Modellierung der Testinfrastruktur auf der Transaktionsebene

Michael A. Kochte, Christian G. Zoellin,
Michael E. Imhof, Rauf Salimi Khaligh,
Martin Radetzki, Hans-Joachim Wunderlich
Universität Stuttgart
Institut für Technische Informatik
Pfaffenwaldring 47
D-70569 Stuttgart

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

***Zusammenfassung*—Dieser Artikel stellt eine Methode vor, den Entwurfsraum beim prüfgerechten Entwurf (engl. Design-for-Test, DfT) zu untersuchen und Teststrategien und Testschedules zu validieren. Alle Teile der Testinfrastruktur, wie etwa die Testeranbindung (Test Access Mechanisms), die Testwrapper, die Testdatenkompression sowie die entsprechenden Steuerwerke werden auf Transaktionsebenenmodelle (TLMs) abgebildet. Die kommunikationsbezogene Sicht der TLMs eignet sich besonders, da viele Aspekte des Tests die Übertragung großer Mengen an Teststimuli und -antworten erfordern. An einer Fallstudie wird der Einsatz von TLMs in frühen Entwurfsphasen erläutert. Der vorgestellte Ansatz hat wesentlich höhere Simulationseffizienz als Ansätze auf Register-Transfer- und Gatterebene.**

I. Einleitung

Aktuelle Systems-on-Chip (SoC) bestehen aus einer Vielzahl unterschiedlicher IP-Module [1]. Infrastruktur, die nicht direkt mit der Systemfunktionalität zusammenhängt, ist von zunehmender Bedeutung für den Test und die robuste Funktion [2, 3]. Ähnlich wie der Entwurf der Systemfunktionalität ist der Entwurf der Test-, Diagnose- und Reparaturstrategien sehr komplex. Neue Teststrategien erfordern zudem weitergehende Interaktion zwischen Tester und zu testender Schaltung [4]. Die Untersuchung des Entwurfsraums (engl. Design Space Exploration) sowie die Verifikation sind damit auch hier ein wichtiger Bestandteil des Entwurfsablaufs [5, 6].

Um den Test von IP Cores aus einer Vielzahl von Quellen und Zulieferern zu vereinfachen, existieren Standards wie z.B. IEEE Std 1500, die die Beschreibung von Testinfrastruktur und -daten vereinheitlichen. Aber selbst wenn diese Informationen vom IP-Entwickler zur Verfügung gestellt werden, so bleiben zahlreiche Entscheidungen, die der Testingenieur treffen muss. Dazu zählen die Auswahl der Verfahren zur Testdatenkompression [7] und zur Testeranbindung (engl. Test Access Mechanism, TAM) [8–11]. Die systematische Exploration des entsprechenden Entwurfsraums bedarf einer Vielzahl von Modellen und Simulationen.

Nachdem die Testarchitektur festgelegt wurde, wird durch Scheduling der Test innerhalb der Grenzen von Verlustleistung und Testerbandbreite parallelisiert. Hierzu wurden zahlreiche Verfahren vorgestellt (z.B. [12, 13]). Wegen der Komplexität des Optimierungsproblems wird in der Regel nur ein Teil der zur Verfügung stehenden Information berücksichtigt. Um genauere Ergebnisse bezüglich Verlustleistung und TAM-Auslastung zu erhalten, sollte der Ablaufplan entsprechend durch Simulation evaluiert werden [14].

Trotz der erwähnten Automatisierung und Standardisierung ist das Testprogramm, das auf dem Automated Test Equipment (ATE) ausgeführt wird, ein komplexes Softwareprodukt aus vielen Einzelbestandteilen. Die Validierung des Testprogramms wird durch virtuellen Test und ähnliche Ansätze unterstützt [15–18]. Für große Schaltungen erreichen Simulationen auf RTL- oder Gatterebene typischerweise einige hundert Taktzyklen pro Sekunde mit Softwaresimulatoren und einige Tausend mit speziellen Beschleunigern [19]. Ein komplettes Testprogramm dauert jedoch üblicherweise mehrere hundert Millionen Taktzyklen.

Exploration und Validierung werden also durch lange Simulationszeiten behindert. Im funktionalen Entwurf wird dieses Problem mit Modellen auf hohen Abstraktionsebenen (z.B. der Transaktionsebene) gelöst [20–22]. Transaktionsebenenmodelle (Transaction Level Models, TLMs) beschleunigen die Simulation um mehrere Größenordnungen. Sie enthalten aber ausreichend Detailinformation, um wichtige Entwurfsentscheidungen bezüglich Performanz, Chipfläche und Verlustleistung zu treffen [23, 24].

