IP/SOC 2005

Session: Industrial Verification and Methodology

A Unified DFT Verification Methodology

Stylianos Diamantidis, Iraklis Diamantidis and Thanasis Oikonomou,
Globetech Solutions
Thessaloniki Greece

Abstract :

In today's fast growing Systems-on-Chip (SoC), incomplete or ineffective DFT support due to poor specification or loose design practices can quickly become the critical path to making market windows and delivering products within cost restrictions.

This paper will introduce a unified DFT Verification Methodology, aimed at providing a complete, methodical and fully automated path from test specification to DFT closure. We will also examine the benefits of this approach, looking at how this methodology can help bridge the widening gap between design and test.

I. INTRODUCTION

As modern IC transistor counts continue their frenzied climb according to Moore's Law, test infrastructures, the collection of logic dedicated to testing the structural integrity of silicon, are also fast growing in both area and complexity[1]. In a nanometer design era where silicon debug already takes up to 30% of project time and semiconductor test cost typically accounts for 30-50% of total fabrication cost, Design-For-Test, or DFT, is assuming a critical role in product definition, design and delivery.

Although DFT is a concept that has been around for a long time, semiconductor companies are today experiencing unprecedented pressure to provide more complex DFT features in their designs. This trend is largely attributed to the need for controllability and observability within highly integrated SoCs and is driven by the inevitabilities of test economics.

I-A. WHY DFT FAILS

Design verification tools and methodologies have made tremendous progress in the last few years, directly benefiting design quality and shortening development cycles. However, DFT-specific circuitry tends to be overlooked in most test plans. There have been a series of contributing factors for this oversight:

- ♦ No clean test intent is specified and communicated to the design teams
- ♦ Lack of formal end-goal or associated Quality of Result (QoR) for DFT
- ♦ Low prioritization compared to core functionality
- ♦ Loose IP-based design methodologies
- ♦ A clear cultural gap between design and test teams, including "over-the-wall" communication breakdowns

These and many more reasons are today resulting in typical DFT failures, manifested in a variety of forms:

- ♦ Lack of strict protocol compliance and loose interoperability
- ♦ Deviation from strict functional behavior for test implementation (e.g. accuracy in scan based delay-path test setup and extraction)
- ♦ Poor testability coverage due to logical errors in the implementation (e.g. inability to access BIST controllers or error status reporting registers)
- ♦ Decrease in test efficiency (time, test data set size) due to non-coherent test implementation

I-B. MOTIVATION

DFT failures due to loose design practices, however, have been commonplace throughout the history of modern IC design. What has changed recently to accentuate the problem? The answer lies in the inevitabilities of test economics. Cost of Test (COT) in the nanometer era is breaking semiconductor economics:

- ♦ COT does not scale. Although silicon fab costs have been steadily decreasing to accommodate industry needs, the capital costs of testing wafers have remained flat [*1*][*2*]

[1] Already reaching 20% of total gates in some SoCs

- Large Automated Test Equipment (ATE) system cost is driving capital COT, due to complexity of modern SoCs
- Exploding test time and test vector sets combined with low yield are putting imense pressure on COT

In order to deal with the inevitabilities of COT, the industry is beginning to turn to massive DFT implementations :

- Enable low-cost tester deployment by partitioning test resources on-chip
- Design scalability into test schemes
- Increase controllability/observability for silicon debug
- Implement on-chip instrumentation

These trends are leading to highly complex and sophisticated DFT structures. However, associated methodologies and design practices have not yet caught on to this pressure:

- Although the industry is transitioning to IP-based design to tackle complexity, test is still very flat
- Developing ecosystem of IP vendors and integrators is leading to more heterogeneous and unpredictable test infrastructures
- DFT insertion at different levels of abstraction (RTL, gate, physical) is increasing unpredictability and making it difficult to define QoR requirements
- Test infrastructures are inherently heterogeneous. IP-based design places a new requirement to build coherent system level test schemes from incoherent components

DFT has hence become too important to treat as a secondary design function and too complex to tackle with traditional approaches. Instead, design teams need to take special care to ensure the behavioral functionality, strict compliance and efficient operation of their test infrastructures. As silicon test transitions from a design afterthought to a critical manufacturability requirement, companies need to rediscover "Design" in DFT. We start with *verification*.

