Test Exploration and Validation Using Transaction Level Models

Michael A. Kochte, Christian G. Zoellin,
Michael E. Imhof, Rauf Salimi Khaligh,
Martin Radetzki, Hans-Joachim Wunderlich
University of Stuttgart
Institute of Computer Architecture and Computer Engineering
Pfaffenwaldring 47, 70569 Stuttgart, Germany

Stefano Di Carlo, Paolo Prinetto
Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24
I-10129 Torino TO, Italy

***Abstract*—The complexity of the test infrastructure and test strategies in systems-on-chip approaches the complexity of the functional design space. This paper presents test design space exploration and validation of test strategies and schedules using transaction level models (TLMs). Since many aspects of testing involve the transfer of a significant amount of test stimuli and responses, the communication-centric view of TLMs suits this purpose exceptionally well.**

***Index Terms*—Test of systems-on-chip, design-for-test, transaction level modeling**

I. Introduction

Test, yield and reliability are of great importance to the profitability, safety and security of current SoCs [1]. This has increased the significance of test infrastructure and other infrastructure not directly related to system functionality [2]. Similar to the functional design task, test, diagnosis and repair strategies for SoCs are very complex topics, requiring design space exploration and design verification [3].

In functional design, transaction level models (TLMs) are used to facilitate simulation-driven design space exploration and design verification [4]. The modularity and separation of communication and functionality in TLMs allow to quickly explore different implementation alternatives. Also, TLMs provide significant improvements in simulation performance by orders of magnitude. Still, they provide enough detail to make important design decisions regarding performance, die area and power [5, 6].

Even when the IP vendor provides the core's test strategy using IEEE Std 1500, there are numerous decisions to be made by the test engineer. For example, the choice among a large number of test data compression [7] and test access mechanisms (TAMs) [8]. Then, test scheduling tries to optimize the concurrency of tests, but the complexity of the scheduling problem requires that only very coarse information is taken into account. In order to gain accurate information regarding power and TAM utilization, the final schedule should be evaluated using simulation [9]. Furthermore, the final test program to be executed by the Automated Test Equipment (ATE) is a complex piece of software. Today, validation of the test program is facilitated by Virtual ATEs that can be simulated together with the RTL or gate level netlist of the SoC [10]. A complete test program usually requires several hundred million clock cycles which makes simulation of the whole test computationally expensive.

The communication-centric view of TLMs provides an abstraction that is well-suited to test, since test involves the exchange of significant amounts of data. Furthermore, performance modeling in TLMs tries to accurately capture the concurrency in a system, which is easily adapted to model the concurrency in testing. Initial work in this direction shows the relationship of design verification, design validation and test for TLMs [11]. For debug, the authors of [12] introduce debug transactions to deal with issues of concurrency during functional tracing and monitoring. The subsequent sections show how to model system test at the transaction level in order to facilitate test design space exploration, as well as the validation of test strategies and schedules.

The rest of the paper is organized as follows: Section 2 shows how Design-for-Test (DfT) properties can be modeled at high abstraction levels. Section 3 gives examples how TLMs of the most important test building blocks like test wrappers, TAMs, pattern generators and test controllers are implemented. In section 4, we show the benefits of test infrastructure TLMs using a case study with a JPEG encoder SoC.

II. Test Modeling at Transaction Level

Transaction level modeling [4] enables system level design and simulation of large hardware/software systems, for which RTL simulation would require an unacceptable amount of time. This is achieved by abstracting from signal level communication and by modeling complex communication operations as atomic transactions, thereby reducing the number of events to be processed by event-driven simulators as well as the number of context switches between simulation processes. Transactions are implemented as object-oriented methods (subprograms) that allow to modify the state of channels, which in turn represent the system's communication infrastructure [13].

