Arm® CoreSight Base System Architecture 1.0

Arm Platform Design Document

Non-confidential

Document number: ARM-DEN-0068

Contents

Release information

Date	Version	Changes
2018/Jul/23	1.0	• First Non-Confidential release.

Non-Confidential Proprietary Notice

1 About this document

1.1 Terms and abbreviations

Term	Meaning
ARE	Affinity Routing Enable (GICv3 [1]).
Arm ARM	Arm Architecture Reference Manual; see [2] and [3].
Base Server System	A system compliant with the Server Base System Architecture.
CTI	Cross Trigger Interface, see [3].
ETB	Embedded Trace Buffer. A component used to capture trace into dedicated memory. See [4].
ETF	Embedded Trace FIFO. A superset of the ETB with the additional ability to use the dedicated memory as an inline FIFO. See [4].
ETR	Embedded Trace Router. A component used to capture trace into system memory. See [4] and [5].
GIC	Generic Interrupt Controller.
I/O Coherent	A device is I/O Coherent with the PE caches if its transactions snoop the PE caches for cacheable regions of memory. The PE does not snoop the device's cache.
LPI	Locality-specific Peripheral Interrupt (GICv3 [1]).
PE	Processing Element, as defined in the Arm ARM. Typically a single hardware thread of a PE.
PMU	Performance Monitor Unit.
PPI	Private Peripheral Interrupt.
Program	A Program is a software entity and all software invoked by that software entity at the same Exception level or lower. For example, each of these is considered a Program: • A single application. • A single operating system and all applications invoked by that operating system. • A hypervisor and all operating systems invoked by that hypervisor.
SBBR	Server Base Boot Requirements [6].
SBSA	Server Base System Architecture.
SGI	Software Generated Interrupt.
SPI	Shared Peripheral Interrupt.
SRE	System Register interface Enable (GICv3 [1]).
STM	System Trace Macrocell, see [7].
System	A System is the combination of hardware, firmware, and software that are needed to run a Program. This typically includes all of the PEs, memory, and peripherals. Furthermore, a System might not be limited to a single chip, should multiple chips be used to run a Program.
System firmware data	System description data structures such as ACPI or FDT.
Trace unit	A logical component which generates a trace stream from a PE.
VM	Virtual Machine.

1.2 References

This section lists publications by Arm and by third parties.

See Arm Infocenter (http://infocenter.arm.com) for access to Arm documentation.

[1] *ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0.* (ARM IHI 0069) Arm Ltd.

[2] *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition.* (ARM DDI 0406) Arm Ltd.

[3] *ARM® Architecture Reference Manual ARMv8, for the ARMv8-A architecture profile.* (ARM DDI 0487) Arm Ltd.

[4] *CoreSight Trace Memory Controller Technical Reference Manual.* (ARM DDI 0416) Arm Ltd.

[5] *ARM® Embedded Trace Router Architecture Specification.* (ARM IHI 0081) Arm Ltd.

[6] *Server Base Boot Requirements, System Software on ARM® Platforms.* (ARM DEN 0044) Arm Ltd.

[7] *ARM® System Trace Macrocell Programmers Model Architecture.* (ARM IHI 0054) Arm Ltd.

[8] *Server Base System Architecture.* (ARM DEN 0029) Arm Ltd.

[9] *ARM® Embedded Trace Macrocell Architecture Specification ETMv4.* (ARM IHI 0064) Arm Ltd.

[10] *ARM® Debug and Trace Configuration and Usage Models.* (ARM DEN 0034) Arm Ltd.

[11] *ARM® CoreSight System-on-Chip SoC-600 Technical Reference Manual.* (100806) Arm Ltd.

[12] *ARM® CoreSight Architecture Specification.* (ARM IHI 0029) Arm Ltd.

1.3 Rules-based writing

This specification consists of a set of individual rules. Each rule is clearly identified by the letter R.

Rules must not be read in isolation, and where more than one rule relating to a particular feature exists, individual rules are grouped into sections and subsections to provide the proper context. Where appropriate, these sections contain a short introduction to aid the reader. An implementation which is compliant with the architecture must conform to all of the rules in this specification.

Some architecture rules are accompanied by rationale statements which explain why the architecture was specified as it was. Rationale statements are identified by the letter X.

Some sections contain additional information and guidance that do not constitute rules. This information and guidance is provided purely as an aid to understanding the architecture. Information statements are clearly identified by the letter I.

Implementation notes are identified by the letter U.

Software usage descriptions are identified by the letter S.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Rules, rationale statements, information statements, implementation notes and software usage statements are collectively referred to as *content items*.

1.3.1 Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

- Content items are assigned random alphabetical identifiers (*HJQS*, *PZWL*, . . .).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

1.3.2 Examples

Below are examples showing the appearance of each type of content item.

R	This is a rule statement.
R_{X001}	This is a rule statement.
I	This is an information statement.
X	This is a rationale statement.
U	This is an implementation note.
S	This is a software usage description.

1.4 Feedback

Arm welcomes feedback on its documentation.

