

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 3.0



01 June 2012

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. Version 3.0 was approved by the membership and published on 01 June 2012.

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.

© 2012, IEEE Industry Standards and Technology Organization. All rights reserved.
The IEEE-ISTO is affiliated with the IEEE and the IEEE Standards Association.
IEEE-ISTO 5001 and Nexus 5001 Forum are trademarks of the IEEE-ISTO.

Copyright

This document may be copied and furnished to others, and derivative works that comment on, or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice, this paragraph and the title of the Document as referenced below are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the IEEE-ISTO and the Nexus 5001 Forum™.

Title: The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface,
Version 3.0

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

SECTION 1

Introduction

1.1	Overview	1
1.2	Development Needs for Embedded Processors	2
1.2.1	Requirements for run-control:.....	2
1.2.2	Requirements for trace analysis:	2
1.2.3	Requirements for (cost-effective) development tools:	2
1.3	Terms and Definitions	3
1.4	Conventions	5
1.5	Other Terminology within the Nexus Standard.....	5

SECTION 2

Compliance and Performance Classifications

2.1	Compliance Classification	7
2.1.1	Compliance Sub-Class for Application-Specific Development Needs.....	9
2.2	Performance Classification.....	9
2.2.1	Interpreting Performance Classification	10

SECTION 3

Nexus Development Features

3.1	Development Control and Status.....	11
3.1.1	Overview	11
3.1.2	Nexus Recommended Registers (NRRs).....	11
3.2	Ownership (Process ID) Trace	11
3.2.1	Overview	11
3.2.2	Ownership Trace Messaging (OTM)	12
3.3	Program Trace	12
3.3.1	Overview	13
3.3.2	Branch Trace Messaging (BTM)	13
3.3.3	Program Trace Errors.....	14
3.3.4	Program Trace Synchronization.....	14
3.4	Data Trace.....	15
3.4.1	Overview	16
3.4.2	Data Trace Messaging (DTM)	16
3.4.3	Data Trace Errors.....	16
3.4.4	Data Trace Synchronization	16
3.5	Breakpoints/Watchpoints.....	17
3.5.1	Overview	17
3.5.2	Breakpoint/Watchpoint Messaging.....	17
3.6	Read/Write Access.....	18
3.6.1	Overview	18
3.6.2	Read/Write Access Messaging	18
3.7	Memory Substitution.....	19
3.7.1	Overview	19
3.7.2	Memory Substitution Messaging (MSM)	20
3.8	In-Circuit Trace (Optional)	21

3.8.1	Overview	21
3.8.2	In-Circuit Trace Messaging (ICT)	21
3.8.3	In-Circuit Trace Synchronization	21
3.9	Vendor Defined Trace (Optional)	22
3.10	Port Replacement and Port Sharing (Optional)	22
3.10.1	Overview	22
3.10.2	Port Replacement for Low-Speed I/O (LSIO) Pins	22
3.10.3	Port Replacement for High-Speed I/O (HSIO) Pins (Port Sharing)	24
3.11	Data Acquisition (Optional)	24
3.11.1	Overview	25
3.11.2	Data Acquisition Messaging (DQM)	25
3.12	Timestamping (Optional)	25
3.12.1	Overview	25
3.12.2	Timestamping via AUX	26
3.12.3	Timestamping via pins	26

SECTION 4

Nexus Public Messages

4.1	Compliance Requirements for Public Messages	27
4.2	Definitions and Terminology	29
4.3	Detailed Description of Public Messages	35
4.3.1	Debug Status Message	36
4.3.2	Device ID Message	37
4.3.3	Ownership Trace Message	37
4.3.4	Program Trace - Direct Branch Message	38
4.3.5	Program Trace - Indirect Branch Message	39
4.3.6	Program Trace - Direct Branch with Sync Message	40
4.3.7	Program Trace - Indirect Branch with Sync Message	41
4.3.8	Program Trace - Resource Full Message	42
4.3.9	Program Trace - Indirect Branch History Message	43
4.3.10	Program Trace - Indirect Branch History with Sync Message	44
4.3.11	Program Trace - Synchronization Message	45
4.3.12	Program Trace - Correction Message	46
4.3.13	Program Trace - Repeat Branch Message	47
4.3.14	Program Trace - Repeat Instruction Message	48
4.3.15	Program Trace - Repeat Instruction with Sync Message	49
4.3.16	Program Trace - Correlation Message	50
4.3.17	Data Trace - Data Write/Read Messages	52
4.3.18	Data Trace - Data Write/Read with Sync Messages	54
4.3.19	In-Circuit Trace Message	55
4.3.20	In-Circuit Trace Message with Sync	56
4.3.21	Data Acquisition Message	57
4.3.22	Error Message	58
4.3.23	Watchpoint Message	60
4.3.24	Port Replacement - Output Message	60
4.3.25	Port Replacement - Input Message	61

4.3.26	Auxiliary Access - Read Message	61
4.3.27	Auxiliary Access - Write Message	62
4.3.28	Auxiliary Access - Read/Write Next Messages	63
4.3.29	Auxiliary Access - Response Message	64

SECTION 5

Nexus Message Protocol

5.1	Rules for Messages	71
5.2	Parallel AUX example	71

SECTION 6

Nexus Port Interfaces

6.1	Nexus Auxiliary Pin Interface	75
6.2	Nexus Auxiliary (AUX) Interface	77
6.2.1	Nexus Auxiliary Pin Functions	77
6.2.2	Example AUX Port Implementations	78
6.3	IEEE 1149.1 (JTAG) Interface	79
6.3.1	TAP Interface	81
6.3.2	TAP Compatibility	81
6.3.3	TAP Pin Functions	82
6.3.4	Accessing the TAP Device ID	83
6.3.5	Optional Ready (RDY) Output Pin	84
6.3.6	Accessing NRRs via the TAP Port	84
6.3.7	Read/Write Access via the TAP Port	87
6.3.8	Accessing Nexus Public Messages via the TAP Port	87
6.3.9	Sample TAP Access Sequences	90
6.4	High Speed Serial Interface (Aurora)	92
6.4.1	Overview	92
6.4.2	Aurora Transmit (Tx)	93
6.4.3	Aurora Receive (Rx) (optional)	93
6.4.4	Aurora Tx/Rx Configurations	93
6.4.5	Tx / Rx Data Types	94
6.4.6	Channel Initialization (Link Training)	95
6.4.7	Recommended Transmission Procedure	99
6.4.8	Recommended Reception Procedure	101
6.4.9	Aurora Flow Control	101
6.4.10	Error Handling	103

SECTION 7

Implementation Topics

7.1	Nexus Reset Configuration	105
7.1.1	Reset for AUX-Only (Full-Duplex) Implementations	105
7.1.2	Reset for TAP Implementations	105
7.1.3	Reset and Port Replacement	106
7.2	Multiple Processor/Client Implementations	106
7.3	Multiple Address Threads	106

7.4	Simultaneous Development of Multiple Embedded Processors.....	107
7.5	Security	107
7.6	Single Master for Tool Connection	107

APPENDIX A

Connector and Electrical Specifications

A.1	Connection Options.....	109
A.1.1	Signal Descriptions.....	120
A.1.1.1	CLOCKOUT	121
A.1.1.2	CLK+, CLK-	121
A.1.1.3	RESET	121
A.1.1.4	General Purpose IO Signals	121
A.1.1.5	VREF	121
A.1.1.6	PORT[15:0].....	121
A.1.1.7	VSTBY	122
A.1.1.8	UBATT	122
A.1.2	Nexus Combined Implementation Considerations	122
A.1.3	Nexus Aux-Only Implementation Considerations	123
A.2	DC Electrical Characteristics	124
A.3	AC Electrical Characteristics - General	124
A.4	AC Electrical Characteristics - IEEE 1149.1 Interface.....	124
A.5	AC Electrical Characteristics - IEEE 1149.7 Interface.....	126
A.6	AC Electrical Characteristics - Parallel Auxiliary Port.....	127
A.7	DC and AC Electrical Characteristics - High Speed Serial Auxiliary Port ...	128
A.7.1	Transmitter Timing	129
A.7.2	Receiver Timing	130
A.7.3	Receiver Eye Patterns.....	131
A.7.4	CLK+, CLK- Specifications	132
A.8	Terminations.....	133

APPENDIX B

Recommendations for Access to Control and Status Registers

B.1	Overview	135
B.2	Reset	138
B.3	Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7).....	138
B.4	Access via the Auxiliary Port (parallel) or Aurora (high speed serial)	139
B.5	Control and Status Registers	140
B.5.1	Device ID (DID) Register	140
B.5.2	Client Select Control (CSC) Register	141
B.5.3	Development Control (DC1) Register 1	142
B.5.4	Development Control (DC2) Register 2.....	144
B.5.5	Development Control (DC3) Register 3.....	145
B.5.6	Development Control (DC4) Register 4.....	145
B.5.7	User Base Address (UBA) Register	146
B.5.7.1	Ownership Trace Messaging (OTM).....	147
B.5.7.2	Data Acquisition Messaging (DQM).....	147

B.5.8	Development Status (DS) Register	148
B.5.9	Overrun Control (OVCR) Register.....	149
B.6	Read/Write Access Registers.....	150
B.6.1	Read / Write Access Control and Status (RWCS) Register	151
B.6.2	Read / Write Access Address (RWA) Register	152
B.6.3	Read / Write Access Data (RWD) Register	153
B.7	Data Trace Registers	154
B.7.1	Data Trace Control (DTC) Register.....	154
B.7.2	Data Trace Start Address (DTSA) and End Address (DTEA) Registers	156
B.8	Breakpoint/Watchpoint Registers	156
B.8.1	Breakpoint / Watchpoint Control (BWC) Register	157
B.8.2	Breakpoint / Watchpoint Address (BWA) Register	158
B.8.3	Breakpoint / Watchpoint Data (BWD) Register	158
B.8.4	Watchpoint Mask (WMSK) Register.....	159
B.8.5	Watchpoint Trigger (WT) Register.....	160
B.9	Aurora Registers	161
B.9.1	Aurora Trace Transmission Control Register (ATTC)	162
B.9.2	Aurora Trace Transmission Status Register (ATTS).....	163
B.9.3	Aurora Link Control Register	163
B.9.4	Aurora Training Control Register.....	164
B.9.5	Aurora Link Status Register	164
B.9.6	Aurora Link Error Register.....	165

APPENDIX C

Application Notes

C.1	Read/Write Access Procedures	167
C.1.1	Block Access with the TAP Interface.....	167
C.1.2	Access with the Auxiliary Port	168
C.1.2.1	Auxiliary Access Messages - Example Sequences	168
C.1.2.2	Termination of Tool/Target Messaging	172
C.2	Recommendations for Nexus Data Formatting (using serial AUX)	172
C.3	Data Acquisition in Tuning for Applications	173
C.3.1	Additional Needs for Automotive Powertrain and Disk Drive Development .	174
C.3.2	Data Acquisition or Measurement of Calibration Variables	175
C.3.2.1	DTM Option	175
C.3.2.2	Read/Write Access Option.....	175
C.3.3	Tuning of Calibration Constants	175

APPENDIX D

Nexus Aurora Details

D.1	Back Channel and Flow Control	177
D.2	Aurora Registers	177
D.3	Simplex Back Channel	177
D.4	Initialization.....	177
D.4.1	Lane Initialization.....	178
D.4.2	Channel Bonding.....	178

D.4.3	Channel Verification	178
-------	----------------------------	-----

APPENDIX E

References

APPENDIX F

Revision History

SECTION 1

Introduction

1.1 Overview

The Nexus 5001 Forum™ (<http://www.ieee-isto.org/Nexus5001/>), was formed in the late 90's to develop an embedded processor debug interface standard for embedded control applications. The internal name given to this standard was “Nexus,” which is used throughout the specification documents (including this one) only.

The goal of the original standard (1999) was to define a general-purpose specification that addressed the rigorous challenges for debug interfaces. The standard also addressed the need for efficient use of embedded processors that require software and hardware development tools to access critical processor functionality.

Applications that continue to benefit from this standard interface include automotive powertrain, data communications, computer peripherals, wireless systems, networking applications, and other control applications.

As processor performance levels and the amount of logic (IP) integrated on the embedded processor continue to rise, providing visibility into the embedded processor has become even more critical. This is especially evident in the trend toward multi-core (multiple processors on a single die) to achieve future performance gains while reducing power consumption.

IEEE-ISTO 5001-2012 builds on the work of the previous versions of the standard (1999 and 2003) on several fronts. By extending the transport mechanisms to support a minimum pin interface (IEEE 1149.7) and a high speed serial protocol (Aurora), the standard covers a wide range of embedded processor performance classes. The standard also continues to support existing transport mechanisms: parallel auxiliary (AUX) and IEEE 1149.1.

Several years of implementation and feedback have also been incorporated into the 2012 version of the standard. These changes are designed to make the Nexus standard more extensible for future embedded processor and system-level development.

1.2 Development Needs for Embedded Processors

Requirements defined by both hardware developers of embedded processors and software developers working with those processors drive a continual need for robust silicon debug features as well as development tools to support them. Basic needs driving this standard fall into three categories: run-control, trace analysis, and development tool requirements.

1.2.1 Requirements for run-control:

- To query and modify processor (or system) resources 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

1.2.2 Requirements for trace analysis:

- 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 flows 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

1.2.3 Requirements for (cost-effective) development tools:

- A standard protocol and standard pin interfaces are needed to support development with multiple clients (processor cores or intelligent peripherals) on the embedded processor. The development pin interfaces comprise basic visibility and controllability of each processor independently.
- A standard pin interface to allow 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.
- Guidelines to eliminate improper multiplexing of development pins, 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.

1.3 Terms and Definitions

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

Term	Definitions
Address	The term is used to indicate logical (virtual) 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.
Aurora	High speed serial protocol defined by Xilinx for high bandwidth transfer rates
Auxiliary Port (AUX)	Refers to the Nexus auxiliary port. There are two version of the AUX: parallel and serial. In either case, it is used as auxiliary port to the TAP interface or as a stand-alone development port.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions. Also, the number of sequential instruction units executed between each taken branch.
Calibration Constants	Performance-related constants that must be tuned for automotive power-train 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 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.
Data Trace Messaging (DTM)	Visibility of how data flow through the embedded system. Includes writes and reads (optionally). Refer to 1.2 - Development Needs for Embedded Processors for more information on data trace requirements.
Embedded Processor	Term for a silicon device with embedded IP. This IP includes: CPUs, on-chip memories, peripherals, interconnect logic and other IP blocks. Also known as a System on a Chip (SoC).
Full-duplex	Messages can be transmitted in both directions between tool and target simultaneously.
Half-duplex	Messages can be transmitted in only one direction at a time between tool and target.
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. Hardware Breakpoints encompasses both Data and Instruction Breakpoints.
High-Speed Input/Output (HSIO)	The term <i>HSIO</i> , as used in the Nexus standard, is intended to refer to an external bus of the embedded processor. Assertion and negation timing is critical to system integrity.

Table 1-1—Terms and Definitions

Term	Definitions
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.
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 is not critical to system integrity.
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 .
Nexus	Internal code name for this 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 TAP 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” is used to indicate “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, which is connected to a development tool.
TAP	Test Access Port. Throughout this document refers to interfaces compliant to IEEE 1149.1 or IEEE 1149.7
TAP Instruction Register (IR) / Data Register (DR) Sequence	TAP IR scan loads an opcode value for selecting a development register. The selected development register is then accessed via an TAP DR scan.
Vendor-Defined Message	Vendor-Defined Messages are allowed for development features that may be specific to each vendor. These messages must follow the protocol defined for the standard (MSEI/O).
Watchpoint	A data or instruction breakpoint that does not cause the processor to halt. Instead a pin or trace message is used to signal that the condition occurred.

Table 1-1—Terms and Definitions (Continued)

1.4 Conventions

This document uses the notational conventions described in **Table 1-2**.

Convention	Description
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.
<u>ACTIVE_LOW</u>	A bar over a signal name indicates that the signal is active low. Active-low signals are referred to as asserted (active) when they are low and negated when they are high.
0xFF	Hexadecimal Numbers
0b0011	Binary Numbers
LSB	Least Significant Bit. The LSB is the lowest bit number (e.g. bit-0).
MSB	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)

Table 1-2—Notational Conventions

1.5 Other Terminology within the Nexus Standard

The terms *vendor-defined extensions*, *vendor-defined operations* and *vendor-defined information* are used to indicate Application Programming Interface (API) extensions that may be implemented as needed for a silicon vendor's embedded processor.

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

The terms and definitions specifically associated with the Nexus-defined Public Messages can be found at the top of **SECTION 4 - 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 protocol and the performance capability for downloading and uploading via the relevant transport mechanism. 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 the transport mechanism (interface) and by transfer bandwidth in gigabits or 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
- Interface: TAP, parallel AUX (PAUX), or serial AUX (SAUX) - (options outlined in **Table 2-3**)
- Upload/Download rate (gigabits or megabits per second)

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. Details on the Nexus defined messages can be found in **SECTION 4 - Nexus Public Messages**.

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
STATIC DEVELOPMENT FEATURES					
Read or write user registers in debug mode	V	V	V	V	Refer to Nexus API
Read or 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 a Nexus compatible API. “V” indicates a required vendor-defined development feature implemented in the Nexus standard.					

Table 2-1—Compliance Classification for Static Development Features

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
DYNAMIC DEVELOPMENT FEATURES					
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
Ability to send out an event occurrence when watchpoint matches	P ¹	P	P	P	Watchpoint Trace
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 or 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
Monitor data reads while processor runs in real time	—	—	O	O	Data Trace (Reads and Writes)

Table 2-2—Compliance Classification for Dynamic (Real-Time) Development Features

Development Feature	Class 1	Class 2	Class 3	Class 4	Nexus Feature
LSIO port replacement and HSIO port sharing	—	O	O	O	Port Replacement/ Sharing
Transmit data values for acquisition by tool	—	—	O	O	Data Acquisition
Timing of events	—	O	O	O	Timestamping
Note: “A” indicates a required development feature that must be implemented via a Nexus compatible 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.					

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

1. Because no auxiliary port is required for Class 1, the event occurrence should be provided via the Event Out (EVTO) pin, or via a message mechanism defined in **6.3 - IEEE 1149.1 (JTAG) Interface**.

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. In the absence of a well defined standard for sub-classes, many implementations simply append the plus (+) sign to an existing compliant class. For example, a Class 2 compliant implementation may also implement data trace (but not read/write access). It is not fully Class 3 compliant, but may be designated Class 2+.

2.2 Performance Classification

Complying embedded processors shall be designated by a performance classification. The embedded processors shall be designated by interface type and by transfer bandwidth in gigabits or megabits per second for both downloads to the embedded processor and uploads from the embedded processor.

Nexus supported interfaces options are outlined in **Table 2-3**, and detailed in **SECTION 6 - Nexus Port Interfaces**. Connector options and electrical characteristics for each type of interface are described in **APPENDIX A - Connector and Electrical Specifications**.

Nexus Port Options	TAP	AUX	Aurora
1149.1/1149.7 TAP	I/O	--	--
Parallel Auxiliary (PAUX)	--	I/O	--
AUX & TAP (combination PCAUX)	Input	Output	--

Table 2-3—Nexus Interface Options

Nexus Port Options	TAP	AUX	Aurora
Aurora (serial) Simplex Auxiliary Port (SSAUX)	Input	--	Output
Aurora (serial) Duplex Auxiliary Port (SDAUX)	--	--	I/O

Table 2-3—Nexus Interface Options

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. This requires a basic knowledge 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.

In calculating the average input and output bandwidth requirements for an application, factors that may be considered are:

- Frequency of taken direct and indirect changes of flow (output)
- Frequency and size of internal data reads/writes that must be visible (output)
- Frequency and size of data that must be read from device (output)
- Frequency and size of data that must be written to device (input)
- Number of trace clients
- Frequency of other trace messages

The specific bandwidth requirements (& pin budgets) will determine the appropriate interface required for each embedded processor implementation.

SECTION 3

Nexus Development Features

The development features and compliance requirements for each feature are described within this section.

3.1 Development Control and Status

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

3.1.1 Overview

Standardized development control and status, and standardized access to it, offer a significant degree of commonality. This 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 needs.

3.1.2 Nexus Recommended Registers (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 a Nexus compliant API.

Development control and status registers (Nexus recommended or vendor-defined) shall be accessed via the TAP interfaces or the AUX interfaces according to the Nexus standard. A sequence for each interface is recommended as part of the standard.

3.2 Ownership (Process ID) Trace

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

3.2.1 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. This 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 (MMU), 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.

3.2.2 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.

Optionally, for embedded processors that implement virtual addressing or address translation, an Ownership Trace Message may be transmitted periodically during runtime at a minimum frequency of every 256 Program/Data Trace Messages.

For embedded processors which do not provide an architected Process ID (PID) register, The Nexus standard defines an OTM Register whose user memory map location is accessed via the TAP or AUX interfaces. The OTM Register is then 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.5.7 - User Base Address (UBA) Register** for more information.

The specific fields defined for OTM messages can be found in **Section 4.3.3 - Ownership Trace Message**.

Ownership trace information can also be transmitted via Program Correlation Messages in lieu of Ownership Trace Messages as outlined in **Section 4.3.16 - Program Trace - Correlation Message**.

3.3 Program Trace

Embedded processors complying with Class 2, 3, and 4 shall provide:

1. Program trace visibility via the AUX according to the Nexus standard
2. Capability to detect and signal program trace 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.
 - Optional support for “re-trying” messages may be implemented to avoid

errors due to contention with higher priority messages.

3. Capability to synchronize program trace according to the Nexus standard

3.3.1 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.

3.3.2 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 which incorporate predicated instructions. For Class 2 (and higher) embedded processors, it is optional which type of program trace messages are used.

3.3.2.1 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 also* 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 also* included in the count of sequential instruction units.

3.3.2.2 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 indirect taken branch or exception; 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 also* 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 **4.3.7 - 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:

- *Trace which direct branch is taken*
- *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.

3.3.3 Program Trace Errors

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

3.3.4 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{EVTI}) pin¹*
- *Program trace error*
- *Upon overflow of the sequential instruction unit counter*
- *Optionally, upon occurrence of a watchpoint*

NOTE

A synchronization-type 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.

Details on the synchronization conditions can be found in **Table 4-4**.

3.4 Data Trace

Embedded processors complying with Classes 3 and 4 shall provide:

1. *Data trace for write visibility via the AUX according to the Nexus standard*
 - *Embedded processors complying with Classes 3 and 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 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*

¹If the processor is not capable of generating the Program Trace - Synchronization Message, the next branch message should be converted to a synchronization message.

delay the processor and avoid overruns.

- *Optional support for “re-trying” messages may be implemented to avoid errors due to contention with higher priority messages.*

3. Capability to synchronize data trace according to the Nexus standard

3.4.1 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.

3.4.2 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.

3.4.3 Data Trace Errors

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

3.4.4 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 data trace 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 for 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
- Data trace error
- Optionally, as a result of a watchpoint, the next data trace message should be a sync message

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.

3.5 Breakpoints/Watchpoints

Embedded processors complying with Class 1, 2, 3, or 4 shall provide a minimum of two instruction/data hardware breakpoints.² 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.

3.5.1 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.

3.5.2 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

²Optionally, 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.

associated with a pre-selected address.

- Watchpoint—a data or instruction breakpoint that does not cause the processor to halt. A Watchpoint Message via the AUX is used to signal that the condition occurred.

3.6 Read/Write Access

Embedded processors complying with Classes 3 and 4 shall provide read/write access to user memory-mapped resources according to the Nexus standard, either via the TAP interface or one of the AUX interfaces (parallel or serial). The capability to perform read/write access shall be provided when the processor is halted or running.

3.6.1 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.

3.6.2 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 two options may be used to implement this feature on the embedded processor:

- *Read/Write Access using Auxiliary Access Messages (AUX port)*
- *Read/Write Access using the **TAP** port*

3.6.2.1 Read/Write Access Using Auxiliary Access Messages

Public Messages are defined in **SECTION 4 - 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 port. 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 hit and is ended by the tool. Refer to **3.7 - 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.

3.6.2.2 Read/Write Access Using the TAP Port

Read/Write Access through the TAP port can utilize the NRRs or use vendor-defined registers. In either case, Public Messages defined in **SECTION 4 - Nexus Public Messages** are not used. Refer to **B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** for detail on how the TAP port is used to access the NRRs.

3.7 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 and may optionally be 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.

3.7.1 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 AUX 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.

3.7.2 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 TAP 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 TAP port or the AUX. It can be activated on watchpoint occurrence by configuring watchpoint trigger information via the TAP 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.

3.8 In-Circuit Trace (Optional)

Embedded processors complying with Class 2, 3 or 4 may optionally implement messages which provide additional visibility to specific events, signals or circuits of interest via the AUX according to the Nexus standard.

3.8.1 Overview

For software development and silicon debug of deeply embedded circuits, providing a minimally intrusive mechanism to view implementation specific resources can significantly reduce debug and development time.

3.8.2 In-Circuit Trace Messaging (ICT)

The majority of the fields defined within the ICT message are vendor specific. Each implementation may choose to implement the fields as desired. Only the TCODE and q data field are required. Optional fields defined by the standard may be implemented to provide additional information.

3.8.3 In-Circuit Trace Synchronization

The AUX port bandwidth requirements may be minimized for in-circuit trace by transmitting compressed data - similar to the relative address for program and data trace messages.

Any number of vendor defined conditions may cause in-circuit trace data to be lost. Synchronization messages provide a starting reference for subsequent In-Circuit Trace Messages. These messages will provide uncompressed data.

For implementations which do not distinguish between compressed and uncompressed in-circuit trace data, synchronization may not be necessary.

3.9 Vendor Defined Trace (Optional)

Embedded processors complying with Class 2, 3 or 4 may optionally implement additional messages defined by implementation. These messages will be transmitted via the AUX. Additional development tool support may be required for these messages.

3.10 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.

3.10.1 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.

3.10.2 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 3-1**. 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 4 - Nexus Public Messages**.

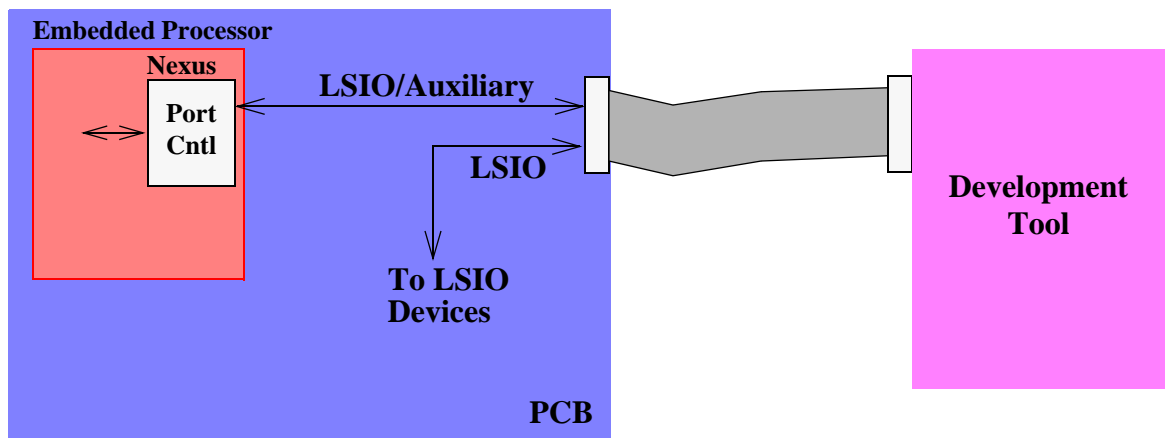


Figure 3-1—Port Replacement for LSIO Pins

Most messages transmitted in a typical application will comprise 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 and another indicating the state of all LSIO. Port Replacement Messages from the tool will contain one essential packet indicating the state of all LSIO. This information will be used by the embedded processor and tool to maintain the correct state and direction for all LSIO.

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).

3.10.3 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 auxiliary output port 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 comprise 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 3-2** for an illustration.

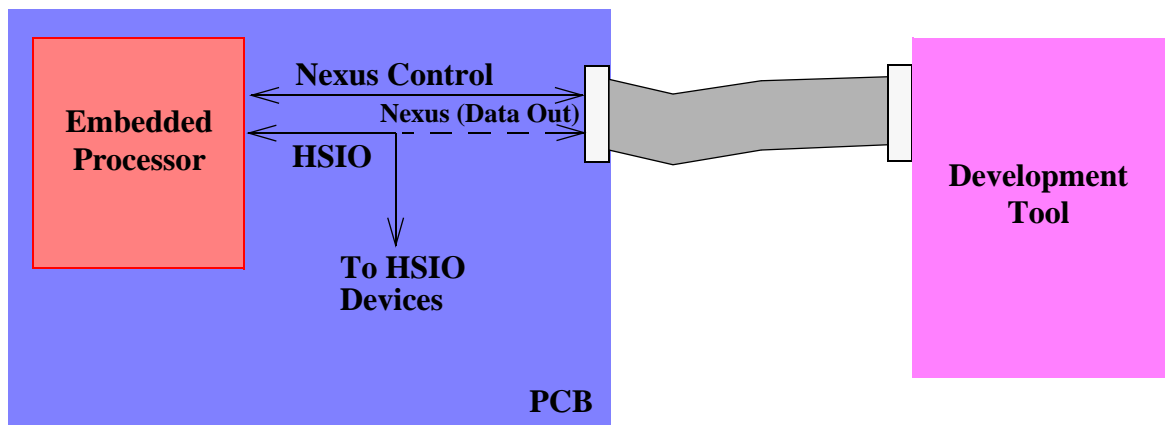


Figure 3-2—Port Replacement for HSIO Pins (Port Sharing)

3.11 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.

3.11.1 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.

3.11.2 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 TAP or auxiliary pin interface. The recommended protocol consists of writing to a designated user memory-mapped location (or dedicated CPU specific resource) to generate a Data Acquisition Message with a specific display data ID tag.

3.12 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.

3.12.1 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. This 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 Nexus message*
2. *Timestamping using optional pins*

3.12.2 Timestamping via AUX

For each message defined in **SECTION 4 - 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).

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. The method chosen for timestamping through the auxiliary port is vendor-defined.

3.12.3 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, 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 with a parallel AUX, the signal will assert for one cycle of Message Clockout (MCKO) for each timestamped event. If EVTO is used with a serial AUX port, the signal will assert for a pre-determined minimum number of embedded processor clock cycles such that the external development tool can capture the signal.

SECTION 4

Nexus Public Messages

The AUX (parallel or serial) port 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.

The Nexus Public Messages are defined in this section. The Nexus message protocol is defined in **SECTION 5 - Nexus Message Protocol**.

There are several transport mechanisms defined for transmitting Nexus Public Messages: parallel AUX, TAP (IEEE 1149.1 and IEEE 1149.7) and over a high speed serial link (serial AUX). The transport mechanisms are defined in **SECTION 6 - Nexus Port Interfaces**.

4.1 Compliance Requirements for Public Messages

Embedded processors complying with Class 2, 3, or 4 shall implement messaging using the Nexus defined protocol and standard defined interfaces. Embedded processors complying with Class 1 may optionally implement messaging via the TAP interface.

Embedded processors shall implement the minimum Public Messages as required per the compliance class. **Table 4-1** lists the minimum required Public Messages per compliance class. Embedded processors may optionally implement Vendor-Defined Messages.

Required Feature	Minimum Required Public Messages	Class 2	Class 3	Class 4
Device Identification	- Device ID Message ¹	X	X	X
Task/Process ID	- Ownership Trace Message	X	X	X
Watchpoint Indication	- Watchpoint 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 ²	X	X	X
	OR			
	Program Trace using branch history messages: - Program Trace - Indirect Branch History Message - Program Trace - Synchronization Message ³ - 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 or vendor-defined registers: - Auxiliary Access - Read Message - Auxiliary Access - Write Message - Auxiliary Access - Read Next Message - Auxiliary Access - Write Next Message - Auxiliary Access - Response Message	--	X	X
	OR			
	Read/Write Access via TAP (JTAG) port : - No Public Messages Required (see Section 6.3.7 - Read/Write Access via the TAP Port)			
Memory Substitution	- Auxiliary Access - Read Message - Auxiliary Access - Read Next Message - Auxiliary Access - Response Message	--	--	X

Table 4-1—Minimum Required Public Messages

1. For embedded processors that do not implement an **IEEE 1149.1**-compliant IDCODE
2. The Direct Branch with Sync Message and Indirect Branch with Sync Message may be implemented instead of the Synchronization Message.
3. The Indirect Branch History with Sync Message may be implemented instead of the Synchronization Message.

