

Aktuelle Trends zur Standardisierung des Multi-Core-Debugging

Kai-Uwe Irrgang, Rainer G. Spallek
Kai-Uwe.Irrgang@mailbox.tu-dresden.de, rainer.spallek@tu-dresden.de
TU Dresden, Fakultät Informatik, Institut für Technische Informatik

Jens Braunes
Jens.Braunes@pls-mc.com
pls Programmierbare Logik & Systeme GmbH, Technologiepark, 02991 Lautz

Kurzfassung

Zur Entwurfsunterstützung von SoC auf der Basis von IP mehrerer Hersteller existieren Bestrebungen, alle IP mit einem einheitlichen Interface, wie beispielsweise dem Open Core Protokoll auszustatten, so dass die IP im Baukastenprinzip kombiniert werden können. Für Aspekte des Tests, des Debuggings und der Performance-Analyse der Systemsoftware muss auch die dafür dedizierte Debughardware eingebunden werden. Für diese Erweiterung wird der OCP Debug Socket vorgestellt. Die Komposition von SoC aus vordefinierten IP muss auch auf der Seite der Softwarewerkzeuge durch eine flexible Debugumgebung unterstützt werden. Wegen der Vielzahl möglicher Kombinationen von IP-Cores und damit SoC sollten die Debugwerkzeuge in Analogie zu den Hardwarestrukturen hierarchisch aufgebaut sein. Ein mögliches standardisiertes Interface zum Softwaredebugger ist die durch das SPRINT Projekt definierte Standard Debug API. Zur Architekturbeschreibung der Komponenten existiert das vom SPIRIT Consortium definierte IP-XACT.

1 Einleitung und Motivation

Wesentliche Kennzeichen der aktuellen Entwicklung von System-on-Chip (SoC) sind stetig wachsende Systemkomplexitäten bei gleichzeitig immer kürzer werdenden Entwicklungszeiten und Produktlebenszyklen, steigende Integrationsdichten und stark zunehmende Maskenkosten aufgrund immer kleinerer Strukturgrößen im Halbleiterfertigungsprozess. Hinzu kommen die stärkere Wiederverwendung von vorgefertigten und verifizierten Hardwarekomponenten (Intellectual Properties, IP) und der Trend, mehrere heterogene Prozessorkerne in einem SoC zu vereinigen.

Zur Entwurfsunterstützung von SoC auf der Basis von IP mehrerer Hersteller existieren Bestrebungen, alle IP mit einem einheitlich definierten Interface auszustatten und anzubieten. Ein Richtungweisendes Beispiel hierfür ist das Open Core Protokoll (OCP) [1], welches Signale und Sockets für die Verschaltung der IP auf dem SoC definiert. Der wesentliche Vorteil ist, dass die Komponenten im Baukastenprinzip zusammengesetzt werden können, ohne dass langwierige Anpassungsarbeiten erforderlich sind.

Neben diesen eher hardwarerelevanten Aspekten des SoC-Entwurfes muss auch der Tatsache Rechnung getragen werden, dass die Programmierbarkeit sowohl der einzelnen IP als auch des Gesamtsystems immer stärker zunimmt. Die unmittelbare Konsequenz daraus ist eine bezüglich Umfang und Komplexität stetig wachsende Systemsoftware. Folglich ist dieser Softwareaspekt an dem erwähnten Wachstum der Gesamtsystemkomplexitäten von SoC mindestens ebenso stark beteiligt.

Eine sehr große Herausforderung beim SoC-Entwurf ist die Sicherstellung der Fehlerfreiheit, sowohl bezüglich der Hardware als auch bezüglich der Software. Wegen der Forderung nach einem möglichst schnellen „Time-To-Market“, was gleichbedeutend mit einer möglichst kurzen Entwicklungszeit ist, werden Hardware und Software von Anfang an parallel entwickelt.

