrams. The output of Timing Analysis includes "slack" at each block to provide a measure of the severity of the timing problem. The program also generates standard deviations for the times so that a statistical timing design can be produced rather than a worst case approach.

I INTRODUCTION

Product Verification consists of three main parts, Functional Design Verification, Physical Design Verification, and Timing Verification. Functional Design Verification means simulating at a high level and/or low level to insure that the design as implemented produces the desired results. Physical Design Verification means checking the physical parameters: the separations of wires and other physical shapes to be sure they meet the required minimums, the total capacitance in the nets to be sure they are within the allowable bounds, the geometric arrangements of the physical nets to be sure they fit the configurations for which delays can be predicted, and so on. Timing Verification consists of validating the path delays (primary input or storage element to primary output or storage element) to be sure they are not too long or too short and checking the clock pulses to be sure they are not too wide or too narrow.

Problems found by any of these verification steps can impose changes to the design. As the onslaught of VLSI continues, it becomes more and more important to perform complete analyses of each of these steps prior to the building of actual hardware.

To insure that a digital computer will operate at a desired speed, the designer must verify that several conditions are true within the design:

1. All internal paths, which can be activated from storage element or PI to storage element or PO, will have delays neither too long nor too short (Figure 1).
2. All clock signals propagating though the repowering circuitry will retain the required minimum pulse width despite the shrinking effect of non-symmetric delays (Figure 2).

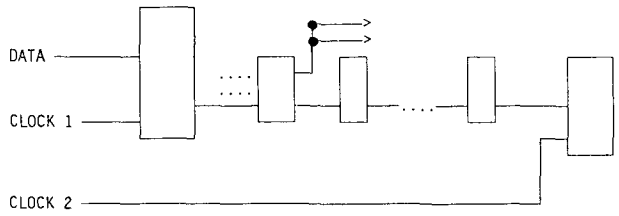
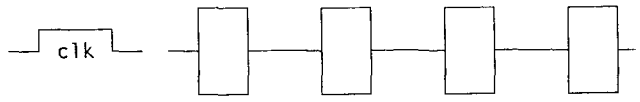


Figure 1 - The storage element timing problem: Make sure that every transition stored by one clock will arrive at the next storage element in time to be gated in by the other clock.



Rising Delay	10	5	10	5
Falling Delay	6	3	6	3

Clock at Input: Rise at t=0 , Fall at t=10.

Clock at Output: Rise at t=26 , Fall at t=32.

Figure 2 - Clock Pulse Width Shrinkage Illustration: The Blocks are all assumed to invert, so the asymmetric delay, coupled with the overall differences in delay between blocks in subsequent stages, causes the clock to shrink from a separation of ten units to a separation of six units.

Several automated approaches have been reported; some of these have centered on the analysis of the detailed electrical properties of the components along a path, using an approach which enumerated all paths [PI73, WO78, SA81, KA81]. These so-called **path oriented** approaches will be described in greater de-

0420-0098/82/0000/0594$00.75

tail later. Others have taken the delays of the blocks to be well defined numbers and have attempted to optimize these delays within a power constraint [AG77, RU77].

Techniques which tend to process each block a relatively small number of times have also been described. [KI66, MC80, DO81, HI82a] These so-called **block oriented** approaches will also be described in more detail later.

To illustrate how these types of algorithms operate, a diagram used in [HI82a] will be used. It has been redrawn with single letters designating the individual blocks for more easy reference (Figure 3).

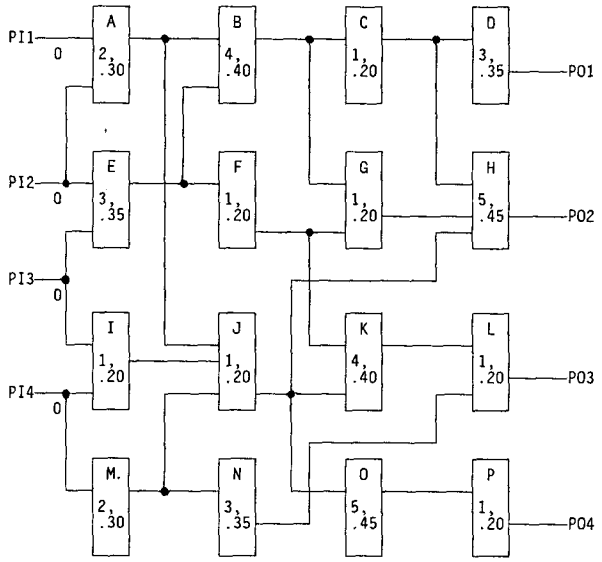


Figure 3 - Basic Logic Diagram: This diagram is used to illustrate the operation of several of the algorithms described. The delays are shown at the bottom of the block, mean first and standard deviation (sigma) second.

The main emphasis of this report is to familiarize the reader with the two main techniques for verifying performance in computer designs: path enumeration and block oriented techniques. Section II describes examples of each kind of algorithm in detail using a small block diagram to illustrate some of the features of each. Section III discusses the Timing Analysis program developed at IBM. TA is a block oriented program which not only identifies the most critical path(s), but also provides a value at each block, called the **slack**, which is a measure of the degree to which the worst path through that block meets its timing constraints.

II PREVIOUSLY DESCRIBED APPLICATIONS

Previously described techniques tend to group into two main categories: path enumeration techniques and block oriented analysis techniques. These main techniques will be introduced, and then the programs themselves will be described, with a small circuit used to illustrate how they would operate. Both of these techniques are quite different from simulators or test generators. Contrary to what one would expect in a simulator, no attempt is made to prove that a path can be sensitized before its delay is calculated and no values are assigned to inputs to trigger actions along a path whose delay is being calculated.

Path Enumeration Techniques

Path enumeration techniques are characterized by programs which start at certain "start points", and trace back through the blocks feeding the start points until a PI or a "terminal point" is reached. At this stage, the information is complete and the data for the path can be accumulated. Data accumulation can be done by simply adding up the delays, or by combining the delays statistically.

