

Delay Test: The Next Frontier for LSSD Test Systems

by

B. Könemann

J. Barlow, P. Chang,
R. Gabrielson, C. Goertz,
B. Keller, K. McCauley,
J. Tischer

V. Iyengar, B. Rosen

T. Williams

IBM Poughkeepsie

IBM Endicott

IBM Yorktown

IBM Boulder

Abstract

Delay testing, as opposed to static testing introduces the parameter of time as a new variable. Time impacts the way defects manifest themselves and are modeled as faults, as well as how defect sizes and system timing statistics interact with tester timing constraints in the detection of causes for dynamic system malfunctions. This paper briefly discusses some of the issues that had to be addressed in the development of a comprehensive system for delay testing in a Level Sensitive Scan Design environment.

1. Introduction and Background

Level Sensitive Scan Design, LSSD, combines the concept of Scan Design with a race-free clocking discipline. LSSD testing, despite its traditional focus on static defects, has been successful because sophisticated automatic test generation tools achieve virtually 100% stuck-at fault coverage. Experience indicates, however, that there remains a painful residue of dynamic defects that slip through the static test process.

The basic concepts of Scan Design, including proposals for dynamic testing, were formalized the late '60s and early '70s (e.g., [1,2,3]). The so called 2-cycle test introduced in [4], for example, suggests to gate two consecutive system clock pulses (to first release, then capture signal transitions) into the logic under test after the network has been initialized by Scan. Any failure to capture the proper response state would indicate a dynamic defect in the network.

The original LSSD test system also included delay test generation facilities ([6,7]). Delay faults were treated as conditional stuck-at faults activated by transitions. Each test sensitized a specific path type and was individually timed to allow for characterizing any delay faults as accurately as possible. Because of this, the method generated too many tests and timing sets to be cost efficient, and was not used in production testing.

The LSSD delay test activities were revived in the early '80s, when a particular delay defect caused larger than normal chip quality problems in a mainframe assembly. A simple method based on converting stuck-at fault tests into 2-cycle tests with pattern-independent timings derived from system timing [8] was adopted this time. These 2-cycle tests, despite being derived without

an explicit delay fault model, proved to be economical and quite effective in detecting a large number of delay defects, and have since become widely used in LSSD based production testing.

This, together with successes reported by others [9], has led to the pursuit of a new LSSD test system for explicit delay fault test generation, which builds on the experiences gained thus far.

2. Delay Defects and LSSD Test Formats

Figure 1 shows a simple example of a so called LSSD "Double Latch Design".

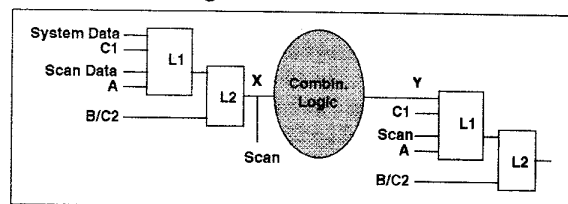


Figure 1: LSSD Double Latch Design

The basic storage elements of LSSD are Shift Register Latches, SRLs, consisting of L1 and L2 latches. The L1 has a System Data port controlled by a System C1 clock and Scan Data port controlled by a Shift A clock. The L2 has a single data port controlled by B/C2 which is the logical OR of a Shift B clock and a System C2 clock. For scanning, the latches are serially connected and operated as a shift register under the control of the A and B clock signals. For system operation the C1 and C2 clocks are used to propagate System Data through the L1/L2 latch pairs and the combinational logic in between. A typical LSSD test for defects in the combinational logic of Figure 1 entails the following sequence of events:

- Establish the Scan State and load (A/B clocks) the SRLs with test stimulus data.
- Capture test responses into L1s with C1.
- Establish the Scan State and unload (A/B clocks) the responses.

The crucial idea for a dynamic LSSD test is to apply the **release** of signal transitions and the subsequent **capture** of responses at system speed. The input events corresponding to release and capture for testing defects in

the combinational logic of Figure 1 are the leading edge of the B/C2 clock (release) and the trailing edge of the C1 clock (capture). If it is possible to apply these two events within the test at or close to system speed, then dynamic defects can be detected.

3. A Fault Model for Dynamic Faults

The choice of a fault model is central to any test system. The experience with the original LSSD delay test method strongly suggests to carefully control the number of generated tests and timing sets. This concern has had a significant impact on the technical decisions in the test system development.

The **path-oriented** fault view argues that defects can accumulate along paths and that the fault model should be associated with paths. Large networks have orders of magnitude more paths than gates. This makes a path-oriented model unattractive from a numerical point of view. A **node-oriented** fault view, by contrast, attaches the delay faults to the gate inputs and outputs in the network. While being potentially less accurate, the node-oriented model limits the number of faults just like the stuck-at fault model does.

An overriding practical argument for using a node-oriented fault model is that this greatly simplifies the extension of the traditional node-oriented DC test generation/fault simulation tools to cover delay defects. Furthermore, even with a node-oriented fault model, many paths are sensitized by accident, giving at least partial path fault coverage.

A second issue to be resolved is which conditions to impose on fault activation and propagation. The most restrictive method is based on a **single-path-sensitization** paradigm. Both, the activating signal transition and the fault propagating transition, are restricted to a single path and gating signals must be stable and glitch-free.

