RISC-V Trace & Diagnostic Data Transport Encapsulation

Authors: Iain Robertson

Version v1.0.1, 2023-09-08: Draft

Table of Contents

Preamble

This document is in the Development state

Expect potential changes. This draft specification is likely to evolve before it is accepted as a standard. Implementations based on this draft may not conform to the future standard.

Change Log

PDF generated on: 2023-09-08 10:56:22 +0100

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

- Iain Robertson (Siemens) <iain.robertson@siemens.com> - author
- Michael Schleinkofer (Lauterbach) - reviews

Chapter 1. Introduction

The Efficient Trace for RISC-V (E-trace) standard defines packet payloads for instruction and data trace but does not fully define how this should be encapsulated for transport, nor how instruction and data trace should be differentiated. Chapter 7 gives some illustrative examples but this is insufficiently detailed and informative only.

Although the primary motivation for developing this standard was to define an encapsulation format for E-Trace packets, the encapsulation format defined in this document is agnostic to the packet payload structure and meaning and so can be used for any kind of data. In addition to processor trace, it could also be used for a wide variety of other uses, for example: performance counter metrics, trace or other diagnostic data from a bus fabric monitor or on-chip logic analyser.

This specification defines an encapsulation format suitable for use with a variety of transport mechanisms, including but not limited to AMBA Advanced Trace Bus (ATB) and Siemens' Messaging Infrastructure.

1.1. Glossary

- **ATB** - Advanced Trace Bus, a protocol described in ARM document IHI0032B;
- **E-Trace** - Abbreviation for Efficient Trace for RISC-V;
- **PIB** - Pin Interface Block, a parallel or serial off-chip trace port feeding into a trace probe, as defined in the RISC-V Trace Control Interface Specification;
- **Trace Encoder** - Hardware module that accepts execution information from a hart and generates a stream of trace packets;
- **TFP** - Trace Formatter protocol, a trace framing protocol described in ARM document IHI0029E. Also adopted by MIPI as Trace Wrapper Protocol (TWP);
- **TWP** - See **TFP**.

Chapter 2. Packet Encapsulation

Two types of encapsulation are defined: *normal* and *null*. A transmitted stream of encapsulated packets comprises a mixture of *normal* and *null* packets. Each packet is atomic, and must be transmitted in its entirety before another packet can be sent.

2.1. Normal Encapsulation Structure

The normal encapsulation structure is comprised of four field-groups as shown in Table 1. In this and the following tables, the field-groups and fields are listed in transmission order: the uppermost field-group or field in a table is transmitted first, and multi-bit fields are transmitted LSB first.

Table 1. Encapsulation Field Groups

Group Name	# Bits	Description
header	8	Encapsulation header. See Section 2.1.1.
srcID	0 - 16	Source ID. See Section 2.1.2.
timestamp	T*8	Time stamp. See Section 2.1.3.
payload	1-248	Packet payload. See Section 2.1.4.

The groups are concatenated together with the LSB of the **header** group transmitted first. The groups are defined in the following tables:

2.1.1. Header

The header is a single byte comprising the fields defined in Table 2.

Table 2. Header Fields

Field Name	# Bits	Description
length	5	Encapsulated payload length (1 to 31) bytes.
flow	2	Flow indicator. This can be used to direct packets to a particular sink in systems where multiple sinks exist, and those sinks include the ability to accept or discard packets based on the flow value.
extend	1	Indicates presence of timestamp when 1. Reserved if timestamp width is 0.

2.1.2. srcID

The **srcID** field identifies the source of the packet. It can be between 0 and 16 bits in length. This length must be fixed and discoverable for a given system.

The **srcID** may be omitted (i.e. zero bits in length) if there is only one source in the system, or if the transport scheme includes a sideband bus for the source ID (for example, ATB).

When present, an 8-bit **srcID** will be sufficient for most use cases, and is simplest in terms of determining the packet length, keeping all field groups aligned to byte boundaries. However, the length of the field can be reduced to improve efficiency for small systems, or increased if required for larger systems. See also Section 2.1.5.

2.1.3. Timestamp

The **timestamp** field provides a means to include time information with every packet. It is included in the encapsulation if **header.extend** is 1. When included, the timestamp must be T bytes in length. The length must be discoverable, and fixed for a given system.

Timestamps may be omitted either because time is not of interest to the user, or if time information is already included within the encapsulated payload.

2.1.4. Payload

The encapsulation payload can be up to 248 bits (31 bytes) in length, and comprises the fields shown in table Table 3.

Table 3. Payload Fields

Field Name	# Bits	Description
type	≥ 2	Packet type. Default length is 2 bits. May optionally be increased to provide for more than 3 data packet types per source. Length must be fixed for a given **srcID**, and discoverable if > 2. 0: control packet other values: data packet of some sort (source type dependent) Control packets may be used in some applications (for example, the Siemens message infrastructure) for accessing configuration state.
trace_payload	≤ R	Packet payloads such as those defined for E-Trace. Maximum value of R is defined as 248 - Y - **srcID**%8 where Y is the length of the **type** field.