4.2 Definitions and Terminology

The following general terms relate to Public Messages:

Term	Definition
Message	Each message starts with a 6-bit TCODE, which defines the type of information carried in the message and its format. The TCODE field 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 5 - Nexus 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.
Field	A field is a distinct piece of the information contained within a message, and messages may contain one or more fields. Field definitions can be found in the Public Message descriptions in Table 4-3 .
Packet	A packet is a collection of fields. Each packet may contain multiple fixed length fields, but may contain at most one (1) variable length field. The variable length field must be the last field in a packet. The MSEI/O signals determine the width of the packet.
Port Boundary	This relates the length of a field to the width of the auxiliary input port or auxiliary output port.
Variable	Specifying that a field is variable-size means that the message must contain the field, but that the field's size may vary from a minimum of 1 bit. When messages are transferred via the parallel AUX, variable-size fields 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 fields may be of different lengths in messages of the same type, the tool must use the MSEI/O signaling protocol to determine the end of packet boundaries. In the case of the serial AUX port (Aurora), variable length fields must end on a "virtual" port boundary.
Vendor-Fixed	<p>The term vendor-fixed is used to indicate allowances in the Nexus standard to match characteristics of a vendor's device. Vendor-fixed fields 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 fields exist in each type of message.</p> <p>Vendor-fixed fields are target processor dependent and have a fixed size determined by the processor vendor.</p>

Table 4-2—General Terms and Definitions

Term	Definition
Vendor-Variable	<p>The term <i>vendor-variable</i> is used to indicate allowances in the Nexus standard to match characteristics of a vendor's device. Vendor-variable fields 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 fields exist in each type of message. When messages are transferred via the parallel AUX, vendor-variable fields 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 fields may be of different lengths in messages of the same type, the tool must use the MSEI/O signaling protocol to determine the end of packet boundaries.</p> <p>These vendor-variable fields 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.</p> <p>For messages that have multiple vendor-variable fields, either dynamic allocation of the field (zero value in some cases, non-zero 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.</p>
Sync and Non-Sync Trace Messages	<p>Program/Data Trace Messages fall into two broad categories—normal (or <i>non-sync</i>) versions and <i>with-sync</i> versions. The main difference between the two categories is that <i>with-sync</i> versions include full addresses whereas normal versions contain addresses that are relative to a previous trace message.</p> <p>Conditions for generating <i>with-sync</i> messages can be found in Table 4-4.</p>
Periodic Message Counter	<p>Because addresses contained in normal 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 <i>with-sync</i> version at least every 256 trace messages.</p> <p>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.</p>
Program Address Threads and Data Address Threads	<p>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 field will be encoded and compressed using the data address most recently transmitted, thus creating a data address thread. Likewise, messages containing an instruction address field will be encoded and compressed using the instruction address most recently transmitted, thus creating an instruction address thread.</p>

Table 4-2—General Terms and Definitions

Table 4-3 defines some of the fields used within Nexus Public Messages.

Field	Description
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.
Source of Message Transmission (SRC)	All of the Public Messages contain a vendor-fixed field that may be used to identify which client was the source of the message transmission. In embedded processors that comprise only a single client, this field need not be transmitted. For embedded processors that comprise multiple clients, this field must be transmitted as part of the message to identify the source of the message transmission. Within a given device, the SRC should be the same size across all client messages.
Instruction Count (I-CNT)	<p>Program Trace Messages have a field that indicates the number of instructions completed since the last transmitted instruction count.</p> <p>For most target architectures (instructions are the same size), this field contains the actual number of instructions. Optionally, if instructions are of variable size, then the number reported may be 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.</p>
Branch History (HIST)	<p>For Indirect Branch History Messages, the branch history field provides a history of direct branch execution used for re-constructing program flow. Its purpose is to track execution history since the previously transmitted program trace message.</p> <p>The HIST field contains an initial stop bit (0b1) followed by bits that represent each direct branch (conditional or non-conditional¹) and a status representing whether the branch was taken (0b1) or not taken (0b0).</p> <p>The HIST field can be 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 field.</p> <p>A value of 1 is shifted into the history buffer on a taken branch (conditional or unconditional¹) 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.</p> <p>The HIST field is reset each time it is transmitted as part of a program trace message that includes history or after a Resource Full Message is transmitted.</p>

Table 4-3—Public Message Field Definitions

Field	Description
Number of Messages Cancelled (CANCEL)	<p>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 field for the number of messages cancelled. There are three vendor-defined interpretations of this field:</p> <ol style="list-style-type: none"> 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/data 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.
Full Address (F-ADDR) of a branch Target	To maintain and confirm coherency with a trace tool, the full address of a branch target is transmitted for Synchronization messages.
Program Counter (PC)	Some messages generate the address of the program counter that is not necessarily the target of a branch instruction. This is used in the Program Trace - Synchronization Message.
Next Address Generation (U-ADDR)	<p>To minimize the size of trace messages, the address fields in the normal (<i>non-sync</i>) versions of all trace messages contain a compressed address. This compressed address, called <i>the unique portion of the address</i>, is relative to the address associated with a previous trace message of the same type. Program Trace Messages contain an address that is relative to the previous BTM; Data Trace Messages contain an address that is relative to the previous DTM.</p> <p>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 4-1).</p>
Instruction/Data Address Map (MAP)	<p>For program trace messages, the MAP field is used to indicate which instruction address space is being utilized by the code being traced. For data trace messages, this field is used to indicate the data address space or memory map for the data accesses being traced.</p> <p>Support for alternate bus masters is overlaid on this field by providing the ID of the master driving the data access - useful for having a single Nexus client support multiple alternate masters on a specific bus.</p> <p>This field may be implemented (or not) as needed per message type.</p>
Timestamp (TSTAMP)	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 Section 3.12 for detail on timestamp implementations.

Table 4-3—Public Message Field Definitions

1. To save trace bandwidth, only conditional direct branches may be recorded in the execution history as unconditional direct branches are always taken, thus the execution flow of these is statically known. Unconditional direct branches are, however, always included in instruction counts. Recording history bits for unconditional branches may simplify program flow re-construction tools slightly, but can consume up to twice as many history bits per loop iteration for simple loops with one unconditional branch back to the start of the loop and a conditional branch out of the loop. Note that a processor should either always include unconditional branches or never include unconditional branches.

Example of how the target 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

$A1 \oplus A2$ = 0000 0000 0000 0000 0000 1111 0110 0100

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

$A1 \oplus M1$ = 0000 0000 0000 0011 1111 0011 0110 0101

Address recreated by the tool = 0x0003F365

Figure 4-1—Next Address Generation Example

As explained in **Section 3.3.4 - Program Trace Synchronization** and **Section 3.4.4 - Data Trace Synchronization**, there are various conditions which require uncompressed data (address for program and data trace messages) to be transmitted through the AUX port. Trace synchronization may apply directly after a SYNC condition occurs (e.g. Program Trace Synchronization after Exit from Debug) or upon first subsequent trace message (e.g. Data Trace Synchronization upon the first qualifying data access). These conditions are outlined in **Table 4-4**.

SYNC Condition	Description
$\overline{\text{EVTI}}$ Assertion	The $\overline{\text{EVTI}}$ pin has been asserted (high to low transition) and the EIC field in the DC1 Register (B.5.3 - Development Control (DC1) Register 1) determines that $\overline{\text{EVTI}}$ pin action is to generate trace synchronization messages.
Exit from System Reset	<p>The embedded processor has successfully exited system reset.</p> <p>For program trace messages, this is required to allow the number of instruction units executed packet in a subsequent BTM to be correctly interpreted by the tool as well as providing the start address of a program flow segment.</p> <p>For data trace messages, this is required to allow the unique portion of the data write (read) address of following Data Write/Read Messages to be correctly interpreted by the tool.</p>
Periodic Message Counter	The periodic trace message counter has expired indicating that 255 <i>without-sync</i> versions of trace messages (program or data) have been sent since the last <i>with-sync</i> version. The value of 255 is a maximum number; target processors may use a smaller value.
Exit from Debug	The embedded processor has exited from debug mode. ¹
Sequential Instruction Counter (Program Trace)	The sequential instruction unit counter has expired. 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.
Trace Enable	Trace (program or data) is enabled during normal execution of the embedded processor.
FIFO Overflow	An overrun condition had previously occurred in which one or more trace occurrences (program and/or data) 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) prior to a <i>with-sync</i> message. The error message contains an ECODE value indicating the type(s) of messages lost due to the overrun condition.
Message Contention	One or more messages was lost due to contention with a higher priority message. To inform the tool that this condition occurred, the target outputs an Error Message (TCODE = 8) prior to a <i>with-sync</i> message. The error message contains an ECODE value indicating the type of message lost due to the contention.
Exit from Power-down	<p>The embedded processor has exited from a power-down state.</p> <p>For program trace messages, this is required to allow the number of instruction units executed packet in a subsequent BTM to be correctly interpreted by the tool, as well as providing the start address of a program flow segment.</p> <p>For data trace messages, this is required to allow the <i>unique portion of the data write (read) address</i> of following Data Write/Read Messages to be correctly interpreted by the tool.</p>
Watchpoint (OPTIONAL)	A watchpoint hit has occurred.

Table 4-4—Synchronization Conditions

1. This can be handled by a Program Trace Correlation Message.

4.3 Detailed Description of Public Messages

In this subsection, the formats of all Public Messages are defined. The complete list of Nexus Public Messages is listed in **Table 4-5**.

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 Hit	15	From target
Reserved	16-19	--
Port Replacement - Output	20	From target
Port Replacement - Input	21	From tool
Auxiliary Access - Read	22	Both ways
Auxiliary Access - Write	23	Both ways
Auxiliary Access - Read Next	24	Both ways
Auxiliary Access - Write Next	25	Both ways
Auxiliary Access - 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
Program Trace - Repeat Instruction	31	From target
Program Trace - Repeat Instruction with Sync	32	From target
Program Trace - Correlation	33	From target
In-Circuit Trace	34	From target

Table 4-5—Nexus Public Messages

Message Name	TCODE Value	Direction
In-Circuit Trace with Sync	35	From target
Reserved	36–55	--
Vendor-Defined Message	56–62	Both ways
Vendor-Defined Extension Message	63 (0x3F)	Both ways

Table 4-5—Nexus Public Messages (Continued)

In each Public Message description below, a 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 4-6** through **Table 4-45**).

4.3.1 Debug Status Message

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**, the tool requests the current debug status by sending a *Auxiliary Access - Read Message* containing the Development Status (DS) register index. For target processors which implement device-specific resources, the register(s) accessed to obtain device status are defined by an API.

Debug Status Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	STATUS	Vendor-fixed	Status information ¹
0	SRC	Vendor-fixed	Client that is source of message.
6	TCODE	Fixed	Value = 0

Table 4-6—Debug Status Message Format

1. For target processors that implement the NRRs described in **APPENDIX B**, the status packet contains the same information as the DS Register.

4.3.2 Device ID Message

The *Device ID Message* is used for AUX-only Nexus implementations. TAP-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 only after the tool has reset the target's debug logic.

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 the target sending 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**, the tool requests the device ID by sending an *Auxiliary Access - Read Message* containing the Device ID register index. For target processors which implement device-specific resources, the register(s) accessed to obtain the device ID are defined by an API

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

Table 4-7—Device ID Message Format

4.3.3 Ownership Trace Message

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. *Optionally*, for target processors using virtual memory, this message may be output immediately prior to (or immediately following) a Program/Data Trace Message with synchronization produced when a periodic message counter expires. This allows a tool to be regularly updated with the latest process ID. Disabling this feature is controlled by DC1[POTD] (see **Section B.5.3**).

Ownership Trace Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	PROCESS	Variable	Task/Process ID.
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this field can be omitted.
6	TCODE	Fixed	Value = 2

Table 4-8 Ownership Trace Message Format

Message Notes:

- Program Correlation Messages may be implemented in lieu of OTMs. See **Section 4.3.16 - Program Trace - Correlation Message** for detail.

4.3.4 Program Trace - Direct Branch Message

The *Direct Branch Message* is output whenever there is a change of program flow and the target address is statically known caused by a branch or subroutine call instruction. This applies regardless if they are conditional (and have a true condition) or are unconditional.

Program Trace - Direct Branch Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	I-CNT	Variable	Number of instruction units executed since the last taken branch or transmitted instruction count.
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, 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 = 3

Table 4-9—Direct Branch Message Format

4.3.5 Program Trace - Indirect Branch Message

The *Indirect Branch Message* is output whenever there is a change of program flow and the target address is determined at runtime caused by a subroutine call, return instruction, asynchronous interrupt/trap, or indirect branch instruction. This applies if the branch is conditional (and have a true condition) or are unconditional.

Program Trace - Indirect Branch Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - 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 or transmitted instruction count.
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see Table 4-11).
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, 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 = 4

Table 4-10—Indirect Branch Message Format

Message Notes:

- Targets may implement Branch History Messages in lieu of Indirect (and Direct) Branch Messages
- The optional B-TYPE field can be used to distinguish between events that cause Indirect Branch Messages (i.e., indirect branch vs. exception). **Table 4-11** shows recommended encodings for the B-TYPE field. The size of the packet transmitted is determined by the number of encodings a client uses.

B-TYPE Value	Description
0b00	Indirect Branch
0b01	Exception
0b10	Hardware Loop

Table 4-11—Recommended B-TYPE Encodings

0b11	Vendor-defined (This could be defined as a thread switch, for example)
------	--

Table 4-11—Recommended B-TYPE Encodings**4.3.6 Program Trace - Direct Branch with Sync Message**

For target processors that do not implement Branch History Messages, the *Direct Branch with Sync Message* is output following a direct branch when any of the synchronization conditions in **Table 4-4** has occurred.

Program Trace - Direct Branch with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - 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 or transmitted instruction count.
0	CANCEL	Vendor-variable	Number of previous Program Trace Messages that should be ignored by the tool. For processors which do not perform speculative execution, this field may be omitted (see Table 4-3).
0	SYNC	Vendor-fixed	Indicates which synchronization condition occurred to generate the <i>with-sync</i> message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization (see Table 4-13).
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, this packet may be omitted (see Table 4-3).
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

Table 4-12—Direct Branch with Sync Message Format

Message Notes:

- If the target processor is not capable of generating a (Program Trace) Synchronization Message, this message should be used in response to an $\overline{\text{EVTI}}$ synchronization request.
- An optional MAP field has been added for processors which support multiple instruction address spaces. This is similar to the MAP field in data trace

messages to support multiple memory maps (or data address spaces).

- The SYNC field replaces the DCONT field from IEEE-ISTO 5001-2003. This field provides an indication of which condition caused a *with-sync* message to be generated. In order to maintain backward compatibility, implementing a SYNC field of 1-bit and following the encodings in **Table 4-13** provides the same functionality as DCONT (i.e. encodings ending in “1” will result in program discontinuity, encodings ending in “0” will not).

SYNC Value	Description
0b0000	EVTI Assertion
0b0001	Exit from System Reset
0b0010	Periodic Message Counter
0b0011	Exit from Debug
0b0100	Sequential Instruction Counter
0b0101	Trace Enable
0b0110	Watchpoint (OPTIONAL)
0b0111	FIFO Overrun
0b1000	Reserved
0b1001	Exit from Power-down
0b1010	Reserved
0b1011	Message Contention
0b1100 - 0b1101	Reserved
0b1110 - 0b1111	Vendor Defined

Table 4-13 Recommended SYNC Encodings

4.3.7 Program Trace - Indirect Branch with Sync Message

For target processors that do not implement Branch History Messages, the *Indirect Branch with Sync Message* is output following an indirect branch (change of program flow caused by a subroutine call, return instruction, or asynchronous interrupt/trap) when any of the synchronization conditions in **Table 4-4** has occurred.

Program Trace - Indirect Branch with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - 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 or transmitted instruction count.
0	CANCEL	Vendor-variable	Number of previous Program Trace Messages that should be ignored by the tool. For processors which do not perform speculative execution, this field may be omitted (see Table 4-3).
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see Table 4-11).
0	SYNC	Vendor-fixed	Indicates which synchronization condition occurred to generate the <i>with-sync</i> message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization (see Table 4-13).
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, 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 = 12

Table 4-14—Indirect Branch with Sync Message Format

Message Notes:

- If the target processor is not capable of generating a (Program Trace) Synchronization Message, this message should be used in response to an EVTI synchronization request.

4.3.8 Program Trace - Resource Full Message

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 Resource Full Message is sent out when any of the resources in **Table 4-16** have reached their maximum value.

Program Trace - Resource Full Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - 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 ¹ . The exact count must be vendor-defined based on the client.
0	RCODE	Vendor-fixed	Resource code. This code indicates which internal resource has reached its maximum value. Refer to Table 4-16 .
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 4-15—Resource Full Message Format

1. Using multiple RDATA packets may cause ambiguity due to the RDATA and the TSTAMP being vendor-variable packets.

Resource Code (RCODE)	Resource	RDATA Value
0b0000	Sequential Instruction Counter	Full instruction counter value (see Table 4-3 for description)
0b0001	Branch/Predicate History Buffer	Full Branch/Predicate instruction history (see Table 4-3 for description)
0b0010-0b0111	Reserved	Reserved for future Nexus standard versions.
0b1000-0b1111	Vendor-defined	Vendor-defined

Table 4-16—Recommended Resource Code (RCODE) Description

4.3.9 Program Trace - Indirect Branch History Message

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 Messages, the *Indirect Branch History Message* is output whenever there is a change of program flow where the target address is determined at runtime caused by an indirect subroutine call, return instruction, or asynchronous interrupt/trap or indirect branch instruction. This applies regardless of whether they are conditional (and have a true condition) or are unconditional.

The history field represents (conditional) direct branch/predicate instruction information. See explanation of the history field in **Table 4-3**.

Program Trace - Indirect Branch History Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	HIST	Variable	Branch/Predicate Instruction History. Refer to Table 4-3 for detailed description.
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 indirect taken branch or transmitted instruction count.
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see Table 4-11).
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, 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 = 28

Table 4-17—Indirect Branch History Message Format

4.3.10 Program Trace - Indirect Branch History with Sync Message

For target processors that do not implement traditional Branch Messages, the *Indirect Branch History with Sync Message* is output following an indirect branch (change of program flow caused by a subroutine call, return instruction, or asynchronous interrupt/trap) when any of the synchronization conditions in **Table 4-4** has occurred.

This message is output by target processors not capable of immediately generating a (Program Trace) Synchronization Message.

Program Trace - Indirect Branch History with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	HIST	Variable	Branch/Predicate Instruction History. Refer to Table 4-3 for detailed description or transmitted instruction count.
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 indirect taken branch or transmitted instruction count.
0	CANCEL	Vendor-variable	Number of previous Program Trace Messages that should be ignored by the tool. For processors which do not perform speculative execution, this field may be omitted (see Table 4-3).
0	B-TYPE	Vendor-fixed	Branch type. For targets that do not need to differentiate branch types, this packet can be omitted (see Table 4-11).
0	SYNC	Vendor-fixed	Indicates which synchronization condition occurred to generate the <i>with-sync</i> message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization (see Table 4-13).
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, 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 = 29

Table 4-18—Indirect Branch History with Sync Message Format

4.3.11 Program Trace - Synchronization Message

The *Synchronization Message* is output by the target processor when any of the synchronization conditions in **Table 4-4** occurs:

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.

Program Trace - Synchronization Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	PC ¹	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 or transmitted instruction count.
0	SYNC	Vendor-fixed	Indicates which synchronization condition occurred to generate the <i>with-sync</i> message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization (see Table 4-13).
0	MAP	Vendor-fixed	A number to indicate the instruction address space currently in use by the target processor. For targets which utilize only a single address space, 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 = 9

Table 4-19—Synchronization Message Format

1. The term PC is used to differentiate the difference between an actual program counter value and the target address of a branch that is normally referenced as F-ADDR.

4.3.12 Program Trace - Correction Message

The *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.

Program Trace - Correction Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	ADJUST	Variable	A number correcting the number of instruction units executed since the last taken branch or transmitted instruction count. 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

Table 4-20—Correction Message Format

Message Notes:

- 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.

4.3.13 Program Trace - Repeat Branch Message

When a series of instructions are executed instead of a single instruction (e.g. hardware loop), the **Repeat Branch Message** may be transmitted to indicate how many times a branch repeated.

A branch is determined to be repeated when the number of executed instruction units (I-CNT) and the target address matches the previous message. The original branch message is only transmitted once, followed by the Repeat Branch Message.

Program Trace - Repeat Branch Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .

Table 4-21—Repeat Branch Message Format

Program Trace - Repeat Branch Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
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

Table 4-21—Repeat Branch Message Format**Message Notes:**

- When using traditional Program Trace Messages, the 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 Branch History Messages are implemented and the repeated branch is a *direct* branch, the Repeat Branch Message is not necessary. The direct branch information is recorded in the history buffer and either the Indirect Branch with History Message or Resource Full Message (with an RCODE indicating Branch/Predicate History) is transmitted as necessary. In some cases, only a single message is generated.
- When Branch History Messages are implemented 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.

4.3.14 Program Trace - Repeat Instruction Message

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 ***Repeat Instruction Message*** is used to trace repeated instructions whose repetition count is a run-time variable.

Program Trace - Repeat Instruction Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	HIST	Variable	Branch/Predicate Instruction History. Refer to Table 4-3 for detailed description.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch or transmitted instruction count.
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

Table 4-22—Repeat Instruction Message Format

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.

4.3.15 Program Trace - Repeat Instruction with Sync Message

For target processors that do not implement traditional Branch Messages, the *Repeat Instruction with Sync Message* is output when a repeated instruction is executed after any of the conditions in **Table 4-4** occurs.

Program Trace - Repeat Instruction with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	HIST	Variable	Branch/Predicate Instruction History. Refer to Table 4-3 for detailed description.
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 or transmitted instruction count.
1	R-CNT	Variable	Repeat Instruction Count. The number of times the instruction was repeated.
0	SYNC	Vendor-fixed	Indicates which synchronization condition occurred to generate the <i>with-sync</i> message. This field can be used to determine whether the I-CNT field is valid for a given source of synchronization (see Table 4-13).
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

Table 4-23—Repeat Instruction with Sync Message Format

4.3.16 Program Trace - Correlation Message

Program Trace Correlation Messages (PTCM) 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.

Program Trace - Correlation Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .

Table 4-24—Correlation Message Format

Program Trace - Correlation Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
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). The CDF field determines how many CDATA packets are transmitted for a specific EVCODE.
1	I-CNT	Variable	Number of instruction units executed since the last taken branch or transmitted instruction count.
0	CDF	Vendor-fixed	Number of CDATA fields that are included with each PTCM (see Table 4-26).
0	EVCODE	Vendor-fixed	Event Code. Refer to Table 4-25 .
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 4-24—Correlation Message Format**Message Notes:**

- The I-CNT value (and HIST field if included) should be cleared after sending the Program 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 Correlation Message.
- For targets that incorporate multiple EVCODEs, the CDF field may be used to determine the number of CDATA field(s) associated with a specific PTCM.
- Individual EVCODEs can be masked from generating Program Correlation Messages by writing a “1” to the corresponding EVCDM bit(s) in Development Control Register 4 (see **B.5.6 - Development Control (DC4) Register 4**).
- A PTCM can be implemented in lieu of Ownership Trace Messages when the Process ID has been updated. This allows the target to send additional instruction information (e.g. I-CNT, HIST) with the ProcessID.

NOTE

If PTCMs are implemented with EVCODE = 0b0101, enabling OTM will continue to transmit periodic OTMs unless DC1[POTD] is also set

Event Code (EVCODE)	Mnemonic	Event Description
0b0000	EVCODE #1	Entry into debug mode

Table 4-25—Recommended Event Code (EVCODE) Description

Event Code (EVCODE)	Mnemonic	Event Description
0b0001	EVCODE #2	Entry into low-power mode
0b0010	EVCODE #3	Data Trace - Write
0b0011	EVCODE #4	Data Trace - Read
0b0100	EVCODE #5	Program Trace Disabled
0b0101	EVCODE #6	Process ID Change (with instruction info)
0b0110-0b0111	EVCODE #7 - EVCODE #8	Reserved for future functionality
0b1000-0b1111	EVCODE #9 - EVCODE #16	Vendor-defined

Table 4-25—Recommended Event Code (EVCODE) Description

CDATA Fields (CDF)	Description
0b00	No CDATA fields associated with the PTCM
0b01	One (1) CDATA field associated with the PTCM
0b10	Two (2) CDATA fields associated with the PTCM
0b11	Three (3) CDATA fields associated with the PTCM

Table 4-26 Recommended CDF Encodings**4.3.17 Data Trace - Data Write/Read Messages**

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.

Data Trace - Data Write/Read Messages			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - 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 DSZ 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 data trace messages to the program flow (e.g. I-CNT or instruction address). For targets which do not need correlation or implement this feature using Program Correlation Messages, this field may be omitted.

Table 4-27—Data Write/Read Message Formats

Data Trace - Data Write/Read Messages			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	ELSZ	Vendor-fixed	Indication of the size of the element within the data access. For processors which do not need to distinguish the element size within the data field, this field may be omitted (see Table 4-29).
0	DSZ	Vendor-fixed	Indication of the size of the write/read. For targets in which the size can be determined from the DATA packet or otherwise, this field may be omitted (see Table 4-28).
0	MAP/ MASTER	Vendor-fixed	For processor cores, this field indicates the memory map currently in use by the target processor. For alternate bus master clients, this field indicates which master initiated the data access. For targets which implement only one memory map (or a single bus master), this field may be omitted (Table 4-3).
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)

Table 4-27—Data Write/Read Message Formats**Message Notes:**

- IEEE-ISTO 5001-2012 supports data trace for alternate bus master clients as well as for traditional processor core clients. The MASTER field (overlaid on the processor MAP field) provides the ability for a single Nexus client to monitor data accesses from multiple bus masters. The development tool will use the SRC field to determine which field (MAP vs. MASTER) is implemented.
- ELSZ provides processors which operate on individual elements within a data field a means to uniquely distinguish these accesses from operations on the entire data field.

DSZ ¹	Data Size	Description
0000	0-byte ²	0-bit
0001	1-byte	8-bit
0010	2-byte	16-bit / halfword
0011	3-byte	24-bit / string
0100	4-byte	32-bit / word
0101	5-byte	Mis-aligned accesses
0110	6-byte	
0111	7-byte	
1000	8-byte	64-bit / double

Table 4-28 Recommended Data Size (DSZ) Encodings

DSZ ¹	Data Size	Description
1001	16-byte	128-bit
1010	32-byte	256-bit
1011	64-byte	512-bit
1100-1101	Reserved	
1110-1111	Vendor Defined	

Table 4-28 Recommended Data Size (DSZ) Encodings

1. Not all DSZ values must be supported
2. Implied data instructions may support a “zero-data” size

ELSZ ¹	Element Size
000	ELSZ = DSZ
001	8-bit
010	16-bit
011	32-bit
100	64-bit
101	128-bit
110	Reserved
111	Vendor Defined

Table 4-29—Recommended Element Size (ELSZ) Encodings

1. Not all ELSZ values must be supported

4.3.18 Data Trace - Data Write/Read with Sync Messages

Data Write/Read with Sync Messages are transmitted in lieu of Data Write/Read Messages whenever a memory access occurs that matches the debug logic’s data trace attributes, and when one of the conditions in **Table 4-4** has occurred.

Data Trace - Data Write/Read with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - 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 DSZ is included.
1	F-ADDR	Variable	The full address of the memory location written/read. MSBs with a value of 0 may be truncated.

Table 4-30—Data Write/Read with Sync Message Formats

Data Trace - Data Write/Read with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	CANCEL	Vendor-variable	Number of previous Data Trace Messages that should be ignored by the tool. For processors which do not perform speculative execution, this field may be omitted (see Table 4-3).
0	DCORR	Vendor-fixed	Field used for correlating data trace messages to the program flow (e.g. I-CNT or instruction address). For targets which do not need correlation or implement this feature using Program Correlation Messages, this field may be omitted.
0	ELSZ	Vendor-fixed	Indication of the size of the element within the data access. For processors which do not need to distinguish the element size within the data field, this field may be omitted (see Table 4-29).
0	DSZ	Vendor-fixed	Indication of the size of the write/read. For targets in which the size can be determined from the DATA packet or otherwise, this field may be omitted (see Table 4-28).
0	SYNC	Vendor-fixed	Indication of specific synchronization condition causing the <i>with-sync</i> message (see Table 4-13).
0	MAP/ MASTER	Vendor-fixed	For processor cores, this field indicates the memory map currently in use by the target processor. For alternate bus master clients, this field indicates which master initiated the data access. For targets which implement only one memory map (or a single bus master), this field may be omitted (Table 4-3).
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 4-30—Data Write/Read with Sync Message Formats**4.3.19 In-Circuit Trace Message**

The *In-Circuit Trace Message* provides a mechanism to trace target signals and events of interest. This trace information can be useful for debugging silicon problems.

In-Circuit Trace Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .

Table 4-31 In-Circuit Trace Message Format

In-Circuit Trace Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
1	CKDATA	Variable	This packet is a vendor-defined field. It can represent independent events and/or signals. This data may be compressed to reduce bandwidth.
0	CKDF	Vendor-fixed	Number of CKDATA fields that are included with each ICT Message
0	CKSRC	Vendor-fixed	Circuit trace source. For clients only supporting one ICT source, this field may 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 = 34

Table 4-31 In-Circuit Trace Message Format

Message Notes:

- The CKSRC field indicates which ICT source has generated the ICT message data. This allows each Nexus client to trace ICT from multiple sources.
- The CKDF field enables ICT messages to support multiple CKDATA fields.

4.3.20 In-Circuit Trace Message with Sync

The *In-Circuit Trace Message with Sync* provides a mechanism to trace target signals and events of interest. The *with-sync* version of the message shall provide uncompressed data and may optionally provide information on the synchronization condition.

In-Circuit Trace with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	CKDATA	Variable	This packet is a vendor-defined field. It can represent independent events and/or signals. This data should be uncompressed for trace reconstruction.
0	CKDF	Vendor-fixed	Number of CKDATA fields that are included with each ICT Message.
0	SYNC	Vendor-fixed	Indication of specific synchronization condition causing the <i>with-sync</i> message (see Table 4-13).
0	CKSRC	Vendor-fixed	Circuit trace source. For clients only supporting one ICT source, this field may be omitted.,

Table 4-32 In-Circuit Trace with Sync Message Format

In-Circuit Trace with Sync Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
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 = 35

Table 4-32 In-Circuit Trace with Sync Message Format

4.3.21 Data Acquisition Message

The *Data Acquisition Message* provides a convenient and flexible mechanism for the external development tool to observe the architectural state of the target through software instrumentation.

Data Acquisition Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	DQDATA	Variable	One or more packets of data values.
1	IDTAG	Vendor-fixed	Data ID tag. This specifies which group of data is included in the Data Acquisition Message.
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 = 7

Table 4-33—Data Acquisition Message Format

Message Notes:

- For implementations which incorporate the User Base Address Register (UBA) for Data Acquisition, DQM Messages are initiated when the target processor writes a value of 0x0 to the register. Specific steps on DQM generation can be found in **C.3.2 - Data Acquisition or Measurement of Calibration Variables**.
- DQMs may also be generated through vendor-defined mechanisms for instrumenting software.
- The IDTAG field specifies the complementary control or attribute information for the data (DQDATA) included in the DQM. Usage of the IDTAG is left to the discretion of the instrumentation software and external development tool.

4.3.22 Error Message

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. The types of errors (ETYPE) are defined in **Table 4-35**.

