Specification of RISC-V Trace Control Interface

Version 0.9.4, Aug 15, 2022: This document is in Stable state. Assume it may change.

Table of Contents

Preamble

Document State

2022/8/15: Version 0.9.4-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.

2022/8/15: 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. Core 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 core 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 core 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 core 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. Control Interface

The Trace Encoder control interface consists of a set of 32-bit registers occupying up to a 4K-byte space. The control interface can be used to set up and control a trace session, retrieve collected trace history, and control any trace system components embedded in or directly connected to a Trace Encoder.

The Trace Encoder control registers will typically be accessed by a debugger through RISC-V debug module. The Trace Encoder may also be accessible through loads and stores performed by one or more harts in the system. Typically, the Trace Encoder connects to the system bus as a peripheral device, but it may use a dedicated bus connection from the Debug Module, or could attach to the DMI bus defined in the RISC-V Debug Specification.

Additional control path(s) may also be implemented, such as a dedicated debug bus or message-passing network. It is recommended to allow access to trace registers while core is running (using SBA access for example) as otherwise debugger will not be able to monitor trace status.

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).

There is typically one Trace Encoder per hart. A core with multiple harts (i.e., multi-threaded) will generate messages with a field indicating which hart is responsible for that message. Cores capable of retiring more than one instruction per cycle are accommodated with a single Trace Encoder.

The Trace Funnel allows to combine several trace sources into one trace stream. Each Trace Encoder and the Trace Funnel has its own set of control registers in its own register block.

3.1. Register Map

The 4K block occupied by a Trace Encoder or Trace Funnel is divided into sixteen logical sections of 256 bytes. Section 0 is required and is used for local control registers. Other sections are used for control registers of trace components that are conceptually separate, even if they are physically part of the Trace Encoder/Funnel. Examples of possible additional subcomponents are:

- PC Sampling
- Filtering
- Instrumented Trace
- Additional Sink Types

Registers in the 4K range that are not implemented 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.

Address Offset	Trace Encoder	Trace Funnel	Compliance	Description
0x000	teControl	tfControl	Required	Trace Encoder/Funnel control register
0x004	teImpl	tfImpl	Required	Trace Encoder/Funnel implementation information
0x008	teInstFeatures	—	Optional	Extra instruction trace encoder features
0x00C	teDataControl	—	Optional	Data trace control and features
0x010	teRamBase	teRamBase	Optional	Base address of circular trace buffer
0x014	teRamBaseHigh	teRamBaseHigh	Optional	Bits N:32 of the circular buffer address
0x018	teRamLimit	teRamLimit	Optional	End address of circular trace buffer
0x01C	teRamWP	teRamWP	Optional	Current write location for trace data in circular buffer
0x020	teRamRP	teRamRP	Optional	Access pointer for trace readback
0x024	teRamData	teRamData	Optional	Read/write access to trace memory
0x02C - 0x03F	-	-	Optional	Reserved for more teRam… registers (64-bit)
0x040	tsControl	—	Optional	Timestamp control register
0x044	tsLower	—	Optional	Lower 32 bits of timestamp counter
0x048	tsUpper	—	Optional	Upper bits of timestamp counter
0x050	xTrigInControl	—	Optional	External Trigger Input control register
0x054	xTrigOutControl	—	Optional	External Trigger Output control register
0x058	dbgTriggerControl	—	Optional	Debug triggers control register
0x060 - 0x2FF	—	—	Optional	Reserved for more registers
0x300 - 0x3FF			Optional	Control registers for Trace Filter block
0x400 - 0x7FF			Optional	Blocks reserved for Vendor-specific Trace Components
0x800 - 0xDFF			Optional	Blocks reserved for Future Trace Components
0xE00 - 0xEFF	atbSink	atbSink	Optional	Control registers for ATB trace sink, if attached to this TE/TF

Address Offset	Trace Encoder	Trace Funnel	Compliance	Description
0xF00 - 0xFFF	pibSink	pibSink	Optional	Control registers for PIB trace sink, if attached to this TE/TF

Chapter 4. Trace Encoder Introduction

This section briefly describes features of the Trace Encoder (TE) as background for understanding some of the control interface register fields.

4.1. Trace Encoder Types

By monitoring the Ingress Port, the TE 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.

4.2. 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 TE 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.

4.3. 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 is encoded in single 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.

4.4. 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.

4.5. 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.

4.5.1. SRAM Sink

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

4.5.2. 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.

