SOC DFT verification with static analysis and formal methods

D Sargent - November 17, 2010

To achieve higher quality on today's multimillion-gate designs and high-speed ASICs, structured DFT (design-for-test) methodologies such as scan, at-speed test, scan compression, and BIST (built-in self test) have become the most important features for manufacturing test. Typically, testability issues are detected at the very last stage of the design flow during ATPG (automatic test-pattern generation) or gate-level simulation. This stage is very close to tape-out, and it becomes necessary to fix the original RTL (register-transfer-level) design for re-use and testability and then repeat the steps of the implementation flow, a process that can have a large impact on the project schedule.

Many design teams have adopted static verification checks and methodologies to catch testability issues early at RTL for both stuck-at and at-speed testing. Yet, the connectivity of different IP blocks at the SOC level—whether respect to memory BIST, 1149.1 JTAG, or IEEE 1500 standards—is mainly verified through functional simulation. Test benches are not always the best solution for verifying connectivity of the logic at the SOC level with signals from SerDes, PLLs, IEEE 1149.1/1500 circuits, and BIST logic that go to various blocks in the design.

There are several general disadvantages to using functional simulation for verifying connectivity:

- Test benches are manually created and need several thousands of simulation cycles to put the design in a certain configuration for validation.
- The verification engineer must exhaustively test all conditions, including both positive and negative cases, which is very time-consuming.
- The default configuration through the test bench might be configuring some of the paths to be exercisable by chance and not by design.

A better solution involves using static and formal verification methods to quickly verify connectivity at the SOC level, preferably at RTL. This method provides several advantages:

- No test benches are required.
- Assertions prove that the connections exist and the desired value is correct at internal nodes with a minimum set of input constraints.
- Run times are faster and debug is simpler and easier with highlighted target failure points.
- This technique can be applied to new revisions of the design through regressions at both RTL and gates.
Using Atrenta’s SpyGlass-DFT tool, STMicroelectronics has applied these static techniques to verify different kinds of integrations in several SOCs with promising results.

**SOC DFT architecture**

SOC designs have several DFT structures and associated test modes. The total number of external and internal test modes could be on the order of tens to hundreds to include stuck-at test, at-speed test, scan compression, memory BIST, logic BIST, boundary scan, and multiple internal modes for accessing internal IP blocks and cores with IEEE 1500 wrappers. **Figure 1** shows the complexity in the DFT architecture that is necessary to achieve a high-quality manufacturing test for an SOC design.

![SOC DFT architecture](image)

**Figure 1.** An SOC DFT architecture requires complexity.

These external and internal test modes require hundreds of test-initialization sequences applied on the JTAG/IEEE 1149.1 TMS, TRST, TDI, and TCK signals on the TAP (test access port). The DFT architect derives the test sequence manually, and the required internal values of the test modes and connections are verified manually through dynamic simulations. The test benches for dynamic simulations are also created manually. The manual techniques for the test-sequence derivation and test-bench generation are error-prone and require many iterations resulting in long turnaround times.

SOC designs in 40-nm and 32-nm technology may also have several different IP blocks interconnected to SerDes, CPUs, PLLs, and BIST and repair logic. The connections could range from a few hundred to thousands in an SOC with hundreds of memories. A typical networking chip in 40 nm could have 1000 to 2000 memories, and the memory-BIST logic shares local controllers with memories and top-level controllers with daisy-chain connections to the repair logic and fuse box. The static-verification solution needed to verify these connection scenarios should be able to address the following conditions:

- a point-to-point connection exists on buffered paths,
• paths are sensitized through design constraints on ports or internal nodes,
• paths are sensitizable through design constraints,
• nodes achieve a particular value at the end of the test sequence, and
• nodes achieve a particular value as a result of a static configuration.

Functional-design verification scenario

In Figure 2, the CPU, XBAR, and memory logic modules have been designed as stand-alone elements. Top-level verification is needed to ensure that all the blocks talk together with the desired performance.

![Figure 2. In this SOC-design connectivity scenario for functional modes, connectivity issues might exist due to human errors as the RTL code is assembled manually.](image)

In the Figure 2 SOC design example, connectivity issues might exist due to human errors as the RTL code is assembled manually. The functional design-operation mode works based on the XBAR_CTRL[0..n] signal value. It is much more efficient to first verify that each XBAR_CTRL configuration connects the desired memory and the CPU, with both address and data pins in the correct order, and then run the functional verifications to ensure that the subsystem works as expected. In case of errors in connectivity, no time is wasted running long simulations that might fail after several thousand clock cycles. In the case of functional errors, the bug must be functional if connectivity is proved correct.

By using static analysis for connectivity checks, functional simulations and regressions can be streamlined. Static checks first verify that the new version of the RTL code or netlist did not change the desired configuration; then simulations verify the functional behavior. Often designs contain very long serial control registers for functional purposes, too, and related problems can be avoided initially by establishing correct connectivity.

Scan-chain connectivity verification

Place-and-route tools change the order of the scan flops to minimize congestion. In this process, scan chains might be disconnected and lockup latches removed, thereby affecting test-pattern generation or memory-BIST functionality.

Hence, another requirement is for static extraction of the list of flops connected between scan-in and scan-out ports. Static scan-chain integrity checks allow for immediate verification that both the order and number of flops is as expected for different revisions of the netlist.
As an example, scan-chain integrity checks can help in memory-BIST validation. The integrity of the memory-BIST chains can only be verified after the BIST algorithm is complete. Multiple dynamic simulation tests are typically needed to make sure that all complete pass/fail scenarios are properly verified. If the serial chain can be traced with static methods, it is known for sure that all BIST engines are indeed present and connected in the proper order and that the correct test-data register has been selected inside the BIST wrapper.