Optionally, the Error Message may also include an additional field to inform the tool that the target has discarded trace occurrences either due to insufficient space in its trace output queue or collision/contention with one or more higher priority messages. Recommended error code (ECODE) values can be found in **Table 4-36**.

Error Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
0	ECODE	Vendor-fixed	Error code. This indicates the types of messages lost during contention or queue overrun conditions. Refer to Table 4-36 . For targets that do not provide this granularity of error indication, this field may be omitted.
4	ETYPE	Fixed	Error type. Refer to Table 4-35 .
0	SRC	Vendor-fixed	Client that is source of message. For targets with only a single client, this field may be omitted.
6	TCODE	Fixed	Value = 8

Table 4-34—Error Message Format

Message Notes:

- In the queue overrun case (ETYPE = 0000), 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.
- In the contention case (ETYPE = 0001), the Error Message is sent immediately after the message which caused a lower priority message to be dropped. If the lost message type is an OTM, BTM or DTM, a synchronization message is sent out subsequently.

Error Type (ETYPE)	Description
0000	Queue Overrun caused messages (one or more) to be lost
0001	Contention with higher priority messages caused message(s) to be lost
0010	Reserved
0011	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). ¹
0100	Invalid message (message type not implemented). The Error Message is sent by the target as soon as the invalid message is detected.
0101	Invalid access opcode (NRR not implemented). This error code applies only to targets that support the Read/Write Access Messages for NRRs (APPENDIX B). The Error Message is sent by the target as soon as the invalid opcode is detected.
0110 - 0111	Reserved
1000 - 1111	Vendor Defined Error(s)

Table 4-35—Recommended Error Types (ETYPE)

1. 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 AUX Access - Response Message.

Error Code (ECODE)	Description
xxxxxxxxxx1	Watchpoint Message(s) lost (one or more)
xxxxxxxxxx1x	Data Trace Message(s) lost
xxxxxxxxxx1xx	Program Trace Message(s) lost
xxxxxxxxxx1xxx	Ownership Trace Message(s) lost
xxxxxxx1xxxx	Status Message(s) lost (i.e. Debug Status)
xxxxxx1xxxxx	Data Acquisition Message(s) lost
xxxxx1xxxxxx	In-Circuit Trace Message(s) lost
xxxx1xxxxxxx	Vendor Defined Message(s) lost
xxx1xxxxxxx	Reserved
xx1xxxxxxx	Reserved
x1xxxxxxx	Reserved
1xxxxxxx	Reserved

Table 4-36—Recommended Error (ECODE) Codes

4.3.23 Watchpoint Message

The *Watchpoint Message* is sent by the target whenever a watchpoint hit occurs. Multiple watchpoint hits can be indicated in the same message.

Watchpoint Message			Direction: from target
Minimum Size (bits)	Field Name	Field Type	Description
0	TSTAMP	Vendor-variable	Timestamp value. For targets that do not implement timestamping, this field may be omitted. Refer to 3.12.2 - Timestamping via AUX .
1	WPHIT	Variable	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 hit 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

Table 4-37—Watchpoint Message Format

Message Notes:

- The debug logic in the target must ensure that Watchpoint Messages can never be cancelled once they have been generated. If watchpoint hit occurrences are discarded because of insufficient space in the trace output queue, the target must send an **Error Message** prior to the next Watchpoint Message so that the tool knows that one or more watchpoint hit occurrences were discarded.
- Individual watchpoints within the WPHIT field can be masked from generating Watchpoint Messages by writing a “1” to the corresponding bit(s) in the Watchpoint Mask Register (see -).

4.3.24 Port Replacement - Output Message

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.

Port Replacement - Output Message			Direction: from target
Minimum Size (bits)	Field Name	Field 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

Table 4-38—Output Message Format**4.3.25 Port Replacement - Input Message**

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.

Port Replacement - Input Message			Direction: from tool
Minimum Size (bits)	Field Name	Field 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

Table 4-39—Input Message Format**4.3.26 Auxiliary Access - Read Message**

The *Auxiliary Access - Read Message* provides a mechanism for reading memory locations within the target or the tool as well as access to Nexus defined registers (NRRs)

on the target.

Auxiliary Access - Read Message			Direction: from tool, from target
Minimum Size (bits)	Field Name	Field Type	Description
1	ADDRESS/INDEX	Variable	Vendor-defined field specifying the memory location or NRR index to be read. The size of the address must match the address space supported by the target or tool.
0	DSZ	Vendor-fixed	Data Size. DSZ encodings are recommended in Table 4-28 . A target/tool does not need to support all of the defined data sizes.
0	XMAP	Vendor-fixed	A number to indicate the memory map to be used in the target/tool. For targets or tools which only implement a single memory map, or only access NRRs, this packet can be omitted (see Table 4-41).
6	TCODE	Fixed	Value = 22

Table 4-40—AUX Read Message Format

Message Notes:

- The tool (target) sends the AUX Read Message when it wants to read a location in the target's (tool's) memory-mapped address space or access a Nexus defined register (NRR).
- The XMAP field is used to distinguish among memory maps (if multiple are implemented), as well as between memory accesses and NRR accesses. See **Table 4-41** for recommended XMAP encodings.

XMAP	Description
00	NRR Access
01	Access to Memory Map #1
10	Access to Memory Map #2
11	Access to Memory Map #3

Table 4-41—Recommended XMAP Encodings

4.3.27 Auxiliary Access - Write Message

For embedded processors, the *Auxiliary Access - Write Message* provides a mechanism for writing memory locations within the target or the tool as well as Nexus defined registers

(NRRs).

Auxiliary Access - Write Message			Direction: from tool, from target
Minimum Size (bits)	Field Name	Field Type	Description
1	DATA	Variable	Write data value of size DSZ.
1	ADDRESS/ INDEX	Variable	Vendor-defined field specifying the memory location or NRR index to be written. The size of the address must match the address space supported by the target or tool.
0	DSZ	Vendor-fixed	Data Size. DSZ encodings are recommended in Table 4-28 . A target/tool does not need to support all of the defined data sizes.
0	XMAP	Vendor-fixed	A number to indicate the memory map to be used in the target/tool. For targets or tools which only implement a single memory map, or only access NRRs, this packet can be omitted (see Table 4-41).
6	TCODE	Fixed	Value = 23

Table 4-42—AUX Write Message Format

Message Notes:

- The tool (target) sends the AUX Write Message when it wants to write to a location in the target's (tool's) memory-mapped address space.
- The XMAP field is used to distinguish among memory maps (if multiple are implemented), as well as between memory accesses and NRR accesses. See **Table 4-41** for recommended XMAP encodings.

4.3.28 Auxiliary Access - Read/Write Next Messages

For embedded processors that implement vendor-defined development registers (not NRRs) or MSM, the *Auxiliary Access - Read (Write) Next Message* provides a mechanism for reading (writing) consecutively addressed data (block reads) within the target or tool.

Auxiliary Access - Read Next Message			Direction: from tool, from target
Minimum Size (bits)	Field Name	Field Type	Description
6	TCODE	Fixed	Value = 24

Table 4-43—AUX Read Next Message Format

Auxiliary Access - Write Next Message			Direction: from tool, from target
Minimum Size (bits)	Field Name	Field Type	Description
8	DATA	Variable	Write Data - size is determined by the DSZ packet in the most recent AUX Write Message.
6	TCODE	Fixed	Value = 25

Table 4-44 AUX Write Next Message Format**Message Notes:**

- The tool (target) sends the AUX Read Next Message when it has processed a prior AUX Response Message from the target (tool) and the tool's (target's) receive buffer can accommodate more read data.
- The tool (target) sends the AUX Write Next Message to write to the next consecutively addressed location within the target (tool) after it has processed a prior AUX Response Message from the target (tool) and has more data available to send.
- There is no limit to the amount of data that can be transferred using consecutive AUX Read Next Message and AUX Write Next Message commands.
- Both tool and target are required to increment their internal address pointers according to the size of the data being transferred.

4.3.29 Auxiliary Access - Response Message

For embedded processors, the *Auxiliary Access - Response Message* is output by the target (tool), but the contents differ depending on whether the most recent read/write command issued by the tool (target) was contained in a AUX Read Message or a AUX Write Message.

Auxiliary Access - Response Message			Direction: from tool, from target
Minimum Size (bits)	Field Name	Field Type	Description
8	DATA	Variable	Read Data - size is determined by the DSZ packet in the most recent AUX Read Message. This field does not exist if the previous message issued by the tool (target) was an AUX Write Message or if the target (tool) is unable to complete the previously requested read operation.
2	ST	Fixed	Transaction Status: <i>00 = Read/Write operation completed successfully</i> <i>01 = Read/Write operation could not be completed</i> <i>1X = Reserved</i>
6	TCODE	Fixed	Value = 26

Table 4-45—AUX Response Message Format**Message Notes:**

- For read commands, the target (tool) sends an AUX Response Message (containing data) as soon as it has retrieved the data requested by a previous AUX Read Message or AUX Read Next Message from the tool (target).
- For write commands, the target (tool) sends an AUX Response Message (with no data) as soon as the target's (tool's) receive buffer is able to accept more data
- If the target (tool) is unable to process the function requested by the previous AUX Read Message, AUX Write Message, AUX Read Next Message or AUX Write Next Message, it sends a 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. when the substitution process should end). The tool informs the target not to request more data by setting the ST field = 10 in an AUX Response Message that contains the final read data.

SECTION 5

Nexus Message Protocol

The Nexus standard defines a messaging protocol which is independent of the transport mechanism to transfer the data off the embedded processor. In the parallel auxiliary (AUX) port case, the Nexus protocol may be transmitted directly to the I/O pins. In the serial AUX port and TAP cases, the Nexus protocol will be converted to a serial transport protocol for off chip transmission. These transport mechanisms are described in **SECTION 6 - Nexus Port Interfaces**. The Nexus messaging scheme is described below.

The protocol for the embedded processor to generate and interpret messages via the auxiliary port shall be accomplished with the Message Start/End In ($\overline{\text{MSEI}}$) and $\overline{\text{MSEO}}$ functions, respectively. A minimum of one and a maximum of two $\overline{\text{MSEI}}$ signals shall provide the protocol for the embedded processor to receive messages, and a minimum of one and a maximum of two $\overline{\text{MSEO}}$ signals 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” indicates the start of a message.*
- *“0” followed by two or more “1”s indicates the end of a message.*
- *“0” followed by “1” followed by “0” indicates 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 $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ signals. However, when using two $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ signals, 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 (see timestamp note below). If the signals are transmitted via a parallel AUX interface, $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ pins are sampled on the rising edge of Message Clock-IN/OUT (MCKI or MCKO).

MSEO Function	Single MSEO data (serial)	Dual 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
End of message/End of packet	1-1	10
Idle (no message)	1s	11s

Table 5-1—MSEO Signal(s) Protocol

Figure 5-1 illustrates the state diagram for single $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ transfers. When using only one $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ signal, the “End Message” state does not contain valid data on the Message Data IN/OUT (MDI or MDO) signals. 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 signals. This ensures that a false end of message state is not entered by transmitting two consecutive ones on the $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ signal before the actual end of message.

Figure 5-2 illustrates the use of dual $\overline{\text{MSEO}}$ transfers. The dual $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ option is more efficient than the single signal 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 single $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ option. The dual-signal option also allows for consecutive “End Packet” states. This can be an advantage when small, variable-sized packets are transferred.

NOTE

Previous versions of the standard supported using Start to Start for single clock messages. This is not recommended, use the End to End states instead.

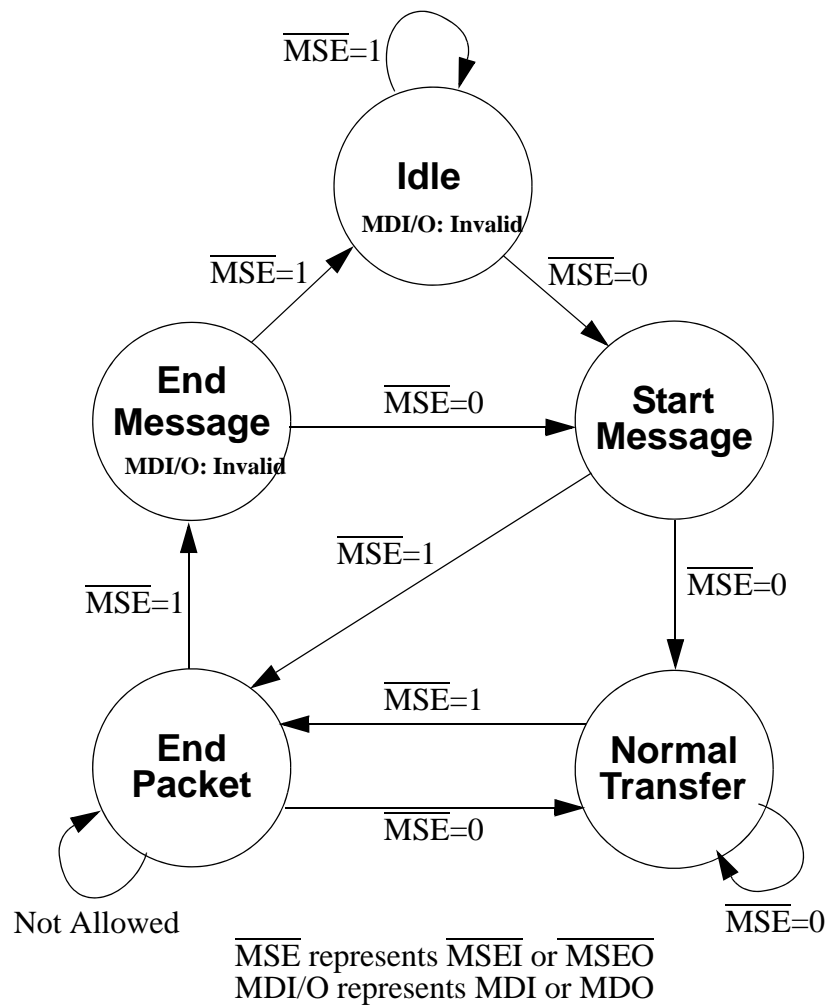
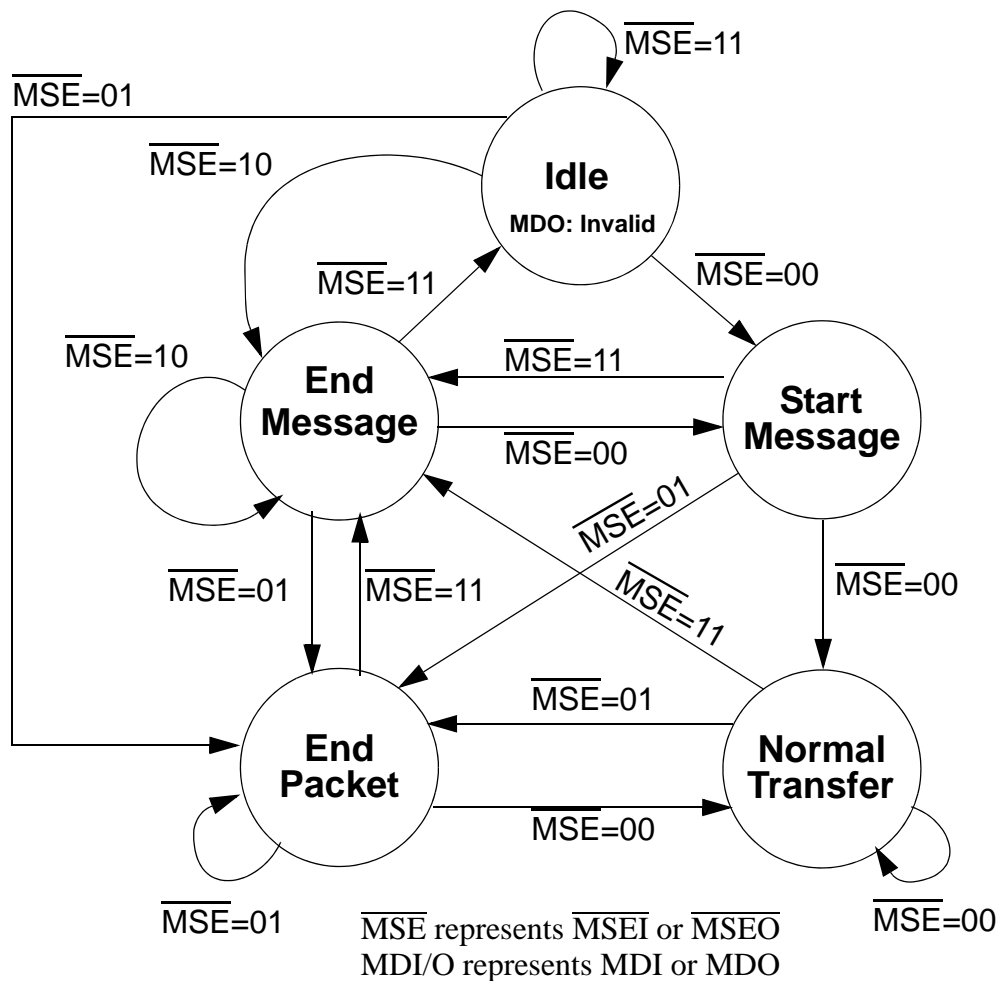


Figure 5-1—Single $\overline{\text{MSEI}}/\overline{\text{MSEO}}$ Transfers



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 (for parallel AUX cases).
3. The dual $\overline{\text{MSE}}$ option should be chosen when performance is the top priority and pin count is secondary (for parallel AUX cases).

Figure 5-2—Dual $\overline{\text{MSEI}}$ / $\overline{\text{MSEO}}$ 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 dual $\overline{\text{MSEI}}$ / $\overline{\text{MSEO}}$ option.

5.1 Rules for Messages

Embedded processors complying with Classes 2, 3, and 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 field is 5 bits, then the remaining 3 bits of MDO must be packed with 0s.

Clock	MDO[3:0]				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**³ 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. Any kind of packet can begin on a port boundary.
- MSEL/MSEO protocol must be followed for both input and output messages.

5.2 Parallel AUX example

Figure 5-3 illustrates the transfer protocol for the Indirect Branch Message transmitted on a parallel AUX interface. For purposes of illustration only, one MDO pin and one MSEO

³For the timestamp field, the preceding fixed length packet may be padded out to the port boundary to remain consistent with messages that contain timestamp fields which follow variable length packets.

pin are shown. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

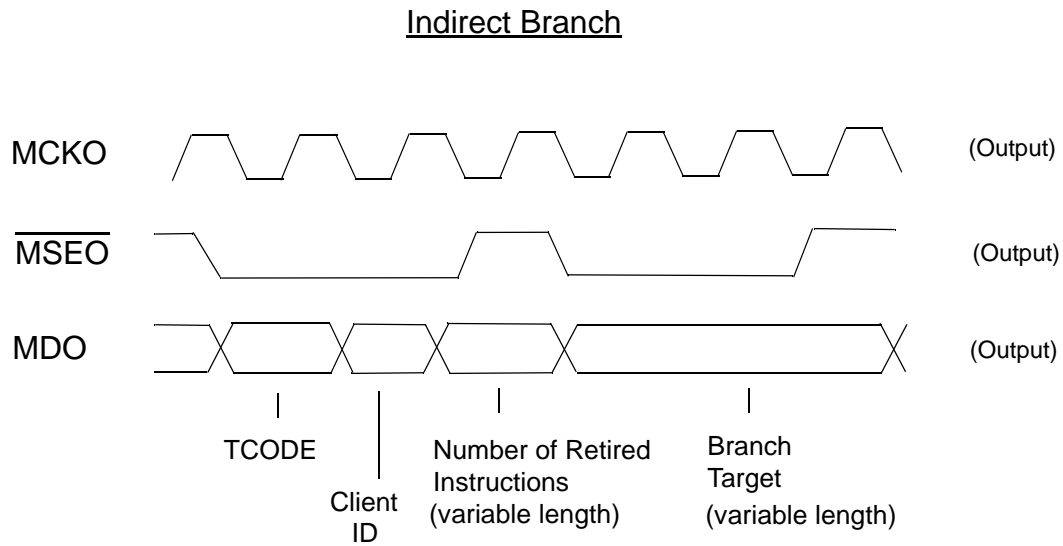


Figure 5-3—Timing Diagram for Indirect Branch Message

SECTION 6

Nexus Port Interfaces

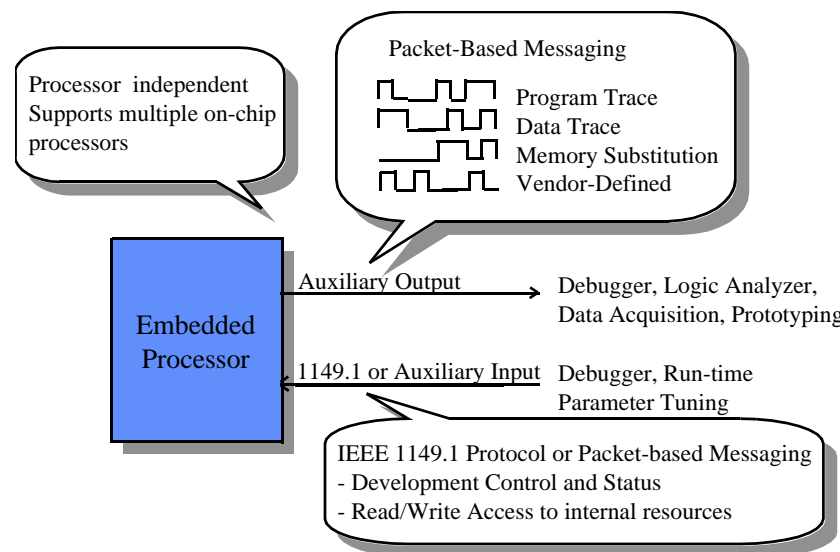
The Nexus standard supports multiple transport mechanisms for transmission of Nexus messages to/from embedded processors. These transport mechanisms are divided into three categories: parallel AUX (MDI, MDO, MSEO, etc.), TAP interfaces (IEEE 1149.1 and IEEE 1149.7), and high speed serial AUX. The Nexus protocol “layer” behind the transport mechanism is the same in each case.

The transport mechanisms (and combinations) are determined by the embedded processor implementation. Various factors such as pin budgets, embedded processor performance, bandwidth requirements, and application specific functions all determine which transport mechanisms will be supported.

The development interface shall be implemented by Nexus Class 1, 2, 3, and 4 embedded processors using one of the options outlined in **Table 6-2**. The development features shall be implemented by Nexus Class 1, 2, 3, and 4 embedded processors as described in **SECTION 2 - Compliance and Performance Classifications**.

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

Figure 6-1 illustrates Nexus development interface options for a Class 2, 3, or 4 embedded processor.



Note: The auxiliary input port is input only. Although the **TAP** interface is bidirectional, for simplicity it is illustrated as input only.

Figure 6-1—Illustration of Nexus Development Interface Options

For implementation of the TAP interface options for Class 2, 3, and 4 embedded processors, as referenced in **Figure 6-1**, in addition to the IEEE 1149.1-defined pins⁴, only three 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 TAP interface, an auxiliary input port 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 6-1** shows recommendations (not requirements) 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

Table 6-1—Recommendations for Auxiliary Port Type

⁴**TAP** (JTAG) pins include Test Clock (TCK), Test Mode Select (TMS), Test Data Input (TDI), Test Data Output (TDO), and Test Reset (TRST).

6.1 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 TAP interface option is not selected by the embedded processor IC developer.

The auxiliary interface shall provide the following external visibility according to the Nexus standard:⁵

- 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 interface shall provide the following access according to the Nexus standard, if the TAP 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

This section describes the three types of transport. Electrical characteristics, physical layer specifics (PHY) and connector information for all three can be found in **APPENDIX A - Connector and Electrical Specifications**.

Table 6-2 outlines all of the interface options supported by Nexus and the associated pins. It also identifies the required and optional pins for each type of interface implementation. Required and optional pin functions are designated by “R” and “O” respectively. Pins not allowed for an interface are shaded. Pin descriptions can be found within the sections describing each type of interface.

⁵Refer to **Section 1.3 - Terms and Definitions** for definitions of all new terms in the list.

Pin Name	Direction	Parallel AUX	Parallel AUX & TAP (combination)	1149.1 TAP (only)	1149.7 TAP (only)	Serial AUX (simplex)	Serial AUX (duplex)
MCKI	In	R					
RSTI	In	R					
MDI	In	R					
MSEI	In	R					
MCKO	Out	R	R				
MDO	Out	R	R				
MSEO	Out	R	R				
	In	R	O	O	O	O	O
EVTO	Out	O	O	O	O	O	O
TCK / TCKC	In		R	R	R	R	
TDI	In		R	R	O	O	
TDO	Out		R	R	O	O	
TMS / TMSC	In		R	R	R	R	
TRST	In		O	O	O	O	
RDY	Out		O	O	O	O	
CLK (pair)	In					O	O
Tx0 (pair)	Out					R	R
Tx1 - Tx7 (pair)	Out					O	O
Rx0 (pair)	In						R
Rx1 - Rx3 (pair)	In						O

Table 6-2 Nexus Interface Minimum Pin Requirements**NOTE**

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

NOTE

For **TAP** implementations, the $\overline{\text{EVTI}}$ pin is optional. However, tool-initiated message synchronization and breakpoint generation functionality defined by

the Nexus standard is lost if the pin is not implemented.

6.2 Nexus Auxiliary (AUX) Interface

When the Nexus protocol signals outlined in **SECTION 5 - Nexus Message Protocol** are directly pinned out of the embedded processor, this interface is defined as the parallel auxiliary or AUX port. The pin descriptions for these signals are defined the following sections, while electricals and connectors supporting this interface are described in **APPENDIX A - Connector and Electrical Specifications**.

6.2.1 Nexus Auxiliary Pin Functions

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

Auxiliary Pins	Description of Auxiliary Pins
MCKO	Message Clockout (MCKO) is a free-running output clock to development tools for timing MDO and $\overline{\text{MSEO}}$ 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.
$\overline{\text{MSEO}}[1:0]$	Message Start/End Out ($\overline{\text{MSEO}}$ [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 $\overline{\text{MSEO}}$ pin is required, but up to two pins may be implemented for more efficient transfers. External latching of $\overline{\text{MSEO}}$ 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 $\overline{\text{MSEI}}$ 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.
$\overline{\text{MSEI}}[1:0]$	Message Start/End In ($\overline{\text{MSEI}}$ [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 $\overline{\text{MSEI}}$ pin is required, but up to two pins may be implemented for more efficient transfers. Internal latching of $\overline{\text{MSEI}}$ shall occur on the rising edge of MCKI.
$\overline{\text{EVTI}}$	Event In ($\overline{\text{EVTI}}$) is an input where, when a high-to-low transition occurs, a processor is halted (breakpoint) or Program and Data Synchronization Messages are transmitted from the embedded processor.
$\overline{\text{RSTI}}$	Reset In ($\overline{\text{RSTI}}$) is for resetting the Nexus port resources.

Table 6-3—Auxiliary Pins

Auxiliary Pins	Description of Auxiliary Pins
$\overline{\text{EVTO}}$	Event Out ($\overline{\text{EVTO}}$) is an optional output pin to development tools comprising exact timing for a single breakpoint status indication. Upon a breakpoint occurrence of the programmed breakpoint source, $\overline{\text{EVTO}}$ is asserted for a minimum of one clock period of MCKO.

Table 6-3—Auxiliary Pins

6.2.2 Example AUX Port Implementations

For a full-duplex AUX with TAP 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. $\overline{\text{EVTI}}$ 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**.

Number of Pins for Each Example					Comments
MDO	$\overline{\text{MSEO}}$	MCKO	$\overline{\text{EVTI}}$	Total Pins	
1	1	0	0	2	Base implementation (without $\overline{\text{EVTI}}$)
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 (with $\overline{\text{EVTI}}$)
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	

Table 6-4—Example of Auxiliary Output Ports

Table 1.3 and **Table 1.3** illustrate examples of one-pin and two-pin MSEO options for the same Indirect Branch Message (Traditional) using 4 MDO signals. The actual port width is vendor defined.

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

Clock	MDO[3:0]				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 6-5—Indirect Branch Using the One-Pin MSEO Option

Clock	MDO[3:0]				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

Table 6-6—Indirect Branch Using the Two-Pin MSEO Option

6.3 IEEE 1149.1 (JTAG) Interface

With the Nexus standard embedded processors complying with Class 1, 2, 3, or 4 may optionally implement messages via the Test Access Port (TAP) defined by either the IEEE

1149.1™ or the IEEE 1149.7™ standard. Hereafter this interface is called the TAP interface.

The IEEE 1149.1™ interface is a four pin interface with an additional/optional test reset pin as shown in **Figure 6-2 - IEEE 1149.1™ Interface**. It provides access to boundary scan and other scan chains in a device, some of which are utilized for Nexus specific purposes.

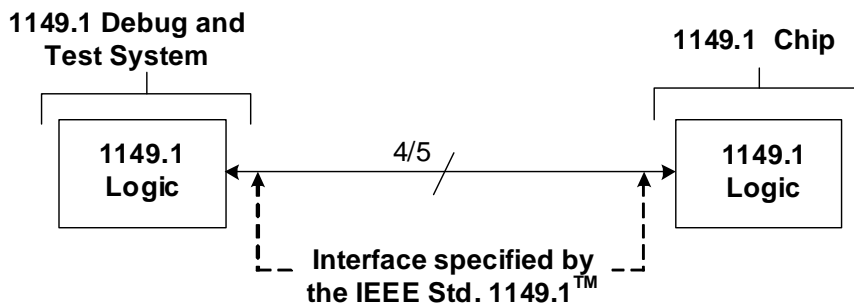


Figure 6-2—IEEE 1149.1™ Interface

The IEEE Std. 1149.7™ standard is based on the IEEE Std. 1149.1™ standard. It adds a IEEE 1149.7™ protocol converter in front the IEEE 1149.1™ interface as shown in **Figure 6-3 - Figure IEEE Std. 1149.7™ Interface**, leaving the IEEE 1149.1™ standard unchanged.

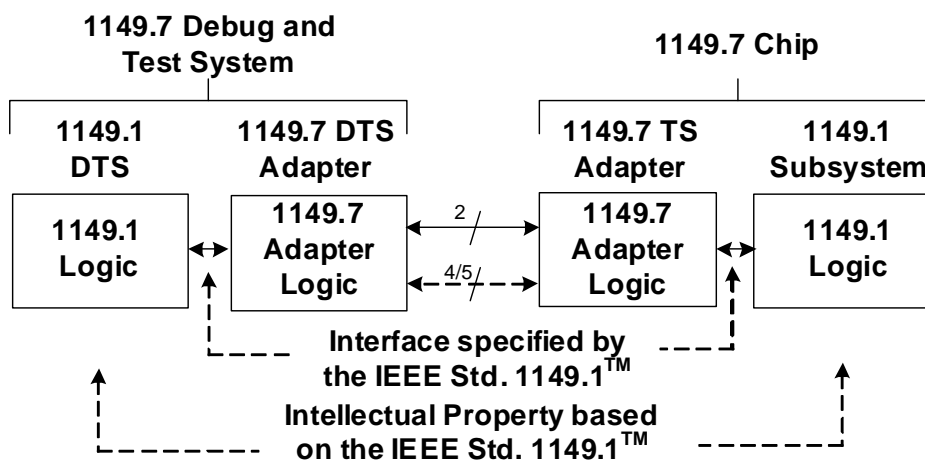


Figure 6-3—Figure IEEE Std. 1149.7™ Interface

There are multiple configurations of the IEEE 1149.7 adapter, some supporting the native

IEEE 1149.1 four/five-pin interface (clock, three data, and optional reset signal), and others supporting two-pin interfaces (clock and data). With two pin operation (clock and data signals) the IEEE 1149.7 adapter serializes the information exchanged with the IEEE 1149.1 interface with the bi-directional data pin. It also provides for time division multiplexing the use of the data signal between IEEE 1149.1 protocol exchanges and one or more other protocols. This capability may be used to instrument the system behavior for other purposes.

6.3.1 TAP Interface

The standard defining the TAP interface specifies the required protocol used to access the minimum development features of compliance Class 1. Additionally, the TAP standard defines the required protocol for access to the minimum development features of compliance Classes 2, 3, and 4, if an alternative interface option is not selected by the developer of the embedded processor IC.

