Specification of RISC-V Trace Control Interface

Version 0.9.10, Dec 20, 2022: This document is in Stable state. Assume it may change.

Table of Contents

Preamble

Document State

2022/12/20: Version 0.9.10-Stable

See wiki.riscv.org/display/HOME/Specification+States for explanation of specification states.

History and status

After approval by the group and status change from Stable to Frozen this chapter will be removed.

2022/12/20: Version 0.9.10-Stable

- All updates as outlined in email after TG meeting (and emails from Iain and Markus)

2022/12/19: Version 0.9.9-Stable

- Expanded Accessing Trace Control Registers
- Added chapter 'Versioning of Components' and tr??VerMajor and tr??VerMinor (instead of tr??Version)
- Elaborated 'Reset and Discovery' and 'Enabling and Disabling' chapters
- Order with width of columns (a lot of pain ...)

2022/12/6: Version 0.9.8-Stable

- Going back to fixed columns (to be adjusted) - automatic resize was horrible
- Updates after discussion by email and recent TG meeting
  - Explicitly enforcing one encoder for each hart
  - Renamed 'ATB Sink' to 'ATB Bridge' to clarify purpose
  - Moved 'trTeWrapOnFull' to 'trRamControl'
  - Using 'hart' instead of 'core' ('core' remains in several places)
  - Clarifying reserved registers (making room for custom range and Coresight compliance registers)
  - Other minor changes/typos
- Other changes
  - Fixing spec of some trRam... registers (some names were not correct)
  - Changed 'tref/tdata' to TRC_CLK/TRC_DATA (in PIB section and PIB related diagrams) to match names in Trace Connectors.

2022/11/29: Version 0.9.7-Stable

- Improved "Accessing Trace Control Registers" chapter
- Fixed too narrow columns in some tables

2022/11/28: Version 0.9.6-Stable

- Big re-structuring (split into separared components) as discussed.
- Consistent naming for all registers and fields (tr...) - each name of 'unique'.
- Allowing 64-bit values for RAM Sink WP/RP/Limit registers (allowing trace bigger than 4GB).

- Other minor clarifying changes.

2022/11/1: Version 0.9.5-Stable (never generated)

2022/8/15: Version 0.9.4-Stable.

- Most clarifying notes/suggestions (by Markus in email from 2022/8/12) taken into account
  - STILL TODO: Elaborate on 'funnels' and 'timestamps' sections (with graphics)

2022/8/9: Version 0.9.3-Stable. Key changes (after discussions/comments)

- Added dedicated Trace Filter block at offset 0x300
- Changed 'teTracing' into 'teInstTracing'
- Removed 'write 0 to clear'
- Adding 'teControl.teContext'
- Clarification of disable and flush

2022/5/10: Creation of PDF and adjustments of table columns widths. Referenced "Efficient Trace for RISC-V Version 2.0"

2021/12/13: Candidate for Frozen - compatible with E-Trace 1.1.3-Frozen

2021/3/02: Changes toward control share with E-Trace specifications

2020/6/21: Initial ADOC creared (from PDF)

Chapter 1. Introduction

This document presents a control interface for the Trace Encoder described in the *Efficient Trace for RISC-V Specification Version 2.0* and for the RISC-V N-Trace specification (in progress). Both Trace Working Groups agreed to standardize the control interface so that trace control software development tools can be used interchangeably with any RISC-V device implementing processor and/or data trace.

Instruction Trace is a system that collects a history of processor execution, along with other events. The trace system may be set up and controlled using a register-based interface. Hart execution activity appears on the Ingress Port and feeds into a Trace Encoder where it is compressed and formatted into trace messages. The Trace Encoder transmits trace messages to a Trace Sink. In multi-core systems, each hart has its own Trace Encoder, and typically all will connect to a Trace Funnel that aggregates trace data from multiple sources and sends the data to a single destination.

This specification does not define the hardware interconnection between the hart and Trace Encoder, as this is defined in the *Efficient Trace for RISC-V Specification Version 2.0*. This document also does not define the hardware interconnection between the Trace Encoder and Trace Funnel, or between the Trace Encoder/Funnel and Trace Sink.

This specification allows a wide range of implementations including low-gate-count minimal instruction trace and systems with only instrumentation trace. Implementation choices include whether to support branch trace, data trace, instrumentation trace, timestamps, external triggers, various trace sink types, and various optimization tradeoffs between gate count, features, and bandwidth requirements.

Chapter 2. Glossary

Trace Encoder - Hardware module that accepts execution information from a hart and generates a stream of trace messages/packets.

Trace Message/Packet - Depending on protocol different names can be used, but it means the same. It is considered as continous sequence of (usully bytes) describing program and/or data flow.

Trace Funnel - Hardware module that combines trace streams from multiple Trace Encoders into a single output stream.

Trace Sink - Hardware module that accepts a stream of trace messages and records them in memory or forwards them onward in some format.

Trace Decoder - Software program that takes a recorded trace (from Trace Sink) and produces readable execution history.

WARL - Write any, read legal. If a non-legal value is written, the writen value must be ignored and register will keep previous, legal value. Used by debugger to determine system capabilities. See Discovery chapter.

ATB - Advanced Trace Bus, a protocol described in ARM document IHI0032B.

PIB - Pin Interface Block, a parallel or serial off-chip trace port feeding into a trace probe.

Chapter 3. Trace System Overview

This section briefly describes features of the Trace Encoder and other trace components as background for understanding some of the control interface register fields.

3.1. Trace Encoder Types

By monitoring the Ingress Port, the Trace Encoder determines when a program flow discontinuity has occurred and whether the discontinuity is inferable or non-inferable. An inferable discontinuity is one for which the Trace Decoder can statically determine the destination, such as a direct branch instruction in which the destination or offset is included in the opcode. Non-inferable discontinuities include all other types as interrupt, exception, and indirect jump instructions.

3.2. Branch Trace Messaging

Branch Trace Messaging is the simplest form of instruction trace. Each program counter discontinuity results in one trace message, either a Direct or Indirect Branch Message. Linear instructions (or sequences of linear instrucions) do not result in any trace messages/packets.

Indirect Branch Messages normally contain a compressed address to reduce bandwidth. The Trace Encoder emits a Branch With Sync Message containing the complete instruction address under certain conditions. This message type is a variant of the Direct or Indirect Branch Message and includes a full address and a field indicating the reason for the Sync.

3.3. Branch History Messaging

Both the Efficient Trace for RISC-V (E-Trace) Specification and the Nexus standard define systems of messages intended to improve compression by reporting only whether conditional branches are taken by encoding each branch outcome in single taken/not-taken bit. The destinations of non-inferable jumps and calls are reported as compressed addresses. Much better compression can be achieved, but an Encoder implementation will typically require more hardware.

3.4. Other Optimizations

Several other optimizations are possible to improve trace compression. These are optional for any Trace Encoder and there should be a way to disable optimizations in case the trace system is used with code that does not follow recommended API rules. Examples of optimizations are a Return-address stack, Branch repetition, Statically-inferable jump, and Branch prediction.

3.5. Trace Sinks

The Trace Encoder transmits completed messages to a Trace Sink. This specification defines a number of different sink types, all optional, and allows an implementation to define other sink types. A Trace Encoder must have at least one sink attached to it.

Trace messages/packets are sequences of bytes. In case of wider sink width, some

padding/idle bytes (or additioanl formatting) may be added by particular sink. Nexus format allows any number of idle bytes between messages.

3.5.1. SRAM Sink

The Trace Encoder packs trace messages into fixed-width trace words (usually bytes). These are then stored in a dedicated RAM, typically located on-chip, in a circular-buffer fashion. When the RAM has filled, it may optionally be stopped, or it may wrap and overwrite earlier trace.

3.5.2. PIB Sink

The Trace Encoder sends trace messages to the PIB Sink. Each message is transmitted off-chip (as sequence of bytes) using a specific protocol described later.

3.5.3. System Bus (SBA) Memory Sink

The Trace Encoder packs trace messages into fixed-width trace words. These are then stored in a range of system memory reserved for trace using a DMA-type bus master in a circular-buffer fashion. When the memory range has been filled, it may optionally be stopped, or it may wrap and overwrite earlier trace. This type of sink may also be used to transmit trace off-chip through, for example, a PCIe or USB port.

3.5.4. Funnel Sink

The Trace Encoder sends trace messages to a Trace Funnel. The Funnel aggregates trace from each of its inputs and sends the combined trace stream to its designated Trace Sink, which is one or more of the sink types above.

It is assumed, that each input to funnel (trace encoder or another funnel) has unique message source ID defined (this is 'trTeSrcID' field in trTeControl register).

3.5.5. ATB Bridge

The ATB Bridge transmits bytes of RISC-V trace messages as an ATB bus master.

ATB has width, which is either 8 or 32-bit what will well match 'packet=sequence-of-bytes' definition.

Chapter 4. Control Interface

The Trace Control interface consists of a set of 32-bit registers. The control interface can be used to set up and control a trace session, retrieve collected trace, and control any trace system components.