Dennoch gibt es im Projektfortschritt einen Zeitpunkt, ab dem die Hardware als fest stehend angesehen wird, während an der Software zumindest noch Optimierungen und Anpassungsarbeiten durchzuführen sind. Hardwarefehler sind zu diesem Zeitpunkt im Idealfall vollständig beseitigt, wenigstens aber diagnostiziert und dokumentiert, so dass in der Software sogenannte „Work-Arounds“ eingebaut werden können.

Neben diesem reinen Entwurfsaspekt bezüglich des Gesamtsystems inklusive der ersten Softwareversion ist es in der Praxis üblich, weitere Versionen der Software zu entwickeln und in das System zu laden, während die Hardware nicht mehr geändert wird. Dieses Laden der Software-Updates erfolgt zumeist im Feld, so dass hier nur ein Teil der Entwurfswerkzeuge einsetzbar ist.

Aus dieser hardwareseitigen Konstellation eines hochintegrierten SoC, welches zudem noch (wenn auch nur wenige und gut dokumentierte) funktionale Fehler enthalten kann, und der Tatsache, dass ein SoC sehr oft innerhalb einer Systemumgebung mit harten Echtzeitanforderungen arbeiten muss, ergeben sich für den Test und für das Debugging der Software eine Reihe von Restriktionen bezüglich der Beobachtbarkeit und Beeinflussbarkeit von Systemzuständen.

Bereits für Systeme mit nur einem Prozessor besteht seit mehreren Jahren das kontinuierlich wachsende Problem des Softwaredebuggings, da sich innere Systemzustände immer schwieriger beobachten lassen. Die hauptsächlichsten Gründe hierfür sind die Integration von Speicher mit auf den Chip der CPU, so dass die Busse mit externen Werkzeugen nicht mehr zugreifbar sind, sowie hohe Taktfrequenzen und miniaturisierte Schaltkreisgehäuse mit entsprechender Pin-Dichte.

In Folge dessen sind die sehr leistungsfähigen und etablierten Werkzeuge Logikanalysator und In-Circuit-Emulator nur noch sehr eingeschränkt einsetzbar. Da deren Leistung aber unverzichtbar ist, wird seit längerem nach Alternativen gesucht, wobei der grundlegende Ansatz in der Integration von zusätzlicher, nur für Test- und Debugzwecke benötigter Hardware wie Scanpfade und Trace-Logik liegt. Unabhängig von deren Umfang und Leistungsfähigkeit wird immer eine Schnittstelle zwischen dem Prozessor und den Werkzeugen benötigt.

Das Problem des Software-Debuggings verschärft sich für Multi-Core-Systeme weiter. Die beiden Hauptgründe hierfür sind der Systemzugang, der nur ein Mal pro System statt ein Mal pro Prozessor vorhanden ist und die durch die Interaktion der Einzelkomponenten zusätzlich entstehenden und für das Debugging relevanten Informationen.

Auf der Seite der Softwarewerkzeuge existieren derzeit eine Vielzahl von Debuggern, wobei ein konkreter Prozessortyp meist von Debuggern mehrerer Hersteller unterstützt wird und ein konkreter Debugger meist mit mehreren Prozessortypen arbeiten kann. Dabei ist derzeit jedoch nicht sichergestellt, dass mit einem Debugger alle heterogenen Prozessoren eines Multi-Core-SoC behandelt werden können. Somit sind mehrere parallel laufende Debugger notwendig, die aufgrund ihrer zumeist proprietären Implementierung nicht miteinander kommunizieren können. Dieser Umstand stellt für die Softwareentwickler einen entscheidenden Nachteil dar, da keine konsistente Systemsicht existiert.

Die Tatsache, dass für das Debugging nur eine Schnittstelle zum System existiert, wirkt sich nicht nur auf die Hardware restriktiv aus, sondern muss auch auf der Seite der Debugger seine Entsprechung finden. Es sind geeignete Mechanismen erforderlich, welche die gemeinsame und kollisionsfreie Nutzung ermöglichen.

Damit das aus der Hardwaresicht angestrebte unkomplizierte Synthetisieren von SoC aus heterogenen IP im Baukastenprinzip praktikabel wird, sind zusätzlich zu den Anstrengungen bezüglich einer Schnittstellendefinition im Hardwarebereich adäquate Mechanismen im Softwarebereich notwendig.