The TAP interface shall provide the following capability:

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

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 **6.3.8.2 - Nexus Public Message Access Protocol** that allows for the retrieval of unsolicited messages via an TAP interface.

6.3.2 TAP Compatibility

A TAP port used for the Nexus standard shall implement all the mandatory features of a standard TAP port, including the “BYPASS” and “IDCODE” instructions. A 16-state TAP state machine will be used per the IEEE-1149.1 standard as illustrated in **Figure 6-4**.

Refer to the IEEE Std 1149.1™-2001 (R2008) for the timing characteristics of signals, protocols, and functionality of the 1149.1 interface. Refer to the IEEE Std 1149.7™-2009 for the timing characteristics of signals, protocols, and functionality of the 1149.7 interface. These standards provide a complete description of electrical and pin protocol compliance requirements.

Additional information on the TAP 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**.

6.3.3 TAP Pin Functions

The IEEE 1149.1 TAP pins are described in **Table 6-7**.

TAP Pin	Pin Description
TDI	Test Data Input (TDI) is an input pin that provides for serial movement of data into the TAP port.
TDO	Test Data Output (TDO) is an output pin that provides for serial movement of data out of the TAP port. All target accesses initiated via the TAP port should be transmitted by the target via TDO (not via auxiliary output port).
TCK	Test Clock (TCK) is an input pin that provides the clock for the TAP port.
TMS	Test Mode Select (TMS) is an input pin that provides access to the TAP state machine.
$\overline{\text{TRST}}$	Test Reset ($\overline{\text{TRST}}$) is an optional pin that provides for asynchronous initialization of the TAP controller.
$\overline{\text{RDY}}$	Ready ($\overline{\text{RDY}}$) is an optional output pin used to accelerate data accesses through the TAP port (Refer to 6.3.5 - Optional Ready (RDY) Output Pin).

Table 6-7—TAP (JTAG) Pins

Four/five pin version of 1149.7 TAP pins are described in **Table 6-7**. The two pin version of IEEE 1149.7 TAP pins is shown in **Table 6-8**. The Ready function of the 4/5pin TAP is included in the IEEE 1149.7 two pin protocol. The optional $\overline{\text{TRST}}$ pin is also shown. The function of this pin is also available using only the TCKC and TMSC pins.

TAP Pin	Pin Description
TCKC	Test Clock Compact (TCKC) is an input pin that provides the clock for the TAP port.
TMSC	Test Mode Select Compact (TMSC) is an input pin that provides access to the TAP state machine.
$\overline{\text{TRST}}$	Test Reset ($\overline{\text{TRST}}$) is an optional pin that provides for asynchronous initialization of the TAP controller.

Table 6-8—IEEE 1149.7 TAP (JTAG) Pins

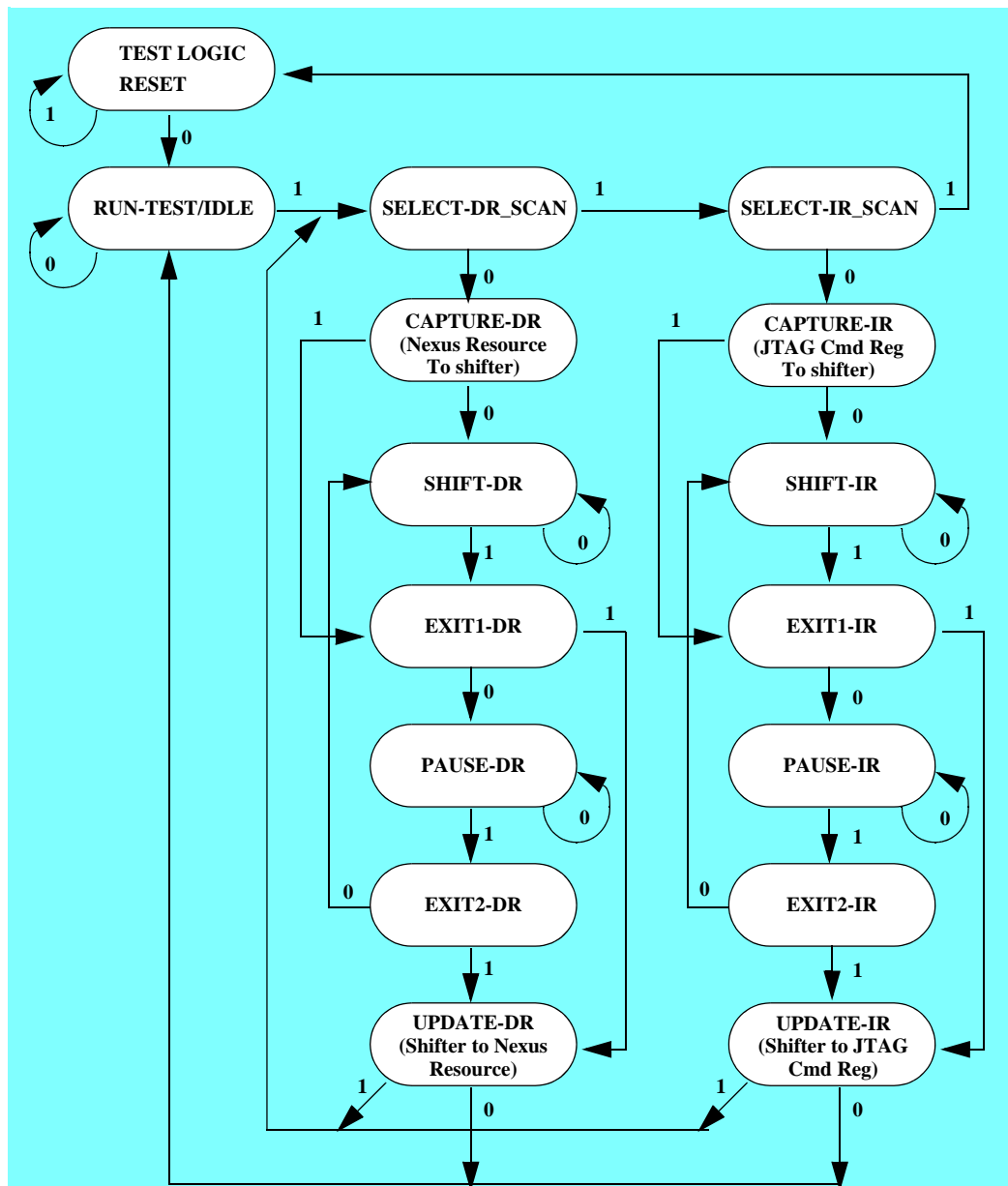


Figure 6-4—Sixteen-State TAP Finite State Machine

6.3.4 Accessing the TAP Device ID

Assertion of a power-on-reset signal on the embedded processor or the $\overline{\text{TRST}}$ pin causes the TAP 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 TAP 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 TAP controller is decoding the “IDCODE” instruction. All NRRs must be accessible through the TAP port independent of the state of the target processor.

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

To increase the transfer rate of the TAP 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 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 TAP state machine has reached the CAPTURE_DR state.

The RDY pin may also be used for Nexus Public Messages as described in **6.3.8 - Accessing Nexus Public Messages via the TAP Port**.

6.3.6 Accessing NRRs via the TAP 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 TAP port. This is especially useful for embedded processors that implement TAP pins for static debug and/or boundary scan.

The mechanism defined in this section can be used in lieu of implementing the Auxiliary Access Public Messages defined in **SECTION 4 - Nexus Public Messages**.

6.3.6.1 NRR Access Protocol

Access to NRRs is enabled when the TAP controller is decoding a vendor-defined “NEXUS-ACCESS” instruction entered via the SELECT-IR_SCAN path. When the TAP 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 (TAP only) implementations that may not implement the $\overline{\text{EVTI}}$ pin. Refer to **7.1.2 - Reset for TAP Implementations**.

When the “*NEXUS-ACCESS*” instruction is being decoded by the TAP controller, the TAP 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 6-5**.

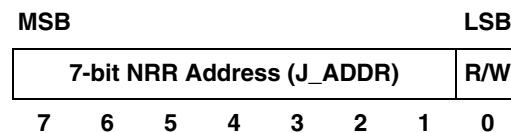


Figure 6-5—TAP Controller Command Input

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 TAP shifter during the CAPTURE-DR state. When writing to an NRR, the value will be loaded by the TAP 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 6-6** illustrates the relationship between an TAP state machine and a Nexus controller state machine.

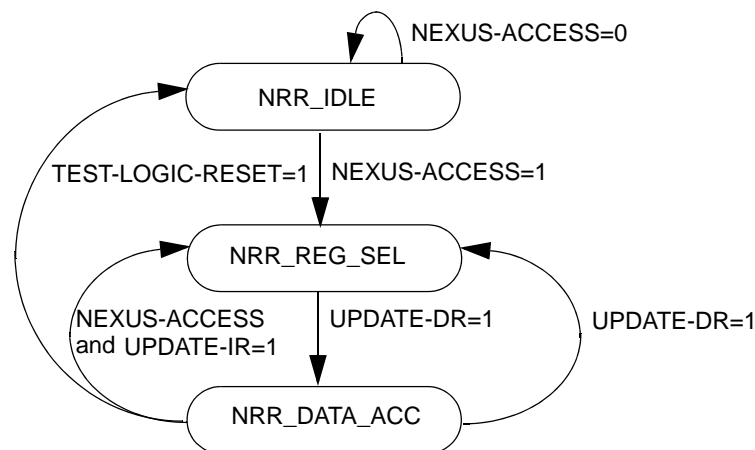


Figure 6-6—TAP State Machine Relationship to Nexus Controller State Machine

6.3.6.2 NRR Access Status (Optional)

In order to reduce the number of clock cycles required to poll registers to retrieve status information, an additional (optional) TAP 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 TAP 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 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 TAP 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 **TAP** 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.

6.3.7 Read/Write Access via the TAP 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 auxiliary or TAP port. When using the TAP 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 TAP port and perform the next read/write access. Use of a RDY pin permits data transfers in $[16 + (\text{data width})]$ TCKs, assuming the TAP controller starts from and ends in the SELECT-DR_SCAN state.

If a 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.

6.3.8 Accessing Nexus Public Messages via the TAP Port

Nexus Public Messages may be read from or written to the target via the TAP port. The method outlined in the following sections 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 output port as well as the TAP port, the selection mechanism is vendor defined.

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

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

Output Public Messages are generated by the target processor. Because the TAP 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 TAP controller.

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

6.3.8.2 Nexus Public Message Access Protocol

The IPMR and OPMR are used to transmit Nexus Public Messages via the TAP port instead of the AUX port. These registers can be viewed as partitioned into slots. Each slot contains a predetermined number of equivalent AUX bits and 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 MSE bits for each implementation is vendor defined. **Table 6-9** and **Table 6-10** show an example 32-bit OPMR/IPMR implemented using 6-bit and 14-bit AUX equivalents, as well as the two-pin MSE option.

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 6-9—IPMR/OPMR Register (6-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"

Table 6-10—IPMR/OPMR Register (14-bit AUX equivalent)

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 6-11 below shows a typical Indirect Branch Message using two-pin MSE0 and six-pin MDO. **Table 6-12** shows how this AUX message is formatted for an OPMR transmission via the TAP port.

Clock	Slot	AUX[5:0]						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 6-11—Indirect Branch - AUX 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					

Table 6-12—Indirect Branch - OPMR Example

6.3.8.3 Using 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 RDY pin. If RDY is a logic 1, the external TAP controller may terminate OPMR shifting. If the 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 Public Message is ready for retrieval when 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 6-4** illustrates the 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.

6.3.9 Sample TAP Access Sequences

Table 6-13 illustrates the TAP sequence required to read the Device IDCODE immediately after assertion of the TRST pin or after 5 TCKs with the TMS pin at logic 1.

Step	TMS	TAP State	Nexus State	Description
1	1	TEST-LOGIC-RESET	NRR_IDLE	TAP controller in reset state.
2	0	RUN-TEST-IDLE	NRR_IDLE	IDCODE loaded into TAP 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 TAP 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	TAP controller ready for instruction.

Table 6-13—TAP Sequence to Read Device IDCODE After TRST Pin Assertion

Table 6-14 illustrates the TAP sequence to select the Nexus controller.

Step	TMS	TAP State	Nexus State	Description
1	0	RUN-TEST-IDLE	NRR_IDLE	TAP 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 TAP shifter is ready. Shift $N-1$ bits of size of vendor-defined “NEXUS-ENABLE” 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	TAP 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 6-14—TAP Sequence to Initiate Nexus Communication

Table 6-15 illustrates an TAP sequence that will write a 32-bit value to an NRR.

Step	TMS	TAP State	Nexus State	Description
1	0	RUN-TEST-IDLE	NRR_REG_SEL	TAP controller in idle state.
2	1	SELECT-DR_SCAN	NRR_REG_SEL	
3	0	CAPTURE-DR	NRR_REG_SEL	TAP shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.

Table 6-15—TAP Sequence to Write to an NRR

Step	TMS	TAP State	Nexus State	Description
4	0	SHIFT-DR	NRR_REG_SEL	TDO becomes active, and NRR address and write bit is shifted in.
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 reg.
7	1	SELECT-DR_SCAN	NRR_DATA_ACC	Second pass through SELECT-DR_SCAN.
8	0	CAPTURE-DR	NRR_DATA_ACC	TAP 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	
N-1 TCKs			NRR_DATA_ACC	TDO becomes active and outputs current value of register while new value is shifted in through TDI.
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.
12	0	RUN-TEST/IDLE	NRR_REG_SEL	TAP 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 6-15—TAP Sequence to Write to an NRR (Continued)

Step	TMS	TAP State	Nexus State	Description
1	1	SELECT-DR_SCAN	NRR_REG_SEL	Starting point of this example.
2	0	CAPTURE-DR	NRR_REG_SEL	TAP shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.
De-assert RDY pin and set SB bit				
3	0	SHIFT-DR	NRR_REG_SEL	TDO becomes active, and Nexus R/W Data 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 R/W Data Register shifted from TDI.
5	1	UPDATE-DR	NRR_REG_SEL	Nexus controller decodes and selects reg.
6	1	SELECT-DR_SCAN	NRR_DATA_ACC	Second pass through SELECT-DR_SCAN.
Wait for RDY pin assertion or SB bit clear (read operation)				
7	0	CAPTURE-DR	NRR_DATA_ACC	TAP shifter may be loaded with last value of register being decoded by Nexus controller or Nexus status information.
De-assert RDY pin and set SB bit				

Table 6-16—TAP Sequence for Read/Write Access with RDY Pin

Step	TMS	TAP State	Nexus State	Description
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	TAP controller returns to SELECT-DR_SCAN state for new NRR select. Total number of TCKs = 48 in this example.
Wait for RDY pin assertion or SB bit clear (write operation)				
The TAP state machine can be taken either to the SELECT-DR_SCAN state if another operation is pending or to the RUN-TEST-IDLE state.				

Table 6-16—TAP Sequence for Read/Write Access with RDY Pin (Continued)

6.4 High Speed Serial Interface (Aurora)

For higher performance embedded processors, traditional TAP or parallel (AUX) interfaces may not provide sufficient bandwidth in transmitting trace information from multiple on-chip clients. IEEE-ISTO 5001-2012 introduces a high speed serial protocol (serial AUX). The Aurora protocol developed by Xilinx™ will be used to send and receive debug information over the high-speed serial link.

Aurora is an industry standard (open) light-weight link protocol - ideal for a high-speed serial debug link. Nexus adheres to the Xilinx *Aurora Protocol Specification V2.x*. More detail on the protocol itself can be found on Xilinx's website.

6.4.1 Overview

The Nexus serial AUX interface provides two-way high-speed serial communication with an external debugger. The following is a high level summary of features supported by IEEE-ISTO 5001-2012:

- *Xilinx Aurora Protocol Specification V2.x compliant*
- *Support for Simplex (Tx only) and Duplex (Tx and Rx) modes*
- *Support for 1 to 8 lanes of transmit (Tx) data*
- *Support for 1 to 4 lanes of receive (Rx) data*
- *Support for multiple clock frequencies (see Table xx)*
- *Support for both duplex mode and static (timer-based) channel training*
- *(Optional) Support for Cyclic Redundancy (Error) Checking (CRC)*
- *(Optional) Support for Native and User Flow Control*

6.4.2 Aurora Transmit (Tx)

The Nexus standard supports up to eight (8) Aurora transmit lanes. In order to support data transmission as described in the Aurora Protocol specification, the embedded processor must perform the data-packaging, and protocol translations to prepare the data stream for transmission. It must also support clock-compensation, data framing, data striping (across all available lanes), channel bonding (part of the training procedure), 8b/10b encoding/decoding and data alignment.

Nexus modules may optionally support native flow-control (NFC) and user flow control (UFC) as defined by the Aurora specification.

6.4.3 Aurora Receive (Rx) (optional)

The Nexus standard supports up to four (4) Aurora data receive lanes as well as data transmission lanes. The Aurora-in path is predominantly used for run-control and debug access (read and write) to memory-mapped facilities on the embedded processor.

In this mode, processors perform the same functions as they do in transmit mode, only in the opposite direction: 10b/8b decoding, protocol demuxing, data destriping and deframing.

6.4.4 Aurora Tx/Rx Configurations

Nexus supports three combinations of transmit (Tx) and receive (Rx) configuration: simplex, balanced duplex, and unbalanced duplex modes. Description for the three modes is captured in **Table 6-17**.

Tx / Rx Mode	# Lanes Supported	Description
Simplex	x1 to x8	In this mode, only transmit lanes are supported. Control and status is performed over the TAP interface. From one (1) to eight (8) transmission lanes are supported.
Balanced Duplex	x1 to x4	In this mode, there is an equal number of transmit and receive lanes. From one (1) to four (4) full-duplex lanes are supported.
Unbalanced Duplex	x1 to x8 (Tx) x1 to x4 (Rx)	In this mode, an implementation utilizes more lanes for data transmission than it does for reception. When in an unbalanced mode, the channel can be seen as consisting of a minimum number of full-duplex lanes plus any additional transmit-only simplex lanes for added trace bandwidth. A block diagram depicting this mode can be found in Figure 6-7 - Example Unbalanced Channel Diagram

Table 6-17—Supported Aurora Modes

A top-level block diagram of an example implementation of a x2-Tx, x4-Rx configuration (tool side) is shown in **Figure 6-7**

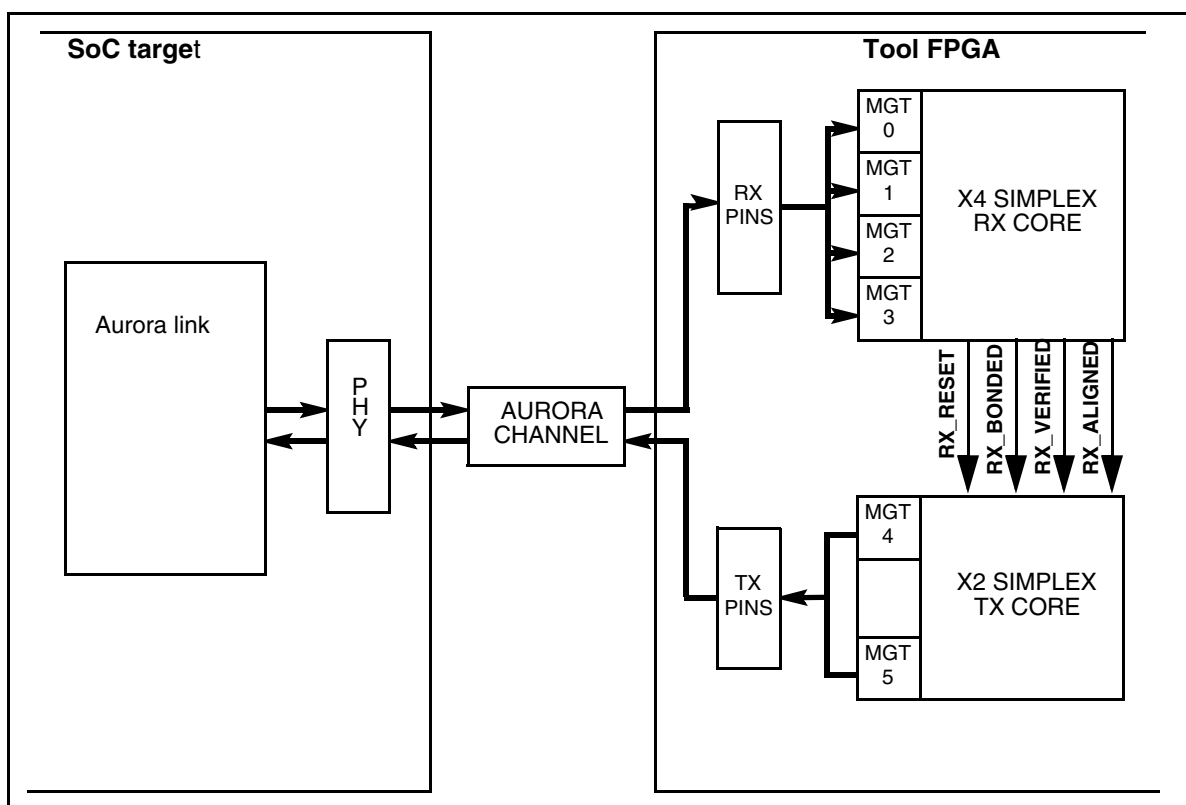


Figure 6-7—Example Unbalanced Channel Diagram

NOTE

The Aurora interface is inherently based on bytes. Byte ordering can be fined by the MCU vendor to be either fully Big-Endian or Little-Endian byte ordering. Typically Little-Endian is used since the reference design from Xilinx uses this format.

6.4.5 Tx / Rx Data Types

Aurora data is transferred across the Aurora channel in streams of symbol-pairs. There are six types of data which are transmitted over the Aurora channel and they are listed according to their priority of transmission in **Table 6-18**.

Aurora Data Type	Priority	Description
Clock compensation sequences	Highest	Sequences of control symbols used to prevent overrun of the receiver due to differences in the clock rate between channel partners

Table 6-18—Aurora Transmission Priorities

Aurora Data Type	Priority	Description
Initialization sequences		Four ordered sets of symbols which prepare an Aurora channel for data transmission
Native Flow Control PDUs		Flow control sequences generated and interpreted by the embedded processor
User Flow Control PDUs		Flow control sequences generated and interpreted by the user application
Channel PDUs		User data encapsulated for transmission over the Aurora channel
Idle sequences	Lowest	Sequences of control symbols that are transmitted whenever there is no data

Table 6-18—Aurora Transmission Priorities

Table 6-19 lists the values of the special ordered sets of Aurora control characters. Refer to the Aurora Protocol Specification for more detail on the character sets.

Character	Description	8b/10b Encoding
/K/	Comma.	/K28.5/
/R/	Skip.	/K28.0/
/A/	Channel Bonding.	/K28.3/
/I/	IDLE characters transmitted during pauses and when nothing else to transmit.	/K/, /R/, /A/ sequence
/CC/	Clock Compensation.	/K23.7/K23.7/
/P/	PAD for odd number of octets in data frame.	/K28.4/ (same as /SUF/)
/SCP/	Start of Channel PDU.	/K28.2/K27.7/
/ECP/	End of Channel PDU	/K29.7/K30.7/
/SNF/	Start of Native Flow Control	/K28.6/
/SUF/	Start of User Flow Control	/K28.4/ (same as /P/)
/SP/	Sync and Polarity	/K28.5/D10.2/D10.2/ D10.2/
/SPA/	Sync and Polarity Acknowledge	/K28.5/D12.1/D12.1/ D12.1/
/V/	Verification	/K28.5/D8.7/D8.7/D8.7/

Table 6-19—Aurora Control Characters

6.4.6 Channel Initialization (Link Training)

Before beginning functional operation, the Aurora link must go through a training process to insure that both sides (target and tool) can correctly reconstruct the transmitted data stream. The training process is outlined in the Aurora Protocol Specification, and consists of three steps: lane initialization, channel bonding, and channel verification. Training proceeds once the link powers up, and must complete successfully before data transmission can occur. To successfully progress through training, the channel partners must be able to communicate their status to one another.

Nexus supports training the Aurora link either in a simplex or duplex configuration.

Duplex training is the more robust method and will result in a more reliable link due to the ability to receive handshake responses on the Rx lanes.

6.4.6.1 Simplex Mode Training

For transmit (Tx) only Nexus implementations (that use JTAG for the back channel), there are effectively two ways to perform the training sequence. The Rx (receive) core can communicate its training status to the Tx (transmit) core through the simplex training sideband signals that are included as part of the Aurora specification: RX_ALIGNED, RX_BONDED, RX_VERIFY, and RX_RESET. The embedded processor would need to support these additional pins.

Training can also be achieved through a back channel using the TAP interface to access the Nexus defined NRRs for Aurora (see **APPENDIX B - Recommendations for Access to Control and Status Registers**). The back channel is used to begin initialization and to indicate that the receiver is ready for channel verification and data reception. During transmission it also shall indicate a loss of synchronization and make requests for channel reset.

In order to avoid potential time-outs during the initialization sequence, a set of counters can be programmed to transition through the training stages (after the initialization process has started). Optional Nexus NRRs are defined for controlling the static training sequence. A counter is programmed for each stage of training and as each timer expires, the training sequence can proceed to the next stage. For all implementations, the maximum counter value should correlate to the time-out value supported by the logic that supports training the Aurora link.

6.4.6.2 Duplex Mode Training

If the channel consists of an equal number of full-duplex lanes (balanced duplex), then training becomes fairly straightforward. The channel partners can communicate via the channel as it is coming up, and the training is a self-contained process between the target and external tool's Aurora cores. Implementations supporting an uneven number of Tx / Rx lanes (unbalanced duplex) is non-standard (w.r.t. the Aurora protocol), but the same basic training procedures can be used.

In duplex-mode training, the channel partners communicate their status by varying the characters they send: in lane-alignment, they send data plus commas; in channel bonding they send idles, and in verification they send idles plus /V/'s. So the embedded processor will know that its partner has passed lane alignment when it starts receiving idles over its two duplex lanes.

In this configuration, the simplest implementation for the tool side of the channel calls for using two simplex cores: one to control the transmit-only lanes, and another for additional receive-only lanes. The Rx (receive) core will then communicate its training status to the

Tx (transmit) core through the simplex training sideband signals outlined above (**Section 6.4.6.1 - Simplex Mode Training**): RX_ALIGNED, RX_BONDED, RX_VERIFY, and RX_RESET. In this implementation, these sideband signals are used to communicate between cores on the same side of a channel, rather than across the channel. The tool's Rx core controls the entire training process: as it detects that each stage of the training process has completed, it communicates this fact to the Tx core, which then begins transmitting a new sequence across the channel. On the target side, the core waits until it knows the tool side has moved to the next stage before moving itself; in this way, every state transition is controlled by the tool's Rx core.

6.4.6.3 Aurora Training Sequence

The Aurora link training procedure is broken down into three stages:

1. **Lane Initialization (Alignment):** each transmitter sends a sequence consisting of data and comma characters; the receive side must analyze the incoming serial data stream and determine the character boundaries by attempting to detect commas (/K/ characters)
2. **Channel Bonding:** each transmitter simultaneously sends a series of idle sequences, which include the /A/ character; the receive side will check to see if it sees the /A/ on every lane on the same cycle, skewing the lanes as necessary to bring them into alignment.
3. **Channel Verification:** each transmitter simultaneously sends the verification sequence (/V/) on all lanes, embedded in a series of idles; if the receive side detects the sequence on all lanes at the same cycle, the channel is brought up.

Status on the training steps can be read from the TSTAT bits within the Aurora Link Status (ALS) Register.

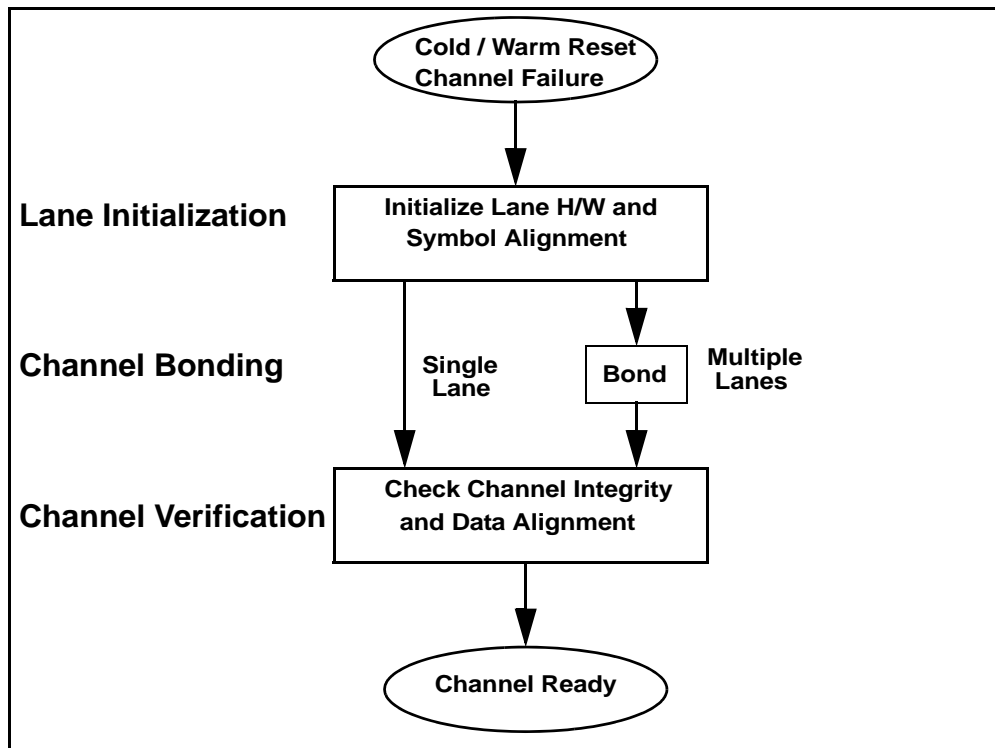


Figure 6-8—Aurora Channel Initialization

6.4.6.4 Lane Initialization (Alignment)

Lane initialization synchronizes the trace port transmitter with the probe receiver through the transmission of a known pattern (/SP/ and /SPA/). In a full-duplex configuration, the lane initialization sequence typically involves a series of handshakes with the lane partner to indicate that the partner is receiving and transmitting data correctly. If more than one full-duplex lane is enabled, the lane initialization logic should broadcast the initialization sequence on all lanes, and wait for the appropriate sequence from all lanes before progressing to the next step.

In a simplex configuration, programmable counters can be used in place of handshaking to progress through the initialization steps. Once the counter duration expires, the initialization process progresses to the next step. The timer duration is programmed via the Aurora Training Control (ATC) Register.

The lane initialization procedure synchronizes each lane transceiver with its partner upon reset or channel failure. Channel failures occur for excessive channel data errors or protocol violations.

6.4.6.5 Channel Bonding

Channel bonding is an optional step used to align multi-lane configurations to ensure that transmissions are synchronized across multiple lanes in a channel. Since the transmit data is broken up and striped across the available lanes, it is important for their reception to be aligned so that the data can be combined back together in the correct order. Channel bonding cannot begin until all lanes have initialized and the link counter has transitioned the transmitter to the channel bonding state.

The channel bonding sequence transmits an /I/ ordered set and expects to receive four (4) /A/ ordered sets on each receive lane at the same time. Once the channel bonding sequence has been repeated a pre-determined number of times (vendor-defined option), a time-out occurs, causing the entire initialization sequence to reset.

NOTE

For single lane applications the channel alignment state is bypassed.

6.4.6.6 Channel Verification

Channel verification verifies to the ability of the channel to transfer valid data across the interface. The verification process uses a known data sequence. The sequence consists of sixty (60) idle symbols (/I/) followed by the /V/ ordered set. The sequence is repeated a minimum of eight (8) times, during which at least four (4) sequences must be detected on the receive lanes. In a simplex configuration, the receive lanes are ignored. Refer to the “8B/10B Channel Verification” section of the Aurora protocol specification for details of the process.

