

# Using New Conformal Custom Features with Encounter Test to Generate Structurally Accurate Test Views for DFT

Colin D. Renfrew  
Freescale Semiconductor  
7700 W. Parmer Ln  
Austin, TX 78729  
[colin.renfrew@freescale.com](mailto:colin.renfrew@freescale.com)

Luis Basto  
Cadence Design Systems, Inc  
4516 Seton Center Parkway  
Austin, TX 78759  
[lbasto@cadence.com](mailto:lbasto@cadence.com)

Sung-Fan (Nick) Peng  
Cadence Design Systems, Inc  
4516 Seton Center Parkway  
Austin, TX 78759  
[sfpeng@cadence.com](mailto:sfpeng@cadence.com)

**Abstract:** *Structurally accurate test views (STAC) of standard cell libraries for Design-for-Test (DFT) are currently being adopted to increase the ability to detect manufacturing defects and enhance overall test quality. With STAC views, the Automatic Test Pattern Generation (ATPG) tools can more accurately target potential defects and increase device testability. This paper will focus on the generation and validation of STAC views using the enhanced features of Conformal Custom and the application of these test views to ATPG through Encounter Test. Included in this discussion are the importance of STAC view for Freescale, especially with the push for Zero Defect and moving towards 45nm technology, and how the Conformal Custom tool was enhanced to support this capability, making use of special Encounter Test fault models and increasing the compatibility between these tools. Also discussed is how the methodology to generate and validate STAC cells using the Cadence tools was developed in Freescale and applied to a full SoC design. Lastly, this paper also discusses plans for future enhancements to the Conformal and Encounter Test tools to further support the use of STAC models.*

## 1. Introduction

The goal of achieving zero defective parts-per-million (PPM), also known as Zero Defect, has been adopted by many as the metric for delivering quality devices to customers. This is particularly relevant to safety-critical or industrial markets such as automotive. Zero Defect directly translates to exceptional device quality, which in turn demands exceptional device test. The major part of silicon testing is achieved through ATPG, for initial silicon debug, production testing, performance testing and device failure analysis. The quality of these ATPG test patterns are therefore extremely important in order to thoroughly test a device, in particular for Zero Defect parts but also for devices with less stringent test targets. Commercial scan ATPG tools require a Boolean gate-level netlist of a design in order to generate the tests, which is

several levels of abstraction above the layout on silicon. One of the risks introduced here is that this model may be too abstract from the physical and structural characteristics of the device to generate tests accurately and effectively. To avoid the need for a Boolean gate-level netlist, design teams can consider performing ATPG at the transistor level. Some research has been done in this area but there exist no practical (i.e., commercial) tools that allow ATPG at transistor level with reasonable pattern generation and simulation time. Furthermore, the transistor level netlists do not support the stuck-at fault model well – the mainstay fault model of the chip industry.

For these reasons, ATPG continues to be performed using a Boolean gate-level netlist of a design. This netlist consists of technology library cells that are in a specific format for ATPG, which is typically different from the formats used for other parts of the design flow. The gate-level equivalent for a design module created for test purposes is called a Testview. The design module can be a standard cell library, an embedded memory, an analog module or a custom designed circuit. For standard cells, which make up the majority of the design, the test views are traditionally accurate to the function of the cell only and do not represent the true structure of the cell as it exists on silicon. This is mainly because there has been no commercially available tool to do this - structurally accurate test views are more complex to create and validate, and can become a very time-consuming manual process. Continuing to use traditional test views presents the problem that the potential faults on the device may not be modeled accurately enough for ATPG tools to test them, risking the quality of the test patterns, and of the device itself.

The solution to this was to build on the capabilities that already exist in the Conformal Custom tool for Verilog gate-level design abstraction. These enhancements provide the capability to generate and validate Testviews for a standard cell library that can be used by ATPG tools, such as Encounter Test.

The purpose of this paper is to:

1. Present some background on the topic and the meaning of “structurally accurate” (STAC).
2. Explain how STAC models are generated and validated in Conformal Custom.
3. Show how STAC models are used in Encounter Test for ATPG.
4. Describe the flow developed in Freescale to generate and use STAC models.
5. Present and discuss the results from experimental designs.

## 2. Previous Work

Various techniques have been used in the past as well as currently to develop circuit views for test pattern generation and simulation. The major idea is to present a gate level model to the automatic test pattern generation tool to generate patterns to detect physical defects as close to the gate model as possible.