Path enumeration techniques tend to have long running times since the number of paths through a graph grows exponentially with the size of the graph. However, there is often an advantage in having an entire path available when the analysis is to be done: the delay of a given circuit may depend not only on the input and output loading and the intrinsic properties of the circuit, but also on the rise time of the signal at its input. By knowing what circuit is the primary driver of the path, one can determine the rise time, and more accurately determine the delays of all the circuits. It is also easier to tell such a path oriented program to ignore certain paths which will never be activated in the actual machine (unless, of course, there are an excessive number of such ignorable paths).

NELTAS [SA81] includes a set of programs which tend to be part path enumeration and part block oriented analysis, since in one mode all paths are enumerated, and in the other mode only "critical paths" are evaluated, much as in the block oriented techniques. The programs or papers describing systems which, for purposes of this discussion, have been assigned to the path enumeration category are:

1. Computer-Aided Prediction of Delays in LSI Logic Systems- (Fairchild and Amdahl), D. J. Pilling and H. B. Sun, [PI73].

2. GRASP - Global Timing Analysis (Honeywell), M. A. Wold [WO78].

3. NELTAS - Network Delay Time Analysis Subsystem (Nippon Electric), T. Sasaki, et. al. [SA81].

4. A Critical Path Delay Check System - (Hitachi), R. Kamikawai, et. al., [KA81]

Block Oriented Techniques

Block oriented analysis techniques are characterized by programs which start with signals at given times at PIs or at storage element outputs (note - the times, when the signals are available at the storage element outputs, will depend on the time at which the clocks permit data to be latched into them). The blocks, which these signals feed, are then processed to find the latest and/or earliest time at which the signal could propagate through them. The blocks are processed so that the times at the output of each block is calculated (ideally) once.

Block oriented analysis techniques tend to run much quicker than enumerative path analysis techniques, but tend to be very pessimistic. Whereas, a path oriented algorithm can accept commands to ignore a specific path, a block oriented algorithm cannot without cutting many other paths at the same time. The block oriented algorithm identifies the worst path leading up to each block (the critical path up

to that point) and then feeds that information forward. Once the worst case delay or time has been calculated, the identity of any specific path causing that information to be what it is has been lost. The programs or papers describing systems which, for purposes of this discussion, have been assigned to the block oriented category are:

1. PERT as an Aid to Logic Design - (IBM), T. I. Kirkpatrick and N. R. Clark, [KI66].

2. SCALD Timing Verifier - (Stanford), T. M. McWilliams, [MC80].

PATH ENUMERATION TECHNIQUES

Computer-Aided Prediction of Delays in LSI Logic Systems - Fairchild and Amdahl, D. J. Pilling and H. B. Sun, [PI73].

Pilling and Sun [PI73] describe a program to take the description of the physical circuits in a chip design (both before and after the physical masks had been designed) and calculate an estimated delay for the circuit. This is then fed into the simulation program for more accurate simulation and, by listing the circuits in a critical path, the total delays in that path are determined.

The delays predicted by this approximation technique are reported to be within 5% of the delays measured on actual circuits under identical conditions.

The exact details of the method for determining the critical paths and adding up the delays is not described. One block in their Figure 7 indicates a process "**Compute Total Delay Along Signal Paths**" where the path could be manually specified or computer traced, we are not told which. Since there is a great deal of ambiguity in this, no attempt will be made to show how the process would attack the simple logic diagram in Figure 3.

GRASP (Global Timing Analysis) - Honeywell, M. A. Wold [WO78].

The designer specifies the set of start points and the set of terminal points, as well as the boundary values for path classification. GRASP enumerates all possible paths from the set of start points to the terminal points and calculates the mean, MIN and MAX delays of each. Paths are then classified as "critical", "marginal", "normal", or "trivial" according to the designer specified values.

The delays are calculated from the physical design data by using linear approximation equations when the physical design configurations are within the constraining parameters, and by "network simulation" (a Monte-Carlo analysis of the behavior of the physical network, with enough points taken to be able to derive both a mean delay and a standard deviation with reasonable accuracy) when the rules are not met. The resulting delays include both a mean and standard deviation.

For each path being evaluated, the mean path delay is derived by taking the sum of the mean delays of the blocks along the path. The standard deviation of the path is calculated by the root-sum-squared of the standard deviations of the blocks along the path. This is the standard way to calculate the mean and standard deviation for the sum of independent random variables if they are assumed to start out with a "Gaussian" distribution.

Considering the example (Figure 3), the start points are defined as the set of PO's: (PO1, PO2, PO3, and PO4), and the terminal points as the set of PI's: (PI1, PI2, PI3, and PI4). The analysis then identifies all 32 paths (see Figure 4). As an example, we will focus on one of these paths, PI1-A-B-C-H-PO2.

```
PI1 - A - B - C - D - PO1
                  - H - PO2
            G - H - PO2
        J - H     - PO2
          - K - L - PO3
          - O - P - PO4
PI2 - A - B - C - D - PO1
                  - H - PO2
            G - H - PO2
        J - H     - PO2
          - K - L - PO3
          - O - P - PO4
PI2 - E - B - C - D - PO1
                  - H - PO2
            G - H - PO2
        F - G - H - PO2
          - K - L - PO3
PI3 - E - B - C - D - PO1
                  - H - PO2
            G - H - PO2
        F - G - H - PO2
          - K - L - PO3
PI3 - I - J - H     - PO2
          - K - L - PO3
          - O - P - PO4
PI4 - I - J - H     - PO2
          - K - L - PO3
          - O - P - PO4
PI4 - M - J - H     - PO2
          - K - L - PO3
          - O - P - PO4
        N - L     - PO3
```

Figure 4 - Basic Logic Diagram Paths

In the program, the delay is actually calculated for each block by looking at the detailed description of the physical design, but for our purposes a delay value has been assigned to each block in the path. The delay values are written in the lower part of the blocks. The means of these delays are, respectively, 2, 4, 1, and 5. Adding these up, the path nominal delay is 12. The sigma values for each delay can be sizeable, the smaller delays often have the largest relative sigmas, so that the individual sigmas are assumed to be .3, .4, .2 and .45 respectively. This yields an overall sigma of .702, the square root of the sum of the individual sigmas squared. For this path, therefore, the total effective (three-sigma) MAX delay is 14.1, the total effective MIN delay is 9.9. The only remaining operation from this point would be to categorize the net as being critical, marginal, normal or trivial. This would depend on the parameters set by the designer using the program.