Die kommunikationsbezogene Sichtweise von TLMs ist zur Modellierung des Tests hervorragend geeignet, da beim Test große Datenmengen ausgetauscht werden müssen. Des Weiteren ist die genaue Modellierung der Nebenläufigkeit ein wichtiger Aspekt von TLMs. Dies lässt sich entsprechend zur Modellierung der Parallelität des Testablaufs nutzen. Erste Arbeiten in diese Richtung zeigen das Verhältnis von Entwurfsverifikation, Entwurfsvalidierung und Test bei TLMs [25]. Bei der Kommunikation zwischen ATE und zu testender Schaltung wurde von [4] die transaktionsbasierte Kommunikation angeregt. Für Debug stellen die Autoren von [26] die Modellierung von Debugtransaktionen vor, um damit die Nebenläufigkeit während des funktionalen Debuggings bzw. Tracings zu untersuchen.

Die folgenden Abschnitte zeigen, wie der Test auf der Transaktionsebene modelliert werden kann, um damit die

Exploration des Testentwurfsraums sowie die Validierung von Teststrategien und -schedules zu unterstützen.

Eine Studie demonstriert die vorgestellte Methode am Beispiel eines JPEG Encoder SoCs. Dieses SoC wird um alle Komponenten erweitert, die benötigt werden, um die erwähnten Aspekte des Tests auf der Transaktionsebene zu modellieren. Die vorgestellte Modellierung kann für jede Art von TAM-Architektur verwendet werden. Die Fallstudie konzentriert sich jedoch auf den aktuellen Trend zur Weiterverwendung funktionaler Busstrukturen für den TAM [9–11] sowie die zunehmende Interaktion zwischen Tester und Schaltung [4].

Der Rest des Artikels gliedert sich folgendermaßen: Abschnitt 2 zeigt, wie Eigenschaften des Design-for-Test auf hohen Abstraktionsebenen modelliert werden können. Abschnitt 3 gibt einige Beispiele, wie die TLMs für die wichtigsten Bausteine wie Testwrapper, TAMs, Testmustergeneratoren und Testcontroller implementiert werden. In Abschnitt 4 werden die vorgeschlagene Methode evaluiert und die Vorteile der Testinfrastruktur-TLMs bewertet.

II. Modellierung des Tests auf der Transaktionsebene

Transaktionsebenenmodellierung [21] ermöglicht den Entwurf auf Systemebene und die Simulation großer Hardware/Software-Systeme, bei denen RTL-Simulationen inakzeptable Laufzeiten zeigen. Dies wird erreicht, indem von der Kommunikation auf Signalebene abstrahiert wird und komplexe Kommunikationsoperationen zu atomaren Transaktionen zusammengefasst werden. Dadurch reduziert sich die Zahl der Ereignisse, die von ereignisgesteuerten Simulatoren verarbeitet werden müssen, sowie die Zahl der Kontextwechsel zwischen Simulationsprozessen. Transaktionen werden als objektorientierte Methoden (Unterprogramme) implementiert, die den Zustand von Kanälen beeinflussen, welche wiederum die Kommunikationsinfrastruktur des Systems darstellen [27].

Bei TLMs wird die Funktionalität der Systemkomponenten getrennt von der kommunikationsbezogenen Sichtweise des Systems beschrieben. Dies erlaubt es dem Entwickler, mühelos Systemkomponenten und Kommunikationsmodelle auszutauschen, deren Kopplung zu verändern, sowie verschiedene Abbildungen der Anwendungsfunktionalität auf Systemkomponenten zu untersuchen. Dadurch wird eine rasche Exploration von Systemalternativen ermöglicht. Nachdem eine geeignete (optimierte) Systemarchitektur gefunden wurde, kann das ursprüngliche TLM verfeinert werden, indem Details über Funktion und Kommunikation hinzugefügt werden. Dies führt zu Modellen, die bezüglich Verhalten und Timing genauer sind und schließlich zu einem taktgenauen (engl. cycle-accurate) Modell, das einer Implementierung bereits ähnlich ist [22].

Systembeschreibungssprachen wie SystemC (IEEE Std 1666) oder SystemVerilog (IEEE Std 1800) stellen die für TLM benötigten Primitive zur Verfügung. SystemC unterstützt den Entwickler zusätzlich mit einer Standardbibliothek (TLM2.0) von Schnittstellen und Bausteinen für busbasierte SoCs [28]. In der vorliegenden Arbeit werden die TLM-Prinzipien auf den Bereich der Test-Infrastruktur und -Architektur zu übertragen. Die entsprechenden Modelle wurden basierend auf den SystemC TLM-Primitiven implementiert. Es wurde jedoch auf die TLM2.0 Bibliothek verzichtet, da TAMs Modellierungsaspekte erfordern, die über normale SoC Busse hinausgehen.

Aktuelle Schaltungsentwürfe haben zahlreiche nichtfunktionale Anforderungen (Test, Debug, etc.) (Abb. 1). Diese Eigenschaften erfordern einen erheblichen Umfang an Infrastruktur, der zusammen mit der eigentlichen Funktion integriert werden muss. Der vorliegende Artikel konzentriert sich hier in erster Linie auf die Transaktionsmodellierung der Infrastruktur und der Teststrategie für den Fertigungstest.

