Modern semiconductor testing is based on the notion of a fault model, that is, the expected behavior (response) of a circuit when a defect is present. ATPG (Automatic test pattern generation) tools use fault models to find the patterns required to detect the presence of defects at all points in the circuit. In part 1 of this series we described the most basic fault models and associated tests: the "stuck-at" and "transition" fault models. In this article, we cover a number of more advanced fault models that are commonly used in conjunction with scan testing.

**IDDQ**
The IDDQ test relies on measuring the supply current ($I_{DDQ}$) of an IC’s quiescent state, when the circuit isn't switching and inputs are held at static values. Test patterns are used to place the device under test in a variety of selected states. By performing current measurements at each of these static states, the presence of defects that draw excess current can be detected. The value of IDDQ testing is that many types of faults can be detected with very few patterns. The drawback is the additional test time to perform the current measurements.

**Toggle**
Toggle fault testing ensures that a node can be driven to both a logic 0 and a logic 1 value, and indicates the extent of your control over circuit nodes. Because the toggle fault model is faster and requires less overhead to run than stuck-at fault testing, you can experiment with different circuit configurations and get a quick indication of how much control you have over your circuit nodes. Because the toggle fault model only excites fault sites and does not propagate the responses to capture points, you can't use it for defect detection. This fault model is sometimes used for burn-in testing to cause high activity in the circuit.

**N-Detect and EMD**
The basic idea of n-detect (or multi-detect) is to randomly target each fault multiple times. The way the fault is targeted is changed randomly, as is the fill (bits that don't matter in terms of the fault being targeted) in the pattern set. This approach starts with a standard stuck-at or transition pattern set targeting each potential defect in the design. The pattern set is analyzed to see which potential defects are addressed by more than one pattern in the total pattern set. Then, additional (different) patterns are generated that specifically target the defects detected fewer times than the user-specified minimum. The combined information for all the resulting patterns increases the potential for detecting a bridge defect that might otherwise escape.

EMD (Embedded multiple detect) is a method of improving multiple detections of a pattern set without increasing the number of patterns within that pattern set. EMD uses the otherwise unspecified—fill or don't care—bits of an ATPG pattern to test for nodes that haven't reached their N-detect target. Standard multiple detect (N-detect) will have a cost of additional patterns but will
also have a higher multiple detection rate than EMD. How much difference there is between EMD and multiple detect defect detection will depend on the particular design's pattern set and the level of test compression used.

As an example, the test results on one 90 nm CMOS design demonstrated that embedded multi-detect tests detect 2.3% to 4.7% more defective devices than conventional single-detect stuck-at tests. This fault model or detection method is becoming more common because it doesn't increase the size of the test set and can produce additional detection.

**Deterministic bridging**

**Deterministic Bridging**
The deterministic bridging test utilizes a combination of layout extraction tools and ATPG. Based on a set of geometric rules, the extraction tool creates a list of net pairs that have the potential of bridging (Figure 1). This list is then fault simulated using existing stuck-at and transition patterns to determine which bridge defects can be detected. The net pairs that are not covered by the initial patterns are identified, and then used by the ATPG tool to generate a specific set of test patterns to completely validate that the remaining nets are not bridged.

![Figure 1. These examples of geometric rules help you identify potential bridges.](image)

**Small-Delay Defects**
At design nodes of 180 nm and larger, the majority of manufacturing defects are caused by random particles that cause bridges or opens. There are very few timing related defects at these larger design nodes because manufacturing process variations cause relatively small parametric changes that would affect the design timing. At design nodes of 90 nm and smaller, however, the same manufacturing process variations can cause on-chip parametric variations to be greater than 50%. This creates a situation where timing-related failures are a significant percentage of overall test failures.

You might expect that transition test patterns, covered in part 1 of this series, would find all of the timing defects in the design. Most or the time, it will, but some of the smallest delay defects can evade the basic transition test pattern. For example, when a path through vias, gates, and interconnects has a minor resistive open or other parametric issue that causes a delay, the accumulative defect behavior may only be manifested by long paths. A common scenario is where the same via type is used multiple times in the same path, and the vias are formed as resistive vias.
To detect this defect, an SDD (small delay defect) test can be performed. Circuit timing and physical layout information is used to guide the test generator to detect faults through the longest paths to improve the ability to detect SDDs. SDD testing is often referred to as timing-aware ATPG and is growing in usage for designs that have tight timing margins and low DPM (defect-per-million) requirements. The resulting patterns have a much higher probability of catching small-delay defects. One case study of a low DPM product has shown a 50% reduction in test escapes when timing-aware ATPG was applied.

User Defined Fault Models
UDFM provides users the ability to define new fault models without waiting for them to be built into their commercial test tools. It allows users to define the stimulus requirements that must be met in order for test patterns to detect a specific defect. The UDFM model is text based and incorporates the standard fault models provided with ATPG tools. Here’s an example of a UDFM:

```
Cell "MUX2" {
  Fault "Z1" {
    test { StaticFault "Z"=1;Condition "D0"=0,"D1"=0,"S"=0; }
    test { StaticFault "Z"=1;Condition "D0"=0,"D1"=1,"S"=0; }
    test { StaticFault "Z"=1;Condition "D0"=0,"D1"=0,"S"=1; }
  }
}
```
This UDFM specifies that ATPG must produce one of the three listed input patterns in order to detect fault Z1. This fault model provides a lot of flexibility and can specify any number of test cycles and can utilize library models, instances, and hierarchical paths within the design.