4.1. Trace Components

This specification defines the following trace components:

Component Name	Component Type (value=symbol)	Base Address (symbol)	Description
Trace Encoder	0x1=TRCOMP_ENCODER	trBaseEncoder	Accepts execution information from a hart (via Trace Ingress Port) and generates a stream of RISC-V trace messages/packets.
Trace Funnel	0x8=TRCOMP_FUNNEL	trBaseFunnel	Acccepts several RISC-V trace message/packet streams (from Trace Encoders or Trace Funnels) and merges them into single stream of trace messages/packes.
Trace RAM Sink	0x9=TRCOMP_RAMSINK	trBaseRamSink	Accepts RISC-V trace messages/packets (from Trace Encoder or Trace Funnel) and stores them into memory buffer (either dedicated SRAM or System Bus).
Trace PIB Sink	0xA=TRCOMP_PIBSINK	trBasePibSink	Accepts RISC-V trace messages/packets (from Trace Encoder or Trace Funnel) and sends them via set of pins (parallel or serial).
Trace ATB Bridge	0xE=TRCOMP_ATBBRIDGE	trBaseAtbBridge	Accepts RISC-V trace messages/packets (from Trace Encoder or Trace Funnel) and forwards them to ATB bus in a system.

This specification is NOT addressing discovery of base addresses of trace components. These base addresses (trBase... symbols in above table) must be specified as part of trace tool configuration. Connections between different trace components must be also defined. Future version of this specification may allow single base address to be sufficient to access all components in the system.

Each Trace Component is controlled by set of 32-bit registers occupying up to a 4K-byte space. Base address of each trace component must be aligned on 4K boundary.

Each hart being traced must have its own separate Trace Encoder control component. This also applies to multiple harts that belong to the same core. A system with multiple harts must allow generating messages with a field indicating which hart is responsible for that message.

The Trace Funnel allows to combine several trace sources (from Trace Encoders or other Trace Funnels) into one trace stream.

The Trace Sink (connected to output from Trace Encoder or Trace Funnel) stores trace messages/packets to memory or sends them out of SoC.

4.2. Accessing Trace Control Registers

For the access method to the trace control registers, it makes a difference whether these registers shall be accessed by an external debug/trace tool, or by an internal debugger running on the chip.

Trace control register access by an external debugger (this is the most common use case):

- External debuggers must be able to access all trace control registers independent of whether the traced harts are running or halted. That is why for external debuggers, the recommended access method for memory-mapped control registers is memory accesses through the RISC-V debug module using SBA (System Bus Access) as defined in the RISC-V Debug Specification.

Trace control register access by an internal debugger:

- Through loads and stores performed by one or more harts in the system. Mapping the control interface into physical memory accessible from a hart allows that hart to manage a trace session independently from an external debugger. A hart may act as an internal debugger or may act in cooperation with an external debugger. Two possible use models are collecting crash information in the field and modifying trace collection parameters during execution. If a system has physical memory protection (PMP), a range can be configured to restrict access to the trace system from hart(s).

Additional control path(s) may also be implemented, such as extra JTAG registers or devices, a dedicated DMI debug bus or message-passing network. Such an access (which is NOT based on System Bus) may require custom implementation by trace probe vendors as this specification only mandates probe vendors to provide access via SBA commands.

4.3. Trace Component Register Map

Each block of 32-bit registers (for each component) has the following layout:

Address Offset	Register Name	Compliance	Description
0x000	tr??Control	Required	Main control register for this trace component
0x004	tr??Impl	Required	Trace Implementation information for this trace component

Address Offset	Register Name	Compliance	Description
0x008	tr??Control2	Optional	Additional controls for this trace component (can be named differently)
0x00C	tr??Control3	Optional	Additional controls for this trace component (can be named differently)
0x010 - 0xDFF	—	Optional	Additional registers (specific for particular type of component). All not used registers are reserved and should read as 0 and ignore writes.
0xE00 - 0xFFF	—	Optional	Registers reserved for implementation specific details. May allow identification of components on a system bus.

Each component has 'tr??Active' bit in 'tr??Control' register. Accesses to other registers are unpredictable when 'tr??Active' bit is 0.

Each trace component has tr??Impl register (at address offset 0x4) where trace component version and trace component type can be identified. This register allows debug tool to verify provided component base address and potentially adjust tool behaviour by looking at component version.

Each component may have different version. Initial version of this specificiation define all components to specify component version as 1.0 (major=1, minor=0).

Registers in the 4K range that are not implemented are reserved and read as 0 and ignore writes.

Most of trace control registers are optional. Some WARL fields may be hard-coded to any value (including 0). It allows different implementations to provide different functionality.

Both N-Trace and E-Trace encoders are controlled by the same set of bits/fields in same registers. As almost every register/field/bit is optional this provides good flexibility in implementation.

Trace Encoder Registers (trTe..., trTs...)

Address Offset	Register Name	Compliance	Description
0x000	trTeControl	Required	Trace Encoder control register
0x004	trTeImpl	Required	Trace Encoder implementation information
0x008	trTeInstFeatures	Optional	Extra instruction trace encoder features
0x00C	trTeInstFilters	Optional	Determine which filters qualify instruction trace
Data trace control (trTeData...)			
0x010	trTeDataControl	Optional	Data trace control and features
0x014 - 0x018	—	Reserved	Reserved for more data trace related registers

Address Offset	Register Name	Compliance	Description
0x01C	trTeDataFilters	Optional	Determine which filters qualify data trace
Reserved			
0x020 - 0x03F	—	Reserved	Reserved for more registers/sub-components
Timestamp control (trTs…)			
0x040	trTsControl	Optional	Timestamp control register
0x044	—	Optional	Reserved
0x048	trTsCounterLow	Optional	Lower 32 bits of timestamp counter
0x04C	trTsCounterHigh	Optional	Upper bits of timestamp counter
Trigger control (trTeTrig…)			
0x050	trTeTrigDbgControl	Optional	Debug Triggers control register
0x054	trTeTrigExtInControl	Optional	External Triggers Input control register
0x058	trTeTrigExtOutControl	Optional	External Triggers Output control register
Reserved			
0x060 - 0x3FF	—	Reserved	Reserved for more registers/sub-components
Filters & comparators (trTeFilter…, trTeComp…)			
0x400 - 0x5FF	trTeFilter…	Optional	Trace Encoder Filter Registers
0x600 - 0x7FF	trTeComp…	Optional	Trace Encoder Comparator Registers

Examples of possible additional sub-components in Trace Encoder are:

- PC Sampling
- Instrumentation Trace

Trace RAM Sink Registers (trRam…)

Address Offset	Register Name	Compliance	Description
0x000	trRamControl	Required	RAM Sink control register
0x004	trRamImpl	Required	RAM Sink Implementation information
0x008 - 0x00F	—	Reserved	Reserved for more control registers
0x010	trRamStartLow	Optional	Lower 32 bits of start address of circular trace buffer

Address Offset	Register Name	Compliance	Description
0x014	trRamStartHigh	Optional	Upper bits of start address of circular trace buffer
0x018	trRamLimitLow	Optional	Lower 32 bits of end address of circular trace buffer
0x01C	trRamLimitHigh	Optional	Upper bits of end address of circular trace buffer
0x020	trRamWPLow	Optional	Lower 32 bits of current write location for trace data in circular buffer
0x024	trRamWPHigh	Optional	Upper bits of current write location for trace data in circular buffer
0x028	trRamRPLow	Optional	Lower 32 bits of access pointer for trace readback
0x02C	trRamRPHigh	Optional	Upper bits of access pointer for trace readback
0x040	trRamData	Optional	Read/write access to SRAM trace memory (32-bit data)
0x044 - 0x07F	—	Optional	Reserved for bigger read buffer

Trace PIB Sink Registers (trPib...)

Address Offset	Register Name	Compliance	Description
0x000	trPibControl	Required	Trace PIB Sink control register
0x004	trPibImpl	Required	Trace PIB Sink Implementation information

Trace Funnel Registers (trFunnel..., trTs)

Address Offset	Register Name	Compliance	Description
0x000	trFunnelControl	Required	Trace Funnel control register
0x004	trFunnelImpl	Required	Trace Funnel Implementation information
Timestamp control (trTs...)			
0x040	trTsControl	Optional	Timestamp control register
0x044	—	Reserved	Reserved for extra timestamp control
0x048	trTsCounterLow	Optional	Lower 32 bits of timestamp counter
0x04C	trTsCounterHigh	Optional	Upper bits of timestamp counter

Trace ATB Bridge Registers (trAtbBridge...)

Address Offset	Register Name	Compliance	Description
0x000	trAtbBridgeControl	Required	Trace ATB Bridge control register
0x004	trAtbBridgeImpl	Required	Trace ATB Bridge Implementation information

Chapter 5. Versioning of Components

Each component has a tr??Impl register, which includes two 4-bit 'tr??VerMinor' and 'tr??VerMajor' fields. These fields are guaranteed to be present in all future revisions of a standard, so trace tools will be able to discover a component version and act accordingly.

- Values 0 as 'tr??VerMajor' is allowed (due to compatibility reasons).
- Different components may report different versions (as some components may be updated more often than others).
- The major version 'tr??VerMajor' field should change, when some incompatible (which will break older trace software ...) change is introduced.
- The minor version 'tr??VerMinor' field should change, when change is considered a compatible extension (for example adding a new field) - from that reason software should always write 0 to undefined bits in registers.
- Version 15.x is reserved for non-compatible version encoding.
- Version n.15 should be used as experimental (in development) implementation.

Versions should be always reported as two decimal numbers *'major.minor'* - initial version of this specification is ***1.0***.

Trace software should handle versions as follows (let's assume hypothetical version 2.3 was defined as current version in moment of release of trace software)

- 0.x ⇒ Reject as not supported or generate a warning and handle as legacy version 0.
- 2.3 ⇒ Accept silently.
- 2.2 ⇒ Accept silently (and trim features or not allow user to set newer features).
- 2.4 ⇒ Generate a warning but continue using 2.3 features.
- 2.15 ⇒ Generate an "experimental version" warning but continue using 2.3 features.
- 1.x ⇒ Generate a warning and continue or reject as an obsolete (referring to last debugger supporting this version).
- 3.x ⇒ Abort with an error that this future version is not compatible with existing software and possibly redirect to the tool update page.

