IEEE-ISTO
Industry Standards and Technology Organization
Affiliated with the IEEE and the IEEE Standards Association

The Nexus 5001 Forum™
Standard for a Global Embedded Processor Debug Interface

Version 2.0

23 December 2003

IEEE- Industry Standards and Technology Organization (IEEE-ISTO)
445 Hoes Lane • Piscataway, NJ 08854 USA
Phone +1732.465.6466 • Fax +1.732.981.9473 • http://www.ieee-isto.org

The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface

Recognized by the IEEE Vehicular Technology Society

History: The Global Embedded Processor Debug Interface Standard (GEPDIS) Consortium was formed in April 1998 to define and develop a much-needed embedded processor debug interface standard for embedded control applications. On 23 September 1999, the GEPDIS Consortium chose the IEEE Industry Standards and Technology Organization (IEEE-ISTO) as the operational and legal forum in which to continue its efforts. During the transition, the group also changed its name to the Nexus 5001 Forum™ to reflect the submission of Version 1.0 of their standard to the IEEE-ISTO for publication, distribution and future management as IEEE-ISTO 5001™ - 1999, The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface. On 15 December 2003, Version 2.0 of the IEEE-ISTO 5001™ - 2003 was approved by membership, and published.

Abstract: A general-purpose specification that addresses the rigorous challenges for debug interfaces is outlined. Auxiliary pin functions, transfer protocols and standard development features are defined.

Keywords: Application Programming Interface (API), auxiliary port, Boolean, breakpoint, bit, client, compliance classification, debug interface, embedded processor, emulator, full-duplex, half-duplex, Hardware Abstraction Layer (HAL), high-speed input/output (HSIO), low-speed input/output (LSIO), Nexus, pin, register, Target Abstraction Layer (TAL), watchpoint.

Title: The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface, Version 2.0

The IEEE-ISTO and the Nexus 5001 Forum™ DISCLAIM ANY AND ALL WARRANTIES, WHETHER EXPRESSED OR IMPLIED, INCLUDING (WITHOUT LIMITATION) ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

The Nexus 5001 Forum™, a program of the IEEE-ISTO, reserves the right to make changes to the document without further notice. The document may be updated, replaced or made obsolete by other documents at any time.

The IEEE-ISTO takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document, or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights.

The IEEE-ISTO invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. The IEEE-ISTO and its programs shall not be responsible for identifying patents for which a license may be required by an IEEE-ISTO Industry Group Standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. Inquiries may be submitted to the IEEE-ISTO by e-mail at: info@ieee-isto.org.

The Nexus 5001 Forum™ acknowledges that the IEEE-ISTO (acting itself or through its designees) is, and shall at all times, be the sole entity that may authorize the use of certification marks, trademarks or other special designations to indicate compliance with these materials.

Use of this IEEE-ISTO Industry Group Standard is wholly voluntary. The existence of an IEEE-ISTO Industry Group Standard does not imply that there are no other ways to produce, test, measure, purchase, market or provide other goods and services related to its scope.

About the IEEE-ISTO

The IEEE-ISTO is a not-for-profit corporation offering industry groups an innovative and flexible operational forum and support services. The IEEE-ISTO provides a forum not only to develop standards, but also to facilitate activities that support the implementation and acceptance of standards in the marketplace. The organization is affiliated with the IEEE (http://www.ieee.org/) and the IEEE Standards Association (http://standards.ieee.org/).

For additional information regarding the IEEE-ISTO and its industry programs visit http://www.ieee-isto.org.

About the Nexus 5001 Forum™

The Nexus 5001 Forum™ (formerly known as the Global Embedded Processor Debug Interface Consortium) is chartered to advance the development, dissemination and implementation of the Global Embedded Processor Debug Interface Standard. The Nexus 5001 Forum™ is open to all interested parties.

For additional information (membership, procedures, articles, news releases, etc.) regarding the Nexus 5001 Forum™, visit http://www.nexus5001.org.

Feedback

Comments and questions may be submitted to the Nexus 5001 Forum™ through the IEEE-ISTO:

Nexus 5001™ Forum Program Manager
C/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854 USA
Telephone: +1.732.465.6466
Fax: +1.732.562.1571
Email: nexus-admin@nexus5001.org

Preface

The Nexus 5001 Forum™ (formerly known as the Global Embedded Processor Debug Interface Consortium) is chartered to advance the development, dissemination and implementation of IEEE-ISTO 5001™-1999, the Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface.

The industry group was formed in April 1998 to define and develop a much-needed embedded processor debug interface standard for embedded control applications. As advances in semiconductor and system design continue, embedded applications are using higher performance embedded processors. Efficient use of these embedded processors requires software and hardware development tools that can easily access critical processor functionality.

However, the lack of a unifying standard among the various embedded processors on the market at the time of publication has impeded this accessibility, preventing tool vendors from creating standard tools with consistent functionality across a broad range of processors. This, in turn, has become a gating factor for chipmakers, tool providers and developers. Ultimately customers are forced to pursue costly custom solutions to meet their tool needs.

IEEE-ISTO 5001-1999, the Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface is an open industry standard that provides a general-purpose interface for the software development and debug of embedded processors. Standardization on this interface benefits customers' reuse of their Nexus 5001™ compliant development tools on compliant processor architectures. A future method and a process by which Nexus 5001 compliance can be validated will further global implementation of the standard.

Although the initial focus of the effort was based on the stringent requirements of the automotive powertrain applications, a general purpose standard has been developed, aimed to also benefit data communications and computer peripherals, wireless systems and other embedded control applications industries.

The Nexus 5001 Forum is open to all interested companies. Members of the Forum represent all aspects of the technologies required for embedded control applications: embedded processor suppliers, independent tools providers, semiconductor and hardware development tools, and software tools (emulators, compilers, simulators, debuggers, RTOS's, etc.).

Additional information regarding the Nexus 5001 Forum can be found its website at http://www.nexus5001.org/.

SECTION 1
Introduction

SECTION 2
Compliance and Performance Classifications

SECTION 3
Nexus Development Interface

SECTION 4
Nexus Development Features

SECTION 5
Nexus Public Messages

SECTION 1
Introduction

1.1 Overview

The Nexus 5001 Forum™ (**http://www.ieee-isto.org/Nexus5001/**), previously referred to as the Global Embedded Processor Debug Interface Standard Consortium (GEPDISC), was formed to develop a much needed embedded processor debug interface standard for embedded control applications. The internal name for this standard is “Nexus,” which is used throughout this document only.

The goal is a general-purpose specification that addresses the rigorous challenges for debug interfaces. Applications that may benefit from this standard interface include automotive powertrain, data communications, computer peripherals, wireless systems, and other control applications.

As advances in semiconductor and system design continue, embedded applications are using higher performance embedded processors. Efficient use of these embedded processors requires software and hardware development tools that can easily access critical processor functionality. The lack of a unifying standard among the various embedded processors on the market has impeded this accessibility and prevented tool vendors from creating standard tools with consistent functionality across a broad range of processors. Ultimately, system developers are forced to pursue costly custom solutions to meet their tool needs.

To provide the best opportunity for achieving a worldwide development interface standard, it is prudent to leverage off an accepted pin interface and hardware transfer protocol that exist today— the **IEEE 1149.1** standard.[1] The Nexus standard defines an extensible Auxiliary Port (AUX) that may either be used with the **IEEE 1149.1** port or as a stand-alone development port. The Nexus standard defines the auxiliary pin functions, transfer protocols, and standard development features.

[1]For information on references, see **Appendix D - Bibliography**.

1.2 Basic Development Needs for Embedded Processors

Developers of embedded processors need to have access to a basic set of development tool functions in order to accomplish their jobs. For run control the basic needs are

- To query and modify when the processor is halted, showing all locations available in the processor's supervisor map.
- To support breakpoint/watchpoint features in debuggers, either as hardware or software breakpoints depending on the architecture. Configuration of breakpoint/watchpoint features may be performed when the processor is halted.

For logic analysis the basic needs are

- To access instruction trace information with acceptable impact to the system under development. The developer needs to be able to interrogate and correlate instruction flow to real-world interactions.
- To retrieve information on how data flow through the system with acceptable impact to the system under development and to understand what system resource(s) are creating and accessing data.
- To assess whether embedded software is meeting the required performance level with acceptable impact to the system under development.

1.3 Useful Development Features for Embedded Processors

The evolution of high-performance microprocessor units (MPUs) and highly integrated microcomputer units (MCUs) has had an impact on development processes and tools. High-performance on-chip caches, flash and random access memory (RAM), and other changes have eliminated the internal visibility needed for instruction and data trace. Thus, there are specific features the Nexus standard should address, as listed below:

1. Program trace visibility is needed for development tools with acceptable impact to the system under development. With high-performance on-chip instruction cache and flash, visibility needed for program trace is restricted. In some applications the external bus is used for a secondary function, such as general-purpose input/output (GPIO), or is not available.
2. Data trace visibility is needed for development tools with acceptable impact to the system under development. With on-chip high-performance data cache and RAM, the visibility needed for data trace is restricted. Two types of data visibility are needed:

   a. Which process (i.e., which instruction address) wrote which data parameter, and what new value was written?

   b. For a chosen data parameter, which process(es) accessed it?

3. A standard development methodology and tool set is needed for embedded applications. Because vendors of embedded processors generally do not support the same development interface/methodology, development methodology and tools are not compatible.

4. A development pin interface standard is needed to support development with multiple clients (processor cores or intelligent peripherals) on the embedded processor. The development pin interface comprises basic visibility and controllability of each processor independently.

5. An independent processor development pin interface standard is needed to support development for all mainstream processor architectures.

6. An embedded development pin interface standard is needed to allow for connection to multiple development tools. Tool arbitration may be needed if multiple development tool boxes are connected to the same target. Arbitration among tools is not addressed in this standard.

7. Multiplexing of development pin functions should be performed in a manner so that undue constraints are not placed on the developer of the embedded system. MCU vendors occasionally multiplex on the same pins development functions and GPIO. Guidelines should be given to eliminate improper multiplexing, especially out of reset, which can lead to unpredictable behaviors and anomalies in development tools. Development pins should be configurable to be in development mode out of reset.

8. A scalable development pin interface standard, which will work for different price targets of embedded MCUs/MPUs, is needed.

9. An embedded development pin interface standard is needed for cost-effective tools.

1.4 Terms and Definitions

Table 1-1 lists terms and definitions used in the Nexus standard.

Table 1-1—Terms and Definitions

Term	Definitions
Address	The term is used to indicate logical address. If there is no address translation in an application, then it also refers to the physical address.
Application Programming Interface (API)	API abstracts the semantics of **Appendix B - Recommendations for Access to Control and Status Registers** so that a tool can perform a common set of operations on any target, irrespective of hardware registers implemented on the target.
Auxiliary Port (AUX)	Refers to the Nexus auxiliary port. Used as auxiliary port to the **IEEE 1149.1** interface or as a stand-alone development port. The AUX consists of separate input and output ports called Auxilliary Input Port (AUX IN) and Auxiliary Output Port (AUX OUT). The ports are optional depending on the implemented class.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions. Also, the number of sequential instruction units executed between each taken branch.
Data Breakpoint	Processor is halted at an appropriate instruction boundary after a trigger is set at a data valid time. The trigger is set when the data address and/or value matches a pre-selected address and/or value.
Calibration Constants	Performance-related constants that must be tuned for automotive powertrain and disk drive applications.
Calibration Variables	Intermediate calculations that must be visible during the calibration or tuning process to enable accurate tuning of calibration constants.
Client	A functional block on an embedded processor that will require development visibility and controllability. Examples are a central processing unit (CPU) and an intelligent peripheral.
Data Acquisition Messaging (DQM)	Visibility of related data parameters stored in internal resources, e.g., related calibration variables for automotive applications.
Data Read Messaging (DRM)	Visibility of data reads to internal memory-mapped resources, e.g., on-chip RAM.
Data Write Messaging (DWM)	Visibility of data writes to internal memory-mapped resources, e.g., on-chip RAM.
Data Trace Messaging (DTM)	Visibility of how data flow through the embedded system. May include DRM and DWM. Refer to **1.3 - Useful Development Features for Embedded Processors** for more information on data trace requirements.
Full-duplex	Messages can be transmitted in both directions between tool and target simultaneously.
Global Embedded Processor Debug Interface Standard Consortium (GEPDISC)	GEPDISC, renamed the Nexus 5001 Forum™ (http://www.ieee-isto.org/Nexus5001/), was formed to develop a much needed embedded processor debug interface standard for embedded control applications. The internal name for this standard is “Nexus,” which is used throughout this document only.
Half-duplex	Messages can be transmitted in only one direction at a time between tool and target.

Table 1-1—Terms and Definitions *(Continued)*

Term	Definitions
Hardware Breakpoint	Typically a hardware comparator used to halt the processor at an appropriate instruction boundary after an address or data value matches a pre-selected address or data value.
High-Speed Input/Output (HSIO)	The term *HSIO,* as used in the Nexus standard, is intended to refer to an external bus of the embedded processor. Assertion and negation timing are critical to system integrity.
Instruction Breakpoint	Processor is halted when all previous instructions are retired and just prior to when any architectural state is changed by the instruction associated with a pre-selected address.
Instruction Units	Most Program Trace Messages have a packet that indicates the number of instruction units executed since the last taken branch. In target architectures in which all instructions are the same size, then this packet contains the actual number of instructions executed since last taken branch. If instructions are of variable size, then the number reported is the number of instruction units. The instruction unit represents the number of bytes or words associated with the highest common denominator of the variable instruction sizes.
IEEE 1149.1 Instruction Register (IR) and Data Register (DR) Sequence	**IEEE 1149.1** IR scan loads an opcode value for selecting a development register. The selected development register is then accessed via an **IEEE 1149.1** DR scan.
Low-Speed Input/Output (LSIO)	LSIO pin functions are typically implemented on MCUs, e.g., an output pin to set system configuration. Assertion and negation timing are not critical to system integrity.
Nexus	Internal code name for this standard.
Nexus API	API required by the Nexus standard.
Ownership Trace Messaging (OTM)	Visibility of the process/function that is currently executing.
Public Messages	Public Messages are defined for the AUX and the **IEEE 1149.1** interface. These messages must be used for designated functions when these functions are implemented. Public Messages are specified pin protocols for accomplishing common configuration, status, and visibility (e.g., DRM and DWM).
Read-Only Memory (ROM)	Read-only memory, such as nonvolatile flash.
Standard	The phrase “according to the Nexus standard” means “according to the Nexus standard contained in this document.”
Target	Generally refers to an end application or evaluation board, containing one or more embedded processors, to which a development tool is connected.
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
Memory Substitution Messaging (MSM)	Messaging for a memory substitution access in which internal accesses are redirected through the auxiliary pins defined in the Nexus standard.
Nexus-Recommended Register (NRR)	NRRs are defined in **Appendix B - Recommendations for Access to Control and Status Registers**.

Table 1-1—Terms and Definitions *(Continued)*

Term	Definitions
Vendor-Defined Message	Vendor-Defined Messages are allowed via the AUX and the **IEEE 1149.1** interface for development features that may be specific to each vendor. These messages must follow the protocol defined for the AUX.
Watchpoint	A data or instruction breakpoint that does not cause the processor to halt. Instead a pin is used to signal that the condition occurred.

1.5 Conventions

This document uses the following notational conventions:

ACTIVE_HIGH — Names for signals that are active high are shown in uppercase text without an overbar. Signals that are active high are referred to as asserted when they are high and negated when they are low.

$\overline{\text{ACTIVE\_LOW}}$ — A bar over a signal name indicates that the signal is active low. Signals that are active low are referred to as asserted (active) when they are low and negated when they are high.

0x0F — Hexadecimal numbers

0b0011 — Binary numbers

LSB — Means least significant bit. The LSB is the lowest bit number, e.g., bit 0

MSB — Means most significant bit. The MSB is the highest bit number, e.g., bit 31

Set bit — To set a bit (or bits) means to establish logic level one on the bit (or bits), i.e., the voltage that corresponds to Boolean true (1) state.

Clear bit — To clear a bit (or bits) means to establish logic level zero on the bit (or bits), i.e., the voltage that corresponds to Boolean false (0) state.

1.6 Other Terminology within the Nexus Standard

The term *vendor-defined bit fields* is used to indicate bit fields that may be defined as needed for the vendor's device.

The terms and definitions specifically associated with the Nexus-defined Public Messages can be found at the top of **Section 5 - Nexus Public Messages**.

SECTION 2
Compliance and Performance Classifications

The capability of Nexus-compliant development ports shall comprise two basic designations: the development features supported by the port and the performance capability for downloading and uploading via the port. All development features described in the Nexus standard are assigned to at least one compliance classification: Class 1, Class 2, Class 3, or Class 4.

Performance capability is designated by full- or half-duplex capability and by transfer bandwidth in megabits per second for both downloads to the embedded processor and uploads from the embedded processor. Thus Nexus-compliant development ports implemented by vendors of embedded processor integrated circuits (ICs) shall be designated as follows:

- Class 1, 2, 3, or 4 compliant
- Download rate (megabits per second)
- Upload rate (megabits per second)
- Full- or half-duplex

2.1 Compliance Classification

Complying embedded processors shall be designated as Class 1, 2, 3, or 4 or as an approved compliance sub-class. Class 1 devices implement the fewest Nexus development features. Class 4 devices implement the most Nexus development features. Thus Class 4 devices offer the most development capability and standardization. Class 1, 2, and 3 devices offer a graduated subset of the Nexus development features, which may be appropriately suited for some applications.

Table 2-1 and **Table 2-2** show the minimum features for the four compliance classifications.

Table 2-1—Compliance Classification for Static Development Features

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
Read/write user registers in debug mode	V	V	V	V	Refer to Nexus API
Read/write user memory in debug mode	A	A	A	A	Read/Write Access
Enter a debug mode from reset	A	A	A	A	Development Control and Status
Enter a debug mode from user mode	A	A	A	A	
Exit a debug mode to user mode	A	A	A	A	
Single-step instruction in user mode and re-enter debug mode	A	A	A	A	
Stop program execution on instruction/data breakpoint and enter debug mode (minimum 2 breakpoints)	A	A	A	A	Breakpoints/ Watchpoints
Note: "A" indicates a required development feature that must be implemented via the Nexus API. "V" indicates a required vendor-defined development feature implemented in the Nexus API.					

Table 2-2—Compliance Classification for Dynamic Development Features

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
Ability to set breakpoint or watchpoint	A	A	A	A	Breakpoints/ Watchpoints
Device identification	A	A and P	A and P	A and P	Device ID Message (see **Section 5 - Nexus Public Messages**)
Ability to send out an event occurrence when watchpoint matches	P[a]	P	P	P	Watchpoint Match Message (see **Section 5 - Nexus Public Messages**)
Monitor process ownership while processor runs in real time	—	P	P	P	Ownership Trace
Monitor program flow while processor runs in real time (logical address)	—	P	P	P	Program Trace
Monitor data writes while processor runs in real time	—	—	P	P	Data Trace (Writes only)
Read/write memory locations while program runs in real time	—	—	A and P	A and P	Read/Write Access
Program execution (instruction/data) from Nexus port for reset or exceptions	—	—	—	P	Memory Substitution
Ability to start ownership, program, or data trace upon watchpoint occurrence	—	—	—	A	Development Control and Status
Ability to start memory substitution upon watchpoint occurrence or upon program access of vendor-defined address	—	—	—	O	Development Control and Status

Table 2-2—Compliance Classification for Dynamic Development Features *(Continued)*

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
Monitor data reads while processor runs in real time	—	—	O	O	Data Trace (Reads and Writes)
LSIO port replacement and HSIO port sharing	—	O	O	O	Port Replacement/ Sharing
Transmit data values for acquisition by tool	—	—	O	O	Data Acquisition
Note: "A" indicates a required development feature that must be implemented via the Nexus API. "P" indicates a required development feature that must be implemented via the Nexus development port as a Public Message or with a Nexus port pin (as appropriate). "O" indicates an optional development feature as defined by the Nexus standard.					

a. Because no AUX is required for Class 1, the event occurrence should be provided via an Event Out ($\overline{\text{EVTO}}$) pin defined in **Section 6 - Nexus Port Signals**, or via a Message Out mechanism defined in **Section 8 - IEEE 1149.1 Message Protocol**.

2.1.1 Compliance Sub-Class for Application-Specific Development Needs

To comply with application-specific development needs, compliance sub-classes for specific applications shall be approved by a standards developing organization. Sub-classes are allowed when standardized support of application-specific development features are needed.

2.2 Performance Classification

Complying embedded processors shall be designated by a performance classification. The embedded processors shall be designated by full- or half-duplex capability and by transfer bandwidth in megabits per second for both downloads to the embedded processor and uploads from the embedded processor.

Full- and half-duplex capability is related to compliance classification as described in **Table 2-3**. Refer to **Appendix A - Connector and Electrical Specifications**, which contains the connector options defined at the time this standard was released. Other connector options are expected as this standard evolves.

Table 2-3—Performance Interface Options

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
IEEE 1149.1 port only	X	—	—	—	Half-duplex
IEEE 1149.1 or AUX IN with an AUX OUT	—	X	X	X	Full-duplex

2.2.1 Interpreting Performance Classification

System developers of embedded processors should interpret the performance classification properly in order to assess the capability needed for their application. To do so, a basic knowledge is needed of Nexus features, the developer's code characteristics and visibility needs.

The transfer bandwidth for downloads can be thought of as the sustainable input bandwidth required to the device. Conversely, the transfer bandwidth for uploads can be thought of as the sustainable output bandwidth required from the device. Bandwidth requirements are typically determined by what Nexus development features are needed during runtime. Bandwidth requirements are satisfied by AUX size and clock rate.

The Nexus AUX is used to fulfill the output bandwidth requirements. In calculating the average output bandwidth requirements for an application, factors that may be considered are:

- Frequency of taken direct and indirect changes of flow
- Frequency and size of internal data reads/writes that must be visible
- Frequency and size of data that must be read from device

The Nexus AUX or the **IEEE 1149.1** port is used to fulfill the input bandwidth requirements. In calculating the average input bandwidth requirements for an application, factors that may be considered are the frequency and size of data that must be written to the device.

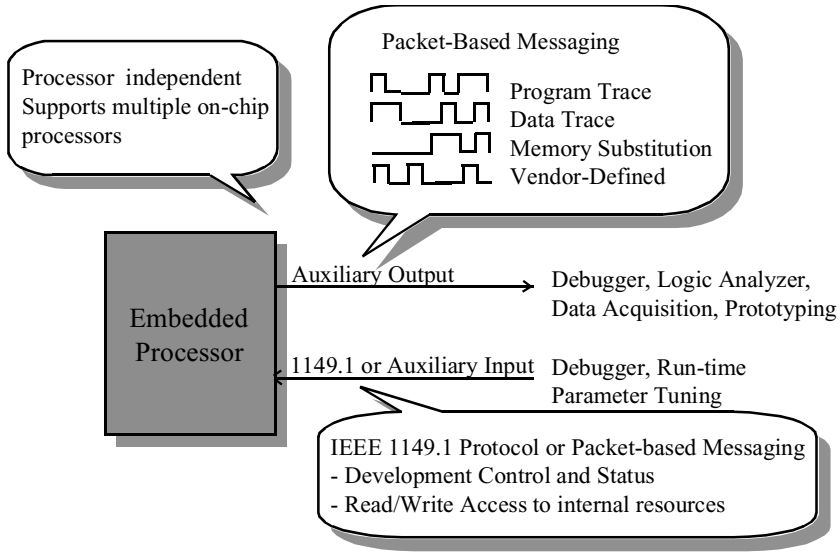
SECTION 3
Nexus Development Interface

The development interface shall be implemented by Nexus Class 1, 2, 3, and 4 embedded processors as described in **Table 2-3**. The development features shall be implemented by Nexus Class 1, 2, 3, and 4 embedded processors as described in **Table 2-1** and **Table 2-2**.

3.1 Overview

Embedded processors complying with Class 1 shall implement the **IEEE 1149.1** standard for access to the minimum development features of compliance Class 1. Embedded processors complying with Class 2, 3, or 4 shall implement a Nexus pin interface according to the Nexus standard, for external visibility required for the minimum development features of compliance Class 2, 3, or 4, respectively. Additionally, compliant embedded processors shall implement either an **IEEE 1149.1** standard or Nexus pin interface (or both) according to the Nexus standard for access to the minimum development features of compliance Class 2, 3, or 4.

Figure 3-1 illustrates Nexus development interface options for a Class 2, 3, or 4 embedded processor.



Note: The AUX IN is input only. Although the **IEEE 1149.1** interface is bidirectional, for simplicity it is illustrated as input only.

Figure 3-1—Illustration of IEEE 1149.1/Nexus Development Interface

For implementation of the **IEEE 1149.1** interface options for Class 2, 3, and 4 embedded processors, as referenced in **Figure 3-1**, in addition to the **IEEE 1149.1**-defined pins[2], only four auxiliary pins are required for compliance. The performance classification, however, would also be minimal and may only meet the transfer bandwidth requirements for low-end applications or for lower compliance classifications. If faster downloads to the embedded processor are required than is possible via the **IEEE 1149.1** interface, an AUX IN should be implemented.

The Nexus standard allows for greater performance capability in either or both of the following ways: with a scalable auxiliary pin interface to transfer more bits on each clock and/or a faster transfer clock to transfer more bits per unit time. **Table 3-1** shows recommendations (not requirements) for AUX type.

Table 3-1—Recommendations for AUX Type

Compliance Class and Port type	Number of Device Data Pins				
	1	2	4	8	16
Class 2 input port	X	X	—	—	—
Class 2 output port	X	X	—	—	—
Class 3 or 4 input port	X	X	X	—	—
Class 3 or 4 output port	—	—	X	X	X

3.2 IEEE 1149.1 Pin Interface

The **IEEE 1149.1** standard defines the required protocol for access to the minimum development features of compliance Class 1. Additionally, the **IEEE 1149.1** standard defines the required protocol for access to the minimum development features of compliance Classes 2, 3, and 4, if the Nexus input interface option is not selected by the developer of the embedded processor IC.

The **IEEE 1149.1** interface shall provide the following capability:

- **IEEE 1149.1** sequences for access to processor identification, development control and status information according to the Nexus standard [e.g., configuring a breakpoint via Application Programming Interface (API)]
- **IEEE 1149.1** sequences for access to user memory-mapped registers during halt or runtime according to the Nexus standard
- **IEEE 1149.1** sequences for access to development messages according to the Nexus standard (e.g., ownership trace)
- **IEEE 1149.1** sequences for access to all vendor-defined development features (e.g., user registers when processor is halted)

[2] **IEEE 1149.1** pins include Test Clock (TCK), Test Mode Select (TMS), Test Data Input (TDI), Test Data Output (TDO), and Test Reset ($\overline{\text{TRST}}$). ($\overline{\text{TRST}}$ is optional.)

3.3 Nexus Auxiliary Pin Interface

The Nexus pin interface shall be implemented according to the Nexus standard for external visibility required for the minimum development features of compliance Classes 2, 3, and 4. Additionally, the Nexus pin interface shall be implemented according to the Nexus standard for access to the minimum development features of compliance Classes 2, 3, and 4, if the **IEEE 1149.1** interface option is not selected by the embedded processor IC developer.

The auxiliary pin interface shall provide the following external visibility according to the Nexus standard:[3]

- Trace of operating system software execution via Ownership Trace Messaging (OTM)
- Program trace via Branch Trace Messaging (BTM)
- Data trace via Data Trace Messaging (DTM)
- Signal watchpoint and breakpoint events
- Runtime system memory substitution via Memory Substitution Messaging (MSM)
- Other high-bandwidth information transfer (vendor-defined)

Additionally, the auxiliary pin interface shall provide the following access according to the Nexus standard, if the **IEEE 1149.1** option is not selected by the embedded processor IC developer:

- Access to processor identification, development control, and status information
- Access to user memory-mapped registers when halted or during runtime
- Access to all vendor-defined development features (e.g., user registers when processor is halted)
- Provide optional access according to standard support for compliant development tools to implement port replacement of development port
- Provide optional access according to standard ability for embedded processor to transmit data values for acquisition by development tool

[3]Refer to **Section 1.4 - Terms and Definitions** for definitions of all new terms in the list.

SECTION 4
Nexus Development Features

The development features are described in **4.1 - Application Programming Interface (API)** through **4.10 - Data Acquisition (Optional)**.

4.1 Application Programming Interface (API)

Embedded processors complying with all classes shall provide an API according to the Nexus standard. The Nexus API is defined in a separate reference document maintained by **IEEE-ISTO 5001**.

4.1.1 Overview

The Nexus API allows tool vendors to use a common “low-level semantic” API to abstract the low-level implementation details of each Nexus-compliant embedded processor. Because the Nexus standard does not mandate that the Nexus-Recommended Registers (NRRs) defined in **Appendix B - Recommendations for Access to Control and Status Registers** are implemented, vendors of embedded processors are free to implement a different set of development registers. The required feature set for each class, however, must be available (refer to **Table 2-1** and **Table 2-2**). The API allows embedded processors not implementing the NRRs to be accessed in a standard manner.

The Nexus API is suitable for use by tools (debuggers, etc.), and also for the Nexus validation suite. It is designed to capture the low-level semantics of the Nexus features, so that it can be used to implement the bottom layers of a tool vendor’s own target debug API.

Tool vendors support a large number of different host platforms, operating systems, and compilation systems. Emulator vendors similarly support a multitude of systems. The Nexus API is suitable for use in a wide variety of systems. It has been designed to not rely on any platform-specific, real-time operating-system-specific (RTOS-specific) or compiler-specific features.

The Nexus API abstracts the semantics of the NRRs, so that tools can perform a common set of operations on any target, irrespective of its class or its underlying register set. The Nexus API is divided into two sections, which are described in a separate document maintained by **IEEE-ISTO 5001**:

- Emulator Hardware Abstraction Layer (HAL)
- Target Abstraction Layer (TAL)

4.2 Development Control and Status

Embedded processors complying with Class 1, 2, 3, or 4 shall provide the development control and status required by the Nexus API.

4.2.1 Overview

Standardized development control and status, and standardized access to the Nexus API, offer a significant degree of commonality. The Nexus API can be leveraged by development tool vendors for creating standard tools with consistent functionality across a broad range of processors. Ultimately, system developers will benefit with more effective tools to meet their tool needs.

4.2.2 NRRs

For development control and status, embedded processors may implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers** or a set of vendor-defined development control and status registers that implement the requirements specified by the Nexus API.

NRRs include the following control and status registers:

- Device Identity (DID) Register
- Client Select Control (CSC) Register
- Development Control (DC) Register
- Development Status (DS) Register
- User Base Address (UBA) Register
- Read/Write Access (RWA/RWD/RWCS) Registers
- Watchpoint Trigger (WT) Register
- Data Trace Attribute (DTSA/DTEA/DTC) Registers (minimum of two)
- Breakpoint/Watchpoint Control (BWC) Registers (minimum of two)[4]

[4]Optionally, the two BWC Registers may be combined with the two Data Trace Attribute Registers so that a total of two registers may be simultaneously active, i.e., two BWC Registers, two Data Trace Attribute Registers, or one BWC Register and one Data Trace Attribute Register.

Development control and status registers (Nexus recommended or vendor-defined) shall be accessed via the **IEEE 1149.1** interface or the auxiliary pin interface according to the Nexus standard. A sequence for each interface is recommended in the Nexus standard.

4.3 Ownership Trace

Embedded processors complying with Class 2, 3, or 4 shall provide ownership trace visibility according to the Nexus standard.

4.3.1 Overview

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high-level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. Ownership trace is especially useful when the developer is not interested in debugging at lower levels.

Ownership trace is especially important for embedded processors with a memory management unit, in which all processes can use the same logical program and data spaces. Ownership trace offers development tools a mechanism to decipher which set of symbolics and sources are associated for lower levels of visibility and debugging.

4.3.2 Ownership Trace Messaging (OTM)

Ownership trace information is transmitted out the AUX using OTM. OTM facilitates ownership trace by providing visibility of which process identity (ID) or operating system task is activated. An Ownership Trace Message is transmitted to indicate when a new process/task is activated, allowing development tools to trace ownership flow. Additionally, for embedded processors that implement virtual addressing or address translation, an Ownership Trace Message is also transmitted periodically during runtime at a minimum frequency of every 256 Program/Data Trace Messages.

