Power.org™ Common Debug Interface Technical Subcommittee White Paper on a Common Debugging Framework

Felix Burton
Wind River

Chris Ng
IBM

Erich Styger
Freescale Semiconductor

Draft

Copyright © 2007 Power.org. All rights reserved.

The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.
Implementation of certain elements of this document may require licenses under third party intellectual property rights, including without limitation, patent rights. Power.org and its Members are not, and shall not be held, responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.
THIS POWER.ORG DOCUMENT IS PROVIDED "AS IS" AND WITHOUT ANY WARRANTY OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTY OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL POWER.ORG OR ANY MEMBER OF POWER.ORG BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, PUNITIVE, OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Questions pertaining to this document, or the terms or conditions of its provision, should be addressed to:

IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Power.org Board Secretary

Table of Contents

Abbreviations 2
Abstract 2
Overview 2
The Problem 3
The Vision 4
Target Communication Framework (TCF) 5
Protocol 5
Services 6
Typical Use Case 6
Conclusion and Outlook 7
References 7
Acknowledgements 7

Abbreviations

API	Application Programming Interface
CDI	Common Debug Interface
TSC	Technical Sub-Committee
TCF	Target Communication Framework
SoC	System on Chip
GDI	Generic Debug Interface
IP	Intellectual Property
VHDL	VHSIC hardware description language
RTL	Register Transfer Level
RTOS	Real Time Operating System
JSON	Java Script Object Notation
OCD	On Chip Debugging

Abstract

This paper proposes a new approach to enable debugging — and more — on embedded targets. Traditional Application Programming Interfaces (API's) and approaches are not scalable enough, universal enough, or open enough to solve the challenges of new, complex, multi-core chips. Furthermore, traditional APIs and approaches do not enable all involved parties in any easy way. This paper presents a new approach that is not limited to the any architecture — an approach that is flexible, powerful, and scalable enough to become an industry wide standard.

Overview

The Power.org Common Debug Interface TSC has chartered a workgroup to define an API for the Power™ architecture debug environment. (The CDI TSC scope document defines this environment.) This workgroup — the API team — consists of members from Freescale, IBM, Lauterbach, Mentor Graphics, Virtutech, and Wind River. Team members looked at existing standards, concluding that we need something more scalable, flexible, and powerful.
The purpose of this paper is to document the vision of the API team for resolving the lack of a unified and scalable communication mechanism between the target and the host for the purpose of debug, performance analysis and other development activities. It gives a high-level view of the problem, the proposed communication framework, and examples of communication services. (For more detailed information about such services, readers should see to API specification to be published by the Common Debug Interface TSC.)

The Problem

Current Power Architecture™ implementations lack uniform debug interfaces, environments and methodologies. This means that tool vendors must develop multiple software and hardware configurations to support the different Power Architecture™ processors, system on chips and cores. These configurations include proprietary APIs for communications between debug hosts and targets, which results in a unique agent, connection, protocol and setup for each tool. This also means that any new feature must be added in multiple places, increasing code verification time and maintenance effort.

The advent of multicore processors imposes a need for new tools and new features in existing tools. To break the cycle of having to create unique agents, connections, protocols and setup for new tools, a new architecture is needed that allows multiple tools to share the same communication link.

Figure 1 depicts the current situation: no interoperability among debug hosts and targets.