Displayed messages should report component name, component base address and current and supported version numbers. It is suggested to display the full hexadecimal value of 'tr??Impl' register as it may aid in debugging of possibly incorrect/incompatible component configuration.

Chapter 6. Trace Encoder Control Interface

Many features of the Trace Encoder are optional. In most cases, optional features are enabled using a WARL (write any, read legal) register field. A debugger can determine if an optional feature is present by writing to the register field and reading back the result.

Register: trBaseEncoder+0x000 trTeControl: Trace Encoder Control Register (Required)

Bit	Field	Description	RW	Reset
0	trTeActive	Primary enable for the TE. When 0, the TE may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written.	RW	0
1	trTeEnable	1: Trace Encode is enabled. Allows trTeInstTracing and trTeDataTracing to turn tracing on and off. Setting trTeEnable to 0 flushes any queued trace data to sink or funnel attached to this encoder. This bit can be set to 1 only by direct write to it.	RW	0
2	trTeInstTracing	1: Instruction trace is being generated. Written from tool or controlled by triggers. When trTeInstTracing=1, instruction trace data may be subject to additional filtering in some implementations (additional trTeInstMode settings).	RW	0
3	trTeEmpty	Reads as 1 when all generated trace has been emitted.	R	1

Bit	Field	Description	RW	Reset
6-4	trTeInstMode	Main instruction trace generation mode 0: Full Instruction trace is disabled, but trace may still emit some records. 1-2: Reserved for subsets of Branch Trace (for example periodic PC sampling) 3: Generate instruction trace using Branch Trace (each taken branch generate trace) 4-5: Reserved for subset of Branch History Trace 6: Generate non-optimized instruction Branch History Trace (each branch adds single history bit) 7: Generate optimized Instruction Trace (trTeInstFeatures register if present define instruction trace features and optimizations).	WARL	SD[(1)]
8-7	—	Reserved for future modes	—	0
9	trTeContext	Send Ownership messages to indicate processor context when scontext, mcontext, v, or prv changes and full context information immediately after all Sync messages.	WARL	SD
10	—	Reserved	WARL	SD
11	trTeInstTrigEnable	1: Allows trTeInstTracing to be set or cleared by Trace-on and Trace-off Debug module or External triggers respectively	WARL	0
12	trTeInstStallOrOverflow	Written to 1 by hardware when an overflow message is generated or when the TE requests a hart stall. Clears to 0 at TE reset or when trace is enabled (trTeEnable set to 1).	R	0
13	trTeInstStallEn	0: If TE cannot send a message, an overflow is generated when trace is restarted. 1: If TE cannot send a message, the hart is stalled until it can.	WARL	SD
14	—	Reserved	—	0

Bit	Field	Description	RW	Reset
15	trTeInhibitSrc	1: Disable source field in trace messages. Unless disabled, a trace source field (of trTeInstFeatures.trTeSrcBits) is added to every trace message to indicate which TE generated each message. If trTeInstFeatures.trTeSrcBits is 0, this bit is not active.	WARL	SD
17-16	trTeSyncMode	Select periodic synchronization mechanism. At least one non-zero mechanism must be implemented. 0: Off 1: Count trace messages/packets 2: Count hart clock cycles 3: Count instruction half-words (16-bit)	WARL	SD
19-18	—	Reserved	—	0
23-20	trTeSyncMax	The maximum interval (in units determined by trTeSyncMode) between synchronization messages/packets. Generate synchronization when count reaches 2^(trTeSyncMax + 4). If synchronization packet is generated from another reason internal counter should be reset.	WARL	SD
26-24	trTeFormat	Trace recording format 0: Format defined by Efficient Trace for RISC-V (E-Trace) Specification 1: Nexus messages with 6 MDO + 2 MSEO bits 2-6: Reserved for future formats 7: Vendor-specific format	WARL	SD
31-28	—	Reserved	—	0

> SD[(1)] = System-Dependent, but these fields should always have same values at reset (trTeActive=0)

Register: trBaseEncoder+0x004 trTeImpl: Trace Encoder Implementation Register (Required)

Bit	Field	Description	RW	Reset
3-0	trTeVerMajor	Trace Encoder Major Version. Value 1 means component is compliant with this document. Value 0 means legacy version - see 'Legacy Interface Version' chapter at the end.	R	1
7-4	trTeVerMinor	Trace Encoder Minor Version. Value 0 means component is compliant with this document.	R	0
11-8	trTeCompType	Trace Encoder Component Type (Trace Encoder)	R	0x1
23-12	—	Reserved for future versions of this standard	—	0
31-24	—	Reserved for vendor specific implementation details	—	SD

Register: trBaseEncoder+0x008 trTeInstFeatures: Trace Instruction Features Register

Bit	Field	Description	RW	Reset
0	trTeInstNoAddrDiff	Do not send differential addresses when set (always full address is sent)	WARL	0
1	trTeInstNoTrapAddr	When set, do not sent trap handler address in trap packets	WARL	0
2	trTeInstEnSequentialJump	Treat sequentially inferrable jumps as inferable PC discontinuities when set.	WARL	0
3	trTeInstEnCallStack	Treat returns as inferable PC discontinuities when returning from recent call on stack.	WARL	0
4	trTeInstEnBranchPrediction	Branch predictor enabled when set.	WARL	0
5	trTeInstEnJumpTargetCache	Jump target cache enabled when set.	WARL	0
15-6	—	Reserved for additional instruction trace control/status bits	—	0
27-16	trTeSrcID	This TE’s source ID. If trTeSrcBits>0 and trace source is not disabled by trTeInhibitSrc, then trace messages from this TE will all include a trace source field of trTeSrcBits bits and all messages from this TE will use this value as trace source field. May be fixed or variable.	WARL	SD
31-28	trTeSrcBits	The number of bits in the trace source field (0..12), unless disabled by trTeInhibitSrc. May be fixed or variable.	WARL	SD

Register: trBaseEncoder+0x00C trTeInstFilters: Trace Instruction Filters Register

Bit	Field	Description	RW	Reset
15-0	trTeInstFilters ***n***	Determine which filters qualify instruction trace. If bit ***n*** is a 1 then instructions will be traced when filter ***n*** matches. If all bits are 0, all instructions are traced.	WARL	0

Register: trBaseEncoder+0x010 trDataControl: Data Trace Control Register (for encoders supporting data trace)

Bit	Field	Description	RW	Reset
0	trTeDataImplemented	Read as 1 if data trace is implemented.	R	SD
1	trTeDataTracing	1=Data trace is being generated. Written from tool or controlled by triggers. When trDataTracing=1, data trace may be subject to additional filtering in some implementations.	WARL	SD
2	trTeDataTrigEnable	Global enable/disable for data trace triggers	WARL	0
3	trTeDataStallDelta	Set to 1 if data trace caused stall since last read. It is clear on read.	R	0
4	trTeDataStallEnable	Stall execution if data trace message cannot be generated.	WARL	0
5	trTeDataDropDelta	Set to 1 if data trace was dropeed since last read. It is clear on read.	R	0
6	trTeDataDropEnable	Allow dropping data trace to avoid instruction trace overflows. Seting this bit will not guarantee that instuction trace overflows will not happen.	WARL	0
15-7	—	Reserved for additional data trace control/status bits.	—	0
16	trTeDataNoValue	Omit data values from data trace packets when set.	WARL	SD
18-17	trTeDataAddressMode	'00':Omit data address from data trace packets. '01': Compress data addresses in XOR mode (only LSB bits changed), '10': Compress data addresses in differental mode (+-N offset),'11': reserved or automatic mode.	WARL	SD

Register: trBaseEncoder+0x01C trTeDataFilters: Trace Data Filters Register

Bit	Field	Description	RW	Reset
15-0	trTeDataFilters ***n***	Determine which filters qualify data trace. If bit ***n*** is a 1 then data accessed will be traced when filter ***n*** matches. If all bits are 0, all data accesses are traced.	WARL	0

Chapter 7. Timestamp Unit

Timestamp Unit is an optional sub-component present in either Trace Encode or Trace Funnel. An implementation may choose from several types of timestamp units:

- **Internal System** - fixed clock in a system (such as bus clock) is used to increment the timestamp counter
- **Internal Core** - core clock is used to increment the timestamp counter (only applicable to Trace Encoders)
- **Shared** - shares timestamp with another Trace Encoder or Trace Funnel
- **External** - accepts a binary timestamp value from an outside source such as ARM CoreSight™ trace

Implementations may have no timestamp, one timestamp type, or more than one type. The WARL field trTsType is used to determine the system capability and to set the desired type.

The width of the timestamp is implementation-dependent, typically 40 or 48 bits (40 bit timestamp will overflow every 4.7 minutes assuming 1GHz timestamp clock).

In a system with Funnels, typically all the Funnels are built with a Timestamp Unit. The top-level Funnel is the source of the timestamp (Internal System or External) and all the Encoders and other Funnels have Shared timestamp. To perform the forwarding function, the mid-level Funnels must be programmed with trFunnelActive=1 (which is natural as all trace messages must pass through that funnel).

An Internal Timestamp Unit may include a prescale divider, which can extend the range of a narrower timestamp and uses less power but has less resolution.

