



Power.org™ Target Debug Capabilities Specification

Version 1.0 – 9 May 2008

Copyright © 2008 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 SPECIFICATION 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

Introduction

Power.org's mission is to develop, enable, and promote Power Architecture™ technology as the preferred open standard hardware development platform for the electronics industry and to administer qualification programs that optimize interoperability and accelerate innovation for a positive user experience. Power.org seeks to solicit the participation of all interested parties on a fair, equitable, and open basis.

Power.org's output includes:

- Open standards and specifications
- Business guidelines documents
- Best practices and education
- Certifications to validate implementations and drive adoption

Power.org's specifications will enable:

- Interoperability between community members
- Sustainability built on driving open standards and convergence

Revision History

Version	Date	Editor	Description
.1	5/17/2007	Dan Helm	1 st Pass TGT Group Recommendations
.2	6/24/2007	Dan Helm	Reformatted to more formal style.
.3	7/09/2007	Mike Johnson	Additional reformatting and suggested changes for group review.
.4	7/10/2007	Dan Helm	Updated per 7/10/07 Meeting.
.5	8-27-2007	Hunter Scales	Edited the Trace section to make it more general and include Nexus functionality.
.6	9-11-2007	Dan Helm Hunter Scales	Removed Levels and reduced to only optional and required. Updated Non-Trace System-Level Features and Trace Features.
.7	10-07-2007	Hunter Scales Chris Ng	Added Single Step & format updates. Reformatted.
.8	10-29-2007	Chris Ng	Reformatted to specification style.
.9	10-31-2007	Dan Helm	Added Instruction Jamming section.
.9	1-15-2008	Chris Ng	Edited based on comments from AMCC TC rep Edited based on comments from Marketing Committee Chair Added Section 5.4
.9e	1-23-2008	Hunters Scales	Edited Section 3.9
.9f	2-4-2008	Chris Ng	Changed “must” to “shall”. Changed “can” to “may”. Grammar changes. Changed copyright year to 2008. Changed document date.
.10	3-10-2008	Chris Ng	Corrected misspellings
1.0	5-9-2008	Joan Woolery	Spec approved by the board. Removed DRAFT and “Power.org Internal” from the header and footer.

Table of Contents

Introduction	2
1 Overview	7
1.1 Scope	7
1.2 Purpose	7
2 Terminology	7
2.1 Definitions	7
2.2 Acronyms.....	7
3 Non-Trace Core-Related Features	8
3.1 Halted and Frozen States	8
3.1.1 Halting Mechanisms	9
3.1.2 Freezing Mechanisms	9
3.2 Exiting and Resuming Halted or Frozen States	9
3.2.1 Resuming Mechanisms	10
3.3 Status	10
3.3.1 Status Mechanisms.....	10
3.4 Event Signaling.....	10
3.4.1 Signaling Mechanisms	10
3.5 Instruction Jamming.....	10
3.6 Read or Alter Core Resources	11
3.7 Debug Event Counter.....	11
3.8 Multithread	12
* Required if multithreading is implemented.....	12
3.9 State After Reset	12
3.10 Single Step.....	13
3.11 Debug Hardware Limits.....	14

4	Non-Trace System-Level Features.....	14
4.1	Required Features	14
4.1.1	Halting	14
4.1.2	Status	14
4.1.3	Resume	14
4.2	Optional Features.....	14
4.2.1	Freezing	14
4.2.2	Self-Discovery	15
4.2.3	Cross-Triggering.....	15
5	Trace Features	15
5.1	Required Features	16
5.1.1	Program Trace	16
5.2	Optional Features.....	17
5.2.1	Data Trace	17
5.2.2	Flow Control.....	17
5.3	Trace Port	17
5.4	Interoperability	18

Power.org™ Target Debug Capabilities Specification

1 Overview

1.1 Scope

The goal of the Target group (TGT) of the Common Debug Interface Technical Subcommittee (CDI TSC) of Power.org is to define capabilities to observe the system under test with minimal (or no) intrusion on its normal operation. These capabilities will be directed at, but not limited to, a set of functions that the trace interface will implement, synchronized debugging in a multiprocessor or core environment, and scaleable debugging facilities on the target. This document should be seen as an extension of the debugging features provided by the Power ISA™ specification. (<http://www.power.org/resources/downloads/>)

1.2 Purpose

The purpose of this document is to capture the discussions of the Target group while defining a common set of debugging requirements for Power Architecture–based products. The target features are divided into required and optional features. The optional features are included as best-practice features, although the total cost of implementation might be too high.