As soon as the channel verification pattern has been transmitted, the transmitter assumes that the receiver has successfully verified and subsequently moves to the Channel Up state. The transmitter shall commence sending idle patterns /I/ until the TCR[VERIF] bit is set by the receiver.

If the receiver fails to successfully receive the verification pattern as expected after the channel alignment then it should clear the TCR[INIT] bit forcing the transmitter to return to the reset and lane initialization state and repeat the initialization process.

When the receiver is fully ready to receive data, (all lanes initialized, bonded, verified, and logic ready to receive packets) it shall write the TCR[VRF] bit. This bit indicates that the transmitter can commence sending trace data.

Once the channel is verified and the Aurora transmitter is ready the Channel Up status is reflected by the TSR[CU] bit.

6.4.7 Recommended Transmission Procedure

The frame of data transmitted on the Aurora channel as a channel PDU according to the

following procedure:

1. **CRC calculation (recommended)**
2. **Padding (if necessary)**
3. **Encapsulation with channel PDU delimiters**

If implemented, CRC for each frame of data should be calculated according to the CRC-16-CCITT polynomial: $G(x) = X^{16} + X^{15} + X^2 + 1$

The Aurora channel requires that all transmissions contain an even number of symbols. Once the CRC octets are added, the frame is encapsulated with ordered sets which delimit the beginning and end of the channel PDU for identification by the channel partner. The SCP and ECP ordered sets are used to indicate start-of-channel and end-of-channel, respectively. **Figure 6-9** shows how user PDUs are formatted for transmission.

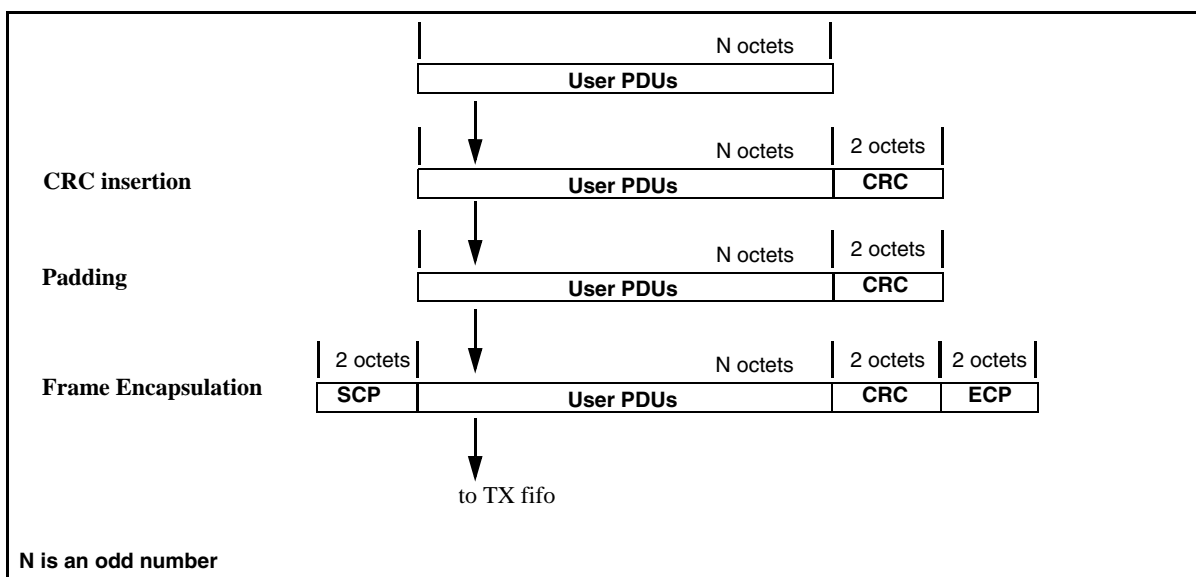


Figure 6-9—Example Data Transmission

NOTE

If an even amount of transmission data is generated (i.e. 8-bytes at a time), and CRC is encoded into 2 bytes, padding will not be necessary. Received data, however, may contain PAD characters and should be processed according to the Aurora protocol specification.

6.4.8 Recommended Reception Procedure

Data received is processed according to the following procedure:

1. *Link Layer Stripping*
2. *Pad Stripping*
3. *CRC computation*

After the physical layer (PHY) decodes the incoming 8b/10b symbols, the channel PDU encapsulation (SCP, ECP ordered sets), and any embedded IDLE sequences from the channel PDU should be removed. It is a requirement that the IDLE sequences contain an even number of symbols and that any idle sequence begin after an even number of symbols in the channel PDU.

Any existing PAD symbol should also be removed from the channel PDU. If CRC is enabled, the CRC is computed for the channel PDU and compared with the CRC information contained in the frame of data and an error is flagged if they do not match.

Figure 6-10 shows how received data is processed.

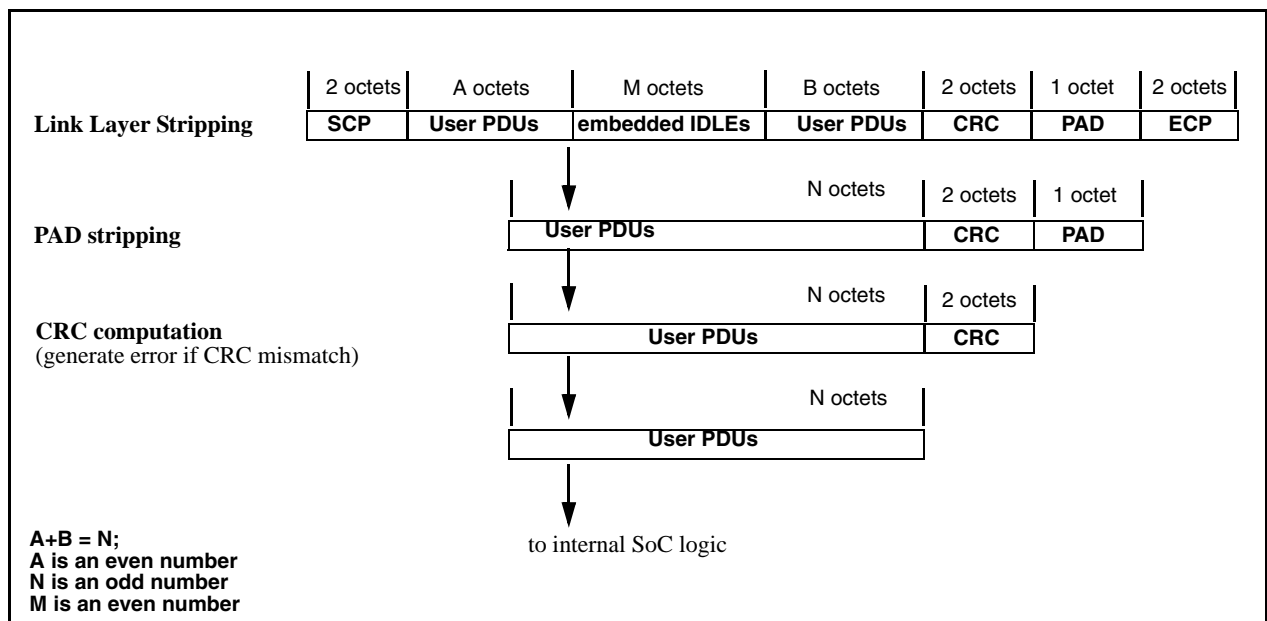


Figure 6-10—Example Data Reception

6.4.9 Aurora Flow Control

The Nexus standard supports both Native Flow Control (NFC) and User Flow Control (UFC) as defined by the Aurora specification to accommodate for any differences in

transmission rate between channel partners.

6.4.9.1 Native Flow Control

NFC is the built-in link-layer flow control mechanism. NFC PDUs are typically generated when the receiving partner's buffers are close to being filled.

Native Flow Control is triggered based on the state of the RX fifo of either of the channel partners in an Aurora communication link. NFC PDUs are generated whenever the RX fifo is close to overflowing. The NFC PDU's are transmitted to request the channel partner to pause transmission for a specified number of symbol times. When an NFC PDU is received by a channel partner, IDLE sequences should be transmitted for the specified duration. NFC or UFC PDUs may also be transmitted during the interval as needed since they will not be stored in the RX fifo.

If a user PDU transmission is in progress when an NFC request is received, configuration information can be set via ALC[NFCM] to either wait for the current user PDU to complete (completion mode), or interrupt the user PDU (immediate mode) to pause the transmission. In immediate mode, the maximum round trip delay between the NFC PDU request and the first pause sequence arriving at the originating channel partner must not exceed 256 symbol times.

NFC PDUs are composed of two octets, the SNF (K28.6) - start of native flow control octet, and the NFC command octet. The NFC command octet specifies the number of IDLE characters the channel partner must send in response to the NFC PDU. **Figure 6-11** shows the NFC PDU format.

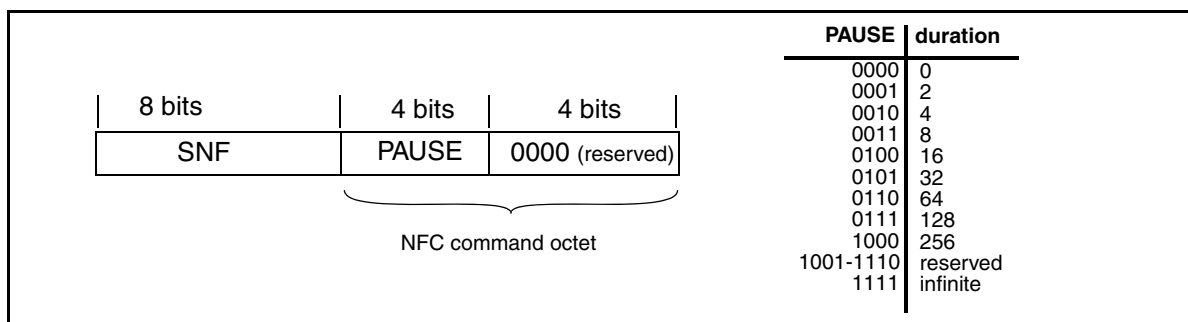


Figure 6-11—NFC PDU Format

6.4.9.2 User Flow Control (Optional)

UFC messages are typically generated by the user application. Although UFC messages are usually not processed by the Aurora interface, UFC messages can be used as a substitute for NFC in unbalanced configurations of the Aurora channel partner (i.e. the channel partner may be configured in separate receive/transmit simplex blocks and therefore cannot generate NFC).

Once UFC transmission has begun, it can not be interrupted by NFC PDUs, IDLE sequences, or Clock Compensation sequences. UFC PDUs consist of 4 to 18 octets. They begin with the SUF (K28.4⁶) - start of user flow control octet, followed by a UFC command octet which specifies the length of the UFC message which follows. The content of the final “UFC Message” octets is vendor-defined.

Figure 6-12 shows the format of the UFC PDU.

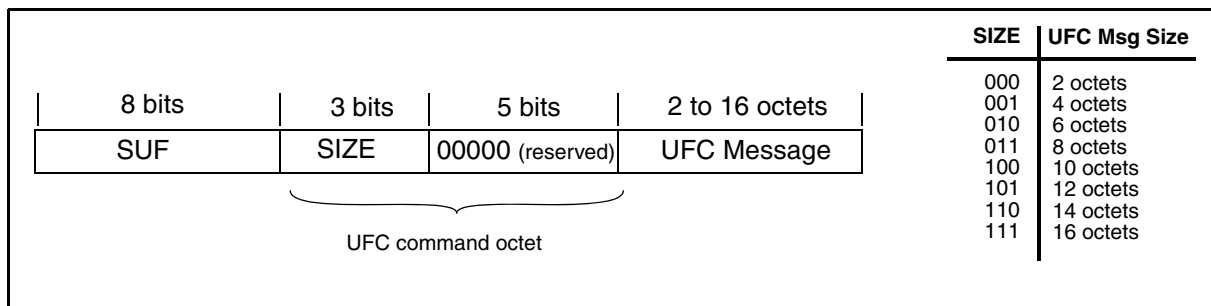


Figure 6-12—UFC PDU format

6.4.10 Error Handling

The Aurora channel is susceptible to hard and soft transmission errors. Optionally, CRC errors or Framing errors can also be detected.

Hard errors are considered unrecoverable and cause the channel to immediately be re-initialized. Any data transmission in progress when a hard error occurs is corrupt and should be discarded. Upon hard error occurrence, the ALS[HERR] status bit is set. The ALS[HETP] bits attempt to distinguish the specific type of error that triggered the hard error. A hard error occurs under the following conditions:

- *An overflow/underflow occurs in the transmit or receive elastic buffers*
- *The soft error count exceeds the maximum limit*
- *The channel partner undergoes a reset (indicated by the receipt of initialization sequences)*
- *The physical connection between channel partners is broken*

Soft errors are typically transient bit errors occurring on a lane. Soft errors do not require the channel to reset unless they exceed a certain threshold as described by the hard error conditions above. The following soft errors can occur:

- *Disparity errors - occur when the running disparity in the physical layer exceeds the threshold*

⁶Note: Although K28.4 is the same encoding used for PAD, the PAD symbol is always followed by a control character and can therefore be distinguished from SUF.

- *Symbol errors - occur when an illegal symbol value is transmitted or received*

If a soft error occurs, the ALS[SERR] bit is set. The ALS[SETP] bits attempt to distinguish the specific type(s) of errors that triggered the soft error. Soft errors will automatically be classified as a hard error once a certain threshold is reached.

SECTION 7

Implementation Topics

7.1 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 auxiliary output port. 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.

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

For Nexus implementations that are comprised 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 comprise a pull-up resistor with the following reset configuration states.

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

Table 7-1

7.1.2 Reset for TAP Implementations

For Nexus modules that use a combination of auxiliary port and TAP port or that use an TAP 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 $\overline{\text{TRST}}$. Refer to **A.4 - AC Electrical Characteristics - IEEE 1149.1**

Interface for detail on $\overline{\text{EVTI}}$ electrical characteristics.

- $\overline{\text{EVTI}}$ assertion during **TAP** “TEST-LOGIC-RESET” state

Because the $\overline{\text{TRST}}$ pin is optional, this mechanism allows Nexus modules implementing the **TAP** port without $\overline{\text{TRST}}$ to have a mechanism to enable Nexus. The “TEST-LOGIC-RESET” state can be reached by cycling through the JTAG 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 **6.3.7 - Read/Write Access via the TAP Port** for detail on how the “NEXUS-ACCESS” IR value is used to access Nexus registers.

7.1.3 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 auxiliary output port. 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.

7.2 Multiple Processor/Client 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. The SRC field size should be the same for all clients sharing the same AUX port.

7.3 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 Read Messages and 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 Correlation Messages

It is possible to use the Program 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 **4.3.16 - 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).

7.4 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.

7.5 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.

7.6 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

Four possible connector options are available to be used for an auxiliary port only connection or a combined IEEE-1149.1-Auxiliary connection. These four options use AMP System 50 connectors, GlenAir MicroD connectors, Mictor connectors, or SAMTEC ERx8 connectors.

The AMP System 50 connector option is scalable and can be configured as 26, 40, or 50 pins.

The Tyco Mictor 38-pin connector option is also scalable and can use either one or two connectors, depending on the number of signals required.

The GlenAir MicroD connector option provides a robust connector and can have 51 pins.

The Samtec ERx8 connectors provide a stable connector option that supports high speed trace that is configurable for either high speed parallel trace or high speed serial trace with optional JTAG.

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

Table A-1 lists all of 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)
- HS = Samtec high speed serial connector
- HP = Samtec high speed parallel connector

Number of pins is 2 by 10, 2 by 11, 26, 2 by 17, 40, 2 by 23, 2 by 25, 50, 51, 38, 2

by 35, 38 times 2 connectors, 2 by 40.

Interface type is

C = Combined IEEE-1149.1-Auxiliary (Out)

A = Auxiliary only (In and Out)

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			Application
AMP Style	S26x	1-104068-2		AMP System 50	Low performance - 1 MDO signal
	S40x ¹	104549-6		AMP System 50	Low performance - 6 MDO signals
	S50x	104549-7		AMP System 50	Low performance - 8 MDO signals
Robust	R51x	MR7580-51P2BNU		Glenair MicroD	Low Performance Robust 8 MDO signals
Mictor	M38x	767054-1		AMP MICTOR	Medium performance - 8 MDO signals
	M38-2x	Two times 767054-1		AMP MICTOR	Medium performance - 16MDO signals
ERx8	HSxx HS22 HS34 HS46 HS70	Vertical Mount ASP-137969-01 2x11 position ASP-137973-01 2x17 position ASP-130368-01 2x23 position ASP-135029-01 2x35 position	Right Angle Mount ASP-149700-01 2x11 position T.B.D. 2x17 position ² T.B.D. 2x23 position ² T.B.D. 2x35 position ²	Samtec ERF8 Series	High Performance Serial Trace
ERx8	HP20x HP25x HP80x	ASP-148421-01 2x10 position ASP-148422-01 2x25 position ASP-148424-01 2x40 position	T.B.D. ²	Samtec ERF8 Series	High Performance Parallel Trace

1. Not recommended.

2. Part numbers will be assigned if sufficient demand exists to tool this connector.

Table A-2—AMP System 50 Definition - Sxxx

S50	S40	S26	Nexus Combined Signal C	Nexus Auxiliary Signal A	I/O	Pin Number	Pin Number	I/O	Nexus Auxiliary Signal A	Nexus Combined Signal C
A50	A40	A26	UBATT	UBATT	OUT	1	2	OUT	UBATT	UBATT
			VSTBY	VSTBY	OUT	3	4	I/O	GEN_IO0	GEN_IO0
			TDO	GEN_IO1	I/O	5	6	I/O	GEN_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	GEN_IO3	GEN_IO3
			MDO1	MDO1	OUT	27	28	—	GND	GND
			MDO2	MDO2	OUT	29	30	—	GND	GND
			MDO3	MDO3	OUT	31	32	—	GND	GND
			GEN_IO4	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
			GEN_IO5	MDI2	IN	45	46	—	GND	GND
			GEN_IO7	MDI3	IN	47	48	—	GND	GND
			GEN_IO8	GEN_IO8	I/O	49	50	—	GND	GND

Table A-3—GlenAir Robust Combined Definition - RxxC

Nexus Combined Robust Connector R51C Robust MR7580-51P2BNU		
		1 UBATT
36 GND	19 MDO0	2 UBATT
37 MDO4	20 GND	3 VSTBY
38 GND	21 MCKO	4 GEN_IO4
39 MDO5	22 GND	5 TDO
40 GND	23 EVTO	6 /RDY
41 MDO6	24 GND	7 /RESET
42 GND	25 /MSEO0	8 VREF
43 MDO7	26 GEN_IO8	9 /EVTI
44 GND	27 MDO1	10 GND
45 GEN_IO7	28 GND	11 /TRST
46 GND	29 MDO2	12 GND
47 GEN_IO9	30 GND	13 TMS
48 GND	31 MDO3	14 GND
49 GEN_IO10	32 GND	15 TDI
50 GND	33 GEN_IO5	16 GND
51 GEN_IO0	34 GND	17 TCK
	35 /MSEO1	18 GND

Table A-4—GlenAir Robust Auxiliary Definition RxxA

Nexus Auxiliary Robust Connector R51A Robust MR7580-51P 2BNU		
		1 UBATT
36 GND	19 MDO0	2 UBATT
37 MDO4	20 GND	3 VSTBY
38 GND	21 MCKO	4 GEN_IO0
39 MDO5	22 GND	5 GEN_IO4
40 GND	23 /EVTO	6 GEN_IO2
41 MDO6	24 GND	7 /RESET
42 GND	25 /MSEO0	8 VREF
43 MDO7	26 GEN_IO8	9 /EVTI
44 GND	27 MDO1	10 GND
45 MDI2	28 GND	11 /RSTI
46 GND	29 MDO2	12 GND
47 MDI3	30 GND	13 /MSEI
48 GND	31 MDO3	14 GND
49 GEN_IO10	32 GND	15 MDI0
50 GND	33 MDI1	16 GND
51 GEN_IO0	34 GND	17 MCKI
	35 /MSEO1	18 GND

Table A-5—MICTOR Connector M38x and 1/2 of M38-2x

Combined M38C or M38-2C	Aux Only M38A or M38-2A					Aux Only M38A or M38-2A	Combined M38C or M38-2C
/MSEO0	/MSEO0	OUT	38	37	OUT	VSTBY	VSTBY
/MSEO1	/MSEO1	OUT	36	35	IN/OUT	GEN_IO4	GEN_IO4
MCKO	MCKO	OUT	34	33	OUT	UBATT	UBATT
/EVTO	/EVTO	OUT	32	31	OUT	UBATT	UBATT
MDO0	MDO0	OUT	30	29	IN/OUT	MDI1	GEN_IO5
MDO1	MDO1	OUT	28	27	IN/OUT	MDI2	GEN_IO2
MDO2	MDO2	OUT	26	25	IN/OUT	MDI3	GEN_IO0
MDO3	MDO3	OUT	24	23	IN/OUT	GEN_IO10	GEN_IO10
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	GEN_IO_2	IN/OUT	14	13	IN/OUT	GEN_IO9	GEN_IO9
VREF	VREF	OUT	12	11	IN/OUT	GEN_IO1	TDO
/EVTI	/EVTI	IN	10	9	IN	/RESET	/RESET
GEN_IO7	GEN_IO7	IN/OUT	8	7	IN/OUT	GEN_IO6	GEN_IO6
CLKOUT	CLKOUT	IN	6	5	IN/OUT	GEN_IO8	GEN_IO8
USER_IO4	USER_IO4		4	3		USER_IO3	USER_IO3
USER_IO2	USER_IO2		2	1		USER_IO1	USER_IO1 ¹

1. Pins 1 to 4 are defined as User IO pins. Previously, these were considered reserved by logic analysers.

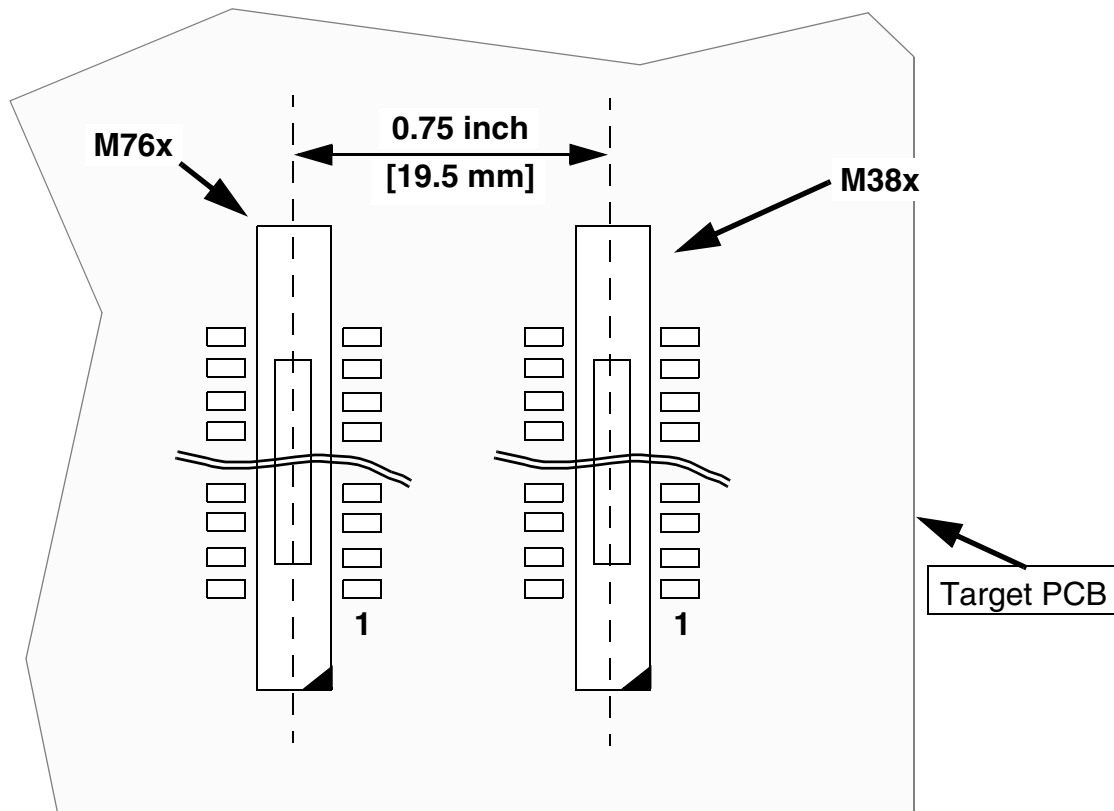
Table A-6—MICTOR Connector M38-2x (Second Half)

Combined M38-2C	Aux Only M38-2A					Aux Only M38-2A	Combined M38-2C
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	GEN_IO_4	GEN_IO4
PORT9	PORT9	IN/OUT	20	19	IN/OUT	GEN_IO5	GEN_IO5
PORT10	PORT10	IN/OUT	18	17	IN/OUT	GEN_IO6	GEN_IO6
PORT11	PORT11	IN/OUT	16	15	IN/OUT	GEN_IO7	GEN_IO7
PORT12	PORT12	IN/OUT	14	13	IN/OUT	GEN_IO9	GEN_IO9
PORT13	PORT13	IN/OUT	12	11	IN/OUT	GEN_IO8	GEN_IO8
PORT14	PORT14	IN/OUT	10	9	IN/OUT	GEN_IO7	GEN_IO7
PORT15	PORT15	IN/OUT	8	7	IN/OUT	GEN_IO6	GEN_IO6
-	-	-	6	5	IN/OUT	GEN_IO5	GEN_IO5
RSVD4	RSVD4		4	3		RSVD_3	RSVD3
RSVD2	RSVD2		2	1		RSVD1	RSVD1 ¹

1. Pins 1 to 4 should be considered reserved¹ by logic analysers.

Figure A-1 - Nexus M38-2x Connector Layout shows the recommended layout for the dual M76 connectors.

Figure A-1—Nexus M38-2x Connector Layout



The following figure shows the pin out of the Samtec connector when using a parallel Auxiliary port. The connector options are 2 by 10, 2 by 25, and 2 by 40 positions (20, 50, and 80 signals). For superior signal integrity, signals should use coaxial cables should be used, however, twin-ax cabling is not required. An Auxiliary Only is not defined at this time.

Table A-7—Samtec Connector HPxxx (ERx8) Legacy (Parallel Aux)

	Combined HPxxC					Combined HPxxC
1	MSEO0	OUT	1	2	OUT	VREF
2	MSEO1	OUT	3	4	IN	TCK(C)
3	GND	GND	5	6	IN	TMS(C)
4	MDO0	OUT	7	8	IN	TDI
5	MDO1	OUT	9	10	OUT	TDO
6	GND	GND	11	12	IN	/TRST
7	MDO2	OUT	13	14	OUT	/RDY
8	MDO3	OUT	15	16	IN	/EVTI
9	GND	GND	17	18	OUT	/EVTO
10	MCKO	OUT	19	20	IN	/RESET
11	MDO4	OUT	21	22	IN	GEN_IO0
12	GND	GND	23	24	GND	GND
13	MDO5	IN/OUT	25	26	IN/OUT	CLKOUT ¹
14	MDO6	OUT	27	28	IN/OUT	GEN_IO1
15	GND	GND	29	30	GND	GND
16	MDO7	IN/OUT	31	32	IN/OUT	GEN_IO2
17	MDO8	IN	33	34	IN/OUT	GEN_IO3
18	GND	GND	35	36	GND	GND
19	MDO9	OUT	37	38	IN/OUT	GEN_IO4
20	MDO10	OUT	39	40	IN/OUT	GEN_IO5
21	GND	GND	41	42	GND	GND
22	MDO11	OUT	43	44	OUT	MDO13
23	MDO12	OUT	45	46	OUT	MDO14
24	GND	GND	47	48	GND	GND
25	MDO15	OUT	49	50	OUT	MDO16 ²
26	MDO17	OUT	51	52	OUT	MDO18

Table A-7—Samtec Connector HPxxx (ERx8) Legacy (Parallel Aux)

	Combined HPxxC					Combined HPxxC
27	GND	GND	53	54	GND	GND
28	MDO19	OUT	55	56	OUT	MDO21
29	MDO20	OUT	57	58	OUT	MDO22
30	GND	GND	59	60	GND	GND
31	MDO23	OUT	61	62	OUT	MDO25
32	MDO24	OUT	63	64	OUT	MDO26
33	GND	GND	65	66	GND	GND
34	MDO27	OUT	67	68	OUT	MDO29
35	MDO28	OUT	69	70	OUT	MDO30
36	GND	GND	71	72	GND	GND
37	MDO31	OUT	73	74	OUT	
38		OUT	75	76	OUT	
39	GND	GND	77	78	GND	GND
40		OUT	79	80	OUT	

1. RTCK on ARM based systems

2. MDO16 would not be used on a standard 2 by 25 connector with 16 MDO signals.

The following figure shows the pinout of the Samtec connector when used for High Speed Serial Auxiliary port, with either a Simplex High Speed Serial Auxiliary with JTAG or with a full Duplex High Speed Serial Auxiliary ports. This connector definition supports trace output of up to 8 lanes and has the option for up to 4 lanes of High Speed input. The connector options are 2 by 11, 2 by 17, 2 by 23, and 2 by 35 positions (22, 34, 48, and 70 signals). For best signal integrity, twin-ax cable should be used on the odd side of the connector. The even side of the connector should use coax cable.

NOTE

It is recommended that signal pins on the mating connector (to the target system connector) be implemented with shorter traces than the ground pins of the

connector to allow grounds to connect prior to any signal pins.

Table A-8—Samtec Connector ERx8 High Speed Serial

	High Speed Simplex	High Speed Duplex			High Speed Duplex	High Speed Simplex
1	TX0+	TX0+	1	2	VREF	VREF
2	TX0-	TX0-	3	4	TCK	TCK
3	GND	GND	5	6	TMS/TMSC	TMS/TMSC
4	TX1+	TX1+	7	8	TDI	TDI
5	TX1-	TX1-	9	10	TDO	TDO
6	GND	GND	11	12	/TRST	/TRST
7	TX2+	RX0+	13	14	GEN_IO0	GEN_IO0
8	TX2-	RX0-	15	16	/EVTI	/EVTI
9	GND	GND	17	18	/EVTO	/EVTO
10	TX3+	RX1+	19	20	GEN_IO3	GEN_IO3/ CLKOUT ¹
11	TX3-	RX1-	21	22	/RESET	/RESET
12	GND	GND	23	24	GND	GND
13	TX4+	TX2+	25	26	CLK+	CLK+
14	TX4-	TX2-	27	28	CLK-	CLK-
15	GND	GND	29	30	GND	GND
16	TX5+	TX3+	31	32	/RDY	/RDY
17	TX5-	TX3-	33	34	GEN_IO5	GEN_IO5
18	GND	GND	35	36	GND	GND
19	TX6+	RX2+	37	38	RESERVED	RESERVED
20	TX6-	RX2-	39	40	RESERVED	RESERVED
21	GND	GND	41	42	GND	GND
22	TX7+	RX3+	43	44	RESERVED	RESERVED
23	TX7-	RX3-	45	46	RESERVED	RESERVED
24		GND	47	48	GND	
25		TX4+	49	50	RESERVED	
26		TX4-	51	52	RESERVED	

Table A-8—Samtec Connector ERx8 High Speed Serial

	High Speed Simplex	High Speed Duplex			High Speed Duplex	High Speed Simplex
27		GND	53	54	GND	
28		TX5+	55	56	RESERVED	
29		TX5-	57	58	RESERVED	
30		GND	59	60	GND	
31		TX6+	61	62	RESERVED	
32		TX6-	63	64	RESERVED	
33		GND	65	66	GND	
34		TX7+	67	68	RESERVED	
35		TX7-	69	70	RESERVED	

1. Optionally CLKOUT or RTCK on ARM implementations if required.

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 5 - Nexus Message Protocol** 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, $\overline{\text{TRST}}$

Refer to **Section 6.3 - IEEE 1149.1 (JTAG) Interface** for a description of the following signals:

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

Refer to **SECTION Section 6.4 - High Speed Serial Interface (Aurora)** for a description of the following signals:

TXx+, TXx-, RXx+, RXx-, CLK+, CLK-

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 is an optional signal but 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 Auxiliary port.