Aus Sichtweise der Funktion ist der Fertigungstest üblicherweise eine Kombination verschiedener Einzeltests, die je nach Art der verwendeten Cores und Randbedingungen wie Testzeit und Verlustleistung, ausgewählt werden. Jedes Modul eines Entwurfs stellt spezifische Anforderungen an dessen Test [29, 30]. Module mit freier Logik können über Prüfpfade mit deterministischen oder Zufalls-Mustern getestet werden. Dies wird häufig noch durch funktionale und In-the-loop Tests (z.B. in Mixed-Signal Schaltungen) ergänzt. Speicherfelder

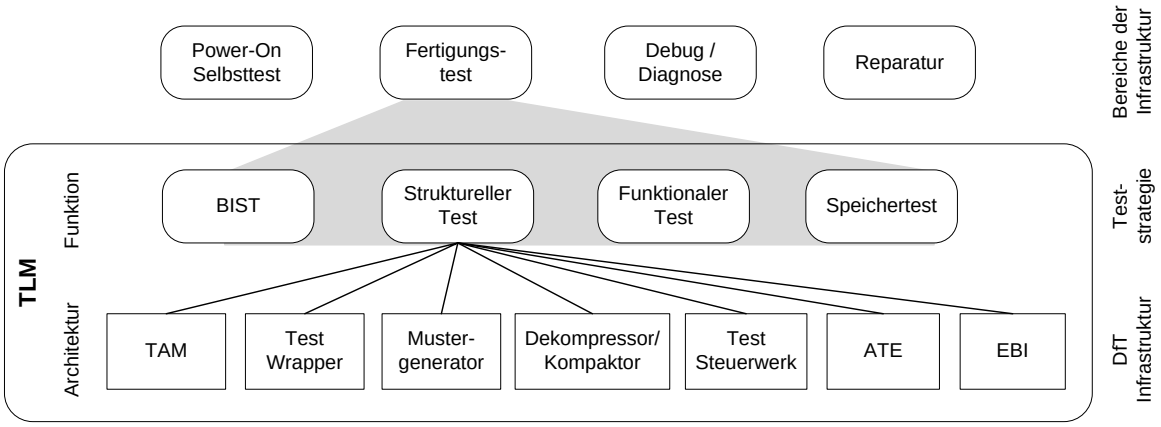

Abbildung 1. Design-for-Test Modellierung

werden für gewöhnlich durch spezielle March-Tests getestet, die entweder mit einem Selbsttest-Steuerwerk oder einem Prozessor durchgeführt werden.

Diese Testfunktionalität wird dann auf eine Auswahl von Strukturblöcken wie z.B. TAMs, Testwrapper, Testmustergeneratoren, Interfaces zum Tester (External Bus Interface, EBI), Dekompressoren, Kompaktoren, Steuerwerke und den ATE selbst abgebildet. Diese Ebene des TLM entspricht der Architektur eines Systems zusammen mit der Hardware/Software-Partitionierung der Funktion. Im Kontext des Tests umfasst der Softwareteil das Testprogramm auf dem ATE, die Programme für einen funktionalen Test, z.B. bei Prozessorkernen, und den Mikrocode zur Programmierung des eingebauten Teststeuerwerks.

III. TLMs für Testinfrastruktur

Im Folgenden wird beschrieben, wie die wichtigsten Bausteine der Testinfrastruktur auf der Transaktionsebene modelliert werden.

A. Test Access Mechanism

Die Aufgabe des TAM ist die Übertragung von Testdaten (d.h. Testeingaben und Testantworten) von einer Quelle zu einem zu testenden Modul und von diesem Modul zu einer Senke. In einem einzelnen SoC können mehrere, verschiedene TAMs integriert sein, und jeder TAM kann wiederum mehrere Knoten verbinden. TAMs können entweder ausschließlich für den Test vorgesehen werden (dedizierte TAMs) oder Strukturen für den funktionalen Betrieb (z.B. Busse oder NoCs) wiederverwenden.

TAMs besitzen verschiedene funktionale und nichtfunktionale Charakteristika. Auf der Transaktionsebene wird der TAM als Kommunikationskanal modelliert. Funktionale Aspekte des TAMs wie Bandbreite, Latenz, Adressierung und Arbitrierung müssen in diesem Modell berücksichtigt werden.