Robust tests [10,11] make the detection of delay faults independent of the timing (good or defective) of the gating signals, but without requiring that the propagation path exclusively dominates the timing.

A **non-robust** test has even fewer constraints. A delay fault is activated by an appropriate transition at the fault site and exhibits temporary stuck-at fault behavior after the transition has taken place. A temporary stuck-at 0 fault is called a **slow-to-rise** (str) fault, while the temporary stuck-at 1 fault is called a **slow-to-fall** (stf) fault. The surrogate stuck-at fault effect is propagated to the capture point by the final state of the gating signals after any transition. The initial values of the transition are only important for fault activation, but immaterial for fault propagation.

The temporary stuck-at fault persists for a limited amount of time, the **size** of the fault. For detection, the fault must be of large enough size to persist until captured. If the fault is unilaterally presumed to be large enough for detection it is called a **transition fault**, and no timing information is needed for test generation/fault simulation. The **small delay fault** model, by contrast takes the size of detectable faults explicitly into account.

The additional degree of freedom in non-robust propagation makes test generation easier and adds network activity compared to more restrictive tests. It is envisioned that the freedom of choosing the initial values more freely can be utilized to tune the test generation algorithm towards selecting more timing-critical paths and enhance the ability to detect timing-critical path delay faults. Furthermore, there seem to be no quantitative hardware studies available which confirm that a significant amount of defect masking occurs in non-robust tests.

- o **Based on these considerations, it was decided not to use the more complicated and more expensive single-path-sensitization or robust fault models, but to implement a node-oriented non-robust transition fault model with an optional small delay fault model.**

NOTE:

It is understood that practical hardware experience could eventually lead to changing the fault model. However, at this time there is no hard evidence available that would make this likely.

4. System Timing, Defect Sizes, and Tester Timing

By their nature, dynamic defects are dependent on timing. This chapter attempts to illustrate how the characteristics of system timing distribution, defect size distribution, tester timing capabilities, and sensitized path lengths interact in the testing of dynamic defects.

4.1 System Timing Distribution

Realistic manufacturing processes and device operating conditions are subject to variations which can be modelled by statistical distributions. The timing of electronic devices is one such statistically varying parameter.

4.2 Defect Size Distribution

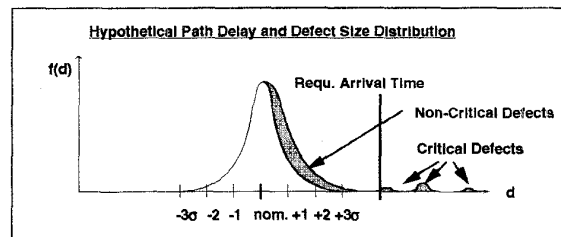


Figure 2: Path Delay Distribution with Defects

The fact that delays vary statistically makes an exact definition of defects difficult, because it implies that there exist "good" process variations which result in an acceptable "good" delay distribution. Only deviations from this distribution due to "other" local or global process mishaps are called defects. We assume that somehow the susceptibility of a technology to delay

defects can be characterized by a delay **defect size distribution** [12]. The shaded areas in Figure 2 depict how a hypothetical defect size distribution might overlay a path delay distribution.

Figure 2 qualitatively reflects that delay defects can be of a parametric, "continuous" kind (for example, excessive fluctuations in resistor values), or of a more "discrete" kind (for example, due to an open pull-up resistor contact or a partial metal open in layered metal wires [13.]). Defects that cause a malfunction at system speed (indicated by the Required Arrival Time), are called **critical delay defects**.

4.3 Tester Timing Capabilities

The relationship between delay defect detection and tester timing accuracy is illustrated for the path from Figure 2. The at-speed portion of the test consists of a release clock pulse followed by a capture clock pulse. The tester has to apply these two pulses from primary inputs with some programmed timing offset between the clock edges. Most testers allow for placing several clock pulses into the same tester cycle, and tester cycle time is not the limiting factor for the test. The more important parameter instead is how accurately the capture edge can actually be positioned relative to the release edge within a tester cycle.

Figure 3 illustrates the effect of tester inaccuracies by showing a range for the actual latching clock edge timing on a tester relative to the path/defect delay distribution of Figure 2.

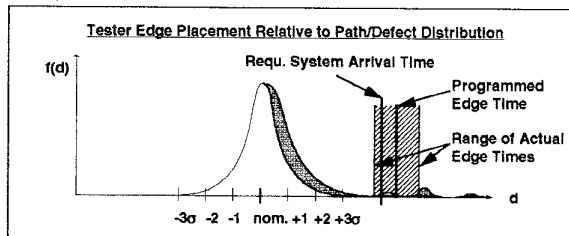


Figure 3: Relative Placement of Tester Edge

The programmed capture edge placement shown is close to the functional timing for the path. Such timing will be referred to as **pseudo-functional** timing. The decision of where to exactly place the programmed edge depends on trade-offs between yield and defect detection.

The tester edge timings are generally not kept in the pattern data buffers, but a limited number of timing sets are set up ahead of time. This suggests that from an economical point of view, tests with a limited set of common **pattern-independent** timings for groups of tests are the preferred mode of operation.