The Nexus standard defines an OTM Register whose user memory map location is accessed via the **IEEE 1149.1** or auxiliary pin interfaces. The OTM Register is to be updated as determined by the operating system software to provide task/ process ID information. When new information is updated in the register by the embedded processor, the information is transmitted out via the AUX. Refer to **B.5.5 - User Base Address (UBA) Register** for more information.

4.4 Program Trace

Embedded processors complying with Class 2, 3, or 4 shall provide

1. Program trace visibility via the AUX according to the Nexus standard
2. Capability to detect and signal program trace overrun errors according to the Nexus standard, if the condition occurs during application of the embedded processor
   - A program trace error shall be detected and transmitted out the AUX according to the Nexus standard when any Program Trace Message is lost and not signaled via the AUX.
   - Embedded processors complying with Class 4 shall provide the capability to delay the processor and avoid overruns.
3. Capability to synchronize program trace according to the Nexus standard

4.4.1 Overview

The Program Trace feature defines a standard protocol for program trace visibility that is processor independent. Additionally, the number of program trace signals that must be visible external to the device is significantly reduced over conventional methods. The benefit is standard logic analysis tools with consistent functionality.

4.4.2 Branch Trace Messaging (BTM)

The Program Trace feature implements a Program Flow Change Model in which program trace is synchronized at each program flow discontinuity. A program flow discontinuity occurs at taken branches and exceptions. The messages generated using this model are referred to as Branch Trace Messages.

Development tools can interpolate what transpires between program flow discontinuities by correlating information from Branch Trace Messages and static source or object code files. Self-modifying code cannot be traced with the Program Flow Change Model because the source code is not static.

There are two types of Branch Trace Messages: traditional and history. History messages are generally used for increased bandwidth requirements of higher performing processors or by processors that incorporate predicated instructions. For Class 2 (and higher) embedded processors, it is optional which type of program trace messages are used.

4.4.2.1 BTM Using Traditional Messages

Traditional BTM facilitates program trace by providing several key types of visibility. The visibility comprises the following:

- Messaging for taken direct branches includes the number of sequential instruction units that were executed since the last taken branch or exception and an indication of which client (if more than one are present on the embedded processor) took the branch. Direct (or indirect) branches that are not taken are included in the count of sequential instruction units. Direct branches that are taken are not included in the count of sequential instruction units.

- Messaging for taken indirect branches and exceptions includes the number of sequential instruction units that were executed since the last taken branch or exception, the unique portion of the branch target address or exception vector address, and an indication of which client (if more than one are present on the embedded processor) took the branch. Indirect branches that are not taken are included in the count of sequential instruction units. Indirect branches that are taken are not included in the count of sequential instruction units.

4.4.2.2 BTM Using Branch History Messages

BTM using Branch History Messages also facilitates program trace by providing several key types of visibility. The visibility comprises the following:

- Messaging for taken indirect branches and exceptions includes the number of sequential instruction units that were executed since the last taken branch, exception, or predicate instruction; the unique portion of the branch target address or exception vector address; an indication of which client (if more than one are present on the embedded processor) took the branch; and a branch/predicate instruction history field. Indirect branches that are not taken are included in the count of sequential instruction units. Indirect branches that are taken are not included in the count of sequential instruction units.

- There is no messaging for direct branch events or predicated instructions using history messages. Each bit is logged in a history buffer where a value of 1 indicates taken and a value of 0 indicates not taken. The history buffer is transmitted in the history field of Indirect Branch Messages. Refer to **5.3.7 - Program Trace - Indirect Branch with Sync Message** for detail on Branch History Messages.

For both types of messages, the information regarding the number of sequential instruction units executed since the last taken branch is used to facilitate the following:

1. Trace which direct branch is taken
2. Detect which instruction may have caused an exception

The unique portion of the indirect branch target address transmitted out the AUX is relative to a prior address transmitted out the AUX.

BTM can also be triggered during runtime at the occurrence of watchpoint.

4.4.3 Program Trace Overrun Errors

The overrun Error Message is to be used by development tools to notify the developer that program trace information has been lost. A BTM overrun error occurs when the number of messages to be transmitted via the AUX in a given time period exceeds the bandwidth capacity of the AUX.

4.4.4 Program Trace Synchronization

Due to the nature of some processor architectures, such as reduced instruction set computer (RISC) processors, some application programs may comprise a significant number of direct branch instructions and very few indirect branch instructions.

Because BTM for taken direct branches does not provide the target address, program trace for these application programs must be accomplished in a relative manner (possibly without branch target address information). Synchronization messages ensure that development tools fully synchronize with the program flow regularly.

A Program Trace Message for synchronization shall be transmitted via the AUX by the embedded processor for the following conditions:

- Initial Program Trace Message upon exit of system reset, exit of a power-down state, or exit of a debug mode
- Periodically during runtime at a minimum frequency of every 256 Program Trace Messages
- When program trace is enabled during normal execution of the embedded processor
- Upon assertion of an Event-In ($\overline{\text{EVTI}}$) pin

- Program trace overrun error
- Upon overflow of the sequential instruction unit counter
- Optionally upon occurrence of a watchpoint

NOTE

A synchronization message, displayed as an individual message or as part of another message, always includes an indication of which client (if more than one are present on the embedded processor) is being synchronized and the full address of a recently executed instruction.

4.5 Data Trace

Embedded processors complying with Class 3 or 4 shall provide

1. Data trace for *write* visibility via the AUX according to the Nexus standard
   - Embedded processors complying with Class 3 or 4 may optionally provide data trace for *read* visibility via the AUX according to the Nexus standard.
2. Capability to detect and signal data trace overrun errors according to the Nexus standard, if the condition occurs during application of the embedded processor
   - A data trace error shall be detected and transmitted out the AUX according to the Nexus standard when any Data Trace Message is lost and not signaled via the AUX.
   - Embedded processors complying with Class 4 shall provide the capability to delay the processor and avoid overruns.
3. Capability to synchronize data trace according to the Nexus standard

4.5.1 Overview

The Data Trace feature defines a standard protocol for data trace visibility of accesses to vendor-defined internal peripheral and memory locations. Practical limitations exist that constrain the number of locations that may be traced via the AUX. In application use, limiting the number of traced locations is necessary for effective use of data trace. Additionally, excluding processor stack area from data trace is beneficial.

4.5.2 Data Trace Messaging (DTM)

The Data Trace feature provides a minimum of two data trace windows that include the following qualifiers:

- Start and end user address for data trace
- Trace reads, writes, or both within the start/end address range

Data accesses are monitored, and qualifying data accesses are then transmitted out the AUX using Data Trace Messages. DTM facilitates data trace by providing several key types of visibility. The messaging for data trace includes the unique portion of the data address and the data value. The unique portion of the data address transmitted out the AUX is relative to the prior data trace address transmitted out the AUX.

4.5.3 Data Trace Overrun Errors

The overrun error message is to be used by development tools to notify the developer that data trace information has been lost. A DTM overrun error occurs when the number of messages to be transmitted via the AUX in a given time period exceeds the bandwidth capacity of the AUX.

4.5.4 Data Trace Synchronization

The output bandwidth requirements for the AUX are minimized for data trace by messaging out only the unique portion of the data address (instead of the complete address). Consequently a data trace address is reconstructed relative to each prior message.

Synchronization messages provide the full address and ensure that development tools fully synchronize with the data trace regularly. Synchronization messages provide a reference address for subsequent Data Trace Messages, in which only the unique portion of the data trace address is transmitted.

A Data Trace Message with Synchronization shall be transmitted out the AUX by the embedded processor for the following conditions:

- Initial Data Trace Message upon exit of system reset, exit of a power-down state, or exit of a debug mode
- Periodically during runtime at a minimum frequency of every 256 Data Trace Messages
- When data trace is enabled during normal execution of the embedded processor

- Upon assertion of an $\overline{\text{EVTI}}$ pin
- Optionally upon occurrence of a watchpoint
- Data trace overrun error

NOTE

Synchronization information includes an indication of which client (if more than one are present on the device) is being synchronized and the full address for a recent data trace.

4.6 Read/Write Access

Embedded processors complying with Class 3 or 4 shall provide read/write access to user memory-mapped resources according to the Nexus standard, either via the **IEEE 1149.1** interface or the auxiliary pin interface. The capability to perform read/write access shall be provided when the processor is halted or running.

4.6.1 Overview

The Read/Write Access feature supports runtime development visibility needed for real-time embedded applications. This feature also supports program tuning needs of automotive powertrain and disk drive applications.

4.6.2 Read/Write Access Messaging

The Read/Write Access feature provides DMA-like access to user memory-mapped resources when the client is halted or during runtime. One of three options may be used to implement this feature on the embedded processor:

- Read/Write Access using NRR Access Messages
- Read/Write Access using Memory Access Messages
- Read/Write Access using the **IEEE 1149.1** port

4.6.2.1 Read/Write Access Using NRR Access Messages

For target processors that implement the NRRs defined in **Appendix B - Recommendations for Access to Control and Status Registers**, Public Messages are defined in **Section 5 - Nexus Public Messages** and allow a tool to read or write memory locations in the target processor's internal address space via the AUX.

4.6.2.2 Read/Write Access Using Memory Access Messages

Public Messages are defined in **Section 5 - Nexus Public Messages**, allow a tool to read or write memory-mapped locations in the target processor's internal address space, and allow a target to read or write memory locations in the tool via the AUX. This type of read/write access is used in the following circumstances:

- By a tool that implements vendor-defined development control and status registers (instead of the NRRs as defined in **Appendix B - Recommendations for Access to Control and Status Registers**).
- By a target that implements memory substitution in which code and/or data that are normally fetched from target memory are instead fetched from tool memory. This function is started by a target processor watchpoint match and is ended by the tool. Refer to **4.7 - Memory Substitution** for detail on the Memory Substitution feature.
- By a target that allocates a fixed portion of its internal memory map to provide AUX access. Processor reads or writes to this address range result in Read/Write Messages being issued to access memory that exists within the tool. As an example, a target's debug exception handler program may exist only within the tool and be fetched from the tool each time a debug exception occurs.

4.6.2.3 Read/Write Access Using the IEEE 1149.1 Port

Read/Write Access through the **IEEE 1149.1** port can utilize the NRRs or use vendor-defined registers. In either case, Public Messages defined in **Section 5 - Nexus Public Messages** are not used. Refer to **B.3 - Access with the IEEE 1149.1 Interface** for detail on how the **IEEE 1149.1** port is used to access the NRRs.

4.7 Memory Substitution

Embedded processors complying with Class 4 shall provide the capability to activate user memory substitution via the AUX according to the Nexus standard. Memory substitution shall be capable of being activated upon exit of reset. A Class 4 processor optionally may also support memory substitution activated upon the occurrence of a watchpoint or upon the occurrence of a data access or an instruction fetch from a vendor-defined address range. If supported, these optional capabilities must be implemented according to the Nexus standard.

4.7.1 Overview

Memory substitution facilitates the software development process with program execution via the AUX upon exit of reset. Instructions are fetched and data are read from the development tool. Providing this capability via the AUX eliminates the need for a second development port dedicated to the software development process. Additionally, single-stepping with instruction and data fetches via the AUX can be used for a non-real-time, read-only memory (ROM) monitor.

Optionally, the feature can be activated upon the occurrence of a watchpoint. This can support runtime patching for portions of internal ROM, with the patch provided via the AUX. ROM patching during runtime, however, is limited by capability factors of the complying embedded processor. Some factors that may limit the embedded processor are

- The number of watchpoints implemented (one data value patch or one instruction sequence patch per watchpoint)
- The port size and clock rate of the auxiliary pin interface implemented
- The portion of AUX bandwidth allocated for this feature if other messaging activities are also enabled at the same time

Another option is to activate memory substitution upon the occurrence of a data access or an instruction fetch from a vendor-defined address range. For a full memory emulation capability, data reads, data writes, and instruction fetches continue via the AUX until the address of a data access or instruction fetch falls outside a specified address range. The address range is vendor-defined and not typically programmable.

Memory substitution is not intended to be used for tuning parameters during runtime, such as is required for development of automotive powertrain and disk drive applications. There may be other applications, however, that may be able to use this feature during runtime.

4.7.2 Memory Substitution Messaging (MSM)

A Class 4 embedded processor shall be capable of the following three types of memory substitution operations:

- Reading data and fetching instructions via the AUX (both data and instructions substituted by tool)
- Only reading data via the AUX (only data operands substituted by tool)
- Only fetching instructions via the AUX (only instruction operands substituted by tool)

NOTE

Class 4 embedded processors are not required to write data via the AUX.

In memory substitution, the processor will make all qualifying memory-mapped fetches (data, instructions, or both) via the AUX, in a single-step or normal processor mode. Operands that are not enabled for memory substitution shall be accessed normally from user memory. Qualifying memory-mapped fetches are selected by configuring control and status information via the **IEEE 1149.1** port or the AUX.

A Class 4 embedded processor shall be capable of activating memory substitution upon exit from reset and optionally capable of activating it upon the occurrence of a watchpoint or upon the occurrence of a data access or an instruction fetch from a vendor-defined address range.

The Memory Substitution feature can be activated upon exit from reset by configuring control and status information via the **IEEE 1149.1** port or the AUX. It can be activated on a watchpoint occurrence by configuring watchpoint trigger information via the **IEEE 1149.1** port or the AUX. It can be activated on occurrence of a data access or an instruction fetch from a vendor-defined address range. No configuration is required for the latter.

When memory substitution is activated upon exit of reset or a watchpoint occurrence, the processor will make all qualifying memory-mapped fetches via the AUX, until the development tool disables memory substitution. When memory substitution is activated upon the occurrence of a data access or an instruction fetch from a vendor-defined address range, the processor will make all qualifying memory-mapped fetches via the AUX until the address of a data access or instruction fetch falls outside the vendor-defined address range. Once memory substitution is disabled, user memory shall be accessed normally.

MSM facilitates memory substitution by providing messages for access requests and transfers via the AUX. These comprise the following:

- Messaging for a memory substitution access request provided from the processor to an external development tool containing access attributes such as instruction/data, size, and the memory-mapped address. The full address is transmitted for each memory substitution access request.

- Messaging for a memory substitution transfer provided from the external development tool to a processor containing the instruction or data specified by access attributes.

- Messaging for the *last* memory substitution transfer provided from the external development tool to a processor containing the *last* instruction or data specified by access attributes and containing a disable command for MSM. Subsequent memory-mapped accesses will be accessed normally from the internal memory-mapped resource designated by the access attributes.

For patching a ROM instruction sequence, the last memory substitution transfer may consist of a direct branch to the address following the patched instruction sequence.

4.8 Breakpoints/Watchpoints

Embedded processors complying with Class 1, 2, 3, or 4 shall provide a minimum of two instruction/data hardware breakpoints.[5] Embedded processors complying with Class 2, 3, or 4 shall provide, according to the Nexus standard, the capability to message via the AUX any occurrence of a watchpoint.

4.8.1 Overview

The Breakpoint and Watchpoint features facilitate the software development process by allowing the developer to halt at a specific processor state or to signal a specific processor state. If there is an internal ROM or if a breakpoint or trap instruction does not exist in the vendor's architecture, then these features become a valuable tool for development.

4.8.2 Breakpoint/Watchpoint Messaging

Breakpoints and watchpoints comprise the following:

- Data breakpoint—processor is halted at an appropriate instruction boundary after a trigger is set at a data valid time. The trigger is set when the data address and/or data value matches a pre-selected address and/or value.
- Instruction breakpoint—processor is halted when all previous instructions are retired and just prior to when any architectural state is changed by the instruction associated with a pre-selected address.
- Watchpoint—a data or instruction breakpoint that does not cause the processor to halt. A Watchpoint Match Message via the AUX is used to signal that the condition occurred.

[5]Optionally, the two BWC Registers may be combined with the two Data Trace Attribute Registers so that a total of two registers may be simultaneously active, i.e., two BWC Registers, two Data Trace Attribute Registers, or one BWC Register and one Data Trace Attribute Register.

4.9 Port Replacement and Port Sharing (Optional)

Embedded processors complying with Class 2, 3, or 4 may optionally implement support on the auxiliary pins according to the Nexus standard for LSIO port replacement. Embedded processors complying with Class 2, 3, or 4 may optionally share AUX pins according to the Nexus standard with a second HSIO port.

4.9.1 Overview

In embedded processor applications, the use of every pin is scrutinized by developers of embedded processors. Inevitably there are never enough pins available on the embedded processor to meet both the application and development needs. Pins that are designated for product development are often reduced or removed to make way for other pin functions directly used in the application. Port replacement and sharing support is intended to solve this dilemma by using common embedded processor ports for a secondary development support function.

4.9.2 Port Replacement for Low-Speed I/O (LSIO) Pins

Port replacement provides a mechanism for LSIO pin functions to be replaced using messages via the AUX. The standard messages between the development tool and AUX provide the necessary information for the development tool to replace the LSIO port (with additional delay).

The mechanism is enabled in a plug-and-play manner. When a development tool is connected to the AUX, it enables the AUX with Port Replacement Messages. When no development tool is connected, the port functions as only an LSIO port.

Up to 16 bits of LSIO port replacement are allowed with the standard Port Replacement Messages transmitted via the AUX, as shown in **Figure 4-1**. The standard messages transmitted between the development tool and embedded processor provide the necessary information for the development tool to replace the LSIO port (with additional delay). The specific format of the Port Replacement Messages is outlined in **Section 5 - Nexus Public Messages**.

.

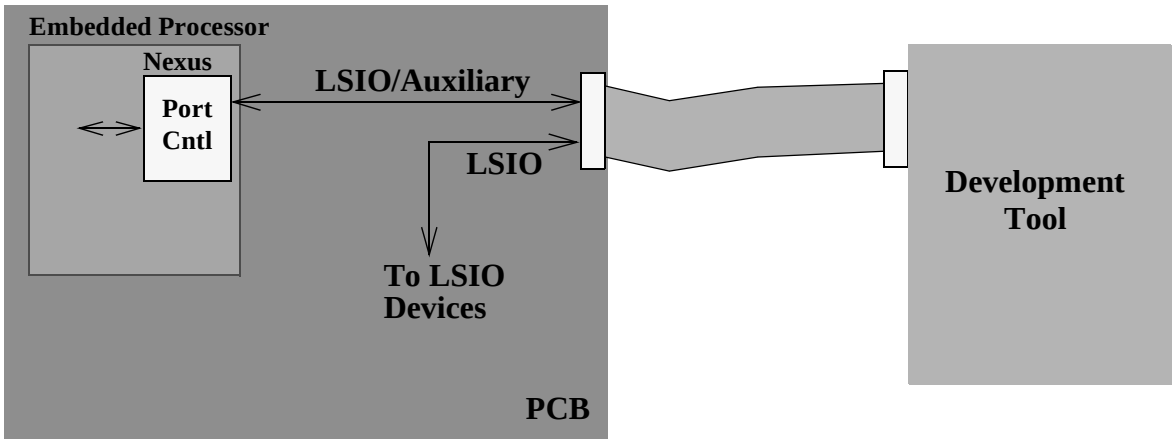


Figure 4-1—Port Replacement for LSIO Pins

Most messages transmitted in a typical application will contain development information. Upon occurrence of an LSIO state change, however, a Port Replacement Message will be transmitted. Port Replacement Messages will be transmitted either by the embedded processor to the tool (messages containing information for low-speed output pins) or by the tool to the embedded processor (messages containing information for low-speed input pins).

Port Replacement Messages from the embedded processor will contain two essential packets: one packet indicating the direction of each LSIO pin and another indicating the state of all LSIO pins. Port Replacement Messages from the tool will contain one essential packet indicating the state of all LSIO pins. This information will be used by the embedded processor and tool to maintain the correct state and direction for all LSIO pins.

Note that if the development tool is not connected to perform the port replacement function, a special connector should be connected so that the LSIO signals are connected to the LSIO devices on the target board. For production boards that do not require the port replacement function, no connector is required for signals that are connected directly by board traces.

The development tool shall implement the following rules to assure proper port replacement:

- Prior to receiving the first Port Replacement Message after the embedded processor port has been reset, all replacement pins on the tool should default to input.
- Prior to receiving the first Port Replacement Message after the embedded processor port has been reset, the tool should not generate Port Replacement Messages to the embedded processor.

- When the processor writes to the LSIO port registers, a Port Replacement Message will be transmitted to the tool. The tool then drives the pins configured as outputs to their programmed states.

- Whenever any pin configured as an input changes, the tool transmits a Port Replacement Message to the embedded processor for update of the state internally (enabled interrupt may be generated).

4.9.3 Port Replacement for High-Speed I/O (HSIO) Pins (Port Sharing)

For embedded processors that incorporate HSIO pins, a slightly different technique is provided for simultaneously using both a primary pin function, such as an external bus port, and a secondary pin function, such as Nexus development pins. For example, an L2 cache bus function and the AUX OUT function may utilize the same pins. This procedure is also referred to as port sharing.

Most bus traffic in a typical application will be due to external bus cycles on the shared pins for accessing system resources. Due to the high-speed nature of the HSIO external bus port, only the data-out portion of the Nexus port can be simultaneously shared with the primary function, because the Nexus data-out signals have the most stringent bandwidth requirements. During external bus cycles, AUX control signals are negated and the development tool ignores the external bus information. Upon occurrence of a condition that generates development information (e.g., BTM and DTM), a corresponding message is sent out via the shared pins and captured by the tool.

This solution provides a tremendous advantage in reducing the total number of actual development support pins. Refer to **Figure 4-2** for an illustration.

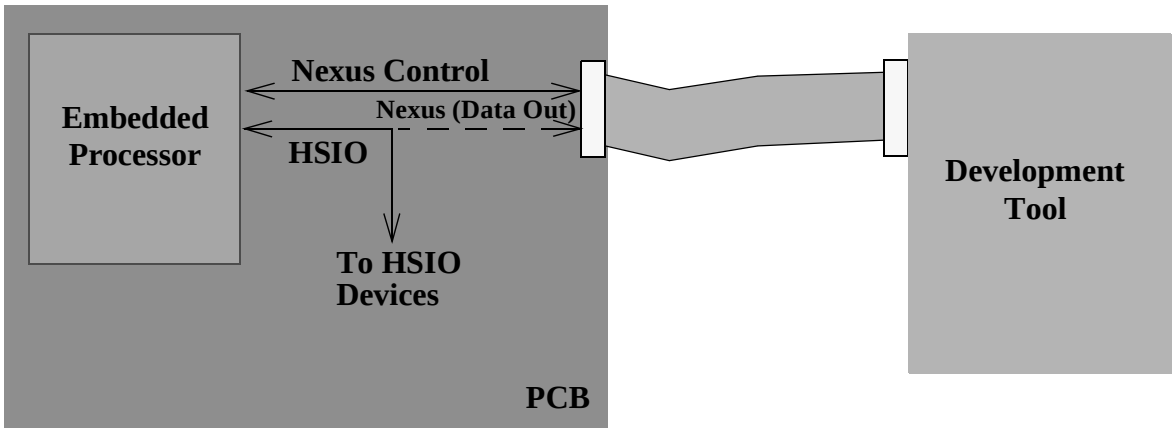


Figure 4-2—Port Replacement for HSIO Pins (Port Sharing)

4.10 Data Acquisition (Optional)

Embedded processors complying with Class 2, 3, or 4 may optionally implement support for data acquisition by the development tool from the embedded processor, via the AUX, according to the Nexus standard.

4.10.1 Overview

The Data Acquisition feature provides a mechanism for visibility of intermediate variables calculated by the embedded processor. An application includes time-critical parameters passed to an external coprocessor for rapid prototyping. The embedded processor is required to queue up data for acquisition by the development tool.

4.10.2 Data Acquisition Messaging (DQM)

DQM provides the capability to message, on the AUX, internal data related to one another. Because of construction, DQM provides a more efficiently packed message than DTM.

DQM facilitates data acquisition by providing several key types of visibility: display data ID tag (to specify which group of data) and all data values. The display data ID tag is typically a reference number to identify the data, e.g., “3” may represent time-critical parameters passed to an external coprocessor for rapid prototyping.

As mentioned above, the embedded processor must queue up Data Acquisition Messages. In the Nexus standard, a user memory-mapped interface and protocol are recommended (not required) for the embedded processor to queue up Data Acquisition Messages. The user memory-mapped locations are configured via the **IEEE 1149.1** or auxiliary pin interface. The recommended protocol consists of writing to a designated user memory-mapped location to generate a Data Acquisition Message with a specific display data ID tag.

4.11 Timestamping (Optional)

Embedded processors complying with Class 2, 3, or 4 may optionally implement support for timestamping via the AUX or through vendor-defined pins.

4.11.1 Overview

Because messages defined in the Nexus standard are queued up within the embedded processor before being transmitted to the external development tool, the exact time that a particular event (i.e., indirect branch or data write) occurred is lost. The exact event time can be especially important for software quality

assurance (SQA) tools such as performance analysis and code coverage measurement tools.

Two different mechanisms may be optionally implemented to meet compliance in accordance with the Nexus standard:

1. Timestamping using an optional field within each AUX OUT message
2. Timestamping using optional pins

4.11.2 Timestamping via AUX

For each message defined in **Section 5 - Nexus Public Messages**, a timestamping field may be added to the end of the message that indicates to the tool the time the specific message occurred. This value can be either an absolute timestamp generated within the embedded processor or a timestamp relative to when the last message entered the internal first-in, first-out (FIFO) buffer.

The absolute timestamp entails significant bandwidth overhead. The relative timestamp requires the development tool to calculate timestamp values from the time the first trace message enters the FIFO buffer. The method chosen for timestamping through the AUX is vendor-defined.

4.11.3 Timestamping via pins

To avoid the bandwidth impact of adding a field to each message, an alternative method for timestamping entails using either vendor-defined pins or using a secondary function on the optional EVTO pin. Each time a message enters the internal FIFO buffer, the output signal asserts for the period of one clock cycle (see Note). The number of pins utilized and the specific functionality of each pin are vendor-defined.

NOTE

If vendor-defined pins are used for timestamping, the frequency of the clock output is also vendor-defined. If EVTO is used, the signal will assert for one cycle of Message Clockout (MCKO) for each timestamped event.

SECTION 5
Nexus Public Messages

The AUX provides a high-speed communication link between the tool and a target processor. All communication over this link uses messages in one or both directions.

The format and meaning of certain messages, called Public Messages, are defined by this specification. Other messages can be vendor defined and defined by the target processor vendor. Tools require enhanced capability to be able to support Vendor-Defined Messages.

All messages start with a 6-bit transfer code (TCODE), which uniquely defines the type of message. Fifty-six TCODEs (values 0 to 55) indicate that the message is a Public Message defined by the Nexus standard or reserved for future definition by the Nexus standard. Seven TCODEs (values 56 to 62) indicate that the message is a Vendor-Defined Message. One TCODE (value 63) indicates that the message is a Vendor-Defined Message, and then a second level code designated by the vendor further identifies the specific message.

In addition to being transferred over the AUX, Public Messages can also be transferred via an **IEEE 1149.1** port using the method described in **Section 8 - IEEE 1149.1 Message Protocol**.

5.1 Compliance Requirements for Public Messages

Embedded processors complying with Class 2, 3, or 4 shall implement messaging via the AUX according to the Nexus standard. Embedded processors complying with Class 1 may optionally implement messaging via the **IEEE 1149.1** interface.

Embedded processors shall implement the minimum Public Messages as required per the compliance class. **Table 5-1** lists the minimum required Public Messages per compliance class. Embedded processors may optionally implement Vendor-Defined Messages.

Table 5-1—Minimum Required Public Messages

Required Feature	Minimum Required Public Messages	Class 2	Class 3	Class 4
Device Identification	- Device ID Message[a]	X	X	X
Task/Process ID	- Ownership Trace Message	X	X	X
Watchpoint Indication	- Watchpoint Match Message	X	X	X
Error	- Error Message	X	X	X
Program Trace	Program Trace using traditional branch messages: - Program Trace - Direct Branch Message - Program Trace - Indirect Branch Message - Program Trace - Synchronization Message[b]	X	X	X
	OR			
	Program Trace using branch history messages: - Program Trace - Indirect Branch History Message - Program Trace - Synchronization Message[c] - Program Trace - Resource Full Message			
Data Trace	- Data Trace - Data Write Message - Data Trace - Data Write with Sync Message	--	X	X
Read/Write Access	Read/Write Access using registers defined in **Appendix B - Recommendations for Access to Control and Status Registers**: - NRR Access - Target Ready Message - NRR Access - Read Register Message - NRR Access - Write Register Message - NRR Access - Read/Write Response Message	--	X	X
	OR			
	Read/Write Access via **IEEE 1149.1** port: - No Public Messages Required (see **Section 8.3 - Read/Write Access via the IEEE 1149.1 Port**)			
	OR			
	Read/Write Access using vendor-defined registers: - Memory Access - Read Target Message - Memory Access - Write Target Message - Memory Access - Read Next Target Data Message - Memory Access - Write Next Target Data Message - Memory Access - Target Response Message			
Memory Substitution	- Memory Access - Read Tool Message - Memory Access - Read Next Tool Data Message - Memory Access - Tool Response Message	--	--	X

a. For embedded processors that do not implement an **IEEE 1149.1-**compliant IDCODE
b. The Direct Branch with Sync Message and/or Indirect Branch with Sync Message may be implemented instead of the Synchronization Message.
c. The Indirect Branch History with Sync Message may be implemented instead of the Synchronization Message.

5.2 Definitions and Terminology

The following terms relate to Public Messages:

Message: Each message starts with a 6-bit TCODE, which defines the type of information carried in the message and its format. The TCODE packet length for all Public Messages must be 6 bits. When messages are transferred via the AUX, message start/end (MSE) signaling protocol, described in **Section 7 - AUX Message Protocol**, defines the start and the end of each message.

Transmission Order: Messages are transmitted LSB(s) first. Additionally, Program/Data Trace Messages are transmitted in a temporal order so that the transmission of messages should correlate as closely as possible with the temporal occurrence of activity on the embedded processor.

Packet: A packet is a distinct piece of the information contained within a message, and messages may contain one or more packets. A common alternative term for a packet is a *field*. When messages are transferred via the AUX, MSE signaling protocol defines the end of each variable-length packet.

Port Boundary: A port boundary relates the size of a packet to the width of the AUX IN or AUX OUT.

Variable: Specifying that a packet is variable-size means that the message must contain the packet, but that the packet's size may vary from a minimum of 1 bit. When messages are transferred via the AUX, variable-size packets must end on a port boundary. If necessary, they must zero-fill bit positions beyond the highest order bit of the variable data. Because variable-size packets may be of different lengths in messages of the same type, the tool must use the MSE signaling protocol to determine the end of packet boundaries.

Vendor-Fixed: The term *vendor-fixed* is used to indicate allowances in the Nexus standard to match characteristics of a vendor's device. Vendor-fixed packets may be of zero length (not implemented). For a tool to interpret message content, it must determine from the device ID (or **IEEE 1149.1**-defined IDCODE) whether vendor-fixed packets exist in each type of message.

Some vendor-fixed packets have their lengths fixed by the Nexus specification. Other vendor-fixed packets are target processor dependent and have a fixed size determined by the processor vendor.

Vendor-Variable: The term *vendor-variable* is used to indicate allowances in the Nexus standard to match characteristics of a vendor's device. Vendor-variable packets may be of zero length (not implemented). For a tool to interpret message content, it must determine from the device ID (or **IEEE 1149.1**-defined IDCODE) whether vendor-variable packets exist in each type of message. When messages are transferred via the AUX, vendor-variable packets must end on a port

boundary. If necessary, they must zero-fill bit positions beyond the highest order bit of the variable data. Because variable-size packets may be of different lengths in messages of the same type, the tool must use the MSE signaling protocol to determine the end of packet boundaries.

These vendor-variable packets are target processor dependent and have a variable size determined by the processor vendor. These packets are normally reserved for the end of a Public Message where the vendor may implement additional fields.

For messages that have multiple vendor-variable packets, either dynamic allocation of the field (zero value in some cases, nonzero value in others) must be controlled by the external development tool, or a vendor-defined mechanism must be created to inform the development tool when the allocation of these fields is changed internally by the target.

Sync and Non-Sync Trace Messages: Program/Data Trace Messages fall into two broad categories—*non-sync* (or normal) versions and *sync* versions. The main difference between the two categories is that *sync* versions include full addresses whereas *non-sync* versions contain addresses that are relative to a previous trace message.