Ein dedizierter TAM, bei dem einzelne Leitungen zu unterschiedlichen Modulen individuell gemultiplext werden können, wird dadurch modelliert, dass seine Gesamtbandbreite auf die verschiedenen Kommunikationspartner verteilt wird. Bei einem NoC oder Systembus, der während des Tests wiederverwendet wird, wird das bestehende TLM um das TAM Interface erweitert, indem die TAM Methoden auf die funktionalen Kommunikationsprimitive des zugrundeliegenden TLM abgebildet werden. Bei einem seriellen TAM, wie z.B. einer Boundary-Scan-Kette, ist die Zeitdauer oder Latenz einer Operation prinzipbedingt konstant, unabhängig von der Zahl der Register, auf die zugegriffen wird. Die Dauer hängt hier von der Gesamtzahl der Elemente im Prüfpfad ab, da stets die ganze Kette geschoben werden muss.

Die TLM Schnittstelle eines TAMs wird durch zwei Methoden *write* und *read* für den Datentransfer dargestellt. Einige TAM Architekturen erlauben es, in einem einzigen Transfer eine Schreib- und Leseoperation zu kombinieren. Beispiele sind TAM Slaves, die Prüfpfade enthalten, so dass Daten gleichzeitig hinein- und hinausgeschoben werden können. Dazu enthält die Schnittstelle eine weitere Methode *write_read*.

Das Klassendiagramm in Abbildung 2 zeigt das Zusammenwirken des TAM Interface (TAM_IF), das von einem generischen SystemC Interface und den Klassen, die die Testinfrastruktur modellieren, abgeleitet ist. Das TAM Interface wird sowohl vom eigentlichen TAM Kanal, als auch von allen Infrastrukturblöcken, die mit dem TAM verbunden sind, implementiert. Diese Blöcke, wie z.B. Testwrapper, Testmustergeneratoren oder Dekompressoren, besitzen einen Port, der es erlaubt, sie mit dem TAM zu verbinden. Hierfür wird der SystemC *bind* Mechanismus verwendet.

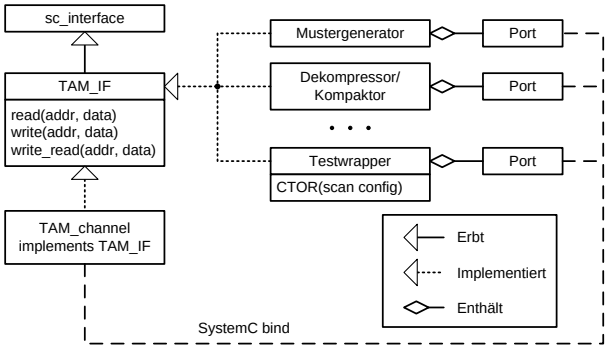

Abbildung 2. UML Klassendiagramm der Testinfrastruktur auf Transaktionsebene

B. Testwrapper

Der Testwrapper umschließt einen Core und ermöglicht es, den Core funktional oder in einem Testmodus zu betreiben. Zur Steigerung der Interoperabilität und der Wiederverwendung von Cores wurde der IEEE 1500 Standard definiert. Der Standard spezifiziert die Schnittstelle des Wrappers und mehrere Betriebsmodi, z.B. zum Test core-interner Logik oder von core-externen Verbindungen.

Abb. 3 zeigt den Aufbau des TLM eines typischen Testwrappers. Der Wrapper besteht aus einem Befehlsregister (WIR, Wrapper Instruction Register), das über einen speziellen Konfigurationsbus beschrieben werden kann. Weiterhin ist der Wrapper an den Systembus angeschlossen. Transaktionen, die den eingebetteten Core adressieren, werden vom Wrapper empfangen. Je nach konfiguriertem Betriebsmodus im WIR werden diese Transaktionen direkt an den Core weitergeleitet (Funktionaler/Systemmodus) oder als Testdaten interpretiert (Testmodus).

Das Modell des eingebetteten Cores kann entweder ein rein funktionales Modell auf Transaktionsebene sein oder ein verfeinertes Modell, z.B. auf Register-Transfer-Ebene (RT) oder sogar auf Gatterebene. Liegt eine Beschreibung der Schnittstelle des Cores vor, kann das Modell des Wrappers automatisch erzeugt werden. Eine solche Beschreibung kann z.B. in der Core-Test-Language (CTL, IEEE Std 1450.6) erfolgen und umfasst die Ein- und Ausgänge für den funktionalen und Testbetrieb.

Die Test-Features, die im Modell des Cores abgebildet sind, hängen vom Detailgrad und von der Art des Core-Modells ab. Im Gegensatz zu Modellen auf RT- oder Gatterebene enthält