4.4 The Effect of Sensitized Path Lengths

Figure 4 illustrates the consequences of sensitizing a different, shorter path. The associated path delay distribution is narrower and shifted to the left. The defect size distribution is assumed to be the similar to the one from Figure 2 and is overlaid on the new path delay

distribution. With a pattern-independent timing approach, it is not possible to optimize the timing for each test and it must be assumed that the programmed edge time must accommodate longer delay paths. Hence, the edge time remains the same pseudo-functional time as before.

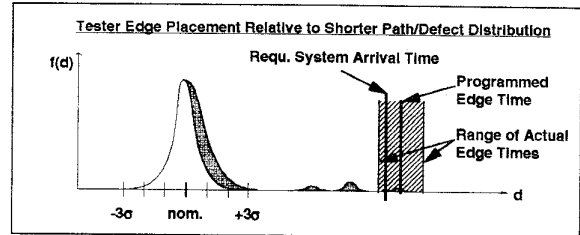


Figure 4: Relative Placement of Tester Edge for Shorter Path

Shifting the delay distribution to the left moves some critical defects outside of the tester capture edge range and they escape detection. This suggests that for best possible critical fault detection in a pattern-independent pseudo-functional timing method the test generation process should favor the sensitization of the **least-slack** (e.g., longer) path through each fault site.

To separate small delay defects from the "good" process variations, by contrast, would require to reduce the width of the "hump" in the "good" process distribution by sensitizing shorter paths and to use a very accurate tester with per pattern timing.

o Because of the practical constraints of tester availability, cost and throughput, and an emphasis on the detection of critical delay defects, the **pattern-independent pseudo-functional timing approach** has been adopted for the LSSD delay test system.

5. Outline of a Dynamic LSSD Test System

In the following, some elements of the LSSD delay test system are outlined and discussed in the context of the preceding motivation.

Figure 5 depicts the high-level architecture of the system.

5.1 Model Build

The network under test is described as a netlist of cell types (so called **N-Blocks**) from a technology library. Each N-Block has an associated logic expansion in terms of primitive **E-Block** elements known to the test generation system. Model Build generates both, an expanded unit/zero delay **E-Block model** without realistic timing information and an **N-Block model**. The two models are linked by cross-references. Timing information is provided in the form of delay data for N-Block input/output delays, wire delays, set-up and hold times for latches and memory arrays, and so on.

Information about the Tester Constraints (number of pulse generators, number of pins, maximum edges per

tester cycle, pin loading, accuracy, etc.) is also loaded and reformatted for use by the tools.

Separate **fault lists** for stuck-at and transition faults are generated for the nodes in the E-Block model. The fault lists contain appropriate slots for each fault to record the fault detection status during and after test generation.

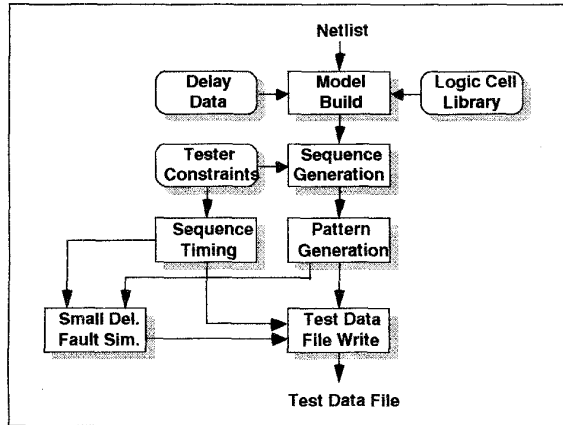


Figure 5: Software System Architecture to Support LSSD Delay Test

5.2 Clock Sequence Generation

The pattern-independent timing concept is implemented at the level of groups of tests characterized by so called **clock sequences**. Each LSSD test consists of many patterns: an LSSD scan string load/unload and some additional specific events. The clock sequences can be viewed as pre-defined skeletons of events and event types (for example, which clocks are pulsed, which logic value certain control signals should have, or which PI data signals are allowed to change) which still must be filled in with specific data values to produce an actual test. Selecting the proper clock sequences is crucial for the detection of delay faults.

The sequence generation tool, PROTEUS, performs a topological analysis of the E-Block model to determine the "best" release and capture events. Figure 6 shows a high level description of the generic clock sequence format.

Only the events in the **Timed Test Cycle** are applied with dynamic timing. Loading/unloading the scan strings and initializing the network are assumed to be static. The choice of release, gating and capture events determines where and how in the network signal transitions are released, propagated and observed. To cover all delay defects in a sequential LSSD network requires a multiplicity of different clock sequences.

It is beyond the scope of this paper to fully address the complexity of the clock sequence generation task. A few examples for the simple network in Figure 1 may serve to illustrate some of this complexity.