Next Address Generation: To minimize the size of trace messages, the address packets in the *non-sync* versions of all trace messages contain a compressed address. This compressed address, called *the unique portion of the address,* is relative to the address associated with a previous trace message of the same type. Thus, Program Trace Messages contain an address that is relative to the previous Program Trace Message; Data Trace Messages contain an address that is relative to the previous Data Trace Message.

The target processor computes the relative address by exclusive-OR-ing the current program or data address with the full address associated with the previous Program Trace Message or Data Trace Message (see **Figure 5-1**).

Number of Messages Cancelled: Several messages for program and data trace synchronization (Direct Branch with Sync Message, Indirect Branch with Sync Message, Indirect Branch History with Sync Message, Data Write with Sync Message, Data Read with Sync Message) contain a packet for the number of messages cancelled. There are three vendor-defined interpretations of this field:

1. For embedded processors that transmit only valid messages, this field can be omitted.

2. For embedded processors that do not queue up Program/Data Trace Messages as they become backlogged, but can truncate the current message as it is being transmitted to send out a fresher message, this field will have a value of 1 if the previous message has been truncated.

3. For embedded processors that send out preliminary Program Trace Messages (e.g., speculative execution) and later correct the trace information by cancelling fully transmitted messages, this field will notify the tool of the number of fully transmitted program (or data) messages to be cancelled.

Example of how the target processor generates the address to send in a trace message:

Previous absolute address (A1) = 0x003FC01,
Absolute address associated with new trace occurrence (A2) = 0x0003F365

```
     A1 =  0000 0000 0000 0011 1111 1100 0000 0001
     A2 =  0000 0000 0000 0011 1111 0011 0110 0101
A1⊕A2 =  0000 0000 0000 0000 0000 1111 0110 0100
```

The unique portion of the address (M1), sent in the message (high-order zeros are suppressed):

```
     M1 =  1111 0110 0100
```

Example of how the tool recreates the address based on its previously calculated address and the address contained in the trace message:

Previously calculated address (A1) = 0x003FC01,
Address in message (M1) = 0xF64

```
     A1 =  0000 0000 0000 0011 1111 1100 0000 0001
     M1 =  0000 0000 0000 0000 0000 1111 0110 0100
A1⊕M1 =  0000 0000 0000 0011 1111 0011 0110 0101
```

Address recreated by the tool = 0x0003F365

Figure 5-1—Next Address Generation Example

Periodic Message Counter: Because addresses contained in *non-sync* Program/Data Trace Messages are relative, the loss or corruption of a trace message means that the tool will be unable to correctly recreate addresses following the corruption or loss. To minimize the effect of any such loss or corruption of a trace message, the target processor must send a *sync* version at least every 256 trace messages.

To provide this function, the target processor must maintain two periodic message counters, one for counting normal Program Trace Messages and the other for counting normal Data Trace Messages.

Instruction Address Threads and Data Address Threads: On embedded processors that implement data and program trace, there will be an address thread for each type of trace: the data address thread and the instruction address thread. Messages containing a data address packet will be encoded and compressed using the data address most recently transmitted, thus creating a data address thread. Likewise, messages containing an instruction address packet will be encoded and compressed using the instruction address most recently transmitted, thus creating an instruction address thread.

Source of Message Transmission (SRC field): All of the Public Messages contain a vendor-fixed packet that may be used to identify which client was the source of the message transmission. In embedded processors that consist of only a single client, this packet need not be transmitted. For embedded processors that consist of multiple clients, this packet must be transmitted as part of the message to identify the source of the message transmission.

Branch History (HIST field): For Indirect Branch History Messages, the branch history packet provides a history of direct branch executions used for re-constructing program flow. This packet is implemented as a left-shifting shift register. The register is always pre-loaded with a value of 1. This bit acts as a stop bit so that the development tools can determine which bit is the end of the history information. This pre-loaded bit itself is not part of the history, but is transmitted with the packet.

A value of 1 is shifted into the history buffer on a taken branch (conditional or unconditional) and on any instruction whose predicate condition resolved as true. A value of 0 is shifted into the history buffer on any instruction whose predicate condition executed as false as well as on branches not taken. This includes indirect as well as direct branches not taken.

Instruction Units (I-CNT field): Most Program Trace Messages have a packet that indicates the number of instruction units executed since the last taken branch. In target architectures in which all instructions are the same size, this packet contains the actual number of instructions executed since the last taken branch.

If instructions are of variable size, then the number reported is the number of instruction units. The instruction unit represents the number of bytes or words associated with the highest common denominator of the variable instruction sizes.

5.3 Detailed Description of Public Messages

In this subsection, Public Messages are grouped according to their function. The complete list of Nexus Public Messages is listed in **Table 5-2**.

Table 5-2—Nexus Public Messages

Message Name	TCODE Value	Direction
Debug Status	0	From target
Device ID	1	From target
Ownership Trace	2	From target
Program Trace - Direct Branch	3	From target
Program Trace - Indirect Branch	4	From target
Data Trace - Data Write	5	From target
Data Trace - Data Read	6	From target
Data Acquisition	7	From target
Error	8	From target
Program Trace - Synchronization	9	From target
Program Trace - Correction	10	From target
Program Trace - Direct Branch with Sync	11	From target
Program Trace - Indirect Branch with Sync	12	From target
Data Trace - Data Write with Sync	13	From target
Data Trace - Data Read with Sync	14	From target
Watchpoint Match	15	From target
NRR Access - Target Ready	16	Both ways
NRR Access - Read Register	17	From tool
NRR Access - Write Register	18	From tool
NRR Access - Read/Write Response	19	Both ways
Port Replacement - Output	20	From target
Port Replacement - Input	21	From tool
Memory Access - Read Target/Tool	22	Both ways
Memory Access - Write Target/Tool	23	Both ways
Memory Access - Read Next Target/Tool Data	24	Both ways
Memory Access - Write Next Target/Tool Data	25	Both ways
Memory Access - Target/Tool Response	26	Both ways
Program Trace - Resource Full	27	From target
Program Trace - Indirect Branch History	28	From target
Program Trace - Indirect Branch History with Sync	29	From target
Program Trace - Repeat Branch	30	From target

Table 5-2—Nexus Public Messages *(Continued)*

Message Name	TCODE Value	Direction
Program Trace - Repeat Instruction	31	From target
Program Trace - Repeat Instruction with Sync	32	From target
Program Trace - Correlation	33	From target
Reserved	34–55	Both ways
Vendor-Defined Message	56–62	Both ways
Vendor-Defined Extension Message	63 (0x3F)	Both ways

In the Public Message descriptions in **5.3.1 - Debug Status Message** through **5.3.33 - Memory Access - Tool Response Message**, each description table lists the MSBs (transmitted last) at the top of the table and the LSBs (transmitted first) at the bottom of the table (see **Table 5-3** through **Table 5-40**).

5.3.1 Debug Status Message

Message Description: The Debug Status Message is output by the target whenever there is a change of state of any of the following:

- Entry to the debug exception handler
- Exit from the debug exception handler
- Change in power-managed state of the processor
- Detection of a breakpoint

In addition to having the target processor send a Debug Status Message whenever the debug status changes, the tool is able to request the current debug status at any time. For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the tool requests the current debug status by sending a Read Register Message containing the Development Status opcode. For target processors that implement device-specific registers, the API knows which vendor register(s) to read to obtain debug status.

Table 5-3—Debug Status Message Format

Debug Status Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	STATUS	Vendor-fixed	Status information[a]
0	SRC	Vendor-fixed	Client that is source of message.
6	TCODE	Fixed	Value = 0

a. For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the status packet contains the same information as the DS Register. For target processors that implement device specific registers, the status packet must provide all the information required by the API.

5.3.2 Device ID Message

Message Description: The Device ID Message is used for AUX-only Nexus implementations. JTAG-based Nexus implementations should use the **IEEE 1149.1**-defined JTAG ID Register for device ID.

For AUX-only implementations, if the AUX is enabled, i.e., a tool is connected, this message is output by the target processor only after the target's debug logic has been reset by the tool. The tool resets the target's debug logic by asserting and de-asserting the Reset In ($\overline{\text{RSTI}}$) signal.

NOTE

A Device ID Message is not automatically output following power-on reset, even when a tool is connected. The tool must specifically reset the target's debug logic for this message to occur.

In addition to having the target processor send a Device ID Message following a debug logic reset, the tool is able to request the device ID at any time. For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the tool requests the device ID by sending a Read Register Message containing the device ID opcode. For target processors that implement device-specific registers, the API knows which register to read to obtain the device ID.

Table 5-4—Device ID Message Format

Device ID Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
32	ID	Fixed	ID information. Refer to **B.5.1 - Device ID (DID) Register**.
6	TCODE	Fixed	Value = 1

5.3.3 Ownership Trace Message

Message Description: There are three ways in which the Ownership Trace Message may occur:

1. For target processors in which the OTM Register is a read-only alias of a process ID register, this message is output whenever the process ID changes.

2. For target processors where the OTM Register is directly written by the operating system or application code to indicate the current process or task, this message is output whenever the operating system writes to the OTM Register.

3. For target processors using virtual memory, this message is output immediately prior to (or immediately following) a Program/Data Trace Message with synchronization produced when a periodic message counter expires. Ownership Trace Messages allow a tool to be regularly updated with the latest process ID.

Table 5-5—Ownership Trace Message Format

Ownership Trace Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	PROCESS	Vendor-fixed	Task/process ID.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 2

5.3.4 Program Trace - Direct Branch Message

Message Description: For target processors that do not implement Branch History Messages, the Program Trace - Direct Branch Message is output whenever there is a change of program flow caused by a conditional or unconditional branch. To conserve AUX bandwidth and trace buffer space, the target processor may queue trace information about taken direct branches and output one message containing up to eight I-CNT packets.

Table 5-6—Program Trace - Direct Branch Message Format

Program Trace - Direct Branch Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch. Each message may contain up to eight of these packets, each one corresponding to a direct branch taken.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 3

5.3.5 Program Trace - Indirect Branch Message

Message Description: For target processors that do not implement Branch History Messages, the Program Trace - Indirect Branch Message is output whenever there is a change of program flow caused by a subroutine call, return instruction, asynchronous interrupt/trap, or conditional/unconditional indirect branch instruction where the target address is determined at runtime.

Table 5-7—Program Trace - Indirect Branch Message Format

Program Trace - Indirect Branch Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	U-ADDR	Variable	The unique portion of the branch target address for a taken indirect branch or exception.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see **Table 5-8**).
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 4

The optional B-TYPE field can be used to distinguish between events that cause Indirect Branch Messages (i.e., indirect branch vs. exception). **Table 5-8** shows recommended encodings for the B-TYPE field. The size of the packet transmitted is determined by the number of encodings a client uses.

Table 5-8—Recommended B-TYPE Encodings

B-TYPE Value	**Description**
0	Indirect Branch
1	Exception
2	Hardware Loop
Other	Reserved

5.3.6 Program Trace - Direct Branch with Sync Message

Message Description: For target processors that do not implement Branch History Messages, the Program Trace - Direct Branch with Sync Message is output when any of the following conditions occurs:

1. Upon exit from system reset to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool. This trace message follows exit from reset for target processors not capable of immediately generating a Program Trace - Synchronization Message.

2. Upon detection of a direct branch after program trace is enabled during normal execution of the embedded processor.

3. Upon exit from a power-down state to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool.

4. Upon detection of a direct branch following the processor's exit from debug mode.

5. Upon detection of a direct branch after an overrun condition had previously occurred in which one or more branch trace occurrences were discarded by the target processor's debug logic. To inform the tool that an overrun condition occurred, the target outputs an Error Message (TCODE = 8) with an Error code (ECODE) value of 00001 or 00111 immediately prior to the Program Trace - Direct Branch with Sync Message.

6. Upon detection of a direct branch after the periodic Program Trace Message counter has expired indicating that 255 *non-sync* Program Trace Messages have been sent since the last *sync* Program Trace Message. The value of 255 is a maximum number; target processors may use a smaller value.

7. Upon detection of a direct branch after the $\overline{\text{EVTI}}$ pin has been asserted and the EIT field in the DC Register (**Appendix B - Recommendations for Access to Control and Status Registers**) determines that $\overline{\text{EVTI}}$ pin action is to generate program trace synchronization. This message is output by target processors not capable of immediately generating a Program Trace - Synchronization Message.

8. Upon overflow of the sequential instruction unit counter. Because a limited counter size must be implemented in the embedded processor, there will likely be sequential instruction sequences (with no taken branches), which will cause the counter to overflow.

9. (Optional) Upon the occurrence of a watchpoint match and the next taken direct branch. This trace message follows the Watchpoint Match Message for target processors not capable of immediately generating a Program Trace - Synchronization Message.

Table 5-9—Program Trace - Direct Branch with Sync Message Format

Program Trace - Direct Branch with Sync Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	F-ADDR	Variable	The full target address for a taken direct branch. MSBs that have a value of 0 may be truncated.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	CANCEL	Vendor-variable	Number of previous Program Trace Messages that should be ignored by the tool. This packet is generated only by processors performing speculative execution where a trace message may be output before it is known whether the branch was actually taken.
0	DCONT	Vendor-fixed	Indicates whether synchronization is a result of a discontinuity in the program flow relative to the prior Program Trace Message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 11

5.3.7 Program Trace - Indirect Branch with Sync Message

Message Description: For target processors that do not implement Branch History Messages, the Program Trace - Indirect Branch with Sync Message is output when any of the following conditions occurs:

1. Upon exit from system reset to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool. This trace message follows exit from reset for target processors not capable of immediately generating a Program Trace - Synchronization Message.

2. Upon detection of an indirect branch after program trace is enabled during normal execution of the embedded processor.

3. Upon exit from a power-down state to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool.

4. Upon detection of an indirect branch following the processor's exit from debug mode.

5. Upon detection of an indirect branch (a change of program flow caused by a subroutine call, return instruction, or asynchronous interrupt/trap) after an overrun condition had previously occurred in which one or more branch trace occurrences were discarded by the target processor's debug logic. To inform the tool that an overrun condition occurred, the target outputs an Error Message (TCODE = 8) with an ECODE value of 00001 or 00111 immediately prior to the Program Trace - Indirect Branch with Sync Message.

6. Upon detection of an indirect branch after the periodic Program Trace Message counter has expired, indicating that 255 *non-sync* Program Trace Messages have been sent since the last *sync* Program Trace Message. The value of 255 is a maximum number; target processors may use a smaller value.

7. Upon detection of an indirect branch when a debug control register field specifies that $\overline{\text{EVTI}}$ pin action is to generate program trace synchronization and the $\overline{\text{EVTI}}$ pin has been asserted. This message is output by target processors not capable of immediately generating a Program Trace - Synchronization Message.

8. Upon overflow of the sequential instruction unit counter. Because a limited counter size must be implemented in the embedded processor, there will likely be sequential instruction sequences (with no taken branches), which will cause the counter to overflow.

9. (Optional) Upon the occurrence of a watchpoint match and the next taken indirect branch. This trace message follows the Watchpoint Match Message for target processors not capable of immediately generating a Program Trace - Synchronization Message.

Table 5-10—Program Trace - Indirect Branch with Sync Message Format

Program Trace - Indirect Branch with Sync Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	F-ADDR	Variable	The full target address for a taken indirect branch or exception. MSBs that have a value of 0 may be truncated.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	CANCEL	Vendor-variable	Number of previous Program Trace Messages that should be ignored by the tool. This packet is generated only by processors performing speculative execution where a trace message may be output before it is known whether the branch was actually taken.
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see **Table 5-8**).
0	DCONT	Vendor-fixed	Indicates whether synchronization is a result of a discontinuity in the program flow relative to the prior Program Trace Message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 12

5.3.8 Program Trace - Resource Full Message

Message Description: Certain resources internal to the device, such as counters and history buffers, have hardware limitations to their size. To avoid losing information when these resources become full, a Resource Full Message can be transmitted. The information from this message is added or concatenated with information from subsequent messages to interpret the full picture of what has transpired. Multiple Resource Full Messages can occur before the arrival of the message with which the information belongs.

The Program Trace - Resource Full Message is sent out when any of the resources defined in **Table 5-12** have reached their maximum value.

Table 5-11—Program Trace - Resource Full Message Format

Program Trace - Resource Full Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for times-tamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
0	RDATA	Vendor-variable	Data defined by the resource code (RCODE). Zero to eight variable length packets can be included within a single message. The exact count depends on the RCODE value. Refer to **Table 5-12** for detail.
4	RCODE	Fixed	Resource code. This code indicates which inter-nal resource has reached its maximum value. Refer to **Table 5-12** for detail.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 27

Table 5-12—Recommended Resource Code (RCODE) Description

Resource Code	Resource	Data Packet Value
0b0000	Program Trace - Sequential Instruction Counter	Number of instruction units executed since the last taken branch.
0b0001	Program Trace - Branch/ Predicate History	Branch/Predicate instruction history. This packet is terminated by a stop bit set to 1 after the last history bit. This ending allows the tool to determine which bits are part of the history field and which are padded zeros.
0b0010-0b0111	Reserved	Reserved
0b1000-0b1111	Vendor-defined	Vendor-defined

5.3.9 Program Trace - Indirect Branch History Message

Message Description: In order to alleviate the bandwidth concerns on higher performing processors or to trace predicated instructions, an alternative method for generating Program Trace Messages is supported.

For target processors that do not implement traditional Branch History Messages, the Program Trace - Indirect Branch History Message is output whenever there is a change of program flow caused by a subroutine call, return instruction, or asynchronous interrupt/trap or conditional/unconditional indirect branch instruction where the target address is determined at runtime.

The history field represents direct branch/predicate instruction information. See explanation of the history field in **5.2 - Definitions and Terminology**.

.

Table 5-13—Program Trace - Indirect Branch History Message Format

Program Trace - Indirect Branch History Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	HIST	Variable	Branch/Predicate Instruction History. This packet is terminated by a stop bit set to 1 after the last history bit. This ending allows the tool to determine which bits are part of the history field and which are padded zeros.
1	U-ADDR	Variable	The unique portion of the branch target address for a taken indirect branch or exception.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see **Table 5-8**).
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 28

5.3.10 Program Trace - Indirect Branch History with Sync Message

Message Description: For target processors that do not implement traditional Branch History Messages, the Program Trace - Indirect Branch History with Sync Message is output when any of the following conditions occurs:

1. Upon exit from system reset to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool. This trace message follows exit from reset for target processors not capable of immediately generating a Program Trace - Synchronization Message.

2. Upon detection of an indirect branch after program trace is enabled during normal execution of the embedded processor.

3. Upon exit from a power-down state to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool.

4. Upon detection of an indirect branch following the processor's exit from debug mode.

5. Upon detection of an indirect branch after an overrun condition had previously occurred in which one or more branch trace occurrences were discarded by the target processor's debug logic. To inform the tool that an overrun condition occurred, the target outputs an Error Message (TCODE = 8) with an ECODE value of 00001 or 00111 immediately prior to the Program Trace - Indirect Branch History with Sync Message.

6. Upon detection of an indirect branch after the periodic Program Trace Message counter has expired, indicating that 255 *non-sync* Program Trace Messages have been sent since the last *sync* Program Trace Message. The value of 255 is a maximum number; target processors may use a smaller value.

7. Upon detection of an indirect branch when a debug control register field specifies that $\overline{\text{EVTI}}$ pin action is to generate program trace synchronization and the $\overline{\text{EVTI}}$ pin has been asserted. This message is output by target processors not capable of immediately generating a Program Trace - Synchronization Message.

8. Upon overflow of the sequential instruction unit counter. Because a limited counter size must be implemented in the embedded processor, there will likely be sequential instruction sequences (with no taken branches), which will cause the counter to overflow.

9. (Optional) Upon the occurrence of a watchpoint match and the next taken indirect branch. This trace message follows the Watchpoint Match

Message for target processors not capable of immediately generating a Program Trace - Synchronization Message.

Table 5-14—Program Trace - Indirect Branch History with Sync Message Format

Program Trace - Indirect Branch History with Sync Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	HIST	Variable	Branch/Predicate Instruction History. This packet is terminated by a stop bit set to 1 after the last history bit. This ending allows the tool to determine which bits are part of the history field and which are padded zeros.
1	F-ADDR	Variable	The full target address for a taken indirect branch or exception. MSBs that have a value of 0 may be truncated.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	CANCEL	Vendor-variable	Number of previous Program Trace Messages that should be ignored by the tool. This packet is generated only by processors performing speculative execution where a trace message may be output before it is known whether the branch was actually taken.
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see **Table 5-8**).
0	DCONT	Vendor-fixed	Indicates whether synchronization is a result of a discontinuity in the program flow relative to the prior Program Trace Message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 29

5.3.11 Program Trace - Synchronization Message

Message Description: The Program Trace - Synchronization Message is output by the target processor when any of the following conditions occurs:

1. Upon exit from reset to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool.

2. When program trace is enabled during normal execution of the embedded processor.

3. Upon exit from a power-down state to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool.

4. Upon exiting from debug mode.

5. After an overrun condition had previously occurred in which one or more branch trace occurrences were discarded by the target processor's debug logic. To inform the tool that an overrun condition occurred, the target outputs an Error Message (TCODE = 8) with an ECODE value of 00001 or 00111 immediately prior to the Program Trace - Synchronization Message.

6. Upon expiration of the periodic Program Trace Message counter, indicating that 255 *non-sync* Program Trace Messages have been sent since the last *sync* Program Trace Message. The value of 255 is a maximum number; target processors may use a smaller value.

7. After the $\overline{\text{EVTI}}$ pin has been asserted when a debug control register field specifies that $\overline{\text{EVTI}}$ pin action is to generate program trace synchronization.

8. Upon overflow of the sequential instruction unit counter. Because a limited counter size must be implemented in the embedded processor, there will likely be sequential instruction sequences (with no taken branches), which will cause the counter to overflow.

9. (Optional) Upon the occurrence of a watchpoint match. This trace message immediately follows the Watchpoint Match Message for target processors capable of immediately generating a Program Trace - Synchronization Message. The program counter (PC) value included is the value of the PC at the time of the watchpoint match.

NOTE

The Direct Branch with Sync Message, Indirect Branch with Sync Message, and/or Indirect Branch History with Sync Message may be implemented in lieu of the Synchronization Message in order to keep temporal ordering of Program Trace Messages.

Table 5-15—Program Trace - Synchronization Message Format

Program Trace - Synchronization Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	PC	Variable	The full current instruction address. MSBs that have a value of 0 may be truncated.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	DCONT	Vendor-fixed	Indicates whether synchronization is a result of a discontinuity in the program flow relative to the prior Program Trace Message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 9

5.3.12 Program Trace - Correction Message

Message Description: The Program Trace - Correction Message is output by the target processor when it determines after a Program Trace Message has been sent, that the value in the *number of instruction units executed* packet is incorrect.

Note that if the ADJUST packet = 1, the last taken branch was actually cancelled. Consequently, in the next I-CNT packet transmitted, the taken branch that was cancelled will be counted as part of the sequential instruction units.

Table 5-16—Program Trace - Correction Message Format

Program Trace - Correction Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	ADJUST	Variable	A number correcting the number of instruction units executed since the last taken branch. This number (unsigned) should be subtracted by the tool from the last I-CNT packet transmitted.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 10

5.3.13 Program Trace - Repeat Branch Message

Message Description: When a series of instructions are executed instead of a single instruction (hardware loop), the Program Trace - Repeat Branch Message can be transmitted to indicate how many times a branch repeated. A branch is determined to be repeated when the I-CNT value matches the previous message. The original branch message is only transmitted once, followed by the Program Trace - Repeat Branch Message.

Message Notes:

- When using traditional Program Trace Messages, the Program Trace - Repeat Branch Message can be used for hardware loops or for normal direct branches where the I-CNT value matches the previous direct branch's I-CNT value. In the hardware loop case, both the I-CNT value and target address will match the previous branch. The external development tool will need to distinguish the two cases.

- When using Branch History Messages, the Program Trace - Repeat Branch Message is not necessary for direct branches. The direct branch information is recorded in the history buffer and the Indirect Branch with History Message is transmitted as necessary. Only a single message is generated.

- When Branch History Messages are used and the repeated branch is an indirect branch, the Indirect Branch with History Message can be executed repeatedly, or a Repeat Branch Message can be generated when the branch qualifies (I-CNT and target address match) and the history buffer is empty.

Table 5-17—Program Trace - Repeat Branch Message Format

Program Trace - Repeat Branch Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	B-CNT	Variable	Repeat Branch Count. The number of times the branch was repeated.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 30

5.3.14 Program Trace - Repeat Instruction Message

Message Description: A repeat instruction is a single instruction that executes a multiple number of times. Since the number of times the instruction is executed is a run-time variable, it is unknown to the debug/development tool how many times the instruction will be repeated. The Program Trace - Repeat Instruction Message is used to trace repeated instructions whose repetition count is a run-time variable.

Message Notes:

- For predicated repeat instructions, no bit should be added to the branch/ predicate history. Instead, the true predicate generates a message, and the false predicate is added to the sequential instruction count.
- The repeat instruction count (R-CNT) value is the number of times the instruction is repeated, which is one less than the total number of times the instruction was executed. An R-CNT value of 0 shall be interpreted as 2^N repetitions where N is the maximum R-CNT field size for that target.

Table 5-18—Program Trace - Repeat Instruction Message Format

Program Trace - Repeat Instruction Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	HIST	Variable	Branch/Predicate instruction history. This packet is terminated by a stop bit set to 1 after the last history bit. This ending allows the tool to determine which bits are part of the history field and which are padded zeros.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
1	R-CNT	Variable	Repeat instruction count. The number of times the instruction was repeated.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 31

5.3.15 Program Trace - Repeat Instruction with Sync Message

Message Description: For target processors that do not implement traditional Branch History Messages, the Program Trace - Repeat Instruction with Sync Message is output when any of the following conditions occurs:

1. Upon exit from system reset to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool. This trace message follows exit from reset for target processors not capable of immediately generating a Program Trace - Synchronization Message.

2. Upon execution of a repeated instruction after program trace is enabled during normal execution of the embedded processor.

3. Upon exit from a power-down state to allow the *number of instruction units executed* packet in a subsequent Program Trace Message to be correctly interpreted by the tool.

4. Upon execution of a repeated instruction following the processor's exit from debug mode.

5. Upon execution of a repeated instruction after an overrun condition had previously occurred in which one or more branch trace occurrences were discarded by the target processor's debug logic. To inform the tool that an overrun condition occurred, the target outputs an Error Message (TCODE = 8) with an ECODE value of 00001 or 00111 immediately prior to the Program Trace - Indirect Branch History with Sync Message.

6. Upon execution of a repeated instruction after the periodic Program Trace Message counter has expired, indicating that 255 *non-sync* Program Trace Messages have been sent since the last *sync* Program Trace Message. The value of 255 is a maximum number; target processors may use a smaller value.

7. Upon execution of a repeated instruction when a debug control register field specifies that EVTI pin action is to generate program trace synchronization and the EVTI pin has been asserted. This message is output by target processors not capable of immediately generating a Program Trace - Synchronization Message.

8. Upon overflow of the sequential instruction unit counter. Because a limited counter size must be implemented in the embedded processor, there will likely be sequential instruction sequences (with no taken branches), which will cause the counter to overflow.

9. (Optional) Upon the occurrence of a watchpoint match and the next repeated instruction. This trace message follows the Watchpoint Match

Message for target processors not capable of immediately generating a Program Trace - Synchronization Message.

Table 5-19—Program Trace - Repeat Instruction with Sync Message Format

Program Trace - Repeat Instruction with Sync Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	HIST	Variable	Branch/Predicate instruction history. This packet is terminated by a stop bit set to 1 after the last history bit. This ending allows the tool to determine which bits are part of the history field and which are padded zeros.
1	F-ADDR	Variable	The full target address of the repeated instruction. MSBs that have a value of 0 may be truncated.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
1	R-CNT	Variable	Repeat Instruction Count. The number of times the instruction was repeated.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 32

5.3.16 Program Trace - Correlation Message

Message Description: Program Trace - Correlation Messages are used to correlate events to the program flow that may not be associated with the instruction stream (i.e., Data Trace Messages). The occurrence of a Nexus-defined or vendor-defined event will cause this message to be transmitted.

Message Notes:

- In cases where this message is sent due to events that may disable program trace (i.e., low-power mode, debug mode, program trace disabled), the I-CNT value may be cleared after sending the Program Trace - Correlation Message.
- In cases where this message is sent to correlate events that do not necessarily affect the program flow (i.e., data read or write), the I-CNT value should not be cleared upon sending the Program Trace - Correlation Message.
- For targets that incorporate multiple ECODEs, the CDATA field must be implemented consistently. If a Program Trace - Correlation Message due to one event requires a CDATA value, all Program Trace - Correlation Messages for that target must incorporate the CDATA field for other events as well.

Table 5-20—Program Trace - Correlation Message Format

Program Trace - Correlation Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
0	CDATA	Vendor-variable	This packet is a vendor-defined field. It can represent a value used in correlating an event with the program flow (i.e., branch history).
1	I-CNT	Variable	Number of instruction units executed since the last taken branch.
0	EVCODE	Vendor-fixed	Event Code. Refer to **Table 5-21**.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 33

Table 5-21—Recommended Event Code (EVCODE) Description

Event Code (EVCODE)	Event Description
0b0000	Entry into debug mode
0b0001	Entry into low-power mode
0b0010	Data Trace - Write
0b0011	Data Trace - Read
0b0100	Program Trace Disabled
0b0101-0b0111	Reserved for future functionality
0b1000-0b1111	Vendor-defined

5.3.17 Data Trace - Data Write/Read Messages

Message Description: Data Trace - Data Write/Read Messages are output by the target processor when it detects a memory write/read that matches the debug logic's data trace attributes.

Table 5-22—Data Trace - Data Write/Read Message Formats

Data Trace - Data Write/Read Messages			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	DATA	Variable	The data value written/read. The length of this packet must be equal to the size of the data write/read if the DSZ field is omitted. MSBs that have a value of 0 may be truncated if the DSZ field is included.
1	U-ADDR	Variable	The unique portion of the data write/read address, which is relative to the previous Data Trace Message (read or write).
0	DCORR	Vendor-fixed	Field used for correlating the Data Trace Message to the program flow (e.g., I-CNT value or instruction address). For target processors that do not need to correlate data threads to instruction threads or that implement this feature within Program Correlation Messages, this field may be omitted.

Table 5-22—Data Trace - Data Write/Read Message Formats *(Continued)*

Data Trace - Data Write/Read Messages			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	DSZ	Vendor-fixed	Indication of the size of the write/read. The width and values for this field are vendor defined. For targets in which the size can be determined from the size of the DATA packet, this field can be omitted. See **Table 5-24** for recommended size encodings for targets that implement this field.
0	MAP	Vendor-fixed	A number to indicate the memory map currently in use by the target processor. For targets with only a single memory map, this packet can be omitted.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 5 (write); Value = 6 (read)

5.3.18 Data Trace - Data Write/Read with Sync Messages

Message Description: Data Trace - Data Write/Read with Sync Messages are an alternative to the Data Trace - Data Write/Read Messages. They are output instead of a Data Trace - Data Write/Read Message when a memory write (read) occurs that matches the debug logic's data trace attributes and when one of the following conditions has occurred:

1. Upon exit from reset to allow the *unique portion of the data write (read) address* of following Data Trace - Data Write/Read Messages to be correctly interpreted by the tool.
2. When data trace is enabled during normal execution of the embedded processor.
3. Upon exit from a power-down state to allow the *unique portion of the data write (read) address* of following Data Trace - Data Write/Read Messages to be correctly interpreted by the tool.
4. After the $\overline{\text{EVTI}}$ pin has been asserted when a debug control register field specifies that $\overline{\text{EVTI}}$ pin action is to generate data trace synchronization.
5. After an overrun condition had previously occurred in which one or more data trace occurrences were discarded by the target processor's debug

logic. To inform the tool that an overrun condition occurred, the target outputs an Error Message (TCODE = 8) with an ECODE value of 00010 or 00111 immediately prior to the Data Trace - Data Write with Sync Message.

6. Upon expiration of the periodic Data Trace Message counter, indicating that 255 *non-sync* Data Trace Messages have been sent since the last *sync* Data Trace Message. The value of 255 is a maximum number; target processors may use a smaller value.