**Cell-Aware test**

As the industry moves to increasingly smaller geometries, we have discovered that fault models and associated test patterns are becoming less effective for ensuring desired quality levels. All of the existing fault models only consider faults on cell inputs and outputs, and on the interconnect lines between these cells. It turns out, however, that a growing number of defects occur within the cell structures. With more recent fabrication technologies, the population of defects occurring within cells is significant—as much 50% of all defects. In addition, the performance advantages of FinFETs have led to widespread adoption of these new transistors at advanced technology nodes. Consequently, we need a way to generate test patterns that efficiently target potential new defects at the transistor level.

A recent approach, called cell-aware test, targets specific shorts, opens, and other physical defects internal to each cell by modeling it at the transistor level. Analog simulations are performed to characterize the effects of potential short and open defects. Based on the analog simulation results, a cell-aware fault model is created that directs ATPG to generate patterns targeting these internal cell defects. There have been multiple studies that show that tests generated using cell-aware UDFM find defects that the normal test methods miss. One such study by AMD found that after 400,000 die had been tested, 880 defective die per million passed the standard manufacturing test, but were detected by the cell-aware tests.

The first step in the cell-aware methodology is to characterize each cell in a technology library. The flow is illustrated in Figure 3. The transistor layout for each cell, typically in GDSII format, is the starting point. An extraction tool, such as Calibre xRC, is used to extract a transistor-level analog netlist, including parasitic capacitors and resistors. The netlist is used to identify the location of possible bridge and open defects. To model a potential cell internal bridge, the parasitic capacitor is replaced by a resistor model. Opens occur when there is a gap in a connection. In this case, a parasitic resistor that describes connectivity is replaced by a high-impedance resistor.

Analog simulation is then performed to generate the cell-aware model. The analog simulation process iteratively modifies each parasitic element in the netlist, performs the simulation, and
compares the results to the fault-free analog simulation to conclude if the inserted defect is detected or not. Defects are determined as “detected” when the cell’s output voltage deviates from the “good circuit” voltage by a specified percentage (typically 50%). Not all bridge or open defects are, however, detected by a static change in the output voltage. Some may result in a delay in the output voltage swing. For these defects, a dual-cycle analog fault simulation is performed at-speed in order to detect even small delays.

The final process in cell-aware characterization is to convert the list of input combinations into a set of the necessary input values for each fault within each cell. Because this fault information is defined at the cell inputs as logic values, it is basically a logic fault model representation of the analog defect simulation. This set of stimulus for each cell represents the cell-aware fault model file for ATPG.

Within this file, a simulated defect (now a fault) can have one or more input combinations. An example of a cell-aware fault model is shown here:

```plaintext
udfm1.0 {
udfmtree "my_stuck_at" {

cell "XOR2" {
Fault "my_stuck_01" {
Test { StaticFault "Z"=0; Condition "A"=0; Condition "B"=1; }
Test { StaticFault "Z"=0; Condition "A"=1; Condition "B"=0; }
Test { StaticFault "Z"=1; Condition "A"=0; Condition "B"=0; }
}
}
}
```

For this example fault ‘my_stuck_01’, ATPG will try to find any of the three input combinations when targeting this fault in a design. If any one of the combinations can be applied to an instance of the cell and the fault effect can be propagated to an observation point, then the fault is marked as detected for this instance; the other combinations are no longer necessary.

Because the cell characterization process is performed for all cells within a technology library, any design using that technology can read in the same cell-aware fault model file. Characterization only needs to occur once, and can then be applied to any design using that technology node library.

### Modeling FinFET defects

The cell-aware methodology is well suited for addressing defect mechanisms specific to FinFETs. Consider a FinFET transistor with three fins, as illustrated in Figure 4. Research suggests that two defect types should be considered for such transistors: leakage defects that force the transistor partially or completely on, and drive-strength defects that force the transistor partially or completely off.
Figure 4. A three-fin FinFET transistor.

The leakage defects can be analyzed by placing a resistor across the gates (from drain to source) of each of the transistor’s three fins as shown in Figure 5. During the cell-aware characterization process, analog simulation is performed with varying resistive values for all resistors for all FinFETs in a given library cell. Exhaustive analog simulation must be performed for both single-cycle and dual-cycle tests, as many of these resistive defects will only result in small extra delays to the transistor’s response and the output of the cell (Figure 5).

Figure 5. Leakage defect simulation.

The drive-strength defects can be analyzed by placing a resistor between the drain and each of the fin’s gates and between the source and the fin’s gates as shown in Figure 6. As with leakage defects, analog simulation is performed with varying resistive values for each of the resistors. Once again, both single and dual cycle tests must be simulated to detect delay-related effects.
Any additional defect types that are discovered and are relevant to FinFETs can be handled in a similar fashion. The generic approach to analog defect simulation used by the cell-aware methodology makes this straightforward.

Once the cell library characterization is completed, the result is a UDFM model that can be used with the ATPG tool. After reading the UDFM file, the ATPG cell-aware pattern creation process proceeds similar to any of the other fault models.

Also see
- Design for test boot camp, part 2: Test compression
- Design for test boot camp, part 1: Scan test
- Improve test quality and reduce DFT costs
- Contactless Testing
- IDQ failure diagnosis is here