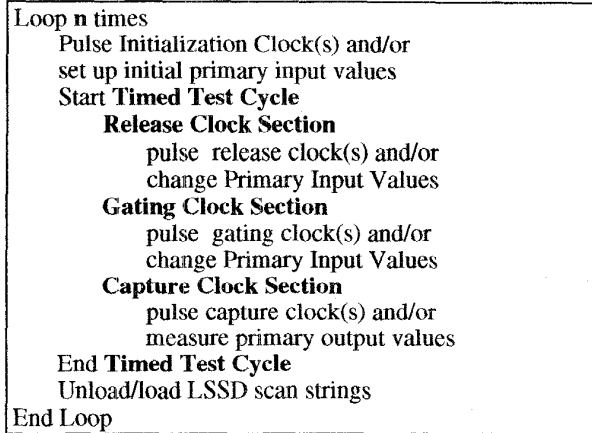


Figure 6: Generic Delay Test Clock Sequence Format

Faults in the Logic Between L2s and L1s

Most transition faults in simple double-latch designs are in this category. The transitions for activating the faults are released from the output of the L2s. The release occurs with the leading edge of a B/C2 clock pulse, provided that the L2 and the L1 feeding it have been initialized to opposite values. The **initialization** of the L1 to the opposite value can be accomplished in one of two ways:

- Pulse the Shift A clock to load scan data
- Pulse the C1 clock to load system data

The first option ("**Skewed Load**") obtains the value directly from the scan path. It's drawback is that the values in adjacent latches in the scan string are not truly independent [17] making it impossible to load certain value pairs. The second option, by contrast, requires to get the appropriate values from the preceding level of combinational logic, leading to sequential test generation. This adversely affects diagnoseability by involving more logic and propagating values through latches, making it impossible to use a combinational simulator for diagnostics as described in [18]. PROTEUS, thus, preferentially uses the Skewed Load option.

There are also several options for the **release** of transitions from L2s, e.g.:

- Stay in Scan State and pulse the Shift B clock
- Leave the Scan State and pulse the C2 clock

The first option is preferred for diagnostics, since it only uses the already verified Scan State and avoids any complicated gating that might be used in the C2 clock for system purposes. Thus, if diagnostics are important, PROTEUS will preferentially use the Shift B clock. User controls are available to instruct PROTEUS to use the often more accurate C2 clock rather than the B clock in cases where test accuracy is more important than micro net diagnostics.

To activate different clocks sometimes requires to switch some gating signals. Such actions, if required after

release but before transitions can be captured at Primary Outputs (POs) or L1s, are included in the **Gating** clock section.

Once a transition is released and the required gating is set up, the **Capture** of the propagated responses can occur by latching the responses into L1s with the trailing edge of a C1 clock pulse.

Faults Between L1s and L2s

Although in double latch designs there is very little logic between the L1s and the L2s, it is necessary to check for delay defects on such nets as well. For these faults, the roles of the L1s and L2s are reversed, with the L1s becoming the release latches and the L2s becoming the capture latches.

Faults on Clock Lines and in Latches

In high performance computer systems, a sizeable portion of the logic (e.g., 20% or more) is associated with the clock distribution and the latches. A comprehensive delay fault system cannot ignore delay faults in this part of the logic. PROTEUS analyzes the association between clock distributions, latches, and the logic feeding those latches, to generate possible sequences for detecting defects that could lead to clock timing problems.

Minimum pulse width violations are tested by "racing" the leading edge of a clock (release) against the trailing edge of the same clock pulse (capture). These sequences target defects that delay the clock turn-on or slow down the feedback loop inside the latches.

Set-up time violations are tested by "racing" data signal changes at latch data inputs against the trailing edge of the latching clock. This type of sequence is aimed at defects in the combinational logic, the data ports of the receiving latches, or the turn-on of release clocks. The sequences described above for testing delay faults in the logic between latches are of this type.

Hold time violations are tested by "racing" the trailing edge of a clock against a subsequent data change at an associated latch data input; i.e., the trailing edge is the release event while the leading edge of the clock causing the data change becomes the capture event. Hold time type sequences test for delays in clock turn-off and for short path problems.

There are additional details to be considered for **Clock Gating** and **Clock Shaping** circuit structures. PROTEUS does recognize some of these structures and includes their special requirements into the clock sequences. PROTEUS also has to understand the operation and clocking of **embedded memory arrays** and, if needed, include memory read/write operations into the timed test cycle construction.

General Considerations

PROTEUS analyzes event interdependencies. Events are **independent**, if the final network state does not depend on the order in which these events occur. Mutually independent events can be merged into a single pattern to improve the test effectiveness. Release, optional gating,

and capture events are by nature **dependent** and are always placed into adjacent patterns which are specially marked as the timed test sections. All other events (including the LSSD scan chain load/unload, any additional network initialization, etc.) are considered static.

Networks with complex clocking may require several dozen sequences.

5.3 Pattern Generation/Fault Simulation

The objective of the test generation/fault simulation tools in TARP (Testability Analysis for Random Patterns) is to create tests for all faults in the stuck-at and transition fault lists. Test generation/fault simulation proceeds on a clock sequence by clock sequence basis and is orchestrated by PROTEUS.

Test Generation

The test generator is based on a Weighted Random Pattern (WRP) concept similar to the one described in [14]. The transition faults are presumed to persist for the duration of the timed test cycle, that is from release until after any capture events have occurred. Under this assumption, no timing information is needed during test pattern generation/fault simulation.

PROTEUS orders the clock sequences such that easy to diagnose sequences with the potential of detecting the most faults are used first. The algorithmic, fault-oriented test generator is invoked only **after** an initial pass of **flat random** patterns (same probability for 0s and 1s) has been applied and fault simulated. The fault simulator (see below) in this stage produces a list of **random pattern resistant** fault clusters from which the test generator selects initial target faults (stuck-at and/or transition faults).

