OCI: Open Compression Interface

Bruce Cory, Rohit Kapur, Mick Tegethoff, Mark Kassab, Brion Keller, Kee Sup Kim, Dwayne Burek, Steve Oakland, and Benoit Nadeau-Dostie

Abstract

Before on-chip scan compression, it was possible to use different EDA tool vendors to do scan insertion, pattern generation, and diagnosis. On-chip scan compression changed that use model since each tool vendor supplied a different type of scan compression logic and had tool-specific ways to pass necessary information from scan insertion to pattern generation and from pattern generation to diagnosis. OCI (Open Compression Interface) is a standardization of how the necessary data is passed from test logic insertion to pattern generation to diagnosis such that different vendors can be used for each step independent of the on-chip scan compression logic used. This document discusses the need for OCI and gives a conceptual overview of the OCI standard.

1. Introduction

As technology shrinks, more and more logic is able to fit into the same die area. While a few design companies are shrinking the size of their die, most are combining multiple functionalities or increasing the power of the functionality available and keeping the total die size the same or even larger. This, plus the need for more types of test patterns to maintain quality of test levels at lower technology nodes, is causing the amount of test data needed to verify that the parts are manufactured correctly to skyrocket. The rate is far greater than the growth of tester memory can economically accommodate and also causes issues with the time required to test each part. The main way used to handle these problems is to use more and more complex on-chip scan compression logic. The use of on-chip scan compression logic has been steadily growing for the last several years.

Due to growth in demand for on-chip scan compression, solutions are available from most EDA test vendors. Some example products include OPMISR, TestKompress, DFT compiler Max, VirtualScan, and ELT-Comp. Some design companies have published papers with details of their in-house compression schemes. One example is X-compact. Academic papers on test compression have been published for many years with many of the concepts being used by industry. One example of this is Illinois scan. Each of these compression schemes has similar concepts but different implementation details.

Flows for scan-based test tools are broken into three main stages. Test logic insertion, pattern generation, and diagnosis. Test logic insertion does insertion and verification of test logic. Pattern generation uses the test logic to make test patterns that can be used to verify if the design is fabricated correctly. Diagnosis is used to identify the failing location in a specific device. Diagnosis information can be used to increase future yield and to solve problems that keep a design from going to market.

Most on-chip scan compression schemes are only supported by a limited number of test vendors with most only supported by one vendor. This causes many issues in the test community. Figure 1 shows one example of the issues caused by vendor-specific on-chip scan compression logic and flows.

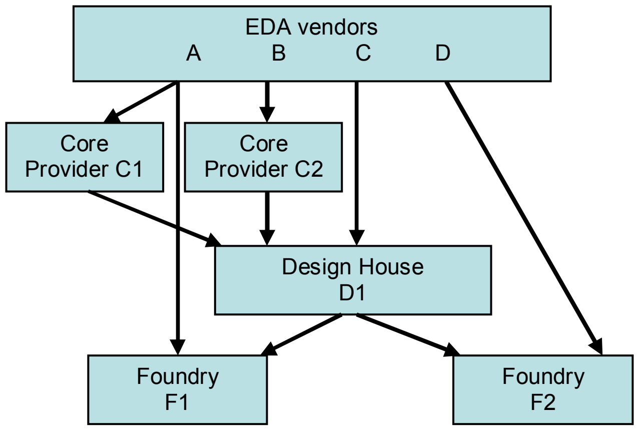

Figure 1: Example use model when on-chip scan compression is not interoperable

Companies A, B, C, and D each have different compression logic and flows only supported by their tool. Hard core provider C1 requires in-core scan compression logic using tool A while hard core provider C2 requires in-core scan compression logic using tool B. Both core developers need to own one tool to insert and verify the on-chip scan compression logic. The design house (D1) is making a design using core C1, C2, plus other in-house developed logic and uses tool C for its in-house on-chip scan compression insertion, pattern generation, and diagnosis. The design house must own three vendor tools to do pattern generation and diagnosis for the different compression structures of the design. Design house D1 then plans to produce the design using foundries F1 and F2. F1 uses tool A for diagnosis while F2 uses tool D for diagnosis. Foundries F1 and F2 need to support 3 or 4 different tools for diagnosis. The quality tradeoffs, support burden, design size overhead, and expense incurred by the design house and foundry because tool C and D can't do pattern generation and diagnosis using the