2 Terminology

2.1 Definitions

Halt: All instructions in the pipeline have finished, and no new instructions have been fetched.

Freeze: Core clocks other than those needed for TAP access are no longer running.

2.2 Acronyms

CDI TSC	Common Debug Interface Technical Subcommittee
DE	Debug Event
iJAM	Instruction Jam
LOS	Loss of Synchronization

Power ISA™	Power Instruction Set Architecture
SE	Synchronization Event
SOC	System on Chip
TAP	Test Access Port
TGT	Target Team – a sub team of the CDI TSC

3 Non-Trace Core-Related Features

This section pertains to those features that are considered standard debugging features for cores that are unrelated to trace. It is assumed that each core will have a JTAG interface. However, with the new multicores and system-on-chips (SOC), new core debug interfaces might become available. All core-specific features with JTAG capabilities shall be able to be accessed either directly through JTAG or from a system-level interface that translates from JTAG. This requirement ensures a common physical interface for debuggers. Included in the section are the following topics:

- Halted and Frozen States
 - Halting Mechanisms
 - Freezing Mechanisms
- Exiting Halted/Frozen States, Resuming
- Status
- Event Signaling
- Read/Alter Core Resources
- Debug Event Counter
- MultiThread
- Reset
- Single Step
- Debug Hardware Limits

3.1 Halted and Frozen States

When a core is in the halted state, the clocks are still running, but the core is not fetching or executing instructions. While in this state, an external debugger may jam instructions (see Instruction Jamming section) into the pipeline or alter the core by any other provided method. If a system implements HW cache coherency, the core will continue to snoop the core complex bus and will maintain cache coherency. Behaviors of the core during access of core-specific, nonarchitected features are also core-specific. Timer and counter behavior while in a halted state is determined by the implementer. It is recommended, however, to pause all timers and counters while the core is halted for debug purposes.

When a core is in the frozen state, the primary core clocks are frozen, but the debug control unit should still be active. The caches are not snooped. If the clocks are frozen while the caches contain modified data,

coherency might be lost because other bus masters cannot see the modified data. Coherency might also be lost if the caches contain shared or exclusive data when the clocks are frozen and then the clocks are restarted. In this case, other masters might have changed the data while the frozen core retains the stale data, which might be used when the clocks are restarted.

There are several mechanisms for halting and freezing the cores. Each core shall have a status mechanism that can be read by the debugger to indicate what caused the state.

3.1.1 Halting Mechanisms

Mechanism	(R)required/(O)ptional	Comments
JTAG Command Based	R	May be single command or sequence
Signal Based	O	Aids in <i>synchronous</i> halting of multiple cores in a system. Shall be separate from any power management features.
Event Based	R	Minimum of DBCR events is required; masking of events for external debugging is optional, though useful. The provider may determine other events.

3.1.2 Freezing Mechanisms

Mechanism	(R)required/(O)ptional	Comments
JTAG Command Based	O	May be single command or sequence
Signal Based	O	Aids in <i>synchronous</i> freezing of multiple cores in a system. Should be separate from any power management features.
Event Based	O	

3.2 Exiting and Resuming Halted or Frozen States

Resuming is the process of moving the core from a halted or frozen state back to normal core execution. All halted states can be resumed. Whether frozen states, which are core-dependent, can be resumed depends on how the state was reached. Resuming mechanisms are listed in the following table. Before the core actually resumes, all halting or freezing mechanisms shall be cleared. In addition to the listed mechanisms, each core has a RESET signals and a TRST (or equivalent) signal, which can also be used to resume, although the core's state can change (see Reset section).