Ausgehend vom OCP-IP Standard werden im folgenden aktuelle Entwicklungen sowie Standardisierungsbemühungen auf verschiedenen Abstraktionsebenen des Multi-Core-Debuggings vorgestellt.

2 OCP-IP Debug Socket

Das Open Core Protokoll (OCP) ist ein skalierbares und konfigurierbares Interface für die Verschaltung und Kommunikation der Subsysteme innerhalb eines SoC. Es zeichnet sich durch seine Busunabhängigkeit sowie seine umfassende Definition aus.

Die Entwicklung und Lizenzierung des OCP wird von der OCP International Partnership Association, Inc. (OCP-IP) übernommen. Diese ist eine unabhängige Vereinigung von zahlreichen Herstellern, vorrangig aus den Bereichen der IP-Provider und EDA. Die OCP-IP hat es sich zur Aufgabe gestellt, alle mit dem Entwurf, der Verifikation und dem Test von IP-basierten SoC zusammenhängenden Probleme aufzugreifen und allgemein verwendbare Lösungen in Form eines Standards anzubieten.

Ein Nachteil des OCP war, dass sich dessen Definition nicht auf Debugstrukturen erstreckte, so dass diese separat behandelt werden mussten. Im OCP sind zwar die bekannten JTAG-Signale sowie Signale für das Einbinden von Scanchains vorhanden, jedoch ist das aus der Sicht des Softwareentwurfes unzureichend. Breitbandige Schnittstellen für Tracedaten sind im OCP nicht vorgesehen. Weiterhin existieren keine Strukturen oder Mechanismen zur Behandlung der Interaktion der einzelnen Cores, so dass sich ein Multi-Core-Debugging unter diesen Randwerten nur schwer umsetzen lässt.

Während die Realisierung klassischer Run-Control-Mechanismen wie das Starten und Stoppen oder der Einzelschrittbetrieb eines Prozessors sowie das Lesen und Schreiben von Registern vergleichsweise unproblematisch sind, ist die Implementierung einer Trace-Funktionalität eine größere Herausforderung. Mit Hilfe des Tracings lassen sich Aufgaben wie Performance-Messung, Analyse der Codeabdeckung oder Messung der Aufrufhäufigkeit von Funktionen realisieren. Es entstehen bei in Echtzeit laufenden Systemen große Mengen an Tracedaten, welche entweder auf dem Chip zwischengespeichert oder mit entsprechend hoher Geschwindigkeit ausgelesen werden müssen.

Für integrierte Multi-Core-Systeme verschärft sich die Problematik dahingehend, dass die vorhandenen Trace-Ressourcen von jedem Core nur anteilig genutzt werden können. Aktuelle Entwicklungen zielen deshalb auf eine Erweiterung des OCP-Standards um einen Debug Socket ab. Der gegenwärtige Entwicklungsstand ist in [2] veröffentlicht worden.

Der OCP Debug Socket definiert zusätzliche Signale für den OCP Socket, nicht aber die konkrete Implementierung auf der IP-Seite. Desweiteren ist die Definition des externen Debuginterfaces des SoC kein Bestandteil von OCP. Das Hauptziel ist die Definition und Dokumentation von Signalen und Richtlinien für deren Verwendung im Sinne eines Protokolls, welche sowohl für einfache als auch für komplexe Systeme mit mehreren Taktomänen oder verschiedenen Power-Management-Domänen anwendbar sind.