There exist some non-commercial tools that are capable of generating gate-level representations of library cells from the transistor level equivalent, including GateMaker [2], which can generate gate-level models specifically for the purposes of ATPG. GateMaker is a tool developed by IBM and Intel which automatically generates gate level models from transistor schematics. It has algorithms that collapse parallel transistors, map resistors to plain nets and capacitors to opens. Then it uses path tracing to map groups of transistors to a CCC (a Channel Connected Component, which is explained later). Finally, it uses path pruning and simplification to reduce the circuit into its final equivalent form. It appears to be a fairly capable tool but of course it’s proprietary so generally unavailable to the design community. The successful use of tools such as GateMaker and the adoption of this methodology have been proven previously in [3] and [4].

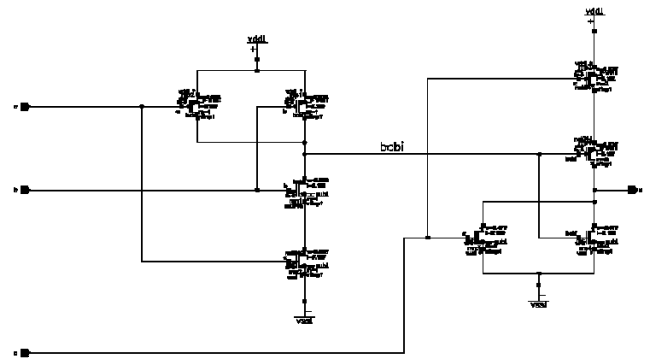
Other users perhaps use a combination of commercial tools, custom scripts, and manual editing to generate gate level models from transistor libraries. Clearly, there is a need for automation to generate structurally accurate test views with minimal to no manual intervention.

## 3. What is “Structurally Accurate”?

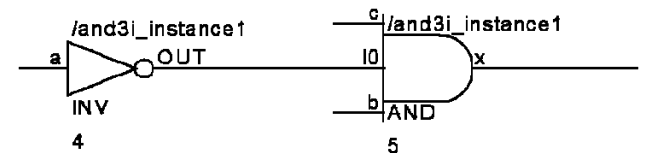
A standard cell library provides the essential building blocks for a chip design and is made up of simple cells such as AND/OR/NAND/NOR/NOT gates; complex gates such as AOI (And-Or-Invert), XOR (Exclusive-Or); and

specialized cells such as Adder, Comparator, Parity and one-hot checker [1]. A typical standard cell library in 90nm technology can have around 700 unique cells, covering various drive strengths and physical characteristics for all of the different cell types.

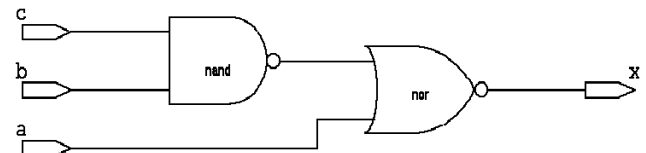
Figure 1 shows the transistor level circuit of an example cell. This cell is, functionally, a 3-input AND gate with a single inverted input. Figure 2 shows a possible representation of this cell as a testview. At first glance, the testview appears to be reasonable representation of the actual circuit – it is functionally equivalent and it requires a minimal number of Boolean gates, to speed up ATPG simulation and pattern generation. However, the testview shown in Figure 3 represents the structure and placement of the transistors more accurately.



**Figure 1: Transistor-Level 3-input AND gate with single inverted input**



**Figure 2: Possible testview for 3-input NAND gate with single inverted input**



**Figure 3: Structurally accurate testview of a 3-input AND gate with single inverted input**

So, what is the impact of using the testview in Figure 2, which is functionally equivalent but not structurally equivalent, versus using the testview in Figure 3, which is both functionally and structurally equivalent to the actual circuit?

There are three aspects to consider

1. The number of stuck-at faults
2. The number of test patterns generated and their coverage against the faults
3. The accuracy of the diagnostic resolution

With the circuit shown in Figure 2, faulting internal to the cell would be incorrect, since the internal net does not exist in the transistor level circuit, misrepresenting what is on silicon and possibly causing misleading diagnostic results. With this type of cell representation, it is therefore necessary to fault at the boundary of the cell only. The circuit in Figure 3, being structurally accurate, allows the designer to include the faults internal to the cell and not only at the boundary, increasing the fault universe and enabling the ATPG tool to generate specific test patterns to target those faults.

## 3.2 Challenges

### 3.2.1 Physical Cell Structure

In order to generate a structurally accurate testview model, the actual physical structure of the cell has to first be determined before selecting a gate-level cell (or cells) to represent the function. Remembering, however, that the testview is modeled at gate-level, how can the physical structure be accurately abstracted to a higher level model without compromising the ability of the ATPG tools to effectively and efficiently generate test patterns?

The following key criteria were recognized as features that must be maintained during this abstraction process from transistor to gate-level:

1. Maintain intended functionality
2. Preserve channel connected logic in the same logic stage.
3. Preserve the number of logic stages between the cell input and output.
4. Use Boolean gates, as represented by the cells (no transistors) and use exclusive gates only.
5. Use built-in ATPG tool primitives for latches and flip-flops, 3-state cells and pass-gate muxes.