3.2.1 Resuming Mechanisms

Mechanism	(R)quired/(O)ptional	Comments
JTAG Command-Based	R	May be single command or sequence
Signal-Based	O	Aids in <i>synchronous</i> resuming of multiple cores in a system. Should be separate from any power management features.

3.3 Status

3.3.1 Status Mechanisms

Mechanism	(R)quired/(O)ptional	Comments
JTAG Command-Based	R	May be single command or sequence
Signal-Based, Halted or Frozen	O	Aids in reading status of multiple cores in a system from system or platform logic.
Signal-Based, Running	O	
Signal-Based, Reset Indicator	O	

3.4 Event Signaling

As systems change to multicore environments, the importance of a core's ability to signal and react to debug events in the system increases. To aid in this increased capacity, two signals are recommend.

3.4.1 Signaling Mechanisms

Mechanism	(R)quired/(O)ptional	Comments
Signal-Based, Debug Event (DE) Out	O	Asserted as a minimum ORing of DBSR and control bit, recommended maskable of DBSR and control bit
Signal-Based, Debug Event (DE) In	O	Takes a debug event, maskable to halt. UDE is acceptable.

3.5 Instruction Jamming

Instruction jamming is the ability to take a core in a halted state, insert an instruction into the core's execution path as though it had been fetched, and have the instruction executed. Included in this process is generalized mechanism for performing debugging operations with the processor's existing facilities.

Instruction jamming varies by implementation, although typical instruction jams include mtspr/mfspr, loads, and stores.

3.6 Read or Alter Core Resources

Core resources may include the following items:

- Registers, including core-level registers accessible directly from the debugging interface
- Arrays
- Caches
- Any other hardware entities in the core definition that are accessible from the instruction set

Mechanism	(R)required/(O)ptional	Comments
iJAM	R	All core resources accessible from the instruction set should be accessible by instruction jamming.
Scan	O	Additional core resources not accessible from the instruction set might be accessible from scan chain access. Some core resources accessible by instruction jamming might also be accessible from scan chain access.
Scan while Running	O	The DBSR should be readable by scan while the core is running (see also Trace section).
Software Hardware Communication Channel	O	A mechanism for software to write and read the same registers as the debugger without halting the core.

3.7 Debug Event Counter

A Debug Event Counter is an optional feature that counts the number of debug events that occurred. It has the option to take action based on a given threshold. An example of this feature would be the ability to halt the core after the completion of each ten instructions.

Mechanism	(R)quired/(O)ptional	Comments
Count Debug Events	O	A control register shall be able to mask the debug events to count.
Count Threshold	O	Enables the debugger to take action when a threshold count of debug events is exceeded.

3.8 Multithread

This feature aids in implementing multithreaded debug thread runtime control features.

Mechanism	(R)quired/(O)ptional	Comments
Thread switching	R*	Change thread for core access (facility read/alter and iJAM) and activate thread upon resuming
Thread disable	R*	

* Required if multithreading is implemented

3.9 State After Reset

At a minimum, the processor shall support two types of reset signals: one to reset the processor and initialize all the state within it and one to reset the JTAG interface. TRST is the optional JTAG signal which is used to reset the JTAG TAP controller state machine. TRST shall not be used to reset any logic within the processor except the JTAG tap controller. Processor reset can also be caused by a JTAG command or sequence of JTAG commands. If the processor implements a Halt command in the external debug registers (accessible thru JTAG), and also allows the Halt state to remain in effect after a processor reset, then some other external reset signal may be implemented to reset the external debug registers, including the Halt state bit.