If you have comments on the content of this manual, send an e-mail to errata@arm.com. Give:

- The title (CoreSight Base System Architecture).
- The document ID and version (ARM-DEN-0068 1.0).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

2 Background

Arm processors are used in a wide variety of system-on-chip products in many diverse markets. Each market has unique constraints, which means that it is impossible to design a single product that meets all the requirements of every context. The Arm architecture profiles, Application, Real-time, and Microcontroller segment the solutions produced by Arm and align with the varying functional requirements of different target markets. This means that there are substantial differences in the products of the profiles.

The wide-ranging use of these products also means that there is great diversity within an architectural profile, for example, where cost-sensitive markets require hardware changes to reduce silicon area usage. In these cases, it might be cheaper to develop a software solution than it is to accommodate the hardware resource.

In other markets, such as those which require an open platform with complex software, the opposite is true. The savings gained from removing a hardware feature are outweighed by the cost of developing software to support the variants. Furthermore, uncertainty about whether new features are widely deployed can be a substantial impediment to the adoption of those features.

The Arm Application profile, for example, must balance these two competing business pressures. It offers a wide range of features, such as Advanced SIMD and floating point support, and TrustZone system security technology, to tackle an increasing range of problems, whilst also offering the flexibility to reduce silicon space by the removal of hardware features in cost-sensitive contexts. Arm processors are built into a large variety of systems. Aspects of this system functionality are crucial to the fundamental function of system software.

Variability in PE features and certain key aspects of the system impact on the cost of software system development and the associated quality risks.

Base System Architecture specifications are part of Arm’s strategy of addressing this variability.

3 Introduction

3.1 Goals

This document specifies a hardware architecture for a system designed around one or more Armv8-A PEs, where debug functionality running on an Operating System can rely on the hardware resources. It addresses PE features and key aspects of system architecture.

The primary goal is to ensure sufficient system architecture to enable a suitably-built driver and debug software framework to run on all hardware compliant with this specification. It is anticipated that a machine-readable description of the hardware configuration is needed to ensure that the driver and debug software are appropriately configured for the specific system.

Arm does not mandate compliance to this specification, but anticipates that OEMs and software providers will require compliance to maximize out-of-box software compatibility and reliability.

This specification is a companion to the Server Base System Architecture [8], which also defines requirements for PE and system features.

3.2 Levels of functionality

This specification introduces the idea of levels of functionality, where each level provides a specified set of capabilities that software can rely on.

An implementation is consistent with a level of the CoreSight Base System Architecture if it implements all of the functionality of a given level, with performance appropriate for that particular level. This means that all of the functionality of a level can be exploited by software without unexpectedly poor performance.

While the levels are numbered, the numbering scheme does not always mean that software written for a particular level will work on hardware designed for any lower-numbered or higher-numbered level. For example, some higher-numbered levels restrict the options provided at lower-numbered levels. As such, software written only for the higher level might not work with hardware compliant with a lower-numbered level.

3.3 Self-hosted debug capabilities

A suitably-built debug software framework running on hardware compliant with one or more levels of this specification should be able to provide debug functionality consistent with the levels implemented. These capabilities include, but are not limited to:

- Tracing of PE program flow, providing detailed history of program execution. PE trace has multiple uses, including:
  - Post-mortem analysis.
  - Reverse debugging.
  - Performance analysis.
- Performance monitoring, providing detailed information about the performance and timing characteristics of programs running on a PE.

Self-hosted debug capabilities do not require an external debugger to be connected, and do not require external debug capabilities to be enabled.

3.4 External debug capabilities

This document does not specify any capabilities for external debug. It is anticipated that external debug scenarios might use the same hardware functions as self-hosted debug scenarios. However, both external debug software and self-hosted debug software must accommodate the possibility that hardware functions might not be available because they are being used by another debug agent. To provide external debug functions, a system might include functionality or components in addition to those specified, which might require initialization or programming to provide the self-hosted debug capabilities.

This document does not specify any mechanism for software agents to arbitrate over the use of hardware functions.

4 PE Trace

4.1 Introduction

I_{JGQVY} PE trace provides a detailed history of program flow, and is useful for both debugging and performance analysis. The objective of PE trace is to provide a debug software framework such as gdb, Linux perf, or WinDbg, with a history of executed instructions, branches, or function calls.

I_{WCXDS} This specification details the following sets of requirements:

- The trace information in the generated trace and the observation capabilities of the trace functionality in each PE, see Trace Generation (TG) in Section 4.2.
- The methods of capturing the generated trace, see Trace Capture (TC) in Section 4.3.

4.2 Trace Generation (TG)

I_{NLFZN} The levels of Trace Generation (TG) provide the ability to generate a program trace and to set trace filtering trigger conditions.

4.2.1 TG Level 0

I_{LYPCY} TG Level 0 provides basic program flow tracing capability for all PEs in the system, with tracing provided from all PEs concurrently.

R_{ZKTTX} In a system with multiple PEs, each PE that is to be used in the same operating system or hypervisor must be compliant with at least the same TG level.

R_{BGFBV} Each PE in the system must be provided with a trace unit compliant with the ETMv4 architecture, see [9].