### 3.2.2 Entire Structurally Accurate Libraries

As mentioned previously, a standard cell library for a 90nm process can contain hundreds of unique cells, some of which are simple combinatorial cells, others of which are more complex. Given the different possible ways to

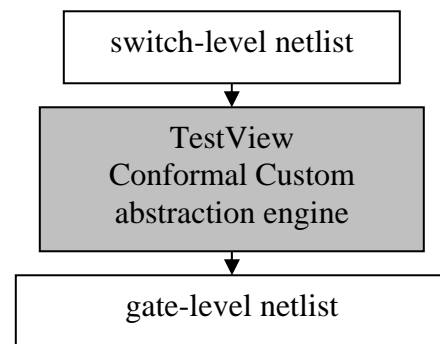
represent different cell types, one of the main challenges is how to automate this process such that an entire library can be generated and classed as structurally accurate. Secondly, how can the verification of such cells be automated, to ensure that they are still functionally accurate to the original cells and also understandable by the ATPG tools?

## 4. Testview Generation with Conformal Custom

The Conformal Custom tool already has a very powerful abstraction engine to generate gate-level equivalent Verilog netlists from the original transistor level (switch level) circuit, for many different purposes. The existing capabilities of the tool were leveraged and enhanced to support the new feature for the generation of STAC views, in gate-level Verilog, referred to in the Conformal tool as “TestView”.

### 4.1 Tool Enhancement - TestView Feature

TestView makes use of the current abstraction engine of Conformal Custom. The flow is shown as Figure 4.



**Figure 4: TestView for Conformal Custom**

There are two major steps of the implementation of TestView. The first step is to use the formal methodology to abstract the function of a circuit based on Channel Connected Components (CCC), which is the same as the current abstraction technique of Conformal Custom. A CCC is defined by the maximal set of transistors and nets such that every net can be reached by traversing the source and drain of the transistors in the component [2]. The second step is to deposit the gates abstracted from each CCC into a new module. Taking the circuit shown in Figure 5 as an example, the figure shows that there are four CCCs in the circuit, with each CCC covered with different color of boxes. The gate-level TestView of the circuit is shown in Figure 6.

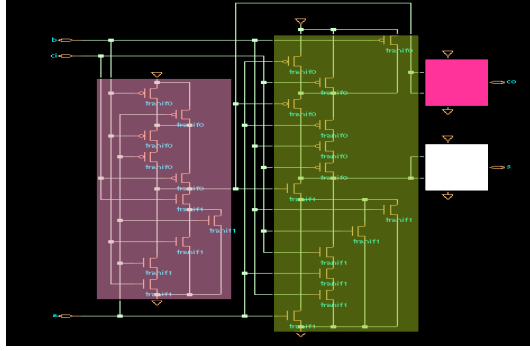


Figure 5: A circuit of 4 CCCs

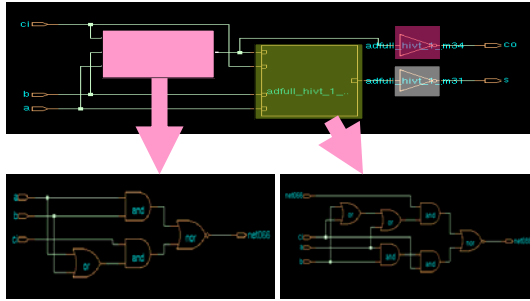


Figure 6: TestView of circuit in Figure 5

## 4.2 Abstraction Algorithm Overview

### 4.2.1 Forming Gates

As indicated in the last section, the first step of the TestView abstraction is to form AOI(OAI) gates based on CCC. Figure 7 shows a typical static complementary logic expressed in CCC [2]. There are three CCCs in this network. The gate-level expression of each CCC can be abstracted by analyzing the BDD function of pMOS and nMOS network in the CCC.

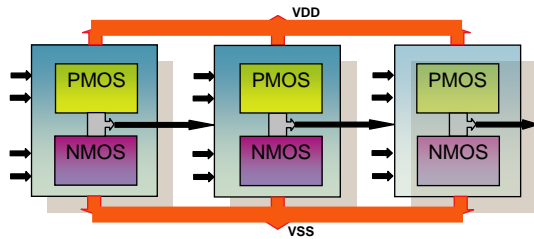


Figure 7. Circuit shown in CCC

In the general case, each CCC may contain any combination of p-type transistors and n-type transistors. As an example, for a single CCC each transistor can be viewed as a switch by controlling the gate voltage of the

transistor (see Figure 8). A pMOS transistor will be turned on when its gate voltage is low and turned off when its gate voltage is high. On the other hand, an nMOS transistor has an opposite situation, which means an nMOS transistor is turned on with its gate voltage high and turned off with its gate voltage low.

