Targeting small delay defects

By Deepak Gaur
Senior Application Engineer
Mentor Graphics Corp.

In the nanometer technology era, as the designs going further below 65nm, maintaining quality of test is a big concern in the test community. Test engineers are facing new test challenges at lower geometries and the small delay defects are one of them which pose a serious concern for maintaining good DPM levels.

The lower technology nodes are more susceptible to delay defects. At such lower nodes, the process needs to be accurate. Inaccuracy may manufacture transistors that are slower in switching and operation. Such small delay defects may not show up in normal transition testing but appear when a test is targeted on timing critical paths. Thus, to maintain the quality of test for the reasonable DPM levels, more and more test engineers are looking to target small delay defects with ATPG.

TestKompress from Mentor Graphics is currently the most widely used EDA tool in the industry for test compression and ATPG. It offers two strategies for targeting small delay defect: path delay testing and timing-aware ATPG. In the following sections, we'll explore the usage of both the techniques and also look at their pro's and con's.

Path delay testing

The path delay testing is somewhat similar to traditional transition testing. The difference is that for path delay, a test is generated for testing the combined delay through a specific path that is defined by the user. In the path delay testing, the critical paths information from the static timing analysis tool can be read in TestKompress, and patterns are generated to test these paths on the ATE. If there is a small timing delay defect on these paths, it's likely to be detected with these path delay patterns. Path delay testing tests the gross delay defect on the path at system speed.

To prepare for path testing, the user needs to convert the STA critical path report to TestKompress readable format "path definition file." The user can also manually define the path in this file. One example of the path definition file is shown in **Figure 1**.

In the path definition file, the user needs to specify the starting (launch) and end (capture) point of the path, which would be generally state elements or the system IO, as shown in **Figure 2**.

There could be path or edge ambiguity (Figure 2), which can be defined in the tool. The user can load the paths from path definition file into internal path list of the tool. In path delay testing, one path is equivalent to two faults. User can add these paths as faults in TestKompress and generate test patterns.

```
PATH "path0" =
     PIN /I$6/Q + ;
     PIN /I$35/B0 + ;
     PIN /I$35/C0 + ;
     PIN /I$1/I$650/IN + ;
     PIN /I$1/I$650/OUT - ;
     PIN /I$1/I$951/I$1/IN - ;
     PIN /I$1/I$951/I$1/OUT + ;
     PIN /A_EQ_B + ;
END ;
```

Figure 1: Shown is an example of the path definition file.

Generating path delay patterns

The Path delay pattern uses two at-speed cycles for launch and capture events, as shown in **Figure 3**. The transition is launched at the launch state element or PI and the response is captured at the capture state element or PO. **Figure 4** shows an example of the robust path delay test. The clocks for the

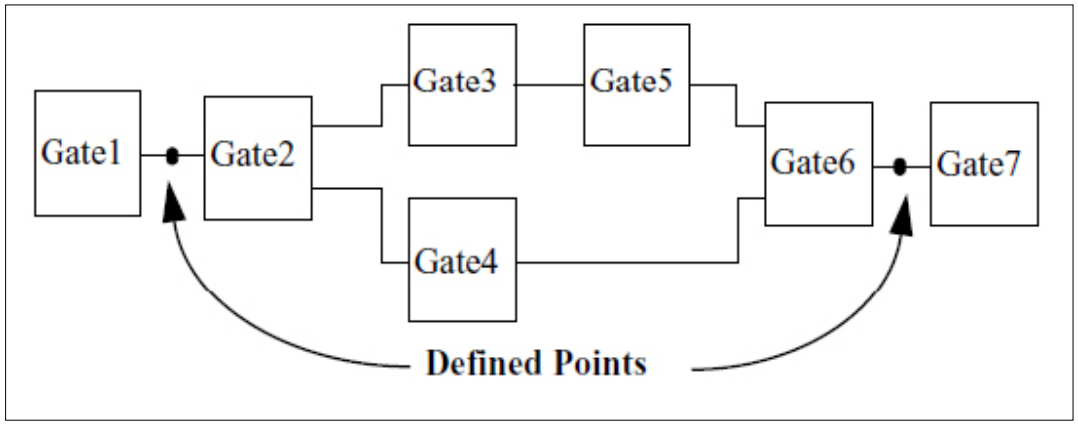

Figure 2: In the path definition file, the user needs to specify the starting (launch) and end (capture) point of the path, which would be generally state elements or the system IO.