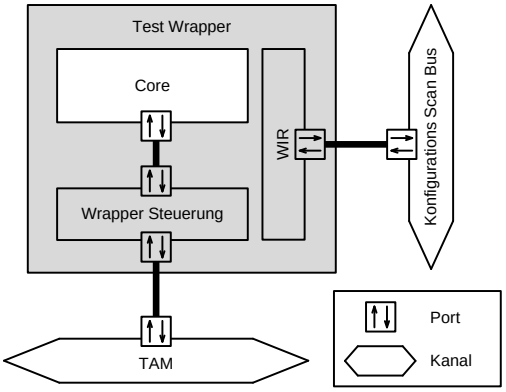

Abbildung 3. Modell eines Testwrappers auf Transaktionsebene

ein rein funktionales Modell z.B. keine Testpunkte. Für die Test-Partitionierung und Ablaufplanung wird dennoch eine näherungsweise Modellierung einiger Eigenschaften benötigt [31]. Zur Simulation und Validierung der Teststrategie müssen zumindest die Zahl der Scan-Ein- und Ausgänge, die Länge der Scanketten des Cores und die Zahl der Patterns pro Test vorliegen. Diese Information kann ebenso aus der CTL-Beschreibung des Cores extrahiert werden. Liegt keine CTL-Beschreibung eines Cores vor, kann das Modell des Wrappers so konfiguriert werden, dass die Dauer eines Tests durch Emulation von Scan-In und Scan-Out Operationen abgeschätzt wird.

C. Testmuster-Quelle

Das TLM einer Musterquelle liefert Testdaten über einen TAM an eine Mustersenke. Beim Logic-BIST werden pseudozufällige Muster dabei durch ein rückgekoppeltes Schieberegister (LFSR) erzeugt. Bei Verwendung von deterministischen BIST-Verfahren werden vorberechnete, hochkomprimierte Muster ausgegeben. Wird ein Core durch externes Testequipment getestet, wird die Musterquelle als Schnittstelle zum ATE modelliert (*External Bus Interface*, EBI) und übersetzt Transaktionen des ATE auf TAM-Transaktionen. Entsprechend kann dann das EBI als Schnittstellen-Adapter im TLM aufgefasst werden.

D. Dekompressor/Kompaktor

Aktuelle Algorithmen zur Testdatenkompression erzielen Kompressionsraten von bis zu 1000x und durch Kompaktierung der Testantworten kann die Signatur auf ein einzelnes Wort reduziert werden. Das TLM des Dekompressors übersetzt Transaktionen vom TAM in Transaktionen für den Testwrapper. Entsprechend bildet das TLM des Kompaktors Transaktionen des Testwrappers auf Transaktionen für den TAM ab. Wie das EBI sind die Modelle von Dekompressoren und Kompaktoren Schnittstellen-Adapter. Ähnlich zum Testwrapper sind diese Modelle über einen speziellen Konfigurationsbus parametrisierbar, damit zur Simulationszeit zwischen Kompression und funktionalem Betriebsmodus bzw. transparentem Bypassmodus umgeschaltet werden kann.

Einfache Kompressionsmethoden verwenden eine konstante Kompressionsrate, während fortgeschrittene Verfahren variable Raten in Abhängigkeit von der Entropie der Daten aufweisen [32]. Das genaue Verhalten hängt von den Testdaten ab, für die Exploration verschiedener Teststrategien ist aber ein näherungsweises funktionales Modell ausreichend. Zur Validierung von Testprogrammen werden tatsächliche Testdaten zwischen Dekompressor und Kompaktor ausgetauscht, um genauere Simulationsergebnisse zu erzielen.

E. Test-Steuerwerk / Externes Testequipment (ATE)

Zusammen mit dem ATE implementiert das Test-Steuerwerk auf dem Chip das Testprotokoll und den Testablauf. Dieses Steuerwerk implementiert dabei die Funktionen zur Steuerung des BIST. Der ATE konfiguriert die Testinfrastruktur, startet einzelne Tests, liefert Teststimuli an den EBI und wertet Testantworten aus. Falls erforderlich kann der ATE auch Reparaturmaßnahmen durchführen. Der ATE kommuniziert direkt mit dem Test-Steuerwerk im TLM des SoC und liefert über das EBI Testdaten an den TAM.

In der Entwurfsraumexploration werden sowohl ATE als auch das ausgeführte Testprogramm durch ihr funktionales Verhalten, d.h. die Interaktion mit dem Testkontroller und EBI, modelliert. Zur Validierung von Testprogrammen können *Virtual ATE* Modelle mit dem Testkontroller und EBI verbunden werden, um die Befehle im Testprogramm zu simulieren [17, 18]. Hierbei kann ein virtuelles ATE Modell ähnlich wie Legacy-Instruction-Set-Simulatoren eingebunden werden [33].

IV. Evaluierung der Modellierungsmethodik

Im Folgenden wird die vorgeschlagene Modellierung der Testinfrastruktur und die Verwendung zur Testexploration an einem approximately-timed TLM eines JPEG Encoder SoCs demonstriert. Dieses Beispielsystem besteht aus häufig verwendeten Core-Typen, die auch in komplexeren Systemen zu finden sind. Die Benchmark-Schaltungen für den modularen Test von SoCs aus [31] beschreiben die Anzahl und die Schnittstellen der Cores der SoCs und weitere Informationen, die zur Evaluierung von Algorithmen zur Testplanung benötigt werden. Die Benchmarks beinhalten jedoch keine Informationen über die eigentliche Funktion der Cores oder der jeweiligen Tests (z.B. Arraytest für Speicher). Das hier gewählte System besteht aus Komponenten, wie Systembus und Prozessor, die im Test wiederverwendet werden können.

