RISC-V Trace & Diagnostic Data Transport Encapsulation

Authors: Iain Robertson

Version v1.0.0, 2023-08-08: 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: 2023-08-08 17:24:18 +0100

Copyright and license information

Contributors

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

- Iain Robertson <iain.robertson@siemens.com>

Chapter 1. Introduction

The Enhanced Trace for RISC-V standard defines packet payloads for instruction and data trace but does not fully define how this should be encapsulated for transport. 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 Enhanced Trace for RISC-V packets, the encapsulation format is agnostic to the packet payload structure and meaning and so can be used for any kind of data. For example: counter metrics, trace or other diagnostic data from a bus fabric monitor or on-chip logic analyser. It is not restricted to processor trace.

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' Embedded Analytics Messaging Infrastructure.

Chapter 2. Packet Encapsulation

Two types of encapsulation are defined: **normal** and **null**.

2.1. Normal Encapsulation Structure

The normal encapsulation structure is comprised of four field-groups:

Table 1. Encapsulation Field Groups

Group Name	# Bits	Description
header	8	Encapsulation header. See Header Fields table.
srcID	0 - 16	Source ID. See srcID Field table.
timestamp	T*8	Time stamp. See timestamp Field table.
payload	1-248	Packet payload. See Payload Fields table.

The groups are defined in the following tables:

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.

Table 3. srcID Field

Field Name	# Bits	Description
srcID	0 - 16	Identifies the source of the packet. The **srcID** may be omitted (i.e. width = 0) 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, MIPI). If there is no sideband bus for the source ID (as is the case for the Siemens messaging infrastructure), and there is more than one source, the **srcID** field must be non-zero length. 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.

Table 4. timestamp Field

Field Name	# Bits	Description
timestamp	T*8	A Timestamp 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. This provides a means to include time information with every packet. 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 (for example Enhanced Trace for RISC-V sync-start, sync-trap and sync-context packets).

Table 5. 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 is > 2. 0: control packet other values: data packet of some sort For E-Trace packets, 2 indicates instruction trace and 3 indicates data trace. 1 may be used for another type of data, such as performance counters. For other types of source, the mapping is 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.

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 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: *Idle*
- 0x80: *Alignment*
- Others: reserved

An *idle* 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. The longest run of 'null' bytes (i.e. bytes with the 5LSBs all zero) possible within a packet is: N = 31 + T + S (T = **timestamp** width in bytes, S = **srcID** width in int(bits/8)). Therefore, in a sequence of N or more 'null' bytes, at least the final 'null' byte must be a null packet. 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 which may be transported on interfaces less than 8 bits wide, an alignment null must be transmitted as the final null before a 'normal' packet. 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 *idles* and 1 *alignment*
- The synchronization sequence will comprise N *idles* followed by 1 *alignment*

For framed data streams which incorporate synchronization information in their own framing such as MIPI WTF (aka AMBA TPIU) or USB there is no requirement to include alignment nulls.

For writing 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.

Bibliography