7. Upon detection of a data write/read following the processor's exit from debug mode.

Table 5-23—Data Trace - Data Write/Read with Sync Message Formats

Data Trace - Data Write/Read with Sync Messages			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	DATA	Variable	The data value written/read. The length of this packet must be equal to the size of the data write/read if the DSZ field is omitted. MSBs that have a value of 0 may be truncated if the DSZ field is included.
1	F-ADDR	Variable	The full address of the memory location written/read. MSBs that have a value of 0 may be truncated.
0	CANCEL	Vendor-variable	Number of previous Data Trace Messages that should be ignored by the tool. This packet is generated only by processors performing speculative execution where a trace message may be output before it is known whether the data write actually occurred.
0	DCORR	Vendor-fixed	Field used for correlating the Data Trace Message to the program flow (e.g., I-CNT value or instruction address). For target processors that do not need to correlate data threads to address threads or that implement this correlation within Program Correlation Messages, this field may be omitted.

Table 5-23—Data Trace - Data Write/Read with Sync Message Formats *(Continued)*

Data Trace - Data Write/Read with Sync Messages			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	DSZ	Vendor-fixed	Indication of the size of the write/read. The width and values for this field are vendor defined. For targets in which the size can be determined from the size of the DATA packet, this field can be omitted. See **Table 5-24** for recommended data size encodings for targets that implement this field.
0	MAP	Vendor-fixed	A number to indicate the memory map currently in use by the target processor. For targets with only a single memory map, this packet can be omitted.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 13 (write); Value = 14 (read)

Table 5-24—Recommended Data Size Encodings

DSZ Encoding	Data Size
000	8-bit
001	16-bit
010	32-bit
011	64-bit
100-101	Vendor-defined
110-111	Reserved for future sizes

NOTE

A target/tool does not need to support all of the data sizes in **Table 5-24**.

5.3.19 Data Acquisition Message

Message Description: The Data Acquisition Message is sent by a target when the target processor writes the value of 0x0 to the Data Acquisition Control Register.

Table 5-25—Data Acquisition Message Format

Data Acquisition Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
1	DQDATA	Variable	One or more packets of data values.
1	IDTAG	Vendor-fixed	Data ID tag, specifying which group of data is included in the Data Acquisition Message.
6	TCODE	Fixed	Value = 7

5.3.20 Error Message

Message Description: An Error Message provides an indication of which client (if more than one was present on the device) generated an error and what type of error was generated. **Table 5-27** below lists the types of errors which can occur as well as their encoded values.

For any of the trace overrun categories, an Error Message, containing an overrun error code, is to inform the tool that the target has discarded trace occurrences because of insufficient space in its trace output queue. The Error Message is sent immediately prior to a synchronization message (e.g., OTM, BTM with synchronization, or DTM with synchronization) as soon as space is available in the trace output queue.

Table 5-26—Error Message Format

Error Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
5	ECODE	Fixed	Error code. Refer to **Table 5-27**.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 8

Table 5-27—Recommended Error Codes

Error Code	Description
00000	Ownership trace overrun.
00001	Program trace overrun.
00010	Data trace overrun.
00011	Read/write access error (read or write error to user memory map). This error code applies only to targets that support the Read/Write Access Messages for NRRs (**Appendix B - Recommendations for Access to Control and Status Registers**).[a]
00100	Invalid message (message type not implemented). The Error Message is sent by the target as soon as the invalid message is detected.
00101	Invalid access opcode (NRR not implemented). This error code applies only to targets that support the Read/Write Access Messages for NRRs (**Appendix B - Recommendations for Access to Control and Status Registers**). The Error Message is sent by the target as soon as the invalid opcode is detected.
00110	Watchpoint overrun.
00111	Program and/or data and/or ownership trace overrun.
01000	Program trace and/or data trace and/or ownership trace and/or watchpoint overrun.
01001–10111	Reserved.
11000–11111	Vendor defined.

a. For targets that implement vendor-defined debug control and status registers and use Public Messages 22–26 to provide read/write access, an error condition is indicated by the Status (ST) field in the Target/Tool Response Message.

5.3.21 Watchpoint Match Message

Message Description: The Watchpoint Match Message is sent by the target whenever a watchpoint match occurs. Multiple watchpoint matches can be indicated in the same message. The debug logic in the target must ensure that Watchpoint Match Messages can never be cancelled once they have been generated. If watchpoint match occurrences are discarded because of insufficient space in the trace output queue, the target must send the tool an Error Message prior to the next Watchpoint Match Message actually sent so that the tool knows that one or more watchpoint match occurrences were discarded.

Table 5-28—Watchpoint Match Message Format

Watchpoint Match Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
0	TSTAMP	Vendor-variable	Number of cycles message was held in the buffer or the full timestamp value. For targets that do not implement timestamping (or use pins for timestamping), this field may be omitted. Refer to **4.11.2 - Timestamping via AUX**.
2	WPHIT	Vendor-fixed	Each bit position in this *N*-bit field corresponds to a different watchpoint number. Bit positions 0 through *N* correspond to watchpoints 0 through *N*. A “1” in a bit position indicates a watchpoint match occurred.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this packet can be omitted.
6	TCODE	Fixed	Value = 15

5.3.22 Port Replacement - Output Message

Message Description: The Port Replacement - Output Message is sent by the target to set up external port replacement logic on the target system. For LSIO port bits defined as outputs, this message is also used to set the state of the pins.

Table 5-29—Port Replacement - Output Message Format

Port Replacement - Output Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
16	OUT	Fixed	Each bit corresponds to one of the 16 LSIO pins involved with port replacement. When the direction of the pin is an output, the pin state corresponds to the value of the bit in this packet. Pins defined as inputs are unaffected by corresponding bits in this packet.
16	DIR	Fixed	Each bit specifies the direction of one of the 16 LSIO pins involved with port replacement. 0 = input 1 = output
6	TCODE	Fixed	Value = 20

5.3.23 Port Replacement - Input Message

Message Description: The Port Replacement - Input Message is sent by the tool upon the occurrence of a change in the state of one or more input pins.

Table 5-30—Port Replacement - Input Message Format

Port Replacement - Input Message			**Direction: from tool**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
16	IN	Fixed	Each bit corresponds to one of the 16 LSIO pins involved with port replacement. When the direction of the pin is an input, this message is used to read the pin state.
6	TCODE	Fixed	Value = 21

5.3.24 NRR Access - Target Ready Message

Message Description: For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the target sends the NNR Access - Target Ready Message when it has completed the actions that were specified in a previous Read/Write Access Message from the tool and is able to accept another command. The tool sends the NRR Access - Target Ready Message when it is able to accept another message from the target as part of a block read sequence.

NOTE

The NRR Access - Target Ready Message will not normally be used by target processors that implement vendor-defined registers or that access Nexus registers via the **IEEE 1149.1** port.

Table 5-31—NRR Access - Target Ready Message Format

NRR Access - Target Ready Message			**Direction: from tool, from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
6	TCODE	Fixed	Value = 16

5.3.25 NRR Access - Read Register Message

Message Description: For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the tool sends the NRR Access - Read Register Message when it wants to read control or status information from the target. The target responds to the Read Register Message with a Read/Write Response Message containing the requested information.

NOTE

The NRR Access - Read Register Message will not normally be used by target processors that implement vendor-defined registers or that access Nexus registers via the **IEEE 1149.1** port.

Table 5-32—NRR Access - Read Register Message Format

NRR Access - Read Register Message			**Direction: from tool**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
8	OPCODE	Fixed	Refer to **Appendix B - Recommendations for Access to Control and Status Registers** for a list of all opcodes that the tool can use to request the target to return specific status or data.
6	TCODE	Fixed	Value = 17

5.3.26 NRR Access - Write Register Message

Message Description: For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the tool uses the NRR Access - Write Register Message to control debug resources within the target and to send data to the target. When the target has processed this message and is able to accept another Read/Write Access Message from the tool, it responds with a Target Ready Message.

NOTE

The NRR Access - Write Register Message will not normally be used by target processors that implement vendor-defined registers or that access Nexus registers via the **IEEE 1149.1** port.

Table 5-33—NRR Access - Write Register Message Format

NRR Access - Write Register Message			**Direction: from tool**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
8	REGVAL	Variable	Depending on the opcode selected, the message may include one or more fixed-length packets of information downloaded to the target.
8	OPCODE	Fixed	Refer to **Appendix B - Recommendations for Access to Control and Status Registers** for a list of all opcodes that the tool can use to send debug control commands and data to the target.
6	TCODE	Fixed	Value = 18

5.3.27 NRR Access - Read/Write Response Message

Message Description: For target processors that implement the NRRs described in **Appendix B - Recommendations for Access to Control and Status Registers**, the NRR Access - Read/Write Response Message is sent in the following circumstances:

- By a target in response to a Read Register Message issued by the tool. The information packets included in the message depend on the opcode in the original Read Register Message.
- By a target in response to a Write Register Message (with RWA opcode, RW field = Read) from the tool. The message contains a single data word of the size specified in the Write Register Message.
- By a target in response to each Target Ready Message from the tool that follows a block read command issued by the tool. The message contains a single data word of the size specified in the Write Register Message. The target continues to send messages until all data originally specified by the access count (CNT) field have been transferred. The tool can prematurely terminate the data transfer sequence by responding to a message with a Write Register Message (with RWA opcode, RW field = Read, SC = 0) instead of with a Target Ready Message.
- By a tool in response to each Target Ready Message issued by the target that follows a block write command issued by the tool. The message contains a single data word of the size specified in the Write Register Message. The tool continues to send messages until all data originally specified by the access count (CNT) field have been transferred. The tool can prematurely terminate the data transfer sequence by sending a Write Register Message (with RWA opcode, RW field = Write, SC = 0) instead of with another Read/Write Response Message.

NOTE

The NRR Access - Read/Write Response Message will not normally be used by target processors that implement vendor-defined registers or that access Nexus registers via the **IEEE 1149.1** port.

Table 5-34—NRR Access - Read/Write Response Message Format

NRR Access - Read/Write Response Message			**Direction: from tool, from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
8	REGVAL	Variable	Depending on the opcode selected, the message includes one or more fixed-length packets of information.
6	TCODE	Fixed	Value = 19

5.3.28 Memory Access - Read Target/Tool Message

Message Description: For embedded processors that implement vendor-defined development registers (not NRRs) or MSM, the Memory Access - Read Target/ Tool Message provides a mechanism for reading memory locations within the target or the tool.

Message Notes:

- The tool sends the Memory Access - Read Target/Tool Message when it wants to read a location in the target's memory-mapped address space.
- The target sends the Memory Access - Read Target/Tool Message when it wants to read a location in the tool's memory space.

Table 5-35—Memory Access - Read Target/Tool Message Format

Memory Access - Read Target/Tool Message			**Direction: from tool, from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
1	ADDRESS	Variable	Vendor-defined field specifying the location to be read. The size of the address must match the address space supported by the target or tool.
3	DSZ	Vendor-fixed	See **Table 5-24** for recommended data size encodings for targets that implement this field. Note: A target/tool does not need to support all of the data sizes listed in the table.
0	MAP	Vendor-fixed	A number to indicate the memory map to be used in the target/tool. For targets or tools with only a single memory map, this packet can be omitted.
6	TCODE	Fixed	Value = 22

5.3.29 Memory Access - Write Target/Tool Message

Message Description: For embedded processors that implement vendor-defined development registers (not NRRs) or MSM, the Memory Access - Write Target/ Tool Message provides a mechanism for writing memory locations within the target or the tool.

Message Notes:

- The tool sends the Memory Access - Write Target/Tool Message when it wants to write to a location in the target's memory-mapped address space
- The target sends the Memory Access - Write Target/Tool Message when it wants to write to a location in the tool's memory space.

Table 5-36—Memory Access - Write Target/Tool Message Format

Memory Access - Write Target/Tool Message			**Direction: from tool, from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
1	DATA	Variable	Write data value of size DSZ.
1	ADDRESS	Variable	Vendor-defined field specifying the location to be written. The size of the address must match the address space supported by the target or tool.
3	DSZ	Fixed	See **Table 5-24** for recommended data size encodings for targets that implement this field. Note: A target/tool does not need to support all of the data sizes listed in the table.
0	MAP	vendor-fixed	A number to indicate the memory map to be used in the target/tool. For targets or tools with only a single memory map, this packet can be omitted.
6	TCODE	Fixed	Value = 23

5.3.30 Memory Access - Read Next Target/Tool Data Message

Message Description: For embedded processors that implement vendor-defined development registers (not NRRs) or MSM, the Memory Access - Read Next Target/Tool Data Message provides a mechanism for reading consecutively addressed data (block reads) within the target or tool.

Message Notes:

- The tool sends the Memory Access - Read Next Target/Tool Data Message when it has processed a prior Memory Access - Target Response Message and the tool's receive buffer can accommodate more read data.
- The target sends the Memory Access - Read Next Target/Tool Data Message when it has processed a prior Memory Access - Tool Response Message and the target's receive buffer can accommodate more read data.
- There is no limit to the amount of data that can be transferred using consecutive read next target/tool data commands.
- Both tool and target are required to increment their internal address pointers according to the size of the data being transferred.

Table 5-37—Memory Access - Read Next Target/Tool Data Message Format

Memory Access - Read Next Target/Tool Data Message			**Direction: from tool, from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
6	TCODE	Fixed	Value = 24

5.3.31 Memory Access - Write Next Target/Tool Data Message

Message Description: For embedded processors that implement vendor-defined development registers (not NRRs) or MSM, the Memory Access - Write Next Target/Tool Data Message provides a mechanism for writing consecutively addressed data (block writes) within the target or tool.

Message Notes:

- The tool sends the Memory Access - Write Next Target/Tool Data Message to write to the next consecutively addressed location in the target after it has processed a prior Memory Access - Target Response Message and has more data available to send.
- The target sends the Memory Access - Write Next Target/Tool Message to write to the next consecutively addressed location within the tool after it has processed a prior Memory Access - Tool Response Message and has more data available to send.
- There is no limit to the amount of data that can be transferred using consecutive write next target/tool data commands.
- Both tool and target are required to increment their internal address pointers according to the size of the data being transferred.

Table 5-38—Memory Access - Write Next Target/Tool Data Message Format

Memory Access - Write Next Target/Tool Data Message			**Direction: from tool, from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
8	DATA	Variable	Write data value, the size of which is determined by the DSZ packet in the most recent Memory Access - Write Target/Tool Message.
6	TCODE	Fixed	Value = 25

5.3.32 Memory Access - Target Response Message

Message Description: For embedded processors that implement vendor-defined development registers (not NRRs), the Memory Access - Target Response Message is output by the target, but the contents differ depending on whether the most recent read/write command issued by the tool was contained in a Memory Access - Read Target Message or a Memory Access - Write Target Message.

Message Notes:

- For read commands, the target sends a Memory Access - Target Response Message (containing data) as soon as it has retrieved the data requested by a previous Memory Access - Read Target Message or Memory Access - Read Next Target Data Message from the tool.
- For write commands, the target sends a Memory Access - Target Response Message (with no data) as soon as the target's receive buffer is able to accept more data from the tool.
- If the target is unable to process the function requested by the previous Memory Access - Read Target Message, Memory Access - Write Target Message, Memory Access - Read Next Target Data Message, or Memory Access - Write Next Target Data Message, it sends a Memory Access - Target Response Message with the status (ST) field = 01.

Table 5-39—Memory Access - Target Response Message Format

Memory Access - Target Response Message			**Direction: from target**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
8	DATA	Variable	Read data, the size of which is determined by the DSZ packet in the most recent Memory Access - Read Target Message. This field does not exist if the previous message issued by the tool was a Memory Access - Write Target Message or if the target is unable to complete the previously requested read operation.
2	ST	Fixed	Status 00 = The previously requested read/write operation is able to be processed normally. 01 = The previously requested read/write operation cannot be completed. 1x = Reserved for future use.
6	TCODE	Fixed	Value = 26

5.3.33 Memory Access - Tool Response Message

Message Description: For embedded processors that implement vendor-defined development registers (not NRRs) or MSM, the Memory Access - Tool Response Message is output by the tool, but the contents differ depending on whether the most recent read/write command issued by the target was contained in a Memory Access - Read Tool Message or a Memory Access - Write Tool Message.

Message Notes:

- For read commands, the tool sends a Memory Access - Tool Response Message (containing data) as soon as it has retrieved the data requested by a previous Memory Access - Read Tool Message or Memory Access - Read Next Tool Data Message from the target.
- For write commands, the tool sends a Memory Access - Tool Response Message (with no data) as soon as the tool's receive buffer is able to accept more data from the target.
- If the tool is unable to process the function requested by the previous Memory Access - Read Tool Message, Memory Access - Write Tool Message, Memory Access - Read Next Tool Data Message, or Memory Access - Write Next Tool Data Message, it sends a Memory Access - Tool Response Message with the ST field = 01.
- To support MSM, in which code that is normally fetched from target memory is instead fetched from tool memory, the tool must be able to inform the target when to stop requesting data (i.e., the tool determines when the substitution process should end). The tool informs the target not to request more data by setting the ST field = 10 in the Memory Access - Tool Response Message that contains the final read data.

Table 5-40—Memory Access - Tool Response Message Format

Memory Access - Tool Response Message			**Direction: from tool**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
8	DATA	Variable	Read data, the size of which is determined by the DSZ packet in the most recent Memory Access - Read Tool Message. This field does not exist if the previous message issued by the target was a Memory Access - Write Tool Message or if the tool is unable to complete the previously requested read operation.

Table 5-40—Memory Access - Tool Response Message Format *(Continued)*

Memory Access - Tool Response Message			**Direction: from tool**
Minimum Packet Size (bits)	**Packet Name**	**Packet Type**	**Description**
2	ST	Fixed	Status 00 = The previously requested read/write operation is able to be processed normally. 01 = The previously requested read/write operation cannot be completed. 10 = Memory substitution transfer complete. 11 = Reserved for future use.
6	TCODE	Fixed	Value = 26

5.4 NRR Access Messages - Example Sequences

The examples in **Table 5-41** through **Table 5-44** use the NRRs concatenated as defined in **Section B.9 - NRRs Concatenated for Better Transfer Efficiency** via the AUX.

Table 5-41 outlines the sequence of messages when the development tool requests debug status information from the target.

Table 5-41—Target Debug Status Requested by Tool

Step #	**Description of Action**
1	Tool sends NRR Access - Read Register Message.
2	Target sends NRR Access - Read/Write Response Message.

Table 5-42 outlines the sequence of messages when the development tool sends a debug command to the target.

Table 5-42—Debug Command Sent by Tool to Target

Step #	**Description of Action**
1	Tool sends NRR Access - Write Register Message.
2	Target sends NRR Access - Target Ready Message.

Table 5-43 outlines the sequence of messages when the tool initiates a block write transfer.

Table 5-43—Block Write Command Issued by Tool

Step #	Description of Action
1	Tool sends NRR Access - Write Register Message (includes first write data, block write attributes, & RW = 1).
2	Target sends NRR Access - Target Ready Message.
3	Tool sends NRR Access - Read/Write Response Message (next write data).
4	Target sends NRR Access - Target Ready Message.
5	Tool sends NRR Access - Read/Write Response Message (next write data).
6	Target sends NRR Access - Target Ready Message.
7	Tool continues sequence until # words of data have been transferred.

Table 5-44 outlines the sequence of messages when the tool initiates a block read transfer.

Table 5-44—Block Read Command Issued by Tool

Step #	Description of Action
1	Tool sends NRR Access - Write Register Message (includes block read attributes, & RW = 0).
2	Target sends NRR Access - Read/Write Response Message (first read data).
3	Target sends NRR Access - Read/Write Response Message (next read data).
4	Target sends NRR Access - Read/Write Response Message (next read data).
5	Target continues sequence until # words of data have been transferred.

NOTE

The tool should be able to keep up with the transfer rate of the target for block reads. If the tool does not contain this capability, block reads should not be requested.

5.5 Memory Access Messages - Example Sequences

The following read/write sequences show how the AUX is used by a tool to access target memory space (Read/Write Access Messaging) and by a target to access tool memory space (MSM).

5.5.1 Tool Accessing Target

Table 5-45 through **Table 5-49** as well as **Figure 5-2** show the sequences of messages sent between a tool and a target when the tool wants to read or write memory-mapped address space in the target. To achieve maximum transfer performance, Read Next Target Data Messages or Write Next Target Data Messages should be used wherever possible because these messages have the shortest length.

Table 5-45—Reading Consecutively Addressed Target Locations

Step #	Description of Action
1	Tool sends Memory Access - Read Target Message.
2	Target sends Memory Access - Target Response Message (includes data).
3	Tool sends Memory Access - Read Next Target Data Message.
4	Target sends Memory Access - Target Response Message (includes data).
5	Tool sends Memory Access - Read Next Target Data Message.
6	Target sends Memory Access - Target Response Message (includes data).

Table 5-46—Writing Consecutively Addressed Target Locations

Step #	Description of Action
1	Tool sends Memory Access - Write Target Message.
2	Target sends Memory Access - Target Response Message (includes data).
3	Tool sends Memory Access - Write Next Target Data Message.
4	Target sends Memory Access - Target Response Message (includes data).
5	Tool sends Memory Access - Write Next Target Data Message.
6	Target sends Memory Access - Target Response Message (includes data).

Table 5-47—Reading Randomly Addressed Target Locations

Step #	Description of Action
1	Tool sends Memory Access - Read Target Message.
2	Target sends Memory Access - Target Response Message (includes data).
3	Tool sends Memory Access - Read Target Message.
4	Target sends Memory Access - Target Response Message (includes data).
5	Tool sends Memory Access - Read Target Message.
6	Target sends Memory Access - Target Response Message (includes data).

Table 5-48—Writing Randomly Addressed Target Locations

Step #	Description of Action
1	Tool sends Memory Access - Write Target Message (includes data).
2	Target sends Memory Access - Target Response Message.
3	Tool sends Memory Access - Write Target Message (includes data).
4	Target sends Memory Access - Target Response Message.
5	Tool sends Memory Access - Write Target Message (includes data).
6	Target sends Memory Access - Target Response Message.

Table 5-49—Intermixed Reading/Writing Randomly Addressed Target Locations

Step #	Description of Action
1	Tool sends Memory Access - Read Target Message.
2	Target sends Memory Access - Target Response Message (includes data).
3	Tool sends Memory Access - Write Target Message (includes data).
4	Target sends Memory Access - Target Response Message.
5	Tool sends Memory Access - Write Target Message (includes data).
6	Target sends Memory Access - Target Response Message.
7	Tool sends Memory Access - Read Target Message.
8	Target sends Memory Access - Target Response Message (includes data).

Because trace messages do not need to be acknowledged by the tool, these messages can be output when the AUX OUT is not transmitting other messages. **Figure 5-2** shows how the read/write protocol and trace messages can co-exist. This figure also demonstrates that the AUX is full-duplex, that is, messages can occur in both directions simultaneously.

NOTE

Protocol responses from the target should not pass through the same output queue as trace messages. As soon as a protocol response has been prepared by the target, it must be transmitted immediately following any trace message currently being transmitted, regardless of the number of other trace messages queued for output.

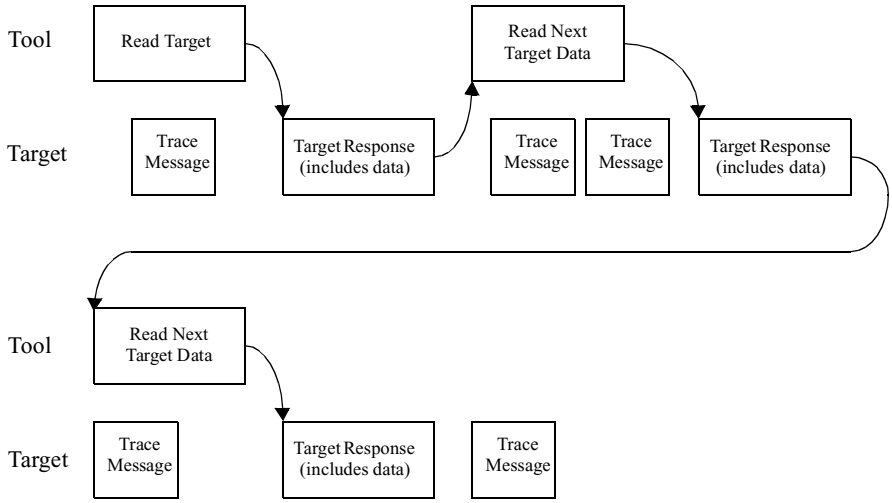


Figure 5-2—Trace Messages Intermixed with Read Target Locations

5.5.2 Target Accessing Tool

A target uses the same Memory Access Messages and sequences as described in **5.5.1 - Tool Accessing Target** to access memory space within a tool, except that all messages occur in the reverse direction. To understand the protocol, simply swap the Tool and Target names on all the protocol sequences in **Table 5-45** through **Table 5-49**.

Memory Access - Read Tool Messages and Memory Access - Write Tool Messages include an optional Map packet for use in situations where multiple memory maps are supported by the tool. These alternative memory maps may be used to select different address spaces within the tool, such as

- Target boot image
- Debug exception handler
- Read/write data space

To support MSM, in which code that is normally fetched from target memory is instead fetched from tool memory, the tool must be able to inform the target when to stop requesting data, i.e., the tool determines when the substitution process should end. Memory substitution will typically be initiated by a target watchpoint match and will be terminated by the tool. The tool informs the target not to request more data by setting the ST field = 10 in the response message that contains the final read data.

Read/Write Access Messaging can occur in both directions simultaneously. For example, a tool may initiate read/write access to the target without affecting any target-to-tool transfer of data currently in progress. In other words, both the tool and the target must have sufficient receive buffer space to support two messages, a request from the other device and the response to a request.

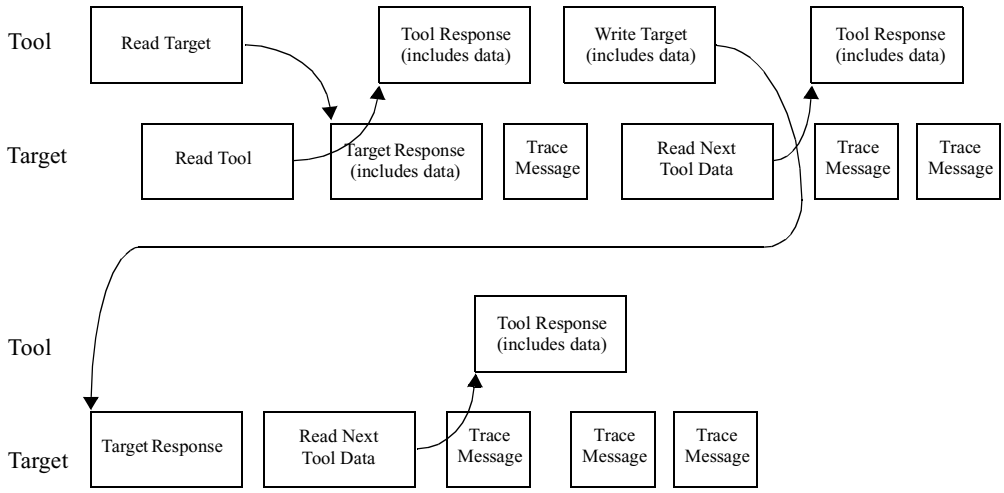


Figure 5-3—Simultaneous Read/Write Accesses by Tool and Target

5.5.3 Termination of Tool/Target Messaging

When large blocks of data are being read, the tool requests the next data word (when its buffer is able to accept more data) by issuing a Read Next Target Data Message. For block writes, the tool sends the next data word after it receives acknowledgment from the target that the target is ready to accept more data.

The tool controls the transfer process. The target has no prior knowledge of the amount of data to be transferred; the tool stops the process by not sending any more Memory Access - Read Next Target Data Messages or Memory Access - Write Next Target Data Messages. The target simply increments an address counter each time it receives a Memory Access - Read Next Target Data Message or Memory Access - Write Next Target Data Message. This address counter is automatically changed to a new value whenever another Memory Access - Read Target Message or Memory Access - Write Target Message is received.

During the transfer of a large block of data using Memory Access - Read Next Target Data Message or Memory Access - Write Next Target Data Message, the target may determine that it is unable to continue supplying read data or accepting write data. This could happen if the incrementing address points to nonexistent memory or to a protected memory area. Upon detecting such a condition, the

target issues a Memory Access - Target Response Message with the ST field = 0b01 (meaning that the previously requested read/write operation cannot be completed). The tool stops sending any more Memory Access - Read Next Target Data Messages or Memory Access - Write Next Target Data Messages and may then perform some recovery or error notification tasks.

SECTION 6
Nexus Port Signals

Embedded processors complying with Class 2, 3, or 4 shall provide the appropriate pin functions as shown in **Table 6-1**. Required and optional pin functions are designated by “R” and “O” respectively. Pins not allowed for an interface are shaded.

Table 6-1—Nexus Pin Functions Required per Interface Type

Pin Type	Direction	Full-duplex (AUX IN/AUX OUT)	Full-duplex w/IEEE 1149.1	Half-duplex w/IEEE 1149.1
MCKI	In	R		
$\overline{\text{RSTI}}$	In	R		
MDI	In	R		
$\overline{\text{MSEI}}$	In	R		
MCKO	Out	R	R	
MDO	Out	R	R	
$\overline{\text{MSEO}}$	Out	R	R	
$\overline{\text{EVTI}}$	In	R	O	O
$\overline{\text{EVTO}}$	Out	O	O	O
TCK	In		R	R
TDI	In		R	R
TDO	Out		R	R
TMS	In		R	R
$\overline{\text{TRST}}$	In		O	O
$\overline{\text{RDY}}$	Out		O	O

NOTE

The MCKO functions may be provided via a system CLOCKOUT pin on the embedded processor.

NOTE

For **IEEE 1149.1** implementations (Full-Duplex or Half-Duplex), the $\overline{\text{EVTI}}$ pin is optional. However, tool-initiated message synchronization and breakpoint generation functionality defined by the Nexus standard are lost if the pin is not implemented.

6.1 IEEE 1149.1 Pin Functions

The **IEEE 1149.1** pins are described in **Table 6-2**.

Table 6-2—IEEE 1149.1 Pins

IEEE 1149.1 Pin	Pin Description
TDI	Test Data Input (TDI) is an input pin that provides for serial movement of data into the **IEEE 1149.1** port.
TDO	Test Data Output (TDO) is an output pin that provides for serial movement of data out of the **IEEE 1149.1** port. All target accesses initiated via the **IEEE 1149.1** port should be transmitted by the target via TDO (not via AUX OUT).
TCK	Test Clock (TCK) is an input pin that provides the clock for the **IEEE 1149.1** port.
TMS	Test Mode Select (TMS) is an input pin that provides access to the **IEEE 1149.1** TAP state machine.
$\overline{\text{TRST}}$	Test Reset ($\overline{\text{TRST}}$) is an optional pin that provides for asynchronous initialization of the **IEEE 1149.1** controller.
$\overline{\text{RDY}}$	Ready ($\overline{\text{RDY}}$) is an optional output pin used to accelerate data accesses through the **IEEE 1149.1** port (Refer to **8.1.2 - Optional Ready (RDY) Output Pin**).

6.2 Nexus Auxiliary Pin Functions

The auxiliary pin functions are described in **Table 6-3**.

Table 6-3—Auxiliary Pins

Auxiliary Pins	Description of Auxiliary Pins
MCKO	Message Clockout (MCKO) is a free-running output clock to development tools for timing MDO and MSEO pin functions. MCKO can be independent of the embedded processor's system clock (CLOCKOUT). An embedded processor's CLOCKOUT pin may be used as a functional equivalent for MCKO.
MDO[M:0]	Message Data Out (MDO[M:0]) are output pin(s) used for OTM, BTM, DTM, reads, memory substitution accesses, etc. External latching of MDO shall occur on the rising edge of MCKO (or system clock). Depending upon bandwidth requirements, one, two, four, eight, or more pins may be implemented.
MSEO[1:0]	Message Start/End Out (MSEO [1:0]) are output pins that indicate when a message on the MDO pins has started, when a variable-length packet has ended, and when the message has ended. Only one MSEO pin is required, but up to two pins may be implemented for more efficient transfers. External latching of MSEO shall occur on the rising edge of MCKO (or system clock).
MCKI	Message Clockin (MCKI) is a free-running input clock from development tools for timing MDI and MSEI pin functions. MCKI can be independent of the embedded processor's system clock.
MDI[N:0]	Message Data In (MDI[N:0]) are input pin(s) used for downloading configuration information, writes to user resources, etc. Internal latching of MDI shall occur on the rising edge of MCKI. Depending upon bandwidth requirements, one, two, four, eight, or more pins may be implemented.
MSEI[1:0]	Message Start/End In (MSEI [1:0]) are input pins that indicate when a message on the MDI pins has started, when a variable-length packet has ended, and when the message has ended. Only one MSEI pin is required, but up to two pins may be implemented for more efficient transfers. Internal latching of MSEI shall occur on the rising edge of MCKI.
EVTI	Event In (EVTI) is an input pin where, when a high-to-low transition occurs, a processor is halted (breakpoint) or Program/Data Messages with synchronization are transmitted from the embedded processor.
RSTI	Reset In (RSTI) is a pin for resetting the Nexus port resources.
EVTO	Event Out (EVTO) is an optional output pin to development tools indicating exact timing for a single breakpoint status indication. Upon a breakpoint occurrence of the programmed breakpoint source, EVTO is asserted for a minimum of one clock period of MCKO.