NELTAS (Network Delay Time Analysis Subsystem) - Nippon Electric, T. Sasaki, et. al.

As described in [SA81], it would appear that, within each packaging level, the NELTAS programs evaluate

the paths from a PI or register output to a PO or register input. The delay analysis operates in two modes: **logical path mode** where all possible paths are enumerated, and each is analyzed in detail, and **critical path mode** where at each intermediate point the characteristic of the worst path up to that point is determined, and whenever that point is reached by another trace, the characteristic of the worst path is used, rather than tracing further.

In **logical path mode**, the operation of NELTAS would be virtually identical to Wold's GRASP system, with each path from start point to terminal point being traced and evaluated. Since the previous section illustrated this operation, it will not be duplicated here.

In **critical path mode**, the NELTAS process is more of a block oriented approach in that each intermediate point would have the worst-case (MIN and MAX) delay time calculated only once.

The information regarding the delays derived from the lower levels is stored in a library for use by the higher level analysis routines. The specific details of this hierarchical process were not presented in [SA81].

To illustrate the **critical path mode**, a trace back from PO1 will be considered. To show what worst case "means" and "sigmas" are known, a table of blocks with known worst cases will be shown after each significant step in the algorithm. The trace would proceed through blocks D and C, since they only have one input each, and at block B would first select input 1, which comes from block A. In this example, A is fed directly from PIs so its output mean and sigma are also the worst mean and sigma.

Block	MAX Mean	MAX Sigma	MIN Mean	MIN Sigma
A	2	.3	2	.3

Current Block	Trace-back Input
D	1
C	1
B	1
A	*(all)

At each stage, all inputs will be tested. If an input has had its value calculated by a previous action, its MIN (MAX) will be used directly. At the end of the traceback from D, the tables will be:

Block	MAX Mean	MAX Sigma	MIN Mean	MIN Sigma
A	2	.3	2	.3
E	3	.35	3	.35
B	7	.532	6	.5
C	8	.566	7	.539
D	11	.667	10	.642

Current Block	Trace-back Input
D	*

To illustrate why this is so powerful, consider the traceback from PO2. The first input from H comes from C which is already marked as being on the worst case list, so the trace goes directly to G. G's first input comes from B, another block already marked as being on the list, so its trace goes directly to F. F's input comes from E which already has its worst case values calculated, so F can be added directly to the list:

Block	MAX Mean	MAX Sigma	MIN Mean	MIN Sigma
A	2	.3	2	.3
E	3	.35	3	.35
B	7	.532	6	.5
C	8	.566	7	.539
D	11	.667	10	.642
F	4	.403	4	.403

Current Block	Trace-back Input
H	2
G	2
F	*

Now G can be calculated from its inputs (B and F), and so on.

By extending this analysis to allow for differing rising and falling delays, the expansion and/or contraction of clock pulses can be determined (see Figure 2). Since clock re-powering networks usually are in the form of an expanding tree structure, the MIN and MAX delays would be, for the most part, the same and the main concern would be with the compression of pulse widths due to the asymmetric delay values themselves.

A Critical Path Delay Check System - Hitachi, R. Kamikawai, et. al. [KA81].

In this system, delay checks are made at each stage of the design process. Delays are approximated at the early stages in the process, with the approximations becoming more realistic as the design progresses to the later stages. Although there is an indication that once the sub-network connecting a start node to a terminal node has been traced out, the blocks in that sub-network are analyzed only once; there is every indication that a block may be reevaluated every time it appears in a sub-network. Therefore, this algorithm has been classified more as a path oriented enumeration routine than a block oriented analysis routine.

The delay check takes part in several stages. The start and end points (defined by the user) are used to guide the "path trace". The "path trace" is similar to the standard wire-routing maze-running algorithm. It extends the nodes in the forward and backward frontier stage by stage unless:

a) the frontier node is a flip-flop

b) the frontier node goes no further, or

c) the user controlled maximum number of frontiers has been reached.

d) for purposes of this discussion, another condition has been borrowed from wiring programs: no node will be included in one frontier as a candidate for further extension if it is already in the other one.

When an end point is reached, a trace back from it to the start point marks the nodes to be kept. The turn-on and turn-off transition delays are then calculated for each net connecting the marked nodes.

The problem then is to find the critical path. Although the algorithm is only hinted at, there is an indication that each node is processed once - when the delay time from the start node to all nodes feeding it have been determined. By this means, the output of the node being analyzed will just be its own

delay time plus the time of the maximal input (the first input whose delay time equals or exceeds all other inputs). When the maximum delay times for all nodes have been determined, the maximum time for the end point will be known. Determining the critical path then corresponds to just tracing back through the maximal inputs feeding each node until the start node (flip-flop or PI) is reached.

To illustrate this technique, we will first select PI3 as the start point and PO2 as the end point. The forward and backward frontiers for the tracing will be shown in two tables at each stage. The initial stage will just include the two primary connections:

Forward Frontier Blocks	level	Backward Frontier Blocks	level
PI3	0	PO2	0

After tracing two levels from the start points, the tables will include the blocks directly connected to PI3 and PO2 and those connected one level further back:

Forward Frontier Blocks	level	Backward Frontier Blocks	level
PI3	0	PO2	0
E, I	1	H	1
B, F, J	2	C, G,(J)	2

The entry (J) in the backward frontier indicates that J could not be added to the table because it already was in the other table. It therefore is a link between the two points. After one more step, the backward frontier can go no further, two additional steps add the last two levels to the forward frontier:

Forward Frontier Blocks	level	Backward Frontier Blocks	level
PI3	0	PO2	0
E, I	1	H	1
B, F, J	2	C, G,(J)	2
(C),(G),K,O	3	(B),(F)	3
L, P	4		
PO3, PO4	5		

After tracing from the connection points back to the origination points and marking the blocks one passes through, a graph of marked blocks is produced consisting of blocks B, C, E, F, G, H, I, and J (see Figure 5). At this point, the delays for the marked blocks would be calculated and then the analysis of the sub-network would find the critical path. Since no explicit algorithm was described, one can just point to the analysis in the previous section as one way in which this could be accomplished.