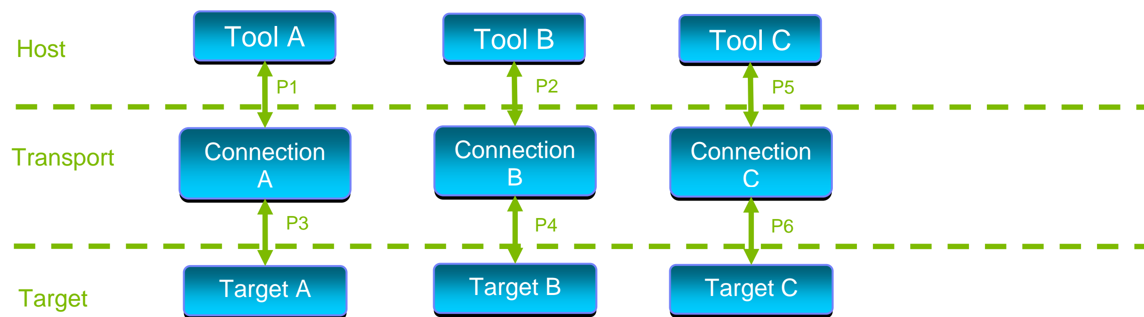

Figure 1 Current, Unique Debug-Host to Target Connections

The first issue for such an ecosystem is the unique protocol for each block. In Figure 1, labels P1 through P6 indicate these protocols. Existing standards such as IEEE P1149.1 (JTAG) do not cover all aspects or levels of today's complicated systems. A new standard must cover all aspects from debug agents to hardware run control. It must cover all levels, from trace low-level run control (such as stepping) to resource management (such as reads and writes of registers and memory). The introduction of multicore architectures further increases system complication.

Traditionally, tool vendors defined their own protocols for communication between the tool portion on the host and the target part of the hardware or simulation. But this approach leads to several problems:

- Inability to find or leverage one tool's functionality in other tools, forcing users to adopt point solutions.
- Unique target configurations for each tool, unique configurations for each protocol, even incompatible configuration languages.
- Inability to share among agents: even if the same target can run multiple software agents (such as debugging helpers, trace collectors, or streaming agents), each agent requires its own protocol or host connection.
- Duplicated maintenance effort: each vendor or tool must maintain its own solution, without leverage from other solutions.
- Inability to add new features at one place: any new target capability (such as advanced cross triggering functionality) forces every vendor to add its own support for the feature.
- Increased target footprint: as embedded application functionality grows, the remaining space for debugging agents decreases. Users need a scalable way to add and remove capabilities.

- Incomplete target matrix, such as serial connectivity that some tools support, but other tools do not.
- Difficult ecosystem: restrictions or limitations that unnecessarily make it harder for silicon vendors and tools vendors to build bases that let everybody build on common things.

Another issue is the tunnel vision of existing standards — too-tight focus on a specific topic. For example, a standard such as GDI covers basic run control, but is only rudimentary with regard to trace. The challenge is defining new standards that we can enhance without harming their coverage of the basics.

Yet another issue is scalability: the design must enable the API and framework to be designed in a way to cover scenarios, including:
- For the IP designer working with any kind of simulators, such as RTL or VHDL based systems.
- For the silicon vendor covering cover silicon validation and low level silicon inspection.
- For the first target adopters performing board bring-up.
- For the firmware and low level software developer working on u-boot and firmware deployment.
- For the RTOS group running and debugging operating systems.
- For the driver software engineer developing drivers and inspecting protocols.
- For the application developer developing and debugging the applications.
- For the system engineer combining all of the above specified actions, creating a system-wide, complex, multicore environment.

The challenge is creating an efficient and scalable solution for all these issues, a solution that lets all tools use the same protocol and infrastructure. Furthermore, this solution should work for all systems: from high-level systems rich in resources, to low-level systems very restricted in resources.

The Vision

Figure 2 shows the vision: standardized services that let various tools interoperate with different targets. A well-defined, extensible transport protocol makes it possible to add services dynamically. It also makes it possible to have a single configuration per target as well as dynamic discovery of targets and services.

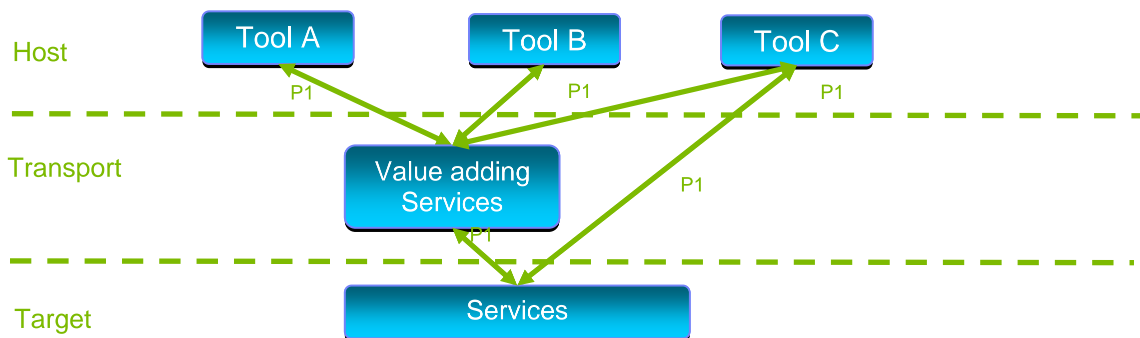

Figure 2 Ideal Debug Host to Target Connections

This approach defines a lightweight, small footprint, open and vendor agnostic way for tools and targets to communicate. It is important that there is just one configuration per target, instead of today's usual configuration per tool per target. The protocol must be simple yet extensible. The whole system must be able to use that same protocol, so that commands, replies, and events can