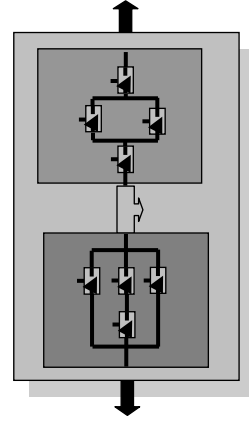
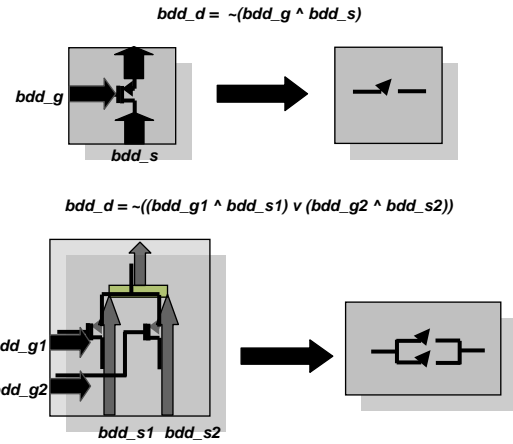


Figure 8: Transistors as switches



**Figure 9: BDD expression of nMOS networks**  
These relations can be easily represented by BDD operations and thus the function of the CCC can be derived. Figure 9 shows two nMOS networks represented by BDD expressions.

### 4.2.2 Creating a New Module for Each CCC

After the function of the CCC is derived, the logic gates representing the function will be deposited into a new module. Figure 10 shows the TestView of the network in Figure 7. If the new modules only contain a single gate, the modules will be resolved to reduce the level of hierarchy.

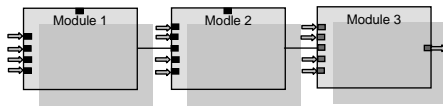


Figure 10: TestView of Figure 7

#### 4.2.3 Abstraction of Sequential Logic

For sequential logic, the abstraction needs to go further. Since the abstraction is module-based, the first step is to pop up the modules created for each CCC. The next step is to traverse the circuit to see if a loop could be found. If a loop exists, the Conformal Custom abstraction engine can verify if it could be formed a latch or not. The gates from the modules not involved in the abstraction will be pushed down to the module in which they are originally located. After latches are abstracted, the next step in the abstraction algorithm will be to see if flip-flops could be formed or not.

## 5. Encounter Test and Fault Modeling

Encounter Test is a suite of tools that support a complete design for test methodology flow from test insertion to test pattern generation, test vector export and defect diagnosis [7]. Since this paper is mainly concerned with generating the correct test view, the focus of this discussion will be on fault models.

ATPG tools usually work on gate level netlists and gate level library models since working with transistor models is too complex. To model a potential fault area of the logic where a defect may occur and be detected on silicon, a variety of fault models exist that a design engineer can use for pattern generation, such as the industry accepted stuck-at fault model. For designs at 90nm or smaller, it is generally accepted also to require tests using transition and path delay fault models.

In addition to these, Encounter Test supports a patented pattern fault model. This is useful for modeling defects not easily covered by the other models, such as bridging and crosstalk defects. Figure 11 shows a diagram of a CMOS multiplex or built using pass gates.

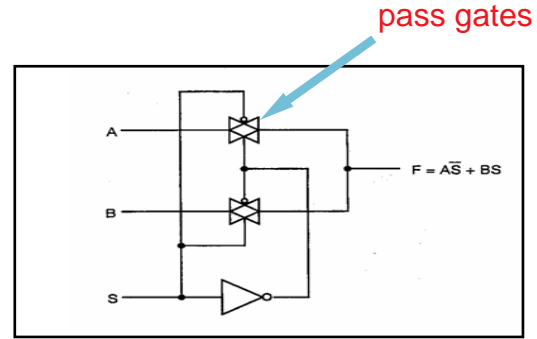


Figure 11: A CMOS MUX built using pass gates

Defects on the select signal can cause high-Z contention which is not detected by the stuck-at or transition fault models. Using the build-in primitive from Encounter Test shown in Figure 3 and enabling pattern faults will result in better quality test patterns.

```
module __MUX2(DOUT, DATA0, DATA1, SEL);
input DATA0, DATA1, SEL;
output DOUT;
endmodule
```

Figure 3: ET TestView of a two-input MUX

Similarly, using built-in ET primitives for latches and flip-flops and activating pattern faults will let the test generator take advantage of the pattern fault model to generate better and higher quality test patterns.

### 5.1 Cell Boundary Modeling

Encounter Test has extensive fault modeling capabilities to suit many design styles and needs. One important consideration is modeling the cell boundary for technology libraries. ET supports modeling which is compatible to other industry ATPG tools.