In TLM, the functionality of system components is described separately from the communication-centric view of the system. This enables the user to change system components and communication models, to modify their interconnection patterns, and to change the mapping of application tasks to system components without much effort. Thereby, a quick exploration of system alternatives is supported. After identifying a suitable (optimized) system architecture, the initial TLM can be refined by adding details about functions and communications, leading to models more precise in terms of

978-3-9810801-5-5/DATE09 © 2009 EDAA

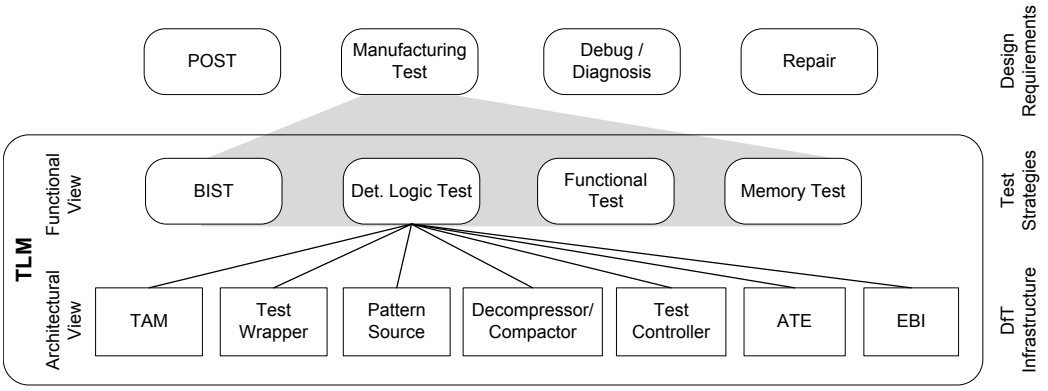

Fig. 1. Design-for-test modeling and refinement

behavior and timing, and ultimately to a cycle-accurate model that closely resembles an implementation [14].

SystemC provides the primitives required for TLM and supports the user with a standard library (TLM2.0) of interfaces and basic blocks for modeling bus-based SoC platforms. In this work, we aim at transferring TLM principles to the field of test infrastructure and architecture. Respective models are implemented based on the SystemC TLM primitives. The TLM2.0 library, however, has not been employed since TAMs require modeling of properties that go beyond those of SoC buses.

Current designs have several non-functional requirements such as high testability, design for manufacturing (DfM), built-in repair, debug and diagnosis, power-on self-test (POST), power-on-reset and -initialization (Figure 1). These properties require a significant amount of infrastructure to be integrated together with the mission logic. In system TLMs, this infrastructure is currently not reflected. In this work, we focus on transaction level modeling of the infrastructure and test strategies for manufacturing test.

The manufacturing test usually consists of several different tests according to the type of cores, test time, power budget, etc. Each core of a design has specific requirements for testing [15]. Cores with random logic may be tested using deterministic or random patterns applied through a large number of scan paths. This is often complemented with functional and in-the loop tests. Memory arrays are usually tested using highly tuned march tests that may be driven by a built-in self-test (BIST) controller or a processor.

These test functionalities are then mapped to a variety of structural building blocks, such as TAMs, test wrappers, pattern generators, external bus interfaces to the tester (EBI), decompressors, compactors, test controllers and the ATE. At this level, the architecture of the target system is defined together with the hardware/software partitioning of the function. For the test domain, the software part consists of the test program executed on the ATE, software modules executed on functional units like embedded processor cores, and the microcode to program the test controllers.

III. TLMs for Test Infrastructure

Below we detail how test infrastructure building blocks can be modeled at transaction level.

A. Test Access Mechanism

The purpose of the TAM is the transfer of test data, i.e. input stimuli and output responses from a source to the core under test and from the core to a sink. The spectrum of different TAMs ranges from serial boundary scan chains to reuse of buses and NoCs. In a single SoC, multiple TAMs can be integrated, while each TAM may connect to multiple test wrappers.

At transaction level, the TAM is modeled as a communication channel. Functional aspects such as bandwidth, latency, addressing and arbitration schemes of the TAM need to be reflected in the model.

The TLM interface of a TAM is defined by the two methods *write* and *read* for data transfer. Moreover, since in some architectures a single access may combine a write and read operation, for example TAM slaves that contain scan chains where data is concurrently shifted in and out, an additional *write_read* method is included in the interface.