2.1.5. Packet Length

Encapsulated packets are a number of whole bytes in length, the exact number depending on the sizes of the **srcID**, **timestamp** (if present) and **header.length**:

Packet length = 1 + S + (T * **header.extend**) + **header.length**

S and T are discoverable constants; S is the number of whole bytes of **srcID**: int(#bits(**srcID**)/8).

For the case where the size of **srcID** is a multiple of 8 bits, **header.length** is simply the number of bits of payload rounded up to the nearest multiple of 8 and expressed in bytes:

header.length = ceiling(#bits(**payload**)/8)

However, if the **srcID** is not a multiple of 8 bits the remaining **srcID** bits not accounted for by 'S' are instead included when determining the value of **header.length**. Thus the more general definition for any **srcID** size is:

header.length = ceiling((#bits(**payload**) + #bits(**srcID**)%8)/8)

2.2. Null Encapsulation

For *normal* encapsulation, the **header.length** field is at least 1, and the overall length of the encapsulated packet will be at least 2 bytes (**header** plus 1 byte of payload).

A *null* packet is defined as a single **header** byte with a zero **length** field. This results in up to 8 different types of null packet, which are defined as follows:

- 0x00: *null.idle*
- 0x80: *null.alignment*
- Others: reserved

A *null* packet is inserted in the data stream when there are no *normal* encapsulated packets to send.

2.3. Synchronization

In a data stream comprised of packets, it's a requirement to be able to determine where packets start and end, when starting from an arbitrary point, without knowledge of the full packet history. This can be achieved by inserting a synchronization sequence into the packet stream periodically. This sequence is comprised of a sufficiently long sequence of *null* packets.

A 'null' byte is defined as a byte with the 5LSBs all zero, which may be a *null* packet, or may be part of a *normal* packet. The longest run N of 'null' bytes possible within a packet is:

N = 31 + T + S (see Section 2.1.3 and Section 2.1.5 for definitions of T and S respectively)

Therefore, in a sequence of N or more 'null' bytes, the first N 'null' bytes may actually be part of a packet. However, any 'null' bytes after this must be *null* packets, and the 1st non-null byte seen after this must therefore be the 1st byte of a *normal* packet.

For unframed data streams such as PIB, a *null.alignment* packet must be transmitted as the final *null* before a *normal* packet. Strictly speaking this is necessary only if the data stream is sent via an interface less than 8 bits wide, but for simplicity this is mandatory for any width. The single 1 at the end of this sequence uniquely identifies the byte boundary, and what follows as the start of a packet. For example, for two *normal* packets with M *nulls* between them, this would comprise M-1 *null.idles* and 1 *null.alignment* (M > 0).

For framed data streams which incorporate synchronization information in their own framing such as MIPI TWP (aka ARM Trace Formatter Protocol) or USB there is no requirement to include *null.alignment* packets.

The synchronization requirements are summarized in the following rules:

- A synchronization sequence must have a length of N+1 bytes (N defined above), comprising:
  - For unframed data streams, N consecutive *null.idle* packets, directly followed by one *null.alignment* packet;
  - For framed data streams, N consecutive *null.idle* packets, directly followed by one *null.idle* or *null.alignment* packet.

For writing unframed data to memory, alternative synchronisation mechanisms may also be employed. For example, by dividing memory into blocks of known size, and requiring that packets do not straddle block boundaries. The first byte of every block will therefore be the start of a packet. Details of such schemes are out of scope of this specification.

Chapter 3. Packet Encapsulation for E-Trace

Chapter 2 describes the packet encapsulation in general terms. This chapter describes how that applies in the context of E-Trace.

The RISC-V Trace Control Interface Specification includes several fields relevant for the construction and synchronization of packets, as described in the following sections.

3.1. srcID

- **trTeInhibitSrc** in the *trTeControl* register of a trace encoder indicates whether a **srcID** is present or absent in the encapsulated packets;
- **trTeSrcBits** in the *trTeInstFeatures* register indicates the length of the **srcID**;
- **trTeSrcID** in the *trTeInstFeatures* register indicates the value of the **srcID** associated with that particular source.

3.2. timestamp

- **trTsEnable** in the *trTsControl* register enables the inclusion of the **timestamp** field in the encapsulated packets (i.e. **header.extend** will be set based on the value of this bit);
- **trTsWidth** in the *trTsControl* register indicates the number of bits in **timestamp** fields.

Note that some E-Trace packets may optionally include a **time** field, as an alternative method of providing time information. Presence or absence of this is fixed for a given system based on a discoverable parameter, but is not run-time configurable.

3.3. type

For E-Trace, the **type** field in the **payload** group is 2 bits long with the following encodings defined:

- 10: Instruction trace packet
- 11: Data trace packet

3.4. Synchronization

- **trPibAsyncFreq** and **trRamSinkAsyncFreq** in the *trPibControl* and *trRamControl* registers respectively are used to determine the interval between insertion of synchronization sequences.