

1. Version History

Version	Date (y/m/d)	Who	Changes
0.01	2019/10/29	Robert Chyla	Initial Version (for kick-off meeting 2019/10/30)

2. Status

Table below shows current status of various parts of this document

Chapter[s]	What	Recent Change	Notes
All	Initial sketch	None	Just a lot of TODO items. We must start somewhere.

3. Why Nexus

Nexus IEER-ISTO 5001™ is a well established and extensively documented standard.

It details a lot of topics considered by Processor Trace TG as out of scope. It is also important from tool vendor perspective to not design new probes to capture RISC-V trace, but rather utilize existing trace capture hardware and established standards.

Nexus trace may not be as well compressed as trace protocol to be soon defined by Processor Trace TG and as such it may be easier to implement. It may be essential for smaller RISC-V core implementations for IoT segment, where gate count for trace logic is important.

It is expected that big parts of this specification can be applied as-is/shared with Processor Trace TG.

It is also expected, that Nexus encoder will utilize same ingress port, so RISC-V SoC designers will have a choice to select which trace format better fits their needs.

4. Overview and key assumptions

1. This specification is based on following Nexus specification (called **Nexus spec** later on):
<http://nexus5001.org/wp-content/uploads/2018/05/IEEE-ISTO-5001-2012-v3.0.1-Nexus-Standard.pdf>
2. This specification will use terminology compatible with **Nexus spec**.
3. It is not required that the reader is familiar with **Nexus spec** itself, but in case of doubt it is suggested to read it as things are well explained there.
 - 3.1. Certain details of the **Nexus spec** may not be applicable to RISC-V, so be careful.
4. Some sections of this specification may directly refer to appropriate parts of **Nexus spec**. This will be done as follows **Nexus spec [SECTION 6]**.
5. Numbering 1./1.1/1.1.1 (restated at each chapter) will be used—for easy referencing.

5. Nexus and RISC-V debug spec

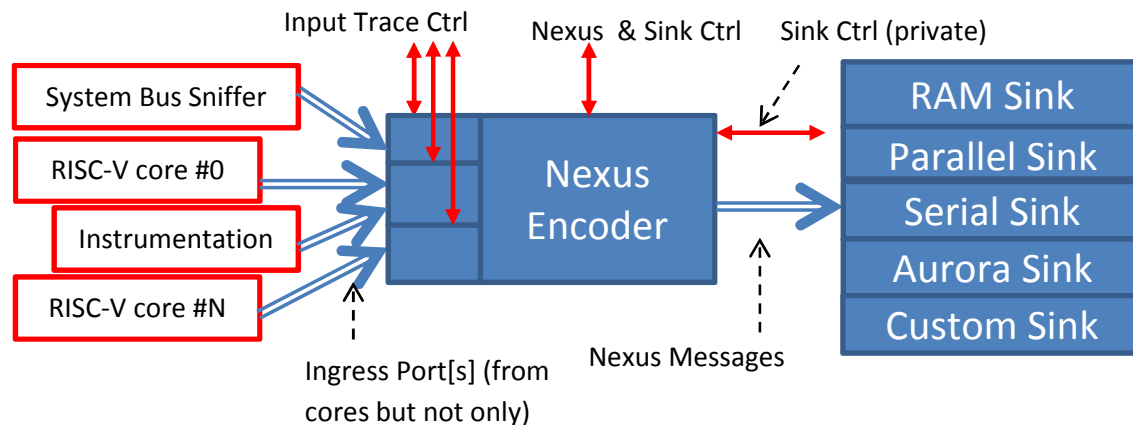
1. This specification does not cover any parts of core debug/control (which is part of original **Nexus spec**).
 - 1.1. Control of cores is assumed to be compatible with RISC-V Debug Spec 0.13.2 found here: <https://riscv.org/specifications/debug-specification/>
2. This specification ONLY deals with trace related topics.