The class diagram in Figure 2 depicts the relation between the TAM interface (TAM_IF), derived from the generic SystemC interface, and the classes modeling test infrastructure. The TAM interface is implemented by the actual TAM channel as well as by the infrastructure blocks that are accessed via the TAM. The TAM is connected to these blocks using the SystemC bind mechanism.

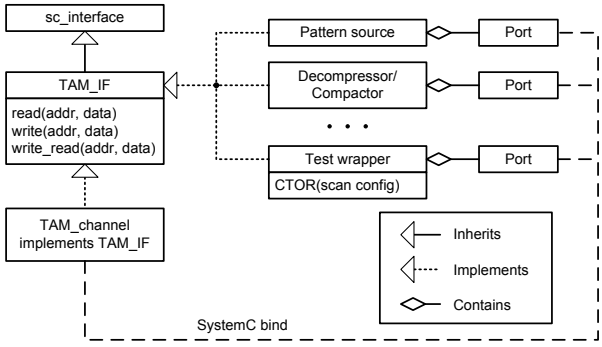

Fig. 2. UML class diagram of test infrastructure entities at transaction level

B. Test Wrapper

A test wrapper is a thin shell placed around a core that allows to access and operate the core either in functional mode or in a test mode. For interoperability and reuse, IEEE Std 1500 specifies the interface of a wrapper and several mandatory operation modes, e.g. modes for the test of internal logic or of external interconnects.

Figure 3 shows the TLM architecture of a typical test wrapper. The wrapper contains a wrapper instruction register (WIR) which can be written using a dedicated configuration scan bus. According to the configured mode in the WIR, transactions are directly forwarded to the core when in functional/bypass mode or interpreted as test data when in one of the test modes.

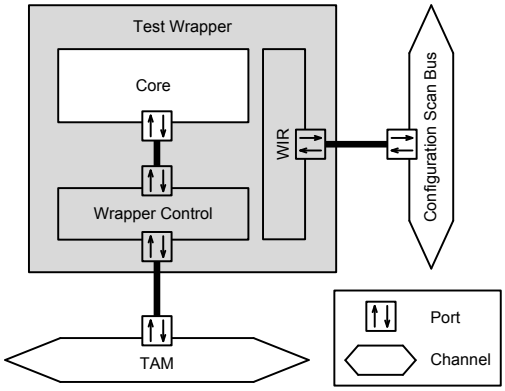

Fig. 3. Transaction level model of a test wrapper

The model of the core to be wrapped can be either a merely functional TLM, a refined approximately timed model, a model at register transfer level (RTL) or even at gate level. Given the Core Test Language (CTL, IEEE Std 1450.6) description of the interface of the core, comprised of functional, system and test in- and outputs, a test wrapper TLM can be generated automatically.

C. Pattern Source

A pattern source TLM supplies test data to a sink via the TAM. For Logic BIST, pseudo-random patterns are generated by a linear-feedback shift register. In a deterministic BIST scheme, highly compressed pre-computed patterns may be supplied. For external test the pattern source is an *external bus interface* (EBI) that translates the ATE protocol to the TAM protocol. Hence, the EBI is implemented as an interface adaptor in the TLM.

D. Decompressor/Compactor

Sophisticated test data compression schemes can compress test stimuli by up to 1000x and compaction may reduce the test responses down to a signature word. The decompressor TLM translates transactions from the TAM into transactions for a core wrapper, thereby expanding compressed test data. Correspondingly, a compactor TLM translates a number of transactions from a core wrapper into transactions for the TAM compacting test responses. Like the EBI, the decompressor/-compactor TLM is an interface adaptor, which allows for plug & play deployment of decompressors and compactors. The decompressor/compactor TLM is configurable via a configuration scan bus and supports a bypass mode similar to the test wrapper. It can model static as well as variable compression ratios.

E. Test Controller / Automated Test Equipment

The on-chip test controller and the ATE implement the aspects of the high level test protocol, flow and strategy. The test controller implements the BIST control functions required for the test. The ATE configures the test infrastructure, initiates individual tests, supplies test stimuli, evaluates test responses and executes repair actions if necessary. The ATE communicates directly with the test controller in the SoC model and interfaces to the TAM via the EBI.