II. A UNIFIED DFT VERIFICATION METHODOLOGY

In trying to design a complete DFT verification environment [*3*] and associated methodology, one needs to define the key objectives this approach is trying to achieve:

- A well-defined entry point into the design process that can be used as the foundation for expressing test intent and expected end QoR
- Mechanisms for verifying classes of DFT components which will handle the stimuli generation and checking aspects of testing at different levels of abstraction
- Flows for deploying and executing verification as well as measuring progress
- Tracking and analyzing results
- High-levels of automation and reuse
- Integration of Test Information Models in the verification flow
- Methods for exchanging information with post-silicon applications such as debug and test

We have hence designed a robust, unified, DFT Verification Methodology (DFT-VM). Keeping the stated objectives in mind, we now proceed to describe the methodology based upon three distinct foundations: Planning, executing and automating.

II-A. PLANNING

The foundation for systematic DFT verification is a well-defined set of goals, supported by a methodology developed to provide integration-oriented test methods into chip-level DFT, enabling compatibility across different embedded cores and incorporating high levels of reuse.

But how can one proactively plan for virtually arbitrary DFT implementations that can be produced by IP-based design, particularly when different vendors follow completely different approaches to DFT? Obviously test plans need to be very modular and reusable, allowing for hierarchical structures to be easily constructed to describe the test infrastructure at hand. Furthermore, test plans need to be polymorphic, very much in the way that object-oriented methodologies define classes of objects, making it possible to use them in a variety of different forms and shapes by specifying simple parameters.

In our solution, we specify a *Plan Case Database*, a repository of plan templates, or *cases*. Such cases contain policies for verifying DFT components such as a JTAG TAP controller [*4*], without making any assumptions for the non-standard or implementation-specific aspects of the components. DFT planning cases have the following characteristics:

- They provide blueprints for verifying classes of DFT components
- They specify QoR metrics that verifiers can use to track progress against the plan
- They allow different views into the verification plan data to be specified, allowing for better analysis of results

Planning cases can be used to instrument the verification of both rudimentary DFT components as well as highly complex structures. This is achieved by *dynamic planning*, the process of hierarchically piecing together a high level verification plan from lower-level plans (see Figure 1). This modularity enables the quick and repeatable composition of detailed verification plans for arbitrary DFT infrastructures at the block, core or system levels. Users can spend time experimenting with these high level plans for optimum results, setting the blueprint for a well designed test infrastructure before a single design decision has been made. The ability to reuse plans at different levels of integration and abstraction is a huge benefit to the predictability and verifiability of the project.

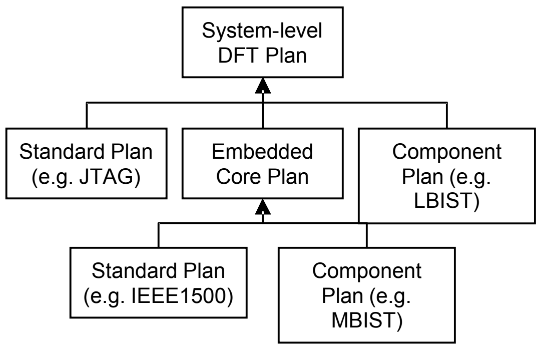
System-level DFT Plan
Standard Plan (e.g. JTAG)
Embedded Core Plan
Component Plan (e.g. LBIST)
Standard Plan (e.g. IEEE1500)
Component Plan (e.g. MBIST)

Figure 1: Hierarchical DFT Verification Planning

Having compiled a plan case database, we now have the necessary building blocks for expressing high-level, complex, and, most importantly, highly configurable verification plans, maximizing reuse and leveraging on existing experience. Building a dynamic chip-level DFT verification plan is now broken down to instantiating and configuring multiple DFT verification case objects.

II-B. EXECUTING

Once the critical task of planning has been properly addressed, the verification environment needs a scalable way of executing verification on the test infrastructure. The most complete and reusable way to achieve this is by deploying *Verification IP* (VIP).

The concept of verification IP is fairly new in the design community. Conceptually, VIP provides a way of separating generic concepts of design verification from application-specific ones. When this separation is well designed, the direct benefits are enhanced reuse and leverage on existing experience. In the context of DFT, generic concepts can include generating pseudo-random vectors and driving them into a scan chain. Application-specific concepts, for instance, could include using this scan chain to configure a Memory Built-In Self Test (M-BIST) controller [*5*].