In a system with an Internal Core timestamp counter (implemented in Trace Encoder associated with a hart), it may be desirable to stop the counter when the hart is halted by a debugger. An optional control bit is provided for this purpose, but it may or may not be implemented.

7.1. Timestamp Registers

Register: trBaseEncoder/Funnel+0x040 trTsControl: Timestamp Control Register (Optional)

Bit	Field	Description	RW	Reset
0	trTsActive	Primary reset/enable for timestamp unit	RW	0
1	trTsCount	Internal timestamp only. 1=counter runs, 0=counter stopped	RW	0
2	trTsReset	Internal timestamp only. Write 1 to reset the timestamp counter	W1	0
3	trTsRunInDebug	Internal Core timestamp only. 1=counter runs when hart is halted (in debug mode), 0=stopped	WARL	0

Bit	Field	Description	RW	Reset
6-4	trTsType	Type of Timestamp unit 0: None 1: External 2: Internal System 3: Internal Core 4: Shared 5-7: Vendor-specific type	WARL	SD
7	—	Reserved	—	0
9-8	trTsPrescale	Internal timestamp only. Prescale timestamp clock by 2^2n (1, 4, 16, 64).	WARL	0
14-10	—	Reserved	—	0
15	trTsEnable	Global enable for timestamp field in trace messages/packets (for Trace Encoder only).	WARL	0
23-16		System-dependent fields to control what message/packet types include timestamp fields.	WARL	0
29-24	trTsWidth	Width of timestamp in bits (0..63)	R	SD

Register: trBaseEncoder/Funnel+0x048 trTsCounterLow: Timestamp Lower Bits (Optional)

Bit	Field	Description	RW	Reset
31-0	trTsCounterLow	Lower 32 bits of timestamp counter.	R	0

Register: trBaseEncoder/Funnel+0x04C trTsCounterHigh: Timestamp Upper Bits (Optional)

Bit	Field	Description	RW	Reset
31-0	trTsCounterHigh	Upper bits of timestamp counter, zero-extended.	R	0

Chapter 8. Trace Encoder Triggers

8.1. Debug Module Triggers

Debug triggers are signals from the hart that a trigger (breakpoint or watchpoint) was hit, but the action associated with that trigger is a trace-related action. Action identifiers 2-5 are reserved for trace actions in the RISC-V Debug Spec, where triggers are defined. Actions 2-4 are defined by the Efficient Trace for RISC-V (E-Trace) Specification. The desired action is written to the action field of the Match Control mcontrol CSR (0x7a1). Not all harts support trace actions; the debugger should read back mcontrol CSR after setting one of these actions to verify that the option exists.

Action (from debug spec)	Effect
0	Breakpoint exception
1	Debug exception
2	Trace-on action. When trTeInstTrigEnable=1 it will start instruction tracing (trTeInstTracing → 1). When trTeDataTrigEnable=1 it will start data tracing (trTeDataTracing → 1).
3	Trace-off action. When trTeInstTrigEnable=1 it will stop instruction tracing (trTeInstTracing → 0). When trTeDataTrigEnable=1 it will stop data tracing (trTeDataTracing → 0).
4	Trace-notify action. It will cause encoder to generate packet with current PC (and possibly timestamp). If trace is not active (trTeInstTracing = 0) it should be ignored.
5	Optional vendor-specific action

If there are vendor-specific features that require control, the trtrTeTrigDbgControl register is used.

Register: trBaseEncoder+0x050 trTeTrigDbgControl: Debug Trigger Control Register

Bit	Field	Description	RW	Reset
31-0	trTeTrigDbgControl	Vendor-specific trigger setup	WARL	0

8.2. External Triggers

The TE may be configured with up to 8 external trigger inputs for controlling trace. These are in addition to the external triggers present in the Debug Module when Halt Groups are implemented. The specific hardware signals comprising an external trigger are implementation-dependent.

External Trigger Outputs may also be present. A trigger out may be generated by trace starting, trace stopping, a watchpoint, or by other system-specific events.

Register: trBaseEncoder+0x054 trTeTrigExtInControl: External Trigger Input Control Register (Optional)

Bit	Field	Description	RW	Reset
3-0	trTeTrigExtInAction0	Select action to perform when external trigger input 0 fires. If external trigger input 0 does not exist, then its action is fixed at 0. 0: No action 1: Reserved 2: Trace-on action. When trTeInstTrigEnable=1 it will start instruction tracing (trTeInstTracing → 1). When trTeDataTrigEnable=1 it will start data tracing (trTeDataTracing → 1). 3: Trace-off action. When trTeInstTrigEnable=1 it will stop instruction tracing (trTeInstTracing → 0). When trTeDataTrigEnable=1 it will stop data tracing (trTeDataTracing → 0). 4: Trace-notify action. It will cause encoder to generate packet with current PC (and possibly timestamp). If trace is not active (trTeInstTracing = 0) it should be ignored. 5-15: Reserved	WARL	0
31-4	trTeTrigExtInAction ***n***	Select actions for external trigger input ***n*** (1..7). If an external trigger input does not exist, then its action is fixed at 0.	WARL	0

Register: trBaseEncoder+0x058 trTeTrigExtOutControl: External Trigger Output Control Register (Optional)

Bit	Field	Description	RW	Reset
3-0	trTeTrigExtOutEvent0	Bitmap to select which event(s) cause external trigger 0 output to fire. If external trigger output 0 does not exist, then all bits are fixed at 0. Bits 2 and 3 may be fixed at 0 if the corresponding feature is not implemented. 0: Start trace transition (trTeInstTracing 0 → 1) will fire the trigger. 1: Stop trace transition (trTeInstTracing 1 → 0) will fire the trigger. 2-3: Vendor-specific event (optional)	WARL	0

Bit	Field	Description	RW	Reset
31-4	trTeTrigExtOutEvent ***n***	Select events for external trigger output ***n*** (1..7). If an external trigger output does not exist, then its event bits are fixed at 0	WARL	0

8.3. Triggers Precedence

It is implementation-dependent what happens when debug triggers or external triggers with conflicting actions occur simultaneously or if debug triggers or external triggers occur too frequently to process.

Chapter 9. Trace Encoder Filter Registers

All registers with offsets 0x400 .. 0x7FC are reserved for additional trace encoder filtering options (context, addresses, modes etc.).

Specifications for different trace encoders should define registers in this range.

N-Trace: Only Debug Trigger based filtering is defined in this version.

E-Trace: Additional trace filtering as defined by register map defined below. Provision is made for upto 16 filters and 8 comparators, indexed by *i* and *j* respectively (*i* is in the range 0 - 15; *j* is in the range 0 - 7).

Address Offset	Register Name	Compliance	Description
0x400 + 0x20*i*	trTeFilter*i*Control	Optional	Filter *i* control
0x404 + 0x20*i*	trTeFilter*i*MatchInst	Optional	Filter *i* instruction match control
0x408 + 0x20*i*	trTeFilter*i*MatchEcause	Optional	Filter *i* Ecause match control
0x40C + 0x20*i*	—	Optional	Reserved
0x410 + 0x20*i*	trTeFilter*i*MatchValueImpdef	Optional	Filter *i* impdef value
0x414 + 0x20*i*	trTeFilter*i*MatchMaskImpdef	Optional	Filter *i* impdef mask
0x418 + 0x20*i*	trTeFilter*i*MatchData	Optional	Filter *i* Data trace match control
0x41C + 0x20*i*	—	Optional	Reserved
0x600 + 0x20*j*	trTeComp*j*Control	Optional	Comparator *j* control
0x604 + 0x20*j*	—	Optional	Reserved
0x608 + 0x20*j*	—	Optional	Reserved
0x60c + 0x20*j*	—	Optional	Reserved
0x610 + 0x20*j*	trTeComp*j*PmatchLow	Optional	Comparator *j* primary match (bits 31:0)

Address Offset	Register Name	Compliance	Description
0x614 + 0x20*j*	trTeComp*j*PmatchHigh	Optional	Comparator *j* primary match (bits 63:32)
0x618 + 0x20*j*	trTeComp*j*SmatchLow	Optional	Comparator *j* secondary match (bits 31:0)
0x61C + 0x20*j*	trTeComp*j*SmatchHigh	Optional	Comparator *j* secondary match (bits 63:32)

Register: trBaseEncoder+0x400 + 0x20*i* trTeFilter*i*Control : Filter *i* Control Register

Bit	Field	Description	RW	Reset
0	trTeFilterEnable	Overall filter enable	WARL	0
1	trTeFilterMatchPrivilege	When set, match privilege levels specified by trTeFilter*i*MatchInstControl.teMatchChoicePrivilege.	WARL	0
2	trTeFilterMatchEcause	When set, start matching from exception cause codes specified by trTeFilter *i*MatchChoiceEcause.teMatchChoiceEcause, and stop matching upon return from the 1st matching exception.	WARL	0
3	trTeFilterMatchInt	When set, start matching from a trap with the interrupt level codes specified by trTeFilter *i*MatchInstControl.teMatchValueInterrupt, and stop matching upon return from the 1st matching trap.	WARL	0
4	trTeFilterMatchComp1	When set, the output of the comparator selected by trTeFilterMatchComp1 must be high in order for the filter to match.	WARL	0
7-5	trTeFilterComp1	Specifies the comparator unit to use for the 1st comparison.	WARL	SD
8	trTeFilterMatchComp2	When set, the output of the comparator selected by trTeFilterMatchComp2 must be high in order for the filter to match.	WARL	0
11-9	trTeFilterComp2	Specifies the comparator unit to use for the 2nd comparison.	WARL	SD
12	trTeFilterMatchComp3	When set, the output of the comparator selected by trTeFilterMatchComp3 must be high in order for the filter to match.	WARL	0
15-13	trTeFilterComp3	Specifies the comparator unit to use for the 3rd comparison.	WARL	SD