1-4244-0292-1/06/$20.00 © 2006 IEEE

test structures of tool A, B, and C are high. The core providers also may need to provide their cores with multiple types of scan compression to avoid losing business.

Although the above structure may seem appealing to EDA vendors, there are many drawbacks as well. Once a design house buys a specific vendors on-chip scan compression tool, it is very unlikely to buy any other vendors tool. This inhibits competition and the ability of an EDA company to find ways to gain more market share.

OCI defines a standard way of passing information from test logic insertion to pattern generation and from pattern generation to diagnosis such that interoperability of EDA tools is possible. Interoperability of test logic insertion is not supported by OCI.

Section 2.1 of this document will discuss conceptually the type of information OCI passes from test logic insertion to pattern generation and from pattern generation to diagnosis to enable interoperability. Section 2.2 goes one step deeper and discusses how STIL and CTL are leveraged and extended to pass the conceptual data discussed in section 2.1.

2. OCI Overview

2.1 Conceptual Data Flow

Figure 2 shows the OCI conceptual data flow from test logic insertion to pattern generation and from pattern generation to diagnosis. Both the OCI flow (assuming different vendor for each stage) and the current flow (same vendor for each stage) are shown. Since same vendor tools have their own way of linking different stages of the flow together, OCI isn't needed in situations when changing stages unless any previous step was done by a different vendor. Then all data OCI would normally add in the previous stages is needed.

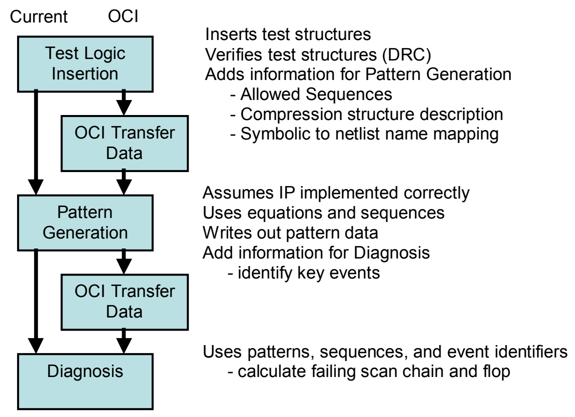
Current
OCI
Test Logic Insertion
OCI Transfer Data
Pattern Generation
OCI Transfer Data
Diagnosis
Inserts test structures
Verifies test structures (DRC)
Adds information for Pattern Generation
- Allowed Sequences
- Compression structure description
- Symbolic to netlist name mapping
Assumes IP implemented correctly
Uses equations and sequences
Writes out pattern data
Add information for Diagnosis
- identify key events
Uses patterns, sequences, and event identifiers
- calculate failing scan chain and flop

Figure 2: Conceptual Flow Used By OCI Interface

The test logic insertion stage has the best understanding of the test structure being implemented. Test logic insertion tools are developed based on specific compression structures and have many special design rule checks implemented so that customers have as good a chance as possible to generate working, highly effective patterns the first time. The time and knowledge necessary to develop all of the design rule checks is significant. Due to this, the OCI flow assumes that the pattern generation tool does not need to perform test structure verification. In the case where a design house is integrating a core with in-house design logic, the test structure in the core is assumed to be verified by the core provider and all core level OCI information needed for pattern generation will be in the OCI compliant STIL file the core provider gives the design house. The design house integrating the core is responsible for generating and verifying the final OCI information passed to pattern generation. The OCI specific information passed from test logic insertion to pattern generation includes pattern sequence restrictions, a description of compression hardware, netlist mapping points, and vendor-specific data to ensure that single vendor compression flows do not require worse ease of use when using OCI. A more detailed description of how CTL is leveraged to pass information from test logic insertion to pattern generation is in section 2.2.