R_{FYSHB} The trace unit must support the following ETMv4 features:

- Cycle counting with a cycle counter that is at least 12-bits in size.
- Branch broadcasting.
- Context ID tracing.
- Virtual context identifier tracing, if the PE implements EL2.
- At least one address comparator pair.
- At least one Context ID comparator.
- At least one Virtual context identifier comparator, if the PE implements EL2.
- At least one single-shot comparator control.
- At least one event in the trace.
- At least one external input.
- At least two counters.
- The sequencer state machine.
- At least four resource selection pairs.
- Global timestamping with a size of 64-bits.
- The Virtual context identifier options, if the PE implements the Virtualization Host Extensions.

R_{XZTDG} Every trace unit must implement at least one of memory-mapped access or system-instruction access to the trace unit.

R_{DBDPJ} If memory-mapped access to the trace unit is implemented, access to the address space occupied by the trace unit must be possible without any system-specific software other than that required to provide power to the trace unit.

I_{TMJFX} If a system provides control over the power to the trace unit, Arm recommends this control provides the means to conserve power when the trace unit is not used, including when transitioning between states where trace is used and not used.

I_{YRZQS} Arm recommends that industry standard system firmware data, such as ACPI or FDT, is used to describe the system. Where power control is required, standard power interfaces accessible by the Non-secure world should be used, such as ASL code in ACPI.

R_{CLQFT} If the trace unit implements the CoreSight authentication interface, system firmware must ensure that Non-invasive debug is enabled before any Hypervisor or Operating System are started. If the PE implements EL3, only Non-secure debug is required to be enabled. If the PE does not implement EL3, all Non-invasive debug is required to be enabled. For more details on the authentication interface see [10].

R_{NPCTH} For each event implemented in the trace unit, the event must drive a generic trace external output event which in turn drives a trigger input event to the Cross Trigger Interface (CTI) implemented in the PE. See [3] for more details on the CTI and connectivity.

R_{HJKLK} For up to the first four external inputs implemented in the trace unit, the input must be driven by a trigger output event in the Cross Trigger Interface (CTI) implemented in the PE. See [3] for more details on the CTI and connectivity.

R_{RYBNG} All trace units must share the same timestamp source, which is one of the following:

- A dedicated CoreSight timestamp counter.
- The system counter in accordance with the Generic Timer in the Arm architecture.

R_{VNYCM} The timestamp source must either be able to be programmed to start by the debug software framework, or must be running before the debug software framework is started.

I_{MTQMS} Arm recommends that the timestamp source is running before the debug software framework is started.

R_{CXTGB} The trace units for all PEs must be able to be enabled concurrently, and generate trace concurrently.

I_{CJJQG} Arm recommends that the following PMU events are provided as external inputs to the trace unit:

- All PMU events required by the Arm architecture.
- All implemented Common architectural events.

For PMU events where more than one occurrence of the event can occur simultaneously, Arm recommends that a single external input is provided for the event, and this external input is asserted once per cycle on which any occurrence of the event occurs.

4.2.2 TG Level 1

I_{STMNC} The Armv8-A Self Hosted Trace Extension [3] provides fine-grained control over when trace is permitted to be generated. The Armv8-A Self Hosted Trace Extension, when combined with the CoreSight Base System Architecture, provides greater control over when trace is generated and how trace is captured, while requiring less intervention from software running at higher levels of privilege.

I_{JBNBR} TG Level 1 extends TG Level 0 to provide additional constraints regarding the generation of trace.

R_{GNQFT} All TG Level 0 rules apply at TG Level 1, except where a TG Level 1 rule contradicts a TG Level 0 rule, whereby the TG Level 1 rule takes precedence.

R_{DQKSW} Each PE must implement the Armv8.4 Self-hosted Trace Extension [3].

R_{ZNYLS} Each trace unit must implement at least version 4.4 of the ETMv4 architecture [9].

R_{TXDFQ} Each trace unit must implement the same version of the ETMv4 architecture.

R_{SMPQQ} Each trace unit must implement the system-instruction interface for access to the trace unit registers.

R_{TRMXZ} The global timestamp source for each trace unit must be the same time source as the Generic Timers in the PE. The Armv8-A Self Hosted Trace Extension provides a mechanism to choose between the physical view of this time value or the virtual view.

4.3 Trace Capture (TC)

I_{GFRSF} The method of capturing PE trace defines the usage models for analyzing the generated trace. This specification defines capture methods providing a minimum of tracing of a single Program, up to tracing all active Programs concurrently. Each Trace Capture (TC) level provides a minimum set of trace capture requirements, but does not preclude other mechanisms from also being implemented in addition to the minimum requirements.

4.3.1 TC Level 0

I_{TGPYP} TC Level 0 provides a capture method to ensure trace from all PEs can be captured, although this method only provides low context-switch and analysis overhead for a single Program. Trace from all PEs is captured in a central buffer of either dedicated memory or shared system memory. However to support trace from more than one concurrent Program usually involves significant software overhead to virtualize the buffer for each Program.

R_{JKKQN} Each trace unit must comply with TG Level 0.

R_{YBQQS} One or more trace buffers must be provided in the system.

R_{XTVKH} The trace from each trace unit must be able to be captured by one or more trace buffers.

R_{LWWRX} The trace for all PEs must be able to be captured concurrently, with the trace for each PE captured in at least one of the trace buffers.