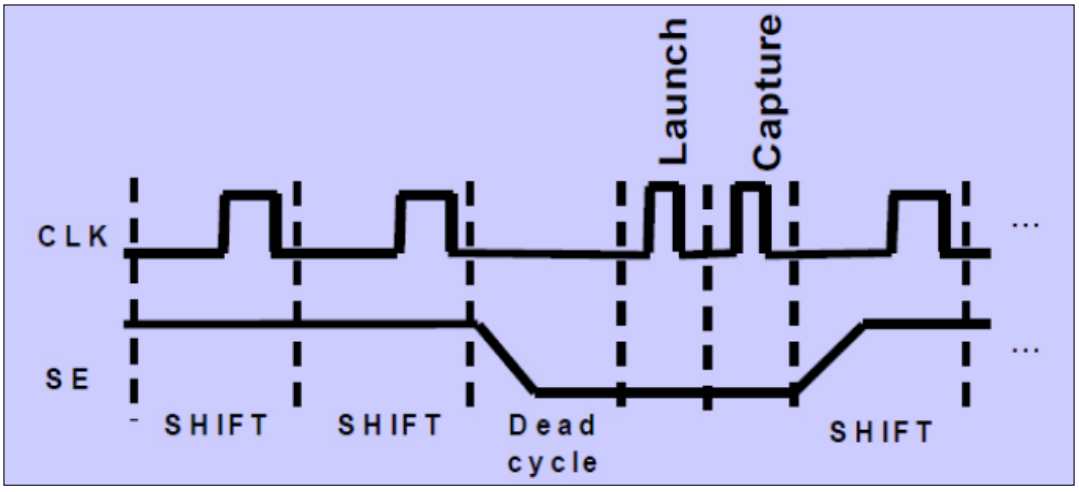

Figure 3: The Path delay pattern uses two at-speed cycles for launch and capture events.

path delay may be external or from internal PLL.

TestKompress automatically determines the scan cell values needed to sensitize all the gates within the path such that any change at the start of the path can propagate to the end of the path. In the case of path delay, the fault universe consists of two faults per path supplied to the tool. Since there are usually a fairly small number of paths targeted, the test coverage number reported is based on a small fault number and is not indicative of the path delay test coverage throughout the design.

The advantage in using path delay is that it is deterministic. The user knows exactly which path is getting tested as the launch and capture flops are specified. Therefore, the critical paths can be tested accurately and user can make sure that the small delays on the critical path do not go unnoticed.

The problem in using path delay is that STA tools report the critical path based on the longest static path irrespective of whether those path are sensitizable or not on the whole. Most of these paths are not sensitizable in ATPG and hence can't be tested. Secondly, if the critical path is not sensitizable, the ATPG drops the fault categorizing it as ATPG un-testable. ATPG doesn't try any alternate path, which may be comparable to the longest static path, for targeting small delay defect.

These shortcomings and issues are handled well in the second technique: Timing-aware ATPG.