Mechanism	(R)quired/(O)ptional	Comments
RESET without Execute	R	There shall be a mechanism for the debugger to reset the processor and have it halt before any instruction is executed. This type of reset shall not affect the external debug interface hardware.
RESET without Fetch	O	A mechanism for the debugger to reset the processor and have it halt before any instruction is fetched. This feature prevents potential bus hangs caused by uninitialized

		bridges, for example. This type of reset shall not affect the external debug interface hardware.
Halt after Reset Mode	O	A mode that the debugger can put the processor into, which will cause it to halt after the next reset (whether that reset is induced by the debugger or by external reset signal).
Preserve Halted or Frozen Core States across Reset	O	If a core is halted or frozen and in external debugging mode, it should ideally remain in the halted or frozen state after RESET

3.10 Single Step

Single stepping through instructions is a basic function of external hardware debugging. Using the features described in this document and those provided in the Power ISA™, single stepping may be accomplished by the following debug events:

- Halting on the event
- Instruction Complete
- Interrupt Taken
- Others provided in the Power ISA™

In addition to halting on debug events, the cores may also implement the DNH (provided in the Power ISA™) instruction, which can be inserted into the code and used as a single step methodology. While these methods should cover all debug scenarios, implementers might feel that they are too time consuming for a feature as commonly used as single stepping. As such, the recommended methodology to overcome this issue is to have a single JTAG command or series that implement the single-step feature.

Mechanism	(R)required/(O)ptional	Comments
Single Step by Halting on Debugging Events	R	Required by Power ISA™ and Halting on Debugging Event
Single Step by inserting DNH into instruction stream	R	Required by Power ISA™
JTAG Command-Based	O	May be single command or sequence

3.11 Debug Hardware Limits

Mechanism	(R)required/(O)ptional	Comments
JTAG Clock Valid Frequency	R	The processor's JTAG clock shall be capable of operating over a minimum range of 1KHz to a maximum a 100MHz. (Faster speeds are acceptable though possibly not supported by debugging probes.)

4 Non-Trace System-Level Features

This section pertains to those features that are considered standard debugging features at a system level that are unrelated to trace.

4.1 Required Features

4.1.1 Halting

Each system shall have a JTAG command (or series) to simultaneously halt all cores in the system. It is recommended to have core-level signals that system-level debugging logic can assert to perform this function. However, it is possible to have a system-level controller sequence through each core's debugging interface to halt, as long as the halting is reasonably simultaneous.

4.1.2 Status

Each system shall have a JTAG command (or series) to simultaneously read the status of all cores in the system. It is recommended to have core-level signals to assist in this task, as in Halting. However, it is possible to have a system-level controller sequence through each core's debugging interface to read the status.

4.1.3 Resume

Each system shall have a JTAG command (or series) to simultaneously resume all cores in the system. It is recommended to have core-level signals to assist in this process, as in Halting. However, it is possible to use a system-level controller sequence through each core's debugging interface to resume.

4.2 Optional Features

4.2.1 Freezing

Each system may have a JTAG command (or series) to simultaneously freeze all cores in the system. It is recommended to have core-level signals that system-level debugging logic can use to perform this function. However, it is possible to have a system-level controller sequence through each core's debugging interface to freeze, as long as the freezing is reasonably simultaneous.

4.2.2 Self-Discovery

Self-discovery is the ability to determine hardware features by logic. Self-discovery may include, but is not limited to, core types, configuration, and array sizes.

4.2.3 Cross-Triggering

Cross-triggering in respect to debugging is the ability to have a DE-OUT on one core activates a DE-IN on another core. Ideally the system will have channels set up to direct events to the proper cores and the ability to halt or freeze cores based on these events.

Non-Trace System-Level Features (Note: Core has signal to aid in these features)

Mechanism	(R)quired/(O)ptional	Comments
Halting	R	Each system shall have a JTAG command (or series) to simultaneously halt all cores in the system.
Status	R	Each system shall have a JTAG command (or series) to simultaneously read the status of all cores in the system.
Resume	R	Each system shall have a JTAG command (or series) to simultaneously resume all cores in the system.
Freezing	O	
Self-Discovery	O	
Cross Triggering	O	

5 Trace Features

Due to the complexity of trace and the span of markets that use Power Architecture-based products, trace is not a required feature. If trace is implemented it shall provide the required functions listed below.