Die JPEG Enkodierung besteht aus 5 verschiedenen Operationen auf den Bild-Daten: *(i)* Farbraumkonvertierung; *(ii)* zweidimensionale, diskrete Kosinus-Transformation (DCT); *(iii)* Quantisierung der Kosinus-Koeffizienten; *(iv)* Lauflängenkodierung (RLE) der quantifizierten Koeffizienten und *(v)* Huffman-Kodierung der RLE-Daten. Bei der Entwurfsexploration wurde dieser Algorithmus partitioniert und auf eine Systemarchitektur abgebildet. Das Beispiel basiert auf einer Busarchitektur mit eingebettetem Prozessorkern, eingebettetem Speicher und zwei zusätzlichen Kernen, die DCT und Farbraumkonvertierung durchführen. Auf dem Prozessor ausgeführte Software implementiert

die Quantisierung, Lauflängenkodierung und Huffman-Kompression.

Abbildung 4 zeigt das TLM des SoC zusammen mit der Test-Infrastruktur (in grau, bestehend aus Wrappern, Test-Steuerwerk, etc.). Das Modell wurde mit SystemC 2.2 implementiert.

In diesem Beispiel wird der System-Bus als TAM wiederverwendet. Die Simulationsumgebung enthält außerdem ein ATE-Modell, das den gesamten Ablauf initiiert und kontrolliert. Das Test-Steuerwerk verwendet den Konfigurations-Scanring, um EBI, Test Wrapper und Dekompressor zu konfigurieren.

Das SoC wird durch eine Abfolge der folgenden Testsequenzen getestet:

1) BIST des Prozessorkerns mit 100.000 Pseudozufallsmustern. Der Prozessorkern ist ein Fullscan-Entwurf mit 32 Prüfpfaden und einer maximalen Kettenlänge von 512 Elementen. Die Zufallsmuster werden direkt vom Steuerwerk generiert.
2) Deterministischer Logiktest des Prozessorkerns mit 20.000 Mustern, die im ATE gespeichert sind.
3) Deterministischer Logiktest des Prozessorkerns mit kompaktierten Testmustern und Dekompression durch kontinuierliches Reseeding [34]. Der Kompressionsfaktor der Testdaten ist 50X. Die Testdaten werden zum Dekompressor geleitet, dort expandiert und an den entsprechenden Zielwrapper weitergeleitet. Die Testantworten werden in einem Signaturregister (Multiple-Input Signature Register, MISR) kompaktiert. Der ATE liest die resultierende Signatur über den TAM.
4) BIST des Moduls zur Farbraumkonvertierung mit 10.000 Zufallsmustern.
5) Deterministischer Logiktest des DCT-Moduls mit 10.000 Mustern aus dem ATE. Der DCT-Kern ist ein Fullscan-Entwurf mit 8 Prüfpfaden und einer maximalen Kettenlänge von 256 Elementen.

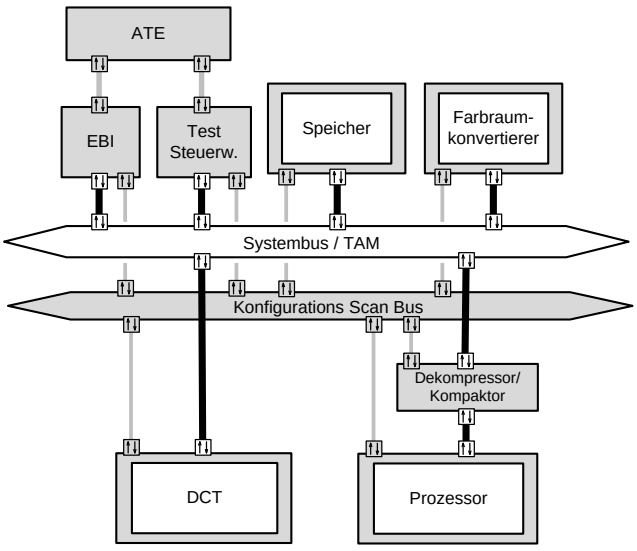

Abbildung 4. TLM des JPEG Encoder SoC mit Test-Infrastruktur (in grau)

6) Array-BIST des eingebetteten Speichers (1MB) durch einen vom Teststeuerwerk getriebenen MATS+ March Test und zwei Schachbrettmuster-Tests.
7) Der Prozessor führt den erwähnten Speichertest aus Test 6 durch, indem ein entsprechendes Programm im L1-Cache abgelegt wird.

Der Test des Systembusses wird in dieser Studie nicht betrachtet. Er kann wie in [25] vorgeschlagen als funktionaler Test implementiert und auf der Transaktionsebene modelliert werden.