4.5.3. System Memory (SBA) 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, the TE may optionally allow trace to 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.

4.5.4. ATB Sink

The ATB Sink transmits bytes of 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.

4.5.5. 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 'SRC' field defined (this is teSrcID field in teControl register).

Chapter 5. 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: 0x000 teControl: Trace Encoder Control Register (Required)

Bit	Field	Description	RW	Reset
0	teActive	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	teEnable	1=TE enabled. Allows teInstTracing to turn all tracing on and off. Setting teEnable to 0 flushes any queued trace data to the designated sink. This bit can be set to 1 only by direct write to it.	RW	0
2	teInstTracing	1=Instruction trace is being generated. Written from tool or controlled by triggers. When teInstTracing=1, instruction trace data may be subject to additional filtering in some implementations (additional teInstMode settings).	RW	0
3	teEmpty	Reads as 1 when all generated trace has been emitted.	R	1

Bit	Field	Description	RW	Reset
6-4	teInstMode	Main instruction trace generation mode 0 = Instruction trace is disabled 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 (teInstFeatures register if present define instruction trace features and optimizations).	WARL	SD[(1)]
8-7	—	Reserved for futue modes	WARL	SD
9	teContext	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	teInstTrigEn	When 1, allows teInstTracing to be set or cleared by trace-on and trace-off Debug module triggers respectively	WARL	0
12	teInstStallOrOverflow	Written to 1 by hardware when an overflow message is generated or when the TE requests a core stall. Clears to 0 at TE reset or when trace is enabled (teEnable set to 1).	R	0
13	teInstStallEn	0 = If TE cannot send a message, an overflow is generated when trace is restarted. 1 = If TE cannot send a message, the core is stalled until it can.	WARL	SD
14	teStopOnWrap	Disable trace (teInstEnable, teDataEnable → 0) when circular buffer fills for the first time.	WARL	SD

Bit	Field	Description	RW	Reset
15	teInhibitSrc	1=Disable source field in trace messages. Unless disabled, a trace source field (of teImpl.teSrcBits) is added to every trace message to indicate which TE generated each message. If teImpl.teSrcBits is 0, this bit is not active.	WARL	SD
17-16	teSyncMode	Select periodic synchronization mechanism. At least one non-zero mechanism must be implemented. 0 = Off 1 = Count trace messages/packets 2 = Count clock cycles 3 = Count instruction half-words (16-bit)	WARL	SD
19-18	Reserved	—	—	0
23-20	teSyncMax	The maximum interval (in units determined by teSyncMode) between synchronization messages/packets. Generate synchronization when count reaches 2^(teSyncMax + 4). If synchronization packet is generated from another reason internal counter should be reset.	WARL	SD
26-24	teFormat	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

Bit	Field	Description	RW	Reset
31-28	teSink	Which sink to send trace to. 0-3 = Reserved 4 = SRAM Sink 5 = ATB Sink 6 = PIB Sink 7 = System Memory Sink 8 = Funnel Sink 9-11 = Reserved for future sink types 12-15 = Reserved for vendor-specific sink types	WARL	SD

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

Register: 0x004 teImpl: Trace Encoder Implementation Register (Required)

Bit	Field	Description	RW	Reset
3-0	teVersion	TE Version. Value 1 means module is compliant with this document. Value 0 means legacy version - see 'Legacy Interface Version' chapter at the end.	R	1
4	hasSRAMSink	1 if this TE has an on-chip SRAM sink. Size of SRAM may be determined by writing all 1s to teRamWP, then reading the value back.	R	SD
5	hasATBSink	1 if this TE has an ATB sink.	R	SD
6	hasPIBSink	1 if this TE has an off-chip trace port via a Pin Interface Block (PIB)	R	SD
7	hasSBASink	1 if this TE has an on-chip system memory bus trace sink.	R	SD
8	hasFunnelSink	1 if this TE feeds into a trace funnel device.	R	SD
11-9		Reserved for future sink types	R	0
15-12		Reserved for vendor-specific sink types	R	SD
19-16		Reserved for vendor-specific features	—	—

Bit	Field	Description	RW	Reset
23-20	teSrcID	This TE's source ID. If teSrcBits>0 and trace source is not disabled by teInhibitSrc, then messages will all include a trace source field of teSrcBits bits. Messages from this TE will use this value as trace source field. May be fixed or variable.	WARL	SD
26-24	teSrcBits	The number of bits in the trace source field, unless disabled by teInhibitSrc. May be fixed or variable.	WARL	SD
27		Reserved	—	—
31-28		Reserved for vendor-specific features	—	—