BLOCK ORIENTED TECHNIQUES

PERT as an Aid to Logic Design - IBM, T. I. Kirkpatrick and N. R. Clark, [KI66].

Kirkpatrick and Clark [KI66] describe a way to apply the PERT project scheduling tool to aid in analyzing logic designs for proper timing. Although the potential for selecting the worst arrival time (completion time, in PERT's terminology) is there, the ability to handle two sets of delays (for turn-on and turn-off) would have required a, so-called,

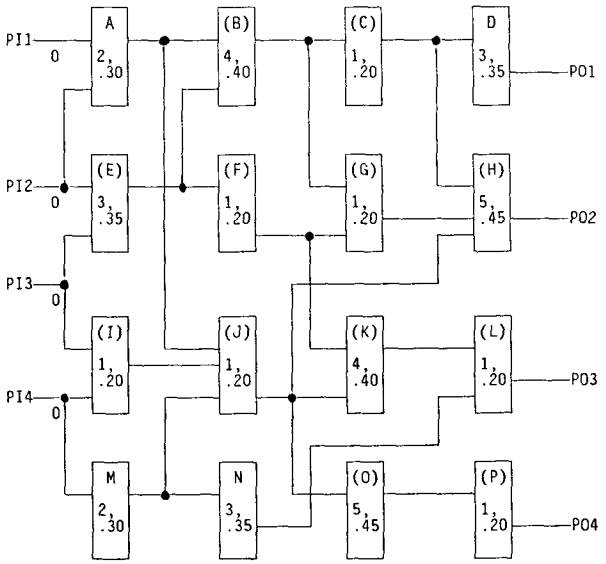


Figure 5 - Result of Tracing Frontiers

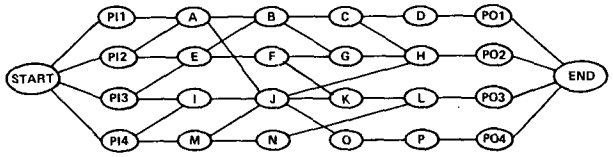


Figure 6 - PERT Model for the Basic Logic Diagram: To model the Basic Logic Diagram, each block becomes a NODE, and a START and END node must be added.

"two-rail" model of the logic design[1]. As described in their article, it would seem that to be able to analyze for statistical arrival times on the critical path, one had to first know, for each gate, whether the turn-on or turn-off delay was to be used (or select the worst of the two delays)[2]. The assumption was also made that processing for worst-case values should be dependent on the logic function of a gate (latest at an AND, earliest at an OR).

More recent analysis [MC80, HI82a] indicates that when looking for the worst-case late arrival time, the function is irrelevant. All that is important is the inverting properties of a functional element and the individual turn-on and turn-off delays.

---

1 A two-rail design is one where each signal and its compliment are computed independently by complementary logic to avoid the need for extra inverters. A two rail model would represent each block by two, one to take into account the rising signal and the other to take into account the falling signal, each with its own delay.

2 It should be remembered that at the time this report was written, there was not the great disparity between rising and falling delays in high-speed circuits that there is today.

Typically, the information being handed to PERT is the nominal, best-case and worst case time for completion of a task. For example, if one is analyzing the time it would take to build a house, one task could be the time it takes to build concrete forms, another could be the time it takes to fill the forms with concrete, a third could be the time it takes the concrete to dry, and so on. Note that with respect to the concrete hardening task, the other two tasks would have had to be done (in order) before it could even start. Kirkpatric and Clark consider a task to be the time it takes a signal transition to propagate from the output of one block to the output of the block it feeds. See Figure 6 for the PERT model of the Basic Logic Diagram.

One way to analyze a PERT model is to levelize the nodes (as we often levelize logic blocks in Design Automation programs - a block is in level k if the maximum number of blocks back to a PI from that block is k) and then process the blocks in level order: first the blocks in level 1, then the blocks in level 2, and so on. Kirkpatrick and Clark were primarily interested in finding the critically long paths, so the analysis would have calculated the mean and sigma for the MAX path from the START node.

Thus, blocks (nodes) being fed by the PI blocks of Figure 6 will have a worst case arrival time (in PERT's terminology, this is just the "completion time") equal to the time it takes to get through the block ("complete its task") plus the maximum arrival time of any primary input (which will all be assumed to be zero). Block B, which has one input at time two and the other at time three, is typical of other blocks in the figure. Three is greater, the internal delay is four, therefore seven is the maximum output arrival time for this block. This is similar to the **critical path analysis** calculations shown to describe the operation of NELTAS.

These calculations yield the outputs of blocks D, H, L and P, at times eleven, thirteen, nine and nine, respectively.

SCALD Timing Verifier - Stanford, T. M. McWilliams, [MC80].

The structure is presented to the Timing Verifier by other programs in the SCALD (Structured Computer-Aided Logic Design, [MC78]) system along with models for each wire or component and any designer specifed overrides to wire delays.

What differentiates the Scald Timing Verifier from the ordinary simulator is the extension of the simulation algebra to contain the values STABLE, (a signal would be at the correct value, either ZERO or ONE) and CHANGING (the signal would be potentially changing from one stable value to another but be unstable during this time period)[MC80]. Using these values, the designer is able to specify the equivalent of many combinations of input values by simply defining them to be STABLE at a certain time. Likewise, the timing checks for storage elements, registers, and the like are in terms of a comparison of the active time of the clock with the STABLE time of the input data signal.

The evaluation tables are defined so that the most dominant value is U (Undefined), with CHANGING often running a close second. If all inputs are defined (non-U), then if any input is CHANGING, the output **may** be CHANGING. The exception to this rule for an OR, for example, is that a ONE on any input will force the output to be a ONE regardless of what the other inputs do. Thus, for an OR, ONE dominates the other inputs. Similarly, a ZERO at any input of an AND block dominates the other inputs. An "exclusive-OR" block, on the other hand, has no single value which dominates the others, and the evaluation tables shown in [MC80] indicate this. Thus, in the absence of a dominating signal, the output of a block will be CHANGING at time t as long as any of its inputs was CHANGING at time t minus the circuit delay. This means that if several inputs to a block go from being CHANGING to STABLE and back to CHANGING, then the output will not start being STABLE until the last input becomes STABLE (plus the delay of the circuit) and will indicate potential CHANGING when the first input indicates it is CHANGING (plus the circuit delay). Note that this means that the leading edge of the STABLE time will be the maximum time at which any input signal becomes stable, and the trailing edge will become CHANGING at the minimum time when any input signal becomes CHANGING.