The most accurate modeling is where all internal nets in a library cell are modeled, shown in Fig 12. This is recommended only if there is a one-to-one relationship between the physical structure and the logical model, otherwise, the default ET model or the “industry compatible” model must be used. Here is where the premise of STAC models comes in. If the nets shown in the model in Fig 12 exist in the actual design, then they can be tested and defects on them can be diagnosed. Otherwise, the ATPG tool will target faults that do not exist on silicon, and the resulting data, test coverage and failures on silicon may be misleading.

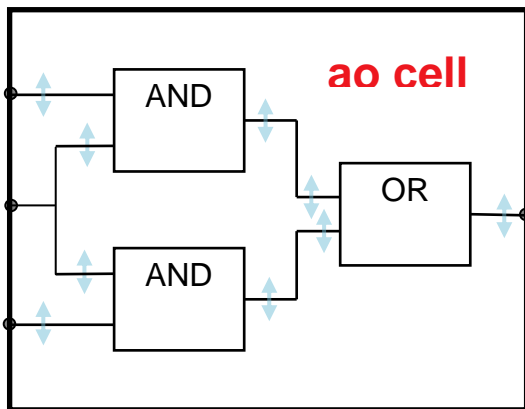


Figure 12: Allowing internal faulting

## 6. Freescale Flow for STAC view Generation and Validation

This section details how the previous sections on the theory of STAC modeling, the enhancements through the Conformal tool, and the features of Encounter Test were brought together to create a methodology in Freescale for automatic STAC view generation and validation for entire standard cell libraries.

The Conformal Custom tool is used to extract STAC gate-level Verilog models of the library cells using the generation flow shown in Fig 13. The input to the process is the “.cdl” file, which is the transistor level circuit for the library cell that comes from the library designers themselves. The new Testview feature of Conformal is used to generate the STAC cell and the built-in LEC feature is then used to verify that it is equivalent to the functional model of the cell.

Suitable scripts can easily be written to run Conformal to process an entire library in this manner. If the resulting STAC Testview is not structurally accurate, or the model does the extraction process must be reviewed for that particular cell type. Otherwise, the STAC cell is considered correct and can be further verified through Encounter Test.

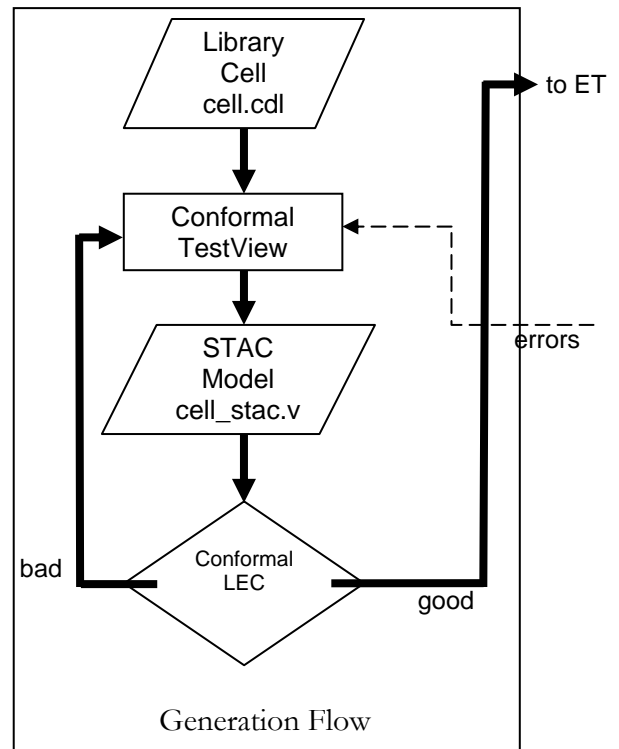


Figure 13: STAC Generation Flow

Before using these STAC cells on a full design, the next step is to run the cells through Encounter Test for further verification. LEC has already verified that they are logically equivalent to the functional views of the cell, but this additional step verifies that they are recognized by the ATPG tool, that they can be faulted correctly for pattern generation, and that the patterns pass in simulation against the functional models without any mismatches. This flow is shown in Figure 14.

In this flow, a “skinny” design netlist is created that instantiates one instance of each cell in the STAC library. This is run through Encounter Test with the library itself and ATPG patterns generated. The reports and fault lists can be analyzed (manually or with scripts) to validate that the cells are being interpreted correctly. The ATPG patterns are then simulated and through the log files and waveforms produced any mismatches can be debugged and fed back to either Encounter Test or Conformal for correction. Once all of the cells pass and there are no mismatches in simulation, the STAC library can be used on any SoC design.