Register: 0x008 teInstFeatures: Trace Instruction Features Register

Bit	Field	Description	RW	Reset
0	teInstNoAddrDiff	Do not send differential addresses when set (always full address is sent)	WARL	0
1	teInstNoTrapAddr	When set, do not sent trap handler address in trap packets	WARL	0
2	teInstEnSequentialJump	Treat sequentially inferrable jumps as inferable PC discontinuities when set.	WARL	0
3	teInstEnCallStack	Treat returns as inferable PC discontinuities when returning from recent call on stack.	WARL	0
4	teInstEnBranchPrediction	Branch predictor enabled when set.	WARL	0
5	teInstEnJumpTargetCache	Jump target cache enabled when set.	WARL	0

Register: 0x00C teDataControl: Data Trace Control Register (for encoders supporting data trace)

Bit	Field	Description	RW	Reset
0	teDataImplemented	Read as 1 if data trace is implemented.	R	SD
1	teDataEnable	Main enable for data trace.	WARL	SD
2	teDataTracing	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
3	teDataTrigEnable	Global enable/disable for data trace triggers	WARL	0

Bit	Field	Description	RW	Reset
4	teDataStallDelta	Set to 1 if data trace caused stall since last read. It is clear on read.	R	0
5	teDataStallEnable	Stall execution if data trace message cannot be generated.	WARL	0
6	teDataDropDelta	Set to 1 if data trace was dropeed since last read. It is clear on read.	R	0
7	teDataDropEnable	Allow dropping data trace to avoid instruction trace overflows. Seting this bit will not guarantee that instuction trace overflows will not happen.	WARL	0
15-8		Reserved for additional data trace control/status bits.	—	—
16	teDataNoValue	Omit data values from data trace packets when set.	WARL	SD
18-17	teDataAddressMode	'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
31-19	Reserved for more data trace features	Bit-mask of allowed/enabled data trace features.	WARL	SD

Chapter 6. Trace RAM Sink Control Interface

Registers defined in this chapter are applicable to both SRAM and SBA sinks. SRAM sink type is using dedicated memory, while SBA type 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 register, while SBA memory 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 0 last byte of each message/packet is specially marked.

Register: 0x010 teRamBase: Trace Encoder Ram Sink Base Register (Optional)

In table below value 'N' define number of address bits on bus where trace memory is connected. For busses with address larger than 32-bit, N=32 and corresponding 'High' register define MSB part of larger address.

FUTURE: As it may be desired to have RAM trace bigger than 4GB in size, all registers holding RAM address must have high-counterparts. It may be also possible to have just one 'high32' register and use it as a 'port' to access one of N physical registers (both read and write). Write 'base+high32' (in that order) would set 'baseHigh', write 'wp+high' may set 'wpHigh'. As this is very rare use cases (4GB trace is really, really big!), maybe this is better option then providing 'teRamLimitHigh'. 'teRamWPHigh' and 'teRamRPHigh' register?

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 'teWrap' 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
N-2	teRamBase	Base byte address 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 teRamBase fixed at 0.	WARL	Undef or fixed to 0

Register: 0x014 teRamBaseHigh: Trace Encoder Ram Sink Base High Bits Register (Optional)

Bit	Field	Description	RW	Reset
M-0	teRamBaseHigh	High order bits (>=32) of address of trace sink circular buffer. This register may not be present if no connected sinks require more than 32 address bits.	WARL	Undef

Register: 0x018 teRamLimit: Trace Encoder Sink Limit Register (Optional)

Bit	Field	Description	RW	Reset
1-0	—	Always 0 (two LSB of 32-bit address)	R	0
N-2	teRamLimit	Highest absolute 32-bit part of address of trace circular buffer. The teRamWP register is reset to teRamBase 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: 0x01C teRamWP: Trace Encoder Sink Write Pointer Register (Optional)

Bit	Field	Description	RW	Reset
0	teWrap	Set to 1 by hardware when teRamWP wraps. It is only set to 0 if teRamWp is written	WARL	0
1	—	Always 0 (bit B1 of 32-bit address)	R	0
N-2	teRamWP	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 teRamWP=teRamLimit, teRamWP is set to teRamBase. This register may not be present if no sinks require it.	WARL	Undef

Register: 0x020 teRamRP: Trace Encoder SRAM Sink Access Pointer Register (Optional)

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

Register: 0x024 teRamData: Trace Encoder SRAM Sink Data Register (Optional)

Bit	Field	Description	RW	Reset
31-0	teRamData	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

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 read 3 times always, 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)

Chapter 7. Timestamp Unit

Timestamp is an optional feature. An implementation may choose from several types of timestamp unit: Internal, External, Slave, or Vendor-specific. Implementations may have no timestamp, one timestamp type, or more than one type. The WARL field tsType is used to determine the system capability and to set the desired type.