R_{KMMJL} Each trace buffer must be based on one of the following:

- Dedicated memory, based on a configuration of a CoreSight Embedded Trace Buffer (ETB).
- Shared system memory, based on a configuration of the CoreSight Embedded Trace Router (ETR), as defined in [4].
- Shared system memory, based on a configuration of the Embedded Trace Router (ETR) Architecture, as defined in [5].

R_{YTNCB} The trace buffer must support the Circular Buffer mode of operation.

R_{DVSXH} Where the trace buffer is based on shared system memory, the trace buffer must be able to be located in any normal memory accessible by the lowest 40 bits of physical address space.

Note

The requirement to support the trace buffer in the memory accessible by the lowest 40 bits of physical address space is due to the CoreSight Embedded Trace Router [4] only supporting up to 40 bits of address space. TC Level 2 increases this requirement to support all of normal memory, by requiring an Embedded

Trace Router which can access all of normal memory.

R_{JDLNQ} Where the trace buffer is based on shared system memory and supports a size greater than the smallest translation granule of the PEs, a page scattering mechanism must be provided to support the trace buffer being partitioned into separate physical pages each of which is no larger than the smallest translation granule of the system.

R_{XTXXM} Where a page scattering mechanism is required, this must be based on one of the following:

- The scatter-gather mechanism integrated into a CoreSight ETR.
- A System Memory Management Unit (SMMU) based on at least SMMUv1.
- The translation service provided by a CoreSight Address Translation Unit [11].

R_{XLYJT} If the PE implements EL2, the page scattering mechanism must support 2 stages of translation using one of the following methods:

- Both stages are implemented using the CoreSight ETR scatter-gather mechanism. The ETR can provide both stages in a single step.
- Both stages are implemented using the CoreSight Address Translation Unit [11]. The CoreSight Address Translation Unit can provide both stages in a single step.
- Both stages are implemented using an SMMU.
- Stage 1 is implemented using either the CoreSight ETR scatter-gather mechanism or the CoreSight Address Translation Unit, and stage 2 is implemented using an SMMU.

R_{CLLYB} Where any part of the page scattering mechanism is implemented using an SMMU, the SMMU infrastructure must support independent streams for each trace buffer.

R_{RTZNK} Where the trace buffer is based on shared system memory, the trace buffer must also support storage of trace into a contiguous physically-address buffer without a page scattering mechanism.

R_{JPHVZ} Where the trace buffer is based on shared system memory and the PE supports both Secure and Non-secure memory, the trace buffer must be able to be located in Non-secure memory, and it is IMPLEMENTATION DEFINED whether the trace buffer is able to be located in Secure memory.

R_{VGQXF} Where a trace buffer can capture trace from multiple trace units, the trace buffer must support packing all of the trace streams in the trace buffer using one of the following:

- The CoreSight formatting protocol, see [12].

R_{KKGTS} Any trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure trace can reach the shared system memory must be accessible without the need for any system specific software other than that required to provide power to the component.

I_{YRZQS} Arm recommends that industry standard system firmware data, such as ACPI or FDT, is used to describe the system. Where power control is required, standard power interfaces accessible by the Non-secure world should be used, such as ASL code in ACPI.

R_{MHFCN} Any trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure trace can reach the trace buffer must be accessible directly in the Physical address space of every PE. In systems where each PE can access both Secure and Non-secure memory, these components must be located in the Non-secure Physical address space.

I_{NGFDQ} If a system provides control over the power to the trace buffer control components, page scattering mechanism, or any components that need to be programmed to ensure trace can reach the trace buffer memory, Arm recommends this control provides the means to conserve power when trace is not used, including when transitioning between states where trace is used and not used.

$\mathrm{R}_{\mathrm{KGDGX}}$ For all components that are involved in ensuring trace reaches the trace buffer, if the component implements the CoreSight authentication interface, system firmware must ensure the authentication interface is set to a condition which ensures trace will reach the trace buffer, before any Hypervisor or Operating System are started. If the PE implements EL3, only Non-secure debug is required to be enabled. If the PE does not implement EL3, debug must be enabled for the implemented security state. For more details on the authentication interface see [10].

$\mathrm{R}_{\mathrm{BNFFR}}$ Only the following programmable components must be involved in ensuring the trace reaches the trace buffer:

- CoreSight ATB Funnel.
- CoreSight ATB Replicator.
- CoreSight Trace Memory Controller, configured as an Embedded Trace FIFO (ETF).

$\mathrm{I}_{\mathrm{YKDQP}}$ Other non-programmable components are permitted to be involved in ensuring the trace reaches the trace buffer, for example any bridging required to cross clock domains or power domains. Such components must not require programming by the debug software framework.

$\mathrm{R}_{\mathrm{MHKBV}}$ For at least one of the events implemented in the trace unit, when this event occurs, any trace buffer which captures trace from this trace unit must be able to detect the event has occurred, and must be able to use this event to stop trace capture. One of the following mechanisms must be used:

- Use of the trigger trace source ID value 0x7D as defined in [12], and use of the ATB Trigger Enable function in the trace unit.
- Use of a dedicated mechanism when Event 0 occurs in the trace unit.