Bit	Field	Description	RW	Reset
16	trTeFilterMatchImpdef	When set, match **impdef** values as specified by trTeFilter*i*MatchValueImpdef.teMatchValueImpdef and trTeFilter *i*MatchValueImpdef.teMatchMaskImpdef.	WARL	0
23-17	—	Reserved	—	0
24	trTeFilterMatchDtype	When set, match **dtype** values as specified by trTeFilter*i*MatchDataControl.teMatchChoiceDtype.	WARL	0
25	trTeFilterMatchDsize	When set, match **dsize** values as specified by trTeFilter*i*MatchDataControl.teMatchChoiceDsize.	WARL	0

Register: trBaseEncoder+0x404 + 0x20*i* trTeFilter*i*MatchInst : Filter *i* Instruction Match Control Register

Bit	Field	Description	RW	Reset
7-0	trTeFilterMatchChoice Privilege	When trTeFilter *i*Control.trTeFilterMatchPrivilege is set, match all privilege levels for which the corresponding bit is set. For example, if bit N is 1, then match if the **priv** value is N	WARL	SD
8	trTeFilterMatchValue Interrupt	When trTeFilter *i*Control.trTeFilterMatchInterrupt is set, match **itype** of 2 or 1 depending on whether this bit is 1 or 0 respectively.	WARL	SD

Register: trBaseEncoder+0x408 + 0x20*i* trTeFilter*i*MatchEcause : Filter *i* Ecause Match Control Register

Bit	Field	Description	RW	Reset
31-0	trTeFilterMatchChoice Ecause	When trTeFilter*i*Control.trTeFilterMatchEcause is set, match all excepion causes for which the corresponding bit is set. For example, if bit N is 1, then match if the **ecause** is N.	WARL	SD

Register: trBaseEncoder+0x410 + 0x20*i* trTeFilter*i*MatchValueImpdef : Filter *i* Impdef Match Value Register

Bit	Field	Description	RW	Reset
31-0	trTeFilterMatchValue Impdef	When trTeFilter*i*Control.trTeFilterMatchimpdef is set, match if (**impdef** & trTeFilterMatchMaskImpdef) == (trTeFilterMatchValueImpdef & trTeFilterMatchMaskImpdef.	WARL	SD

Register: trBaseEncoder+0x414 + 0x20*i* trTeFilter*i*MatchMaskImpdef : Filter *i* Impdef Match Mask Register

Bit	Field	Description	RW	Reset
31-0	trTeFilterMatchMaskImpdef	When trTeFilter*i*Control.trTeFilterMatchimpdef is set, match if (**impdef** & trTeFilterMatchMaskImpdef) == (trTeFilterMatchValueImpdef & trTeFilterMatchMaskImpdef.	WARL	SD

Register: trBaseEncoder+0x418 + 0x20*i* trTeFilter*i*MatchData : Filter *i* Data Match Control Register

Bit	Field	Description	RW	Reset
15-0	trTeFilterMatchChoiceDtype	When trTeFilter*i*Control.trTeFilterMatchDtype is set, match all data access types for which the corresponding bit is set. For example, if bit N is 1, then match if the **dtype** value is N.	WARL	SD
23-16	trTeFilterMatchChoiceDsize	When trTeFilter*i*Control.trTeFilterMatchDsize is set, match all data access sizes for which the corresponding bit is set. For example, if bit N is 1, then match if the **dsize** value is N.	WARL	SD

Register: trBaseEncoder+0x600 + 0x20*j* trTeComp*j*Control : comparator *j* Control Register

Bit	Field	Description	RW	Reset
1-0	trTeCompPInput	Determines which input to compare against the primary comparator. 0: **iaddr** 1: **context** 2: **tval** 3: **daddr**	WARL	SD

Bit	Field	Description	RW	Reset
3-2	trTeCompSInput	Determines which input to compare against the secondary comparator. Same encoding as trTeCompPInput.	WARL	SD
6-4	trTeCompPFunction	Selects the primary comparator function. Primary result is true if input selected via trTeCompPInput is: 0: equal to trTeCompPMatch 1: not equal to trTeCompPMatch 2: less than to trTeCompPMatch 3: less than or equal to trTeCompPMatch 4: greater than to trTeCompPMatch 5: greater than or equal to trTeCompPMatch 6: Result always false (input ignored). Prime latch to 1 if trTeCompMatchMode is 3 7: Result always true (input ignored)	WARL	SD
7	—	Reserved	—	0
10-8	trTeCompSFunction	Selects the secondary comparator function. Secondary result is true if input selected via trTeCompSInput is: 0: equal to trTeCompSMatch 1: not equal to trTeCompSMatch 2: less than to trTeCompSMatch 3: less than or equal to trTeCompSMatch 4: greater than to trTeCompSMatch 5: greater than or equal to trTeCompSMatch 6: Result always true (input ignored). Use trTeCompSMatch as a mask for trTeCompPMatch 7: Result always true (input ignored)	WARL	SD

Bit	Field	Description	RW	Reset
11	—	Reserved	—	0
13-12	trTeCompMatchMode	Selects the match condition used to assert the overal comparator output 0: primary result true 1: primary and secondary result both true: (P && S) 2: Either primary or secondary result does not match: !(P && S) 3: Set when primary result is true and continue to assert until instruction after secondary result is true	WARL	SD
14	trTeCompPNotify	Generate a trace packet explicitly reporting the address of the final instruction in a block that causes a primary match (requires trTeCompPInput to be 0). This is also known as a watchpoint.	WARL	SD
15	trTeCompSNotify	Generate a trace packet explicitly reporting the address of the final instruction in a block that causes a secondary match (requires trTeCompSInput to be 0). This is also known as a watchpoint.	WARL	SD

Register: trBaseEncoder+0x610 + 0x20*j* trTeComp*j*PMatchLow : comparator *j* Primary match (low) Register

Bit	Field	Description	RW	Reset
31-0	trTeCompPMatchLow	The match value for the primary comparator (bits 31:0).	WARL	SD

Register: trBaseEncoder+0x614 + 0x20*j* trTeComp*j*PMatchHigh : comparator *j* Primary match (high) Register

Bit	Field	Description	RW	Reset
31-0	trTeCompPMatchHigh	The match value for the primary comparator (bits 63:32).	WARL	SD

Register: trBaseEncoder+0x618 + 0x20*j* trTeComp*j*SMatchLow : comparator *j* Secondary match (low) Register

Bit	Field	Description	RW	Reset
31-0	trTeCompSMatchLow	The match value for the secondary comparator (bits 31:0).	WARL	SD

Register: trBaseEncoder+0x61C + 0x20*j* trTeComp*j*SMatchHigh : comparator *j* Secondary match (high) Register

Bit	Field	Description	RW	Reset
31-0	trTeCompSMatchHigh	The match value for the secondary comparator (bits 63:32).	WARL	SD

Chapter 10. Trace RAM Sink

Trace RAM Sink may be instantiated or configured to support storing trace into dedicated SRAM or system SBA RAM. SRAM mode is using dedicated memory, while SBA mode is accessign memory via system bus (care should be taken to not overwrite application code or data - it is usually done by reserving part of system memory for trace). Dedicated SRAM memory must be read via dedicated 'trRamData' register, while memory in SBA (System Bus Access) mode should be read as any other memory on system bus.

Trace data is placed in memory in LSB order (first byte of trace packet/data is placed on LSB). For N-trace packets, MSEO bits are placed on LSB bits of each byte.

Be aware that in case trace memory wraps around some protocols may require additional synchronization data - it is usually done by periodically generating sequence of bytes which cannot be part of any valid packet. N-Trace protocol does not require it as it is self-synchronizing - last byte of each message/packet is specially marked.

Register: trBaseRam+0x000 trRamControl: Trace RAM Sink Control Register

Bit	Field	Description	RW	Reset
0	trRamActive	Primary enable for Trace RAM Sink. When 0, the Trace RAM Sink may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written.	RW	0
1	trRamEnable	1=Trace RAM Sink enabled. Setting trRamEnable to 0 flushes any queued trace data to output.	RW	0
2	—	Reserved	—	0
3	trRamEmpty	Reads 1 when Trace RAM Sink internal buffers are empty	R	1
4	trRamModeSBA	0: This RAM Sink will operate in SRAM mode, 1: This RAM Sink will operate in SBA mode	WARL	SD
7-5	—	Reserved	—	0
8	trRamStopOnWrap	1: Disable storing trace to RAM (trRamEnable → 0) when circular buffer gets full.	WARL	0

Register: trBaseRamSink+0x004 trRamImpl: Trace RAM Sink Implementation Register

Bit	Field	Description	RW	Reset
3-0	trRamVerMajor	Trace RAM Sink Component Major Version. Value 1 means component is compliant with this document.	R	1
7-4	trRamVerMinor	Trace RAM Sink Component Minor Version. Value 0 means component is compliant with this document.	R	0
11-8	trRamCompType	Trace RAM Sink Component Type (RAM Sink)	R	0x9
12	trRamHasSRAM	This RAM Sink supports SRAM mode	R	SD
13	trRamHasSBA	This RAM Sink supports SBA mode	R	SD
23-14	—	Reserved for future versions of this standard	—	0
31-24	—	Reserved for vendor specific implementation details	—	SD

Single RAM Sink may support both SRAM and SBA modes, but not both of them may be enabled in the same time. It is also possible to have more than one RAM Sink in a system.