6.3 Sample Port Implementations

For a full-duplex AUX with **IEEE 1149.1** pins, a minimum of two auxiliary pins are required for compliance [Message Data Out (MDO) and Message Start/End Out (MSEO)], assuming a system clockout pin can be used for MCKO. EVTI is also recommended for tool-initiated synchronization. The performance classification, however, would also be minimal and may meet the transfer bandwidth requirements for only low-end applications or lower compliance classifications.

The Nexus standard allows for additional transfer bandwidth with a scalable pin interface or transfer rate, as illustrated by the examples in **Table 6-4**.

Table 6-4—Example of AUX OUTs

Number of Pins for Each Example					Comments
MDO	**$\overline{\text{MSEO}}$**	**MCKO**	**$\overline{\text{EVTI}}$**	**Total Pins**	
1	1	0	0	2	Base implementation (**IEEE 1149.1** and Auxiliary)
2	1			3	2X faster than base implementation
4	1			5	4X faster than base implementation
4	2			6	1 clock faster per transfer
8	2			10	> 8X faster than base implementation
1	1	1		3	Independent clock allows for faster or slower transfer rate than with system clock reference.
2	1			4	
4	1			6	
4	2			7	
8	2			11	
1	1	0	1	3	Base implementation (Auxiliary only)
2	1			4	2X faster than base implementation
4	1			6	4X faster than base implementation
4	2			7	1 clock faster per transfer
8	2			11	> 8X faster than base implementation
1	1	1		4	Independent clock allows for faster or slower transfer rate than with system clock reference.
2	1			5	
4	1			7	
4	2			8	
8	2			12	

SECTION 7
AUX Message Protocol

The protocol for the embedded processor to receive and transmit messages via the auxiliary pins shall be accomplished with the Message Start/End In (MSEI) and MSEO pin functions, respectively. A minimum of one and a maximum of two MSEI pins shall provide the protocol for the embedded processor to receive messages, and a minimum of one and a maximum of two MSEO pins shall provide the protocol for the embedded processor to transmit messages.

The $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ protocol comprises the following:

- Two “1”s followed by one “0” to indicate the start of a message.
- “0” followed by two or more “1”s to indicate the end of a message.
- “0” followed by “1” followed by a “0” to indicate the end of a variable-length packet.
- “0”s at all other clocks during transmission of a message
- “1”s at all clocks during no message transmission (idle)

The same sequence is followed when using one or two $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ pins. However, when using two $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ pins, it is possible for two sequences to occur on the same clock.

$\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ is used to signal the end of variable-length packets but not vendor-fixed or fixed-length packets. $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ are sampled on the rising edge of Message Clockin (MCKI)/MCKO.

Table 7-1—$\overline{\text{MSEO}}$ Pin(s) Protocol

$\overline{\text{MSEO}}$ Function	Single $\overline{\text{MSEO}}$ data (serial)	Dual $\overline{\text{MSEO}}$ data
Start of message	1-1-0	11-00
End of message	0-1-1-(more 1s)	00 (or 01)-11-(more 11s)
End of variable-length packet	0-1-0	00-01
Message transmission	0s	00s
Idle (no message)	1s	11s

Figure 7-1 illustrates the state diagram for one-pin $\overline{\text{MSEI/MSEO}}$ transfers. When using only one $\overline{\text{MSEI/MSEO}}$ pin, the "End Message" state does not contain valid data on the Message Data In (MDI)/MDO pins. Also, it is not possible to have two consecutive "End Packet" Messages. This implies that the minimum packet size for a variable-length packet is two times the number of MDI/MDO pins. This ensures that a false end of message state is not entered by transmitting two consecutive "1"s on the $\overline{\text{MSEI/MSEO}}$ pin before the actual end of message.

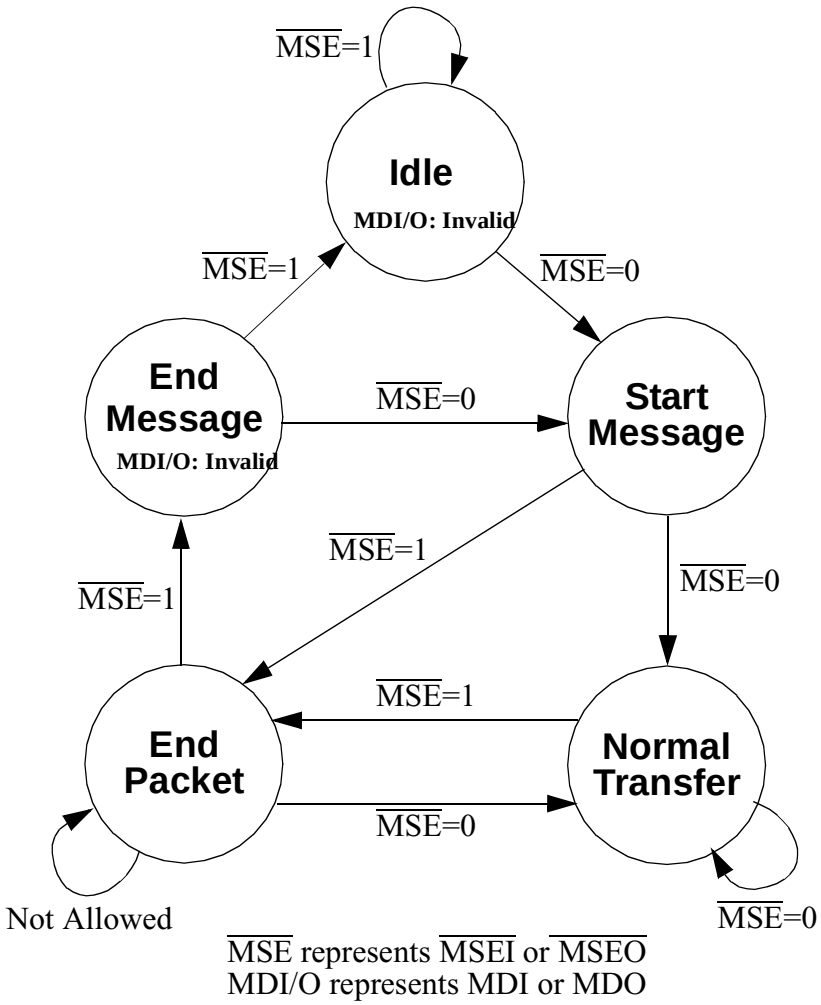
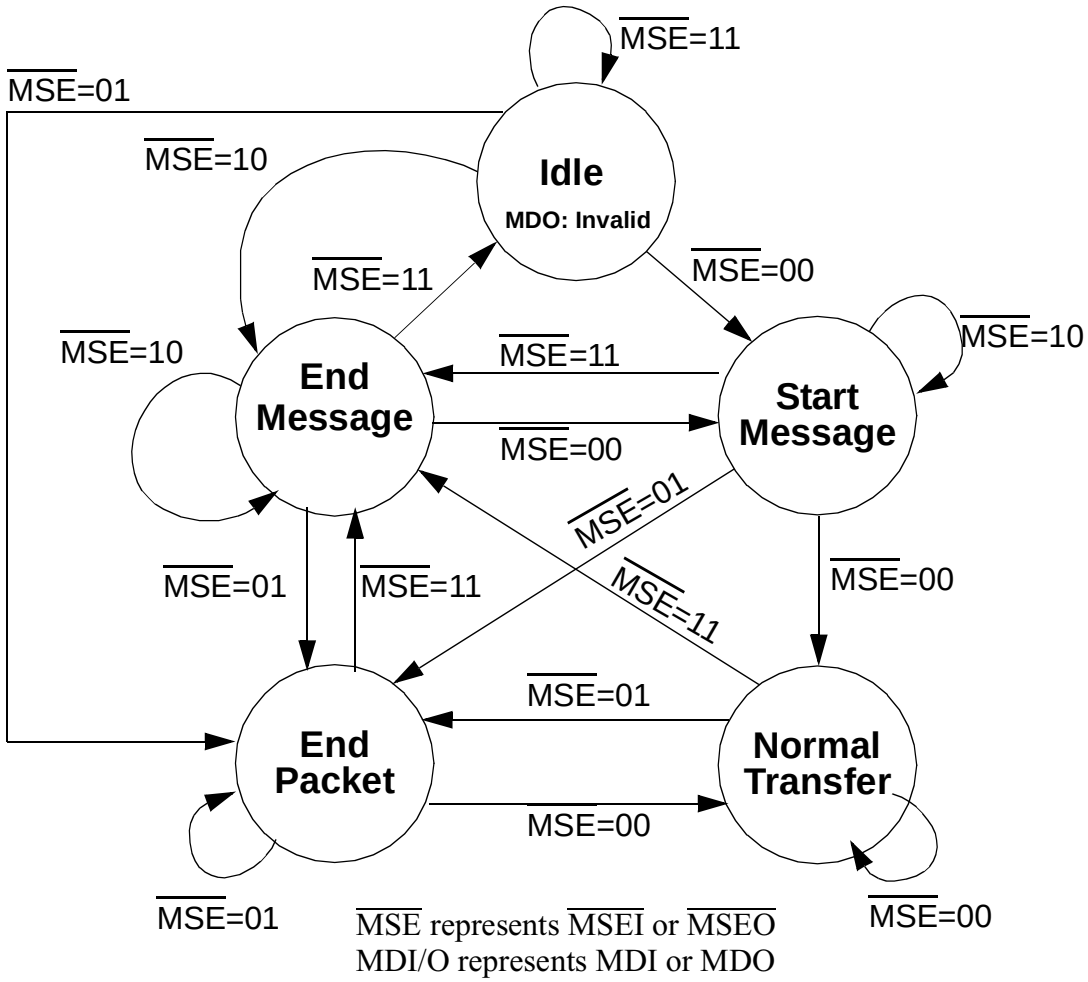


Figure 7-1—One-pin $\overline{\text{MSEI/MSEO}}$ Transfers

Figure 7-2 illustrates the use of two-pin $\overline{\text{MSEO}}$ transfers. The two-pin $\overline{\text{MSEI}}$/ $\overline{\text{MSEO}}$ option is more efficient than the single pin option. Termination of the current message may immediately be followed by the start of the next message on the consecutive clock. An extra clock to end the message is not necessary as with the one-pin $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ option. The two-pin option also allows for consecutive "End Packet" states. This can be an advantage when small, variable-sized packets are transferred.



Notes:
1—The variable port size for MDO and $\overline{\text{MSE}}$ allows for increased transfer rates per clock.
2—The one-pin $\overline{\text{MSE}}$ option should be selected when pin count is the most critical factor in the system and performance is not a priority.
3—The two-pin $\overline{\text{MSE}}$ option should be chosen when performance is the top priority and pin count is secondary.

Figure 7-2—Two-Pin $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ Transfers

NOTE

The “End Message” state may indicate the end of a variable-length packet as well as the end of the message when using the two-pin option.

Figure 7-3 illustrates the transfer protocol for the Indirect Branch Message. For purposes of illustration only, one MDO pin and one MSEO pin are shown. MDO and MSEO are sampled on the rising edge of MCKO.

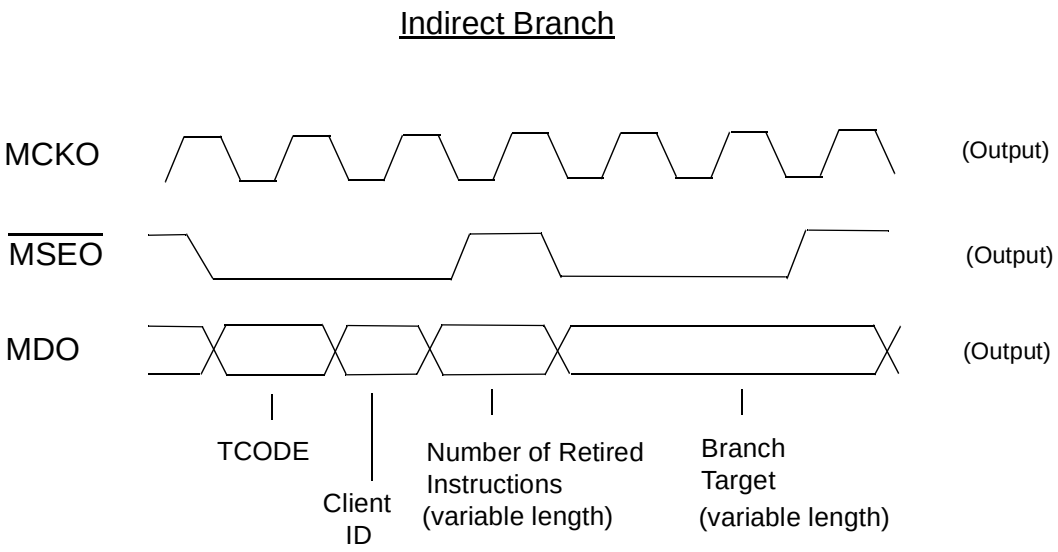


Figure 7-3—Timing Diagram for Indirect Branch Message

7.1 Rules for Messages

Embedded processors complying with Class 2, 3, or 4 shall provide messages via the AUX in a consistent manner as described below:

- A variable-sized packet within a message must end on a port boundary.
- Whenever a variable-length packet is sized so that it does not end on a port boundary, it is necessary to extend and zero-fill the remaining bits after the highest-order bit so that it can end on a port boundary.

  For example, if the MDO port is 4 bits wide, and the unique portion of an indirect address TCODE is 5 bits, then the remaining 3 bits of MDO must be packed with 0s.

Clock	MDO[3:0]				$\overline{\text{MSEO}}$[1:0]		
	3	2	1	0	1	0	
0	A3	A2	A1	A0	0	0	Normal Transfer
1	0	0	0	A4	0	1	End Packet

- A variable-sized packet may start within a port boundary only when following a fixed-length packet. (If two variable-sized packets end and start on the same clock, it is impossible to know which bit is from the last packet and which bit is from the next packet.)
- Processors that do not have A0 and/or A1 address bits must be consistent in their representation of address values within all messages. That is, bits A0/A1 must always be included or excluded from all Public Messages.
- To improve message compression, multiple vendor-fixed or fixed-length packets may start and end on a single clock.
- Each type of vendor-fixed or fixed-length packet must be the same within all messages. For example, if a vendor implements 3 bits to identify the source processor, then all Public Messages with a source processor packet must be 3 bits in length.
- When a vendor-fixed or fixed-length packet follows a variable sized packet, the vendor-fixed or fixed-length packet must start on the port boundary.
- $\overline{\text{MSEI}}$/$\overline{\text{MSEO}}$ protocol must be followed for both input and output messages.

7.2 Example AUX Messages Using Nexus Protocol

Table 7-2 and **Table 7-3** illustrate examples of one-pin and two-pin $\overline{\text{MSEO}}$ options for the same Indirect Branch Message (Traditional).

Note that T0 and S0 are the LSBs where

Tx = TCODE number
Sx = Client that is source of message
Ix = Number of instruction units
Ax = Unique portion of the address

Table 7-2—Indirect Branch Using the One-Pin $\overline{\text{MSEO}}$ Option

Clock	MDO[3:0]				$\overline{\text{MSEO}}$[0]	
	3	2	1	0	0	Idle
0	X	X	X	X	1	Idle (or end of last message)
1	T3	T2	T1	T0	0	Start Message
2	S1	S0	T5	T4	0	Normal Transfer
3	I3	I2	I1	I0	0	Normal Transfer
4	I7	I6	I5	I4	1	End Packet
5	A3	A2	A1	A0	0	Normal Transfer
6	A7	A6	A5	A4	1	End Packet
7	X	X	X	X	1	End Message
8	T3	T2	T1	T0	0	Start Message

Table 7-3—Indirect Branch Using the Two-Pin $\overline{\text{MSEO}}$ Option

Clock	MDO[3:0]				$\overline{\text{MSEO}}$[1:0]		
	3	2	1	0	1	0	
0	X	X	X	X	1	1	Idle (or end of last message)
1	T3	T2	T1	T0	0	0	Start Message
2	S1	S0	T5	T4	0	0	Normal Transfer
3	I3	I2	I1	I0	0	0	Normal Transfer
4	I7	I6	I5	I4	0	1	End Packet
5	A3	A2	A1	A0	0	0	Normal Transfer
6	A7	A6	A5	A4	1	1	End Packet/ Message
7	T3	T2	T1	T0	0	0	Start Message

SECTION 8
IEEE 1149.1 Message Protocol

Embedded processors complying with Class 1, 2, 3, or 4 may optionally implement messages via the **IEEE 1149.1** interface according to the Nexus standard.

Two basic categories of messages may be implemented: solicited and unsolicited. Solicited messages are initiated and transmitted from an external controller to the embedded processor (e.g., to read an NRR). Unsolicited messages are generated by the embedded processor and are normally transmitted at random times. Unsolicited messages are most commonly transmitted via the AUX; however, a mechanism is described in **8.4.2 - Nexus Public Message Access Protocol** that allows for the retrieval of unsolicited messages via an **IEEE 1149.1** interface.

8.1 IEEE 1149.1 Compatibility

An **IEEE 1149.1** port used for the Nexus standard shall implement all the mandatory features of a standard **IEEE 1149.1** port, including the “BYPASS” and “IDCODE” instructions. A 16-state **IEEE 1149.1** TAP state machine will be used per the **IEEE-1149.1** standard as illustrated in **Figure 8-1**.

Refer to **IEEE Std 1149.1-1990** for further details on electrical and pin protocol compliance requirements. Additional information on the **IEEE 1149.1** pin interface to connectors may be found in **Appendix A - Connector and Electrical Specifications.** Details on the NRRs may be found in **Appendix B - Recommendations for Access to Control and Status Registers.**

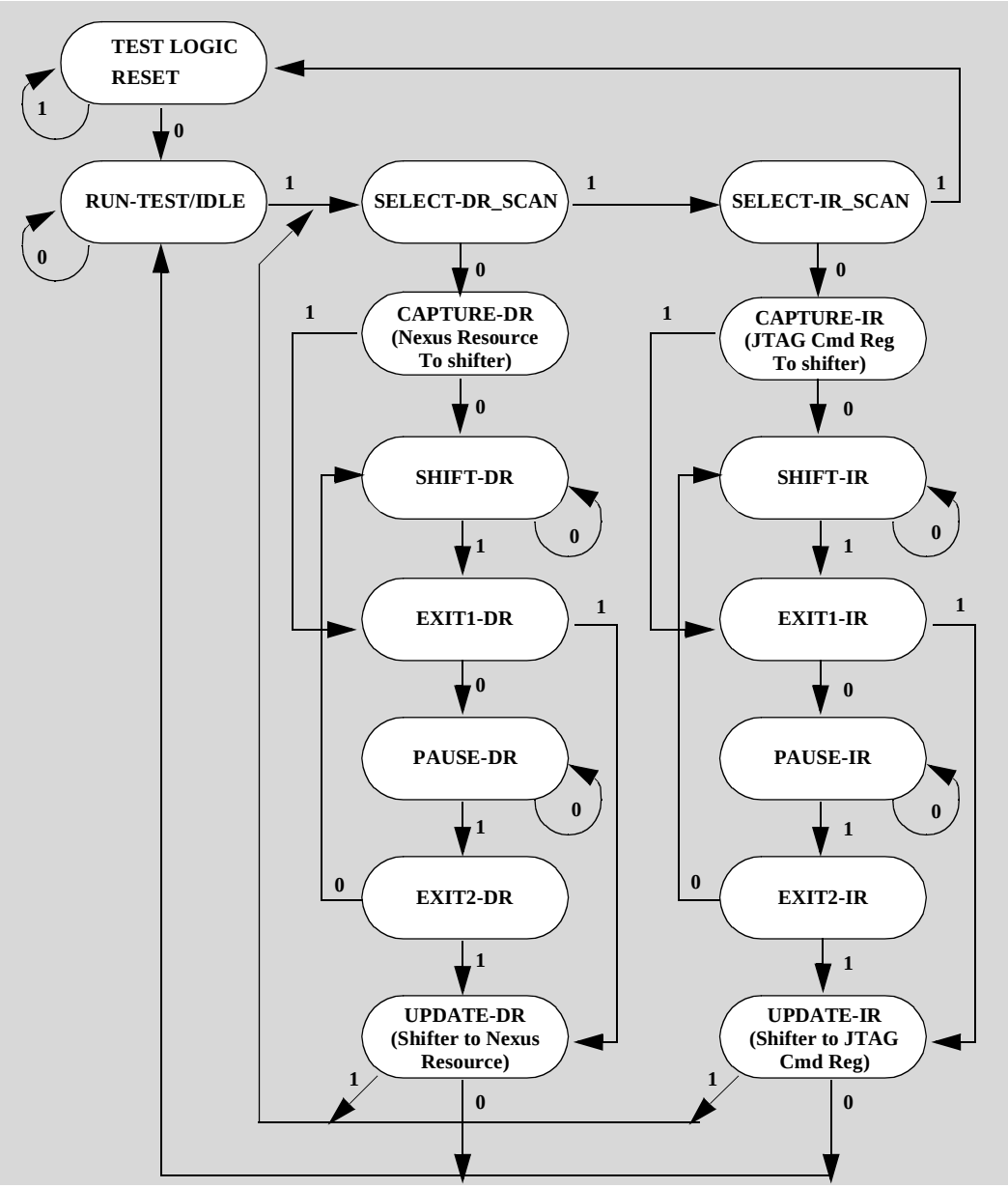


Figure 8-1—Sixteen-State IEEE 1149.1 Finite State Machine

8.1.1 Accessing the IEEE 1149.1 Device ID

Assertion of a power-on-reset signal on the embedded processor or the $\overline{\text{TRST}}$ pin causes the **IEEE 1149.1** controller to default to being loaded with the “IDCODE” instruction upon exit of TEST-LOGIC-RESET controller state. This allows immediate entry to the SELECT-DR_SCAN path to retrieve the contents of the device ID. The LSB of the IDCODE must be a logic 1 so that examination of the first bit of data shifted out of a component during a data scan sequence

immediately following exit from the TEST-LOGIC-RESET controller state will show whether an **IEEE 1149.1** DID Register is included in the design. The debug/ development tool may then retrieve the characteristics of the device to configure the software interface.

The system logic shall continue its normal operation undisturbed when the **IEEE 1149.1** controller is decoding the "IDCODE" instruction. All NRRs must be accessible through the **IEEE 1149.1** port independent of the state of the target processor.

8.1.2 Optional Ready ($\overline{\text{RDY}}$) Output Pin

To increase the transfer rate of the **IEEE 1149.1** port, an additional pin may be implemented to signal when data are ready to be transferred to and from NRRs. This may eliminate the need to poll NRRs for status information for synchronization purposes. This capability becomes especially important when performing read/write access transfers to different speed target memories.

The function of the $\overline{\text{RDY}}$ pin will be to assert (asynchronously) to a logic low whenever the read/write access transfer has completed without error and then de-assert when the **IEEE 1149.1** state machine has reached the CAPTURE_DR state.

The $\overline{\text{RDY}}$ pin may also be used for Nexus Public Messages as described in **8.4 - Accessing Nexus Public Messages via the IEEE 1149.1 Port**.

8.2 Accessing NRRs via the IEEE 1149.1 Port

In order to reduce the number of additional debug/development pins required for dynamic (real-time) debug, the Nexus standard defines an alternative mechanism to accessing NRRs via the **IEEE 1149.1** port. This is especially useful for embedded processors that implement **IEEE 1149.1** pins for static debug and/or boundary scan.

The mechanism defined in this section can be used in lieu of implementing the Read/Write Register Messages defined in **SECTION 5 - Nexus Public Messages**.

8.2.1 NRR Access Protocol

Access to NRRs is enabled when the **IEEE 1149.1** controller is decoding a vendor-defined "NEXUS-ACCESS" instruction entered via the SELECT-IR_SCAN path. When the **IEEE 1149.1** controller passes through the UPDATE_IR state and decodes the "NEXUS-ACCESS" instruction, the Nexus controller will be reset to the NRR select state. The Nexus controller will have three states: idle

(NRR_IDLE), register select state (NRR_REG_SEL), and register data access state (NRR_DATA_ACC).

NOTE

The “NEXUS-ACCESS” instruction can also be used to enable the Nexus module for low-cost (**IEEE 1149.1** only) implementations that may not implement the EVTI pin. Refer to **9.1.2 - Reset for IEEE 1149.1 Implementations**.

When the “NEXUS-ACCESS” instruction is being decoded by the **IEEE 1149.1** controller, the **IEEE 1149.1** port allows tool/target communications via up to 128 NRRs. Each NRR is referenced by a unique register address index in the range 0 through 127. Refer to **Appendix B - Recommendations for Access to Control and Status Registers** for specific register indices.

All communication with the Nexus controller is performed via the SELECT-DR_SCAN path. The Nexus controller will default to a register select state when enabled. Accessing an NRR requires two passes through the SELECT-DR_SCAN path, one pass to select the NRR and the second pass to read or write the NRR data.

The first pass through the SELECT-DR_SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit NRR address, as illustrated in **Figure 8-2**.

Figure 8-2—IEEE 1149.1 Controller Command Input

The second pass through the SELECT-DR_SCAN path is used to read or write the NRR data by shifting in the data LSB first during the SHIFT-DR state. When reading an NRR, the register value will be loaded into the **IEEE 1149.1** shifter during the CAPTURE-DR state. When writing to an NRR, the value will be loaded by the **IEEE 1149.1** shifter to the NRR during the UPDATE-DR state.

NOTE

When reading data from an NRR, there is no requirement to shift out the entire NRR contents, and shifting may be terminated once the required number of bits has been acquired. **Figure 8-3** illustrates the relationship between an **IEEE 1149.1** TAP state machine and a Nexus controller state machine.

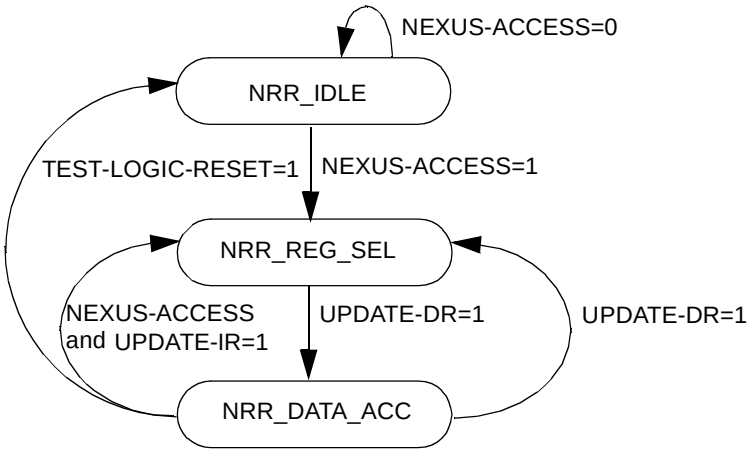


Figure 8-3—IEEE 1149.1 TAP State Machine Relationship to Nexus Controller State Machine

8.2.2 NRR Access Status (Optional)

In order to reduce the number of clock cycles required to poll registers to retrieve status information, an additional (optional) **IEEE 1149.1** instruction has been added. The access to this vendor-defined “NEXUS-STATUS” instruction will be identical to that of the “NEXUS-ACCESS” instruction.

A minimum of two status bits are recommended in the standard: 1 bit to indicate an error condition on DMA accesses (equivalent to the ERR bit within the Nexus-recommended RWCS Register) and 1 bit to indicate the pending status of a Nexus access. The LSBs of the “NEXUS-STATUS” instruction must remain 0b01 to comply with **IEEE 1149.1** interconnect testing requirements.

For both DMA accesses and normal register accesses, a synchronization busy (SB) bit is recommended to indicate to the tool that the outstanding access has not completed. This bit has the same function as the $\overline{\text{RDY}}$ pin, but the opposite polarity.

When performing a DMA access, an error condition will set the error status (ERR) bit. The ERR bit will remain asserted until a new access is initiated.

These status bits allow the **IEEE 1149.1** controller to simply access the “NEXUS-STATUS” Instruction Register (IR) value and eliminate the required polling of the RWCS Register.

NOTE

The width of the **IEEE 1149.1** IR is vendor-defined, but it is recommended to be at least 4 bits wide to facilitate the addition of the SB and ERR bits.

8.3 Read/Write Access via the IEEE 1149.1 Port

The read/write access registers, as described in **Appendix B - Recommendations for Access to Control and Status Registers,** provide a means for transferring single or multiple data values through the AUX or **IEEE 1149.1** port. When using the **IEEE 1149.1** port, the RWCS Register and RWA Register are initialized for the data transfer. Once initialization is complete, synchronization with the target must be handled by an external target controller.

Two methods will be available for synchronization of data transfers. The first method uses an optional pin called Ready for Transmission (RDY). The RDY signal asserts (asynchronously) to indicate that the Nexus module is ready for read access or that the write access has completed without error. An external development tool may then clock the **IEEE 1149.1** port and perform the next read/ write access. Use of a RDY pin permits data transfers in [16 + (data width)] TCKs, assuming the **IEEE 1149.1** controller starts from and ends in the SELECT-DR_SCAN state.

If a $\overline{\text{RDY}}$ pin is not made available, either the “NEXUS-STATUS” instruction must be implemented, or the RWCS Register ERR and DV bits must be polled. The polling method requires 65 TCKs for transfer of a 32-bit value.

8.4 Accessing Nexus Public Messages via the IEEE 1149.1 Port

Nexus Public Messages may be read from or written to the target via the **IEEE 1149.1** port. The method outlined in **8.4.1 - Nexus Input/Output Public Message Registers (IPMR/OPMR)** through **8.4.3 - Using RDY as Output Message Flag** provides a low-cost solution for providing a basic set of Nexus functionality.

This method allows a Class 1 implementation to support Class 2 and Class 3 features (at a reduced bandwidth). The performance classification would be minimal and may meet the transfer bandwidth requirements only for low-end applications.

If the embedded processor supports messaging via the AUX OUT as well as the **IEEE 1149.1** port, the selection mechanism is vendor defined.

8.4.1 Nexus Input/Output Public Message Registers (IPMR/OPMR)

Input Messages are generated by an external **IEEE 1149.1** controller and are placed in an Input Public Message Register (IPMR). The IPMR receives its TCODEs and packets via multiple passes through the SELECT-DR_SCAN path.

Output Messages are generated by the target processor. Because the **IEEE 1149.1** protocol does not permit Public Messages to be generated from an embedded target microcontroller, an Output Public Message Register (OPMR)

must be made available for transmission of Public Messages from the embedded target microcontroller to an external **IEEE 1149.1** controller.

The IPMR and OPMR may be implemented as NRRs, as illustrated in **Appendix B - Recommendations for Access to Control and Status Registers**.

8.4.2 Nexus Public Message Access Protocol

The IPMR and OPMR are used to transmit Nexus Public Messages via the **IEEE 1149.1** port instead of the AUX. These registers can be viewed as partitioned into slots. Each slot contains a predetermined number of equivalent AUX bits and $\overline{\text{MSE}}$ bits.

For an input message, these bits would be shifted into the IPMR and uploaded to the target in the UPDATE-DR state. For an output message, these bits would be loaded into the JTAG shifter during the CAPTURE-DR state and shifted out via TDO. The number of AUX bits and $\overline{\text{MSE}}$ bits for each implementation is vendor defined. **Table 8-1** and **Table 8-2** show an example 32-bit OPMR/IPMR implemented using 6-bit and 14-bit AUX equivalents, as well as the two-pin $\overline{\text{MSE}}$ option.