The pattern generation stage can be very time consuming. The ATPG (automatic test pattern generator) determines which logic values are needed on scan cells to detect the most faulty device locations for each test pattern. After a pattern is generated, it is then simulated to determine how many faulty locations have been detected. This sequence is repeated until as many as possible faulty locations have been detected. For chips without on-chip scan compression, the value loaded and unloaded from each scan cell can be mapped directly to a unique value loaded into the chip and to a unique value loaded out of the chip.

Only sequence definitions that describe how to initialize, load, unload, and apply tests are needed by pattern generation for this case. On-chip scan compression needs more information. Putting a logic value in a scan cell requires a value to be loaded into the chip at a previous point in time and will likely force other scan cells to be put to certain logic values. Checking that a value is in a scan cell may require values in masking logic or on other scan cells. Also, on-chip scan compression hardware may restrict the length of a sequence or require a relationship between sequences because of how the hardware is implemented. To do pattern generation efficiently, OCI passes to pattern generation sequences, sequence limitations/relationships, and symbolic descriptions of the compression hardware created by the test logic insertion stage.

The output test patterns generated by pattern generation need to be diagnosable using both EDA diagnosis, that can isolate to a gate or net, and on the tester, where isolation is possible to the failing scan cell or scan chain. To enable this capability, the pattern generation stage must identify key events in the pattern data and pass this information to diagnosis. The information is passed through OCI by updating the sequences and/or by adding information to the test pattern data. A more detailed description of how CTL is leveraged to pass information from pattern generation to diagnosis is in section 2.2.

The diagnosis stage uses the key event information added by pattern generation to map each ATE failure back to a list of scan flops that might have caused that failure. Then the diagnosis tool uses these lists to determine which flops are most likely to have failed and which gates or nets are most likely to have caused those flops to fail. The diagnosis stage doesn't add any new information to the OCI file.

2.2 High level Implementation Details

OCI data is passed from stage to stage to allow tool interoperability. The format of the OCI data passed leverages CTL (IEEE 1450.6 standard) as a foundation with some new syntax and semantics added where needed.

Figure 3 shows the information in an OCI file after test logic insertion.