A.1.1.2 CLK+, CLK-

The Aurora based Serial High Speed Trace port requires a high quality clock and Phase Lock Loop to support a given bit rate. Depending on the design of the embedded device, this clock can either be generated on the device or it can be generated by the tool. If generated by the tool, a differential clock signal can be supplied to the device.

A.1.1.3 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.1.1.4 General Purpose IO Signals

Previous versions of the specification defined both Vendor-defined signals and Tool-defined Signals. These have been replaced with General Purpose IO signals. They are defined for use by particular needs/requirements of specific SOCs. Tool vendors should design their generic 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. GEN_IO pins can also be used as Time Stamp pins if defined for the SOC.

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 VSTBY

VSTBY 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. (This signal was named VALTREF in the 5001-2003 revision of the standard.)

A.1.1.8 UBATT

UBATT pins are vendor-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. If implemented, the connection should be reverse voltage protected. See **Table A-9**.

Table A-9—Recommended UBATT Specifications

Specification	Minimum	Maximum	Units
Voltage Range	5	20	V
Maximum Current	—	300	mA
Maximum In-rush Current ¹		1.0	A

1. Maximum duration of 3 mS.

A.1.2 Nexus Combined Implementation Considerations

It is recommended that the signals in **Table A-10** be connected to pull-ups on the target to prevent floating signals when the tool is not connected or powered on. $\overline{\text{TRST}}$ 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-10—Signals for Pull-Ups on the Target

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

It is recommended that the signals in **Table A-11** be connected to pull-ups on the tool to prevent floating signals when the target is not connected or powered on.

Table A-11—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-12** be connected to pull on the target to prevent floating signals when the tool is not connected or powered on. RSTI should have a pull-down.

Table A-12—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-13** 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 tool is not connected.

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

Signals	Pull-up
MDO[15:0], 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-14 lists the electrical characteristics for the signals used in the Nexus interface.

Table A-14—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 $V_{REF} + 20\%$.

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

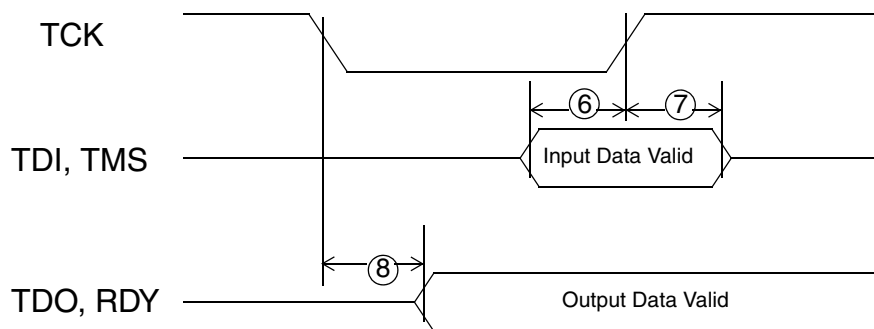
Table A-15 lists the timing constraints for the **IEEE 1149.1** interface.

Figure A-2 gives a pictorial representation of critical timing in **Table A-15**.

Table A-15—Timing Constraints for the IEEE 1149.1 Interface

Number	Characteristic	Up to 33 MHz		Up to 50 MHz		Up to 100 MHz		Unit
		Min	Max	Min	Max	Min	Max	
1	TCK Cycle Time (T_c)	30	—	20	—	10	—	ns
2	TCK Duty Cycle	40	60	45	55	45	55	%
3	Rise and Fall Times (20%–80%)	0	3	0	1.5	0	1.5	ns
4	$\overline{\text{TRST}}$ Setup Time to TCK Falling Edge	$(0.30)T_c$	—	$(0.15)T_c$	—	$(0.15)T_c$	—	ns
5	$\overline{\text{TRST}}$ Assert Time	$(0.30)T_c$	—	$2T_c$	—	$2T_c$	—	ns
6	TMS, TDI Data Setup Time	$(0.20)T_c$	—	$(0.15)T_c$	—	$(0.15)T_c$	—	ns
7	TMS, TDI Data Hold Time	$(0.10)T_c$	—	$(0.15)T_c$	—	$(0.15)T_c$	—	ns
8a	TCK Low to TDO Data Valid (Easy Timing)	$(-0.10)T_c$	$(0.20)T_c$	$(-0.10)T_c$	$(0.20)T_c$	$(-0.10)T_c$	$(0.20)T_c$	ns
8b	TCK Low to TDO Data Valid High speed support	—	T_c-4	—	T_c-4	—	T_c-4	ns
8c	TCK Low to TDO hold time (high speed support)	1	—	1	—	1	—	ns
9	$\overline{\text{EVTO}}$ Pulse Width ¹	5	—	5	—	5	—	ns
10	$\overline{\text{EVTI}}$ Pulse Width	5	—	4	—	5	—	ns

1. $\overline{\text{EVTO}}$ and $\overline{\text{EVTI}}$ were previously specified as a minimum of 4 MCKO periods.

**Figure A-2—IEEE 1149.7 Timing Diagram**

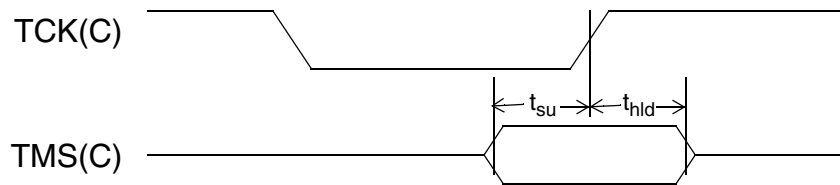
A.5 AC Electrical Characteristics - IEEE 1149.7 Interface

The IEEE 1149.7 allows for rising- and falling-edge timing when using the Advanced protocol and rising-edge timing when using the Standard Protocol. **Table A-16** lists the timing constraints for the **IEEE 1149.7** interface.

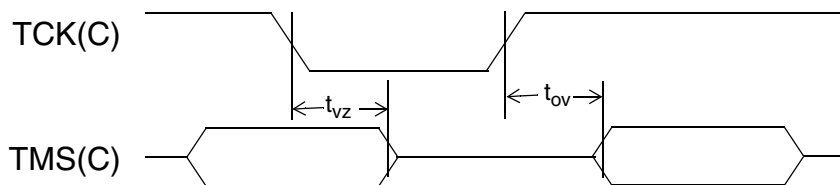
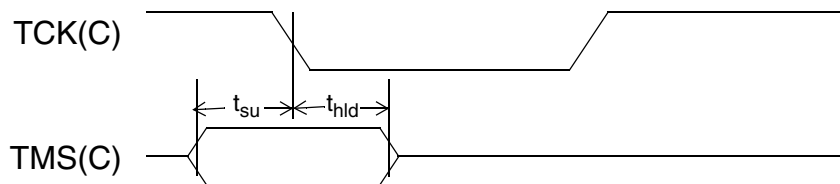
Table A-16—Timing Constraints for the IEEE 1149.7 Interface

Number	Characteristic		Min	Max	Unit
1	Minimum Clock Period	T_c	-	-	ns
2	Setup time before Clock TCK(C)	t_{su}	$0.20(T_c)$	-	
3	Hold time after Clock	t_{hld}	0	-	ns
4	Time from clock rising valid high impedance TCK(C)	t_{vz}	$0.30(T_c)$	-	
5	Time from clock falling to output valid TCK(C)	t_{ov}	$0.20(T_c)$	-	ns

Rising Edge Timing



Falling Edge Timing



A.6 AC Electrical Characteristics - Parallel Auxiliary Port

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 Parallel AUX Interface

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

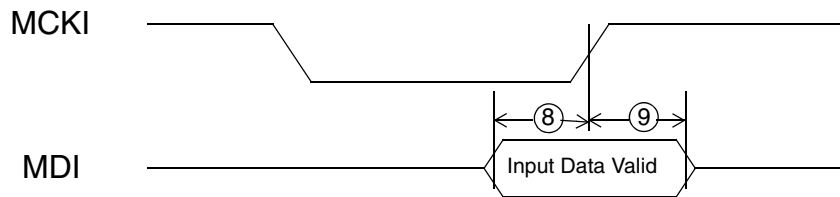


Figure A-3—Parallel Auxiliary Port Data Input Timing Diagram

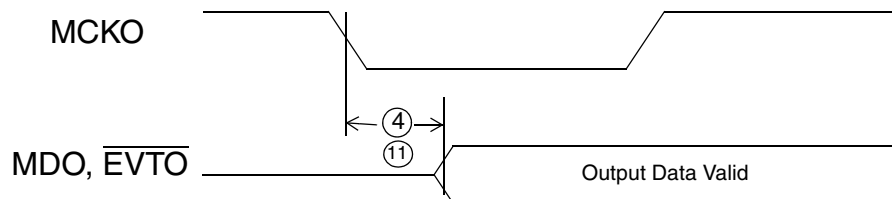


Figure A-4—Parallel Auxiliary Port Data Output Timing Diagram

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

When $\overline{\text{RSTI}}$ is asserted, $\overline{\text{EVTI}}$ 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.

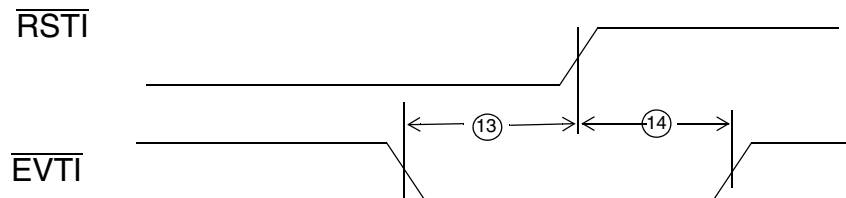


Figure A-5—Enable Parallel Auxiliary from $\overline{\text{RSTI}}$

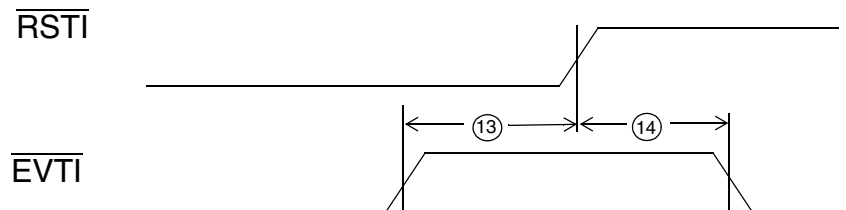


Figure A-6—Disable Parallel Auxiliary from $\overline{\text{RSTI}}$

A.7 DC and AC Electrical Characteristics - High Speed Serial Auxiliary Port

The electrical specifications for the Aurora High Speed Serial Auxiliary port are taken from the Xilinx Aurora Electrical Specifications. The Aurora channel speed is not limited to the values listed in this specification.

A.7.1 Transmitter Timing

Table A-18—High Speed Serial Auxiliary Interface Transmitter AC Timing Constraints for the 1.25 Gbps baud rate¹

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Output Voltage	V_{DIFF}	800	1600	mV	Peak-to-peak differential
2	Rise/Fall Time	TM_{RF}	60		ps	At driver output
3	Deterministic jitter	J_D		0.17	UI	
4	Total jitter	J_T		0.35	UI	
5	Output skew	S_O		25	ps	Skew at transmitter output between the differential pair
6	Multiple Output Skew	S_{MO}		1000	ps	Skew at transmitter output between lanes of a multi-lane channel
7	Unit Interval	UI	800	800	ps	+/- 100 ppm

1. AC coupling required to guarantee inter-operability between vendors.

Table A-19—High Speed Serial Auxiliary Interface Transmitter AC Timing Constraints for the 2.5 Gbps baud rate¹

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Output Voltage	V_{DIFF}	800	1600	mV	Peak-to-peak differential
2	Rise/Fall Time	TM_{RF}	40		ps	At driver output
3	Deterministic jitter	J_D		0.17	UI	
4	Total jitter	J_T		0.35	UI	
5	Output skew	S_O		20	ps	Skew at transmitter output between the differential pair
6	Multiple Output Skew	S_{MO}		1000	ps	Skew at transmitter output between lanes of a multi-lane channel
7	Unit Interval	UI	400	400	ps	+/- 100 ppm

1. AC coupling required to guarantee inter-operability between vendors.

Table A-20—High Speed Serial Auxiliary Interface Transmitter AC Timing Constraints for the 3.125 Gbps baud rate¹

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Output Voltage	V_{DIFF}	800	1600	mV	Peak-to-peak differential
2	Rise/Fall Time	TM_{RF}	40		ps	At driver output
3	Deterministic jitter	J_D		0.17	UI	
4	Total jitter	J_T		0.35	UI	
5	Output skew	S_O		15	ps	Skew at transmitter output between the differential pair
6	Multiple Output Skew	S_{MO}		1000	ps	Skew at transmitter output between lanes of a multi-lane channel
7	Unit Interval	UI	320	320	ps	+/- 100 ppm

1. AC coupling required to guarantee inter-operability between vendors.

A.7.2 Receiver Timing

Table A-21—High Speed Serial Auxiliary Interface Transmitter AC Timing Constraints for the 1.25 Gbps baud rate¹

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Input Voltage	V_{IN}	200	1600	mV	Peak-to-peak differential input voltage
2	Deterministic jitter	J_D		0.37	UI	
3	Total jitter	J_T		0.67	UI	
4	Input skew	S_I		75	ps	Skew at receiver input between signal of the differential pair
5	Multiple Output Skew	S_{MI}		24	ps	Skew at receiver input between lanes of a multi-lane channel
6	Bit Error Rate	BER		10^{-12}		
7	Unit Interval	UI	800	800	ps	+/- 100 ppm

1. AC coupling required to guarantee inter-operability between vendors.

Table A-22—High Speed Serial Auxiliary Interface Transmitter AC Timing Constraints for the 2.5 Gbps baud rate¹

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Input Voltage	V_{IN}	200	1600	mv	Peak-to-peak differential input voltage
2	Deterministic jitter	J_D		0.37	UI	
3	Total jitter	J_T		0.65	UI	
4	Input skew	S_I		75	ps	Skew at receiver input between signal of the differential pair
5	Multiple Output Skew	S_{MI}		24	ps	Skew at receiver input between lanes of a multi-lane channel
6	Bit Error Rate	BER		10^{-12}		
7	Unit Interval	UI	400	400	ps	+/- 100 ppm

1. AC coupling required to guarantee inter-operability between vendors.

Table A-23—High Speed Serial Auxiliary Interface Transmitter AC Timing Constraints for the 1.25 Gbps baud rate¹

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Input Voltage	V_{IN}	200	1600	mv	Peak-to-peak differential input voltage
2	Deterministic jitter	J_D		0.37	UI	
3	Total jitter	J_T		0.65	UI	
4	Input skew	S_I		75	ps	Skew at receiver input between signal of the differential pair
5	Multiple Output Skew	S_{MI}		24	ps	Skew at receiver input between lanes of a multi-lane channel
6	Bit Error Rate	BER		10^{-12}		
7	Unit Interval	UI	320	320	ps	+/- 100 ppm

1. AC coupling required to guarantee inter-operability between vendors.

A.7.3 Receiver Eye Patterns

Figure A-7 shows the receiver eye openings for proper operation.

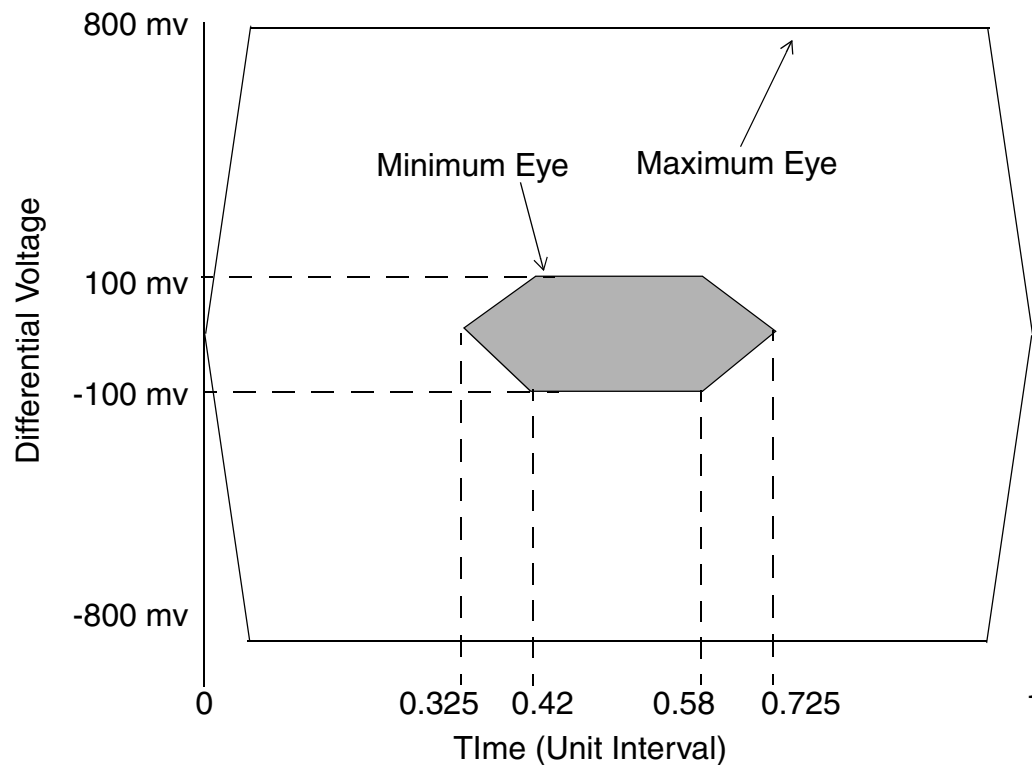


Figure A-7—Receiver Eye Pattern

A.7.4 CLK+, CLK- Specifications

Table A-24—High Speed Serial Auxiliary Interface CLK+, CLK- DC Characteristics

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1	Differential Input Voltage External AC coupled < 3.2 Gb/s	V_{CLK}	150	2000	mV	Peak-to-peak differential input voltage
2	Differential Input Voltage External AC coupled < 3.2 Gb/s	V_{CLK}	180	2000	mV	
3	Recommended external AC coupling capacitor		75	200	nF	100 nF typical
4	Differential output resistance		90	120	ohm	100 ohm typical

Table A-25—High Speed Serial Auxiliary Interface CLK+, CLK- AC Timing

Number	Characteristic	Symbol	Range		Unit	Notes
			Min	Max		
1a	Reference Clock Frequency	CLKHP	50	350	MHz	High performance ¹
1b	Reference Clock Frequency	CLKLC	625	1250	MHz	Low Cost ¹
2	Reference clock rise time			400	ps	20%-80%, 200 ps typical
3	Reference clock fall time			400	ps	80%-20%, 200 ps typical
4	Reference clock duty cycle		45	55	%	
5	Reference Clock total jitter, peak-peak			40	ps	
6	Stability		50		ppm	

1. High performance systems use an on-chip PLL to generate the transmit clock from a lower input frequency. Low cost systems require an external physical interface clock reference.

A.8 Terminations

Because of the high-speed natures of the **IEEE 1149.1** and Auxiliary ports, it is recommended that the target and tool both employ a point-to-point series termination scheme (see **Figure A-8** and **Figure A-9**).

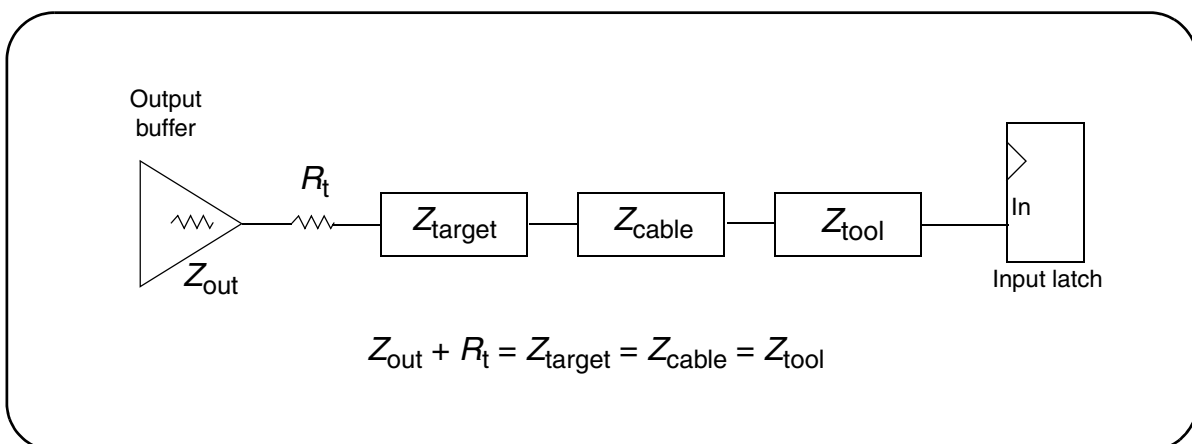
Z_{out} = Output impedance of the driver

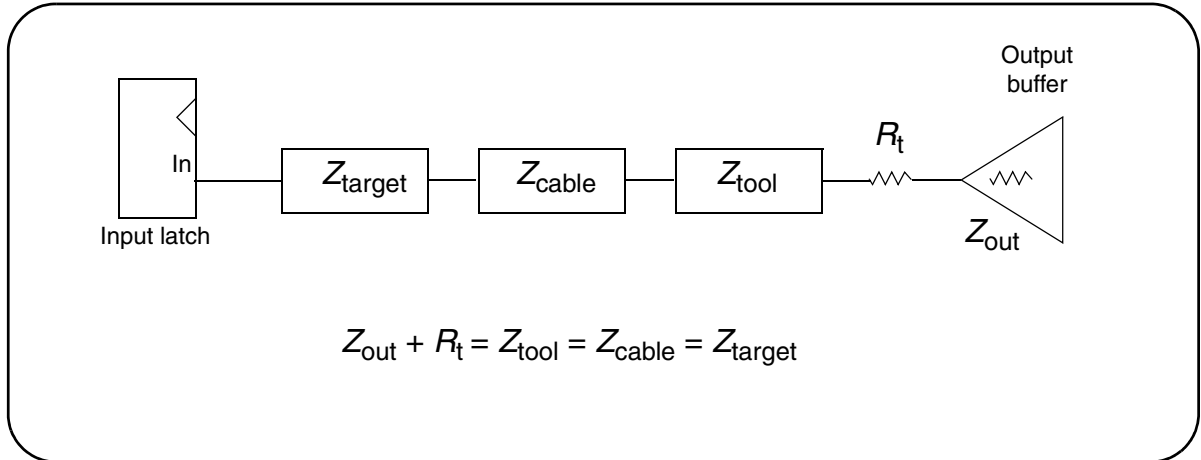
Z_{target} = Impedance of traces on the target printed circuit board

Z_{cable} = Characteristic impedance of cable

Z_{tool} = Impedance of traces on the tool printed circuit board

R_t = Source terminators

**Figure A-8—Target Output Source Termination**

**Figure A-9—Tool Output Source Termination**

APPENDIX B

Recommendations for Access to Control and Status Registers

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 4 - Nexus Public Messages**. There are no requirements in the Nexus standard, however, regarding conformity of development registers that are accessed by tools for control and status.

NOTE

This section has been updated since the 5001-2003 revision. The concatenated register support have been removed. Some register index values have been changed and some bits have been redefined to support additional features of the 5001-2012 standard.

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 multiple clients⁷ 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 TAP 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 parallel AUX or the TAP interface for legacy implementations, or via the Aurora high speed serial link for higher performance implementations.

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**.

⁷Refer to **1.3 - Terms and Definitions**.

Additionally, the Device ID information identifies key attributes to the development tool concerning the embedded processor.

In **Table B-2** the Client Select control 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.

Control/Status	Compliance Class	Access Index	Read/Write
TAP Public Opcodes	—	Vendor-defined	—
DID Register ¹	All	Vendor-defined	R
NEXUS-ACCESS ²	All	Vendor-defined	R/W
NEXUS-STATUS ³	All	Vendor-defined	R

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

1. The DID Register is defined by **IEEE Std 1149.1-1990**.
2. Only needed for **TAP** port (and not AUX). See **6.3 - IEEE 1149.1 (JTAG) Interface**.
3. Optional instruction for TAP port (not AUX). See **6.3.6.2 - NRR Access Status (Optional)**.

Control/Status	Compliance Class	Register Index	Read/Write
Device ID	All	0x0	R
Client Select	4 ¹	0x1	R/W
Shared by all Nexus Clients	—	0x2 – 0x3F	—
Vendor-defined	—	0x40 – 0x7F	—
Reserved	—	0x80 – 0xFF	—

Table B-2—Summary of Nexus Client Registers

1. If an embedded processor contains multiple clients, then Client Select is required (for AUX-only implementations).

The NRR indices as shown in **Table B-3** shall be identical for accesses via the TAP interface and the AUX. The fields associated with each opcode accessed via the TAP interface shall be identical in size and function to the packets accessed for each opcode via the AUX. The Public Messages in **SECTION 4 - 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 comprise the transfer registers for interfacing with a processor, e.g., Program Counter and Processor Status.

NRR	Compliance Class	Register Index	Read/Write
Device ID (DID) (auxiliary only)	All	0x0	R
Client Select Control (CSC)	2, 3, 4 ¹	0x1	R/W
Development Control (DC1 - DC4)	2, 3, 4	0x2 - 0x5	R/W
User Base Address (UBA)	2, 3, 4	0x6	R ²
Read/Write Access Control/Status (RWCS)	3, 4	0x7	R/W
Reserved for Read/Write Access Control/Status	—	0x8	—
Read/Write Access Address (RWA)	3, 4	0x9	R/W
Read/Write Access Data (RWD)	3, 4	0xA	R/W
Watchpoint Trigger (WT)	4	0xB	R/W
Reserved for Watchpoint Trigger	—	0xC	—
Data Trace Control (DTC)	3, 4	0xD	R/W
Data Trace Start Address (DTSA1-DTSA2)	3, 4	0xE – 0xF	—
Data Trace Start Address (DTSA1-DTSA2)	3,4	0x10 – 0x11	—
Data Trace End Address (DTEA1-DTEA2)	3, 4	0x12 – 0x13	—
Data Trace End Address (DTEA1-DTEA2)	—	0x14 – 0x15	—
Breakpoint/Watchpoint Control (BWC) (2)	4	0x16 – 0x17	R/W
Breakpoint/Watchpoint Control (Reserved - 6)	—	0x18 – 0x1D	—
Breakpoint/Watchpoint Address (BWA) (2)	4	0x1E – 0x1F	R/W
Breakpoint/Watchpoint Address (Reserved - 6)	—	0x20 – 0x25	—
Breakpoint/Watchpoint Data (BWD) (2)	4	0x26 – 0x27	R/W
Breakpoint/Watchpoint Data (Reserved - 6)	—	0x28 – 0x2D	—
Input Public Message Register (IPMR)	2,3,4	0x2E	R/W
Output Public Message Register (OPMR)	2,3,4	0x2F	R/W
Development Status (DS)	4	0x30	R
Overrun Control Register (OVCR)	2,3,4	0x31	R/W
Watchpoint Mask Register (WMSK)	2,3,4	0x32	R/W
Aurora Trace Transmission Control Register (ATCR)	2, 3, 4	0x33	R/W
Aurora Trace Transmission Status Register (ATSR)	2, 3, 4	0x34	R
Aurora Link Control Register (ALC)	2, 3, 4	0x35	R/W
Aurora Link Training Control Register (ATC)	2, 3, 4	0x36	R/W
Aurora Link Status Register (ALS)	2, 3, 4	0x37	R

Table B-3—Nexus Recommended Registers (NRRs)

NRR	Compliance Class	Register Index	Read/Write
Aurora Link Error Control Register (ALE)	2, 3, 4	0x38	R/W
Reserved for future expansion	—	0x39 – 0x3F	—
Vendor defined	—	0x40 – 0x7F	—
Reserved for future Nexus functionality ³	—	0x80 – 0xFF	—

Table B-3—Nexus Recommended Registers (NRRs)

1. Needed if there are multiple clients on an embedded processor.
2. May also be read/write access for development tool configuration of UBA.
3. **TAP** is not capable of accessing Future Reserved (0x80 - 0xFF).

B.2 Reset

When an external development tool is connected to the target, all control and status information shall be reset by one of the following:

- **TAP “TEST-LOGIC-RESET” state or assertion of \overline{TRST} pin**
- **Assertion of \overline{RSTI} (auxiliary only)**

In the external tool mode, no control or status information shall be reset for system reset on the embedded processor.

For implementations which enable software controlled debug resources, system reset will be used to reset control and status information.

B.3 Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)

The TAP state machine is shown in **Figure 6-4**. The value shown adjacent to each arc represents the value of the TMS signal sampled on the rising edge of the TCK signal.

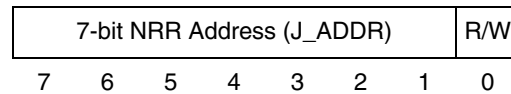
One recommended method for accessing NRRs is enabled by loading a single instruction (“NEXUS-ACCESS”) into the TAP IR. Once the TAP “NEXUS-ACCESS” instruction has been loaded, the TAP port allows tool-target communications with all NRRs according to the register index in **Table B-3**.

Using this mechanism, reading/writing of an NRR then requires two passes through the Data-Scan path of the TAP state machine.

1. The first pass through the Data Register (DR) selects the NRR to be accessed by by loading an 8-bit value into the **TAP DR**: 7-bits for index (see **Table B-3**) and 1-bit for direction (read or write). This register has the following format:

MSB LSB

Figure B-1—TAP DR for NRR Access

**Figure B-1—TAP DR for NRR Access**

Bit Number	Field Name	Class				Description
		1	2	3	4	
7–1	J_ADDR	Y	Y	Y	Y	<u>NRR Address:</u> Selected from values in Table B-3
0	R/W	Y	Y	Y	Y	<u>Read / Write:</u> 0 = Read 1 = Write

Table B-4—TAP DR Fields (for NRR Access)

2. The second pass through the DR then shifts the data in or out of the **TAP** port, LSB first.
 - a. During a read access, data are latched from the selected NRR when the **TAP** state machine passes through the CAPTURE-DR state.
 - b. During a write access, data are latched into the selected NRR when the **TAP** state machine passes through the UPDATE-DR state.

B.4 Access via the Auxiliary Port (parallel) or Aurora (high speed serial)

The control and status registers are accessed via three Public Messages:

- *Auxiliary Access - Read (tool requests information)*
- *Auxiliary Access - Write (tool provides information)*
- *Auxiliary Access - Response (from tool or target)*

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

1. The tool transmits a *Auxiliary Access - Write Message*, which contains write attributes (including the register index) and data to be written to the register.
2. The tool waits for the *Auxiliary Access - Response Message* before initiating the next access. This message indicates the status of the transfer.

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

1. The tool transmits a *Auxiliary Access - Read Message*, which contains read attributes (including the register index).
2. Once the target has accessed the register internally, it transmits a *Auxiliary Access - Response Message* containing the status of the read transfer and the

register data (if the transfer was successful). The target is now ready for the next access.

NOTE

The register access mechanism is the same for high speed serial accesses over Aurora as well as for a parallel AUX port.

B.5 Control and Status Registers

This section describes the fields comprising each control and status register. 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. Three registers are used for control and status:

- Device ID (DID) Register
- Client Select Control (CSC) Register
- Development Control (DC1) Register 1
- Development Control (DC2) Register 2
- Development Control (DC3) Register 3
- Development Control (DC4) Register 4
- Development Status (DS) Register

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 auxiliary output port upon exit of AUX reset.

For embedded processors with a full AUX, the DID Register shown in **Table B-5** 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 TAP 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-5**).