- Internal uses a fixed system clock to increment the timestamp counter
- Slave mode accepts a timestamp broadcast from another Trace Encoder
- External accepts a binary timestamp value from an outside source such as ARM CoreSight™ trace
- The width of the timestamp is implementation-dependent

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 single-hart system with an Internal Timestamp counter, 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. Most other control bits are also optional. A debugger may determine the specific capabilities by writing and reading back a WARL register field.

7.1. Timestamp Register Interface

Register: 0x040 tsControl: Timestamp Control Register (Optional)

Bit	Field	Description	RW	Reset
0	tsActive	Primary reset/enable for timestamp unit	RW	0
1	tsCount	Internal Timestamp only. 1=counter runs, 0=counter stopped	WARL	0
2	tsReset	Internal Timestamp only. Write 1 to reset the timestamp counter	W1	0
3	tsDebug	Internal Timestamp only. 1=counter runs when hart is halted, 0=stopped	WARL	0
6-4	tsType	Type of Timestamp unit 0 = none 1 = External 2 = Internal 3 = Reserved 4 = Slave 5-7 = Vendor-specific type	WARL	SD

Bit	Field	Description	RW	Reset
9-8	tsPrescale	Internal Timestamp only. Prescale timestamp clock by 2^2n (1, 4, 16, 64).	WARL	0
23-15		System-dependent fields to control what message types include timestamps.	WARL	0
31-24	tsWidth	Width of timestamp in bits	R	SD

Register: 0x044 tsLower: Timestamp Lower Bits (Optional)

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

Register: 0x048 tsUpper: Timestamp Upper Bits (Optional)

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

Chapter 8. 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.

8.1. External Trigger Register Interface

Register: 0x050 xTrigInControl: External Trigger Input Control Register (Optional)

Bit	Field	Description	RW	Reset
3-0	xTrigInAction0	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 = start trace (teInstTracing → 1) 3 = stop trace (teInstTracing → 0) 4 = record Program Trace Sync message 5-15 = reserved	WARL	0
31-4	xTrigInAction*n*	Select actions for external triggers 1 through 7. If an external trigger input does not exist, then its action is fixed at 0.	WARL	0

Register: 0x054 xTrigOutControl: External Trigger Output Control Register (Optional)

Bit	Field	Description	RW	Reset
3-0	xTrigOutEvent0	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] = starting trace (teInstTracing 0 → 1) [1] = stopping trace (teInstTracing 1 → 0) [2] = (Optional) Vendor-specific event [3] = (Optional) Vendor-specific event	WARL	0
31-4	xTrigOutEvent*n*	Select events for external trigger outputs 1 through 7. If an external trigger output does not exist, then its event bits are fixed at 0	WARL	0

Chapter 9. Debug Triggers

Debug triggers are signals from the core 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 mcontrol CSR (0x7a1). Not all cores support trace actions; the debugger should read back mcontrol after setting one of these actions to verify that the option exists.

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

action (from debug spec)	**Effect**
0	Breakpoint exception
1	Debug exception
2	Start trace (teInstTracing → 1)
3	Stop trace (teInstTracing → 0)
4	Record Program Trace Sync message
5	Optional vendor-specific action

9.1. Debug Trigger Register Interface

Register: 0x058 dbgTriggerControl: Debug Trigger Control Register

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

9.2. Debug 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 10. Trace Filter Control Interface

All registers with offsets 0x300 .. 0x3FC are reserved for additional trace 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 are defined in chapter 'Filtering'.

Chapter 11. Trace Funnel

The Trace Funnel combines messages from multiple sources into a single trace stream. The Funnel has the same options for sinks as a single Trace Encoder which are controlled by the same registers as described above. It is implementation-dependent how many incoming messages are accepted per cycle and in what order.