be passed thought the value adding service layer without having to do protocol conversion for services that has nothing to do with the value add.

The protocol and architecture must accommodate both slow and high latency connections. They should work in the high-speed, gigabit Ethernet environment, on an RS-232 interface of only 9600 baud, as well as over the internet where the throughput can be very good, but the round trip time can be very long. The whole system must be agnostic with regard to the transport protocol.

For automatic connectivity, the new API must include dynamic discovery of available boards and services. For scalability, the new API must include the ability to add, remove, run, and stop services. For flexibility, the new API must be generic enough to be applicable to multiple environments, for example, the same breakpoint interface that a probe needs also must be applicable to a target-resident agent.

Target Communication Framework (TCF)

The problem statement and vision above lead to a service based, communication framework: the Target Communication Framework (TCF).

TCF consists of two different layers:

1. Protocol
2. Service

Protocol

The TCF protocol lets multiple tools, such as debuggers, analysis and test tools, communicate with target boards, simulators or probes over a shared link. In other words, once the link is configured and works for one tool, it works for all tools. Multiple tools may use different, possibly, overlapping sets of services.

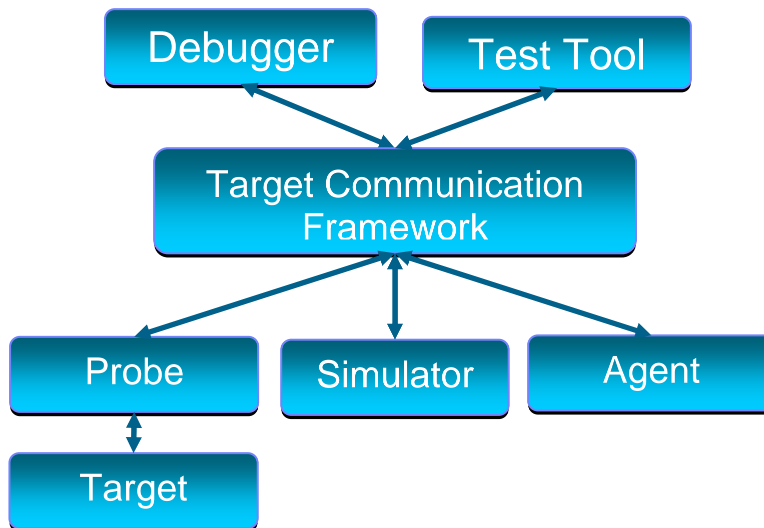

Figure 3 Debugger Connections

The protocol allows commands, replies and events to be sent between the tool and the target in a reliable, ordered and transport independent way. The other important aspect of the TCF protocol is that it allows easy pass-through and forwarding of commands, replies and events. A current prototype is using a text based Java Script Object Notation (JSON) protocol for easier logging and debugging; however this may be changed in the future.

Services

A service is defined as a set of commands, replies and events with a well defined syntax and semantics. Services should be carefully defined to allow them to be:

1. Simultaneously used by multiple independent clients
2. Applicable in multiple environments e.g. probes, simulators and agents
3. Allow value adding layers to enhance the capabilities and information of the service

In order to accomplish goal #2, it is important to minimize the need for clients to have hardcoded knowledge about the system. This can be accomplished by allowing the client to query the service for information about the system.