Essentially, VIP is *mechanism*. It provides the means and capabilities to perform operations and observe DUT behavior; however, it does not include policy. Policy, in this context, is defined as the systematic flow of verifying a complex design, starting with a detailed set of goals, adding a plan of action and targeting a certain quality of result. Hence, starting with a good policy, we can reach our goals by deploying VIP as our mechanism.

We hence define a *VIP Class Database*. This database includes VIP classes which map to types of DFT components such as Test Access Mechanisms (TAMs), scan chain elements, BIST controllers, instruments, etc. Each VIP class includes the tools needed by the verification environment to effectively exercise its corresponding type of DFT logic [*6*]:

- ♦ Constrained-random stimuli generators
- ♦ Automated, dynamic, checkers and assertions
- ♦ Total coverage collectors

As with plan cases, VIP classes do not include application-specific or implementation-dependent aspects of the DFT component types they target. Rather, they are rudimentary verification environments which are highly reconfigurable and reusable, making it easy to put together complex environments in relatively small time and with reduced effort. Furthermore, the VIP class database becomes an experience repository for DFT, where periodic updates ensure uniform design policies and improved interoperability.

Finally, such a repository also helps improve resource utilization and project management. Expert verification engineers can maintain and extend the repository with upgraded capabilities and new functionality while logic designers, usually not entirely familiar with the internal workings of the VIP itself, can simply use the platforms based on its capabilities. Conversely, using this methodology, logic designers can ensure that new features or design changes added directly into VIP classes are made available instantly by regenerating the environment. This enhanced automation of the DFT-VM is discussed next.

II-C. AUTOMATING

Test Information Models (TIMs) are schemas used to convey information about the test infrastructure of an IC or embedded core as well as to describe complete test programs that it can execute. TIMs serve the purpose of delivering test vectors generated using EDA tools to semiconductor testers (ATEs). The IEEE 1450-1999 Standard Test Interface Language (STIL) [7] is quickly becoming the de-facto standard. Recent extensions to TIM standards (IEEE 1450.1-2005 [*8*], IEEE P1450.6 [*9*]) support additional structures in test models to fully describe the DFT architecture itself, hence enhancing the use of such models in semiconductor design environments. These extensions are targeted at enhanced DFT and DFM applications, where ATEs can also be used for analyzing failure data and providing feedback to EDA tools.

In our DFT Verification Methodology, TIMs play a significant role. First, TIMs need to be considered a part of the test infrastructure itself. In fact, the recently published IEEE 1500-2005 Standard for Embedded Core Test (SECT) [*10*] defines a TIM as the only mandatory test infrastructure element for claiming that an embedded core is compliant to the standard. Based on the test intent described in the TIM, designers can provide the necessary functionality while maintaining flexibility in the actual hardware implementation. Hence, TIMs *need to be verified* alongside the DFT components that implement them.

Secondly, TIMs include all the necessary topology, architecture and implementation specific information that must be available to the verification environment. This way, a silicon IP vendor can communicate test intent of a design core to an integrator within specified completeness, interoperability and confidentiality requirements. This information, in the form of a TIM, can then be used by the integrator for a variety of design functions ranging from implementing certain DFT components to shaping the IC-level test infrastructure.

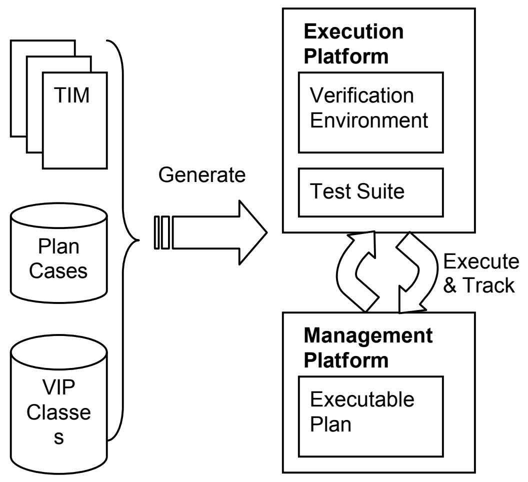

Figure 2: DFT-VM Automation Flow

In our approach, we are extending the applicability of TIMs to design verification, claiming that TIMs can provide an automation link between DFT design and verification. The argument is supported by a variety of technical and business conditions:

- ♦ TIMs can encompass test intent as specified by test engineers without committing to design decisions and hence provide the grounds for an early test specification
- ♦ TIMs bridge the gap between IP vendors and integrators with respect to DFT support in IP cores and hence can be used to verify deliverables
- ♦ TIMs are models that can be used for early test performance exploration
- ♦ Verification automation based on TIMs can be used to maintain a link between the post and pre silicon worlds, allowing testbenches to be reused for debugging silicon and optimizing manufacturing test

In order to better understand how TIMs can be used to build DFT verification environments quickly and effectively, let's consider the following example in the context of an IEEE 1500-2005 compliant embedded core.