Table 8-1—IPMR/OPMR Register (6-bit AUX equivalent)

Bit Number	Field Name	Description
31-26	AUX3	AUX bits for “slot3”
25-24	MSE3	$\overline{\text{MSE}}$ bits for “slot3”
23-18	AUX2	AUX bits for “slot2”
17-16	MSE2	$\overline{\text{MSE}}$ bits for “slot2”
15-10	AUX1	AUX bits for “slot1”
9-8	MSE1	$\overline{\text{MSE}}$ bits for “slot1”
7-2	AUX0	AUX bits for “slot0”
1-0	MSE0	$\overline{\text{MSE}}$ bits for “slot0”

Table 8-2—IPMR/OPMR Register (14-bit AUX equivalent)

Bit Number	Field Name	Description
31-18	AUX1	AUX bits for “slot1”
17-16	MSE1	$\overline{\text{MSE}}$ bits for “slot1”
15-2	AUX0	AUX bits for “slot0”
1-0	MSE0	$\overline{\text{MSE}}$ bits for “slot0”

Messages are packetized and transmitted according to the Nexus standard, but the slots stored within the IPMR/OPMR are treated as a serial stream of data. For example, a Start Message slot (MSE = 00) can immediately follow an End Message slot (MSE = 11) within the same IPMR/OPMR. If no more messages are available, the remaining slots in the registers should be filled with idle slots.

Table 8-3 below shows a typical Indirect Branch Message using two-pin $\overline{\text{MSEO}}$ and six-pin MDO. **Table 8-4** shows how this AUX message is formatted for an OPMR transmission via the **IEEE 1149.1** port.

Table 8-3—Indirect Branch - AUX Example

Clock	Slot	AUX[5:0]						$\overline{\text{MSEO}}$[1:0]	
		5	4	3	2	1	0	0	Idle
0	X	X	X	X	X	X	X	11	Idle (or end of last message)
1	0	T5	T4	T3	T2	T1	T0	00	Start Message
2	1	I3	I2	I1	I0	S1	S0	00	Normal Transfer
3	2	0	0	I7	I6	I5	I4	01	End Packet
4	3	A5	A4	A3	A2	A1	A0	00	Normal Transfer
5	0	0	0	0	0	A7	A6	11	End Message
6	1	0	0	0	0	0	0	11	Idle
7	2	0	0	0	0	0	0	11	Idle
8	3	T5	T4	T3	T2	T1	T0	00	Start Message

Table 8-4—Indirect Branch - OPMR Example

Bit Number	Field Name	OPMR Value (1st transfer)						OPMR Value (2nd transfer)					
31-26	AUX3	A5	A4	A3	A2	A1	A0	T5	T4	T3	T2	T1	T0
25-24	MSE3	00						00					
23-18	AUX2	0	0	I7	I6	I5	I4	0	0	0	0	0	0
17-16	MSE2	01						11					
15-10	AUX1	I3	I2	I1	I0	S1	S0	0	0	0	0	0	0
9-8	MSE1	00						11					
7-2	AUX0	T5	T4	T3	T2	T1	T0	0	0	0	0	A7	A6
1-0	MSE0	00						11					

8.4.3 Using $\overline{\text{RDY}}$ as Output Message Flag

It is possible to detect when a Nexus Public Message is available in the OPMR. This method will require the selection of the OPMR and monitoring of the $\overline{\text{RDY}}$ pin. If $\overline{\text{RDY}}$ is a logic 1, the external **IEEE 1149.1** controller may terminate OPMR shifting. If the $\overline{\text{RDY}}$ is a logic 1, the Nexus controller will not advance to the register data access state, but instead will stay in the register select state.

An Output Message is ready for retrieval when $\overline{\text{RDY}}$ is a logic 0 and the Nexus controller will advance to the register data access state. The width of the OPMR will be vendor defined, where the vendor may optimize the register size depending upon the size of packets transmitted. **Figure 8-1** illustrates the **IEEE 1149.1** TAP state machine for accessing the Public Message registers as well as other NRRs.

NOTE

If the “NEXUS-STATUS” instruction is implemented, the SB status bit can also be used to indicate that a Nexus Public Message is available in the OPMR.

8.5 Sample IEEE 1149.1 Access Sequences

Table 8-5 illustrates the **IEEE 1149.1** sequence required to read the Device IDCODE immediately after assertion of the $\overline{\text{TRST}}$ pin or after 5 TCKs with the TMS pin at a logic 1.

Table 8-5—IEEE 1149.1 Sequence to Read Device IDCODE After TRST Pin Assertion

Step	TMS	IEEE 1149.1 State	Nexus State	Description
1	1	TEST-LOGIC-RESET	NRR_IDLE	**IEEE 1149.1** controller in reset state.
2	0	RUN-TEST-IDLE	NRR_IDLE	IDCODE loaded into **IEEE 1149.1** IR.
3	1	SELECT-DR_SCAN	NRR_IDLE	
4	0	CAPTURE-DR	NRR_IDLE	Load Device ID into TDI/TDO shifter.
5	0	SHIFT-DR	NRR_IDLE	TDO active and **IEEE 1149.1** shifter presents a 1 in LSB.
N–1 TCKs			NRR_IDLE	
6	1	EXIT1-DR	NRR_IDLE	Last bit of Device ID shifted out to TDO.
7	1	UPDATE-DR	NRR_IDLE	
8	0	RUN-TEST-IDLE	NRR_IDLE	**IEEE 1149.1** controller ready for instruction.

Table 8-6 illustrates the **IEEE 1149.1** sequence to select the Nexus controller.

Table 8-6—IEEE 1149.1 Sequence to Initiate Nexus Communication

Step	TMS	IEEE 1149.1 State	Nexus State	Description
1	0	RUN-TEST-IDLE	NRR_IDLE	**IEEE 1149.1** controller in reset state.
2	1	SELECT-DR_SCAN	NRR_IDLE	
3	1	SELECT-IR-SCAN	NRR_IDLE	
4	0	CAPTURE-IR	NRR_IDLE	Load last register select command into TDI/ TDO shifter.
5	0	SHIFT-IR	NRR_IDLE	TDO becomes active and the **IEEE 1149.1** shifter is ready. Shift *N*-1 bits into TDI of size of vendor-defined “NEXUS-ACCESS” instruction.
N–1 TCKs				
6	1	EXIT1-IR	NRR_IDLE	Last bit of Device ID shifted out to TDO.
7	1	UPDATE-IR	NRR_REG_SEL	**IEEE 1149.1** controller decoder. Nexus controller is forced to register select state.
8	0	RUN-TEST-IDLE	NRR_REG_SEL	Nexus controller enabled and ready to receive commands.

Table 8-7 illustrates an **IEEE 1149.1** sequence that will write a 32-bit value to an NRR.

Table 8-7—IEEE 1149.1 Sequence to Write to an NRR

Step	TMS	IEEE 1149.1 State	Nexus State	Description
1	0	RUN-TEST-IDLE	NRR_REG_SEL	**IEEE 1149.1** controller in idle state.
2	1	SELECT-DR_SCAN	NRR_REG_SEL	
3	0	CAPTURE-DR	NRR_REG_SEL	**IEEE 1149.1** shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.
4	0	SHIFT-DR	NRR_REG_SEL	TDO becomes active, and NRR address and write bit are shifted in through TDI.
7 TCKs			NRR_REG_SEL	
5	1	EXIT1-DR	NRR_REG_SEL	Last bit of NRR shifted into TDI.
6	1	UPDATE-DR	NRR_REG_SEL	Nexus controller decodes and selects register.
7	1	SELECT-DR_SCAN	NRR_DATA_ACC	Second pass through SELECT-DR_SCAN.
8	0	CAPTURE-DR	NRR_DATA_ACC	**IEEE 1149.1** shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.
9	0	SHIFT-DR	NRR_DATA_ACC	TDO becomes active and outputs current value of register while new value is shifted in through TDI.
N–1 TCKs			NRR_DATA_ACC	
10	1	EXIT1-DR	NRR_DATA_ACC	Last bit of NRR shifted out to TDO.
11	1	UPDATE-DR	NRR_DATA_ACC	Nexus controller writes value to register.

Table 8-7—IEEE 1149.1 Sequence to Write to an NRR *(Continued)*

Step	TMS	IEEE 1149.1 State	Nexus State	Description
12	0	RUN-TEST/IDLE	NRR_REG_SEL	**IEEE 1149.1** controller returns to idle state or may return to SELECT-DR_SCAN state for new NRR register select. Total number of TCKs = 49 in this example.

Table 8-8—IEEE 1149.1 Sequence for Read/Write Access with $\overline{\text{RDY}}$ Pin

Step	TMS	IEEE 1149.1 State	Nexus State	Description
1	1	SELECT-DR_SCAN	NRR_REG_SEL	Starting point of this example.
2	0	CAPTURE-DR	NRR_REG_SEL	**IEEE 1149.1** shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.
De-assert $\overline{\text{RDY}}$ pin and set SB bit				
3	0	SHIFT-DR	NRR_REG_SEL	TDO becomes active, and Nexus RWD Register is selected for write. Data are then shifted from TDI.
7 TCKs			NRR_REG_SEL	
4	1	EXIT1-DR	NRR_REG_SEL	Last bit of Nexus RWD Register shifted from TDI.
5	1	UPDATE-DR	NRR_REG_SEL	Nexus controller decodes and selects register.
6	1	SELECT-DR_SCAN	NRR_DATA ACC	Second pass through SELECT-DR_SCAN.
Wait for $\overline{\text{RDY}}$ pin to be asserted or for SB bit to be cleared (read operation)				
7	0	CAPTURE-DR	NRR_DATA ACC	**IEEE 1149.1** shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.
De-assert $\overline{\text{RDY}}$ pin and set SB bit				
8	0	SHIFT-DR	NRR_DATA ACC	TDO becomes active and outputs current value of register while new value is shifted in through TDI.
N–1 TCKs			NRR_DATA ACC	
9	1	EXIT1-DR	NRR_DATA ACC	Last bit of NRR shifted out to TDO.
10	1	UPDATE-DR	NRR_DATA ACC	Nexus controller writes value to register.
11	1	SELECT-DR_SCAN	NRR_REG_SEL	**IEEE 1149.1** controller returns to SELECT-DR_SCAN state for new NRR select. Total number of TCKs = 48 in this example.
Wait for $\overline{\text{RDY}}$ pin to be asserted or for SB bit to be cleared (write operation)				
The IEEE 1149.1 state machine can be taken either to the SELECT-DR_SCAN state if another operation is pending or to the RUN-TEST-IDLE state.				

SECTION 9
Implementation Topics

9.1 Nexus Reset Configuration

Embedded processors complying with Class 2, 3, or 4 shall receive reset configuration information according to the Nexus standard to completely enable/ disable message transmission on the AUX OUT. If message transmission is enabled, output messages shall be transmitted normally. If message transmission is disabled, auxiliary output pins shall be tied inactive (or three-stated) and no messages shall be transmitted.

NOTE

If the system clock is used as the MCKO function, then it is not required to tie inactive (or three-state) the system clock via reset configuration.

9.1.1 Reset for AUX-Only (Full-Duplex) Implementations

For Nexus implementations that consist of only auxiliary pins, reset configuration information must be valid on $\overline{\text{EVTI}}$ for at least four system clocks of the embedded processor prior to negation of $\overline{\text{RSTI}}$. The Nexus module samples $\overline{\text{EVTI}}$ at the negation of $\overline{\text{RSTI}}$.

The $\overline{\text{EVTI}}$ pin shall have a pull-up resistor with the following reset configuration states.

Reset State	Description
0	Message transmission enabled.
1	Message transmission disabled (default).

9.1.2 Reset for IEEE 1149.1 Implementations

For Nexus modules that use a combination of AUX and **IEEE 1149.1** port or that use an **IEEE 1149.1** port only (half-duplex), there are three ways in which the Nexus module can be enabled:

- $\overline{\text{EVTI}}$ assertion upon de-assertion of $\overline{\text{TRST}}$ pin

Reset configuration must be valid at least two **IEEE 1149.1**-defined TCKs before the negation of TRST. Refer to **A.4 - AC Electrical Characteristics - IEEE 1149.1 Interface** for detail on EVTI electrical characteristics.

- $\overline{\text{EVTI}}$ assertion during **IEEE 1149.1** "TEST-LOGIC-RESET" state

Because the $\overline{\text{TRST}}$ pin is optional, this mechanism allows Nexus modules implementing the **IEEE 1149.1** port without TRST to have a mechanism to enable Nexus. The "TEST-LOGIC-RESET" state can be reached by cycling through the **IEEE 1149.1** state machine using the TMS pin.

- Upon "NEXUS-ACCESS" IR instruction

For low-cost implementations, it may not be feasible to implement the $\overline{\text{EVTI}}$ pin. Using the **IEEE 1149.1**-defined "NEXUS-ACCESS" IR value to enable Nexus allows low-cost implementations to use only **IEEE 1149.1**-defined pins and low-cost connectors. The specific value of the instruction is vendor defined. See **8.3 - Read/Write Access via the IEEE 1149.1 Port** for detail on how the "NEXUS-ACCESS" IR value is used to access Nexus registers.

9.1.3 Reset and Port Replacement

Embedded processors implementing LSIO port replacement shall receive reset configuration information according to the Nexus standard to enable/disable message transmission on the AUX OUT. If message transmission is enabled, output messages will be transmitted normally with support for Port Replacement Messages according to the Nexus standard. If message transmission is disabled, auxiliary output pins shall provide vendor-defined LSIO capability.

9.2 Multiple Processor Implementations

The Nexus standard allows for embedded processor implementations that comprise multiple clients to utilize a single AUX, depending upon the transfer bandwidth requirement for the application. The AUX may be designated for a single client or shared by multiple clients on the embedded device during runtime. Messages transmitted via the AUX shall contain information defined by the Nexus standard indicating which client generated the message. The SRC field within each Public Message allows the development tools to distinguish which Nexus client sent the particular message.

9.3 Multiple Address Threads

On embedded processors that implement data and program trace, there will be an address thread for each type of trace: the data address thread for both Data Trace - Data Read Messages and Data Trace - Data Write Messages and the instruction

address thread for all Program Trace Messages. Messages containing a data address packet will be encoded and compressed using the data address most recently transmitted, thus creating a data address thread. Likewise, messages containing an instruction address packet will be encoded and compressed using the instruction address most recently transmitted, thus creating an instruction address thread.

It is recommended that separate instruction and data address threads be maintained, but for low-cost applications that may require address correlation between program and data trace, there are two solutions allowed within the Nexus standard:

- Use Program Trace - Correlation Messages

  It is possible to use the Program Trace - Correlation Message to correlate events, such as Data Trace Messages, with the instruction flow. The Nexus standard recommends correlation of certain types of events, but also has left space for vendor-defined events. Refer to **5.3.16 - Program Trace - Correlation Message** for detail.

- Maintain a single instruction/data address thread

  This option is not recommended and highly discouraged. It is discouraged because of the increased bandwidth required and the severe feature set limitations it places on development tools. The bandwidth required is increased because locality of reference is lost when switching back and forth between instruction address space and data address space. For tools, storage-enabling of message types is inhibited and modular design of packet encoding and decoding is prevented, thus limiting execution speed, etc. High-end cores should not even consider a one-thread approach.

  With one address thread, the next address (data or instruction) will be generated from the last message (Data Trace Message or Program Trace Message, respectively).

9.4 Simultaneous Development of Multiple Embedded Processors

To facilitate development of multiple embedded processors interconnected by an existing serial communication bus standard, control and status information defined in the Nexus standard may be required to be accessible in the programmer’s model. If this is required, precautions should be taken to ensure that

- Development resources are used only for development and not for application purposes.
- Security should be provided for proprietary applications to restrict access to the application program.

9.5 Security

Vendor-defined enable/disable mechanisms internal to the embedded processor may be optionally provided for secure visibility of user resources on the embedded processor.

9.6 Single Master for Tool Connection

The Nexus standard does not support multiple tools connected directly to the Nexus input port. In other words, arbitration for multiple external tools is not supported by the port. To connect multiple tools, either the tools should manage the arbitration, or a single low-level tool should be connected with multiple high-level tools interconnected and arbitrated by the single low-level tool.

APPENDIX A
Connector and Electrical Specifications

A.1 Connection Options

Three possible connector options are available to be used for an AUX only connection or a combined IEEE-1149.1-Auxiliary connection. These three options use AMP System 50 connectors, GlenAir MicroD connectors, or Mictor connectors (see **Table A-1**).

On all connector options, unused pins should be left as no connects.

Table A-2 and **Table A-3** list the connector options and the signals in each option.

The connector naming convention is as follows:

<connector style><number of pins><interface type>

where

Connector style is

S = AMP style (System 50)
R = Robust Glenair MicroD
M = Mictor (Matched Impedance Connector)

Number of pins is 25, 26, 37, 38, 40, 50, 51, 78, or 100.

Interface type is

C = Combined IEEE-1149.1-Auxiliary Out
A = Auxiliary only (In and Out)

The AMP System 50 connector option is scalable and can be configured as 26, 40, 50, or 100 pins (see **Table A-4**).

The GlenAir MicroD connector option provides a robust connector and can have 25, 37, 51, or 100 pins (see **Table A-5** through **Table A-7**).

The 3M Mictor 38-pin connector option is also scalable and can use either one or two connectors, depending on the number of signals required (see **Table A-8** and **Table A-9**).

Note: Throughout this section, x is used as a wildcard character to replace any of the above options.

Table A-1—Connector Part Numbers (Target)

Connector		Part Number	
AMP Style	S26x	1-104068-2	AMP System 50
	S40x	104549-6	AMP System 50
	S50x	104549-7	AMP System 50
	S100x	104549-0	AMP System 50
Robust	R25x	MR7580-25P2BNU	Glenair MicroD
	R37x	MR7580-37P2BNU	Glenair MicroD
	R51x	MR7580-51P2BNU	Glenair MicroD
	R100x	MR7580-100P2BNU	Glenair MicroD
Mictor	M38x	767054-1	AMP MICTOR
	M76x	Two times 767054-1	AMP MICTOR

Table A-2—Combined IEEE 1149.1-Auxiliary Connector Options

Pin Name	AMP Style				Robust				MICTOR		Comments
	S26C	S40C	S50C	S100C	R25C	R37C	R51C	R100C	M38C	M76C	
MCKI	—	—	—	—	—	—	—	—	—	—	AUX
MDI	—	—	—	—	—	—	—	—	—	—	
$\overline{\text{MSEI}}$	—	—	—	—	—	—	—	—	—	—	
MCKO	1	1	1	1	1	1	1	1	1	1	
MDO	1	6	8	16	1	5	8	16	8	16	
$\overline{\text{MSEO}}$	1	2	2	2	1	2	2	2	2	2	
$\overline{\text{EVTO}}$	1	1	1	1	1	1	1	1	1	1	
$\overline{\text{EVTI}}$	1	1	1	1	1	1	1	1	1	1	
$\overline{\text{RSTI}}$	—	—	—	—	—	—	—	—	—	—	
PORT	—	—	—	16	—	—	—	16	—	16	Port Replacement
IEEE 1149.1 Pins	5	5	5	5	5	5	5	5	5	5	IEEE 1149.1
$\overline{\text{RDY}}$	1	1	1	1	1	1	1	1	1	1	
VREF	1	1	1	1	1	1	1	1	1	1	System Signals
RESET	1	1	1	1	1	1	1	1	1	1	
CLOCKOUT	—	—	—	—	—	—	—	—	1	1	
Vendor-Defined	1	1	2	3	0	1	3	3	5	9	—
Tool-Defined	1	2	4	4	1	2	4	4	4	9	—
VALTREF	1	1	1	1	1	1	1	1	1	1	Power
UBATT	2	2	2	2	2	2	2	2	2	2	
GROUND	8	15	20	45	8	13	20	45	5[a]	10[a]	
TOTAL SIGNALS[b]	16	23	28	53	15	22	29	53	32	65	—
TOTAL PINS	26	40	50	100	25	37	51	100	38	76	—

a. Built into the Mictor connector.
b. GROUND and UBATT not included in total signals.

Table A-3—Auxiliary-Only Connector Options

Pin Name	AMP Style				Robust				MICTOR		Comments
	S26A	S40A	S50A	S100A	R25A	R37A	R51A	R100A	M38A	M76A	
MCKI	1	1	1	1	1	1	1	1	1	1	AUX
MDI	1	2	4	4	1	2	4	1	4	4	
$\overline{\text{MSEI}}$	1	1	1	1	1	1	1	1	1	1	
MCKO	1	1	1	1	1	1	1	1	1	1	
MDO	1	6	8	16	1	5	8	16	8	16	
$\overline{\text{MSEO}}$	1	2	2	2	1	2	2	2	2	2	
$\overline{\text{EVTO}}$	1	1	1	1	1	1	1	1	1	1	
$\overline{\text{EVTI}}$	1	1	1	1	1	1	1	1	1	1	
$\overline{\text{RSTI}}$	1	1	1	1	1	1	1	1	1	1	
PORT	—	—	—	16	—	—	—	16	—	16	Port Replacement
IEEE 1149.1 Pins	—	—	—	—	—	—	—	—	—	—	IEEE 1149.1
$\overline{\text{RDY}}$	—	—	—	—	—	—	—	—	—	—	
VREF	1	1	1	1	1	1	1	1	1	1	System Signals
RESET	1	1	1	1	1	1	1	1	1	1	
CLOCKOUT	—	—	—	—	—	—	—	—	—	—	
Vendor-Defined	1	1	2	3	0	1	3	3	5	5	—
Tool-Defined	3	3	3	3	3	3	3	3	2	2	—
VALTREF	1	1	1	1	1	1	1	1	2	2	Power
UBATT	2	2	2	2	2	2	2	2	2	2	
GROUND	8	15	20	45	8	13	20	45	—[a]	—[a]	
TOTAL SIGNALS[b]	16	23	28	53	15	22	29	53	31	65	—
TOTAL PINS	26	40	50	100	25	37	51	100	38	76	—

a. Built into the Mictor connector.
b. GROUND and UBATT not included in total signals.

Table A-4—AMP System 50 Definition - Sxxx

S100	S50	S40	S26	Nexus Combined Signal C	Nexus Auxiliary Signal A	I/O	Pin Num-ber	Pin Num-ber	I/O	Nexus Auxiliary Signal A	Nexus Combined Signal C
A100	A50	A40	A26	UBATT	UBATT	OUT	1	2	OUT	UBATT	UBATT
				VALTREF	VALTREF	OUT	3	4	I/O	TOOL_IO0	TOOL_IO0
				TDO	TOOL_IO1	I/O	5	6	I/O	TOOL_IO2	RDY
				RESET	RESET	IN	7	8	OUT	VREF	VREF
				EVTI	EVTI	IN	9	10	—	GND	GND
				TRST	RSTI	IN	11	12		GND	GND
				TMS	MSEI	IN	13	14	—	GND	GND
				TDI	MDI0	IN	15	16	—	GND	GND
				TCK	MCKI	IN	17	18	—	GND	GND
				MDO0	MDO0	OUT	19	20	—	GND	GND
				MCKO	MCKO	OUT	21	22	—	GND	GND
				EVTO	EVTO	OUT	23	24	—	GND	GND
				MSEO0	MSEO0	OUT	25	26	I/O	VENDOR_IO0	VENDOR_IO0
				MDO1	MDO1	OUT	27	28	—	GND	GND
				MDO2	MDO2	OUT	29	30	—	GND	GND
				MDO3	MDO3	OUT	31	32	—	GND	GND
				TOOL_IO1	MDI1	IN	33	34	—	GND	GND
				MSEO1	MSEO1	OUT	35	36	—	GND	GND
				MDO4	MDO4	OUT	37	38	—	GND	GND
				MDO5	MDO5	OUT	39	40	—	GND	GND
				MDO6	MDO6	OUT	41	42	—	GND	GND
				MDO7	MDO7	OUT	43	44	—	GND	GND
				TOOL_IO2	MDI2	IN	45	46	—	GND	GND
				TOOL_IO3	MDI3	IN	47	48	—	GND	GND
				VENDOR_IO1	VENDOR_IO1	I/O	49	50	—	GND	GND
				VEND_IO2	VEND_IO2	OUT	51	52	—	GND	GND
				MDO8	MDO8	OUT	53	54	—	GND	GND
				MDO9	MDO9	OUT	55	56	—	GND	GND
				MDO10	MDO10	OUT	57	58	—	GND	GND
				MDO11	MDO11	OUT	59	60	—	GND	GND
				MDO12	MDO12	OUT	61	62	—	GND	GND
				MDO13	MDO13	OUT	63	64	—	GND	GND
				MDO14	MDO14	OUT	65	66	—	GND	GND

Table A-4—AMP System 50 Definition - Sxxx *(Continued)*

S100	S50	S40	S26	Nexus Combined Signal C	Nexus Auxiliary Signal A	I/O	Pin Num-ber	Pin Num-ber	I/O	Nexus Auxiliary Signal A	Nexus Combined Signal C
A100				MDO15	MDO15	I/O	67	68	—	GND	GND
				PORT0	PORT0	I/O	69	70	—	GND	GND
				PORT1	PORT1	I/O	71	72	—	GND	GND
				PORT2	PORT2	I/O	73	74	—	GND	GND
				PORT3	PORT3	I/O	75	76	—	GND	GND
				PORT4	PORT4	I/O	77	78	—	GND	GND
				PORT5	PORT5	I/O	79	80	—	GND	GND
				PORT6	PORT6	I/O	81	82	—	GND	GND
				PORT7	PORT7	I/O	83	84	—	GND	GND
				PORT8	PORT8	I/O	85	86	—	GND	GND
				PORT9	PORT9	I/O	87	88	—	GND	GND
				PORT10	PORT10	I/O	89	90	—	GND	GND
				PORT11	PORT11	I/O	91	92	—	GND	GND
				PORT12	PORT12	I/O	93	94	—	GND	GND
				PORT13	PORT13	I/O	95	96	—	GND	GND
				PORT14	PORT14	I/O	97	98	—	GND	GND
				PORT15	PORT15	I/O	99	100	—	GND	GND

Table A-5—GlenAir Robust Combined Definition - RxxC

Nexus Combined Robust Connector R25C MR7580-25P2BNU,	
14 GND	1 UBATT
15 TDI	2 UBATT
16 GND	3 VALTREF
17 TCK	4 TOOL_IO0
18 GND	5 TDO
19 MDO0	6 RDY
20 GND	7 RESET
21 MCKO	8 VREF
22 GND	9 EVTI
23 EVTO	10 GND
24 GND	11 TRST
25 MSEO0	12 GND
	13 TMS

Nexus Combined Robust Connector R37C MR7580-37P 2BNU	
20 GND	1 UBATT
21 MCKO	2 UBATT
22 GND	3 VALTREF
23 EVTO	4 TOOL_IO0
24 GND	5 TDO
25 MSEO0	6 RDY
26 VEN_IO0	7 RESET
27 MDO1	8 VREF
28 GND	9 EVTI
29 MDO2	10 GND
30 GND	11 TRST
31 MDO3	12 GND
32 GND	13 TMS
33 TOOL_IO1	14 GND
34 GND	15 TDI
35 MSEO1	16 GND
36 GND	17 TCK
37 MDO4	18 GND
	19 MDO0

Nexus Combined Robust Connector R51C Robust MR7580-51P2BNU		
	19 MDO0	1 UBATT
36 GND	20 GND	2 UBATT
37 MDO4	21 MCKO	3 VALTREF
38 GND	22 GND	4 TOOL_IO0
39 MDO5	23 EVTO	5 TDO
40 GND	24 GND	6 RDY
41 MDO6	25 MSEO0	7 RESET
42 GND	26 VEN_IO	8 VREF
43 MDO7	27 MDO1	9 EVTI
44 GND	28 GND	10 GND
45 TOOL_IO2	29 MDO2	11 TRST
46 GND	30 GND	12 GND
47 TOOL_IO3	31 MDO3	13 TMS
48 GND	32 GND	14 GND
49 VEN_IO1	33 TOOL_IO1	15 TDI
50 GND	34 GND	16 GND
51 VEN_IO2	35 MSEO1	17 TCK
		18 GND

Table A-6—GlenAir Robust Auxiliary Definition RxxA

Nexus Auxiliary Robust Connector R25A MR7580-25P 2BNU,		Nexus Auxiliary Robust Connector R37A MR7580-37P 2BNU		Nexus Auxiliary Robust Connector R51A Robust MR7580-51P 2BNU	
	1 UBATT		1 UBATT		1 UBATT
14 GND	2 UBATT	20 GND	2 UBATT	19 MDO0	2 UBATT
15 MDI0	3 VALTREF	21 MCKO	3 VALTREF	36 GND	3 VALTREF
16 GND	4 TOOL_IO0	22 GND	4 TOOL_IO0	20 GND	4 TOOL_IO0
17 MCKI	5 TOOL_IO1	23 EVTO	5 TOOL_IO1	37 MDO4	5 TOOL_IO1
18 GND	6 TOOL_IO2	24 GND	6 TOOL_IO2	21 MCKO	6 TOOL_IO2
19 MDO0	7 RESET	25 MSEO0	7 RESET	38 GND	7 RESET
20 GND	8 VREF	26 VEN_IO0	8 VREF	39 MDO5	8 VREF
21 MCKO	9 EVTI	27 MDO1	9 EVTI	22 GND	9 EVTI
22 GND	10 GND	28 GND	10 GND	40 GND	10 GND
23 EVTO	11 RSTI	29 MDO2	11 RSTI	23 EVTO	11 RSTI
24 GND	12 GND	30 GND	12 GND	41 MDO6	12 GND
25 MSEO0	13 MSEI	31 MDO3	13 MSEI	24 GND	13 MSEI
		32 GND	14 GND	42 GND	14 GND
		33 MDI1	15 MDI0	25 MSEO0	15 MDI0
		34 GND	16 GND	43 MDO7	16 GND
		35 MSEO1	17 MCKI	26 VEN_IO	17 MCKI
		36 GND	18 GND	44 GND	18 GND
		37 MDO4	19 MDO0	27 MDO1	
				45 MDI2	
				28 GND	
				46 GND	
				29 MDO2	
				47 MDI3	
				30 GND	
				48 GND	
				31 MDO3	
				49 VEN_IO1	
				32 GND	
				50 GND	
				33 MDI1	
				51 VEN_IO2	
				34 GND	
				35 MSEO1	

Table A-7—GlenAir R100x Definition

Nexus Combined Robust Connector R100C - MR7580-100P2BNU

76 GND	52 GND	27 MDO1	1 UBATT
77 PORT4	53 MDO8	28 GND	2 UBATT
78 GND	54 GND	29 MDO2	3 VALTREF
79 PORT5	55 MDO9	30 GND	4 TOOL_IO0
80 GND	56 GND	31 MDO3	5 TDO
81 PORT6	57 MDO10	32 GND	6 RDY
82 GND	58 GND	33 TOOL_IO1	7 RESET
83 PORT7	59 MDO11	34 GND	8 VREF
84 GND	60 GND	35 MSEO1	9 EVTI
85 PORT8	61 MDO12	36 GND	10 GND
86 GND	62 GND	37 MDO4	11 TRST
87 PORT9	63 MDO13	38 GND	12 GND
88 GND	64 GND	39 MDO5	13 TMS
89 PORT10	65 MDO14	40 GND	14 GND
90 GND	66 GND	41 MDO6	15 TDI
91 PORT11	67 MDO15	42 GND	16 GND
92 GND	68 GND	43 MDO7	17 TCK
93 PORT12	69 PORT0	44 GND	18 GND
94 GND	70 GND	45 TOOL_IO2	19 MDO0
95 PORT13	71 PORT1	46 GND	20 GND
96 GND	72 GND	47 TOOL_IO3	21 MCKO
97 PORT14	73 PORT2	48 GND	22 GND
98 GND	74 GND	49 VEN_IO1	23 EVTO
99 PORT15	75 PORT3	50 GND	24 GND
100 GND		51 VEN_IO2	25 MSEO0
			26 VEN_IO

Nexus Auxiliary Robust Connector R100A - MR7580-100P2BNU

76 GND	52 GND	27 MDO1	1 UBATT
77 PORT4	53 MDO8	28 GND	2 UBATT
78 GND	54 GND	29 MDO2	3 VALTREF
79 PORT5	55 MDO9	30 GND	4 TOOL_IO0
80 GND	56 GND	31 MDO3	5 TOOL_IO1
81 PORT6	57 MDO10	32 GND	6 TOOL_IO2
82 GND	58 GND	33 MDI1	7 RESET
83 PORT7	59 MDO11	34 GND	8 VREF
84 GND	60 GND	35 MSEO1	9 EVTI
85 PORT8	61 MDO12	36 GND	10 GND
86 GND	62 GND	37 MDO4	11 RSTI
87 PORT9	63 MDO13	38 GND	12 GND
88 GND	64 GND	39 MDO5	13 MSEI
89 PORT10	65 MDO14	40 GND	14 GND
90 GND	66 GND	41 MDO6	15 MDI0
91 PORT11	67 MDO15	42 GND	16 GND
92 GND	68 GND	43 MDO7	17 MCKI
93 PORT12	69 PORT0	44 GND	18 GND
94 GND	70 GND	45 MDI2	19 MDO0
95 PORT13	71 PORT1	46 GND	20 GND
96 GND	72 GND	47 MDI3	21 MCKO
97 PORT14	73 PORT2	48 GND	22 GND
98 GND	74 GND	49 VEN_IO1	23 EVTO
99 PORT15	75 PORT3	50 GND	24 GND
100 GND		51 VEN_IO2	25 MSEO0
			26 VEN_IO

Table A-8—MICTOR Connector M38x and 1/2 of M76x

Combined M38C or M76C	Aux Only M38A or M76A					Aux Only M38A or M76A	Combined M38C or M76C
MSEO0	MSEO0	OUT	38	37	OUT	VALTREF	VALTREF
MSEO1	MSEO1	OUT	36	35	IN/OUT	TOOL_IO0	TOOL_IO0
MCKO	MCKO	OUT	34	33	OUT	UBATT	UBATT
EVTO	EVTO	OUT	32	31	OUT	UBATT	UBATT
MDO0	MDO0	OUT	30	29	IN/OUT	MDI1	TOOL_IO1
MDO1	MDO1	OUT	28	27	IN/OUT	MDI2	TOOL_IO2
MDO2	MDO2	OUT	26	25	IN/OUT	MDI3	TOOL_IO3
MDO3	MDO3	OUT	24	23	IN/OUT	VEND_IO1	VEND_IO1
MDO4	MDO4	OUT	22	21	IN	RSTI	TRST
MDO5	MDO5	OUT	20	19	IN	MDI0	TDI
MDO6	MDO6	OUT	18	17	IN	MSEI	TMS
MDO7	MDO7	OUT	16	15	IN	MCKI	TCK
RDY	TOOL_IO_2	IN/OUT	14	13	IN/OUT	VEND_IO4	VEND_IO4
VREF	VREF	OUT	12	11	IN/OUT	TOOL_IO1	TDO
EVTI	EVTI	IN	10	9	IN	RESET	RESET
VEND_IO3	VEND_IO3	IN/OUT	8	7	IN/OUT	VEND_IO2	VEND_IO2
CLKOUT	CLKOUT	OUT	6	5	IN/OUT	VEND_IO0	VEND_IO0
RSVD4	RSVD4		4	3		RSVD3	RSVD3
RSVD2	RSVD2		2	1		RSVD1	RSVD1[a]

a. Pins 1 to 4 should be considered reserved" by logic analyzers.