Register: trBaseRamSink+0x010 trRamStartLow: Trace RAM Sink Start Register

For busses with address larger than 32-bit, corresponding 'High' register define MSB part of larger address.

FUTURE: Another extension should deal with signalling (and clearing ...) RAM access errors (especially important for System Bus). Maybe we should have a bit in 'WP' register (where we have 'trRamWrap' already) as this register must be read by decoder anyway.

Bit	Field	Description	RW	Reset
1-0	—	Always 0 (two LSB of 32-bit address)	R	0
31-2	trRamStartLow	Byte address of start of trace sink circular buffer. It is always aligned on 32-bit/4-byte boundary. This register may not be implemented if the sink type doesn't require an address. An SRAM sink will usually have trRamStartLow fixed at 0.	WARL	Undef or fixed to 0

Register: trBaseRamSink+0x014 trRamStartHigh: Trace RAM Sink Start High Bits Register (Optional)

Bit	Field	Description	RW	Reset
31-0	trRamStartHigh	High order bits (63:32) of trRamStart registeer.	WARL	Undef

Register: trBaseRamSink+0x018 trRamLimitLow: Trace RAM Sink Limit Register

Bit	Field	Description	RW	Reset
1-0	—	Always 0 (two LSB of 32-bit address)	R	0
31-2	trRamLimitLow	Highest absolute 32-bit part of address of trace circular buffer. The trRamWP register is reset to trRamStart after a trace word has been written to this address. This register may not be present if the sink type doesn't require a limit address.	WARL	Undef

Register: trBaseRamSink+0x01C trRamLimitHigh: Trace RAM Sink Limit High Bits Register (Optional)

Bit	Field	Description	RW	Reset
31-0	trRamStartHigh	High order bits (63:32) of trRamLimit register.	WARL	Undef

Register: trBaseRamSink+0x020 trRamWPLow: Trace RAM Sink Write Pointer Register

Bit	Field	Description	RW	Reset
0	trRamWrap	Set to 1 by hardware when trRamWP wraps. It is only set to 0 if trRamWPLow is written	WARL	0
1	—	Always 0 (bit B1 of 32-bit address)	R	0
32-2	trRamWPLow	Absolute 32-bit part of address in trace sink memory where next trace message will be written. Fixed to natural boundary. After a trace word write occurs while trRamWP=trRamLimit, trRamWP is set to trRamStart. This register may not be present if no sinks require it.	WARL	Undef

Register: trBaseRamSink+0x024 trRamWPHigh: Trace RAM Sink Write Pointer High Bits Register (Optional)

Bit	Field	Description	RW	Reset
31-0	trRamWPHigh	High order bits (63:32) of trRamWP register.	WARL	Undef

Register: trBaseRamSink+0x028 trRamRPLow: Trace RAM Sink Read Pointer Register (Optional)

Bit	Field	Description	RW	Reset
N-2	trRamRPLow	Absolute 32-bit part of address in trace circular memory buffer visible through trRamData. Auto-increments (with wrap around) following an access to trRamData. Required for SRAM sink and optional for all other sink types.	WARL	0

Register: trBaseRamSink+0x02C trRamRPHigh: Trace RAM Sink Read Pointer High Bits Register (Optional)

Bit	Field	Description	RW	Reset
31-0	trRamRPHigh	High order bits (63:32) of trRamRP register.	WARL	Undef

Register: trBaseRamSink+0x040 trRamData: Trace RAM Sink Data Register (Optional)

Bit	Field	Description	RW	Reset
31-0	trRamData	Read (and optional write) value for trace sink memory access. SRAM is always accessed by 32-bit words through this path regardless of the actual width of the sink memory. Required for SRAM Sink and optional for other sink types.	R or RW	SD

When trace capture was wrapped around (trRamWrap=1) beginning of trace is not available and oldest packets/messages in trace buffer (starting at 'trRamWP') will be most likely not complete. Trace decoder must look for start of message. Also when trace is stopped on wrap aroud, last message recorded in trace memory may not be complete.

Implementations when trace buffer in system memory will be bigger than 4GB is desired will be unlikely.

FUTURE: Add 64-bit extensions as 32 MSB bits of size (reading 3 times is needed to be certain about 64-bit value). In order to relieve trace software to always read 3 times, there should be a field/bit saying if RAM size over 32-bit is implemented. It may be also WARL field, which must be set to '1' in order to allow 64-bit size. In most cases, it will never be settable (as 4GB of RAM for trace is rare requirement)

10.1. Typical Trace RAM Sink Configurations

Table below shows typical Trace RAM Sink configurations. Implementing other configurations if not suggested as trace tools may not support it without adjustments.

Mode	trRamStart	trRamLimit	trRamWP	trRamRP	trRamData
RAM	0	Hard coded to max size at reset, but can be possibly trimmed	Required	Required	Required
SBA Generic	Any (2^N aligned)	Any (trRamStart + 2^M - A) - must be set by trace tool	Required	—	—
SBA Fixed	Fixed (2^N aligned)	Fixed to max size at reset (trRamStart + 2^N - A), but can be possibly trimmed	Required	—	—

Value 'A' means aligment which depends on memory access width. If we have memory access width of 32-bits, A=4 and value of 'Limit' register should be 0x...FC. Some implementations may impose bigger alignment of trace (to allow more efficient transfer rates).

Chapter 11. Trace Funnel

The Trace Funnel combines messages from multiple sources into a single trace stream. It is implementation-dependent how many incoming messages are accepted per cycle and in what order.

Register: trBaseFunnel+0x000 trFunnelControl: Trace Funnel Control Register

Bit	Field	Description	RW	Reset
0	trFunnelActive	Primary enable for trace funnel. When 0, the Trace Funnel may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written.	RW	0
1	trFunnelEnable	1=Trace Funnel enabled. Setting trFunnelEnable to 0 flushes any queued trace data to output.	RW	0
2	—	Reserved	—	0
3	trFunnelEmpty	Reads 1 when Trace Funnel internal buffers are empty	R	1

Register: trBaseFunnel+0x004 trFunnelImpl: Trace Funnel Implementation Register

Bit	Field	Description	RW	Reset
3-0	trFunnelVerMajor	Trace Funnel Component Major Version. Value 1 means component is compliant with this document.	R	1
7-4	trFunnelVerMinor	Trace Funnel Component Minor Version. Value 0 means component is compliant with this document.	R	0
11-8	trFunnelCompType	Trace Funnel Component Type (Trace Funnel)	R	0x8
23-12	—	Reserved for future versions of this standard	—	0
31-24	—	Reserved for vendor specific implementation details	—	SD

Chapter 12. Trace PIB Sink

Trace data may be sent to chip pins through an interface called the Pin Interface Block (PIB). This interface typically operates at a few hundred MHz and can sometimes be higher with careful constraints and board layout or by using LVDS or other high-speed signal protocol. PIB may consist of just one signal and in this configuration may be called SWT (Serial-Wire Trace). Alternative configurations include a trace clock clock (TRC_CLK) and 1/2/4/8/16 parallel trace data signals (TRC_DATA) timed to that trace clock. WARL register fields are used to determine specific PIB capabilities.

The modes and behavior described here are intended to be compatible with trace probes available in the market.

12.1. PIB Register Interface

Register: trBasePib+0x000 trPibControl: PIB Sink Control Register (Optional)

Bit	Field	Description	RW	Reset
0	trPibActive	Primary enable/reset for PIB Sink component	RW	0
1	trPibEnable	0=PIB does not accept input but holds output(s) at idle state defined by pibMode. 1=Enable PIB to generate output	RW	0
2	—	Reserved	—	0
3	trPibEmpty	Reads 1 when PIB internal buffers are empty	R	1
7-4	trPibMode	Select mode for output pins.	WARL	0
8	trPibClkCenter	In parallel modes, adjust TRC_CLK timing to center of bit period. This can be set only if trPibMode selects one of the parallel protocols. Optional.	WARL	SD
9	trPibCalibrate	Set this to 1 to generate a repeating calibration pattern to help tune a probe's signal delays, bit rate, etc. The calibration pattern is described below. Optional.	WARL	0
15-10	—	Reserved	—	0

Bit	Field	Description	RW	Reset
31-16	trPibDivider	Timebase selection for the PIB module. The input clock is divided by pibDivider+1. PIB data is sent at either this divided rate or 1/2 of this rate, depending on pibMode. Width is implementation-dependent. After PIB reset value of this pin should be set to safe setting for particular SoC. Trace tool may set smaller value in order to utilize higher bandwidth.	WARL	SD

Software can determine what modes are available by attempting to write each mode setting to the WARL field pibControl.pibMode and reading back to see if the value was accepted.