The use of "signal assertions" allows the designer to drive portions of the logic, for which the driving circuitry has not yet been completely designed, and permits the driving circuitry to be checked when it is finally completed. A "signal assertion" is a way of coding the timing for a clock or of asserting the time during which a signal is expected to be STABLE. The coding is appended to the end of the signal name which facilitates displaying it on the hardcopy output of SCALD. Undefined signals, with no assertions, are taken to be always STABLE to avoid spurious timing error messages.

Timing checks are done by special blocks which check setup times, hold times, and minimum pulse width times. These blocks may be entered by the designer, but more often they are added to the design by the expansion of macro blocks used to call out latches, registers or entire RAM's.

Due to the pessimistic nature of this analysis, the occurrence of unbalanced designs can cause the analysis to predict point-to-point delays larger or smaller than permitted. When these conditions are identified, "case analysis" is used to introduce some realism into the pessimistic analysis. For example in Figure 7, taken from [MC80], the pessimistic analysis would predict the shortest path to be 20ns and the longest to be 40ns. In fact, regardless of which specific value is assigned to CTL, the delay from A to B will always be 30ns.

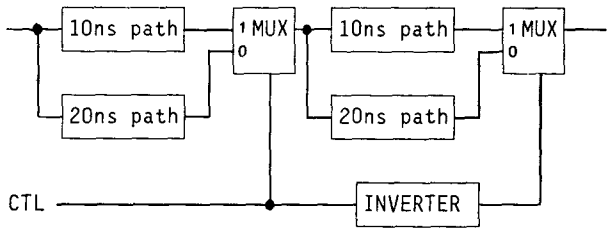


Figure 7 - Case Analysis Example

Assuming these extremes (20ns and 40ns) cause annoying erroneous diagnostics, two cases are set up, the first sets CTL to ZERO, the second sets it to ONE. (If the extremes cause no extraneous diagnostics, they can be ignored.) In each case, one path is analyzed and the other is ignored. The result is a more accurate analysis (removing only enough ambiguity to eliminate the erroneous diagnostics), without having

to go to the opposite extreme where all possible ZERO and ONE patterns at the PI's are tried to insure proper timing.

To parallel the results to be shown for the Timing Analysis initial results, the PI assertions will all be 0S (STABLE starting at time t=0).

The result of processing the blocks fed directly by these primary inputs will be derived assertions for Blocks A, E, I, and M (see below). Since we are assuming that the signals are CHANGING until the first assertion that they are STABLE, this means that in calculating the derived assertion for each block, several choices will have to be made. At block B, at least one input remains CHANGING until time 3, so its output cannot become STABLE until time 7. Similarly, block J cannot become STABLE until time 3. As indicated, the Timing Verifier does not do statistical calculations, so the times shown represent the mean result. Similar calculations for the remaining blocks give assertions as follows:

Signal Name	Derived Assertion	Signal Name	Derived Assertion
A	S2	I	S1
B	S7	J	S3
C	S8	K	S8
D	S11	L	S9
E	S3	M	S2
F	S4	N	S5
G	S8	O	S8
H	S13	P	S9

where Sn means STABLE starting at t=n

Note, that although the results here are rather straight-forward, the Timing Verifier has many more sophisticated features incorporated in it. Checks are made to be sure that clock pulse widths are maintained. Checks are made at special blocks to be sure that data signals are STABLE during critical portions of clock signals (especially during the setup and hold time periods). And, as mentioned before, the use of signal assertions allows the designer to specify what is anticipated to come out of a yet uncompleted portion of the design, and later on use the same assertions to check to be sure that what was expected actually occurred.

III TIMING ANALYSIS - IBM, R. B. HITCHCOCK, G. L. SMITH, AND D. D. CHENG, [HI82A].

Timing Analysis (TA) is a design automation program that assists computer design engineers in locating problem timing in a clocked, sequential machine. The program is effective for large machines because, in part, the running time is proportional to the number of circuits. Thus, in comparing with what has been discussed previously, the Timing Analysis program is a block-oriented algorithm. What distinguishes it from the others is that the output of Timing Analysis includes **slack** at each block as a measure of the severity of any timing problem. The program also generates standard deviations for the times so that a statistical timing design can be produced rather than a worst case approach. This system has successfully detected all but a few timing problems for the IBM 3081 system (consisting of over 700K circuits) prior to the hardware debugging of timing. The 3081 system is characterized by a tight statistical timing design.

THE TIMING ANALYSIS CONCEPTS

Timing Analysis is a program which establishes whether all paths within the design meet stated timing criteria, that is, that data signals arrive at storage elements early enough for valid gating but not so early as to cause premature gating (see Figure 1). The output of the TA program not only identifies any logic with timing problems, but also provides **slack** as a measure of the severity of each problem. By treating the delays statistically, TA permits the creation of a design with a shorter clock cycle than would a worst case analysis.

As with the other programs mentioned earlier, the program does not require the input of a set of test patterns, nor does it overlook paths because patterns were omitted. In addition, the program provides a mechanism, using **delay modifiers**, to alter the propagation characteristics of individual blocks when it is known that logic causes paths to never be sensitized or to correctly analyze abnormal timing criteria (such as a two-cycle path).

In order to apply the TA program, the design must be a synchronous, sequential machine and it must be possible to associate explicit clock times with each storage element.

THE TA BLOCK-ORIENTED ALGORITHM

Our description of the basic algorithm assumes that the analysis is trying to generate late signals, that is, we are looking for long paths. Figure 3 shows a simple example of combinational logic which serves as a basis for the discussion.