$\mathrm{R}_{\mathrm{SJHVC}}$ If the system uses the trace source ID value 0x7D, the trace buffer must support one of the following:

- Ignoring the trigger trace source ID other than for the purposes of using the event to control trace capture. The CoreSight ETR does not obey this rule.
- Packing of the trace streams such that the trigger source ID and its payload are embedded in the data stored in the trace buffer. The CoreSight ETR supports this operation.

$\mathrm{R}_{\mathrm{QRDDL}}$ Each trace buffer must be able to generate an interrupt to indicate when any of the following occur:

- In Circular Buffer mode, when the top of the trace buffer is reached.
- The trace buffer has reached a pre-programmed watermark fill level.

$\mathrm{R}_{\mathrm{GFPLT}}$ Where a trace buffer can capture the trace from exactly one trace unit, the trace buffer interrupt must be at least one of a PPI, SPI or LPI.

$\mathrm{R}_{\mathrm{VNPCS}}$ Where a trace buffer can capture the trace from more than one trace unit, the trace buffer interrupt must be at least one of an SPI or LPI.

$\mathrm{I}_{\mathrm{RDFGQ}}$ Arm recommends that the trace buffer interrupt is available as an SPI or LPI, to allow a PE other than the PEs being traced to service the trace buffer interrupt.

$\mathrm{R}_{\mathrm{CGZDD}}$ The trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure the trace can reach the trace buffer from the trace unit must not be reset on a Warm reset of the PE. See [3] for more details on Warm resets.

$\mathrm{I}_{\mathrm{SMFPY}}$ Arm recommends that the infrastructure for capturing trace in a trace buffer has sufficient bandwidth for common scenarios, while avoiding trace unit overflow scenarios. Common scenarios for tracing include:

- Continuous tracing of one PE, without cycle counts or branch broadcasting, with minimal impact on PE or system performance.
- Continuous tracing of all PEs concurrently, without cycle counts or branch broadcasting.
- Continuous tracing of one PE, with cycle counting and branch broadcasting enabled.

4.3.2 TC Level 1

I_{TYPNM} TC Level 1 extends TC Level 0 to add an independent trace buffer for each trace unit, based on shared system memory. This capture method ensures trace can be captured from all concurrent Programs with lower context-switch and analysis overhead than TC Level 0, and also normally permits much larger trace buffer sizes due to the re-use of shared system memory.

These rules ensure each trace unit, trace buffer, and the path from trace unit to trace buffer can be completely controlled by a single software entity without need for excessive trapping in a virtualized system.

R_{SWRXP} All TC Level 0 rules apply at TC Level 1, except where a TC Level 1 rule contradicts a TC Level 0 rule, whereby the TC Level 1 rule takes precedence.

R_{VCQHQ} Each trace unit must have a separate logical trace buffer based on shared system memory, based on one of the following:

- Shared system memory, based on a configuration of the CoreSight Embedded Trace Router (ETR), as defined in [4].
- Shared system memory, based on a configuration of the Embedded Trace Router (ETR) Architecture, as defined in [5].

R_{NLXRL} Each trace buffer dedicated for a particular trace unit must only allow trace from that trace unit to be captured.

R_{GLQGC} The path from each trace unit to its trace buffer must not rely on any programmable components that also control the path for a different trace unit to its trace buffer.

R_{PKTJS} Only the following programmable components must be involved in ensuring the trace reaches the trace buffer:

- CoreSight ATB Replicator.
- CoreSight Trace Memory Controller, configured as an Embedded Trace FIFO (ETF).

R_{KJRTJ} The registers to control each trace buffer must be located in separate 64KB pages.

R_{PLBLV} Any components which control the path of trace from the trace unit to the trace buffer and the page scattering mechanism must be located in separate 64KB pages from components which control the path of trace from a different trace unit.

R_{GSDRB} Where the CoreSight Address Translation Unit is used for stage 2 translations, each CoreSight Address Translation Unit must be located in a separate 64KB page from all other trace control components.

I_{ZXMDS} Where the CoreSight Address Translation Unit is used, Arm recommends the CoreSight Address Translation Unit is located in a separate 64KB page from all other trace control components.

R_{ZFFFP} The trace buffer interrupt described in TC Level 0 must be at least one of a PPI, SPI or LPI.

I_{ZYCKM} Arm recommends that the trace buffer interrupt is available as an SPI or LPI, to allow a PE other than the PE being traced to service the trace buffer interrupt.

4.3.3 TC Level 2

I_{HVGCJ} The Armv8-A Self Hosted Trace Extension [3] provides fine-grained control over when trace is permitted to be generated. The Armv8-A Self Hosted Trace Extension, when combined with the CoreSight Base System Architecture, provides greater control over when trace is generated and how trace is captured, while requiring less intervention from software running at higher levels of privilege.

I_{GRNGF} TC Level 2 extends TC Level 1 to provide system memory capture of the trace and additional controls over the ability for other debug agents to access and control the captured trace.

R_{NYWVD} All TC Level 1 rules apply at TC Level 2, except where a TC Level 2 rule contradicts a TC Level 1 rule, whereby the TC Level 2 rule takes precedence.

R_{PNBWT} Each trace unit must have a separate logical trace buffer based on shared system memory, and the trace buffer must be based on the Embedded Trace Router architecture [5].