Table A-9—MICTOR Connector M76x (Second Half)

Combined M76C	Aux Only M76A					Aux Only M76A	Combined M76C
PORT0	PORT0	IN/OUT	38	37	OUT	MDO8	MDO8
PORT1	PORT1	IN/OUT	36	35	OUT	MDO9	MDO9
PORT2	PORT2	IN/OUT	34	33	OUT	MDO10	MDO10
PORT3	PORT3	IN/OUT	32	31	OUT	MDO11	MDO11
PORT4	PORT4	IN/OUT	30	29	OUT	MDO12	MDO12
PORT5	PORT5	IN/OUT	28	27	OUT	MDO13	MDO13
PORT6	PORT6	IN/OUT	26	25	OUT	MDO14	MDO14
PORT7	PORT7	IN/OUT	24	23	OUT	MDO15	MDO15
PORT8	PORT8	IN/OUT	22	21	IN/OUT	TOOL_IO_4	TOOL_IO4
PORT9	PORT9	IN/OUT	20	19	IN/OUT	TOOL_IO5	TOOL_IO5
PORT10	PORT10	IN/OUT	18	17	IN/OUT	TOOL_IO6	TOOL_IO6
PORT11	PORT11	IN/OUT	16	15	IN/OUT	TOOL_IO7	TOOL_IO7
PORT12	PORT12	IN/OUT	14	13	IN/OUT	VEND_IO9	VEND_IO9
PORT13	PORT13	IN/OUT	12	11	IN/OUT	VEND_IO8	VEND_IO8
PORT14	PORT14	IN/OUT	10	9	IN/OUT	VEND_IO7	VEND_IO7
PORT15	PORT15	IN/OUT	8	7	IN/OUT	VEND_IO6	VEND_IO6
-	-	-	6	5	IN/OUT	VEND_IO5	VEND_IO5
RSVD4	RSVD4		4	3		RSVD_3	RSVD3
RSVD2	RSVD2		2	1		RSVD1	RSVD1[a]

a. Pins 1 to 4 should be considered reserved" by logic analyzers.

Figure A-1 - Nexus M76x Connector Layout shows the recommended layout for the dual M76 connectors.

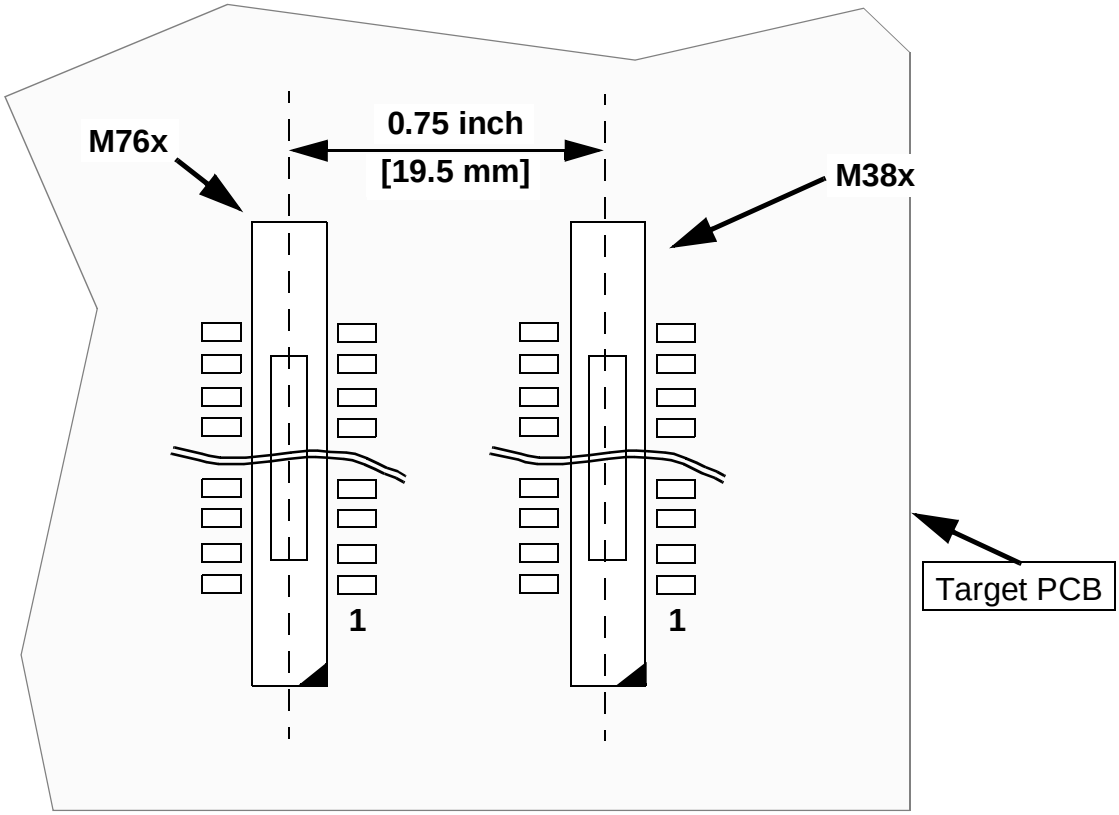


Figure A-1—Nexus M76x Connector Layout

A.1.1 Signal Descriptions

Signal description used throughout this appendix is as follows:

OUT = output from the target to the development tool
IN = input to the target from the development tool

Refer to **SECTION 6 - Nexus Port Signals** for a description of the following signals:

MDO, $\overline{\text{MSEO}}$, MCKO, MDI, $\overline{\text{MSEI0}}$, $\overline{\text{MSEI1}}$, MCKI, $\overline{\text{RSTI}}$, $\overline{\text{EVTI}}$, $\overline{\text{EVTO}}$

Refer to **IEEE Std 1149.1-1990** for a description of the following signals:

TDO, TDI, TCK, TMS, $\overline{\text{TRST}}$

A.1.1.1 CLOCKOUT

CLOCKOUT is the system clock from the target processor. CLOCKOUT helps development tools to determine the proper rate for TCK. CLOCKOUT can also be

used to indicate target activity and used for MCKO where it matches the needs of the interface. It can also be used in cases where the system clock is higher (or lower) than the AUX.

A.1.1.2 RESET

The RESET signal will cause the target to enter its reset state. The tool and target should use open-drain output drivers for this pin.

A.1.1.3 Vendor-Defined Signals

Vendor-defined signals may be used as needed by the target developer. Tool vendors should design their tools so that this signal can be configured as an input or output. These signals should be at a low enough slew rate as to not cause crosstalk on adjacent pins. Vendor_IO pins can also be used as Time Stamp pins if defined by the vendor.

A.1.1.4 Tool-Defined Signals

Tool-defined signals may be used as needed by the tool developer. This signal should be at a low enough slew rate as to not cause crosstalk on adjacent pins.

A.1.1.5 VREF

The VREF signal is used to establish the signaling levels of the debug interface of the target system. Any current drawn from this pin should be limited to that needed for voltage translation and/or signal interpolation and is not intended to supply logic functions or power. VREF is not necessarily at the target processor VDD level.

A.1.1.6 PORT[15:0]

Port replacement is a concept in which up to 16 LSIO pins of the target processor can also be used to carry AUX signals. The development tool connects to the original I/O devices on the target system via the PORT pins. The development tool performs the I/O functions on behalf of the target when the tool receives Port Replacement Messages from the target processor.

A.1.1.7 VALTREF

VALTREF provides an additional vendor-defined voltage reference. In systems that have a keep-alive voltage, it can be defined by the vendor to be the standby voltage to allow the tool to monitor when or if the keep-alive voltage is removed from the target system. In systems without standby requirements, this pin could be defined to provide an additional reference voltage.

A.1.1.8 UBATT

UBATT pins are vender-defined power supply pins. They are not to be used as logic signals. These pins provide a voltage to the tool that supplies a small amount of current. The connection should be reverse voltage protected. See **Table A-10**.

Table A-10—Recommended UBAT Specifications

Specification	Minimum	Maximum	Units
Voltage Range	5	20	V
Maximum Current	—	300	mA
Maximum In-rush Current[a]		1.0	A

a. Maximum duration of 3 ms.

A.1.2 Nexus Combined Implementation Considerations

It is recommended that the signals in **Table A-11** be connected to pull-ups on the target to prevent floating signals when the tool is not connected or powered on. TRST should have a pull-down. The purpose of these pull devices is to ensure all inputs are not left floating when a tool is not connected.

Table A-11—Signals for Pull-Ups on the Target

Signals	Pull Device Value
TMS, TCK, TDI, $\overline{\text{TRST}}$, $\overline{\text{RESET}}$, $\overline{\text{EVTI}}$	10K Ω

It is recommended that the signals in **Table A-12** be connected to pull-ups on the tool to prevent floating signals when the target is not connected or powered on.

Table A-12—Signals for Pull-Ups on the Tool

Signals	Pull-up
CLOCKOUT, TDO, $\overline{\text{EVTO}}$, $\overline{\text{RDY}}$	10K Ω

The target may need a jumper in the CLOCKOUT path near its source to prevent excessive radiated noise on the signal. This target design consideration eliminates the CLOCKOUT path between the central processing unit (CPU) and the debug connector when debug operations are not being performed.

WARNING

Any optional signals not used by the target must be left unconnected at the target debug connector.

A.1.3 Nexus Aux-Only Implementation Considerations

Because of its location on pin 2, VREF is used as a virtual ground for $\overline{\text{RESET}}$ on pin 1. Therefore, VREF should have decoupling capacitors at both ends of the cable connected to ground.

It is recommended that the signals in **Table A-13** be connected to pull-ups on the target to prevent floating signals when the tool is not connected or powered on. RSTI should have a pull-down.

Table A-13—Signals Connected to Pull-Ups on the Target

Signals	Pull Device Value
$\overline{\text{RESET}}$, $\overline{\text{EVTI}}$, $\overline{\text{RSTI}}$, $\overline{\text{MSEI}}$, MDI[0:4], MCKI	10K Ω

It is recommended that the signals in **Table A-14** be connected to pull-ups on the tool to prevent floating signals when the target is not connected or powered on. The purpose of these pull devices is to ensure all inputs are not left floating when a target is not connected.

Table A-14—Signals Connected to Pull-Ups on the Tool

Signals	Pull-up
MDO[16:0], MCKO, $\overline{\text{MSEO}}$, $\overline{\text{EVTO}}$, $\overline{\text{RDY}}$	10K Ω

WARNING

Any optional signals not used by the target must be left unconnected at the target debug connector.

A.2 DC Electrical Characteristics

Table A-15 lists the electrical characteristics for the signals used in the Nexus interface.

Table A-15—Electrical Characteristics in the Nexus Interface

Characteristic	VREF Voltage	Min	Max	Unit
Input Low Voltage	VREF 2.8 V to 5 V	–0.3	0.8	V
Input High Voltage		2.0	1.2 (VREF)	V
Input Low Voltage	VREF below 2.8 V	–0.3	0.3 (VREF)	V
Input High Voltage		0.7 (VREF)	1.2 (VREF)	V
VREF Output Current	—	—	1	mA

All dc characteristics apply to the **IEEE 1149.1** and AUX interfaces.

Output voltage levels need to be sufficient to satisfy the associated input requirements with a suitable margin.

The tool must sense the voltage on the VREF pin before attempting to drive outputs.

Absolute maximum tool output voltage is VREF + 20%.

The tool must not draw more than 1 mA of current from the VREF. It is a good idea to put a current-limiting resistor in series with the VREF, but the value should be minimal so as not to degrade the value of the VREF at the tool.

A.3 AC Electrical Characteristics - General

Input rise and fall times are measured at 20% to 80% values.

All setup and hold times are measured from the 50% point of the respective clock edge and the 50% point of the logic signal.

All measurements are made assuming a minimum capacitive loading of 25 pF.

A.4 AC Electrical Characteristics - IEEE 1149.1 Interface

Table A-16 lists the timing constraints for the **IEEE 1149.1** interface.

Figure A-2 gives a pictorial representation of critical timing in **Table A-16**.

Table A-16—Timing Constraints for the IEEE 1149.1 Interface

Number	Characteristic	Min	Max	Unit
1	TCK Cycle Time (**Tc**)	30	—-	ns
2	TCK Duty Cycle	40	60	%
3	Rise and Fall Times (20%–80%)	0	3	ns
4	$\overline{\text{TRST}}$ Setup Time to TCK Falling Edge	(0.30)Tc	—	ns
5	$\overline{\text{TRST}}$ Assert Time	(0.30)Tc	—	ns
6	TMS, TDI Data Setup Time	(0.20)Tc	—	ns
7	TMS, TDI Data Hold Time	(0.10)Tc	—	ns
8	TCK Low to TDO Data Valid	(–0.10)Tc	(0.20)Tc	ns
9	$\overline{\text{EVTO}}$ Pulse Width	(1.0) System Clock	—	ns
10	$\overline{\text{EVTI}}$ Pulse Width	(4.0)Tc	—	ns

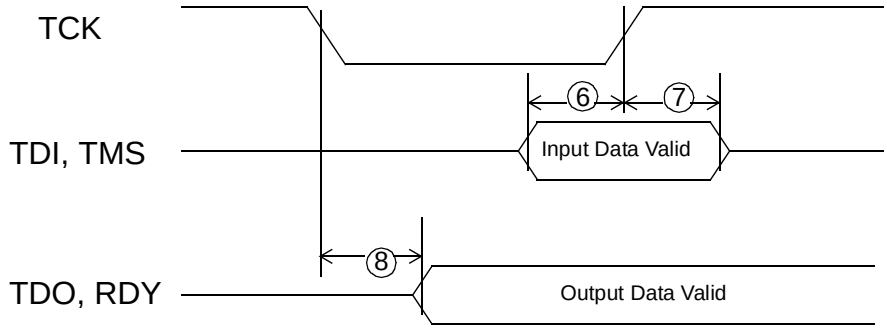


Figure A-2—IEEE 1149.1 Timing Diagram

A.5 AC Electrical Characteristics - AUX

Table A-17 lists the timing constraints for the AUX interface. **Figure A-3** illustrates the critical timing for the clock to data on the input. **Figure A-4** illustrates the critical timing for the clock to data on the output.

Table A-17—Timing Constraints for the AUX Interface

Number	Characteristic	Min	Max	Unit
1	MCKO Cycle Time (**Tco**)	5	—	ns
2	MCKO Duty Cycle	40	60	%
3	Output Rise and Fall Times	0	3	ns
4	MCKO low to MDO Data Valid	(–0.10)Tco	(0.20) Tco	ns
5	MCKI Cycle Time (**Tci**)	5	—	ns
6	MCKI Duty Cycle	40	60	%
7	Input Rise and Fall Times	0	3	ns
8	MDI Setup Time	(0.20)Tci	—	ns
9	MDI Hold TIme	(0.10)Tci	—	ns
10	$\overline{\text{RSTI}}$ Pulse Width	(4.0) Tco	—	ns
11	MCKO low to $\overline{\text{EVTO}}$ Valid	(–0.10)Tco	(0.20) Tco	ns
12	$\overline{\text{EVTI}}$ Pulse Width	(4.0) Tco	—	ns
13	$\overline{\text{EVTI}}$ to $\overline{\text{RSTI}}$ Setup (at reset only)	(4.0) System Clock	—	ns
14	$\overline{\text{EVTI}}$ to $\overline{\text{RSTI}}$ Hold (at reset only)	(4.0) System Clock	—	ns

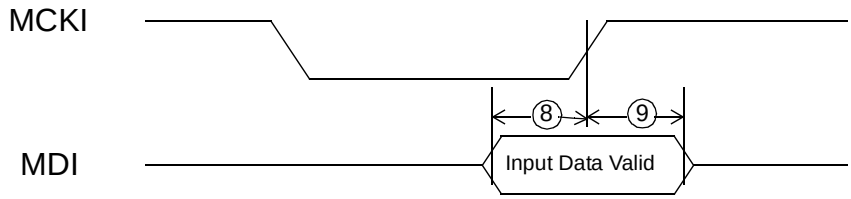


Figure A-3—AUX Data Input Timing Diagram

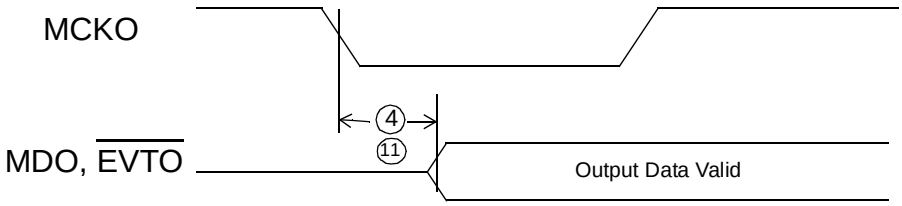


Figure A-4—AUX Data Output Timing Diagram

MDO and $\overline{\text{EVTO}}$ data are held valid until the next MCKO low transition.

When the $\overline{\text{RSTI}}$ pin is asserted, the $\overline{\text{EVTI}}$ pin is used to enable or disable the AUX (see **Figure A-5** and **Figure A-6**). Because MCKO probably is not active at this point, the timing must be based on the system clock. Because the system clock is not realized on the connector, its value must be known by the tool.

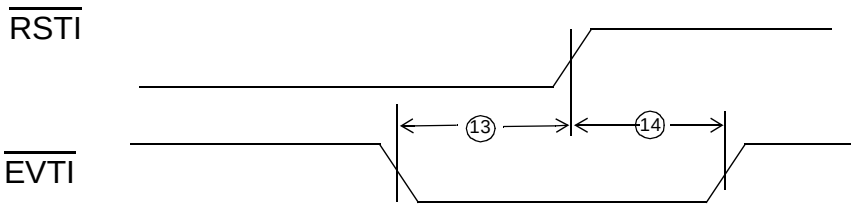


Figure A-5—Enable Auxiliary from $\overline{\text{RSTI}}$ pin

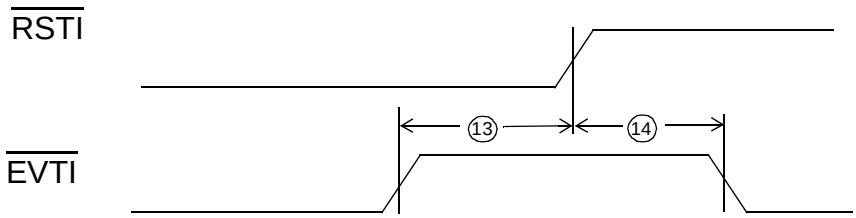


Figure A-6—Disable Auxiliary from $\overline{\text{RSTI}}$ pin

A.6 Terminations

Because of the high-speed natures of the **IEEE 1149.1** port and AUX, it is recommended that the target and tool both employ a point-to-point series termination scheme (see **Figure A-7** and **Figure A-8**).

Z_{out} = Output impedance of the driver
Z_{target} = Impedance of traces on the target printed circuit board
Z_{cable} = Characteristic impedance of cable
Z_{tool} = Impedance of traces on the tool printed circuit board
R_t = Source terminators

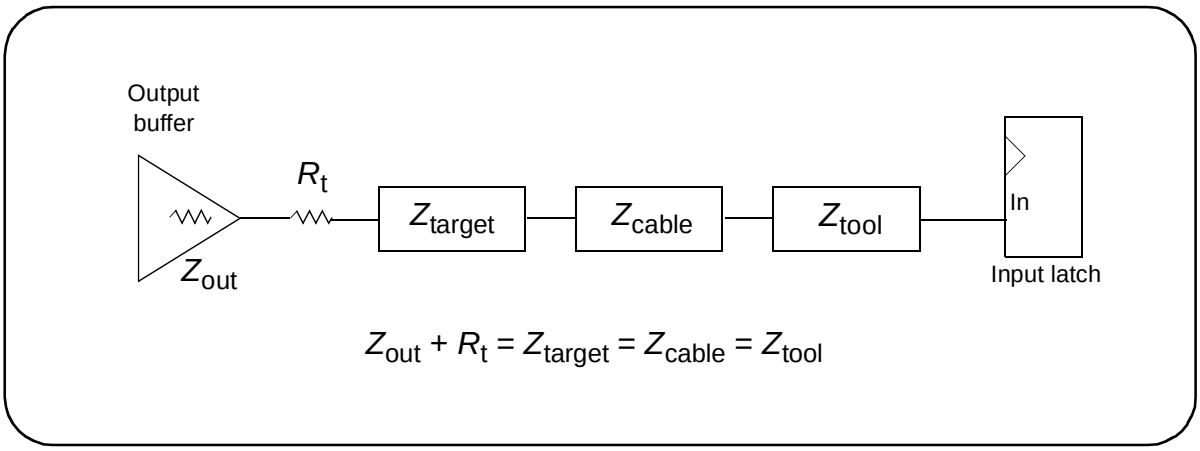


Figure A-7—Target Output Source Termination

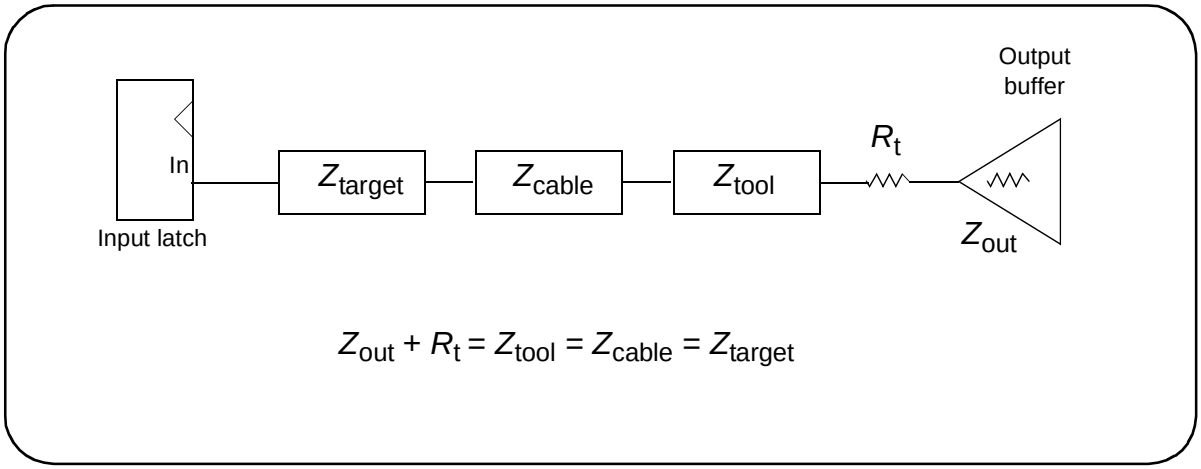


Figure A-8—Tool Output Source Termination

APPENDIX B
Recommendations for Access to Control and Status Registers

Silicon vendors must implement the API requirements for tool software compatibility as described in the Nexus API. In addition, embedded processors complying with Class 2, 3, or 4 are required to implement the AUX message protocol and the required Public Messages as described in **Section 5 - Nexus Public Messages** and **Section 7 - AUX Message Protocol**. There are no requirements in the Nexus standard, however, regarding conformity of development registers that are accessed by tools for control and status.

This appendix contains only recommendations (not requirements) for silicon vendors in implementing development registers that are accessed by tools for control and status.

NOTE

Development tool vendors should not design tools based upon the contents of this appendix. Obtain the silicon vendor's API and product specification for development tool design.

B.1 Overview

The Nexus standard supports development for up to 32 clients[6] on an embedded processor. Each client on embedded processors complying with Class 1 may provide development tool access to control and status according to these recommendations via the **IEEE 1149.1** interface. Each client on embedded processors complying with Class 2, 3, or 4 may provide development tool access to control and status according to these recommendations via either the AUX or the **IEEE 1149.1** interface.

Embedded processors may provide development control and status registers according to **Table B-1**, **Table B-2**, and **Table B-3**. **Table B-1** illustrates the "NEXUS-ACCESS" instruction. Writing an appropriate value to the "NEXUS-ACCESS" instruction, as defined by the silicon vendor, will enable access to NRRs illustrated in **Table B-3**. Additionally, the DID Register information identifies key attributes to the development tool concerning the embedded processor.

[6] Refer to **1.4 - Terms and Definitions**.

In **Table B-2** the Client Select Control (CSC) Register selects one of the clients on the embedded processor for access. Once the client is selected, control and status accesses are directed to the selected client. An alternate client can be selected at any time during operation.

Table B-1—IEEE 1149.1 Register Map for Nexus-Related IR Values

Control/Status	Compliance Class	Access Opcode	Read/ Write
IEEE 1149.1 Public Opcodes	—	Vendor-defined	—
DID Register[a]	All	Vendor-defined	R
NEXUS-ACCESS[b]	All	Vendor-defined	R/W
NEXUS-STATUS[c]	All	Vendor-defined	R

a. The DID Register is defined by **IEEE Std 1149.1-1990**.
b. Only needed for **IEEE 1149.1** port (and not AUX). See **Section 8 - IEEE 1149.1 Message Protocol**.
c. Optional instruction for IEEE 1149.1 port (not AUX). See **8.2.2 - NRR Access Status (Optional)**.

Table B-2—Summary of Nexus Client Registers

Control/Status	Compliance Class	Access Opcode	Read/ Write
DID	All	0	R
CSC	4[a]	1	R/W
Shared by all Nexus Clients	_	2–63	—
Reserved	—	64–127	—
Vendor-defined	—	128–255	—

a. If an embedded processor contains multiple clients, then CSC Register is required (for AUX-only implementations).

The NRR indices as shown in **Table B-3** shall be identical for accesses via the **IEEE 1149.1** interface and the AUX. The fields associated with each opcode accessed via the **IEEE 1149.1** interface shall be identical in size and function to the packets accessed for each opcode via the AUX. The Public Messages in **SECTION 5 - Nexus Public Messages** prescribe the method for accessing recommended control and status registers.

Table B-3 also defines control and status access as indicated per clients of Class 2, 3, or 4 embedded processors. Vendor-defined register space is also provided so that vendor-defined development functions may be implemented. For embedded processors complying with Class 2, 3, or 4, the vendor-defined registers may consist of the transfer registers for interfacing with a processor, e.g., Program Counter and Processor Status.

Table B-3—NRRs for Clients

NRR	Compliance Class	Register Index	Read/ Write
Device ID (DID) (auxiliary only)	All	0	R
Client Select Control (CSC)	2, 3, 4[a]	1	R/W
Development Control (DC)	2, 3, 4	2	R/W
Reserved for Development Control	—	3	—
Development Status (DS)	4	4	R
Reserved for Development Status	—	5	—
User Base Address (UBA)	2, 3, 4	6	R[b]
Read/Write Access Control/Status (RWCS)	3, 4	7	R/W
Reserved for Read/Write Access Control/Status	—	8	—
Read/Write Access Address (RWA)	3, 4	9	R/W
Read/Write Access Data (RWD)	3, 4	10	R/W
Watchpoint Trigger (WT)	4	11	R/W
Reserved for Watchpoint Trigger	—	12	—
Data Trace Control (DTC)	3, 4	13	R/W
Data Trace Start Address (DTSA) (2)	3, 4	14–15	—
Data Trace Start Address (Reserved - 2)	—	16–17	—
Data Trace End Address (DTEA) (2)	3, 4	18–19	—
Data Trace End Address (Reserved - 2)	—	20–21	—
Breakpoint/Watchpoint Control (BWC) (2)	4	22–23	R/W
Breakpoint/Watchpoint Control (Reserved - 6)	—	24–29	—
Breakpoint/Watchpoint Address (BWA) (2)	4	30–31	R/W
Breakpoint/Watchpoint Address (Reserved - 6)	—	32–37	—
Breakpoint/Watchpoint Data (BWD) (2)	4	38–39	R/W
Breakpoint/Watchpoint Data (Reserved - 6)	—	40–45	—
Input Public Message Register (IPMR)	2,3,4	46	R/W
Output Public Message Register (OPMR)	2,3,4	47	R/W
Reserved for future Nexus functionality	—	48–54	—
Re-mapped NRRs (see **B.9 - NRRs Concatenated for Better Transfer Efficiency**)	—	55–63	—
Vendor defined	—	64–127	—
Reserved for future Nexus functionality[c]	—	128–255	—

a. Needed if there are multiple clients on an embedded processor.
b. May also be read/write access for development tool configuration of UBA.
c. **IEEE 1149.1** is not capable of Access Future Reserved.

B.2 Reset

All control and status information shall be reset by one of the following:

- **IEEE 1149.1** "TEST-LOGIC-RESET" state or assertion of $\overline{\text{TRST}}$ pin
- Assertion of $\overline{\text{RSTI}}$ pin (auxiliary only)

No control or status information shall be reset for system reset on the embedded processor.

B.3 Access with the IEEE 1149.1 Interface

The **IEEE 1149.1** state machine is shown in **Figure 8-1**. The value shown adjacent to each arc represents the value of the TMS signal sampled on the rising edge of the TCK signal.

Access to NRRs is enabled by loading a single instruction ("NEXUS-ACCESS") into the **IEEE 1149.1** IR. Once the **IEEE 1149.1** "NEXUS-ACCESS" instruction has been loaded, the **IEEE 1149.1** port allows tool-target communications with all NRRs according to the register index in **Table B-3**.