It is recognized that targets frequently have limited resources and therefore cannot afford large agents or services. Because of this, the TCF protocol and services allow value adding services to be placed between the tool and the target. Examples of value adding services are to convert source files and line numbers to addresses and vice versa, add operating system awareness, etc

The expected set of initial services is:

- **Run Control Service** — for starting, stopping, and other basic run control.
- **Breakpoint Service** — for managing breakpoints.
- **Register Service** — for read from and writing to registers.
- **Memory Service** — for reading from and writing to memory.
- **MMU Service** — for dealing with the Memory Management Unit (MMU).
- **Cache Service** — for dealing with caches.
- **JTAG Service** — for a low level interface to a JTAG device.
- **JTAG Configuration Service** — for configuring a low level JTAG device.

Typical Use Case

Figure 4 shows a typical use case:

- Eclipse is the user interface to a debug engine, and
- The debug engine is connected to a target or simulator.

You need not use TCF as the connection mechanism between each block of Figure 4. But if each connection is TCF based it can significantly simplify the implementation of the blocks.
The picture shows two connection paths to the target:

1) To a target agent — for example, over TCP/IP
2) A JTAG connection through a JTAG probe.

The dashed line between the JTAG probe and the target depicts tunneling TCF commands over the JTAG channel to the target agent. It is possible to have all these connections simultaneously, but that may not be typical. The target even can be a bare board — that is, one without any agent running. In such a case, the only connection is through the JTAG probe — but you still can use TCF to communicate between the debug engine and the JTAG probe.

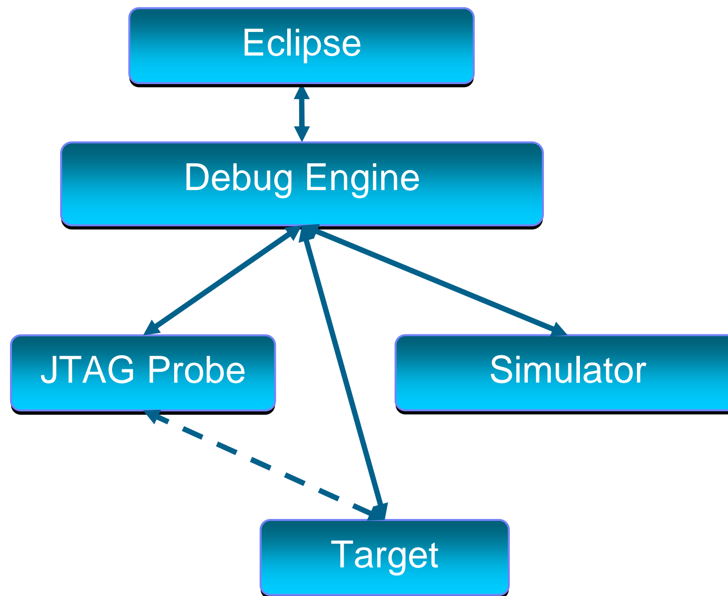

Figure 4 Typical Use Case

Conclusion and Outlook

TCF provides a new approach and vision for a common debug interface. It is open, extensible and allows different vendors to contribute and add services to the framework. The approach is scalable and with the asynchronous protocol it supports both high and low latency connections and transport layers. With both silicon and tool vendors adopting TCF, this will be the foundation of the next generation debug standard interface. The outlook is to have the TCF approach and framework contributed to the open source community so it can be even more widely adopted.

References

- FSLDBG SDK Development Guide, Rev 16. Freescale, March 2007
- Command Converter Server API (CCSAPI), Version 4.5.2, 16-Oct-2003, Freescale
- Generic Debug Instrument Interface (GDI), Revision 1.2.6, January 1998, Tasking. (Downloadable from http://www.tasking.com/request/?code=gdikdi)
- Universal Debugger Interface (UDI) Specification, Version 1.4, Revision 3, 17-Aug-1995, Advanced Micro Devices (http://www.amd.com/epd/29k/extra/udi_spec/udi_spec.pdf)
- OCD API Reference Guide, Rev 3.3, Wind River, March 2007
- MIPS MDI specification: (Downloadable, with registration from: http://www.mips.com/content/Documentation/MIPSDocumentation/ProcessorArchitecture/EJTAG/MD00412-2B-MDI-SPC-02.12.pdf/getDownload)
- Eclipse and DSDP project: http://www.eclipse.org/dsdp

Acknowledgements

The authors extend their thanks to Power.org and to their respective employers for their continued support and contributions. The authors also thank other workgroup members and industry partners for their fruitful discussions and useful comments.