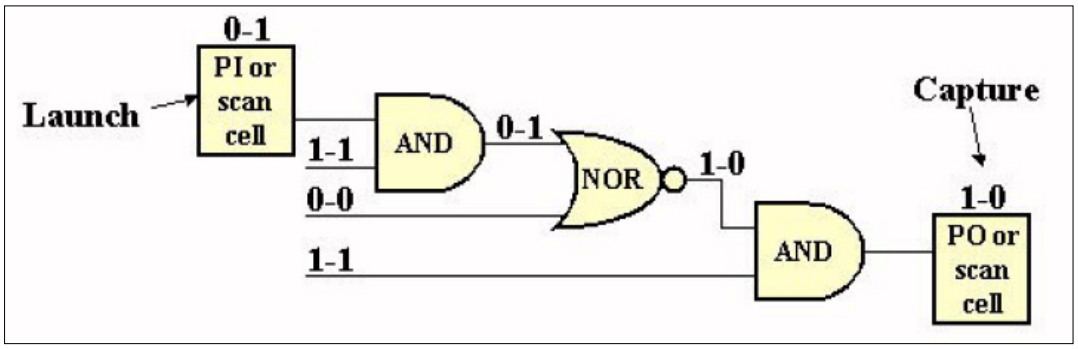

Figure 4: Shown is an example of the robust path delay test.

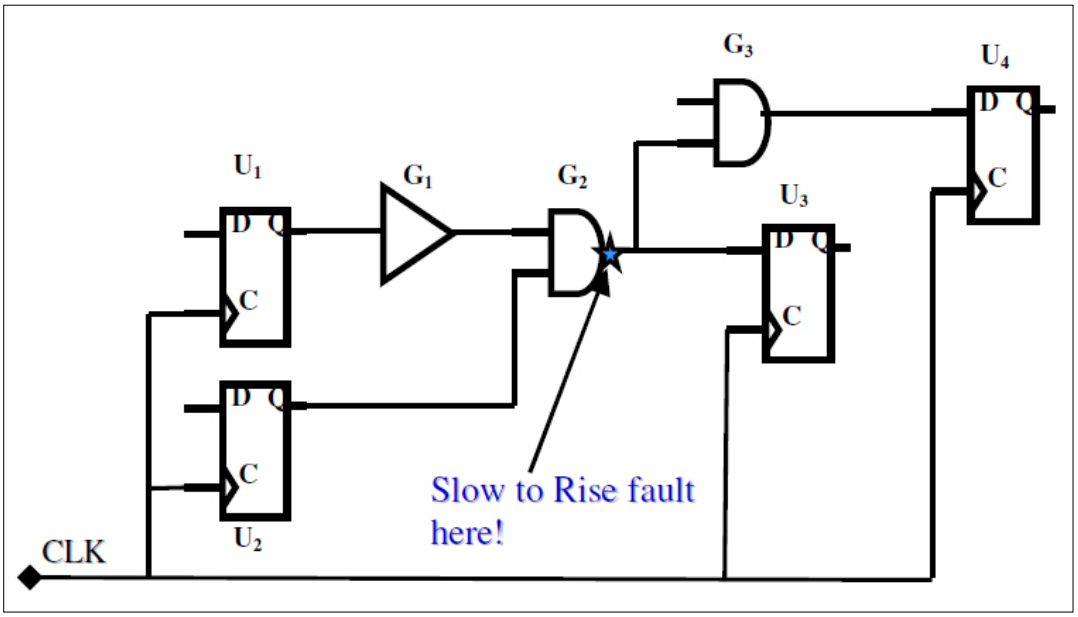

Figure 5: Shown is a sample circuit.

Timing-aware ATPG

Timing-aware ATPG is an improved version of transition fault, with timing information in-form of SDF. The faults are targeted similar to transition fault, i.e. every node in the design is a fault site. In transition faults, the tool launches and captures from randomly selected state elements. In contrast, timing-aware ATPG tool tries to launch from the longest upstream path and captures at longest downstream path based on the timing information provided in the loaded SDF file. The events in timing-aware ATPG are similar to the one showed in Figure 3. The events shown in Figure 3 correspond to launch off capture, which is more popular in test community. For timing-aware ATPG, launch off shift may also be used.