The lower portion of Figure 8 contains a key defining the three values associated with each of the blocks on the diagram. For this portion of the description, the calculation will be limited to the mean delays associated with the block and the critical paths. For example, only one delay **D** is shown for a block, rather than one for rising and one for falling transitions. The number, **AT**, written beneath the output of each block, is the maximum arrival time (in whatever units of time are chosen as appropriate). The primary input (**PI**) arrival times and the primary output (**PO**) required arrival times, which are normally derived from clock arrival times, are assumed to be explicitly given. The slack **S** is discussed subsequently.

The blocks are processed by Timing Analysis in an order functionally equivalent to the levelizing order mentioned in the discussion of NELTAS so that it is guaranteed that every input feeding a block will have its arrival time defined before the block itself is processed. The arrival time of a block is computed by:

> Establishing the arrival time at the block output for each input to the block by propagating each input signal through the block delay,
>
> Selecting the maximum such arrival time.

These calculations are identical to the ones illustrated under the section describing PERT and yield the outputs of blocks D, H, L and P, at times eleven, thirteen, nine and nine, respectively. Note that we have explicitly stated a required arrival time of ten for all the outputs. This means that two signals are late and two signals are early.

At this point, we define the slack **S** as the difference between the required arrival time and the actual arrival time. The slack value is computed so that a negative number indicates a problem; that is, the signal is too late. The output of block D arrives at time eleven, one unit late (slack = -1). Block H is three units late (slack = -3); block L is one early (slack = +1); and P is three early (slack = +3). The

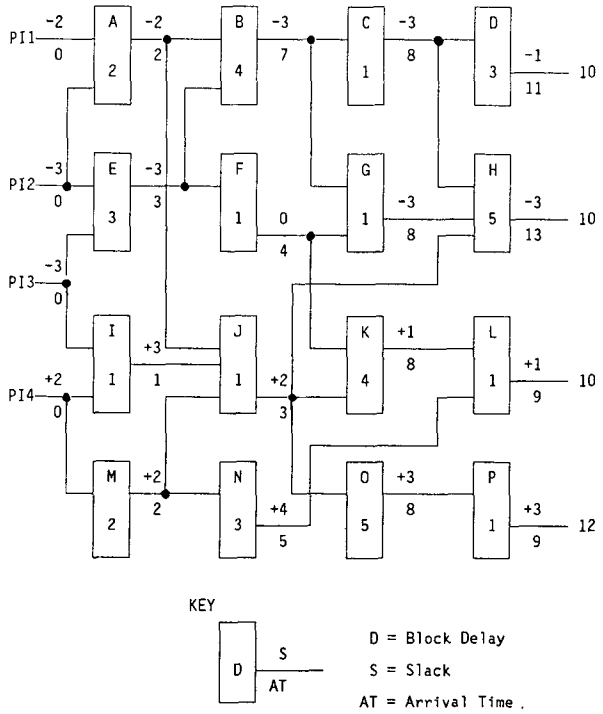


Figure 8 - Basic Diagram with Delays, Arrival Times and Slacks

slack information is then propagated back through the block graph, with all blocks being reprocessed in order opposite to that used for generation of the arrival times.

When each block is processed in the backward direction, the slacks are computed for the sources for each input, one at a time. For example, block D has a delay of three and a required arrival time at its output of ten; therefore, the required arrival time at its input is seven (the output required arrival time minus the internal delay). With respect to this block, the output of block C is one unit late, so the slack is minus one. Block H has a delay of five, so the required arrival time at its inputs is five, and the signal feeding from block C is three units late. Three units late (slack = -3) is worse than one unit late, so the program stores the most negative value (-3) at the output of block C. The slack value stored at the output of the block corresponds to the slack value for the worst path going though that block.

Note that each block is processed once forward and once backward; therefore, the process runs in a time proportional to the number of blocks. Note also, that in addition to calculating the arrival time for the worst path to a block, we have also calculated a measure of how bad that path is.

From Figure 8, one can observe a funneling of negative slack values through block B. If B could be replaced by a circuit having a smaller delay, all the negative slacks could be eliminated without changing the functional design at all. Clearly, there are also alternative techniques for resolving this timing problem.

In addition, notice that there are many blocks with positive slacks. These blocks could use circuits with increased delay and still be within the timing constraints [AG77, RU77].

If we are looking for early signals caused by short paths, the above procedures are modified slightly. The arrival time of a block is based on the minimum arrival time of any input to the block (rather than the maximum). The definition of slack is set up so that once again a negative slack indicates a problem; that is, the signal is too early. The reason for changing this definition was based on a human factor consideration, it is easier to spot a negative number in a column of positive numbers than to spot a positive number in a column of negative numbers, and it is more consistent to say that a negative slack always indicates a problem.

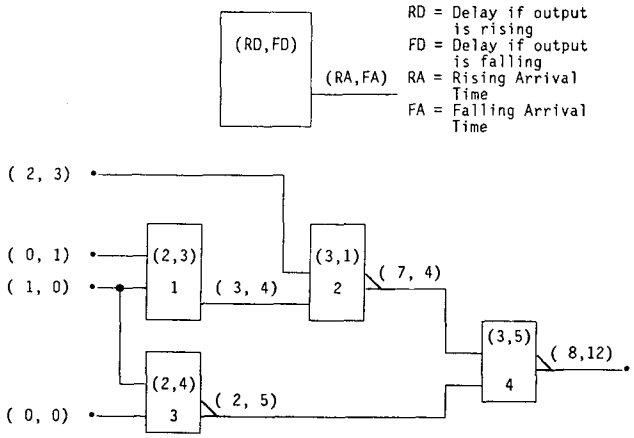


Figure 9 - Using Rising and Falling Delays

Figure 9 shows how the Timing Analysis program just described can also handle rising and falling delays and the inverting properties of a block without requiring the "two-rail" model mentioned earlier in the discussion of PERT. As the key at the top of Figure 9 indicates, the first number inside the block is the delay if the output is rising; the second number is the delay if the output is falling; and the arrival times are kept in a two-tuple with the rising arrival time in the first position and the falling arrival time in the second position. The fact that block 2 inverts is indicated by the triangular wedge symbol at its output. This means that the rising output will be the greater of the two falling inputs (at times three and four) plus the block delay (three), so that the rising arrival time will be seven. The falling output will be the greater of the two rising inputs (at times two and three) plus the delay (one), so the falling arrival time will be four. The same calculation can be done for each of these blocks so that we can compute the rising and falling primary output arrival time of eight and twelve, respectively. Note that if ten were the required arrival time for both the rising and falling arrival times, one would be early and the other

late. For this reason the TA diagnostics distinguish between block output rising and falling arrival times, not just between blocks.