Transition fault tests involve at least two test patterns. First, the faulty node is set to the appropriate initial state (0 for slow-to-rise, 1 for slow-to-fall). Then, the faulty node must make a transition to the opposite value. This injects the associated temporary stuck-at fault at the faulty node. The second pattern must also begin to propagate the temporary stuck-at fault to a capture point for detection. In complicated situations, additional patterns are required for initializing the network, for gating events, and/or for propagating the captured response to observe points.

These sequential patterns are pieced together by multiple calls to a so called **embryonic** pattern generation utility. In a simple case the pattern generator is called twice. For the first pattern, SRL and/or PI values are requested to set the proper initial state at the fault location. For the second pattern, the test generator is asked to return a test for the associated temporary stuck-at fault. In sequences using LSSD shifting, the SRL values for the second pattern are constrained to be obtainable from the first pattern by a half-shift of the LSSD scan strings. If that fails, the test generator will attempt to utilize system clocks.

The PI and/or SRL values returned by the test generator are translated into weights as described in [14]. Where

possible, the tests for additional target faults are merged. With these weights, groups of 32 weighted random patterns are generated and sent to the fault simulator. PROTEUS sets a minimum threshold of how many faults should be detected with each group. If the threshold is not reached, the current weight set is abandoned and the test generator is invoked again with new target faults.

Complex chips require several hundred different weight sets to achieve the desired DC and AC test coverage objectives.

Transition Fault Simulation

The transition fault simulator is an extension of a high-speed, compiled, parallel pattern stuck-at fault simulator ([15]) which looks for appropriate good machine signal transitions at the fault site. If a transition is found, the corresponding temporary stuck-at fault is injected and simulated for the subsequent fault propagation patterns. Fault detection must occur by the section containing the capture event(s) and is, within this constraint, determined exactly like in stuck-at fault simulation. Transition fault detection is marked in the respective fault status slot in the delay fault list.

As mentioned above, flat random patterns are fault simulated at the beginning of each sequence. During that phase, the simulator keeps statistics of the switching activity at each circuit node. Nodes with less than normal activity are identified as "choke points" which indicate random pattern resistant logic. This information is used for selecting the initial target faults in the constructive test generation phase.

General Considerations

Weighted random pattern testing requires special test hardware (cf. [14]) and can produce extremely long test sets. To be more flexible, an option has been added to extract only the fault-detecting tests from the full WRP test sets and to expand these tests into stored stimulus/response data for use on test equipment without WRP hardware.

For use on testers with WRP hardware, only the weights, number of tests per weight, and resulting signatures are needed. To improve tester throughput, an option exists to only apply the subset of fault-detecting tests. On the WRP testers it is not necessary to expand the patterns into stored stimulus/response data to exercise this option.

5.4 Sequence Timing (CHRONOS)

Clock sequence generation, weight generation, and transition fault simulation do not take the circuit timing into account. The timing aspect is handled separately by CHRONOS. CHRONOS operates on a clock sequence by clock sequence basis and attempts to find an "optimal" timing for the dynamic events in a given clock sequence. The method is **pattern-independent** in that all tests with a common clock sequence can be executed with the same timing set on the tester. However, it may be necessary to change the timing set between sequences, or to repeat certain tests with slightly different timings.

The timing trace algorithms used in CHRONOS are very similar to the algorithms used for static timing analysis ([16]). There are, however, some significant differences. Timing analysis tools typically use some clock and data pin timing targets as input and calculate the logic "slack" for this given timing. CHRONOS, by contrast, uses the logic delays and clock sequences as input, and generates the tester timing for clock and data pins as output.

Delay Data for CHRONOS

CHRONOS essentially uses the same circuit delay values that are used for functional timing analysis. Each physical N-Block has associated delay equations representing the intrinsic delays of that N-Block as well as its sensitivity to input and output loading conditions. Delay equations also are provided for the different interconnect net types. After final placement and wiring, the delay equations are evaluated with the correct loading values and physical net configurations. CHRONOS reads the resulting N-Block and net delay values and stores them in tables associated with the N-Block level model.

Both, nominal values and statistical variances (sigma), are provided for the delays. Storage elements have set-up times, hold times, and minimum clock pulse widths specified in addition to propagation delays. Furthermore, a global technology minimum is specified for the clock pulse width on all clock nets.

Timing Trace Controls

Functional timing analysis tools allow the users to cut certain nets (snip sets) or to ignore the timing on some nets (don't cares) based on the user's understanding of the functional operating conditions. CHRONOS, instead, uses logic control values and stable signals to guide the timing trace. During test generation, the users can explicitly "**line-hold**" certain PIs and/or SRLs at fixed values. Other PIs and/or SRLs may implicitly hold their values by virtue of not being exercised in the timed cycle of the currently processed clock sequence. CHRONOS simulates the effect of any line-holds and initialization patterns, and analyzes the timed cycle events to determine which nets in the circuit are guaranteed to be stable or at a known fixed value during the timed cycle of the test. Stable nets are excluded from the timing trace. By the way, latches may qualify as stable data sources even if they are clocked in the timed cycle, as long their data inputs don't change and the latches are glitch-free (as indicated by a special flag in the latch logic model).