Mode	trPibMode	trPibClkCenter	Bit rate
Off	0	X	—
SWT Manchester	4	X	1/2
SWT UART	5	X	1
TRC_CLK + 1 TRC_DATA	8	0	1
TRC_CLK + 2 TRC_DATA	9	0	1
TRC_CLK + 4 TRC_DATA	10	0	1
TRC_CLK + 8 TRC_DATA	11	0	1
TRC_CLK + 16 TRC_DATA	12	0	1
TRC_CLK + 1 TRC_DATA	8	1	1/2
TRC_CLK + 2 TRC_DATA	9	1	1/2
TRC_CLK + 4 TRC_DATA	10	1	1/2
TRC_CLK + 8 TRC_DATA	11	1	1/2
TRC_CLK + 16 TRC_DATA	12	1	1/2

Register: trBasePib+0x004 trPibImpl: Trace PIB Implementation Register

Bit	Field	Description	RW	Reset
3-0	trPibVerMajor	Trace PIB Sink Component Major Version. Value 1 means component is compliant with this document.	R	1
7-4	trPibVerMinor	Trace PIB Sink Component Minor Version. Value 0 means component is compliant with this document.	R	0
11-8	trPibCompType	Trace PIB Sink Component Type (PIB Sink)	R	0xA

Bit	Field	Description	RW	Reset
23-12	—	Reserved for future versions of this standard	—	0
31-24	—	Reserved for vendor specific implementation details	—	SD

Since the PIB supports many different modes, it is necessary to follow a particular programming sequence:

- Activate the PIB by setting trPibActive.
- Set the trPibMode, trPibDivider, trPibClkCenter, and trPibCalibrate fields. This will set the TRC_DATA outputs to the quiescent state (whether that is high or low depends on trPibMode) and start TRC_CLK running.
- Activate the receiving device, such as a trace probe. Allow time for PLL to sync up, if using a PLL with a parallel PIB mode.
- Set trPibEnable. This enables the PIB to generate output either immediately (calibration mode) or when the trace encoder or funnel begins sending trace messages.

Order of bits and bytes:

- Trace messages/packets are considered as sequence of bytes and are always transmitted with LSB bits/bytes first.
- Nexus MSEO bits are transmitted on LSB part and bit#0 first.
- Idle state must be transmitted as all MSEO and MDO bits = 1.
- In 16-bit mode first byte of message is transmitted on LSB part and MSEO of second/odd byte will be transmitted on bits #8-#9 and MDO on bits #10-#15.

Above rules allow receiving probe to skip idle messages.

12.2. Calibration Mode

In optional calibration mode, the PIB transmits a repeating pattern. Probes can use this to automatically tune input delays due to skew on different PIB signal lines and to adjust to the transmitter's data rate (trPibControl.trPibDivider and trPibControl.trPibClkCenter). Calibration patterns for each mode are listed here.

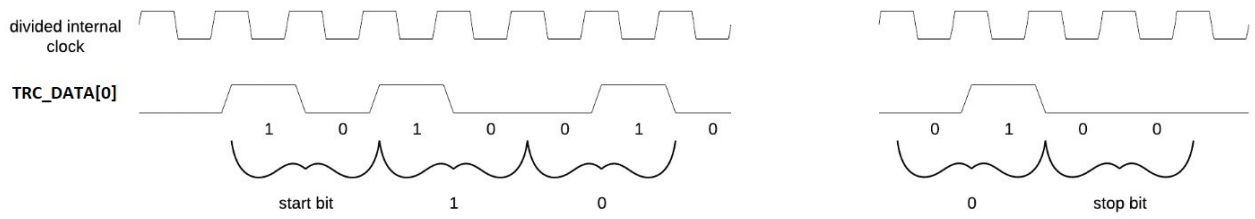
Mode	Calibration Bytes	Wire Sequence
UART, Manchester	AA 55 00 FF	alternating 1/0, then all 0, then all 1
1-bit parallel	AA 55 00 FF	alternating 1/0, then all 0, then all 1
2-bit parallel	66 66 CC 33	2, 1, 2, 1, 2, 1, 2, 1, 0, 3, 0, 3, 3, 0, 3, 0
4-bit parallel	5A 5A F0 0F	A, 5, A, 5, 0, F, F, 0
8-bit parallel	AA 55 00 FF	AA, 55, 00, FF

Mode	Calibration Bytes	Wire Sequence
16-bit parallel	AA AA 55 55 00 00 FF FF	AAAA, 5555, 0000, FFFF

12.3. SWT Manchester Protocol

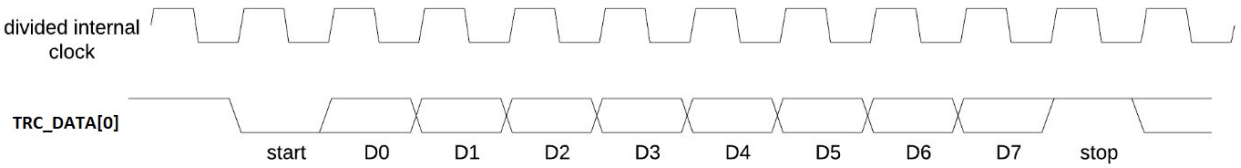
In this mode, the PIB outputs complete trace messages encapsulated between a start bit and a stop bit. Each bit period is divided into 2 phases and the sequential values of the TRC_DATA[0] pin during those 2 phases denote the bit value. Bits of the message are transmitted LSB first. The idle state of TRC_DATA[0] is low in this mode.

Bit	Phase 1	Phase 2
start	1	0
logic 0	0	1
logic 1	1	0
stop/idle	0	0



12.4. SWT UART Protocol

In UART protocol, the PIB outputs bytes of a trace message encapsulated in a 10-bit packet consisting of a low start bit, 8 data bits, LSB first, and a high stop bit. Another packet may begin immediately following the stop bit or there may be an idle period between packets. When no data is being sent, TRC_DATA[0] is high in this mode.



12.5. PIB Parallel Protocol

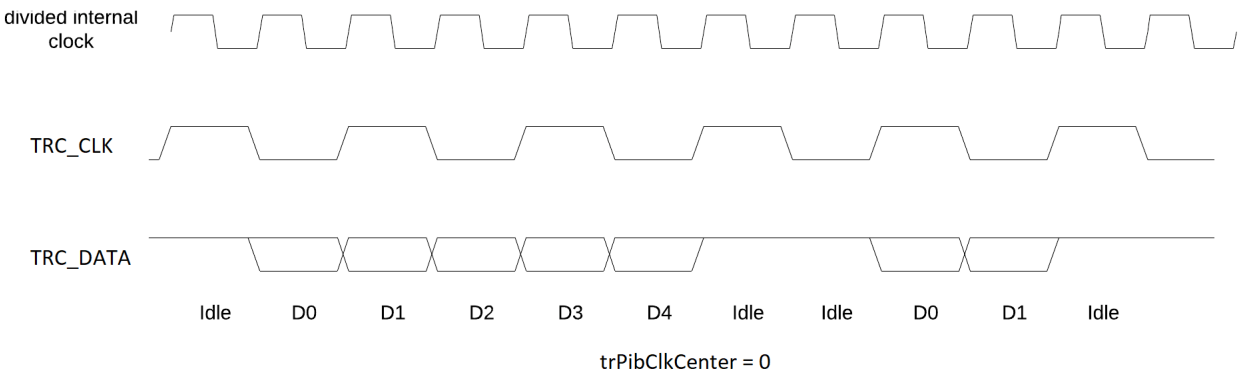
Traditionally, off-chip trace has used this protocol. There are a number of parallel data signals and one continuously-running clock reference. The data rate of several parallel signals can be much higher than either of the serial-wire protocols.

As with SWT modes, this protocol is oriented to full trace messages rather than fixed-width trace words. The idle state of TRC_DATA is all-ones for Nexus trace and (TBD) for Efficient Trace for RISC-V (E-Trace) Specification. When a message start is detected, this sample and possibly the next few (depending on the width of TRC_DATA) are collected until a complete byte has been received. Bytes are transmitted LSB first, with TRC_DATA[0] representing the LSB in each beat of data. The receiver continues collecting bytes until a complete message has been received. The criteria for this depends on the trace format. For Nexus, the last byte of a message is one that has mseo=1,1. For E-Trace, the header byte may include a byte count. After the last byte of a message, the data signals may then go their idle state or a new message may begin in the next bit period.

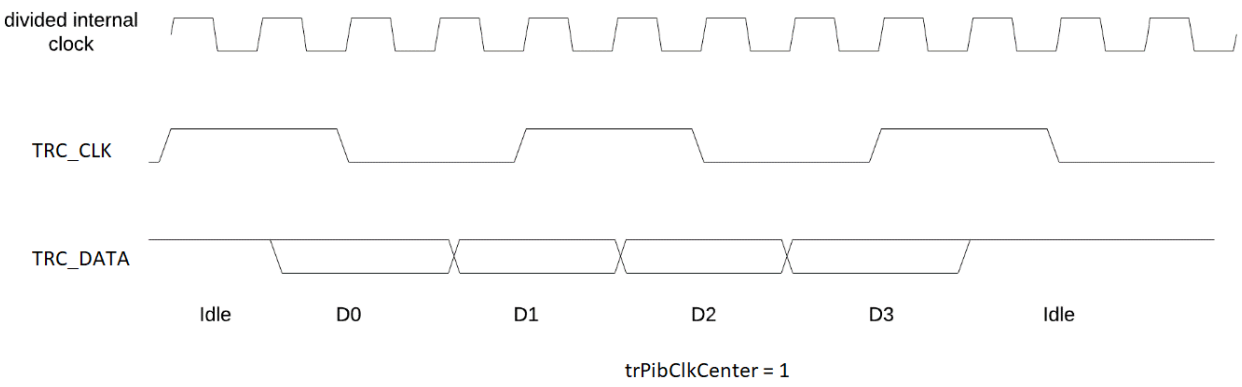
Trace messages may start on any (positive or negative) edge of trace clock. Once message is started all bits of that message must be transmitted on concecutive trace clock edges (both positive and negative). Said so, idle sequence may be sent consist of any number of trace clocks edges (positive or negative). But some implementations may always send idle sequences using even number of trace clocks - in such a case all packets will always start on positive or negative trace clock.