III. TEST CASE

IEEE 1500-2005 (SECT) defines a scalable architecture for independent, modular test development and test application for embedded design blocks. It also enables test of the external logic surrounding these cores. Modular testing is typically a requirement for embedded non-logic blocks, such as memories, and for embedded, pre-designed, non-mergeable IP cores. In addition, the IEEE 1500 architecture can also be used to partition large design blocks into smaller blocks of more manageable size and to facilitate test reuse for blocks that are reused from one system-on-chip (SoC) design to the next.

A typical TIM is that of an IEEE 1450.6 Core Test Language (CTL) description of the IEEE 1500 test infrastructure, commonly referred to as a wrapper, found in a SECT compliant embedded core (for more information please refer to the IEEE 1500 standard). Such a model includes, amongst others, information in a parse-able format about:

- ♦ **Signals**
  The TIM publishes information about signal names and sizes, as well as their default state, so they can be initialized and driven/sampled by an external agent.
- ♦ **Scan Chains**
  This information refers to scan structures that are part of DFT. That includes scan chain sizes and cells' names, so other information regarding the cells like parallel inputs and/or output connecting signal names can also be inferred.

- ♦ **Scan Cells**
  TIMs publish all information pertaining to the scan chain cells, since IEEE 1500 cells follow a standard naming convention that fully describes their structure and function.
- ♦ **Test modes**
  The TIM also includes information about the various test modes that can be reached by loading appropriate instructions. It provides the instruction opcode that triggers this mode, identifies the data register to be used, and provides the macros used to access it.

A TIM parser can parse all this information and infer:

- ♦ IEEE 1500 control signals
  - o The Instruction Set Used
  - o Opcodes
  - o Data registers referenced
- ♦ The collection of test data registers
  - o Sizes
  - o Signal connections
- ♦ The cells contained in those registers
  - o Structure
  - o Signal connections
  - o Behavior during capture, update, transfer operations

As an example, Figure 3 below illustrates an example of a CTL description for an IEEE 1500 compliant embedded core wrapper. The description provides information about the wrapper, including the size and cell type of the instruction register (WIR).

```
// CTL for IEEE 1500 enabled embedded
   core wrapper
...
ScanChain wir_chain{
  ScanLength 4;
  ScanCells wcell[0..3];
}
...
  ...
  Internal{
    'wir_fi[0..3]{ DataType Functional
                   TestData;
      IsConnected In {
        StateElement Scan 'wcell[0..3]';
        ..
      }
    }
    ...
  }
  ...
}
```

Figure 3: IEEE 1450.6-CTL DFT Structure Example

Having extracted these structures from the CTL model, one can envision a process by which:

- ♦ The corresponding IEEE 1500 VIP class is selected from the VIP database and instantiated (see Figure 2, above)
- ♦ The number of cells specified in the WIR structure and signals connecting to the parallel input of those cells are used to configure the VIP (see Figure 4, below)

```
// 'e' language configuration for an
   IEEE 1500 VIP module

extend WIR glbt_sect_ref_model_register
{
  keep size == 4;
};

extend WIR
glbt_sect_ref_mode_register_cell
{
  keep cfi == append("wir_fi", "[",
                     cell_index, "]");
};
```

Figure 4: Environment generation based on a TIM

Collections of TIMs can be grouped together hierarchically to perform system level DFT verification. This can be done by analyzing the TIMs and deducing the respective topology of each embedded core and its corresponding DFT infrastructure in the SoC. With this information, DFT-VM can be used to dynamically create test-benches and tests optimized for a specific DFT configuration.

IV. BENEFITS

The described verification methodology serves as a solid foundation for true Design for Test. By enforcing early verification documentation and planning, it aligns the perspective of different design teams with respect to DFT support and enhances visibility. Automating environment generation, it ensures that logic designers and test engineers have a good auditing system for debugging and regression analysis, while propagation of new features and updates is centralized through the use of plan and VIP databases. Better project management and more efficient resource utilization are also achieved by providing clear interfaces for logic designers and verification engineers.