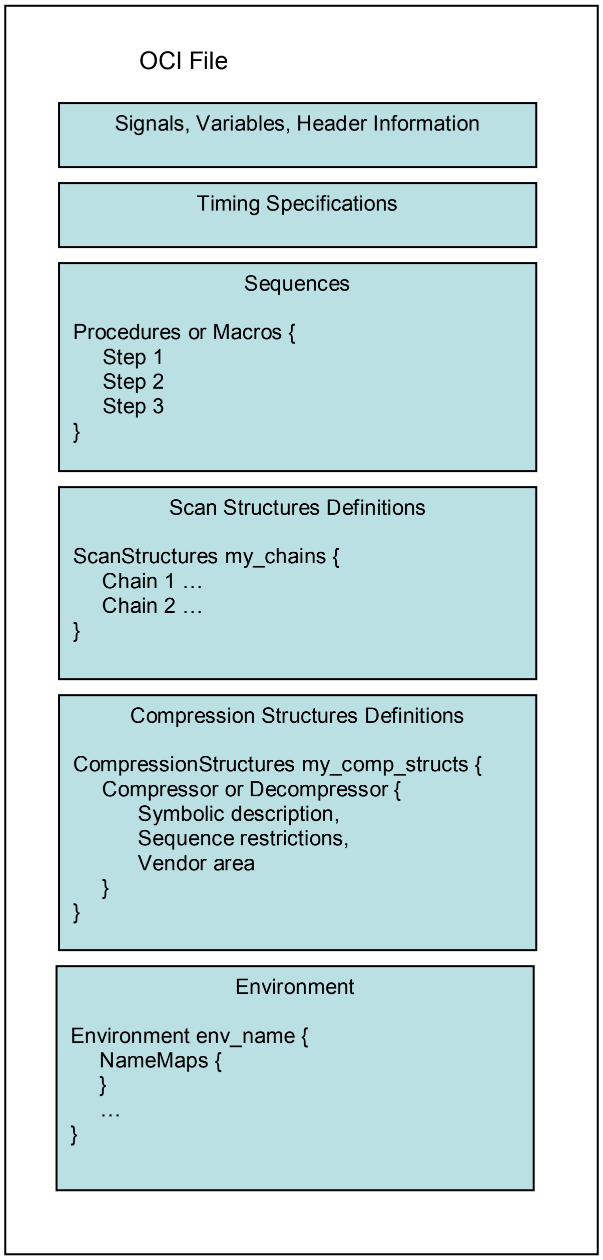

Figure 3: OCI File Structure after Logic Insertion

The goal is to provide syntax such that tools support a uniform interface and no special implementation is needed for the various compression technologies. This is achieved by using sequences to handle the complexities of applying the pattern data and compression equations to handle the data relationships between scan cells and the inputs and outputs of the design. The sequences are procedures and macros provided by the compression technology implementer, while the pattern data provides the data to the sequences. The data relationship of the scan data is provided in equations in the compression structure block. Pattern generation uses the equations to translate the scan data to the scan-in and scan-outs and provide the patterns that invoke the existing macros and procedures with the data passed to them.

The environment syntax allows for defining the compression structures for different modes allowing for a mix and match of multiple vendor solutions as needed.

3. Summary

This document discusses the need for OCI and has given a conceptual overview of how the OCI standard works. The Accellera OCI standard development work has shown that interoperability of tools between test logic insertion, pattern generation, and diagnosis is possible. After ratification of the OCI standard by Accellera this year, OCI will become an IEEE 1450.x working group and will move towards becoming an IEEE standard.

4. Acknowledgements

We would like to thank Johny Srouji from IBM and Victor Berman from Cadence for their strong support and help. Johny is the technical committee chair of Accellera and Victor is our Accellera board of directors representative.

5. References

[1] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller and B. Koenemann, "OPMISR: The Foundation for Compressed ATPG Vectors," Proceedings of the International Test Conference, 2001, pp. 748-757.

[2] I. Hamzaoglu and J. H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," Proceedings of the International Symposium on Fault Tolerant Computing, 1999, pp. 260-267.

[3] A. Jas and N. A. Touba, "Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs," Proceedings of the International Test Conference, 1998, pp. 458-464.

[4] A. Jas, B. Pouya and N. A. Touba, "Virtual Scan Chains: A means for reducing scan length in cores," Proceedings of the VLSI Test Symposium, 2000, pp. 73-78.

[5] A. R. Pandey and J. H. Patel, "Reconfiguration Technique for Reducing Test Time and Test Data Volume in Illinois Scan Architecture Based Designs," Proceedings of the VLSI Test Symposium, 2002, pp. 9-15.

[6] J. Rajski, J. Tyszer, M. Kassab, N. Mukerjee, R. Thompson, K. H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide and J. Qian, "Embedded Deterministic Test for Low Cost Manufacturing Test," Proceedings of the International Test Conference, 2002, pp. 301-310.

[7] S. Samaranayake, N. Sitchinava, R. Kapur, M. B. Amin and T. W. Williams, "Dynamic Scan: Driving Down the Cost of Test," IEEE Computer, October 2002, pp. 63-68.

[8] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, T.W. Williams, "A Reconfigurable Shared Scan-In Architecture", Proceedings of the VLSI Test Symposium, 2003, pp. 9-14.

[9] IEEE Standard 1450.6-2005, "Core Test Language".