**Static verification on an ASIC**

**Figure 3** illustrates an example of a clock-connectivity mistake. In Figure 3, the JTAG_IF circuitry has no direct control over the AND cell that gates the functional PCI_CLK signal to the BIST engine.

![Figure 3](image)

Figure 3. Illustrating a clock-connectivity mistake with regard to starting the BIST engine, the JTAG_IF circuitry has no direct control over the AND cell that gates the functional PCI_CLK signal.

Here is how such a mistake might occur:

- When running the BIST test bench, the designer will most likely use a generic test bench in which PCI_RST_L is active.
- The PCI_RST_L signal will in turn set the flop controlling the clock gate, thus enabling the BIST_CLK to reach the BIST engine.
- JTAG, however, does not have complete control on the BIST_CLK.
- In the field, if JTAG attempts to run the BIST engine after an initialization procedure that disabled the BIST_CLK enable flop, the BIST engine will be stuck because there is no clock present.

This kind of bug requires lots of time to find and fix even if it is caught with dynamic simulation.

STMicroelectronics used Atrenta's SpyGlass-DFT tool in a real project (called M) undertaken for an ASIC customer to test several categories of possible connection failures. The design had a TMGR (test-manager) block that configured the chip into different test modes. This TMGR block was developed at STMicroelectronics and sent to the customer.

The initial versions of TMGR were just empty shells with feed-through wires. New features and additional controls were implemented during the project life. The TMGR block was partly coded by hand and partly implemented via custom-prepared scripts. Most of the code implemented dedicated multiplexing of control signals for each test mode. The checks with SpyGlass-DFT allowed an
independent evaluation of each intended configuration, without actually having to review the complete RTL or gate-level design.

In this project, the various ATPG test-mode configurations were verified, beginning with performing the following three checks:

• Is TM = 0 really sufficient to guarantee no test mode is selected?
• Are all the analog macros (such as a fuse box and thermal sensor) really protected (power down asserted) when the chip is in ATPG mode?
• Is each control/logic value listed in the test spec implemented correctly?

The next step involved the override macro control, in which TMGR ensured that JTAG could override functional control of the PLL, fuse box, or thermal sensor. This step involved the following checks:

• Set the chip in functional mode (set pad TM = 0): Are all the macros controlled by the functional inputs?
• JTAG takes ownership of macro X (by setting the proper JTAG user-defined register to 1): Are all relevant pins of the macro being controlled exclusively by the proper bit in the user-defined register?

The next step involved the STT (scan-through TAP), a test mode in which all flops are connected in a scan chain. STT is controlled exclusively through the TAP controller. This step involved the following checks:

• Is the instruction decode sufficient to put the chip in STT mode?
• Does the STT mode propagate the correct clock?
• In STT mode, is the clock only enabled during shift?
• In STT mode, are all CMOS IOs configured as three-state outputs?

The next step involved macro connectivity (via the TAP controller). The device contains macros that are instantiated several hundred times. For example, every BIST engine has several user-defined register segments, each selected by the appropriate instruction decode signal. Thus, the following checks were required:

• Is each instruction decode signal connected to the correct pin of all modules that contain a segment of the shift register being selected?
• Are the shift-enable and capture signals connected to the correct pins of all modules that contain any segment of a shift register?

The final step involved macros that were connected serially (multiple segments of a shift register):

• Have the macros been connected in the expected order?
• Have any macros been left unconnected and has any incorrect reconvergence taken place?

**Static verification**
The SpyGlass-DFT static-analysis tool was used both at RTL and the netlist stage. This approach can formally verify the following conditions:

- Connectivity exists between a source and a destination node, and there is a possible path between source and destination. Combinational logic in between is allowed, as long as constants do not block the path.

- Direct connectivity exists between a source and a destination. In between the two nodes, there is nothing apart from buffers and inverters with no change to polarity.

- Enabled connectivity exists between a source and a destination. Any combinational logic in between is configured by constant propagation to allow only the selected path to propagate.

The SOC_02 rule was used to check for connectivity and is shown in Figure 4.

![Figure 4. The SOC_02 rule can verify enabled connectivity on source and destination pins.](image)

The SOC_01 rule was used to verify that required constant values occurred with specific test constraints. The value of the constant on an internal node was checked as shown in Figure 5.

![Figure 5. SOC_01 rule was used to verify the required value on internal nodes.](image)
The Scan_24 rule was used to verify for correct scan chains and no flops were missing from the chain (Figure 6).

In conclusion, verifying the connectivity of SOC designs at RTL and netlist for both functional and test modes with complex configurations and DFT architectures presents challenges. Fortunately, those challenges can be addressed through a process that provides for quick validation of the connectivity in both functional and test modes, thereby reducing the number of dynamic simulation test cases and improving overall productivity.

Marco Brambilla is a design manager at STMicroelectronics with 12 years of experience in the design and validation of complex SOC for wired communication products. Brambilla holds an MSEE from the University of Pavia in Pavia, Italy. marco.brambilla@st.com.

Jean Philippe Loison is a design engineer at STMicroelectronics with 20 years of experience in the design and validation of complex SOC for wired and wireless communication products. Loison holds an MSEE from the University of Nice, France. jean-philippe.loison@st.com.

Kiran Vittal is a product marketing director at Atrenta with 20 years of experience in EDA and semiconductor design. Prior to joining Atrenta, he held engineering, field applications, and product marketing positions at Synopsys, ViewLogic, and Mentor Graphics. Vittal holds an MBA from Santa Clara University and a bachelor’s degree in electronics engineering he earned in India. kvittal@atrenta.com.