6. Nexus compliance classes

1. This specification cannot be considered strictly compliant with any of the four Nexus compliance classes (Class 1/2/3/4) as even basic Class 1 mandates debug control of core[s].
2. Logically it is compliant with Class 2 which mandates program trace.
 - 2.1. Certain (optional) features from Nexus Class 3 and Class4 are also used.

7. General Architecture Overview

TODO: Picture below is very preliminary (to be changed):



8. Ingress Port

1. Same signals from core[s] in the system (defined by Processor Trace TG here: <https://lists.riscv.org/g/tech-trace>) will be used.
2. Said so, this specification should be understood as an alternative, Nexus compatible way of encoding trace from RISC-V cores.
 - 2.1. Certain implementations (especially on FPGA for testing) may provide both of them.
3. In limited implementation it is permitted to provide sub-set of ingress signals from the core.
4. Ingress port is considered as 'strong suggestion' only
 - 4.1. This specification defines output from Nexus encoder and as long as this output is compliant, particular implementation will be considered compliant.
 - 4.2. From external trace control and capture perspective ingress port specification does not really matter

- 4.3. If ingress port will impose any limitation on encoded trace, this fact should be known to trace tool by discovery of trace features.

9. Nexus trace messages - overview

1. *Nexus spec [Table 4-5—Nexus Public Messages]* shows list of all nexus messages.
2. Here is this table with added extra column detailing relevance of these messages for RISC-V:

Message Name	TCODE	Direction	RISC-V sub-set
Debug Status	0	From target	Not used (debug)
Device ID	1	From target	Not used (debug)
Ownership Trace	2	From target	Extra – RTOS trace
Program Trace - Direct Branch	3	From target	Level 1
Program Trace - Indirect Branch	4	From target	Level 1
Data Trace - Data Write	5	From target	No used (data trace)
Data Trace - Data Read	6	From target	No used (data trace)
Data Acquisition	7	From target	Extra - Instrumentation
Error	8	From target	Level 1
Program Trace - Synchronization	9	From target	Level 1
Program Trace - Correction	10	From target	Later (maybe not needed)
Program Trace - Direct Branch with Sync	11	From target	Level 1
Program Trace - Indirect Branch with Sync	12	From target	Level 1
Data Trace - Data Write with Sync	13	From target	No used (data trace)
Data Trace - Data Read with Sync	14	From target	No used (data trace)
Watchpoint Hit	15	From target	Extra – Debug Events
Reserved	16-19	--	Not used (reserved)
Port Replacement - Output	20	From target	Not used (not applicable)
Port Replacement - Input	21	From tool	Not used (not applicable)
Auxiliary Access – Read	22	Both ways	Not used (debug)
Auxiliary Access – Write	23	Both ways	Not used (debug)
Auxiliary Access - Read Next	24	Both ways	Not used (debug)
Auxiliary Access - Write Next	25	Both ways	Not used (debug)
Auxiliary Access - Response	26	Both ways	Not used (debug)
Program Trace - Resource Full	27	From target	Extra – Better error handling
Program Trace - Indirect Branch History	28	From target	Level 2
Program Trace - Indirect Branch History with Sync	29	From target	Level 2
Program Trace - Repeat Branch	30	From target	Level 2
Program Trace - Repeat Instruction	31	From target	Later (rare use-case)
Program Trace - Repeat Instruction with Sync	32	From target	Later (rare use-case)
Program Trace - Correlation	33	From target	Level 1
In-Circuit Trace	34	From target	Later (for bus trace?)
In-Circuit Trace with Sync	35	From target	Later (for bus trace?)
Reserved	36-55	--	Not used (reserved)
Vendor-Defined Message	56-62	Both ways	Not used (reserved)
Vendor-Defined Extension Message	63/0x3F	Both ways	Not used (reserved)

Short explanation of different messages (*Nexus spec* provides a lot of details):

- Ownership Trace (**TCODE=2**) – used to track context switching (when OS/RTOS changes task).
- Program Trace - Direct/Indirect Branch (**TCODE=3, 4**) – simplest form of program trace.
- Data Acquisition (**TCODE=7**) – used for instrumentation trace (for different purposes).
- Error (**TCODE=8**) – reports different overrun error conditions.
- Program Trace - ... (**TCODE=9, 11, 12**) – different forms of program trace synchronization.
- Watchpoint Hit (**TCODE=15**) – records watch-points (from core) events in trace stream.