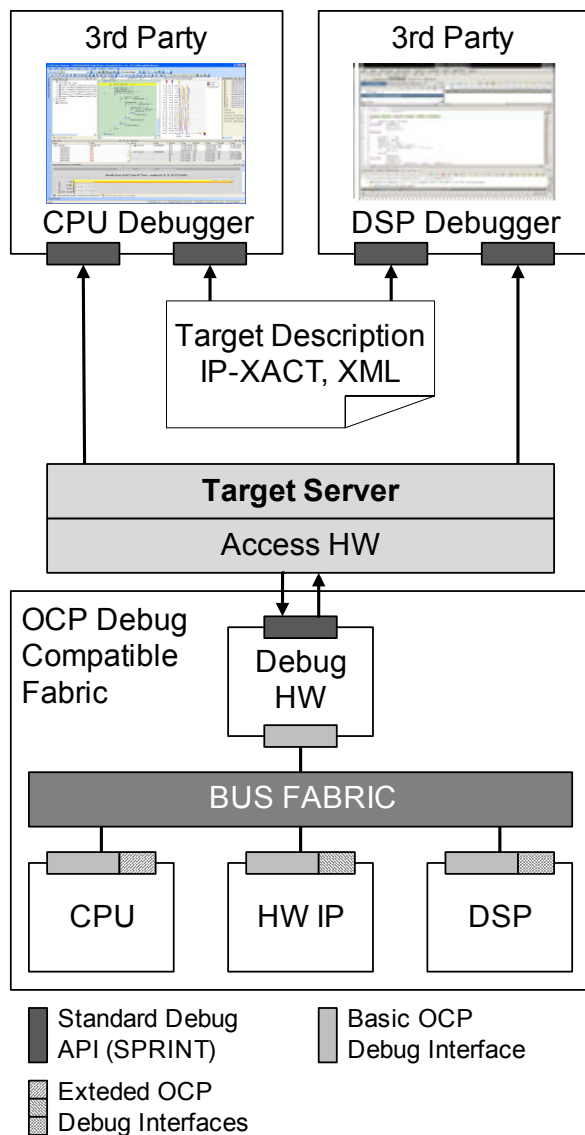


Abbildung 1: OCP-IP Debug Socket und Debug-Softwareinterface

Neben dieser Wrapper-Funktion für die Debugschnittstellen der einzelnen Cores zum Gesamtsystem ist die Behandlung der Interaktion der Cores von besonderem Interesse. Das wesentliche Element hierfür ist ein sogenannter „Cross-Trigger Debug Hardware Block“, welcher in Bezug auf für Ein-Prozessor-Systeme bekannte Debugstrukturen ein neuartiges Element darstellt.

Die Grundidee besteht darin, dass ein Core konfigurierbare Ereignisse (Events oder Trigger) generieren kann, welche innerhalb des Cross-Trigger-Blockes mittels kombinatorischer und zum Teil auch sequentieller Logik zu systemweiten Events kombiniert werden können. Diese ebenfalls konfigurierbaren globalen Events können an die Cores gegeben werden, um diese beispielsweise anzuhalten, das Tracing zu starten oder zu stoppen oder wiederum als Eingangsbedingung für deren lokale Events zu wirken.

Sowohl die lokalen als auch die globalen Events dienen hauptsächlich der Reduzierung des Tracedatenvolumens. Der grundlegende Ansatz besteht in der Bestrebung, nur relevante Trace-Messages in den Puffer zu schreiben bzw. nach außen zu transportieren und den Anteil der nicht relevanten Daten bestmöglich zu minimieren. Dazu ist es erforderlich, die einzelnen Triggerbedingungen in Abhängigkeit der konkreten Analyseaufgabe entsprechend zu setzen.

Weiterhin ist im OCP Debug Socket das systemweite Tracing definiert, wobei zur Wahrung der zeitlichen Relation der Messages der verschiedenen Cores eine Zeitstempelvergabe möglich ist. Die Trace-Messages können entweder unmittelbar über das Debuginterface des SoC ausgegeben oder zunächst zwischengespeichert werden. Beide Szenarien finden im OCP Debug Socket Beachtung.

Dieses Debuginterface wird weder hinsichtlich der Hardware, wie beispielsweise der Art und Anzahl der Leitungen noch bezüglich des Protokolls definiert. Hier geht der Standard davon aus, dass ein existierendes Interface verwendet wird. Favorisiert und als Beispiel angegeben wird Nexus (IEEE 5001) [3].