The calculation of arrival times also includes statistics. As indicated earlier, each block delay actually consists of two values - a mean delay and a delay standard deviation (sigma). The calculation of a rising or falling arrival time at each block of the model involves the calculation of a mean arrival time and an arrival time sigma. Mean arrival times are computed by simply adding block mean delays to previous mean arrival times. The arrival-time sigmas are computed by applying standard convolutions.

The arrival time actually stored at a block output is the result of propagating one of the input arrival times through the block. When the TA program is looking for long paths, the one selected is the one having the latest output arrival time. The latest arrival time is the one for which the expression

$\mu + \beta\sigma$ where μ = nominal arrival time
β = confidence level, provided by the designer
σ = arrival time sigma

is the greatest. This expression is used to compute an arrival time whenever one is to be printed or a slack is to be computed. Conclusions about the probability of a timing failure can be drawn from the confidence level under the assumption that the distributions are Gaussian. When the program is looking for short paths, the expression is modified by replacing the plus sign with a minus sign.

THE TIMING ANALYSIS SYSTEM

A complete TA model consists of the combinational logic, the storage elements, and the clocks.
The clocks are defined using two basic time intervals, the **propagate time** and the **compare time** (see Figure 10). The propagate time is defined so that the storage elements, which trigger the origination of data signals based on the times at which the clock arrive, will know at what time and in what time period the propagation should start. The compare time is defined so that the storage element will know when to expect data signals to arrive at its inputs.

Figure 10 also shows two storage elements, A and B, and some combinational logic connecting them into a loop. Clock 1 (C1) gates storage element A, and clock 2 gates storage element B. The propagate and compare times are shown in the lower half of the figure. The path from A to B and the path back from B to A can be checked in the same Timing Analysis run without requiring a special test for each of the separate loops; the only requirement is a specification of the clocks in a consistent manner so that the signals resulting from the propagate time from C1 will be compared with the compare time of C2, and the signals resulting from the propagate time of C2 will be compared with the compare time for C1.

To facilitate the rapid analysis of the TA results, an interactive analysis program is included in the TA system (see Figure 11):

Timing Analysis Results Analyzer [HI82b]

The chief elements of the TARA (TA Results Analyzer) portion of TA are the use of a non-APL software paging system, APL (for rapid algorithmic development), a full screen manager, and a 618 storage tube with GRAPHPAK used to draw related figures.

The in-core formats from the Timing Analysis program are placed in a software paging space for later access. This process is done while the regular batch job is running, and removes the need to produce a detailed summary of the times, come-froms, and go-tos for each block (this can be quite extensive for large block graphs).

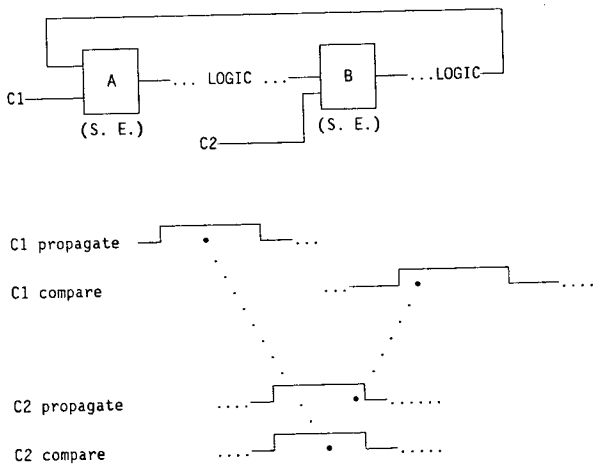


Figure 10 - Clock Descriptions

The data (stored earlier) is then accessed by an auxiliary processor written to interface with the software paging package and reformat the accessed data into APL variable formats.

The extracted data is interrogated by APL programs which do the more mundane functions of name or index searching, reformatting of numbers to be displayed and fielding of user requests.

The blocks extracted are displayed using the same techniques used by the report generation algorithms within Timing Analysis so that the results may be compared directly to any listings that have been obtained.

Completeness

As with the other techniques just described, TA program is a complete system in the sense that all paths are potential candidates for identification as problem paths. This follows from the TA approach which views storage elements as special functions and ignores the actual Boolean properties of other logic blocks with the exception of their inverting properties.

There may be paths, however, which are identified as problem paths by TA which can never be sensitized by logic or which meet special timing criteria at the path outputs. One such example was shown in the discussion of the SCALD Timing Verifier. Another example of a special timing criteria is represented by the so-called two-cycle path in which the path from one storage element to another exceeds one cycle time but logic always ensures that two machine cycles are available for data to propagate through the path. By inserting **delay modifiers** at the inputs to

selected blocks of the block model, one can effectively block the propagation of arrival times or adjust arrival times by fixed constants (e.g., minus one cycle for the two cycle path). These delay modifiers are placed in the model based upon an understanding of the logic. The modifications implied are handled during the delay extraction portion of the basic block algorithm of TA.