- Program Trace - Indirect Branch History ... (**TCODE=28, 29**) –compression of direct branches.
- Program Trace - Repeat Branch (**TCODE=30**) – very good compression of loops in code.
- Program Trace - Correlation (**TCODE=33**) – correlate trace flow with core run/stop events.

10. Nexus trace messages – details

TODO: This chapter should list all Nexus messages so this document can be used without looking at (complex!) descriptions in *Nexus spec*.

11. Compression Considerations

1. In general Nexus provides 'N-instructions + destination address' packets.
 - a. Direct (known from code address) branch is not encoded.
 - b. There is an option to consider non-conditional direct branch as linear instruction.
 - c. There is an option to consider short call/return sequences (return is unknown, but if call is known return may be assumed as returning to instruction after call).
2. Nexus uses 2-bit MSEO field to encode messages – this will affect compression.

12. Trace Configuration and Control

1. *Nexus spec* does not mandate, but suggests some ways of controlling trace.
2. This specification will try to follow *Nexus spec*, but because of lack of debug access it may be difficult.
3. It is highly desired (although not compulsory) to:
 - 3.1. Trace control to be always available (regardless of state of traced core[s]).
 - 3.2. Trace tool may restrict access to trace controls from inside of SoC as modifying trace settings by running code can be confusing to trace tool – for example if code changes width of trace port, trace tool will not be able to capture it.
 - 3.3. Configuring trace via code is sometimes desired, but trace tool must be prepared for it.

13. Nexus Port Interfaces (overview)

1. *Nexus spec* [SECTION 6] define three trace port interfaces
 - 1.1. Parallel Aux - Supported (output-only direction)
 - 1.2. TAP Aux - Replaced by ability to store trace into RAM, which can be read via TAP.
 - 1.3. High Speed Seral Aux - Aurora Gbit serial protocol (considered for future).
2. This specification define the following trace sinks (Nexus term Aux=auxiliary is not fortunate as it is mandated by this specification), so in order to clarify:
 - 2.1. PTS (**P**arallel **T**race **S**ink) - N-bit (1/2/4/8/...) parallel interface.
 - 2.2. STS (**S**erial **T**race **S**ink) - Relatively slow (UART and Manchester) serial wire.
 - 2.3. RTS (**R**AM **T**race **S**ink) - Dedicated (small) RAM or part of (big) system RAM.
 - 2.4. HTS (**H**igh **S**peed **S**erial **T**race **S**ink) - Xilinx Aurora compliant.

- 2.5. CTS (Custom Trace Sink) - Custom trace sink for export via Arm's ATB or USB.

TODO: Instead of 'Trace Sink' we can use 'Export Port', PEP/SEP/REP/HEP/CEP.

- 3. Trace port calibration
 - 3.1. For good quality of trace capture it is desirable to have the ability to force SoC to send known patterns on external (STS/PTS) trace ports.
 - 3.2. It will allow trace probe to check validity of capture and adjust trace capture logic.
 - 3.3. Advanced trace probes may allow calibration of trace capture by delaying some signals or adjust capture threshold levels.