The trace clock, TRC_CLK, normally has edges coincident with the TRC_DATA edges. Typically, a trace probe will delay trace data or use a PLL to recover a sampling clock that is twice the frequency of TRC_CLK and shifted 90 degrees so that its rising edges occur near the center of each bit period. If the PIB implementation supports it, the debugger can set trPibClkCenter to change the timing of TRC_CLK so that there is a TRC_CLK edge at the center of each bit period on TRC_DATA. Note that this option cuts the data rate in half relative to normal parallel mode and still requires the probe to sample TRC_DATA on both edges of TRC_CLK.

This example shows 8-bit parallel mode with trPibClkCenter=0 transmitting a 5-byte message followed by a 2-byte message.



And an example showing 8-bit parallel mode transmitting a 4-byte packet with trPibClkCenter=1


divided internal clock
TRC_CLK
TRC_DATA
Idle
D0
D1
D2
D3
Idle
trPibClkCenter = 1

Chapter 13. Trace ATB Bridge

Some SoCs may have an Advanced Trace Bus (ATB) infrastructure to manage trace produced by other components. In such systems, it is feasible to route RISC-V trace output to the ATB through an ATB Bridge. This module manages the interface to ATB, generating ATB trace records that encapsulate RISC-V trace produced by the Trace Encoder or Trace Funnel. There is a control register that includes trace on/off control and a field allowing software to set the Trace Source ID to be used on the ATB. This Trace Source ID allows software to extract RISC-V trace from the combined trace. This interface is compatible with AMBA 4 ATB v1.1.

Register: trAtbBridgeBase+0x000 trAtbBridgeControl: ATB Bridge Control Register

Bit	Field	Description	RW	Reset
0	trAtbBridgeActive	Primary enable for the ATB Bridge. When 0, the ATB Bridge may have clocks gated off or be powered down, and other register locations may be inaccessible. Hardware may take an arbitrarily long time to process power-up and power-down and will indicate completion when the read value of this bit matches what was written.	RW	0
1	trAtbBridgeEnable	1=ATB Bridge enabled. Setting trAtbBridgeEnable to 0 flushes any queued trace data to ATB.	RW	0
2	—	Reserved	—	0
3	trAtbBridgeEmpty	Reads 1 when ATB Bridge internal buffers are empty	R	1
7-4	—	Reserved	—	0
14-8	trAtbBridgeID	ID of this node on ATB. Values of 0x00 and 0x70-0x7F are reserved by the ATB specification and should not be used.	RW	0

Register: trAtbBridgeBase+0x004 trAtbBridgeImpl: ATB Bridge Implementation Register

Bit	Field	Description	RW	Reset
3-0	trAtbBridgeVerMajor	ATB Bridge Component Major Version. Value 1 means component is compliant with this document.	R	1
7-4	trAtbBridgeVerMinor	ATB Bridge Component Minor Version. Value 0 means component is compliant with this document.	R	0
11-8	trAtbBridgeCompType	ATB Bridge Component Type (ATB Bridge)	R	0xE

Bit	Field	Description	RW	Reset
23-12	—	Reserved for future versions of this standard	—	0
31-24	—	Reserved for vendor specific implementation details	—	SD

An implementation determines the data widths of the connection from the Trace Encoder or Trace Funnel and of the ATB port.

Chapter 14. Reset and Discovery

This chapter describes what trace tool should do to reset and discover trace features.

Trace tool must be provided with base addresses of all trace components.

There are several (independent) reset bits defined by this specification

- trTeActive - reset for Trace Encoder component (this will disable encoder from single hart)
- trFunnelActive - reset for Trace Funnel component
- trPibActive - reset for PIB component (resets Probe Interface Block only)
- trAtbBridgeActive - resets ATB Bridge component (resets ATB Bridge Interface)

These reset bits should (when kept low) reset most of other registers/fields/bits to defined reset values.

Releasing component from reset may take time - debug tool should monitor (with reasonable timeout) if appropriate bit should changed from 0 to 1. Other fields/bits should remain unchanged (as were set during reset).

Some of reset values are defined as 'SD' (system dependent) and these values should reset as well and each time to same value as would be after power-up.

Some bigger registers (holding RAM addresseses) may not reset - debugger is expected to write to them before enabling trace. These registers have 'Undef' in reset field. It should not prevent some implementations to reset these.

Reset and Discovery should be performned as follows:

- Reset particular component by setting 'tr??Active = 0'.
- Read-back and wait until 'tr??Active = 0' is read.
- Save 'tr??Control' register as it holds all reset values (it may prevent tool to do read-modify operation later).
- Release from reset by setting 'tr??Active=1' and wait for 'tr??Active=1' to be read (to confirm component is not in reset).

  When 'tr??Active=1' is set no other bits should be changed.

- Read 'tr??Impl' and compare 'tr??ComType' field with expected value.
- Handle 'tr??VerMinor/Major' as described in 'Versioning of Components' chapter.
- Set some WARL fields and read back to discover supported component configuration - make sure component is NOT enabled by mistake.
- Configure some initial values in all needed registers/fields. Read-back each one to assure these are set properly.

As we are dealing with several independent components, it is important to assure, that component which is in reset (or powered down) is keeping it's outputs on safe values, so garbage trace data is not emitted.

In general it is safer to power-up and enable components starting from sinks/bridge, followed by Funnel and Encoder as last. Each implementation should test this sequence.

Chapter 15. Enabling and Disabling

Enabling should work as follows:

- Release all needed components from reset by setting 'tr??Active=1' as described above.
- Set desired mode and verify if that mode is set (regardless of discovery results).
- For RAM Sink:
  - Setup needed addresses (if possible and desired) as these may not reset.
- For PIB Sink:
  - Calibrate PIB (if possible and desired).
  - Start physical trace capture (probe dependent).
- Configure RAM Sink/PIB Sink/ATB Bridge in appropriate mode.
  - Verify if particular mode is set.
- Set main enable for RAM Sink/PIB Sink/ATB Bridge component by setting 'tr??Enable=1'.
  - Read back and wait for confirmation (tr??Enable=1).
- Enable Trace Funnel[s] in the same way.
- Configure and Enable Trace Encoder[s] in the same way.
- Either manually set 'trTeInstTracing=1' and/or 'trTeDataTracing=1' bits or setup triggesr to start the trace .
- Start hart[s] to be traced (hart could be already running as well - in this case trace will be generated in the moment when 'trTe????Tracing' bit was set)
- Periodically read 'trTeControl' for status of trace (as it may stop by itself due to triggers).
  - If RAM Sink was configured with 'trRamStopOnWrap=1', read 'trRamEnable' to see if RAM capture was stopped.

Discovery may not be necessary to enable and test trace during development of SoC. However discovery must be possible and should be tested by SoC designer - this is necessary for trace tools to work with that SoC without any customization.

Trace tool may verify particular setting once per session, so subsequent starts of trace may be faster.

Trace tools should provide configuration setting allowing more verbose logging mode during discovery and initialization, so potential compatibility issues may be solved.

Disabling the trace should work as follows:

- It is essential to disable the trace from encoders associated with stopped harts as entering debug more is NOT flushing any trace pipelines.

- Disable and flush trace starting from Encoders, then Funnels and finally Sinks or Bridges.
  - Set 'tr??Enable=0' and wait for 'tr??Enable=0' and 'tr??Empty=1' for each trace component.
  - It is important to do it in that order as otherwise data may be lost.
- Stop physical capture if PIB sink was enabled (probe dependent).
- Read the trace.
  - For RAM Trace Sink read 'trRamWP' - depending on trRamWrap bit, you may read trace from two ranges.
  - For RAM Trace Sink in SRAM mode, set 'trRamRP' and read 'trRamData' multiple times.
  - For RAM Trace Sink in SBA mode, read trace from SBA using memory read.
  - For PIB Trace Sink read trace from trace probe.
  - For ATB Bridge, read trace using Coresight compoments (ETB/TMC/TPIU).

Decoding trace

- Decoder (in most cases) must have an access to code which is running on device either by reading it from device or from a file containg the code (binary/hex/srec/ELF).
- Trace collected by trace probes can be read and decoded while trace is being captured (this is called trace streaming mode).
  - There is no guarantee that last trace packet is completed until trace is properly flushed and disabled.
- Decoding of trace should never affect code being traced.

Chapter 16. Legacy Interface Version

Value of 'teVersion/tfVersion' as 0 means this is original version of this interface.

As there are some implementations with trTeVersion = 0 it is important to provide changes, so tools may work with it.

```
Trace components are now separated into 4K blocks
```

- Some registers/fields got renamed for clarity and uniformity
- Field trTeInstTrigEnable was not present, so global enable/disable for instruction trace triggers was not possible
- Field trTeInstStallDelta was not present, so debugger may not know if hart was ever stalled
- Fields teSyncMode and teSyncMax were defined as 'teSyncMaxBTM and teSyncMaxInst'
- Fields pibEmpty and atbEmpty were not defined (trace control should wait to assure that trace was flushed correctly)
- Register trTeInstFeatures was not present (was reading as 0)
- Register trTeDataControl register was not present (as version 0 did not support data trace)
- 16-bit parallel mode for PIB was not defined (these implementations were using max 8-bit of parallel trace)

Chapter 17. Original Version Disclaimer

This document was converted to ADOC from original proposal by SiFive hosted here:

lists.riscv.org/g/tech-nexus/files/RISC-V-Trace-Control-Interface-Proposed-20200612.pdf

During this conversion (automatic) content was not altered. Later formatting details were (manually) adjusted.

Document Version 20200612