Mit den sieben Test-Sequenzen werden nun vier verschiede Schedules implementiert, um deren Testzeit und TAM-Auslastung zu untersuchen:

1) Sequentielle Durchführung der Tests 1, 2, 4, 5 und 7.
2) Sequentielle Durchführung der Tests 1, 3, 4, 5 und 6.
3) Parallele Durchführung der Tests 1 und 5, gefolgt von der parallelen Durchführung der Test 2, 4 und schließlich Durchführung des Speichertests 7.
4) Parallele Durchführung der Tests 1 und 5, gefolgt von der parallelen Durchführung der Test 3, 4 und 6.

Die Szenarios 1 und 2 sind die sequentielle Aneinanderreihung der Einzeltests, während 3 und 4 entsprechende parallele Schedules sind.

Alle Simulationen wurden auf einer Workstation mit 2.4GHz durchgeführt. Tabelle I zeigt die maximale sowie durchschnittliche Auslastung des TAMs, die Testlänge und die benötigte CPU-Zeit in Sekunden für die simulierten Testszenarien.

Test Szenario	Max. TAM Auslastung	Durchs. TAM Auslastung	Testlänge (10^6 Takte)	CPU Laufzeit (s)
1	67%	45%	281	418
2	67%	58%	184	271
3	80%	47%	263	390
4	100%	64%	167	261

Tabelle I
SIMULATIONSERGEBNISSE FÜR VERSCHIEDENE TESTSZENARIEN

Die Ergebnisse der sequentiellen Szenarien weisen für Szenario 1 eine erhöhte Testlänge auf: Erstens wird der Test mit unkomprimierten, deterministischen Testmustern durch die ATE Bandbreite begrenzt und benötigt daher mehr Zeit als mit Kompression. Zweitens ist der vom Prozessor getriebe Test des eingebetteten Speichers langsamer als der Array-BIST im Steuerwerk.

Die Szenarien 3 und 4 führen die Tests parallel durch, um die nötige Testlänge zu reduzieren. Dies führt zu einer höherer maximalen und durchschnittlichen TAM- Auslastung, die an einigen Punkten sogar 100% erreicht. Abgesehen von der TAM- und ATE-Bandbreite kann die Testzeit auch vom zur Verfügung stehenden Verlustleistungsbudget abhängen. Die Untersuchung der Verlustleistung war nicht Teil dieser Arbeit, aber existierende Verlustleistungsmodelle für TLMs können angepasst werden, um die Verlustleistung während des Tests abzuschätzen.

Die vorgestellten Testszenarien erfordern die Simulation von bis zu 300 Millionen Taktzyklen. Derartige Simulationen auf Gatter- oder RT-Ebene sind nicht praktikabel. Zum Vergleich benötigt etwa die Simulation von 300 Millionen Takten des RTL-Modells des Prozessorkerns alleine bereits mehr als zwei Tage CPU-Zeit. Eine Simulation auf Gatterebene erhöht die Simulationszeiten um eine weitere Größenordnung. Die Simulation auf der Transaktionsebene benötigt hingegen weniger als 7 Minuten CPU-Zeit für die Simulation des gesamten Systems inklusive der Testinfrastruktur. Dies unterstützt die Exploration und Verifikation, da schnell verschiedene Teststrategien, Testarchitekturen und Testschedules untersucht werden können.

V. Schlussbemerkung

Die vorgeschlagene Modellierung der Testinfrastruktur auf der Transaktionsebene erlaubt die schnelle Exploration des Testentwurfsraums sowie die Validierung von Teststrategien.

Für die gebräuchliche Testinfrastruktur, wie z.B. TAMs, Testwrapper, Testmustergeneratoren und -dekompressoren, wurden entsprechende Modelle auf der Transaktionsebene vorgestellt. Basierend auf dem TLM eines Beispiel-SoC wurde die Simulation der Testinfrastruktur, verschiedener Teststrategien sowie unterschiedlicher Schedules gezeigt. Die Simulation ist mehrere Größenordnungen schneller als RTL- oder Gatter-Simulation und erlaubt somit die Evaluierung von vollständigen Testabläufen.

VI. Danksagung

Die vorliegende Arbeit wurde von der Deutschen Forschungsgesellschaft (DFG) Wu245/3-3 und Wu245/5-1, sowie vom Deutschen Akademischen Austausch Dienst (DAAD) durch das Vigoni Programm gefördert.

Literatur

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proc. ACM/IEEE Design Automation Conf. (DAC)*, 2004, pp. 681–685. [Online]

[2] Y. Zorian, "What is infrastructure IP?" *Design & Test of Comp., IEEE*, vol. 19, no. 3, pp. 3–5, 2002.