TIM co-verification introduces strong semantics into the description and integration of test infrastructures. DFT designed by separate teams or IP vendors can be merged into the IC-level reliably, while maintaining a link with manufacturing test deliverables. Architectural changes to DFT can quickly propagate to the design environment through fast regeneration and automatic plan updates. Vendor qualification for DFT becomes possible by enforcing TIM deliverables and being able to quickly and reliably validate vendor claims for testability and interoperability. Finally, advanced DFM applications can also be supported through early collaboration with the fabrication and tester providers.

Enhanced design engineering, automation and reuse lead to increased predictability, better productivity and higher overall quality.

IV-A. FUTURE WORK

Having successfully applied DFT-VM to functional verification, we are envisioning several other areas where the methodology can scale.

- **Post-silicon test and validation**
  Having created a completely DFT aware verification environment with QoR measurements and associated test sets, post-silicon validation of DFT becomes a natural step in the methodology. Validating DFT in silicon in a systematic and predictable manner can help save test time and improve reliability in manufacturing test.
- **DFM applications**
  We are also planning to explore some interesting DFM applications such as importing TIM-based test results from silicon test back to verification to gain better understanding of the functionality perspective of common failures and to facilitate analysis and debug.

V. CONCLUSIONS

We have identified the need to systematically verify DFT as part of the total system verification process, in order to increase the quality of the design and by virtue, the end product. The need becomes more apparent in the context of IP-based design of SoCs, where multiple embedded cores from different providers introduce heterogeneity and variation of DFT quality.

We have hence proposed a unified methodology for DFT verification, using Test Information Models to dynamically identify, instantiate and configure executable verification plans and environments. Hence we provide a fast and reliable way to building automated test-benches capable of verifying DFT designs from simple components to complete test infrastructures. Our approach enables true design for test based on measurable quality of result, and enhances productivity, reliability and reusability.

Finally we have demonstrated how our methodology results in a verification infrastructure that can be reused during silicon debug and test vector design for several advanced applications, forming the basis for future work.

VI. REFERENCES

[1] International Technology Roadmap for Semiconductors, 2003 Edition, “Test & Test Equipment"

[2] International Technology Roadmap for Semiconductors, 2004 Update, "Test & Test Equipment"

[3] K. Melocco, H. Arora, P. Setlak, G.Kunselman, and S. Mardhani, "A Comprehensive Approach to Assessing and Analyzing 1149.1 Test Logic," in the proceedings of International Test Conference, Charlotte, NC, USA, Sep 30 - Oct 2, 2003, pp. 358-367.

[4] IEEE Computer Society, "IEEE Standard Test Access Port and Boundary-Scan Architecture - IEEE Std. 1149.1-2001", New York: IEEE, 2001.

[5] D. Appello, F. Corno, M. Giovinetto, M. Rebaudengo, and M. Sonza Reorda, "A P1500 compliant BIST-based approach to embedded RAM Diagnosis," in the proceedings of 10th Asian Test Symposium, Kyoto, Japan, Nov. 19-21, 2001, pp. 97-102.

[6] I. Diamantidis, T. Oikonomou, and S. Diamantidis. "Towards an IEEE P1500 Verification Infrastructure: A Comprehensive Approach", presented at the 3rd IEEE International Workshop on Infrastructure IP (IIP), Santa Clara, CA, USA, May 4-5, 2005

[7] Test Technology Standards Committee of the IEEE Computer Society, "IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data - IEEE Std. 1450-1999", New York: IEEE 1999.

[8] P1450.1 Working Group of the Test Technology Standards Committee, "Draft Standard for Standard Test Interface Language (STIL) for Digital Test Vector Data - Extensions to STIL for Semiconductor Design Environments - P1450.1", New York: IEEE 2005.

[9] CTL Working Group of the Test Technology Standards Committee, "Draft Standard for Standard Test Interface Language (STIL) for Digital Test Vector Data - Core Test Language (CTL) - P1450.6/D1.6", New York: IEEE 2005.

[10] IEEE Computer Society, "IEEE Standard Testability Method for Embedded Core-based Integrated Circuits - IEEE Std. 1500-2005", New York: IEEE 2005.