R_{KPQHG} The trace buffer must be able to be located anywhere in normal memory accessible by the PE.

R_{MFJCK} The trace buffer based on the ETR architecture [5] must be configured as follows:

- Circular Buffer mode is implemented.
- The Address width is large enough to address all of normal memory accessible by the PE.
- One of the following interrupt interrupt strategies:
  - Wired Interrupt controls are implemented.
  - Message-signaled interrupt controls are implemented.
  - No interrupt controls.
- Scatter mode is not implemented.
- All other configuration options are IMPLEMENTATION DEFINED.

I_{TBLTM} Arm recommends that the trace buffer based on the ETR architecture implements one of:

- Wired Interrupt controls.
- Message-signaled interrupt controls.

Note

If no interrupt controls are implemented, the trace buffer still generates an interrupt in accordance with the rules in this specification.

R_{FCGQR} The page scattering mechanism used by each trace buffer must be based on the same mechanism with the same capabilities.

I_{LFBLJ} While each trace buffer has the same mechanism for page scattering, this does not mean each trace buffer must be scattered using exactly the same implementation of the page scattering mechanism. For example, each trace buffer might use an SMMU for page scattering, but each trace buffer does not need to use the same SMMU. Conversely, each trace buffer might use the same SMMU for page scattering and each trace buffer uses a separate stream within that SMMU.

I_{KYBYM} Arm recommends that DEVARCH is implemented in the ETR.

5 PE Debug Capabilities

5.1 Introduction

I_{JSRPL} The Armv8-A architecture provides various debug functions for stepping through execution and performance monitoring. This specification describes the minimum subset of those functions required for compliance. This specification uses some rules defined in [8], however it applies these rules to any PE or system claiming compliance with this specification.

5.2 Specification

5.2.1 PE Level 1

I_{BYFDF} PE Level 1 has the same requirements as SBSA Level 1 for the debug breakpoints, watchpoints, and PMU. SBSA Level 0 compliance is not included in this specification.

R_{RLNVJ} The PMU overflow signal from each PE must be wired to a unique PPI or SPI interrupt with no intervening logic.

R_{FKGPZ} Each PE must implement a minimum of six programmable PMU counters.

R_{HJJND} Each PE must implement a minimum of four synchronous watchpoints.

R_{QGXCN} Each PE must implement a minimum of six breakpoints, two of which must be able to match virtual address, Context ID or VMID.

5.2.2 PE Level 2

I_{HJDRJ} PE Level 2 has the same requirements as SBSA Level 2 or higher for the debug breakpoints, watchpoints, and PMU.

R_{NQLWR} PE Level 2 incorporates all PE Level 1 requirements.

R_{YDSFP} The PMU overflow signal from each PE must be wired to PPI interrupt ID 23 with no intervening logic.

6 System Trace Macrocell

6.1 Introduction

I_{TYQRP} Arm defines a System Trace Macrocell (STM) Architecture [7] which enables tracing of instrumented software and system activity. The STM is used to generate trace for use either by external debuggers or self-hosted debuggers.

I_{LXRYX} Providing an STM implementation in a system provides a memory-mapped programmers model for software instrumentation libraries to target consistently across SoCs.

6.2 Trace Generation (STG)

6.2.1 STG Level 1

I_{MWDGD} STG Level 1 provides a central STM unit for use by instrumentation software running on all PEs in a system.

R_{DWRCS} The system must provide an STM unit compliant with the STMv1.1 architecture as defined in [7].

R_{HJLRB} The STM unit must implement the following options:

- STPv2 trace protocol.
- Absolute timestamping.
- At least one stimulus port master.
- Each master must implement at least 16384 Extended stimulus ports.
- A 64-bit fundamental data size.
- Invariant timing and guaranteed transaction types.
- The following Configuration Register options:
  - STMTSFREQR is read-write.
  - STMTSSTIMR is implemented.
  - STMSYNCR is implemented.
  - STMSPER is implemented.
  - STMSPTER is implemented.
  - Trigger control implements both multi-shot and single-shot.
  - STMSPOVERRIDER is implemented.
  - STMSPMOVERRIDER is implemented.
  - STMSPSCR is implemented.
  - STMSPMSCR is implemented.
  - STMTCSR.SYNCEN is always 0b1.
- Data compression on stimulus ports is either programmable or not implemented.

R_{JWLHW} If any PE implements EL3, the STM unit must implement at least one stimulus port master which is only accessible by Secure software.

R_{KDGSQ} For systems with more than one PE, the system must use stimulus port masters according to one of the following:

- All PEs must be able to use the same stimulus port master.
- Each PE must be able to use a separate stimulus port master.

6.3 Trace Capture (STC)

6.3.1 STC Level 1

I_{ZVVZQ} STC Level 1 provides self-hosted capture of the trace from an STM.

R_{XYYJM} A trace buffer must be provided in the system to capture trace from the STM.

R_{BNHXW} The trace buffer must be based on one of the following:

- Dedicated memory, based on a configuration of a CoreSight Embedded Trace Buffer (ETB).
- Shared system memory, based on a configuration of the CoreSight Embedded Trace Router (ETR), as defined in [4].
- Shared system memory, based on a configuration of the Embedded Trace Router (ETR) Architecture, as defined in [5].