Line-holds to a known fixed value are a means for the user to improve the achievable Timing. For example, if one input of a logic gate is line-held to the dominating value for this gate type, the gate output is forced to a fixed value, blocking the propagation of timing data through the gate. This technique can be used, for example, to block excessively long maintenance signal paths which otherwise would dominate the timing of functional paths. By changing the line-holds between

clock sequences, different path types can be "activated" as seen fit for better test timing.

Minimum Pulse Width and Propagation of Clock Signals

Before constraints arising from logic delays are calculated, CHRONOS analyzes the clock timing. Each clock pin on an N-Block containing a storage element has a minimum required clock pulse width specified in the delay table. CHRONOS propagates these pulse width "requirements" backward to the associated clock PIs, subtracting propagation delays and uncertainties from both clock edges along the way. If at any intermediate net the derived pulse width request falls below the technology minimum, it is increased to that minimum value. At clock fan-outs, only the largest pulse width request from each fan-out branch is propagated further backwards. The derived value at each clock PI indicates the minimum width of a clock pulse applied to the PI which would still guarantee minimum pulse width on all clock nets and receiving latches. This value is compared to the minimum pulse width of the tester, and the larger of the two values is used as the final minimum pulse width constraint on the clock PI: that is, the timing offset between the leading edge and the trailing edge of a pulse on the clock PI must always be larger than the calculated constraint value.

CHRONOS then determines the forward delays from the clock PIs to each storage element clock input by forward tracing from the PIs. All clock gates are assumed to be "open" except for those blocked by lineholds. The local leading and trailing edge arrival times at the storage elements clock pins are stored for later reference.

Logic Signal Propagation

Logic signal edge propagation delays are calculated by a forward trace from the signal sources (PIs, latches) to observe points (latches, POs). For latches, the trace begins at the clock input(s). Transitions at latch outputs are always generated off the leading edge of a clock pulse. The results are stored separately for each source/observe point pair.

"Cause-Effect" Mapping Rules

CHRONOS times only the events in the timed test cycle for each clock sequence. Events can be clock edges, PI stimulus changes, or PO measures. PROTEUS assigns the events into an ordered sequence of sections (see Figure 6): a release section followed by a gating section followed by a capture section. Each section may contain several independent events. Test generation and transition fault simulation assume that any network activity caused by the events in one section has completed before the next section starts. CHRONOS has to translate this discrete event ordering into timing constraints.

The basic rule used by CHRONOS is that all "cause-effect" relationships between the events must be maintained. For example, in a set-up time type test, the data input to a latch is expected to have stabilized at the new value prior to the latching edge of the capture clock pulse. CHRONOS must assure that the relative timing

offset at the network PIs between the event causing the data change and the capture clock edge does not result in violating the set-up time constraint on the latch. Conversely, in a hold time type test the receiving latch clock is pulsed in a pattern preceding the data change, and the relative offset between the receiving clock edge and the data change must not result in violating the hold time constraint on the latch.

The clock sequences are constructed such that all edge-to-edge timing constraints produced by CHRONOS are one-sided (i.e., a minimum separation between the edges, but never a maximum separation). This makes it possible to translate the raw constraints into programmed edge timings for a variety of target testers.

Timing Derivation

The timing derivation step determines proposed tester pin timings for the events in the timed cycle of the currently processed clock sequence. This is done using the "cause-effect" ordering rules in conjunction with the clock and signal edge propagation delays calculated earlier on. A set-up time type rule, for example, generates a forward constraint (minimum offset) between the leading edge of a release clocks (or a data PI stimulus change) and the trailing edge of a capture clock (or a PO strobe), while a hold time type rule generates a reverse constraint between the trailing edge of the "release" clock and the subsequent "capture" event(s) causing a data change at a latch input.

Pin timing constraints are derived by combining the pre-calculated clock and logic delay values. This may be illustrated, for example, by a set-up time test for logic between latches in an LSSD double latch design (see Figure 1). The leading edge of a B/C2 clock pulse is the "release" event, and the trailing edge of a C1 clock pulse is the capture event. The objective is to determine the minimum allowable programmed offset between these clock edges as applied by the tester to the network PIs. Let the programmed time for the leading edge of the release clock at the PI be the reference point. The clock edge delay table contains the local arrival times (t_{rel}) of the leading clock edge at the L2 latch elements. The local arrival time (t_{data}) of the released transitions at the receiving L1 latch data ports is obtained by adding the associated entries in the logic delay table to the local release clock arrival times. Typically, an L1 receives values from more than one L2. For a **set-up** time test only the **latest** of multiple arrival times is recorded. The next step is to add the latch set-up time (t_{setup}) to determine the earliest time at which the trailing edge of the capture clock can arrive. The earliest time the capture clock can be turned off **at the clock PI** is calculated by subtracting the delay (t_{capt}) from the PI to the receiving latch found in the clock delay table. Only the worst case, that is the largest resulting difference, is kept for the clock pin resulting in the constraint:

$$C1\text{"off"} - B/C2\text{"on"} \geq \max(t_{rel} + t_{data} + t_{setup} - t_{capt}).$$

The worst case difference is generally dominated by a subset of the LIs. Other LIs could have been clocked earlier, but the on-product clock distribution delays prevent the signal from getting there at the optimal time without violating the set-up time at other LIs. In other words, using the worst case difference implies a certain **slack** (difference between best possible arrival time and optimal arrival time) at some LIs.