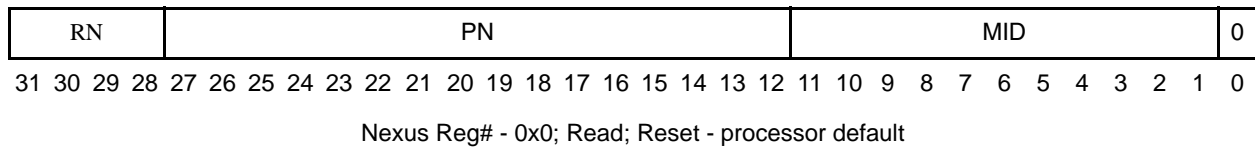


Figure B-2—Device ID Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
31–28	RN	Y	Y	Y	Y	Revision Number
27–12	PN	Y	Y	Y	Y	Product Number
11–1	MID	Y	Y	Y	Y	Manufacturer ID
0	—	--	--	--	--	Reserved

Table B-5—DID Register Fields

B.5.2 Client Select Control (CSC) Register

The CSC Register contains a single 8-bit field that, when written to, selects the client to be accessed via the TAP 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 indices 0x1–0x3F.

The CSC Register is recommended if there are multiple clients on the embedded processor (see **Table B-6**).

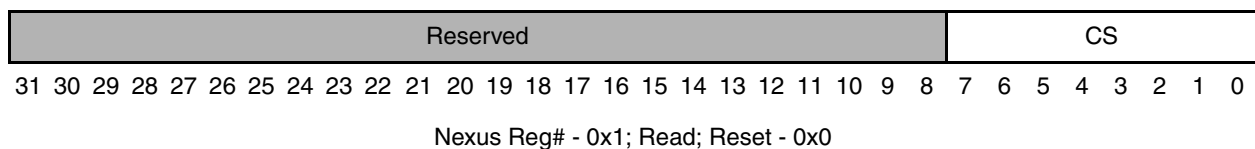


Figure B-3—Client Select Control Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
31-8	--	--	--	--	--	Reserved
7–0	CS	--	Y	Y	Y	Client select

Table B-6—CSC Register Fields

B.5.3 Development Control (DC1) Register 1

The DC1 Register is used for basic development control of a client. Basic run-control (static) debug features are defined in the register (Class4) as well as basic trace features (Class2 - Class4).

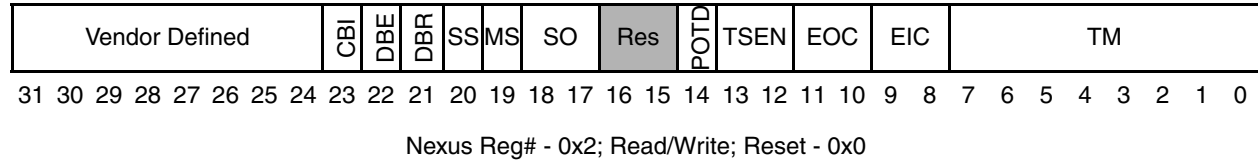


Figure B-4—Development Control Register 1

Bit Number	Field Name	Class				Description
		1	2	3	4	
31–24	—	--	--	--	--	Vendor-defined
23	CBI	Y	Y	Y	Y	<u>CBI - Client Breakpoint Input (optional)</u> 0 = Break for internal breakpoints only 1 = Break for other clients' breakpoints also
22	DBE	--	--	--		<u>DBE - Debug Enable</u> 0 = Debug mode disabled 1 = Debug mode enabled
21	DBR	--	--	--		<u>DBR - Debug Request</u> 0 = Exit debug mode 1 = Request debug mode
20	SS	--	--	--		<u>SS - Step Enable</u> 0 = Single-step disabled 1 = Single-step enabled
19	MS	--	--	--	?	<u>MS - Memory Substitution</u> 0 = Use instructions and data in target memory 1 = Access instructions/data through AUX
18-17	SO	--	--	--	?	<u>SO - Substitution Operands</u> 00 = Instructions and data 01 = Instructions only 10 = Data only 11 = Reserved
16-15	---	--	--	--	--	Reserved
14	POTD	--	?	?	?	<u>POTD - Periodic Ownership Trace Disable (optional)</u> 0 = Periodic OTM enabled 1 = Periodic OTM disabled
13-12	TSEN	--	?	?	?	<u>TSEN - Timestamp Enable (optional)</u> 0x = Timestamp Disabled 10 = Timestamp Enabled for all messages 11 = Vendor-defined

Table B-7—Development Control Register 1 Fields

Bit Number	Field Name	Class				Description
		1	2	3	4	
11-10	EOC	--	Y	Y	Y	<u>EOC - Event Out Control (optional)</u> 00 = $\overline{\text{EVTO}}$ asserted upon occurrence of watchpoint(s) 01 = $\overline{\text{EVTO}}$ asserted upon entry into debug mode 10 = $\overline{\text{EVTO}}$ asserted upon timestamp event 11 = Vendor-defined
9-8	EIC	--	Y	Y	Y	<u>EIC - Event In Control (optional)</u> ¹ 00 = $\overline{\text{EVTI}}$ is used for synchronization 01 = $\overline{\text{EVTI}}$ asserts debug request to processor 10 = $\overline{\text{EVTI}}$ is disabled for this client 11 = Vendor-defined
7-0	TM	--	Y	Y	Y	<u>TM - Trace Mode</u> 00000000 = Trace Disabled xxxxxxx1 = Ownership Trace Enabled xxxxxx1x = Data Trace Enabled xxxxx1xx = Program Trace Enabled xxxx1xxx = Watchpoint Trace Enabled xxx1xxxx = In-Circuit Trace Enabled xx1xxxxx = Data Acquisition Enabled x1xxxxxx = Vendor Defined Trace #1 Enabled 1xxxxxxx = Vendor Defined Trace #2 Enabled

Table B-7—Development Control Register 1 Fields

1. A high to low transition on $\overline{\text{EVTI}}$ will cause a program and/or data trace synchronizing.

Class 2, 3, or 4:

The trace mode (TM) field enables all trace features (including vendor-defined trace). One or all types of trace may be enabled by TM or via a watchpoint occurrence (refer to -).

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-7**).

Class 4 Only:

For Class 4 devices, 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 memory substitution (MS) and step enable (SS) bit fields determine how the processor will operate when DBR is negated. If $\text{MS} = \text{SS} = 0$, then normal operation will commence when DBR is negated. If $\text{MS} = 0$ and $\text{SS} = 1$, then a single step will occur when DBR is negated with internal memory access. If $\text{MS} = 1$ and $\text{SS} = 0$, then operation will commence when DBR is negated with instruction/data access via the AUX. If $\text{MS} = \text{SS} = 1$, 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 TAP interface. If $MS = 0$, memory substitution is not enabled, and memory accesses are made to the target memory system.

Optional:

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.

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

Disabling periodic ownership trace message generation can be achieved by setting the POTD bit.

The timestamp enable field (TSEN) is a two bit field. If TSEN = 10, timestamping is enabled on all implemented trace features. Setting TSEN = 11 allows for vendor-defined functionality to be added to the timestamp enable feature.

If the \overline{EVTO} control (EOC) field = 00, the \overline{EVTO} pin is asserted upon the occurrence of a watchpoint - see **Section B.5.4 - Development Control (DC2) Register 2** for programming watchpoint(s) on \overline{EVTO} . If EOC = 01, entry into debug mode will assert \overline{EVTO} . If EOC = 10, \overline{EVTO} will be asserted upon a timestamp event for targets which implement timestamping via pins - see **Section 3.12.3 - Timestamping via pins** for more information on timestamp options. EOC = 11 provides optional vendor-defined functionality.

B.5.4 Development Control (DC2) Register 2

The Development Control Register 2 is used to control which watchpoint (client defined) will assert \overline{EVTO} (if $DC1[EOC] = 00$). The EWC field supports up to 16 watchpoint sources. Additional watchpoints and/or vendor-defined \overline{EVTO} control can be defined with the additional 16 bits in DC2.

Vendor Defined																EWC															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure B-5—Development Control Register 2

Nexus Reg# - 0x3; Read/Write; Reset - 0x0

Figure B-5—Development Control Register 2

Bit Number	Field Name	Class				Description
		1	2	3	4	
31-16	---	--	--	--	--	Vendor-defined
15-0	EWC	--	Y	Y	Y	<u>EWC - Event Out Watchpoint Control</u> 0000000000000000 = No Watchpoints trigger <u>EVTO</u> xxxxxxxxxxxxxxxx1 = Watchpoint #1 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1x = Watchpoint #2 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1xx = Watchpoint #3 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1xxx = Watchpoint #4 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1xxxx = Watchpoint #5 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1xxxxx = Watchpoint #6 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1xxxxxx = Watchpoint #7 triggers <u>EVTO</u> xxxxxxxxxxxxxxxx1xxxxxxx = Watchpoint #8 triggers <u>EVTO</u> xxxxxxxx1xxxxxxxxx = Watchpoint #9 triggers <u>EVTO</u> xxxxxx1xxxxxxxxxx = Watchpoint #10 triggers <u>EVTO</u> xxxxx1xxxxxxxxxxx = Watchpoint #11 triggers <u>EVTO</u> xxxx1xxxxxxxxxxxx = Watchpoint #12 triggers <u>EVTO</u> xxx1xxxxxxxxxxxxx = Watchpoint #13 triggers <u>EVTO</u> xx1xxxxxxxxxxxxxx = Watchpoint #14 triggers <u>EVTO</u> x1xxxxxxxxxxxxxxx = Watchpoint #15 triggers <u>EVTO</u> 1xxxxxxxxxxxxxxx = Watchpoint #16 triggers <u>EVTO</u>

Table B-8—Development Control Register 2 Fields**B.5.5 Development Control (DC3) Register 3**

The Development Control Register 3 is reserved for Vendor-defined functionality.

Vendor Defined																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Nexus Reg# - 0x4; Read/Write; Reset - 0x0

Figure B-6—Development Control Register 3

Bit Number	Field Name	Class				Description
		1	2	3	4	
31-0	---	--	--	--	--	Vendor-defined

Table B-9—Development Control Register 3 Fields**B.5.6 Development Control (DC4) Register 4**

The Development Control Register 4 defines mask bits for the EVCODEs supported in Program Correlation Messages (PTCM). See **Section 4.3.16 - Program Trace -**

Correlation Message for more information on these encodings. Additional vendor-defined control can be defined using the upper 8-bits of DC4. The remaining bits are reserved for future functionality.

Vendor Defined								Reserved								EVCDM															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Nexus Reg# - 0x5; Read/Write; Reset - 0x0

Figure B-7—Development Control Register 4

Bit Number	Field Name	Class				Description
		1	2	3	4	
31-24	--	--	--	--	--	Vendor-defined
23-16	---	--	--	--	--	Reserved
15-0	EVCDM	--	? Y	Y	Y	<u>Event Code (EVCODE) Mask</u> 0000000000000000 = No EVCODEs masked for PTCM XXXXXXXXXXXXXXXX1 = EVCODE #1 masked for PTCM XXXXXXXXXXXXXXXX1X = EVCODE #2 masked for PTCM XXXXXXXXXXXXXXXX1XX = EVCODE #3 masked for PTCM XXXXXXXXXXXXXXXX1XXX = EVCODE #4 masked for PTCM XXXXXXXXXXXXXXXX1XXXX = EVCODE #5 masked for PTCM XXXXXXXXXXXXXXXX1XXXXX = EVCODE #6 masked for PTCM XXXXXXXXXXXXXXXX1XXXXXX = EVCODE #7 masked for PTCM XXXXXXXXXXXXXXXX1XXXXXX = EVCODE #8 masked for PTCM XXXXXXXX1XXXXXXXXXX = EVCODE #9 masked for PTCM XXXXXX1XXXXXXXXXXXX = EVCODE #10 masked for PTCM XXXXX1XXXXXXXXXXXXX = EVCODE #11 masked for PTCM XXXX1XXXXXXXXXXXXXX = EVCODE #12 masked for PTCM XXX1XXXXXXXXXXXXXXX = EVCODE #13 masked for PTCM XX1XXXXXXXXXXXXXXX = EVCODE #14 masked for PTCM X1XXXXXXXXXXXXXXX = EVCODE #15 masked for PTCM 1XXXXXXXXXXXXXXX = EVCODE #16 masked for PTCM

Table B-10—Development Control Register 4 Fields

B.5.7 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 utilized for OTM and DQM and reserved for other uses. The size of the UBA Register is vendor defined (see **Table B-11**). However, the UBA is not required to support OTM in cases where OTM is controlled by a CPU core

register.

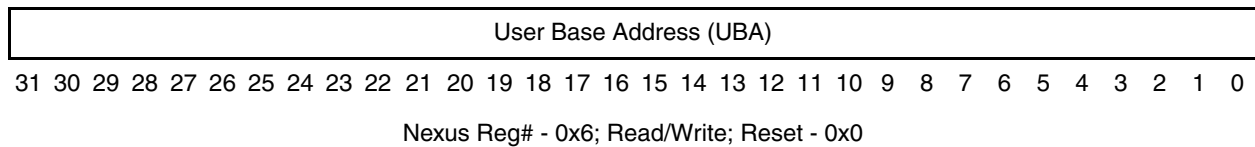


Figure B-8—User Base Address Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor Defined	UBA	--				Vendor-defined user base address

Table B-11—User Base Address Register Field

The memory map for user access of development features is shown in **Table B-12**, where offset is the base word size of the embedded processor.

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

Table B-12—Memory Map for User Accesses

The UBA Register is recommended for embedded processors complying with Class 2, 3, or 4. The UBA Register supports both Ownership Trace (OTM) and Data Acquisition Messaging (DQM) as outlined below.

B.5.7.1 Ownership Trace Messaging (OTM)

For embedded processors which do not implement internal task/process ID registers, the UBA shall be implemented as a register to which an operating system can write an ID for the current task/process. The size of the UBA register is vendor defined. Ownership Trace Messaging is required for clients complying with Classes 2, 3, and 4.

B.5.7.2 Data Acquisition Messaging (DQM)

Data Acquisition can be implemented using the UBA register. DQM is achieved by user writes to appropriate locations in the memory map shown in **Table B-12**. The write information is queued up for messaging via the auxiliary pins. Details on using the UBA to generate DQM messages can be found in **B.5.7 - User Base Address (UBA) Register** and **C.3 - Data Acquisition in Tuning for Applications**.

B.5.8 Development Status (DS) Register

The Development Status Register is used to report system debug status. The register can be read from external development tools at any time. A change of state for any bit within the DS register will trigger a Debug Status Message which can optionally contain the contents of the DS Register.

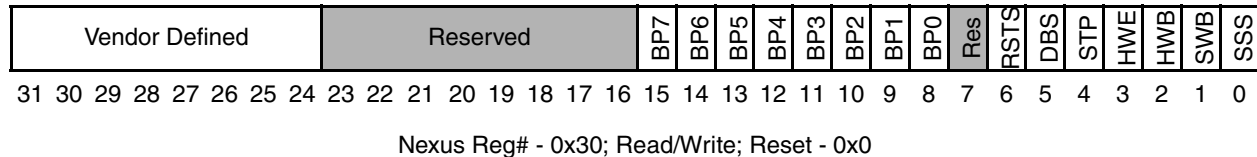


Figure B-9—Development Status Register

Class 4 Only:

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 can be transmitted out the auxiliary pins upon a change in state of any bit (see **Table B-13**).

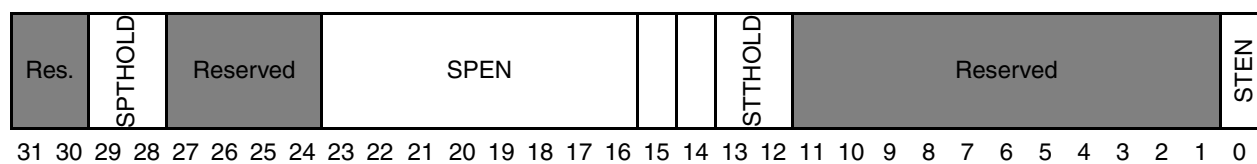
Bit Number	Field Name	Class				Description
		1	2	3	4	
31–24	—	--	--	--	--	Vendor-defined
23–16	—	--	--	--	--	Reserved
15–8	BP7-0	--	--	--	Y	<u>BPn - Breakpoint Status</u> 0 = No breakpoint 1 = Breakpoint occurred
7	—	--	--	--	--	Reserved
6	RSTS	--	--	--	Y	<u>RSTS - Reset Status</u> 0 = Processor not reset 1 = Processor reset since last DS Register read

Table B-13—Development Status Register Fields

Bit Number	Field Name	Class				Description
		1	2	3	4	
5	DBS	--	--	--	Y	<u>DBS - Debug Status</u> 0 = Processor not halted 1 = Processor halted in debug mode
4	STP	--	--	--	Y	<u>STP - Stop Status</u> 0 = Processor not stopped 1 = Processor stopped in low-power mode
3	HWE	--	--	--	Y	<u>HWE - Hardware Error</u> 0 = No hardware error 1 = Nonrecoverable hardware error occurred
2	HWB	--	--	--	Y	<u>HWB - Hardware Breakpoint Status</u> 0 = No hardware breakpoint 1 = Hardware breakpoint
1	SWB	--	--	--	Y	<u>SWB - Software Breakpoint Status</u> 0 = No software breakpoint 1 = Software breakpoint
0	SSS	--	--	--	Y	<u>SSS - Single-Step Status</u> 0 = Processor not halted 1 = Processor halted in debug mode after single step

Table B-13—Development Status Register Fields**B.5.9 Overrun Control (OVCR) Register**

The overrun control (OVC) field is used to determine control for overrun of BTM and DTM. Overruns can be handled by displaying an Error Message to development tools (indicating the types of messages lost), 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.



Nexus Reg# - 0x31; Read/Write; Reset - 0x0

Figure B-10—Overrun Control Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
31-30	—	--	--	--	--	Reserved

Table B-14—Overrun Control Register Fields

Bit Number	Field Name	Class				Description
		1	2	3	4	
29–28	SPTHOLD	--	Y	Y	Y	Suppression Threshold 00 = Suppression threshold when message queues are 1/4 full 01 = Suppression threshold when message queues are 1/2 full 10 = Suppression threshold when message queues are 3/4 full 11 = Vendor-defined
27–24	—	--	--	--	--	Reserved
23–16	SPEN	--	Y	Y	Y	Suppression Enable 00000000 = Suppression is disabled xxxxxxx1 = Ownership Trace message suppression enabled xxxxx1x = Data Trace message suppression enabled xxxx1xx = Program Trace message suppression enabled xxx1xxx = Watchpoint Trace message suppression enabled xxx1xxxx = In-Circuit Trace message suppression enabled xx1xxxxx = Data Acquisition message suppression enabled x1xxxxxx = Vendor Trace #1 message suppression enabled 1xxxxxxx = Vendor Trace #2 message suppression enabled
15–14	—	--	--	--	--	Reserved
13–12	STTHOLD	--	Y	Y	Y	Stall Threshold 00 = Stall threshold when message queues are 1/4 full 01 = Stall threshold when message queues are 1/2 full 10 = Stall threshold when message queues are 3/4 full 11 = Vendor-defined
11–1	—	--	--	--	--	Reserved
0	STEN	--	Y	Y	Y	Stall Enable 0 = Processor stalling is disabled 1 = Processor stalling is enabled

Table B-14—Overrun Control Register Fields**B.6 Read/Write Access Registers**

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:

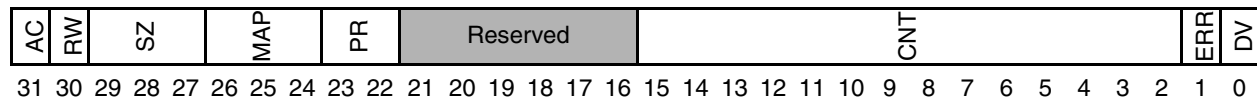
- Read/Write Access Control/Status (RWCS)
- Read/Write Access Address (RWA)
- Read/Write Access Data (RWD)

Read / Write Access is required for embedded processors complying with Class 3 or 4.

More detailed information on using the Read/Write Access feature is included in **Section C.1 - Read/Write Access Procedures**.

B.6.1 Read / Write Access Control and Status (RWCS) Register

The Read Write Access Control/Status Register provides control for Read/Write Access. Read/Write access provides DMA-like access to memory-mapped resources on an internal SoC bus either while the processor is halted, or during runtime. The RWCS Register also provides Read/Write Access Status information per **Section C.1 - Read/Write Access Procedures**



Nexus Reg# - 0x7; Read/Write¹; Reset - 0x0

Figure B-11—Read / Write Access Control and Status Register

1. ERR and DV are read-only

Class 3 and Class 4:

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-16**).

If an error is generated during a block access, the block access will be terminated (see

Table B-15).

Bit Number	Field Name	Class				Description
		1	2	3	4	
31	AC	--	--	Y	Y	<u>AC - Access Control</u> 0 = End Access 1 = Start access
30	RW	--	--	Y	Y	<u>RW - Read/Write</u> 0 = Read access 1 = Write access
29–27	SZ	--	--	Y	Y	<u>SZ - Word Size</u> 000 = 8-bit 001 = 16-bit 010 = 32-bit 011 = 64-bit 1xx = Reserved
26–24	MAP	--	--	Y	Y	<u>MAP - Map Select</u> 000 = Primary memory map 001–111 = Other memory maps
23–22	PR	--	--	Y	Y	<u>PR - Priority</u> bb = Access priority
21–16	—	--	--	Y	Y	Reserved
15–2	CNT	--	--	Y	Y	<u>CNT - Access Count</u> hhhh = Number of accesses of word size SZ
1	ERR	--	--	Y	Y	Last access generated an error
0	DV	--	--	Y	Y	Data valid in RWD

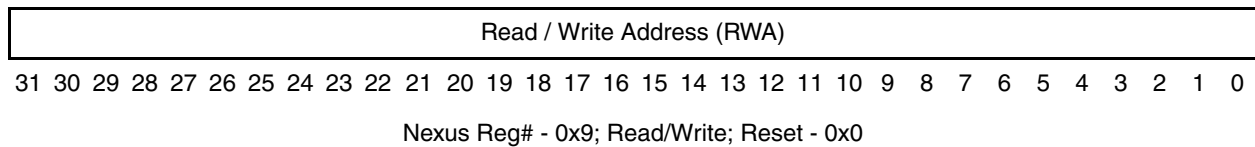
Table B-15—Read/Write Access Control and Status Register Fields

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

Table B-16—Read/Write Access Status Bit Encoding**B.6.2 Read / Write Access Address (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. An example 32-bit RWA is depicted in

Figure B-12— - Read / Write Access Address Register.**Figure B-12—Read / Write Access Address Register**

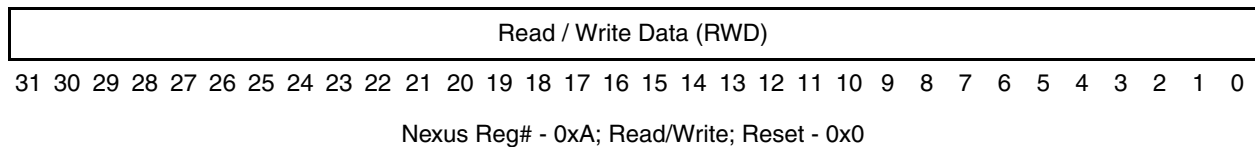
Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor Defined	RWA	--	--	Y	Y	User memory-mapped address to be accessed

Table B-17—Read / Write Access Address Register Field

B.6.3 Read / Write Access Data (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. An example 32-bit RWD is depicted in **Figure B-13— - Read / Write Access Data Register.**

**Figure B-13—Read / Write Access Data Register**

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor Defined	RWD	--	--	Y	Y	Data read from a user memory-mapped location or to be written to a user memory-mapped location

Table B-18—Read / Write Access Data Register Field

For read and write accesses, the register may contain different sizes of data. Table **Table B-19— - RWD Data Placement for Transfers** shows the byte organization for different sizes of read/write data.

Transfer Size and byte offset	RWA[2:0]	RWCS[SZ]	RWD			
			31:24	23:16	15:8	7:0
8-bit (byte)	x x x	0 0 0	-	-	-	X
16-bit (halfword)	x x 0	0 0 1	-	-	X	X
32-bit (word)	x 0 0	0 1 0	X	X	X	X
64-bit (double-word)	0 0 0	0 1 1				
first RWD pass (low order data)			X	X	X	X
second RWD pass (high order data)			X	X	X	X

Table Notes:

“X” indicates byte lanes with valid data

“-” indicates byte lanes which will contain unused data.

Table B-19—RWD Data Placement for Transfers

B.7 Data Trace Registers

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)

Data Trace is required for embedded processors complying with Class 3 or 4.

B.7.1 Data Trace Control (DTC) Register

The Data Trace Control Register controls whether DTM Messages are restricted to reads, writes, or both for a user programmable address range. There are up to six (6) Data Trace

channels controlled by the DTC.

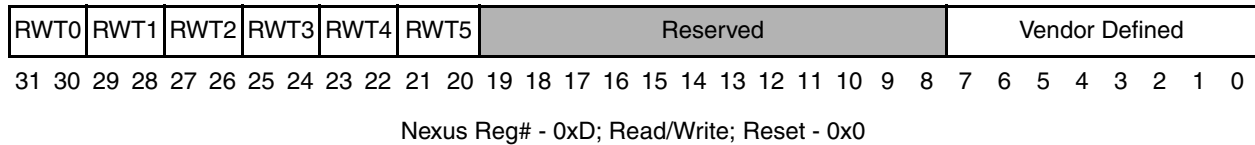


Figure B-14—Data Trace Control Register

Class 3 or Class 4:

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-20**).

Bit Number	Field Name	Class				Description
		1	2	3	4	
31–30	RWT0	--	--	Y	Y	<u>RWT0 - Read/Write Trace 1</u> 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
29–28	RWT1	--	--	Y	Y	<u>RWT1 - Read/Write Trace 2</u> 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
27–26	RWT2	--	--	Y	Y	<u>RWT2 - Read/Write Trace 3</u> 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
25–24	RWT3	--	--	Y	Y	<u>RWT3 - Read/Write Trace 4</u> 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
23–22	RWT4	--	--	Y	Y	<u>RWT4 - Read/Write Trace 5</u> 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
21–20	RWT5	--	--	Y	Y	<u>RWT5 - Read/Write Trace 6</u> 00 = No Data Trace Messages generated x1 = Enable data read trace 1x = Enable data write trace
19–8	—	--	--	--	--	Reserved
7–0	—	--	--	--	--	Vendor-defined

Table B-20—Data Trace Control Register Fields

B.7.2 Data Trace Start Address (DTSA) and End Address (DTEA) Registers

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. An example 32-bit DTSA/DTEA is depicted in **Figure B-15—Data Trace Start / End Address Registers**.

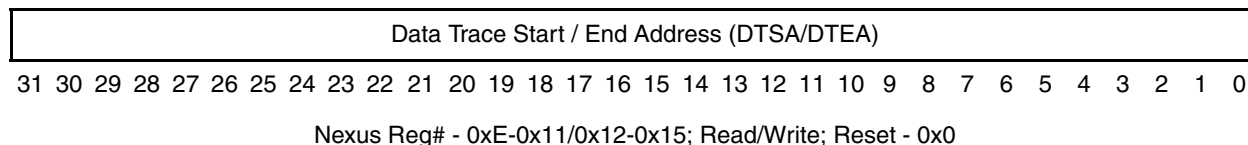


Figure B-15—Data Trace Start / End Address Registers

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor Defined	DTSA	--	--	Y	Y	Start address for data trace visibility

Table B-21—Data Trace Start Address Register Field

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor Defined	DTEA	--	--	Y	Y	End address for data trace visibility

Table B-22—Data Trace End Address Register Field

B.8 Breakpoint/Watchpoint Registers

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)
- Watchpoint Mask (WMSK) Register
- Watchpoint Trigger (WT) Register

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

B.8.1 Breakpoint / Watchpoint Control (BWC) Register

Class 4 Only:

For all breakpoints to be enabled, the DBE field must be set to enable debug mode (refer to **B.5.3 - Development Control (DC1) Register 1**). 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 selects if 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.

Watchpoints can be assigned actions listed in **Table B-27**. 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.

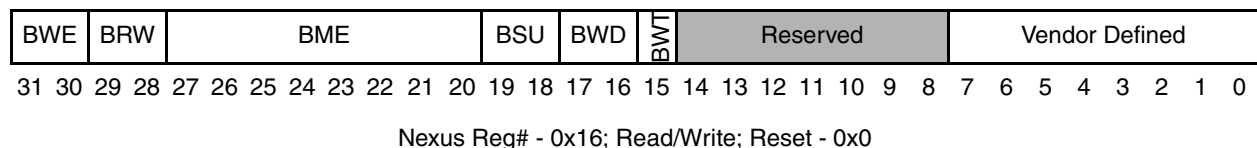


Figure B-16—Breakpoint / Watchpoint Control Register

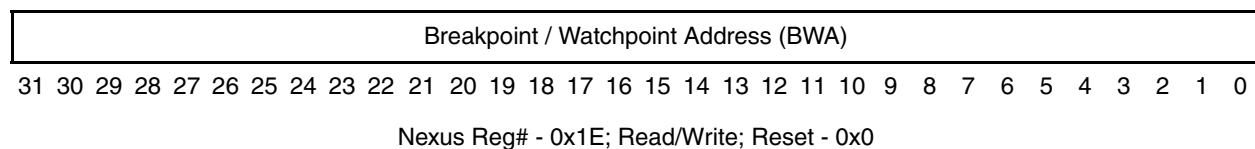
Bit Number	Field Name	Class				Description
		1	2	3	4	
31–30	BWE	--	--	--	Y	<u>BWE - Breakpoint/Watchpoint Enable</u> 00 = Disabled 01 = Breakpoint enabled 10 = Reserved 11 = Watchpoint enabled
29–28	BRW	--	--	--	Y	<u>BRW - Breakpoint/Watchpoint Read/Write Select</u> 00 = Break on read access 01 = Break on write access 10 = Break on any access 11 = Reserved

Table B-23—Breakpoint / Watchpoint Control Register Fields

Bit Number	Field Name	Class				Description
		1	2	3	4	
27–20	BME	--	--	--	Y	<u>BME - Breakpoint/Watchpoint Address/Data Mask Enable</u> 1XXXXXXX = Mask MS address/data size unit : XXXXXXX1 = Mask LS address/data size unit
19–18	BSU	--	--	--	Y	<u>BSU - Breakpoint/Watchpoint Data Size Unit (read only)</u> 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	--	--	--	Y	<u>BWO - Breakpoint/Watchpoint Operand</u> 1X = Compare with BWA value X1 = Compare with BWD value
15	BWT	--	--	--	Y	<u>BWT - Breakpoint/Watchpoint Type</u> 0 = Compare for instruction types 1 = Compare for data types
14–8	—	--	--	--	--	Reserved
7–0	—	--	--	--	--	Vendor-defined

Table B-23—Breakpoint / Watchpoint Control Register Fields**B.8.2 Breakpoint / Watchpoint Address (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-24**). An example 32-bit BWA is depicted in **Figure B-17— - Breakpoint / Watchpoint Address Register**.

**Figure B-17—Breakpoint / Watchpoint Address Register**

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor Defined	BWA	--	--	--	Y	Address of instruction or data for breakpoint or watchpoint generation

Table B-24—Breakpoint / Watchpoint Address Register Field**B.8.3 Breakpoint / Watchpoint Data (BWD) Register**

The BWD Register is used to compare against data operands (instruction encoding or data

value). The size of the BWD Register is vendor defined (see **Table B-25**). An example 32-bit BWD is depicted in **Figure B-18— Breakpoint / Watchpoint Data Register**.

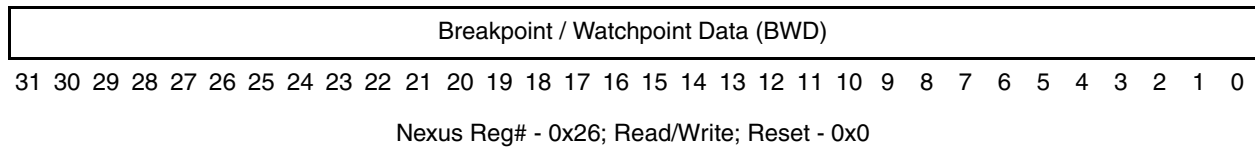


Figure B-18—Breakpoint / Watchpoint Data Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor-Defined	BWD	--	--	--	Y	Instruction opcode or data value for breakpoint or watchpoint generation

Table B-25—Breakpoint / Watchpoint Data Register Field

B.8.4 Watchpoint Mask (WMSK) Register

The Watchpoint Mask register controls which watchpoint events are enabled to produce Watchpoint Trace Messages. The WEM field supports up to 16 watchpoint sources. Additional watchpoints and/or vendor-defined watchpoint message control can be defined with the additional 16 bits in WMSK.