Figures 6 and 7 explain the difference between transition fault and timing-aware ATPG. In **Figures 5**, 6 and 7, there are four example paths to illustrate how timing-aware ATPG operates:

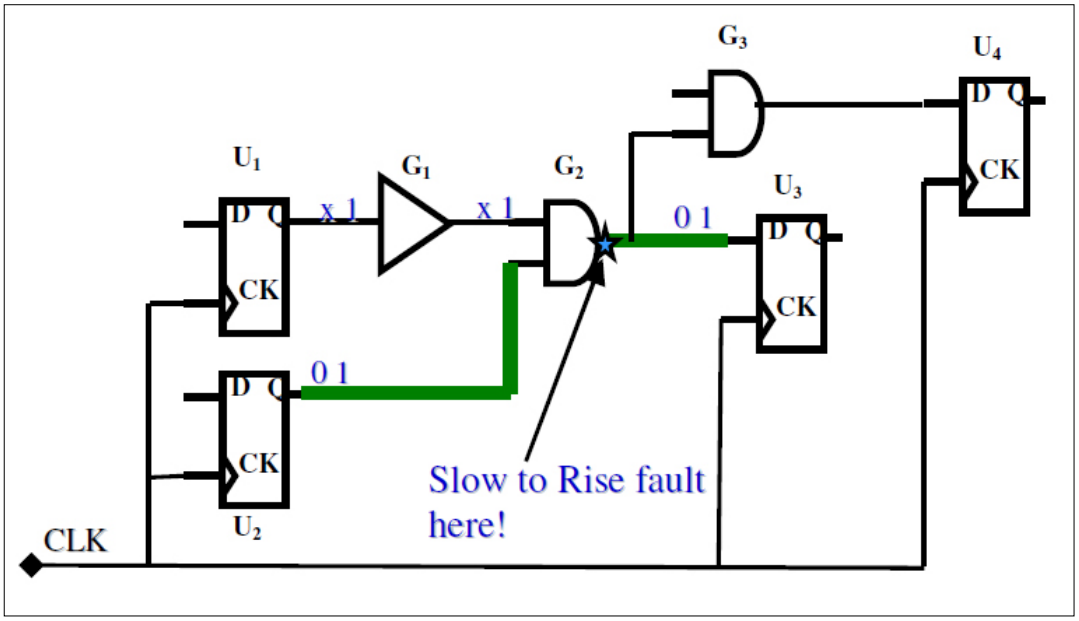

Figure 6: Shown is the transition fault testing at the output of AND gate G2.

```
R1: 4ns U1>G1>G2>G3>U4
R2: 3ns U1>G1>G2>U3
R3: 3ns U2>G2>G3>U4
R4: 2ns U2>G2>U3
```

Figure 6 shows the transition fault testing at the output of AND gate G2. The path used for testing is R4, which is just 2ns.

Figure 7 shows the timing-aware ATPG example. The same fault in Figure 6 is now being targeted with a path R1, which is 4ns.

To understand the impact of timing-aware ATPG, let's assume that the clock period in Figure 7 is 5ns and during manufacturing a small delay defect of 1.5ns introduced in output of G2. This small delay defect can be detected with timing-aware ATPG approach (Figure 7), but will escape the transition testing in Figure 6.

In timing-aware ATPG, it's possible that the longest static path in Figure 7 may not be sensitizable. In that case, the pattern is generated with alternate path but the delay coverage credit is not 100 percent. If the delay of the path used for ATPG is 80 percent of the longest static path, the delay coverage assigned is 80 percent for that fault site.

The disadvantage of timing-aware ATPG is that it needs more simulation efforts as the ATPG tools needs to calculate timing to select the right path for testing. This results in higher ATPG runtime. Secondly, timing-aware ATPG needs more pattern to achieve the same transition fault coverage, though timing-aware ATPG gets better delay coverage, which means better coverage for small delay defects. For optimum results, only a few timing critical faults (**Figure 8**) can be targeted with timing-aware ATPG, and the rest of the faults can be targeted with traditional transition fault patterns.