Fields in 'tfControl' registers are sub-sets of fields in 'teControl' register.

FUTURE: Funnel should be extended to see what TE[s] or other funnels are connected to particular funnel (read-only bit in TE register connected to bit settable in TF register)

11.1. Trace Funnel Register Interface

Register: 0x000 tfControl: Trace Funnel Control Register

Bit	Field	Field in 'teControl'
0	tfActive	teActive
1	tfEnable	teActive
3	tfEmpty	teEmpty
14	tfStopOnWrap	teStopOnWrap
31-28	tfSink	teSink

Register: 0x004 tfImpl: Trace Funnel Implementation Register

Bit	Field	Corresponding 'teImpl' field
3-0	tfVersion	teVersion
4	hasSRAMSink	hasSRAMSink
5	hasATBSink	hasATBSink
6	hasPIBSink	hasPIBSink
7	hasSBASink	hasSBASink
8	hasFunnelSink	hasFunnelSink (next level funnel)

Chapter 12. PIB Trace 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 reference clock (tref) and 1/2/4/8/16 parallel trace data signals (tdata) timed to that reference. 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: 0xF00 pibControl: PIB Sink Control Register (Optional)

Bit	Field	Description	RW	Reset
0	pibActive	Primary enable/reset for PIB Sink block	RW	0
1	pibEnable	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	pibEmpty	Reads 1 when PIB internal buffers are empty	R	1
7-4	pibMode	Select mode for output pins.	WARL	0 (PIB is off)
8	pibRefCenter	In parallel modes, adjust tref timing to center of bit period. This can be set only if pibMode selects one of the parallel protocols. Optional.	WARL	SD
9	pibCalibrate	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
31-16	pibDivider	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.	WARL	SD (safe setting for particular SoC)

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	pibMode	pibRef Center	Bit rate
Off	0	X	—

Mode	pibMode	pibRef Center	Bit rate
SWT Manchester	4	X	1/2
SWT UART	5	X	1
tref + 1 tdata	8	0	1
tref + 2 tdata	9	0	1
tref + 4 tdata	10	0	1
tref + 8 tdata	11	0	1
tref + 16 tdata	12	0	1
tref + 1 tdata	8	1	1/2
tref + 2 tdata	9	1	1/2
tref + 4 tdata	10	1	1/2
tref + 8 tdata	11	1	1/2
tref + 16 tdata	12	1	1/2

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

- Activate the PIB by setting pibActive.
- Set the pibMode, pibDivider, pibRefCenter, and pibCalibrate fields. This will set the tdata outputs to the quiescent state (whether that is high or low depends on pibMode) and start tref 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 pibEnable. This enables the PIB to generate output either immediately (calibration mode) or when the trace encoder 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

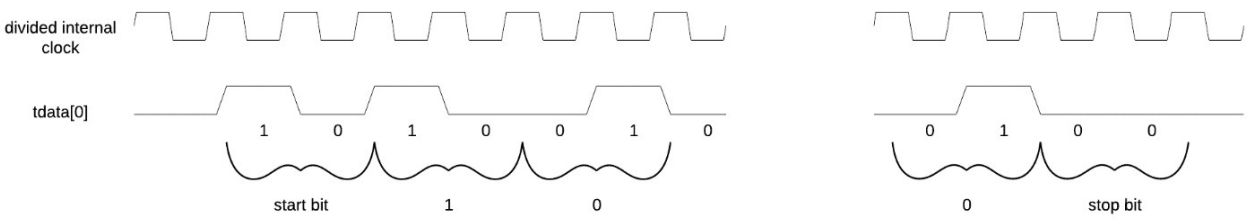
transmitter's data rate (pibContro.pibDivider and pibControl.pibRefCenter). Calibration patterns for each mode are listed here.

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
16-bit parallel	AA AA 55 55 00 00 FF FF	AAAA, 5555, 0000, FFFF

12.3. SWT Manchester Protocol

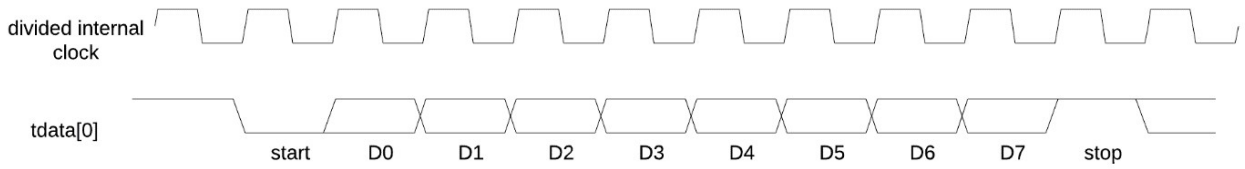
In this mode, the PIB outputs complete trace messages encapsulated between a start bit and a stop bit. Each bit period is divided into 2 phases and the sequential values of the tdata[0] pin during those 2 phases denote the bit value. Bits of the message are transmitted LSB first. The idle state of tdata[0]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