Im Folgenden sind einige ausgewählte Leistungsmerkmale zur Veranschaulichung der weit reichenden Möglichkeiten des Debug Sockets aufgeführt.

- Es existieren keinerlei Beschränkungen bezüglich elektrischer Parameter sowie Art und Anzahl der Leitungen zum Core.
- Proprietäre Debugstrukturen werden durch die Verwendung eines OCP Debug Wrapper kompatibel gemacht und damit ebenfalls unterstützt.
- Der Socket ist unabhängig vom physikalischen Interface zwischen SoC und Debughost.
- Es existiert ein komplexes, hierarchisch aufgebautes Trigger-System zur Extraktion von lokalen und systemweiten Events.
- Konfigurierbarkeit des Trigger-Systems bezüglich der umfangreichen Trigger-Modi
- Unterstützung von Performance-Analyse und Profiling mittels programmierbarer Event-Counter
- Zyklen-akkurater Echtzeit-Trace von Multi-Core- / Multi-Bus-SoC
- Übersetzung der Trace-Rohdaten in kompakte Trace-Messages, um das Datenvolumen zu minimieren
- Trace-Messages können beispielsweise folgendes enthalten: Prozessor-ID, auslösender Break- oder Watchpoint, Programmverzweigungen, etc.
- Konfiguration der Aufzeichnungsmodi des Trace-Speichers bezüglich der Anteile für Messages, welche vor bzw. nach einem Event liegen

3 Standard Debug API (SPRINT)

Die Komposition von SoC aus vordefinierten IP-Cores muss auf der Seite der Softwarewerkzeuge adäquat unterstützt werden, um die durch die Hardware zur Verfügung gestellten Möglichkeiten auch tatsächlich zu nutzen. Ein wesentlicher Bestandteil der Werkzeuge ist der Debugger, welcher hier im Sinne eines Debuggers für das gesamte SoC zu verstehen ist. Dessen wichtigstes Leistungsmerkmal ist die Herstellung einer konsistenten Sicht auf das Gesamtsystem.

Die Schaffung eines speziell auf ein konkretes SoC zugeschnittenen Debuggers ist aus Effektivitätsgründen unrealistisch. Es sind also Lösungsansätze erforderlich, welche die für die einzelnen Cores ohnehin vorhandenen Debugger zu einem systemübergreifenden Werkzeug kombinieren. Hierfür sind zwei Voraussetzungen zu schaffen. Erstens sind geeignete Mechanismen für die gemeinsame und wechselwirkungsfreie Nutzung des Debuginterfaces zur Systemhardware notwendig. Der zweite wesentliche Punkt ist die Behandlung der Core-Interaktion, wobei die Steuerung der Cross-Trigger-Hardware den Schwerpunkt bildet.

Wegen der Vielzahl möglicher Kombinationen von IP-Cores und damit SoC sollten die Debugwerkzeuge in Analogie zu den Hardwarestrukturen hierarchisch aufgebaut sein und ebenso standardisierte Interfaces besitzen, um eine einfache Austauschbarkeit zu gewährleisten. Die auf der Seite der Hardware implementierten Synchronisationsmechanismen zwischen den Cores müssen sich auch im Debugger widerspiegeln.

Ein mögliches standardisiertes Interface zu Debug- und Tracewerkzeugen ist die durch das SPRINT Projekt (Open SoC Design Platform for Reuse and Integration of IPs) definierte Debug API [4]. Diese ist als Multi-Core Debug API konzipiert und enthält Funktionen wie Tool-Konfiguration und Zugriff auf die Steuerung des Debuggings, des Tracings sowie eine Ausnahmenbehandlung. Während einer Systemidentifikationsphase werden die Eigenschaften des Gesamtsystems und die der einzelnen Cores abgefragt. Diese Informationen dienen der Konfiguration der Debugwerkzeuge.

Um diese Funktionalitäten nicht nur in der realen Hardware sondern bereits zum Zeitpunkt der Simulation nutzen zu können, wird im SPRINT Projekt vorgeschlagen, diese Debug API mit entsprechenden Funktionen auszustatten.