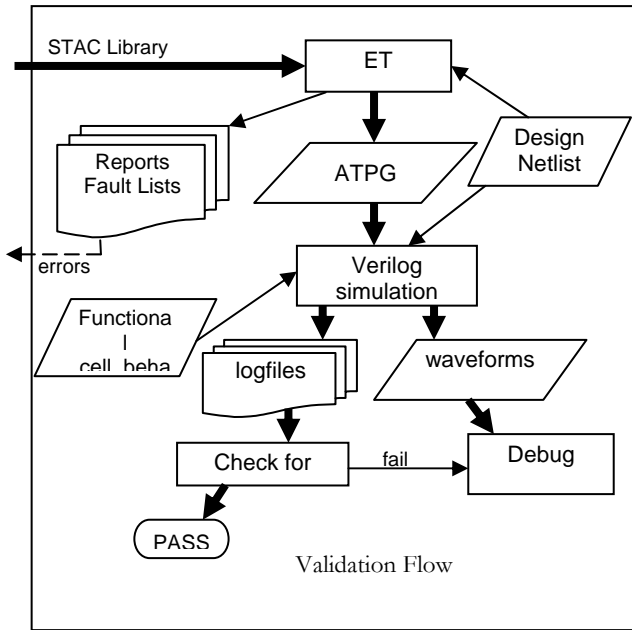


Figure 14: STAC Verification Flow

## 7. Application Results

### 7.1 Single Full Adder Cell

To demonstrate the effectiveness and difference between a STAC and a traditional test-view model, many different cell types were evaluated through this flow. One of the cells where the most dramatic difference was observed was in a full adder cell. Figure 15 shows the original transistor level schematic of the Full Adder cell.

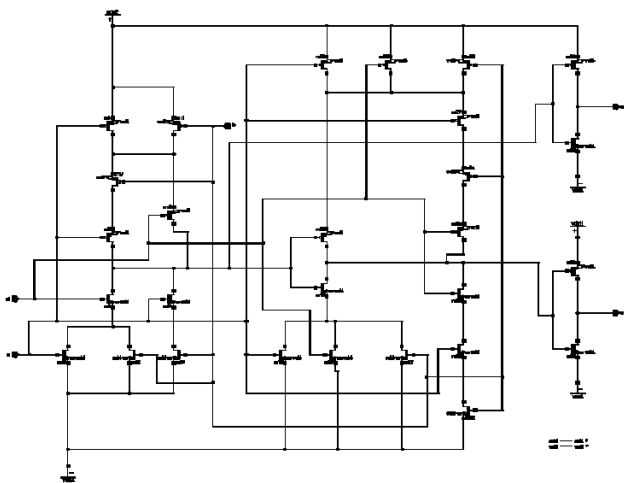


Figure 15: Transistor Level Full Adder Circuit

Figure 16 shows the test view from the original library, and Figure 17 the STAC library extracted by the Conformal Custom tools. As can be observed, the original library model is very different, structurally, to the original circuit and could not be used for faulting internal to the circuit; the nets, gates and logic levels do not map at all to what exists on silicon.

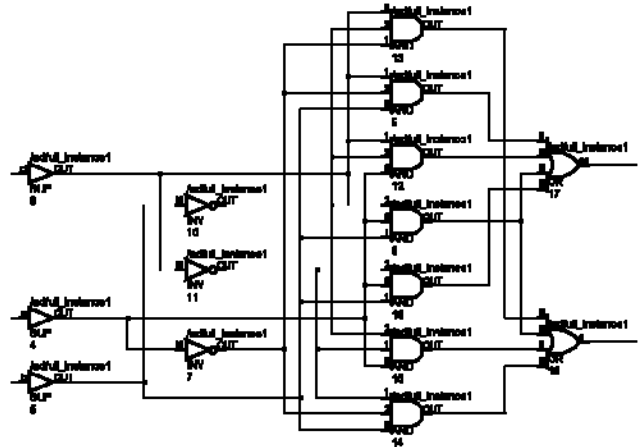


Figure 16: Original Full Adder Circuit Test View

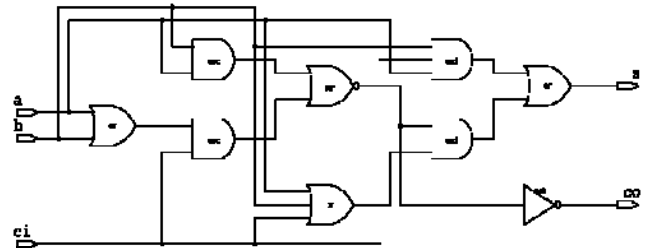


Figure 17: Structurally Accurate Full Adder Circuit

The results for these two testviews are summarized in Table 1. The original testview has 10 faults (faulting at the cell boundary) and the ATPG tool generates 2 test patterns for 100% test coverage. However, the structurally accurate testview allows the ATPG tool to fault internal to the cell and has 46 faults, for which the tool generates 8 test patterns to achieve 100% test coverage. Therefore, if one had settled for the original testview - the ATPG tool would be limited to generating only 2 test patterns. Fault simulating these 2 test patterns on the structurally accurate testview shows (see Table 2) that 30% of the faults would have gone untested!