12.5. PIB Parallel Protocol

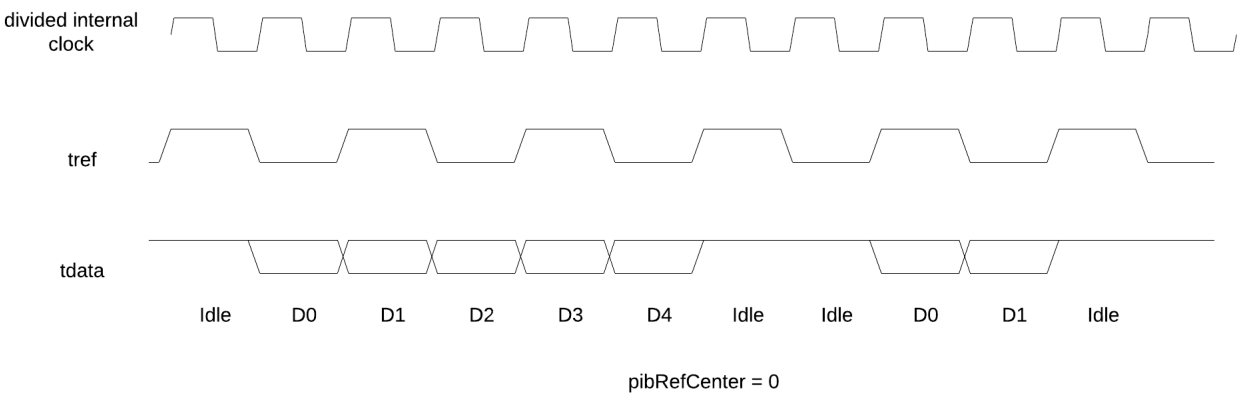
Traditionally, off-chip trace has used this protocol. There are a number of parallel data signals and one continuously-running clock reference. The data rate of several parallel signals can be much higher than either of the serial-wire protocols.

As with SWT modes, this protocol is oriented to full trace messages rather than fixed-width trace words. The idle state of tdata 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 tdata) are collected until a complete byte has been received. Bytes are transmitted LSB first, with tdata[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.

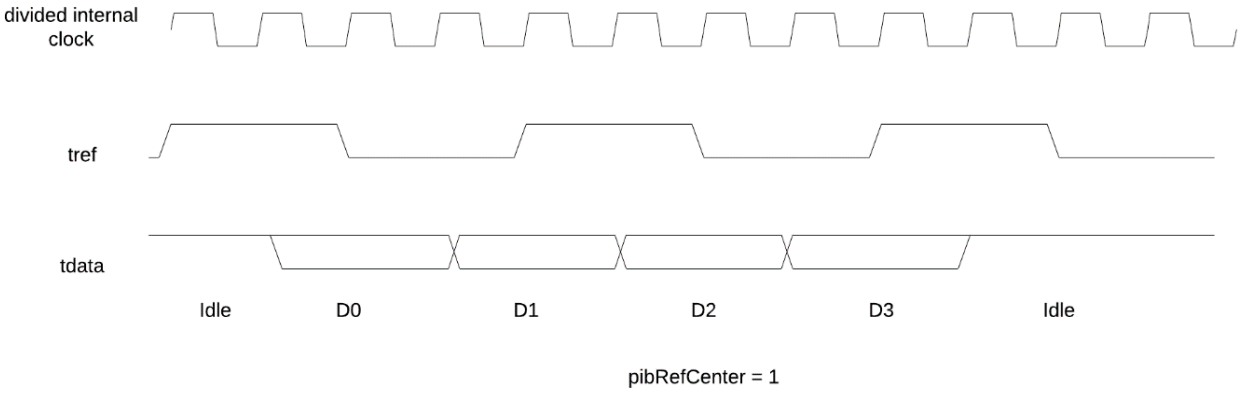
Trace messages may start on any (positive or negative) edge of trace clock. Once message is started all bits of that message must be transmitted on concecutive trace clock edges (both positive and negative). Said so, idle sequence may be sent consist of any number of trace clocks edges (positive or negative). 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 clock reference, tref, normally has edges coincident with the tdata edges. Typically, a trace probe will delay trace data or use a PLL to recover a sampling clock that is twice the frequency of tref 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 pibRefCenter to change the timing of tref so that there is a tref edge at the center of each bit period on tdata. Note that this option cuts the data rate in half relative to normal parallel mode and still requires the probe to sample tdata on both edges of tref.