During design exploration, the ATE and the executed test program are modeled by their functional behavior, i.e. by their interaction with the test controller and EBI. But for verification purposes, Virtual ATE software can be interfaced to the test controller and EBI to simulate the actual test program instructions [10].

IV. Evaluation

The proposed modeling of test infrastructure and its application in test exploration is demonstrated on an approximately timed TLM of a bus-based JPEG encoder SoC. The SoC contains an embedded processor core, a memory core and two dedicated cores for color conversion and the DCT operation.

Figure 4 shows the TLM of the SoC including the test infrastructure (in gray), i.e. the test wrappers around the functional cores, the test controller, the decompressors/compactors etc. The model is implemented using SystemC 2.2.

In this example, the system bus is reused as TAM. The simulation environment also includes an ATE model which initiates and controls the overall test flow. The test controller uses the configuration scan ring to configure the EBI, the test wrappers and the decompressors.

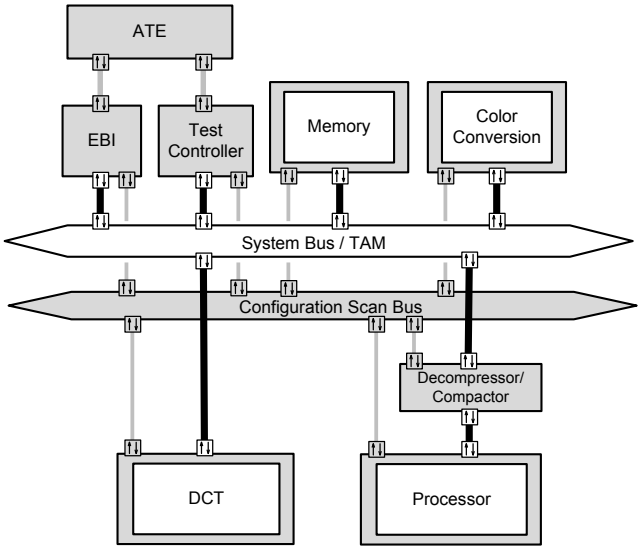

Fig. 4. Transaction level model of the JPEG encoder SoC including test infrastructure (in gray)

The test of the SoC is implemented by several of the following test sequences:

1) BIST of the full-scan processor core with 32 scan chains using 100,000 pseudo-random patterns.

2) Deterministic logic test of the processor core using 20,000 patterns stored in the ATE.
3) Deterministic logic test of the processor core using compressed test data with a compression ratio of 50X.
4) BIST of the color conversion core using 10,000 pseudo-random patterns.
5) Deterministic logic test of the full-scan DCT core with 8 scan chains using 10,000 patterns stored in the ATE.
6) Test controller driven Array BIST of the embedded memory core (1 MByte) using a MATS+ march and pattern tests.
7) The processor drives the same array tests of the embedded memory core as in test 6 using a program stored in L1 cache.

The test of the system bus will not be considered in this evaluation. It can be implemented as a functional test and modeled at transaction level as proposed in [11].

Given the seven test sequences, we target four different test schedules to investigate the test time and TAM utilization:

1) Sequential execution of the core tests 1, 2, 4, 5 and 7.
2) Sequential execution of the core tests 1, 3, 4, 5 and 6.
3) Concurrent execution of core tests 1 and 5, followed by concurrent execution of tests 2, 4 and finally, execution of memory test 7.
4) Concurrent execution of core tests 1 and 5, followed by concurrent execution of tests 3, 4 and 6.

The simulations have been performed on a 2.4 GHz workstation. Table I gives the peak and average TAM utilization, test length and required CPU time in seconds for the simulated test scenarios.