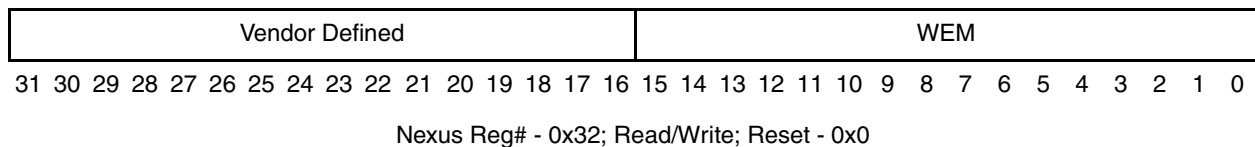


Figure B-19—Watchpoint Mask Register

NOTE

The TM field within the DC1 Register must also be programmed to generate Watchpoint Trace Messages in order to use the watchpoint masking feature.

Table B-26 details the Watchpoint Mask Register fields.

Bit Number	Field Name	Class				Description
		1	2	3	4	
31-16	—	--				Vendor defined.
15-0	WEM	--	Y	Y	Y	<p>WEM - Watchpoint Enable for Watchpoint Trace Messaging (WTM)</p> <p>0000000000000000 = No Watchpoints enabled for WTM</p> <p>XXXXXXXXXXXXXXXXX1 = Watchpoint #1 enabled for WTM</p> <p>XXXXXXXXXXXXXXXXX1X = Watchpoint #2 enabled for WTM</p> <p>XXXXXXXXXXXXXXXXX1XX = Watchpoint #3 enabled for WTM</p> <p>XXXXXXXXXXXXXXXXX1XXX = Watchpoint #4 enabled for WTM</p> <p>XXXXXXXXXXXXXXXXX1XXXX = Watchpoint #5 enabled for WTM</p> <p>XXXXXXXXXXXXXXXXX1XXXXX = Watchpoint #6 enabled for WTM</p> <p>XXXXXXXXXX1XXXXXXXXX = Watchpoint #5 enabled for WTM</p> <p>XXXXXXXXXX1XXXXXXXXX = Watchpoint #8 enabled for WTM</p> <p>XXXXXXXXX1XXXXXXXXX = Watchpoint #9 enabled for WTM</p> <p>XXXXXX1XXXXXXXXXXXXX = Watchpoint #10 enabled for WTM</p> <p>XXXXX1XXXXXXXXXXXXX = Watchpoint #11 enabled for WTM</p> <p>XXXX1XXXXXXXXXXXXX = Watchpoint #12 enabled for WTM</p> <p>XXX1XXXXXXXXXXXXX = Watchpoint #13 enabled for WTM</p> <p>XX1XXXXXXXXXXXXX = Watchpoint #14 enabled for WTM</p> <p>X1XXXXXXXXXXXXX = Watchpoint #15 enabled for WTM</p> <p>1XXXXXXXXXXXXX = Watchpoint #16 enabled for WTM</p>

Table B-26—Watchpoint Mask Register Fields

B.8.5 Watchpoint Trigger (WT) Register

The WT Register allows watchpoints such as those defined in the breakpoint/watchpoint registers (refer to **B.8 - Breakpoint/Watchpoint Registers**) to be assigned trigger actions.

PTS	PTE	DTS	DTE	ITS	ITE	MSS	Vendor Defined
-----	-----	-----	-----	-----	-----	-----	----------------

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Nexus Reg# - 0xB; Read/Write; Reset - 0x0

Figure B-20—Watchpoint Trigger Register

The trace start (PTS, DTS, ITS) and trace end (PTE, DTE, ITE) fields select watchpoints to enable and disable each selected trace feature, effectively producing an address- and/or data-related “window” for triggering trace. Individual trace features are triggered via the WT setting if the TM field (refer to **B.5.3 - Development Control (DC1) Register 1**) has not already enabled that trace feature.

The memory substitution start (MSS) field selects a watchpoint to trigger memory substitution. (See **Table B-27.**) Refer to **B.5.3 - Development Control (DC1) Register 1**

for additional fields related to memory substitution.

Bit Number	Field Name	Class				Description
		1	2	3	4	
31–28	PTS	--	Y	Y	Y	<u>PTS - Program Trace Start Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
27–24	PTE	--	Y	Y	Y	<u>PTE - Program Trace End Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
23–20	DTS	--	Y	Y	Y	<u>DTS - Data Trace Start Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
19–16	DTE	--	Y	Y	Y	<u>DTE - Data Trace End Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
15–12	ITS	--	Y	Y	Y	<u>ITS - In-Circuit Trace Start Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
11–8	ITE	--	Y	Y	Y	<u>ITE - In-Circuit Trace End Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
7–4	MSS	--	Y	Y	Y	<u>MSS - Memory Substitution Start Control</u> 0000 = Trigger disabled 0001 = 1111 Use watchpoint 1–15
3-0	--	--	--	--	--	Vendor-defined

Table B-27—Watchpoint Trigger Register Fields

B.9 Aurora Registers

The Nexus Aurora interface requires some control registers for controlling the features of the Aurora interface

- Aurora Trace Transmission Control (ATTC)
- Aurora Trace TRansmission Status Register (ATTS)
- Aurora Link Control Register (ALC)
- Aurora Training Control Register (ATC)
- Aurora Link Status (ALS)
- Aurora Link Error Register (ASC)

These registers are recommended for embedded processors complying with Class 3 utilizing the Nexus serial Aurora trace interface.

B.9.1 Aurora Trace Transmission Control Register (ATTC)

The Aurora Trace Transmission Control Register controls whether the trace is enabled or not. If enabled, the SERDES PHY is powered, and the FIFO will enter trace packets from the selected CPU core. The format of the Trace Control Register is shown in Table 1 Trace Control Register (CR). Enabling the TE bit causes a reset sequence to occur in the Trace Interface. This sequence includes powering up the PHY and deasserting the TX_Reset signal to the Aurora IP. This combination causes a link initialization sequence. See **D.4 - Initialization** for details on link initialization.

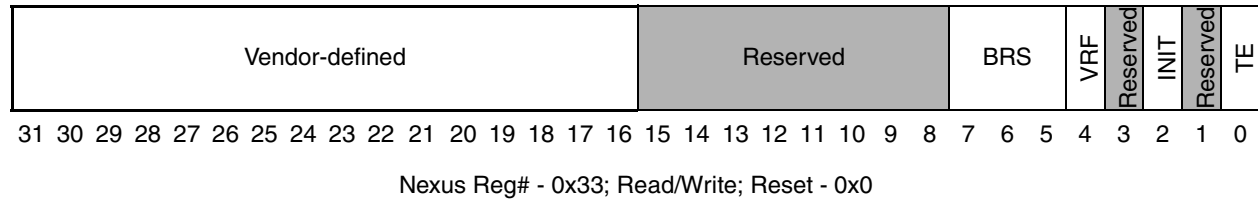


Figure B-21—Aurora Trace Transmission Control Register

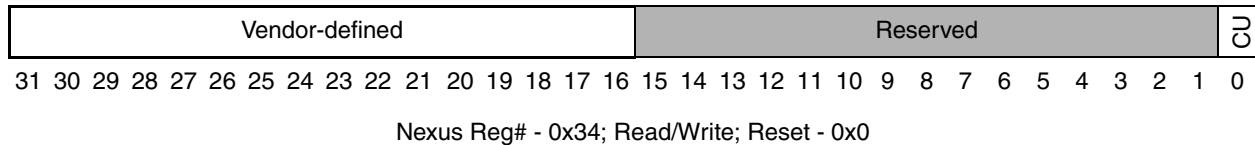
Bit Number	Field Name	Description
31–16	—	Vendor-defined
15–8	—	Reserved
5–7	BRS	<u>BRS - Baud Rate Select</u> 000 = Default baud rate (nominal frequency for trace data transmission) 001-111 = Additional baud rate selection (optional) Selects the baud rate for which the trace port operates. The baud rate selection speeds are vendor-defined and assigned.
4	VRF	<u>VRF - Verification</u> 0 = Receiver is not verified 1 = Receiver is verified (ready to transmit trace packets) Receiver sets this bit when it has successfully verified. When both this bit and the INIT bit are set the device is ready to receive trace packets. This bit is automatically cleared by hardware when the device enters the initialization stage
3	—	Reserved
2	INIT	<u>INIT - Initialization</u> 0 = Receiver is not initialized (initialization pattern should be sent) 1 = Receiver is trained Receiver sets this bit when it has successfully initialized all lanes. Receiver clears this bit when it has lost synchronization that is experienced channel transmission errors (example: wrong 8B/10B pattern)
1	—	Reserved

Table B-28—Aurora Trace Transmission Control Register (ATTC)

Bit Number	Field Name	Description
0	TE	<u>TE - Trace Transmission Enable</u> 0 = Disable Trace Transmission 1 = Enable Trace Transmission Setting this bit enables the trace interface. A transition from disable to enable causes the trace interface to reset. The Aurora interface logic shall begin the initialization sequence as defined in the Aurora specification. The exact behavior of reset toward the rest of the trace logic is implementation specific.

Table B-28—Aurora Trace Transmission Control Register (ATTC)**B.9.2 Aurora Trace Transmission Status Register (ATTS)**

The Aurora Trace Transmission Status Register (ATSR) reports status of the trace interface hardware.

**Figure B-22—Aurora Trace Transmission Status Register**

Bit Number	Field Name	Description
31-16	—	Vendor-defined
15-1	—	Reserved
0	CU	<u>CU - Channel UP Status</u> 0 = Receiver is not initialized 1 = Aurora link initialized / verified This status bit indicates that the Aurora link has initialized and been verified and the transmitter is ready to transmit data.

Table B-29—Aurora Trace Transmission Status Register (ATSR)**B.9.3 Aurora Link Control Register**

The Aurora Link Control Register (ALC) allows control of any programmable features of the Nexus Aurora Link. The size of the ALC Register is vendor defined. An example 32-

bit ALC is depicted in **Figure B-23— Aurora Link Control Register**.

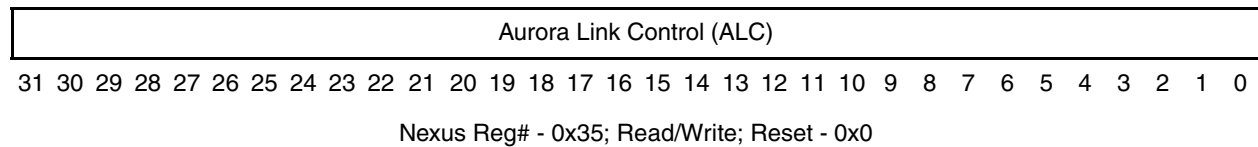


Figure B-23—Aurora Link Control Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor-Defined	—	--		Y	Y	Link Control bits

Table B-30—Aurora Link Control Register Field

B.9.4 Aurora Training Control Register

The Aurora Link Training Control Register (ATC) allows control of the training time for the Nexus Aurora Link. The size of the ATC Register is vendor defined. An example 32-bit ATC is depicted in **Figure B-23— Aurora Link Control Register**.

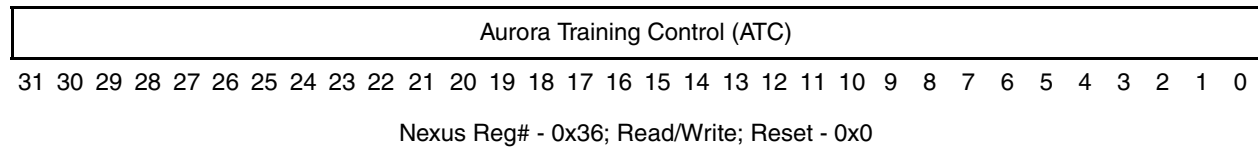


Figure B-24—Aurora Training Control Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor-Defined	—	--	--	Y	Y	Training control time

Table B-31—Aurora Training Control Register Field

B.9.5 Aurora Link Status Register

The Aurora Link Status Register (ALS) provides a read-only status of the Nexus Aurora Link. The size of the ALS Register is vendor defined. An example 32-bit ALS is depicted

in **Figure B-25— Aurora Link Status Register.**

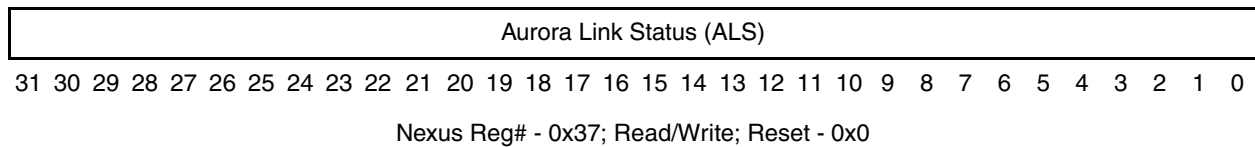


Figure B-25—Aurora Link Status Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor-Defined	—	--	--	Y	Y	Aurora status bits

Table B-32—Aurora Link Status Register Field

B.9.6 Aurora Link Error Register

The Aurora Link Error Register (ALE) provides status of the Nexus Aurora Link. The size of the ALE Register is vendor defined (see **Table B-33**). An example 32-bit ALE is depicted in **Figure B-26— Aurora Error Control Register.**

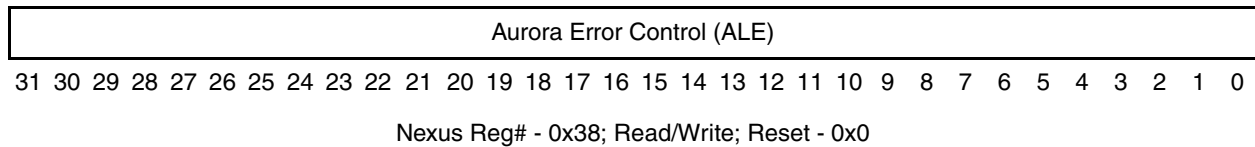


Figure B-26—Aurora Error Control Register

Bit Number	Field Name	Class				Description
		1	2	3	4	
Vendor-Defined	—	--	--	Y	Y	Aurora error bits

Table B-33—Aurora Error Register Field

APPENDIX C

Application Notes

The Application Notes appendix is intended to provide some guidance in interpreting some of the more common uses of the Nexus standard. For example, this chapter outlines steps required to utilize Nexus resources for block accesses as well as implementation recommendations for Nexus data formatting for deployment over the serial AUX port. It also provides background for features such as data acquisition.

C.1 Read/Write Access Procedures

The external development 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.

Specific steps for access via the TAP interface and the AUX interface are outlined below.

C.1.1 Block Access with the TAP Interface

The TAP state machine is shown in **Figure 6-4**. 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 **TAP** access method outlined in **Section B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** using the NRR index of 0x9 (see **Table B-3**).
 - b. Initialize the RWCS Register through the **TAP** access method outlined in **Section B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** using the NRR index of 0x7 (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. This indicates that the device is ready for the next access.
 - d. The data can then be read from the RWD Register through the **TAP** access method outlined in **Section B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** using the NRR index of 0xA (see **Table B-3**).

- e. Once the RWD value has been read, the RWA 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 **TAP** access method outlined in **Section B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** using the NRR index of 0x9 (see **Table B-3**).
 - b. Initialize the RWD Register through the **TAP** access method outlined in **Section B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** using the NRR index of 0xA (see **Table B-3**).
 - c. Initialize the RWCS Register through the **TAP** access method outlined in **Section B.3 - Access with the TAP Interface (IEEE 1149.1/IEEE 1149.7)** using the NRR index of 0x7 (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. This 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.

C.1.2 Access with the Auxiliary Port

Use the Auxiliary Access Public Messages described in **4.3.26 - Auxiliary Access - Read Message** - through **4.3.29 - Auxiliary Access - Response Message**.

C.1.2.1 Auxiliary Access Messages - Example Sequences

The following examples outline example sequences using the Nexus defined AUX access messages to access both Nexus Recommended Registers (NRRs) as well as memory mapped resources on the target (or tool),

Table C-1 outlines the sequence of messages when the development tool requests debug status information from the target.

Step #	Description of Action
1	Tool sends AUX Read Message (with DS Register index and XMAP set to NRR)
2	Target sends AUX Response Message (with DS information)

Table C-1—Target Debug Status Requested by Tool

Table C-2 outlines the sequence of messages when the development tool sends a debug command to the target.

Step #	Description of Action
1	Tool sends AUX Write Message (with XMAP set to NRR)
2	Target sends AUX Response Message

Table C-2—Debug Command Sent by Tool to Target

Table C-3 through **Table C-7** as well as **Figure C-1** 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, AUX Read/Write Next Data Messages should be used wherever possible because these messages have the shortest length.

Step #	Description of Action
1	Tool sends AUX Read Message (with XMAP set to a memory map)
2	Target sends AUX Response Message (includes transaction status and data)
3	Tool sends AUX Read Next Message
4	Target sends AUX Response Message (includes transaction status and data)
5	Tool sends AUX Read Next Message
6	Target sends AUX Response Message (includes transaction status and data)

Table C-3—Reading Consecutively Addressed Target Locations

Step #	Description of Action
1	Tool sends AUX Write Message (with XMAP set to a memory map)
2	Target sends AUX Response Message (includes transaction status)
3	Tool sends AUX Write Next Message
4	Target sends AUX Response Message (includes transaction status)
5	Tool sends Write Next Message
6	Target sends AUX Response Message (includes transaction status)

Table C-4—Writing Consecutively Addressed Target Locations

Step #	Description of Action
1	Tool sends AUX Read Message
2	Target sends AUX Response Message (includes transaction status and data)
3	Tool sends AUX Read Message
4	Target sends AUX Response Message (includes transaction status and data)

Table C-5—Reading Randomly Addressed Target Locations

Step #	Description of Action
5	Tool sends AUX Read Message.
6	Target sends AUX Response Message (includes transaction status and data)

Table C-5—Reading Randomly Addressed Target Locations

Step #	Description of Action
1	Tool sends AUX Write Message (includes data)
2	Target sends AUX Response Message (includes transaction status)
3	Tool sends AUX Write Message (includes data)
4	Target sends AUX Response Message (includes transaction status)
5	Tool sends AUX Write Message (includes data)
6	Target sends AUX Response Message (includes transaction status)

Table C-6—Writing Randomly Addressed Target Locations

Step #	Description of Action
1	Tool sends AUX Read Message
2	Target sends AUX Response Message (includes transaction status and data)
3	Tool sends AUX Write Message (includes data)
4	Target sends AUX Response Message (includes transaction status)
5	Tool sends AUX Write Message (includes data)
6	Target sends AUX Response Message (includes transaction status)
7	Tool sends AUX Read Message
8	Target sends AUX Response Message (includes transaction status and data)

Table C-7—Intermixed Reading/Writing Randomly Addressed Target Locations

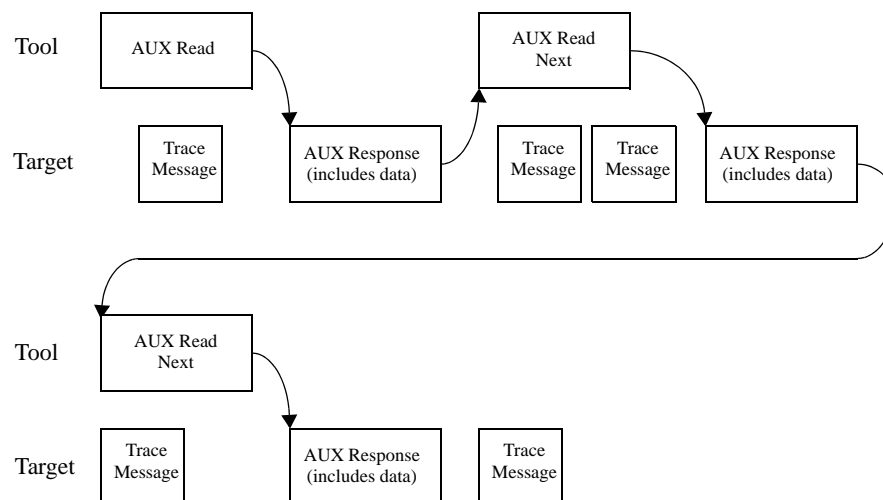


Figure C-1—Trace Messages Intermixed with Read Target Locations

Because trace messages do not need to be acknowledged by the tool, these messages can be output whenever the auxiliary output port is not transmitting other messages. **Figure C-1** 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.

A target uses the same AUX Access Messages and sequences as described above to access memory space within a tool, except that all messages occur in the reverse direction. AUX Read/Write Messages include an optional XMAP 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 hit 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.

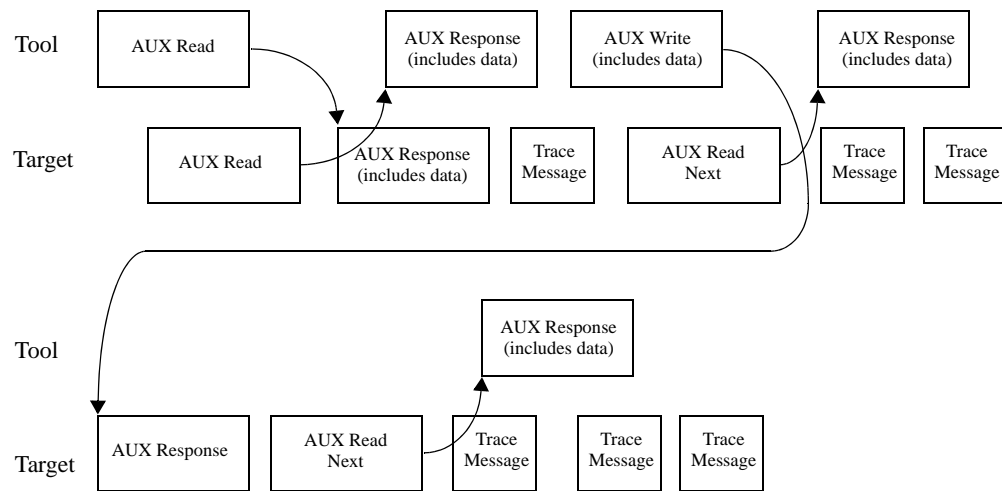


Figure C-2—Simultaneous Read/Write Accesses by Tool and Target

C.1.2.2 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 AUX Read Next 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 AUX Read Next Messages or AUX Write Next Messages. The target simply increments an address counter each time it receives an AUX Read Next Message or AUX Write Next Message. This address counter is automatically changed to a new value whenever another AUX Read/Write Message is received.

During the transfer of a large block of data using AUX Read/Write Next 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 an AUX Response Message with the ST field = 0b01 (meaning that the previously requested read/write operation cannot be completed). The tool stops sending any more AUX Read/Write Next Messages and may then perform some recovery or error notification tasks.

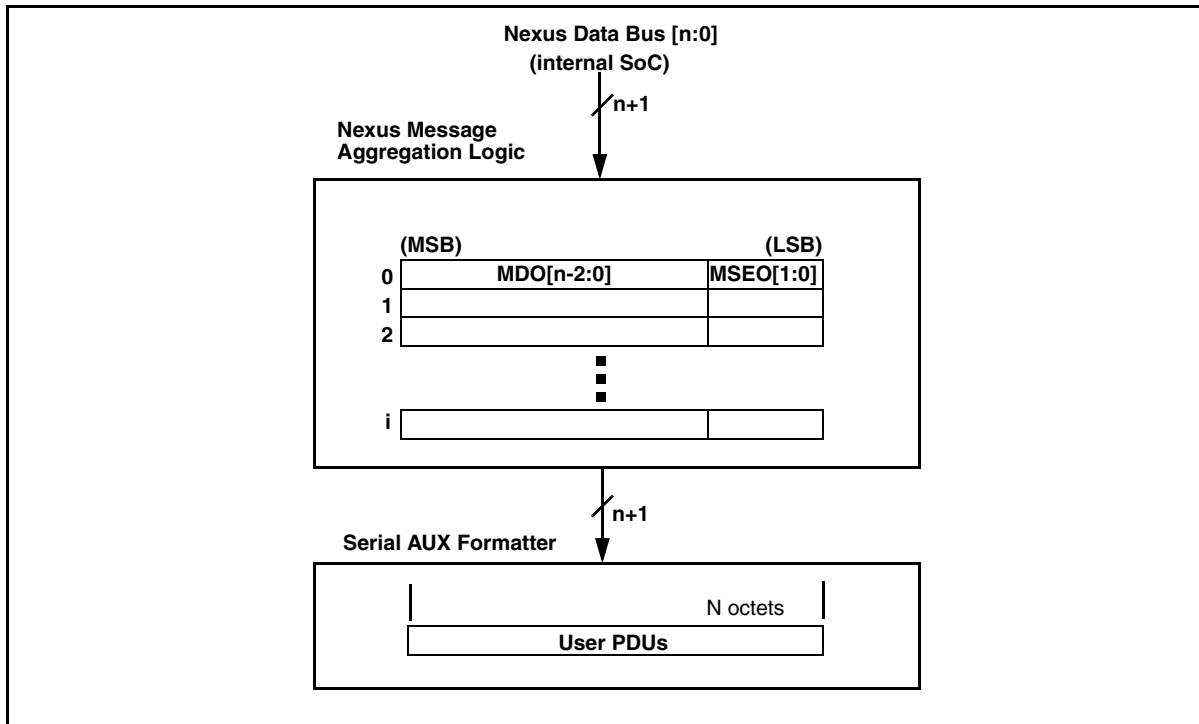
C.2 Recommendations for Nexus Data Formatting (using serial AUX)

As outlined in **SECTION 6 - Nexus Port Interfaces**, IEEE-ISTO 5001 supports various interface protocols for transmitting Nexus formatted from an embedded processor.

In order to support transmission via the serial AUX port, the internal SoC Nexus data needs to be formatted and transmitted from an internal Nexus client to the module which

prepares the data for transmission via the serial AUX port. In other words, converting the Nexus formatted data into the Aurora protocol.

Figure C-3—Nexus Data Formatting Flow



The simplest implementation of Nexus data formatting entails creating internal data buses of standard lengths $n+1$ (16/32/64/etc) in which the two LSB bits will be MSEO[1:0], and the remaining $n-1$ bits will represent MDO data. In this way, each data value represents a “beat” of Nexus data. The number of beats required to transmit a single Nexus message depends on the type of message as well as the size of the individual fields (ADDR, DATA, etc). The choices for Nexus data bus width are implementation specific.

Figure C-3 shows a typical flow for SoC Nexus data as it moves from a Nexus client to the module which prepares the data for serial AUX transmission.

C.3 Data Acquisition in Tuning for Applications

DQM is achieved by user writes to appropriate locations in the memory map shown in **Table B-12**. 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 **3.11 - 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 or greater.

C.3.1 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. This 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. This 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.ieee-isto.org/Nexus5001/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.3.2 Data Acquisition or Measurement of Calibration Variables

Two options are explained in **C.3.2.1 - DTM Option** and **C.3.2.2 - Read/Write Access Option** to meet the data acquisition needs discussed in **C.3.1 - Additional Needs for Automotive Powertrain and Disk Drive Development**. The first utilizes DTM and the second utilizes the Read/Write Access feature.

C.3.2.1 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 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 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.3.2.2 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. A specific embedded processor data write sequence or a watchpoint occurrence and message would be used to signal the tool to acquire the calibration variables. The tool would acquire the calibration variables using the Read/Write Access feature.

C.3.3 Tuning of Calibration Constants

The Nexus standard provides features to support program execution tuning, also referred to as calibration constant tuning. This is required when tuning 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 TAP 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

Nexus Aurora Details

D.1 Back Channel and Flow Control

Flow control and back channel functionality can be fully implemented when operating in any form of duplex mode as stated in the Aurora specification. When operating in a simplex mode, training and control is achieved through JTAG. Control of the trace interface is accomplished with a set of registers that can be read and written through the JTAG interface. Two registers, Trace Transmission Control Register and Trace Transmission Status Register, are described in the sections that follow. The following sections are only for simplex mode operation.

D.2 Aurora Registers

See the register appendix for recommended registers to support the Nexus Aurora trace interface. Specifically, **Section APPENDIX C - Application Notes** and **Section B.9.2 - Aurora Trace Transmission Status Register (ATTS)**.

D.3 Simplex Back Channel

For a simplex trace port, a back channel is required to communicate the status of the probe receiver to the transmitter. This status is communicated to the transmitter using JTAG commands. The back channel is used to begin initialization and indicate that the receiver is ready for channel verification and data reception. During transmission it also shall indicate a loss of synchronization and make requests for channel reset.

In order to avoid potential time-outs during the initialization sequence, it is recommended that the transmitter implement a counter to transition through the three stages (after the initialization process has started). The length for this counter is implementation specific, but the maximum value should correlate to the time-out value supported by the logic that supports training the Aurora link.

D.4 Initialization

Initialization of an Aurora interface is defined as a three-stage process. The stages are lane initialization, channel bonding, and channel verification. For single lane implementations, channel bonding is not required and therefore bypassed.

Figure 7-1— Aurora channel initialization sequence

D.4.1 Lane Initialization

Lane initialization synchronizes the trace port transmitter with the probe receiver through the transmission of a known pattern (/SP/ and /SPA/). The receiver begins the initialization process by setting the TCR[INIT] bit. A channel failure can be triggered by either channel data errors or protocol violations. After a reset event or a channel failure, the receiver resets the interface by clearing the TCR[INIT] bit through JTAG. Refer to the “8B/10B Initialization and Error Handling” section of the Aurora protocol specification for the link initialization sequence.

D.4.2 Channel Bonding

When multiple signal lanes are deployed, the channel alignment sequence provides a mechanism to achieve lane alignment. Channel alignment is a sequence of /I/ and /A/ symbols. Refer to the Aurora protocol specification for the channel alignment sequence specifics. For single lane applications the channel alignment state is bypassed.

Channel bonding cannot begin until all lanes have initialized and the link counter has transitioned the transmitter to the channel bonding state.

D.4.3 Channel Verification

Channel verification verifies to the ability of the channel to transfer valid data across the interface. The verification process uses a known data sequence. The sequence consists of 60 idle symbols (/I/) followed by 4 verification sequences (/V/). Refer to the “8B/10B Channel Verification” section of the Aurora protocol specification for details of the process. As soon as the channel verification pattern has been transmitted, the transmitter assumes that the receiver has successfully verified and subsequently moves to the Channel Up state. The transmitter shall commence sending idle patterns /I/ until the TCR[VERIF] bit is set by the receiver.

If the receiver fails to successfully receive the verification pattern as expected after the channel alignment then it should clear the TCR[INIT] bit forcing the transmitter to return to the reset and lane initialization state and repeat the initialization process.

When the receiver is fully ready to receive data, (all lanes initialized, bonded, verified, and logic ready to receive packets) it shall write the TCR[VERIF] bit. This bit indicates that the transmitter can commence sending trace data.

Once the channel is verified and the Aurora transmitter is ready the Channel Up status is reflected by the TSR[CU] bit.

APPENDIX E

References

IEEE Std 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture (includes IEEE Std 1149.1a-1993).

IEEE Standard 1149.7, IEEE Standard

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

APPENDIX F

Revision History

IEEE-ISTO 5001-1999: Initial draft of the IEEE-ISTO 5001 standard

IEEE-ISTO 5001-2003 (version 2.0): Update to address over-sights in the original revision of the revision. Updated connectors definitions.

IEEE-ISTO 5001-2012 (version 3.0): Document reorganized. High Speed Nexus Aurora option added, including new connector options. Updated connector definitions for the new Samtec parallel connector. Some messages optimized. IEEE 1149.7 option added. 2-pin MSEO option start to start removed. Support for bus monitor, alternate masters added. Recommended register bit positions expanded for more granularity. Support for In-Circuit Trace messages added. Periodic OTM made optional. Additional recommended error codes added. Better support for alternative processor data sizes added. Inclusion of branch instructions in the I-CNT field. VALTREF signal renamed VSTBY. VENDOR_IO and TOOL_IO combined and renamed General_IO.