This example shows 8-bit parallel mode with pibRefCenter=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 pibRefCenter=1.



Chapter 13. ATB Trace Sink

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 Trace Sink. This module manages the interface to ATB, generating ATB trace records that encapsulate RISC-V trace produced by the TE. There is a control register that includes trace on/off control and a field allowing software to set the DeviceID to be used on the ATB. This DeviceID allows software to extract RISC-V trace from the combined trace. This interface is compatible with AMBA 4 ATB v1.1.

Register: 0xE00 atbControl: ATB Sink Control Register

Bit	Field	Description	RW	Reset
0	atbActive	Primary enable/reset for ATB Sink block	RW	0
1	atbEnable	Enable trace words to pass through from the Trace Encoder to ATB	RW	0
2	Reserved	—	—	0
3	atbEmpty	Reads 1 when ATB internal buffers are empty	R	1
14-8	atbId	ID of this node on ATB. Values of 00 and 70-7F are reserved by the ATB specification and may not be used.	RW	0

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 to to discover

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

- teActive - reset for TE block (this will disable encoder from single core)
- pibActive - reset for PIB block (resets Probe Interface Block only)
- atbAcitive - resets ATB Sink Block (resets ATB Sink Interface)

All reset bits should (when kept low) reset most of other 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.

When component is in reset (single 'active' bit = 0), all control bits (and most registers) should reset.

Discovery should be performned as follows:

- Reset particular component and capture default values of all registers.
- Release from reset (waiting for acknowledge).
- Set (interesting) WARL fields and read-back values.

Chapter 15. Enabling and Disabling

Enabling should work as follows:

- Release TE from reset and wait for confirmation (teActive=1)
- Set desired mode and verify if that mode is set (regardless of discovery results)
- Enable sink and verify all settings
  - For RAM sink, setup addresses (if possible and desired)
- Release PIB from reset and calibrate PIB (if possible and desired)
- Enable PIB capture in appropriate mode (and verify if particular mode is set)
- Enable physical capture (probe dependent)
- Start core (core could be already running as well)
- Periodically read 'teControl' for status of trace (as it may stop by itself due to triggers)

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.

Disabling the trace should work as follows:

- In all Trace Encoders, set teControl.teTracing to 0. This inserts a trace-stop message.
- In all Trace Encoders, set teControl.teEnable to 0. This flushes any partial trace words to the sink or funnel.
- Poll teControl.teEmpty in all Trace Encoders. Wait until all are 1.
- For any Trace Encoder whose teControl.teSink indicates PIB sink (6), poll pibControl.pibEmpty and wait until it reads 1.
- For any Trace Encoder whose teControl.teSink indicates ATB sink (5), poll atbControl.atbEmpty and wait until it reads 1.
- In all Trace Funnels, set tfControl.tfEnable to 0. This flushes any partial trace words to the sink.
- Poll tfControl.tfEmpty in all Trace Funnels. Wait until all are 1.
- For any Trace Funnel whose tfControl.tfSink indicates PIB sink (6), poll pibControl.pibEmpty and wait until it reads 1.
- For any Trace Funnel whose tfControl.tfSink indicates ATB sink (5), poll atbControl.atbEmpty and wait until it reads 1.
- Stop physical capture if PIB sink was enabled (probe dependent)

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 file containg it (binary/hex/srec/ELF)
- Trace can be read and decoded while trace is captured
- There is no guarantee that last trace packet is completed until trace is properly flushed and disabled

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 teVersion = 0 it is important to provide changes, so tools may work with it.

- Some registers/fields got renamed for clarity and uniformity
- Field teInstTrigEnable was not present, so global enable/disable for instruction trace triggers was not possible
- Field teInstStallDelta was not present, so debugger may not know if core 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 teInstFeatures was not present (was reading as 0)
- Register teDataControl 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