4 Target-Beschreibung mit IP-XACT

Ein weiterer essentieller Bestandteil einer standardisierten Debugumgebung für Multi-Core-Systeme ist die detaillierte Architekturbeschreibung der einzelnen Cores. Während der Systemidentifikationsphase werden zwar die einzelnen Cores sowie das Gesamtsystem identifiziert, jedoch ist der weitaus größere Teil der zur Konfiguration der einzelnen Debugger notwendigen Daten nicht in der Hardware des SoC hinterlegt.

Die zur automatisierten Konfiguration eines Debuggers notwendigen Informationen lassen sich in drei grundlegende Teile aufgliedern. Die Basis bildet die Architekturbeschreibung der Hardware aus Softwaresicht. Hierzu zählen insbesondere alle Details der Prozessoren und Peripherieelemente wie z.B. Register- oder Speicher-Mappings. Zusätzlich zu dieser Strukturbeschreibung sind Informationen über den Zugriff auf die Hardware aus der Sicht der Werkzeuge erforderlich. Den dritten Teil bilden Informationen über das Zusammenwirken der Werkzeuge untereinander. Insbesondere die von Compilern generierten Informationen zur Unterstützung des Debuggings stehen hier im Fokus der Betrachtungen.

Erstrebenswert ist auch auf dieser Abstraktionsebene eine einheitliche Form der Beschreibung von Debugfunktionalitäten in Bezug auf architektonische Merkmale. Eine Möglichkeit ist das vom SPIRIT Consortium definierte IP-XACT [5], welches zunehmend zur Beschreibung von IP verwendet wird.

Bei IP-XACT handelt es sich im Wesentlichen um eine XML-Datenstruktur mit einer Schemadefinition. In dieser Struktur hinterlegt werden Metadaten zur Weiterverwendung während der Entwicklung, Implementierung und Verifikation von elektronischen Systemen. Zusätzlich definiert wird in diesem Standard ein Generator-Interface zu einem von den konkreten Werkzeugen unabhängigen Zugriff auf die Metadaten.

Aus dem aktuellen Entwicklungsstand heraus muss hier einschränkend erwähnt werden, dass die Erweiterungen des zugrundeliegenden und ursprünglich für die IP-Integration vorgesehenen XML-Schemas für Debug-Zwecke noch im Standardisierungsverfahren [6] sind. Damit sind diese derzeit noch nicht nutzbar und auch nur fragmentarisch bewertbar. Es lassen sich jedoch schon folgende Fakten ableiten:

1. Debugregister, welche zur Steuerung und Konfiguration der Debugfunktionalität in Debugkomponenten dienen, können mittels IP-XACT in Analogie zu allen anderen Registern im System strukturell beschrieben werden. Adressierungen, Zugriffsmodi und die Registerstruktur (Bitfelder) sind dem Debugger damit bekannt.
2. IP-XACT-Interfacedefinitionen erlauben die Beschreibung von Cross-Trigger-Verbindungen zwischen den einzelnen Debugkomponenten und sind so auf der Verdrahtungsebene abbildbar.
3. Semantische Informationen zur Debugfunktionalität, z.B. welche Registerwerte eine bestimmte Debugoperation auslösen, sind nicht verfügbar.

Eine einheitliche Lösung für den letzten Punkt würde es vor allem für komplexe Debug- und Tracehardware in Multi-Core-Systemen gestatten, möglichst flexible Debugwerkzeuge zu entwickeln. Diese können dann ohne große Änderungen des Programmcodes an unterschiedliche Targets angepasst werden.