Reading/writing of an NRR then requires two passes through the Data-Scan path of the **IEEE 1149.1** state machine.

1. The first pass through the Data Register (DR) selects the NRR to be accessed by providing an index (see **Table B-3**) and the direction (read or write). This is achieved by loading an 8-bit value into the **IEEE 1149.1** DR. This register has the following format:

Read/Write:

0 = Read
1 = Write

NRR Address:

Selected from values in **Table B-3**

2. The second pass through the DR then shifts the data in or out of the **IEEE 1149.1** port, LSB first.

    a. During a read access, data are latched from the selected NRR when the **IEEE 1149.1** state machine passes through the CAPTURE-DR state.

b. During a write access, data are latched into the selected NRR when the **IEEE 1149.1** state machine passes through the UPDATE-DR state.

B.4 Access with the AUX

The control and status registers are accessed via the four Public Messages—NRR Access - Target Ready, NRR Access - Read Register (tool requests information), NRR Access - Write Register (tool provides information), and NRR Access - Read/Write Response (from tool or target).

To write control or status information, the following sequence would be required:

- If a prior NRR Access - Target Ready Message was transmitted by the target, then the tool transmits a NRR Access - Write Register Message, which contains write attributes and a register value to be written.
- The tool waits for the NRR Access - Target Ready Message before initiating the next access.

To read control or status information, the following sequence would be required:

- If a prior NRR Access - Target Ready Message was transmitted by the target, then the tool transmits a NRR Access - Read Register Message, which contains read attributes.
- When the target reads data, it transmits a NRR Access - Read/Write Response Message containing read data. The target is now ready for the next access.

B.5 NRRs - Control and Status

This section describes the fields composing each control and status register. The control registers in this section are organized so that the most used bits are located in the MSBs of the registers. This allows for short write sequences from the tool to write only a few bit fields.

For many of the control and status opcodes defined in this section, there are bits reserved as vendor defined. Vendor-defined development features and operations may be included in these designated bits. For tools not implementing these vendor-defined development features, the fields should not be written or set to a value of 0. The setting of 0 is designated as the default state.

B.5.1 Device ID (DID) Register

Accessing the DID Register provides key attributes to the development tool concerning the embedded processor. This information assists the development tool in determining configuration and features of the device. For Classes 2, 3, and 4 embedded processors, this information is also transmitted via the AUX OUT upon exit of AUX reset.

For embedded processors with a full AUX, the DID Register shown in **Table B-4** should be implemented in compliance with the register organization and bit field definitions as specified in **IEEE Std 1149.1-1990**. For embedded processors with an **IEEE 1149.1** interface used for the Nexus standard, the DID Register defined by **IEEE Std 1149.1-1990** must be implemented. In this case, the DID Register defined in this subsection is not necessary.

The fields include embedded processor information containing the manufacturer ID, product number, and revision number. In general, the revision number must be changed (i.e., incremented) whenever the embedded processor has a mask revision that will disrupt the tools in any manner (see **Table B-4**).

Table B-4—DID Register

Bit Number	Field Name	Description
31–28	RN	Revision Number
27–12	PN	Product Number
11–1	MID	Manufacturer ID
0	—	Reserved

B.5.2 Client Select Control (CSC) Register

The CSC Register contains a single 5-bit field that, when written to, selects the client to be accessed via the **IEEE 1149.1** interface or the AUX. The encodings of the CSC Register are vendor defined. The setting of the CS field selects which client is accessed for access opcodes 1–127.

The CSC Register is recommended if there are multiple clients on the embedded processor (see **Table B-5**).

Table B-5—CSC Register

Bit Number	Field Name	Description
7–5	—	Reserved
4–0	CS	Client select

B.5.3 Development Control (DC) Register

The DC Register is used for basic development control of a client. The debug enable (DBE) field enables debug mode, and the debug request (DBR) field allows for a software mechanism to enter debug mode. If debug mode is enabled, then asserting DBR, power-on reset, or an exception may cause the processor to halt and enter debug mode. Enabling debug mode is necessary to use features such as single-stepping and breakpoints.

The trace mode (TM) field enables BTM, DTM, and OTM. One or all types of trace may be enabled by TM or via a watchpoint occurrence (refer to **B.5.6 - Watchpoint Trigger (WT) Register**).

If the $\overline{\text{EVTI}}$ control (EIC) field = 00 and program and/or data trace are enabled, a high-to-low transition on $\overline{\text{EVTI}}$ will cause program and/or data trace synchronization, respectively. If EIC = 01, a high-to-low transition on $\overline{\text{EVTI}}$ will cause a breakpoint to occur. If EIC = 10, no operation will occur regardless of the state on $\overline{\text{EVTI}}$.

The memory substitution (MS) and step enable (SS) bit fields determine how the processor will operate when DBR is negated. If MS = SS = 0, then normal operation will commence when DBR is negated. If MS = 0 and SS = 1, then a single step will occur when DBR is negated with internal memory access. If MS = 1 and SS = 0, then operation will commence when DBR is negated with instruction/data access via the AUX. If MS = SS = 1, then a single step will occur when DBR is negated with instruction/data access via the AUX.

When MS = 1, the state of the substitution operand (SO) bits determines which combination of instruction and data accesses are substituted so that memory accesses are made via the AUX or **IEEE 1149.1** interface. If MS = 0, memory substitution is not enabled, and memory accesses are made to the target memory system.

The overrun control (OVC) field is used to determine control for overrun of BTM and DTM. Overruns can be handled by displaying an Overrun Message to development tools, delaying the processor to avoid BTM overruns, delaying the processor to avoid DTM overruns, or delaying the processor to avoid both BTM and DTM overruns.

The client breakpoint input (CBI) bit is an optional control bit that, when enabled, gates a global, wired-OR breakpoint signal to the client. When the global breakpoint signal is asserted and the CBI bit is asserted, it causes a breakpoint to occur on the client. Each client should also wire-OR its breakpoint status output to this global breakpoint signal. When CBI is negated, the client will only break for breakpoint conditions internal to the client.

For embedded processors complying with Class 2 or 3, the only development control field required is TM. For this case all other fields except the vendor-defined field shall be reserved and contain the same number of bits (see **Table B-6**).

Table B-6—DC Register

Bit Number	Field Name	Description
31–24	—	Vendor-defined
23–15	—	Reserved
14	CBI	CBI - Client Breakpoint Input (optional) 0 = Break for internal breakpoints only 1 = Break for other clients' breakpoints also
13	DBE	DBE - Debug Enable (Class 4) 0 = Debug mode disabled 1 = Debug mode enabled
12	DBR	DBR - Debug Request (Class 4) 0 = Exit debug mode 1 = Request debug mode
11	MS	MS - Memory Substitution (Class 4) 0 = Use instructions and data in target memory 1 = Access instruction/data through AUX
10–9	SO	SO - Substitution Operands (Class 4) 00 = Instructions and data 01 = Instructions only 10 = Data only 11 = Reserved
8	SS	SS - Step Enable (Class 4) 0 = Single-step disabled 1 = Single-step enabled
7–5	OVC	OVC - Overrun Control (Class 4) 000 = Generate overrun messages 001 = Delay processor for BTM overruns 010 = Delay processor for DTM and OTM overruns 011 = Delay processor for BTM, DTM and OTM overruns 100–111 = Reserved
4–3	EIC	EIC - $\overline{\text{EVTI}}$ Control (Class 2, 3, 4) 00 = $\overline{\text{EVTI}}$ for program and data trace synchronization 01 = $\overline{\text{EVTI}}$ for breakpoint generation 10 = No operation 11 = Reserved
2–0	TM	TM - Trace Mode (Class 2, 3, 4) 000 = No Trace 1XX = BTM Enabled X1X = DTM Enabled XX1 = OTM Enabled

B.5.4 Development Status (DS) Register

When debug mode is entered the condition is detected by reading the debug status (DBS) bit in the DS Register or by observing the Debug Status Message on the auxiliary pins. The single-step status (SSS) field will also be set if debug mode is entered after a single step. The hardware breakpoint (HWB) field and software breakpoint (SWB) field also indicate whether a hardware breakpoint (e.g., address comparator) or a software breakpoint (e.g., breakpoint instruction) caused the processor to halt and enter debug mode. The breakpoint status (BPn) bits indicate which breakpoint occurred.

Other conditions that may impact development support are detecting when the processor is in a low-power mode or when a nonrecoverable hardware error has occurred. A stop (STP) and hardware error (HWE) bit may be implemented to indicate these conditions.

The DS Register is read-only. All status bits are dynamic and do not require clearing. This register is recommended for embedded processors complying with Class 4. The contents of the DS Register are transmitted out the auxiliary pins upon a change in state of any bit (see **Table B-7**).

Table B-7—DS Register

Bit Number	Field Name	Description
31–24	—	Vendor-defined
23–16	—	Reserved
15–8	BP7-0	BPn - Breakpoint Status 0 = No breakpoint 1 = Breakpoint occurred
7	—	Reserved
6	RSTS	RSTS - Reset Status 0 = Processor not reset 1 = Processor reset since last DS Register read
5	DBS	DBS - Debug Status 0 = Processor not halted 1 = Processor halted in debug mode
4	STP	STP - Stop Status 0 = Processor not stopped 1 = Processor stopped in low-power mode
3	HWE	HWE - Hardware Error 0 = No hardware error 1 = Nonrecoverable hardware error occurred
2	HWB	HWB - Hardware Breakpoint Status 0 = No hardware breakpoint 1 = Hardware breakpoint

Table B-7—DS Register *(Continued)*

Bit Number	Field Name	Description
1	SWB	SWB - Software Breakpoint Status 0 = No software breakpoint 1 = Software breakpoint
0	SSS	SSS - Single-Step Status 0 = Processor not halted 1 = Processor halted in debug mode after single step

B.5.5 User Base Address (UBA) Register

The UBA Register provides visibility for the development tool to determine what the setting is for the vendor-defined user base address. The UBA Register is the memory map base address for user access to specific resources of the Nexus development port. If needed, the UBA Register may be writable by the development tool to configure the memory map base address for user access.

User access to the Nexus development port is required for OTM and DQM and reserved for other uses. The size of the UBA Register is vendor defined (see **Table B-8**).

Table B-8—UBA Register

Bit Number	Packet Name	Description
Vendor defined	UBA	Vendor-defined user base address

The memory map for user access of development features is shown in **Table B-9**, where offset is the base word size of the embedded processor.

Table B-9—Memory Map for User Accesses

Memory Map Location	Description
(UBA) + 2 x Offset	Reserved for future use
(UBA) + 1 x Offset	Reserved for future use
(UBA)	Ownership Trace Register (OTR)
(UBA) – 1 x Offset	Data Acquisition Control
(UBA) – N x Offset	Location for Date Acquisition Messages where N is data ID

The UBA Register is recommended for embedded processors complying with Class 2, 3, or 4.

B.5.5.1 Ownership Trace Register (OTR)

The OTR shall be provided only for general-purpose processor clients of embedded processors complying with Class 2, 3, or 4. The OTR provides a register to which an operating system can write an ID for the current task/process. The size of the OTR is vendor defined.

B.5.5.2 Data Acquisition Messaging (DQM)

DQM is achieved by user writes to appropriate locations in the memory map shown in **Table B-9**. The write information is queued up for messaging via the auxiliary pins. The location in the DQM portion of the UBA Register map to which data are written determines the data ID tag for the message, with the exception of the Data Acquisition Control, which is used for DQM queue control.

DQM data written to a location in the UBA Register map are queued up until a value of 0x0 is written to Data Acquisition Control, at which point the ID tag and data values are transferred on the auxiliary pins (refer to **4.10 - Data Acquisition (Optional)**). A Data Acquisition Message transfer is also started if the message queue fills up or if another location in the DQM portion of the UBA Register map is written to before 0x0 is written to Data Acquisition Control. If the queue fills up before 0x0 is written to the address pointed to by the UBA Register, subsequent writes to locations in the DQM portion of the UBA Register map will be stalled until queue space becomes available.

For simplicity of hardware implementation, Data Acquisition Message IDs will be 2 bits or greater.

B.5.6 Watchpoint Trigger (WT) Register

The WT Register allows the watchpoints defined in the breakpoint/watchpoint registers (refer to **B.8 - NRRs - Breakpoint/Watchpoint**) to be assigned to trigger actions. The program trace start (PTS) and program trace end (PTE) fields select watchpoints to enable and disable program trace, effectively producing an address- and/or data-related “window” for triggering program trace. The data trace start (DTS) and data trace end (DTE) fields select watchpoints to enable and disable data trace, effectively producing an address- and/or data-related “window” for triggering data trace. Program and/or data trace is triggered via the WT setting if the TM field (refer to **B.5.3 - Development Control (DC) Register**) has not already enabled program and/or data trace.

The memory substitution start (MSS) field selects a watchpoint to trigger memory substitution. (See **Table B-10.**) Refer to **B.5.3 - Development Control (DC) Register** for additional fields related to memory substitution.

The WT Register is recommended for embedded processors complying with Class 4.

Table B-10—WT Register

Bit Number	Field Name	Description
31–29	PTS	PTS - Program Trace Start 000 = Trigger disabled 001 = 111 Use watchpoint 1–7
28–26	PTE	PTE - Program Trace End 000 = Trigger disabled 001 = 111 Use watchpoint 1–7
25–23	DTS	DTS - Data Trace Start 000 = Trigger disabled 001 = 111 Use watchpoint 1–7
22–20	DTE	DTE - Data Trace End 000 = Trigger disabled 001 = 111 Use watchpoint 1–7
19–17	MSS	MSS - Memory Substitution Start 000 = Trigger disabled 001 = 111 Use watchpoint 1–7
16–8	—	Reserved
7–0	—	Vendor-defined

B.6 NRRs - Read/Write Access

The Read/Write Access feature provides DMA-like access to internal memory-mapped resources when the client is halted or during runtime. Three registers are used for the Read/Write Access feature:

- Read/Write Access Control/Status (RWCS)
- Read/Write Access Address (RWA)
- Read/Write Access Data (RWD)

The tool will write to a user memory map location first by updating the RWA Register and RWD Register with the user address and data to be written and then by updating the RWCS Register with the write access attributes. The tool will read from a user memory map location first by updating the RWA Register with the user address to be read and then by updating the RWCS Register with the read access attributes.

These registers are recommended for embedded processors complying with Class 3 or 4.

More detailed information on using the Read/Write Access feature is included in **B.6.1 - Access with the IEEE 1149.1 Interface** through **B.6.5 - RWD Register**.

B.6.1 Access with the IEEE 1149.1 Interface

The **IEEE 1149.1** state machine is shown in **Figure 8-1**. The value shown adjacent to each arc represents the value of the TMS signal sampled on the rising edge of the TCK signal.

1. For a block read the following sequence would be required:
    a. Initialize the RWA Register through the **IEEE 1149.1** access method outlined in **B.3 - Access with the IEEE 1149.1 Interface** using the NRR index of 9 (see **Table B-3**).
    b. Initialize the RWCS Register through the **IEEE 1149.1** access method outlined in **B.3 - Access with the IEEE 1149.1 Interface** using the NRR index of 7 (see **Table B-3**).
    c. The read data will then be transferred to the RWD Register. When completed (without error), the Nexus block decrements the number in the CNT field and sets the DV bit. Setting the DV bit indicates that the device is ready for the next access.
    d. The data can then be read from the RWD Register through the **IEEE 1149.1** access method outlined in **B.3 - Access with the IEEE 1149.1 Interface** using the NRR index of 10 (see **Table B-3**).
    e. Once the RWD value has been read, the RWA Register will then be incremented to the next word of size SZ and Step 1c will be repeated. When the CNT field reaches a value of 0, the AC bit is cleared indicating the end of the read access.
2. For a block write the following sequence would be required:
    a. Initialize the RWA Register through the **IEEE 1149.1** access method outlined in **B.3 - Access with the IEEE 1149.1 Interface** using the NRR index of 9 (see **Table B-3**).
    b. Initialize the RWD Register through the **IEEE 1149.1** access method outlined in **B.3 - Access with the IEEE 1149.1 Interface** using the NRR index of 10 (see **Table B-3**).
    c. Initialize the RWCS Register through the **IEEE 1149.1** access method outlined in **B.3 - Access with the IEEE 1149.1 Interface** using the NRR index of 7 (see **Table B-3**).
    d. The Nexus block will then transfer the data value from the RWD Register to the memory-mapped address in the RWA Register.

When completed (without error), the Nexus block decrements the number in the CNT field and clears the DV bit. Clearing the DV bit indicates that the device is ready for the next access.

e. Repeat Step 2b until the CNT field has a value of 0. When this value is reached, the AC bit will be cleared indicating the end of the write access.

B.6.2 Access with the AUX

Use the NRR Access Public Messages described in **5.3.24 - NRR Access - Target Ready Message** through **5.3.27 - NRR Access - Read/Write Response Message**.

B.6.3 RWCS Register

The word size (SZ), read/write (RW), priority (PR), map select (MAP), and access count (CNT) fields are written to by the tool to set up access attributes. The access control (AC) field is asserted by the tool to initiate an access or is negated by the tool to cancel an access in progress. The AC field is negated by the embedded processor upon completion of the access requested by the tool.

The SZ and RW bits determine the access size and whether it is a read or a write. The PR bits are intended to allow for implementations that perform a variety of access priorities, from a lowest intrusive access (0b00) to a highest intrusive access (0b11). The exact meaning of the encodings are vendor defined.

The MAP bits are intended to allow for multiple memory maps to be accessed. The primary processor memory map should be designated as the default (MAP = 000). Secondary memory maps, such as special-purpose processor memory maps, which are implemented in some processor architectures, may also require access.

To request a block move, the CNT field is set by the tool to a value greater than 0. The address range for a block move is from RWA to RWA + CNT. The CNT field should not be decremented by the embedded processor during an in-progress block move. Upon completion of a block move, the embedded processor should negate the AC field and set the CNT field to a value of 0.

If the RWCS Register is written to while any single or block access is in progress, the target will terminate the access, including any remaining block accesses, within one access cycle of the target. In this case, the access in progress when the RWCS Register is written to is not guaranteed to complete (see **Table B-11**).

Table B-11—Read/Write Access Status Bit Encoding

DV	ERR	Read Action	Write Action
0	0	Read Access has not completed	Write Access completed without error
0	1	Read Access error has occurred	Write Access error has occurred
1	0	Read Access completed without error	Write Access has not completed
1	1	Not Allowed	Not allowed

If an error is generated during a block access, the block access will be terminated (see **Table B-12**).

Table B-12—Read/Write Block Access

Bit Number	Field Name	Description
31	AC	AC - Access Control 0 = End Access 1 = Start access
30	RW	RW - Read/Write 0 = Read access 1 = Write access
29–27	SZ	SZ - Word Size 000 = 8-bit 001 = 16-bit 010 = 32-bit 011 = 64-bit 1xx = Reserved
26–24	MAP	MAP - Map Select 000 = Primary memory map 001–111 = Other memory maps
23–22	PR	PR - Priority bb = Access priority
21–16	—	Reserved
15–2	CNT	CNT - Access Count hhhh = Number of accesses of word size SZ
1	ERR	Last access generated an error
0	DV	Data valid in RWD

B.6.4 RWA Register

The RWA Register is used by the tool to program the address of user memory-mapped resource to be accessed or the lowest address (i.e., lowest unsigned value) for a block move (CNT > 0). The address range for a block move is from RWA to RWA + CNT.

The size of the RWA Register is vendor defined (see **Table B-13**).

Table B-13—RWA Register

Bit Number	Packet Name	Description
Vendor-defined	RWA	User memory-mapped address to be accessed

B.6.5 RWD Register

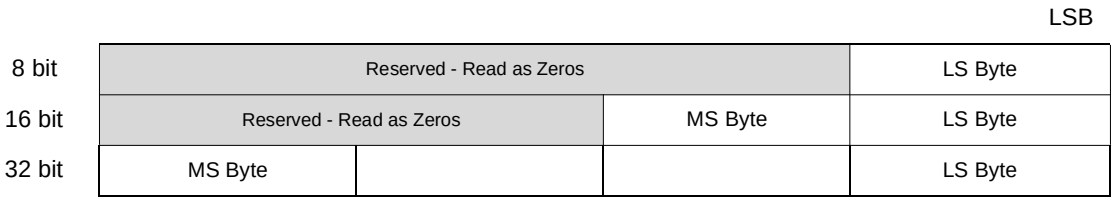
The RWD Register is used to contain the data to be written for the next block write access and the read data for completed read accesses.

The size of the RWD Register is vendor defined (see **Table B-14**).

Table B-14—RWD Register

Bit Number	Packet Name	Description
Vendor-defined	RWD	Data read from a user memory-mapped location or to be written to a user memory-mapped location

For read and write accesses, the register may contain different sizes of data. The following is the organization for three different sizes of data.



B.7 NRRs - Data Trace

The data trace registers allow DTM to be restricted to reads, writes, or both for a programmable user address range. Three registers are used for selecting the data trace attributes:

- Data Trace Control (DTC)
- Data Trace Start Address (DTSA)
- Data Trace End Address (DTEA)

These registers are recommended for embedded processors complying with Class 3 or 4.

B.7.1 DTC Register

Read/Write trace (RWTn) bits select for each data trace channel (up to six data trace channels) if no trace messages are generated or if reads, writes, or both generate Data Trace Messages. If the RWTn bit selects data trace for reads and/or writes, all selected accesses within the address range specified by DTSA to DTEA, end points inclusive, will generate Data Trace Messages (see **Table B-15**).

Table B-15—DTC Register

Bit Number	Field Name	Description
31–30	RWT0	RWT0 - Read/Write Trace 0 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
29–28	RWT1	RWT1 - Read/Write Trace 1 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
27–26	RWT2	RWT2 - Read/Write Trace 2 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
25–24	RWT3	RWT3 - Read/Write Trace 3 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
23–22	RWT4	RWT4 - Read/Write Trace 4 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
21–20	RWT5	RWT5 - Read/Write Trace 5 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
19–8	—	Reserved
7–0	—	Vendor-defined

B.7.2 DTSA Register and DTEA Register

The DTSA Register and DTEA Register are used by the tool to program the start and end addresses for a data trace channel. If RWTn selects data trace for reads and/or writes, all selected accesses within the address range specified by DTSA to DTEA, end points inclusive, will generate Data Trace Messages.

The size of the DTSA Register and DTEA Register is device specific (see **Table B-16** and **Table B-17**).

Table B-16—DTSA Register

Bit Number	Packet Name	Description
Vendor-defined	DTSA	Start address for data trace visibility

Table B-17—DTEA Register

Bit Number	Packet Name	Description
Vendor-defined	DTEA	End address for data trace visibility

B.8 NRRs - Breakpoint/Watchpoint

The breakpoint/watchpoint registers provide control for breakpoint and watchpoint logic. Three registers are used for controlling the breakpoints/watchpoints:

- Breakpoint/Watchpoint Control (BWC)
- Breakpoint/Watchpoint Address (BWA)
- Breakpoint/Watchpoint Data (BWD)

These registers are recommended for embedded processors complying with Class 4.

B.8.1 BWC Register

For all breakpoints to be enabled, the DBE field must be set to enable debug mode (refer to **B.5.3 - Development Control (DC) Register**). When debug mode is enabled, individual breakpoints can be enabled with the breakpoint/watchpoint enable (BWE) field. Watchpoints are enabled with the BWE field regardless of the state of the DBE field.

The breakpoint/watchpoint read/write select (BRW) field determines whether a read, write, or any access will cause a breakpoint. The breakpoint/watchpoint address/data mask enable (BME) field selects the data-mask-enable to be on a particular byte, half-word (2-byte), or word (4-byte) lane. Because the breakpoint data size unit is device specific, the breakpoint/watchpoint data size unit (BSU) field is read-only to indicate to the tool if the data size unit is 1 byte, 2 bytes, or 4 bytes. For example, with 32-bit machines, the 4 MSBs of the BME field may be reserved and the LSBs may be used to enable masking of byte lanes (assuming BSU = 00). The breakpoint/watchpoint operand (BWO) field indicates whether the

BWA Register and/or the BWD Register is used for the breakpoint condition, and the breakpoint/watchpoint type (BWT) field selects the breakpoint operand as instruction or data.

The $\overline{\text{EVTO}}$ control (EOC) field selects if the breakpoint status indication is output on the $\overline{\text{EVTO}}$ pin (see **Table B-18**).

Watchpoints can be assigned actions listed in **Table B-10.**

If logical conditions of breakpoint or watchpoint detections are needed or if counting *N* watchpoints is needed for development, the vendor-defined field can be defined to provide these or other features.

Table B-18—BWC Register

Bit Number	Field Name	Description
31–30	BWE	BWE - Breakpoint/Watchpoint Enable 00 = Disabled 01 = Breakpoint enabled 10 = Reserved 11 = Watchpoint enabled
29–28	BRW	BRW - Breakpoint/Watchpoint Read/Write Select 00 = Break on read access 01 = Break on write access 10 = Break on any access 11 = Reserved
27–20	BME	BME - Breakpoint/Watchpoint Address/Data Mask Enable 1XXXXXXX = Mask MS address/data size unit ⋮ XXXXXXX1 = Mask LS address/data size unit
19–18	BSU	BSU - Breakpoint/Watchpoint Data Size Unit (read only) 00 = Data size unit is 1 byte 01 = Data size unit is 2 bytes 10 = Data size unit is 4 bytes 11 = Reserved
17–16	BWO	BWO - Breakpoint/Watchpoint Operand 1X = Compare with BWA value X1 = Compare with BWD value
15	BWT	BWT - Breakpoint/Watchpoint Type 0 = Compare for instruction types 1 = Compare for data types
14	EOC	EOC - $\overline{\text{EVTO}}$ Control (optional) 0 = Breakpoint/watchpoint status indication not output on $\overline{\text{EVTO}}$ 1 = Breakpoint/watchpoint status indication is output on $\overline{\text{EVTO}}$
13–8	—	Reserved
7–0	—	Vendor-defined

B.8.2 BWA Register

The BWA Register is used to compare against address operands (address of instruction or data). The size of the BWA Register is vendor defined (see **Table B-19**).

Table B-19—BWA Register

Bit Number	Packet Name	Description
Vendor-defined	BWA	Address of instruction or data for breakpoint or watch-point generation

B.8.3 BWD Register

The BWD Register is used to compare against data operands (instruction opcode or data value). The size of the BWD Register is vendor defined (see **Table B-20**).

Table B-20—BWD Register

Bit Number	Packet Name	Description
Vendor-defined	BWD	Instruction opcode or data value for breakpoint or watchpoint generation

B.9 NRRs Concatenated for Better Transfer Efficiency

The NRRs may be concatenated as shown in **Table B-21** for better efficiency of transfers between the target and tool. For example, performing writes to configure the RWCS Register, RWA Register, and RWD Register to write a value to a user memory-mapped location requires only one Write Register Message instead of three (one for each register).

Table B-21—NRRs Concatenated

Concatenated NRRs	Register Index	Read/ Write
RWCS || RWA || RWD	55	R/W
BWC0 || BWA0 || BWD0	56	R/W
BWC1 || BWA1 || BWD1	57	R/W
BWC2 || BWA2 || BWD2	58	R/W
BWC3 || BWA3 || BWD3	59	R/W
BWC4 || BWA4 || BWD4	60	R/W
BWC5 || BWA5 || BWD5	61	R/W
BWC6 || BWA6 || BWD6	62	R/W
BWC7 || BWA7 || BWD7	63	R/W

For the Write Register Message or Read/Write Response Message, the REGVAL packet will contain the right-most register (LSB first), followed by the center register (LSB first), followed by the left-most register of **Table B-21**.

APPENDIX C
Data Acquisition in Tuning for Applications

C.1 Additional Needs for Automotive Powertrain and Disk Drive Development

The development cycle for mechanical and electro-mechanical control applications includes additional needs for calibration of mechanical performance-related constants that are tuned for specific loads. The calibration process is performed during runtime. For calibration, the basic needs for development tools are

1. To acquire during mechanical operation (e.g., running an engine) rotational position synchronous data relating to calibration factors as they are being used or modified during high-speed transient events. Data acquisition should be accomplished with minimal impact to the system under development.

2. To acquire during mechanical operation, time synchronous data relating to calibration factors as they are being used or modified during high-speed transient events. Data acquisition should be accomplished with minimal impact to the system under development.

3. To coherently modify table(s) of calibration constants during mechanical operation.

Refer to the white paper, *The Evolution of Powertrain Microcontrollers and its Impact on Development Processes and Tools,* for more information on automotive powertrain development needs. A copy of the paper can be found on the Nexus web site, http://www.nexus5001.org/microcontrollers_evolution.pdf.

For applications such as automotive powertrain, disk drive control, and wireless, visibility of selected program variables (called calibration variables) must be provided to enable accurate tuning of selected program constants (called calibration constants). When calibration variables are stored in internal RAM, the data must be acquired from the embedded processor during runtime. Additionally, when calibration constants are stored in internal ROM, these constants must be tuned during runtime to determine the optimal values.

C.2 Data Acquisition or Measurement of Calibration Variables

Two options are explained in **C.2.1 - DTM Option** and **C.2.2 - Read/Write Access Option** to meet the data acquisition needs discussed in **C.1 - Additional Needs for Automotive Powertrain and Disk Drive Development**. The first utilizes DTM and the second utilizes the Read/Write Access feature.

C.2.1 DTM Option

One technique to accomplish data acquisition would be to set up a data trace window for all internal embedded processor memory-mapped locations that require acquisition. Depending upon the application, this window may include non-calibration data. Coherency (i.e., demarcating old data from new data) would be provided with a specific embedded processor data write sequence or with a watchpoint occurrence and message. Care should be taken to assure that the data trace bandwidth requirements do not exceed the performance capability of the AUX.

Alternately, the embedded processor could queue up calibration variables for acquisition by the development tool by writing them to contiguous locations in a data trace window, e.g., contiguous locations in system RAM. Dedicated locations in the data trace window would be used to distinguish each group of calibration variables. Coherency would be provided with a specific embedded processor data write sequence or with a watchpoint occurrence and message. Again, care should be taken to assure that the data trace bandwidth requirements do not exceed the performance capability of the AUX.

C.2.2 Read/Write Access Option

Another technique to accomplish data acquisition would be to designate contiguous locations in a system RAM for all calibration variables. Calibration variables would be copied by the embedded processor from the source to these RAM locations prior to acquisition by the tool. The use of a specific embedded processor data write sequence or of a watchpoint occurrence and message signals the tool to acquire the calibration variables. The tool would acquire the calibration variables using the Read/Write Access feature.

C.3 Tuning of Calibration Constants

The Nexus standard provides features to support program execution tuning, also referred to as calibration constant tuning, which is required to tune electro-mechanical systems for a variety of loads, such as for automotive powertrain and disk drive applications.

The Nexus standard provides download capability for calibration constants to be tuned during runtime using a vendor-defined tuning block internal to the embedded processor. The Read/Write Access feature provides access to vendor-defined blocks, either via the **IEEE 1149.1** interface or the auxiliary pin interface, when the processor is halted or running. The auxiliary pin interface may be preferred for better performance capability, e.g., if simultaneous tuning and rapid prototyping are required.

Prior art solutions used as the vendor-defined tuning block include a bondout version of the embedded processor that allows an external RAM to overlay calibration constants in the internal ROM. The overlay RAM is accessible by the development tool. To provide coherency of modifications from the development tool, the overlay may comprise two identical RAMs, which are alternately enabled for overlay. The disabled RAM would be available to the tool for the latest tuning information and would then be swapped in. For this prior art, all accesses could be managed by the development tool via the AUX.

APPENDIX D
Bibliography

IEEE Std 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture (includes IEEE Std 1149.1a-1993).

The Evolution of Powertrain Microcontrollers and its Impact on Development Processes and Tools, Motorola/Hewlett Packard white paper.

Proposal to Extend the IEEE-ISTO 5001 Global Embedded Processor Debug Interface Standard to Incorporate Software Quality Assurance Capability, Hugh O'Keeffe, Ashling Microsystems Ltd.

Proposal to Extend the IEEE-ISTO 5001 GEPDIS (Global Embedded Processor Debug Interface Standard), Jean-Francis Duret, ST Microelectronics

Comments about Nexus Trace, Laurent Regnier, ST Microelectronics

Nexus Extensions, Steve Allen, Alphamosiac

IEEE-ISTO 5001 Change Proposal, Bryan Weston, Motorola