It is only when the structurally accurate testview is presented to an ATPG tool is it able to generate the necessary additional test patterns (in this case 6 additional test patterns) to get 100% test coverage.

ATPG Test Pattern	Total Static Faults	# Tested	# Patterns	%TC
Original Cell	10	10	2	100%
Fault Simulated	46	32	2	66.67%
Internal Faulting	46	46	8	100%

**Table 1: Full Adder Test Coverage Comparison Results**

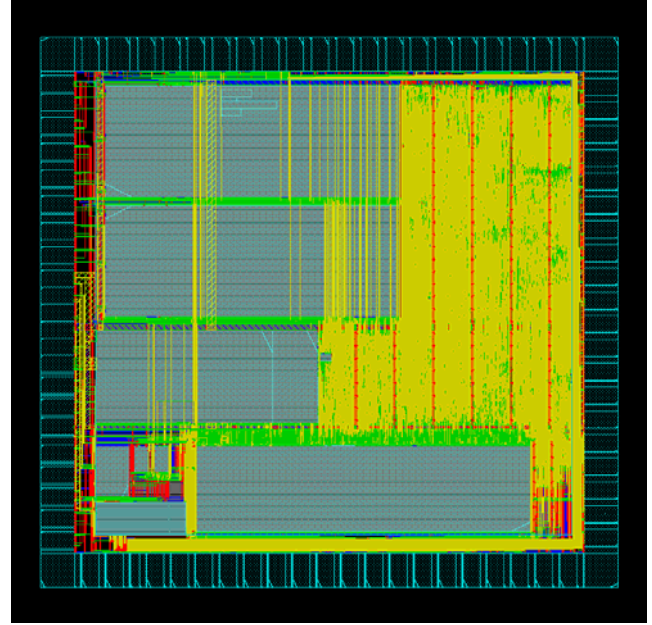
From the above two examples, the following conclusions can be drawn:

- Purely functionally equivalent testviews are logical for simulation and appear adequate for ATPG. However they are not adequate for high-quality ATPG and are not accurate for a high level of diagnostic resolution.
- There needs to be a method for automatically generating structurally accurate testviews from the transistor level representation of the library cells.
- Furthermore, there needs to be a method for verifying the functional equivalence of the structurally accurate testviews.

## 7.1 Entire Standard Cell Library

So far, results have only been discussed for single cells, so what is the impact of using a STAC library on a full SoC design?

To find out, a relatively small SoC design that was already using traditional test views from a mature, proven technology was chosen as the test case. This particular device is a low power, high performance, HCS08 8-bit microcontroller based SoC. The device was implemented in 0.25um low voltage, low power process technology and includes a 24 channel ADC along with several communications peripherals. Figure 18 shows the layout of this device.



**Figure 18: Microcontroller DUT Layout**

The flow described in Section 6 was followed using the source files for this library to generate and validate structurally accurate testviews. Two ATPG pattern generation runs were then performed on the design; the first using the original testviews and a second using this new library, faulting internal to the cells, and the results compared.

Table 2 shows the results where ATPG patterns were generated for the design using both the traditional and structurally accurate libraries. The top line details the design using the original test views, faulting at the boundary of the cell, as normal.

Pattern Set and Library	Internal faulting	# Static Faults	Dynamic TC	Static TC	# Patterns
ATPG on Original Lib	No	261447	87.95%	94.59%	6804
Fault Sim against STAC lib	Yes	477904	85.76%	93.08%	6356
ATPG on STAC lib	Yes	477904	88.02%	94.76%	7619

**Table 2: Full Design Static Test Coverage Comparison Results**



As was done for the Full Adder cell in section 2, the patterns generated here were then run on the structurally accurate library to find out the true coverage of the device gained by these original patterns, this time faulting internal to the cells.

Two observations can be made here – 1) faulting internal to the cells almost doubles the fault universe and 2) in reinforcing the theory at the end of section 2 of this paper, around 2% of this new, more accurate, fault universe is not covered by the existing patterns, reflecting the actual, true test coverage of the device.

This method allows us to then instruct the tool to generate additional patterns to cover the gap in fault coverage. In the third line of this table the patterns were generated using the structurally accurate library. It can be observed that with a small increase in pattern count the remaining faults were covered, enhancing the test quality of the device.

In section 2, it was shown that on a single cell the difference in test coverage was 30% and the increase in pattern count to cover the additional fault with a structurally accurate test view was 4 times the original pattern count. So, why on a full design is the difference not on the same scale?