R_{GFNCW} The trace buffer must support the Circular Buffer mode of operation.

R_{KFFNJ} Where the trace buffer is based on shared system memory, the trace buffer must be able to be located in any normal memory accessible by the lowest 40 bits of physical address space.

Note

The requirement to support the trace buffer in the memory accessible by the lowest 40 bits of physical address space is due to the CoreSight Embedded Trace Router [4] only supporting up to 40 bits of address space. TC Level 2 increases this requirement to support all of normal memory, by requiring an Embedded Trace Router which can access all of normal memory.

R_{DWLKD} Where the trace buffer is based on shared system memory and supports a size greater than the smallest translation granule of the PEs, a page scattering mechanism must be provided to support the trace buffer being partitioned into separate physical pages each of which is no larger than the smallest translation granule of the system.

R_{RVZRY} Where a page scattering mechanism is required, this must be based on one of the following:

- The scatter-gather mechanism integrated into a CoreSight ETR.
- A System Memory Management Unit (SMMU) based on at least SMMUv1.
- The translation service provided by a CoreSight Address Translation Unit [11].

R_{PSXCL} If the PEs in the system implement EL2, the page scattering mechanism must support two stages of translation using one of the following methods:

- Both stages are implemented using the CoreSight ETR scatter-gather mechanism. The ETR can provide both stages in a single step.
- Both stages are implemented using the CoreSight Address Translation Unit [11]. The CoreSight Address Translation Unit can provide both stages in a single step.
- Both stages are implemented using an SMMU.
- Stage 1 is implemented using either the CoreSight ETR scatter-gather mechanism or the CoreSight Address Translation Unit, and stage 2 is implemented using an SMMU.

R_{NMSPB} Where any part of the page scattering mechanism is implemented using an SMMU, the SMMU infrastructure must support independent streams for each trace buffer.

R_{MBWMF} Where the trace buffer is based on shared system memory, the trace buffer must also support storage of trace into a contiguous physically-addressed buffer without a page scattering mechanism.

R_{CKVMH}

R_CKVMH Where the trace buffer is based on shared system memory and the PE supports both Secure and Non-secure memory, the trace buffer must be able to be located in Non-secure memory, and it is IMPLEMENTATION DEFINED whether the trace buffer is able to be located in Secure memory.

R_NCMWX Where a trace buffer can capture trace from multiple trace sources, the trace buffer must support packing all of the trace streams in the trace buffer using one of the following:

- The CoreSight formatting protocol, see [12].

R_FBKSC Any trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure trace can reach the shared system memory must be accessible without the need for any system specific software other than that required to provide power to the component.

R_DMNCD Any trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure trace can reach the trace buffer must be accessible directly in the Physical address space of every PE. In systems where each PE can access both Secure and Non-secure memory, these components must be located in the Non-secure Physical address space.

I_VGQLY If a system provides control over the power to the trace buffer control components, page scattering mechanism, or any components that need to be programmed to ensure trace can reach the trace buffer memory, Arm recommends this control provides the means to conserve power when trace is not used, including when transitioning between states where trace is used and not used.

I_YRZQS Arm recommends that industry standard system firmware data, such as ACPI or FDT, is used to describe the system. Where power control is required, standard power interfaces accessible by the Non-secure world should be used, such as ASL code in ACPI.

R_GCWXF For all components that are involved in ensuring trace reaches the trace buffer, if the component implements the CoreSight authentication interface, system firmware must ensure the authentication interface is set to a condition which ensures trace will reach the trace buffer, before any Hypervisor or Operating System are started. If the PE implements EL3, only Non-secure debug is required to be enabled. If the PE does not implement EL3, debug must be enabled for the implemented security state. For more details on the authentication interface see [10].

R_QHCLH Only the following programmable components must be involved in ensuring the trace reaches the trace buffer:

- CoreSight ATB Funnel.
- CoreSight ATB Replicator.
- CoreSight Trace Memory Controller, configured as an Embedded Trace FIFO (ETF).

R_HMBKH For an STM with a dedicated output trigger signal, when this trigger signal is asserted, any trace buffer which captures trace from this STM must be able to detect the event has occurred, and must be able to use this event to stop trace capture. One of the following mechanisms must be used:

- Use of the trigger trace source ID value 0x7D as defined in [12], and use of the ATB Trigger Enable function in the STM.
- Use of a dedicated mechanism when a trigger occurs in the STM.

R_HWRTB If the system uses the trace source ID value 0x7D, the trace buffer must support one of the following:

- Ignoring the trigger trace source ID other than for the purposes of using the event to control trace capture. The CoreSight ETR does not obey this rule.
- Packing of the trace streams such that the trigger source ID and its payload are embedded in the data stored in the trace buffer. The CoreSight ETR supports this operation.

R_NWXNF Each trace buffer must be able to generate an interrupt to indicate when any of the following occur:

- In Circular Buffer mode, when the top of the trace buffer is reached.
- The trace buffer has reached a pre-programmed watermark fill level.

R_{NSYGM} The trace buffer interrupt must be at least one of an SPI or LPI.