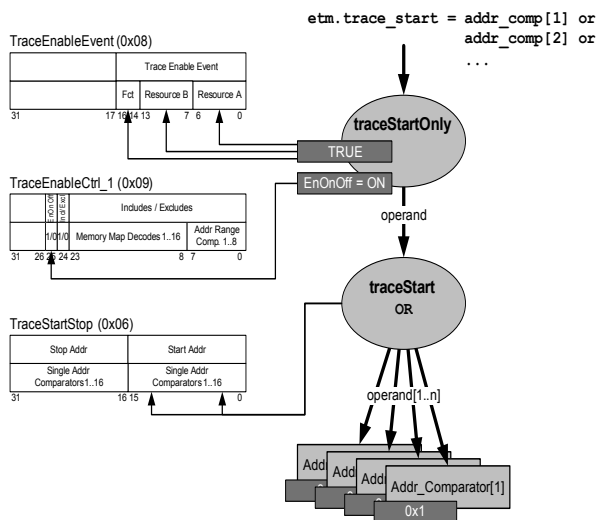


Abbildung 2: Abbildung von Debugoperationen auf Debugregister

Ein erster, in diese Richtung weisender Ansatz wurde in [7] mit dem Ziel vorgestellt, ein Konfigurationswerkzeug für komplexe On-Chip-Trace Hardware [8] an unterschiedliche On-Chip-Debugarchitekturen anzupassen. Eine an IP-XACT angelehnte, XML-basierte Komponentenbeschreibung des Debugsystems wird dabei um die fehlenden semantischen Informationen erweitert. Ziel ist es, die Konstrukte der Konfigurationsprache, welche dem Werkzeug zugrunde liegt, architekturunabhängig auf die Debugregister abzubilden. Dazu wird eine Baumstruktur aufgebaut, welche durch Traversierung nach und nach aus Debugoperationen entsprechende Belegungen der Debugregister erzeugt (Abbildung 2).

5 Zusammenfassung

Die Integration von mehreren programmierbaren Cores in ein SoC wirft auf nahezu allen Abstraktionsebenen der Hard- und Software das Problem von konsistenten und von proprietären Lösungen möglichst unabhängigen Beschreibungsformen auf. Ein Teilaspekt davon ist das Softwaredebugging, wobei eine enge Verzahnung mit den reinen Hard- und Softwareaspekten besteht.

Anstrebenswert ist die Definition und konsequente Nutzung von geeigneten Standards. Auf der Ebene der Debughardware ist mit dem vorgestellten OCP-IP Debug Socket eine geeignete Lösungsvariante vorgeschlagen worden. Darauf aufsetzend wird mit der Standard Debug API (SPRINT) ein standardisiertes Interface zu Debug- und Tracwerkzeugen definiert.

Zur Konfiguration von universellen Werkzeugen zum Debugging müssen diesen die speziellen architektonischen Merkmale des Systems in geeigneter Weise zugänglich gemacht werden. Eine Möglichkeit hierzu ist IP-XACT, wobei dessen Erweiterung um semantische Informationen für das Softwaredebugging ein noch offenes Arbeitsfeld darstellt.

6 Literatur

- [1] Open Core Protocol International Partnership (OCP-IP); <http://www.ocpip.org>
- [2] OCP-IP Debug Working Group: Open Core Protocol Debug Specification 1.0; 2008; <http://www.ocpip.org>
- [3] Nexus 5001 Forum: IEEE-ISTO 5001™-2003: The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface; <http://www.nexus5001.org>
- [4] A. Mayer: SoC Debug Methods; Forum on Specification & Design Languages (FDL'07) – Workshop on SPRINTing IP Integration; Barcelona; 2007
- [5] The SPIRIT Consortium: IP-XACT v1.4: A Specification for XML Meta-Data and Tool Interfaces; <http://www.spiritconsortium.org>
- [6] A Berent: IP-XACT for Debug; ECSI Institute Workshop on System Debug; München; 2008
- [7] J. Braunes: Managing Complex Trace Filtering and Triggering Capabilities of CoreSight™; ARM Developers' Conference & Design Pavilion; 2007
- [8] K.-U. Irrgang, J. Braunes, R.G. Spallek, S. Weiße, T. Gröger: Konfiguration komplexer Test- und Debug-Hardware in System-on-Chip; Dresdner Arbeitstagung Schaltkreis- und Systementwurf (DASS'05); Dresden; 2005