Test scenario	Peak TAM utilization	Avg. TAM utilization	Test length (10^6 cycles)	CPU runtime (sec)
1	67%	45%	281	418
2	67%	58%	184	271
3	80%	47%	263	390
4	100%	64%	167	261

TABLE I
SIMULATION RESULTS FOR DIFFERENT TEST SCENARIOS

The results of the sequential scenarios show an increased test time for schedule 1 due to limiting throughput of the ATE channel when using uncompressed patterns and the slower memory test driven by the processor. Scenario 3 and 4 exploit concurrent test execution to reduce test time. This leads to higher peak and average TAM utilization, even reaching 100% at some points.

The complete scenarios require simulation of up to about 300 million clock cycles, which is not feasible at gate or RT level. For comparison, the simulation of 300 million cycles of the RTL model of the processor core alone already exceeds two days of CPU time. Simulation at gate level increases simulation time by another order of magnitude. The simulation at transaction level requires less than seven minutes of CPU time for simulation of the whole system including the test infrastructure and allows to quickly explore the test design space by evaluation of different test strategies, test infrastructure blocks and test schedules.

V. CONCLUSIONS

The proposed modeling of test infrastructure at transaction level allows fast exploration of the test design space and validation of test strategies.

For the relevant test infrastructures such as TAMs, test wrappers, pattern generators or decompressors, models at transaction level have been described. Based on the TLM of an exemplary SoC, the simulation of test infrastructure, different test strategies as well as test schedules has been demonstrated. The simulation is multiple orders of magnitude faster than RTL or gate level simulation and thus allows evaluation of complete test schedules.

ACKNOWLEDGMENT

This work has been supported by the German Research Foundation (DFG) under grants Wu245/3-3 and Wu245/5-1 and by a Vigoni grant of the German Academic Exchange Service (DAAD).

REFERENCES

[1] J. A. Carballo and S. R. Nassif, "Impact of design-manufacturing interface on SoC design methodologies," *IEEE Design & Test of Computers*, vol. 21, no. 3, pp. 183–191, 2004.
[2] Y. Zorian, "What is infrastructure IP?" *Design & Test of Comp., IEEE*, vol. 19, no. 3, pp. 3–5, 2002.
[3] É. F. Cota, L. Carro *et al.*, "Test planning and design space exploration in a core-based environment," in *Proc. Design, Automation and Test in Europe (DATE)*, 2002, pp. 478–485.
[4] F. Ghenassia, Ed., *Transaction-Level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems*. Springer, 2005.
[5] Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate retargetable performance estimation at the transaction level," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 3–8.
[6] M. Cheema and O. Hammami, "Introducing Energy and Area Estimation in HW/SW Design Flow Based on Transaction Level Modeling," in *Proc. International Conference on Microelectronics (ICM)*, 2006, pp. 182–185.
[7] N. A. Touba, "Survey of test vector compression techniques," *IEEE Design & Test of Comp.*, vol. 23, no. 4, pp. 294–303, 2006.
[8] A. Larsson, E. Larsson *et al.*, "Optimization of a bus-based test data transportation mechanism in system-on-chip," in *Proc. 8th Euromicro Symposium on Digital Systems Design (DSD)*, 2005, pp. 403–411.
[9] S. Samii, M. Selkala *et al.*, "Cycle-accurate test power modeling and its application to SoC test architecture design and scheduling," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 973–977, May 2008.
[10] G. Krampl, M. Rona, and H. Tauber, "Test setup simulation - a high-performance VHDL-based virtual test solution meeting industrial requirements," in *Proc. IEEE International Test Conference (ITC)*, 2002, pp. 870–878.
[11] H. Alemzadeh, S. D. Carlo *et al.*, "Plug & Test at system level via testable TLM primitives," in *Proc. IEEE International Test Conference (ITC)*, 2008.
[12] K. Goossens, B. Vermeulen *et al.*, "Transaction-based communication-centric debug," in *Proc. Int. Symp. on Network-on-Chips (NOCS)*, 2007, pp. 95–106.
[13] M. Radetzki, "Object-oriented transaction level modelling," in *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. Springer, 2007.
[14] A. Donlin, "Transaction level modeling: flows and use models," in *Proc. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2004, pp. 75–80.
[15] L. Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures: Nanometer Design for Testability*. Morgan Kaufmann, 2007.