In a general case, the release and capture sections can contain additional events (e.g., PI signal changes for release and PO measures for captures), which generate additional constraints like the one described above. Hold time, clock minimum pulse width, and clock gating considerations generate even more constraints.

To construct the actual tester timing sequence, all constraints have to be met simultaneously. This involves a non-trivial "optimization" process, since it is generally impossible to match all constraint values perfectly: some constraints must be relaxed (i.e., larger than optimal offset) to not violate others. CHRONOS uses the cumulative slack associated with each constraint to guide a greedy timing sequence construction procedure. The constraints with the least cumulative slack are relaxed first, trying to keep the constraints with the largest cumulative slack values unchanged.

Tester Limitations

Test equipment introduces additional limitations and side-effects, which must be taken into account. One side-effect, for example, is that pin loading on a tester is likely different than the target loading on the same pin in the product. The tester loading effects are taken into account when calculating the delay data used by CHRONOS.

CHRONOS also takes into account **minimum pulse width, rise/fall time, resolution**, as well as **jitter and pin-to-pin skew** of the target test equipment by adding appropriate guardbands to the timing constraints when constructing the sequence timing data.

Further, the target testers are characterized as **per-pin** or **shared resource** testers. For per-pin testers, CHRONOS assumes that the pin timing can be set individually for each pin, while this is not possible for shared resource testers.

Different testers may have different degrees of programmability (e.g., how many edges can be placed in single tester cycle). If CHRONOS finds that a particular sequence generated by PROTEUS exceeds the tester limitations, CHRONOS instructs PROTEUS to abolish that sequence. This, however, occurs only in very rare cases.

Model Access in CHRONOS

Because CHRONOS needs timing information, it has to access the secondary N-Block level model and the associated delay data structures.

NOTE:

The described procedure determines sequence timing on the basis of intrinsic product delays and tester characteristics. For typical mainframe chips and sub-assemblies this tends result in **pseudo-functional** timings (that is the derived timing is similar to but not always exactly the same as the system timing).

5.5 Small Delay Fault Simulation (SDFS)

The Small Delay Fault Simulator, SDFS, is an optional tool for determining the **minimum detectable size** of each detected transition fault. SDFS is implemented as an extension to TARP and is called when a transition fault is detected in a particular test.

SDFS propagates both, logic values, and signal transitions times for the respective test, in order to determine a fault size threshold which guarantees detection under the timing conditions derived by CHRONOS for this test. Since the first test detecting a fault may not have sensitized the tightest possible path(s), SDFS suspends the normal fault dropping in the transition fault simulator until a detection size threshold or an upper bound of repeated calculations are reached for the fault, or it can be determined that no better timing is possible for the fault. The best SDFS result for each fault is recorded in the delay fault list.

SDFS needs access to timing data and the logic behavior of the network under test. Thus, SDFS works with both, the secondary N-Block level model and the expanded logic model. Timing data are only available at the N-Block level, and SDFS analysis is therefore restricted to faults on the boundary of N-Blocks. SDFS at this time also does not handle delay faults on clock lines.

SDFS is described in detail in a companion paper [19].

NOTE:

A main objective of the SDFS option is to better understand how well transition fault test generation handles small delay faults. The results are expected to indicate whether and to what extent the path selection heuristics eventually need to be modified or altogether replaced for good small delay fault coverage.

5.6 Test Data File Write

The Test Data File Write step accumulates the generated information and produces a file structure that fully describes the network and the generated tests. The file contains enough information to rebuild the logic model, correlate the logic model with physical information for the package boundaries, and understand the ordering and operation of the LSSD facilities (scan strings, etc.). The test patterns are described with well specified tester-independent op-codes. Op-codes are available for the clock sequence structures, the logic contents of test patterns (that is logic stimulus and response values, or weights and signatures), as well as the relative timing of dynamic events in the clock sequences.

6. Product Quality Control Objectives

The concept of pro-active product quality control is a key ingredient of the IBM manufacturing strategy. Test coverage is a central measure used to gauge the attainable Shipped Product Quality Level (SPQL) before a product component is released to manufacturing. To assure good SPQL, the manufacturing and product design organizations establish minimum test coverage objectives for the product components. The Test Data File contains the coverage data which are audited for meeting the standard. For chips, an initial minimum transition fault coverage objective of 95% has been set in addition to the "traditional" DC stuck-at fault coverage requirement of well over 99%.

7. Sample Results

The AC test generation system is targeted at a wide range of designs from single chips through Multi Chip Modules (MCMs) to cards and boards. Given today's technology outlook, this means that the system design must provide for the capacity and power to process designs containing well over 1M logic gates. Figure 7 summarizes the early system experience with some smaller circuits. Once the system has been fully debugged and stabilized, it will be tested will larger networks.

For the first three chips shown in the table, separate runs were made for DC test generation and for combined DC/AC test generation. The normal procedure is to only perform a combined run. The separate runs were made for comparison reasons. As expected, the AC fault equivalencing is less efficient than the default DC fault equivalencing, leaving the system more AC faults to contend with than DC faults. Also, as expected, the AC test objective increases the number of generated WRP weight sets. The results show that our initial estimate of

Chip	Blks	DC Flts	AC Flts	DC Wgts	DC Cov	AC Wgts	AC Cov
1	42K	118K	155K	91	99.5	163	91.7
2	69K	177k	234K	58	99.7	83	91.9
3	40K	96K	122K	29	98.1	57	92.2
4	165K	-	-	-	98.5	-	93.2
5	-	269K	-	-	98.3	95	93.8

Figure 7: Sample Results

doubling the number of weight sets was quite realistic.