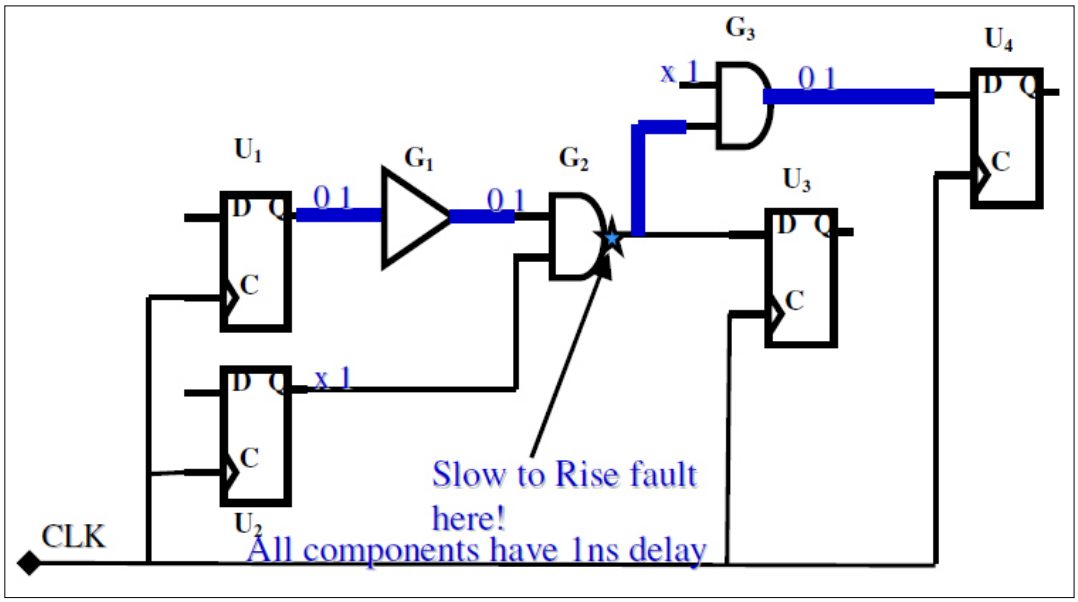

Figure 7: Shown is the timing-aware ATPG example.

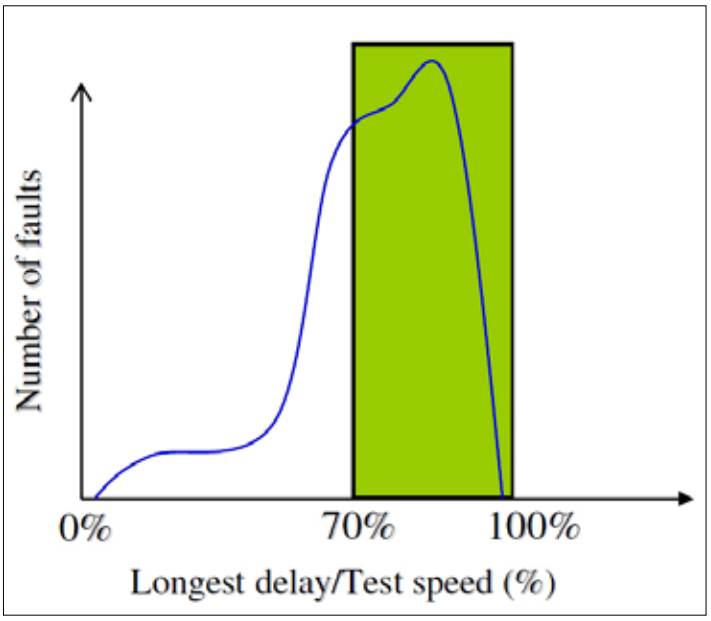

Figure 8: Only a few critical faults can be targeted with timing-aware ATPG.

Conclusion

Timing-aware ATPG will likely gain popularity in the near future as it is effective in dealing with small delay defects. As opposed to path delay fault model, timing-aware ATPG has far better delay coverage across the design. From the silicon results, timing-aware ATPG has shown some promising results in reducing the DPM at 65nm and below. In some designs, the reduction in DPM is about 30 percent to 40 percent with the addition of timing-aware ATPG.