This is because, on a full SoC device, the propagation of the stimulus introduced into the design by a test pattern generated to target one fault on any given cell, combined with the design architecture itself, allows some of the faults (internal or boundary) on other cells to be tested at the same time. Therefore, as a general rule of thumb, as the gate count for a SoC device increases, the proportional increase in pattern count becomes smaller and less significant.

## 7.1 Metrics

When evaluating the structural accuracy of an entire library, different levels of accuracy were defined based on what cell types were made to be accurate and which were still based on a basic functional model. In order to quantify this into a measurable form, a Maturity Matrix was generated using the five main cell types to indicate the overall level of structural accuracy of any standard cell library. This is shown in Table 3.

With this metric and the breakdown of the different cell types, it is not necessarily essential to convert every single cell in the library to be structurally accurate; the quality of a library can be enhanced even if only the combinatorial cells are made to be more accurate. In fact, given that

combinatorial cells make up the majority of the cells used on a typical SoC device, even a Maturity Level of 2 can have a significant impact on the overall test quality of the final ATPG pattern set.

Maturity Level	Simple Cells	Comb. Cells	Complex cells (AOI and Muxes	Sequential Cells	Special Cells
Level 1	Yes	No	No	No	No
Level 2	Yes	Yes	No	No	No
Level 3	Yes	Yes	Yes	No	No
Level 4	Yes	Yes	Yes	Yes	No
Level 5	Yes	Yes	Yes	Yes	Yes

**Table 3: Maturity Matrix for Structurally Accurate Test Views**

Level 1 equates to a test view library where only very simple, single cells are structurally accurate, such as inverters. Level 5 covers the entire library, including sequential cells and any special cells (e.g. Full Adder, Level Shifters etc).

It is worth noting that the structurally accurate library generated for this design was at Level 2 on the Maturity Matrix described in section 4. As shown by the results in Table 2, even only at level 2 the use of structurally accurate cells in a library can make a significant difference to the overall accuracy in a full design. By enhancing all of the AOI, Muxes and special cells for this library, the maturity level can be taken to Level 5 and the overall test quality enhanced even further.

## 8. Conclusion and Future Enhancements

By enabling ATPG tools to generate a more complete and accurate fault universe it can then produce test patterns that target the true potential fault areas of a device and enhance the overall test quality of the test. Structurally accurate test view models (STAC) have been recognized as an essential part of this flow. The enhancements made to the Conformal Custom tools, and the links made to the existing features of Encounter Test, have enabled a complete STAC generation and validation flow that can be applied to any standard cell library. Results in simulation have shown that this method has a significant impact on the overall testability and test quality of the resulting ATPG patterns.

Future challenges for this methodology include enhancing the Conformal Custom tools to take the structural accuracy of a standard cell library to level 5 on the Maturity Matrix.

Beyond that, the modeling of memories and custom designed circuits to be structurally accurate could also be explored. Structurally accurate standard cell testviews could also be used for gate exhaustive testing [7]. Proof of the true effectiveness of using this technique would be observed by running two pattern flows, one using normal and the other with STAC libraries, and monitoring the test escapes on high volume lots. Comparison of the accuracy of diagnostic callouts will be further proof. Freescale plans to run such experiments on a production device in the near future.

and Freescale, and also across several teams within both companies. The authors would like to thank and acknowledge the valuable contributions and support made by (Cadence) Mitch Hines, Gil Vandling, David Scott, Bassilios Petrakis, Manish Pandey and (Freescale) John Scott, Tommy Colunga, Vlado Vorisek, Darrell Carder and Raj Raina.

## References

- [1] N. Weste and K. Eshraghian, "Principles of CMOS VLSI design", Addison Wesley Publishing Company, ISBN 0-201-3376-6
- [2] S. Kundu, "GateMaker: A Transistor to Gate Level Model Extractor for Simulation, Automatic Test Pattern Generation and Verification", Proc. Of Int'l Test Conf, p372, 1998
- [3] L. Day et al, "Test Methodology For A Microprocessor With Partial Scan" Proc. Of Int'l Test Conf, p137, 1998
- [4] M.P. Kusko et al, "Microprocessor Test and Test Tool Methodology for the 500MHz IBM S/390 G5 Chip", Proc. Of Int'l Test Conf, p717, 1998
- [5] C. Pyron et al, "DFT Advances in Motorola's MPC7400, a PowerPC Microprocessor" Proc. Of Int'l Test Conf, p137, 1999
- [6] Encounter Test Documentation, Version 6.2, Feb. 2007.
- [7] E. J. McCluskey et al, "Gate Exhaustive Testing." Proc. Of Int'l Test Conf, Paper 31.3, 2005

## Acknowledgements

The work presented here has been made possible through the close collaboration and teamwork between Cadence