For some chips, the test coverages only include internal logic (Boundary Scan). The final test coverages with peripheral circuits included are higher than those shown in the table.

In these early runs, the AC test coverage falls short of the 95% objective, despite the fact that the least constraining AC fault model (block oriented transition faults) is used. Our conjecture is that significant AC test coverage improvements can still be achieved by improving the sequence generation, weight generation,

and heuristics. The system is too early in its learning curve to predict where the actual limits are.

The early runs on chips 1-3 took between 1 to 3 CPU hours (IBM 3090) for the combined AC/DC run. As with coverage, these are not the final results. The system is being continuously improved and going through significant learning advances. We are cautiously optimistic that our aggressive CPU time targets for processing very large structures can be met, once the system has stabilized. It is clear, however, that adopting a more stringent fault model (e.g., robust faults) would make meeting the CPU time targets very difficult if not impossible.

Early experience with CHRONOS supports the validity of our expectations for achieving pseudo-functional timing. The CHRONOS tester timing for chip 4, for example, is even slightly more aggressive than the system functional timing (not all products drive the technology to its limits). Initial hardware experience seems to indicate that the CHRONOS timing is quite realistic (that is, using a walking strobe technique, chips fail roughly where CHRONOS predicts they should fail). It is noteworthy, that one of the first diagnosed AC test fails has been traced back to a reset clock slow-to-turn-off fault which is detected by a hold time type test. This justifies the emphasis on and investment in testing for clock delay faults in addition to logic delay faults.

8. Summary

Adding delay test capabilities to an LSSD test system is conceptually straightforward. A high-level analysis of some of the crucial parameters shows that significant trade-offs need to be made with regard to fault modelling and pattern timing. The main components and operational objectives of a software support system for LSSD delay test generation have been outlined and discussed.

References

- [1] K.Maling and E.L.Allen, "A Computer Organization and Programming System for Automated Maintenance", IEEE Trans. Electr. Comp., pp. 887-895, 1963
- [2] Kobayashi et al., "Flip-Flop Circuit with FLT Capability", Proc. IECEO Conf., p. 962, 1968
- [3] A.Toth and C.Holt, "Automated Database-Driven Testing", Computer, pp. 13-19, January 1974
- [4] R.J.Preiss, "Fault Test Generation", in "Design Automation of Digital Systems" edited by M.A.Breuer, p. 393, 1972.
- [5] M.A.Breuer, "The Effects of Races, Delays and Delay Faults on Test Generation", IEEE Trans. Comp., Vol. C-23, pp. 1078-1092, 1974.
- [6] E.P.Hsieh, et al, "Delay Test Generation", Proceedings Design Automation Conf., pp. 486-491, 1977.

- [7] T.M.Storey and J.W.Barry, "Delay Test Simulation", Proceedings Design Automation Conf., pp. 492-494, 1977.
- [8] F.Motika et al., "A Logic Chip Delay-Test Method Based on System Timing", IBM Journal of Research & Development, Vol. 34 (2/3), pp. 299-324, 1990.
- [9] T.Hayashi et al., "A Delay Test Generator for Logic LSI", Proceedings FTCS-14, pp. 146-149, 1984.
- [10] G.L.Smith, "Model for Delay Faults Based on Paths", Proceedings Intern. Test Conf., pp. 342-349, 1985.
- [11] C.J.Lin and S.M.Reddy, "On Delay Fault Testing in Logic Circuits", IEEE Trans. CAD, pp.694-703, 1987.
- [12] D.M.Wu et al., "Statistical AC Test Coverage", Proceedings Intern. Test Conf., pp. 538-540, 1986.
- [13] O.Bula et al., "Gross Delay Defect Evaluation for a CMOS Logic Design System Product", IBM Journal of Research & Development, Vol. 34 (2/3), pp. 325-338, 1990.
- [14] J.A.Waicukauski et al., "A Method for Generating Weighted Random Patterns", IBM Journal of Research & Development, Vol. 33 (2), pp. 149-161, 1989.
- [15] B.L.Keller and T.J.Snethen, "Built-In Self-Test Support in the IBM Engineering Design System", IBM Journal of Research & Development, Vol. 34 (2/3), pp. 406-415, 1990.
- [16] R.B.Hitchcock et al., "Timing Analysis of Computer Hardware", IBM Journal of Research & Development, Vol. 26 (1), pp. 100-105, 1983.
- [17] J.A.Waicukauski et al., "Transition Fault Simulation by Parallel Pattern Single Fault Propagation", Proceedings Intern. Test Conference, pp. 542-549, 1986.
- [18] J.A.Waicukauski and E.Lindbloom, "Failure Diagnosis of Structured VLSI", Design & Test, pp. 58-60, August 1989.
- [19] P.Chang et al., "AC Test Quality: Beyond Transition Fault Coverage", Proceedings Intern. Test Conference, this issue, 1992.