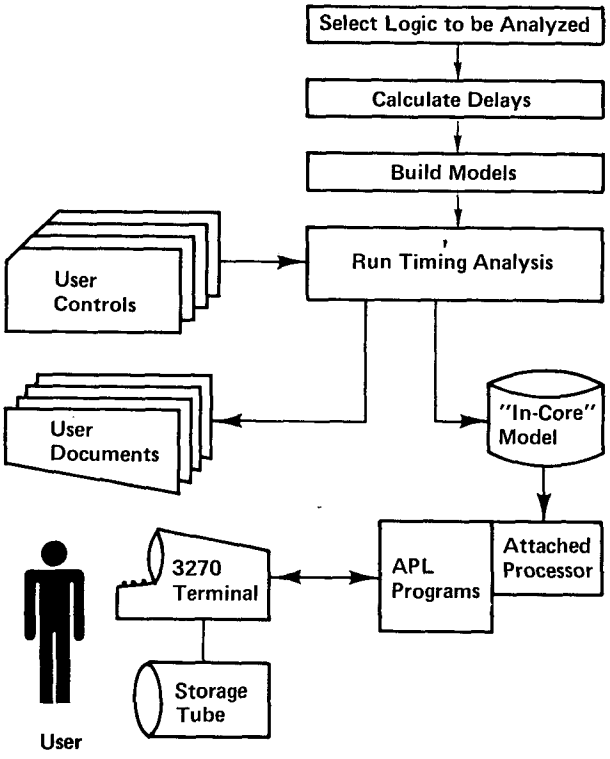


Figure 11 - Timing Analysis System

In certain cases, it may not be possible to place delay modifiers in a model without having an undesirable effect on other paths. For example, it may not be possible to locate any block pin where the placement of a delay modifier will not adversely effect a non-two-cycle path. It is then necessary to make one or more extra runs in which delay modifiers are set to different values.

Due to the use of a software paging system, it is possible to handle very large (over 100,000 circuits) models in a single run. If, for reasons dictated by machine availability, a model is too large to process in a single run, it is broken up into more manageable pieces and the values computed at the primary outputs (inputs) of one piece are saved and automatically fed to the primary inputs (outputs) of the other pieces to which it is connected. Thus, the total analysis is the same as if the whole model were processed at once.

TIMING ANALYSIS APPLICABILITY

There are three main concerns in evaluating the applicability of the timing analysis method. These were described in detail in [HI82a] and can be summarized as:

- The accuracy of the answers are only as good as the accuracy of the delays calculated for each block, and upon the assumption that the delays can be combined as one would for Gaussian distributions.
- Analyzing a path statistically is not the same as analyzing a machine statistically. What we really want is the probability that the machine produced will be free of timing problems. What we have calculated indicates that each path will be within the desired bounds, with a certain probability.
- Delay modifiers can be used, and abused. They are necessary, as was indicated earlier, to cut paths that will never be active or to adjust delays for "two-cycle' paths. However, inserting them is a manual process, and the fewer times they have to be inserted, the more likely the design will be truly free of timing problems.

Timing Analysis is a block-oriented algorithm which effectively checks all paths by computing output arrival times and provides a measure of on-timeness by computing slacks for each block. The TA program does its calculations using statistical approximations and provides methods for dealing with logic with special timing problems.

IV SUMMARY AND CONCLUSIONS

Timing Verification, the validation of path delays and the checking of clock pulse widths, can be accomplished by path oriented and block oriented techniques. The path oriented techniques, which enumerate paths, tend to run considerably longer than the block oriented or non-enumerative path methods, due to the explosive growth in the number of paths as the number of blocks in the design increases.

Several of the previous techniques have been described along with brief illustrations showing how they would apply to a single problem.

Additionally, the Timing Analysis approach has been described in some detail as a method which not only utilizes the less time consuming block oriented method for analysis, but also provides a new mechanism (slack calculation) for highlighting all problems within a design.

V ACKNOWLEDGMENTS

Many people have contributed to the Timing Analysis project since its inception in 1973. In the early formulation of the ideas, R. Bahnsen, W. Donath and G. L. Smith were major contributors. In helping to design a working program, D. D. Cheng, G. L. Smith and H. Ofek contributed heavily. It is not possible to name all the many others on the 3081 project and in the design automation (EDS), and technology areas who contributed to TA.

VI REFERENCES

[AG77] B. J. Agule, J. D. Lesser, A. E. Ruehli, and P. K. Wolff, Sr., "An Experimental System for Power/Timing Optimization of LSI Chips," Proceedings of the 14th Design Automation Conference, New Orleans, (pp 147-152), 1977.

[DO81] W. E. Donath and R. B. Hitchcock, Sr., "Method for determining the characteristics of a logic block graph diagram to provide an indication of path delays between the blocks," U. S. Patent No. 4,263,651, April 1981.

[HI82a] R. B. Hitchcock, G. L. Smith, and D. D. Cheng, "Timing Analysis of Computer Hardware," IBM Journal of Research and Development, Vol. 26, No. 1, (pp 100-105), 1982.

[HI82b] R. B. Hitchcock, B. L. Keller, E. Kellerman, J. F. Schroeder, and A. M. Stankosky, "Timing Analysis Results Analyzer", IBM Technical Disclosure Bulletin, Vol. 24, No. 8, (p 4229), 1982.

[KA81] R. Kamikawai, M. Yamada, T. Chiba, K. Furumaya and Y. Tsuchiya, "A Critical Path Delay Check System," Proceedings of the 18th Design Automation Conference, Opryland, (pp 118-123), 1981.

[KI66] T. I. Kirkpatrick and N. R. Clark, "PERT as an Aid to Logic Design," IBM Journal of Research and Development, Vol 10, No. 2, (pp 135-141), March 1966.

[MC78] T. M. McWilliams and L. C. Widdoes Jr., "SCALD: Structured Computer-Aided Logic Design," Proceedings of the 15th Design Automation Conference, Las Vegas, (pp 271-277), 1978.

[MC80] T. M. McWilliams, "Verification of Timing Constraints on Large Digital Systems," Proceedings of the 17th Design Automation Conference, Minneapolis, (pp 139-147), 1980.

[PI73] D. J. Pilling, and H. B. Sun, "Computer - Aided Prediction of Delays in LSI Logic Systems," Proceedings of the 10th ACM/IEEE Design Automation Workshop, Portland, Oregon (pp 182-186), 1973.

[RU77] A. E. Ruehli, P. K. Wolff, Sr., and G. Goertzel, "Analytical Power/Timing Optimization Technique for Digital Systems," Proceedings of the 14th Design Automation Conference, New Orleans, (pp 142-146), 1977.

[SA81] T. Sasaki, A. Yamada, T. Aoyama, K Hasegawa, S. Kato and S. Sato, "Hierarchical Design Verification for Large Digital Systems," Proceedings of the 18th design Automation Conference, Nashville, (pp 105-112), 1981.

[WO78] M. A. Wold, "Design Verification and Performance Analysis," Proceedings of the 15th Design Automation Conference, Las Vegas, (pp 264-270), 1978.