5.1 Required Features

5.1.1 Program Trace

Program trace is defined as the value of the program counter (PC) as a program executes. Trace provides a way to determine the instructions that were executed and their order. It is the minimum required feature to support the trace facility. Trace output is defined as the data transmitted by the trace facility. Trace output need not be the program trace. It is acceptable, even desirable, for trace output to be a compressed form of the program trace. In all cases, it is assumed that the program trace can be derived from the trace output

and a copy of the machine language program. Thus, it is assumed that the mechanism used to derive the program trace from the trace output has access to the machine language program that was executed.

5.1.1.1 Trace Output Format

The format of trace output is implementation specific. The wide range of expected processor implementations precludes defining a single format for trace data. Some of the features that contribute to the difficulty of extracting trace output from a CPU core are: micro-architecture (including pipeline depth), speculative execution, and out-of-order execution. The only requirement of the trace output format is that it be complete. That is, after acquiring the trace data, a program trace can be created using reconstruction software that has access to the trace output and the original machine language program.

5.1.1.2 Ownership

It is required that the trace output include an indication of the ownership of the currently executing process. The process ID is acceptable for this purpose. It is acceptable to transmit changes in the process ID, as long as the process that is running can be unambiguously determined in the reconstructed program trace.

5.1.1.3 Synchronization Events

To help compress trace data by reducing the trace port bandwidth requirements, prior implementations have sometimes used a synchronization event (SE). A *synchronization event* is a packet of data from which the program trace can be started. Typically, the synchronization event is followed by more data packets in the trace output. The program trace can be reconstructed from the synchronization event and the subsequent trace output. The program trace does not have to be able to be reconstructed from the synchronization event. This specification does not require that synchronization events be used. However, if synchronization events are used, then the following sections apply.

5.1.1.3.1 Frequency of Synchronization Events

The minimum and maximum frequency of synchronization events varies with the implementation and therefore is not specified in this document. It is recommended that a programmable counter be provided which, upon timeout, will cause an SE to be transmitted on the Trace Port. It is recommended as an option to supply an external signal that allows external devices to request a SE from the processor.

5.1.1.3.2 Loss of Synchronization

If the Trace Output is interrupted and the resulting output stream cannot be reconstructed into the program trace, this state is called loss of synchronization (LOS). LOS can result from an internal buffer overflow where data in the trace output is lost. If LOS can happen in a system, then an LOS indication shall be transmitted on the trace output. After an LOS, if a synchronization event is needed to again re-establish the trace output, a synchronization event shall either be automatically transmitted, or else there shall be a means to request a synchronization event. If LOS occurs it is not required to flush the trace output from buffers. As long as the program trace can be unambiguously reconstructed from the trace output, the buffers are not required to be flushed.

5.2 Optional Features

5.2.1 Data Trace

Data trace is optional, and the format is undefined. If data trace is provided, it shall be possible to disable the output.

5.2.1.1 Debug Events

Debug events are defined in the Power ISA™. These events may be augmented by additional events defined by the implementation.

Optionally, a debug event internal to the processor or system can be transmitted on the trace. The debug event packet format is not defined but should be simple enough to be decoded in real-time by the trace probe. In this manner, the trace probe will be able to detect the debug event and perform an action, such as start or end a trace capture operation.

5.2.2 Flow Control

Flow control, an optional feature, is the ability to stall execution within the processor in response to a flow control message. The format of the flow control message is undefined. It can be an input signal or some other method. If both flow control and data trace are provided, it shall be possible to control them separately.

Trace Features

Mechanism	(R)quired/(O)ptional	Comments
Program Trace	R	Value of the program counter
Data Trace	O	
Flow Control	O	

5.3 Trace Port

The trace port is the physical layer which transmits the trace data off the chip. The trace port shall be implemented using the connection method specified in the soon to be released Power.org serial trace specification (www.power.org/resources/downloads).

5.4 Interoperability

Implementation of this specification is vendor specific, as such interoperability is limited at the software level. Refer to the soon to be released Power .org debug API specification (www.power.org/resources/downloads) .