14. Nexus MSEO/MDO

- 1. Nexus messages are encoded as two logically parallel streams of data.
 - MSEO - 2-bit field for detection of idle/start of message/ variable size fields.
 - MDO - N-bit field which carries payload of message (6-bit TCODE followed by other TCODE-dependent fields: addresses, counters, statuses etc.).
- 2. **Nexus spec** permits 1-bit MSEO (being sequence of 2 bits ...), but in order to reduce complexity (on both SoC and trace tool sides) this specification is not allowing it.
 - 2.1. STS (Serial Trace Sink) and PTS (Parallel Trace Sink) chapters define how single-bit transport is handled.
- 3. **Nexus spec** permits any number of MDO bits, but for simplicity this specification permits 'even' number of MDO bits, so entire Nexus message will be always N*8 bits (i.e. N bytes) long.
 - 3.1. Handling generic bit-sized in trace decoding software would be slow and kind of nightmare.
 - 3.2. Said so, permitted supported MDO sizes will be 6/14/22/30-bit + 2bit MSEO (1/2/3/4-byte).
 - 3.3. Bigger MDO widths have less MSEO-related overhead, but from other hand the necessary padding (due to fact that all fields must be MDO bits-aligned) may nullify any gain.
 - 3.4. Said so it is **strongly recommended** to use MSEO=2 and MDO=6 configuration. If case of wider export port (16/32-bit), several Nexus bytes (possibly from different Nexus messages will be packed together). **TODO:** Should we consider only perming 2+6 configuration?
 - 3.5. PTS (Parallel Trace Sink) chapter below details how MSEO/MDO are mapped into different number of pins used to send the trace out of SoC.

15. RTS (RAM trace Sink)

- 1. **TODO:** It should cover both dedicated and system (shared with APP) sinks.
 - 1.1. It is essential to provide 'stop capture on full' and 'interrupt on nearly full' and maybe even 'stop CPU on full'.

16. PTS (Parallel Trace Sink)

- 1. **TODO:** This should be main mode of operation. For many smaller devices, it should be enough to use 1/2/4-bit trace. This will allow compatibility with trace probes designed to work with Cortex-M class devices.

17. STS (Serial Trace Sink)

1. This type of trace sink will provide (relatively) slow serial wire trace in two modes:
 - 1.1. SWO mode (available for Cortex-M cores) was big game-changer (for smaller devices).
 - 1.2. UART (1-start, 8-dat and 1-stop bit) – this may allow less advanced tools (and even UART-USB adapters) to capture some forms of trace.
 - 1.3. Manchester encoding – will allow more robust capture as receiver will be able to synchronize with variable trace port frequency.
 - 1.4. Speed limit on that port may be limiting factor.
2. STS will (most likely ...) not provide enough bandwidth to allow full PC trace, but certain limited (but still very usable trace) can be send and captured via that port.
 - 2.1. Trace infrastructure in SoC should not prevent it – it is up to trace control software to limit bandwidth from all trace sources, so it will be possible to send out via STS.
 - 2.2. Some implementations may only offer STS and in the same time significantly ‘trim’ what trace encoder is capable of producing.

18. HTS (High Speed Serial Trace Sink)

1. **TODO:** This should be referring to elaborated Aurora chapter in *Nexus spec*.

19. CTS (Custom Trace Sink)

1. **TODO:** It would be possible to use different transport of trace out of the SoC.
 - a. ARM’s ATB export (with dedicated ID)
 - b. Possible USB end-point export (length of packet may need to be known upfront).
2. Should we define some ‘generic’ glue-interface (FIFO?)

20. Physical Trace-capable Connectors

1. This part should be shared with export of trace packets as defined by Processor Trace TG.
2. **Nexus spec [Table A-1—Connector Part Numbers (Target)]** define 3 basic types of connectors and 2 high speed connectors.
 - 2.1. As Nexus defines very generic/elaborated debug interface (with nTRST/EVT-in/EVT-out), smallest (26-pin!) connector only permits 1-bit trace.
 - 2.2. 20-pin high-speed connector provides 4-bit parallel trace but it is very expensive.
 - 2.3. As this specification is designed to allow reach trace from systems with small cores (for IoT-type applications) less expensive and smaller form-factor connectors must be permitted.
3. The following connectors are primary trace connectors, which should satisfy both low-end and high-end trace needs:

Connector	Debug Connection	RAM Sink	Serial Sink	Parallel Sink
MIPI10	JTAG	Yes	No	No
MIPI10	cJTAG	Yes	Yes (*1)	No
MIPI20	JTAG	Yes	Yes (*2)	1/2/4-bit (*2)
MIPI20	cJTAG	Yes	Yes (*1/*2)	1/2/4-bit (*2)
Mictor38	JTAG	Yes	Yes (*3)	1/2/4/8/16-bit (*3)
Mictor38	cJTAG	Yes	Yes (*1/*3)	1/2/4/8/16-bit (*3)

NOTES

- (*1) - JTAG TDO (unused in cJTAG mode) is re-purposed as STS trace signal.
JTAG TDI line may be used as additional probe → target signal (trigger or serial input).
 - (*2) - Either 4-bit parallel trace or STS and 1/2-bit parallel trace.
Unused trace lines may be used as additional target → probe signals (trigger, UART)
Advanced trace probes may permit unused trace lines as probe → target signals.
4. Allowing above connectors does not disallow any connectors which are defined in **Nexus spec**.
 - 4.1. This may however be only applicable to some use-cases – both MIPI20 and Mictor38 are used to capture trace from Arm cores.
 5. This specification does not address the following:
 - 5.1. Different (VREF) reference voltage for JTAG/cJTAG and trace (potentially Nexus signal VSTBY).
 - 5.2. nTRST signal (it is not part of RISC-V Debug Spec) – Nexus defines it on all connectors.
 - 5.3. Externally provided trace reference clock – it is not defined by any of Nexus connectors.
 - 5.4. Possible use of LVDS (it may be better to use 2-bit LVDS instead of 4-bit single-ended signals) – not defined by Nexus.
 6. **TODO:** Physical pinout of trace connectors and different options for pins for JTAG/ cJTAG/ STS/ PTS options for each connector.

21. Timestamping

1. **TODO:** *Nexus spec* allows 'TSTAMP' (timestamp) field.
2. Cycle-accurate trace is in contradiction with high-compression.

22. Multi-source trace

1. *Nexus spec* allows 'SRC' (source) field which define source of each nexus message.
2. **TODO:** How to deal with multiple harts, encoders etc.

23. Limitations of this Specification

1. Aurora & Custom Sink are not detailed but specification should allow detecting and enabling.
2. Electrical and timing of trace signals is not detailed as we must be dealing with different flavors of implementation (from implementation on FPGA boards to true silicon).
 - 2.1. Electrical specification from Nexus spec should be used as 'strong suggestions' – each implementation should address possible discrepancies.
3. Cooperation with other Nexus traces (on same system) is not detailed.

24. Reference encoder/decoder implementation

1. This specification will be augmented by two reference software modules (on GIT here: **TODO**).
2. Trace Encoder which will take a file with PC sequence (+small extra details like sizes/types of instructions) and will produce text file with encoded trace packets.
 - 2.1. Such a file can be easily produced by simulator.
 - 2.2. It will be also possible to specify a trim of implementation on SoC side.
3. Trace Decoder, will take file with encoded trace and produce PC sequence.
 - 3.1. Running these two will allow validating of entire flow as resultant PC sequence must be same as input PC sequence.
 - 3.2. Validation of encoder implemented on SoC side will be also possible.
4. In some point of time reference encoder implementation may be based on ingress port.
 - 4.1. It can be done in C/C++ or in VHDL.

25. Compliance testing/consideration

1. **TODO:** We should define what a compliant implementation is (both encoder/decoder) and how to check it.

26. To be addressed in future

1. **TODO:** There will be something here for sure

27. Comparing features with Arm trace

1. **TODO:** Arm is most popular and best known trace by far. RISC-V should offer comparable features and this chapter should address it (perhaps in form of a table).