[3] S. K. Goel and E. J. Marinissen, "Effective and efficient test architecture design for SOCs," in *Proc. IEEE International Test Conference (ITC)*, 2002, pp. 529–538. [Online]

[4] J. Rivoir, "Protocol-aware ATE enables cooperative test between DUT and ATE for improved TTM and test quality," in *IEEE International Test Conference (ITC07)*, 2007, pp. 1–2.

[5] E. Larsson and Z. Peng, "An integrated framework for the design and optimization of SOC test solutions," *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 18, no. 4-5, pp. 385–400, 2002.

[6] É. F. Cota, L. Carro, M. Lubaszewski, and A. Orailoglu, "Test planning and design space exploration in a core-based environment," in *Proc. Design, Automation and Test in Europe (DATE)*, 2002, pp. 478–485. [Online]

[7] N. A. Touba, "Survey of test vector compression techniques," *IEEE Design & Test of Comp.*, vol. 23, no. 4, pp. 294–303, 2006. [Online]

[8] K. Chakrabarty, "Optimal test access architectures for system-on-a-chip," *ACM Trans. Design Autom. Electr. Syst.*, vol. 6, no. 1, pp. 26–49, 2001. [Online]

[9] A. Larsson, E. Larsson, P. Eles, and Z. Peng, "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. [Online]

[10] A. M. Amory, F. Ferlini, M. Lubaszewski, and F. Moraes, "DfT for the reuse of networks-on-chip as test access mechanism," in *Proc. IEEE VLSI Test Symposium (VTS)*, 2007, pp. 435–440. [Online]

[11] T. Yoneda and H. Fujiwara, "Wrapper and TAM co-optimization for reuse of SoC functional interconnects," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 1366–1369. [Online]

[12] V. Iyengar and K. Chakrabarty, "System-on-a-chip test scheduling with precedence relationships, preemption, and power constraints," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 21, no. 9, pp. 1088–1094, 2002. [Online]

[13] A. Larsson, E. Larsson, K. Chakrabarty *et al.*, "Test-architecture optimization and test scheduling for SOCs with core-level expansion of compressed test patterns," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 188–193. [Online]

[14] S. Samii, M. Selkala, E. Larsson *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. [Online]

[15] K. Helmreich, "How Virtual Test Becomes Reality," in *Proc. European Conference on Design and Test (ED&TC)*, 1996.

[16] K. Einwich, P. Schwarz, P. Trappe *et al.*, "Virtual Test of Complex Mixed-Signal Telecommunication Circuits reusing System Level Models," in *Proc. IEEE International Mixed-Signal Testing Workshop*, 1998, pp. 237–242.

[17] T. Hogan and D. Heffernan, "Virtual test reduces semiconductor product development time," *Electronics & Communication Engineering Journal*, vol. 13, no. 2, pp. 77–83, 2001.

[18] 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. [Online]

[19] R. Bell Jr, R. Bhatia, L. John *et al.*, "Automatic testcase synthesis and performance model validation for high performance PowerPC processors," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, 2006, pp. 154–165.

[20] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *Proc. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2003, pp. 19–24. [Online]

[21] F. Ghenassia, Ed., *Transaction-Level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems*. Springer, 2005.

[22] 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. [Online]

[23] 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. [Online]

[24] 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. [Online]

[25] H. Alemzadeh, S. D. Carlo, F. Refan *et al.*, "Plug & Test at system level via testable TLM primitives," in *Proc. IEEE International Test Conference (ITC)*, 2008.

[26] K. Goossens, B. Vermeulen, R. van Steeden, and M. T. Bennebroek, "Transaction-based communication-centric debug," in *Proc. Int. Symp. on Network-on-Chips (NOCS)*, 2007, pp. 95–106. [Online]

[27] M. Radetzki, "Object-oriented transaction level modelling," in *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. Springer, 2007.

[28] Open SystemC Initiative (OSCI) TLM Working Group, "Transaction level modeling standard 2 (OSCI TLM 2)," June 2008, www.systemc.org. [Online]

[29] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-signal VLSI Circuits*. Kluwer Academic, 2002.

[30] L. T. Wang, C. E. Stroud, and N. Touba, *System-on-Chip Test Architectures: Nanometer Design for Testability*. Morgan Kaufmann, 2007.

[31] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," in *Proc. IEEE International Test Conference (ITC)*, 2002, pp. 519–528. [Online]

[32] A.-W. Hakmi, H.-J. Wunderlich, C. G. Zoellin *et al.*, "Programmable deterministic built-in self-test," in *Proc. IEEE International Test Conference (ITC)*, 2007, pp. 1–9.

[33] L. Benini, D. Bertozzi, D. Bruni *et al.*, "Legacy SystemC co-simulation of multi-processor systems-on-chip," in *Proc. International Conference on Computer Design (ICCD)*, 2002, pp. 494–499. [Online]

[34] E. H. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *Proc. IEEE VLSI Test Symposium (VTS)*, 2003, pp. 232–240. [Online]