R_{PFKGY} The trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure the trace can reach the trace buffer from the STM must not be reset on a Warm reset of the PEs. See [3] for more details on Warm resets.

6.3.2 STC Level 2

I_{YMJFG} STC Level 2 extends STC Level 1 to require an independent trace buffer for the STM, based on shared system memory. This capture method ensures the PE trace and STM trace are captured independently, ensuring trace buffers for PE tracing can be independently managed. STC Level 2 is intended to be used in conjunction with TC Level 1.

R_{XZPWB} All STC Level 1 rules apply at STC Level 2, except where a STC Level 2 rule contradicts a STC Level 1 rule, whereby the STC Level 2 rule takes precedence.

R_{QVGZJ} The STM must have a separate logical trace buffer based on shared system memory, based on one of the following:

- Shared system memory, based on a configuration of the CoreSight Embedded Trace Router (ETR), as defined in [4].
- Shared system memory, based on a configuration of the Embedded Trace Router (ETR) Architecture, as defined in [5].

R_{GXCBK} The trace buffer must be able to be located anywhere in normal memory accessible by the PE.

R_{QQRXP} The trace buffer for the STM must not allow trace from any PE trace unit to be captured.

R_{KVXWW} The path from the STM to its trace buffer must not rely on any programmable components that also control the path for a different trace unit to its trace buffer.

R_{VPWWW} The registers to control each trace buffer must be located in separate 64KB pages.

R_{BDZYC} Any components which control the path of trace from the STM to the trace buffer and the page scattering mechanism must be located in separate 64KB pages from components which control the path of trace from a different trace unit.

R_{CBZTM} Where the CoreSight Address Translation Unit is used for stage 2 translations, each CoreSight Address Translation Unit must be located in a separate 64KB page from all other trace control components.

I_{GTFXB} Where the CoreSight Address Translation Unit is used, Arm recommends the CoreSight Address Translation Unit is located in a separate 64KB page from all other trace control components.

7 Permitted Level Combinations

7.1 Level Combinations

I_{BTFNK} To prevent an excessive cross-product of possible system configurations, this specification permits only a subset of combinations of the levels described.

R_{PHZSJ} A system must be compliant with one or more of the combinations of levels shown in Table 3.

Table 3: Permitted Level Combinations

Combination	TG Level	TC Level	STG Level	STC Level
A	0	0	1	1
B	0	1	1	2
C	1	2	1	2

R_{HRSTJ} STG and STC Levels are optional, but when implemented for a particular combination must be compliant with the levels described.

8 Example Systems Using Arm CoreSight IP

8.1 TC Level 0 examples

8.1.1 Example 1

Example 1 provides a single CoreSight ETB for capturing trace from all trace units.

Table 4: Example 1 compliance

	Compliance Level
Trace Generation (TG)	0
Trace Capture (TC)	0

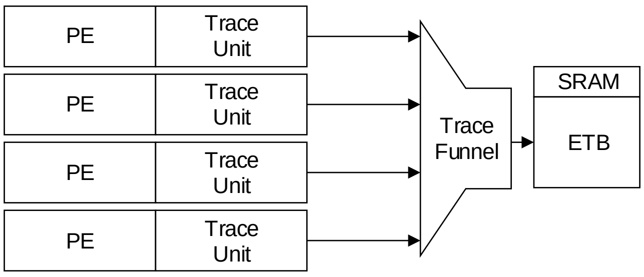

Figure 1: Example 1, with a shared ETB

8.1.2 Example 2

Example 2 provides a single CoreSight ETR for capturing trace from all trace units. The ETR implements the scatter-gather mechanism. An optional SMMU can be provided for Stage 2 translations.

Table 5: Example 2 compliance

	Compliance Level
Trace Generation (TG)	0
Trace Capture (TC)	0

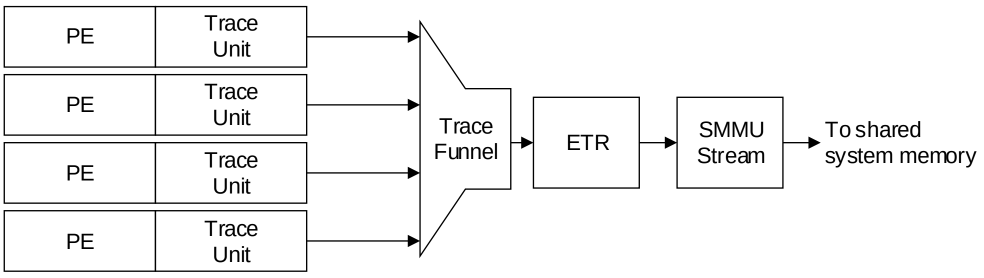

Figure 2: Example 2, with a shared ETR

8.1.3 Example 3

Example 3 provides multiple CoreSight ETFs, where each ETF is able to capture trace from a subset of all trace units. A CoreSight ETF is a superset of a CoreSight ETB with the additional functionality of using the dedicated memory as an inline FIFO. A single CoreSight ETR is also provided for capturing trace from all trace units. The ETR implements the scatter-gather mechanism. An optional SMMU can be provided for Stage 2 translations.

Table 6: Example 3 compliance

	Compliance Level
Trace Generation (TG)	0
Trace Capture (TC)	0

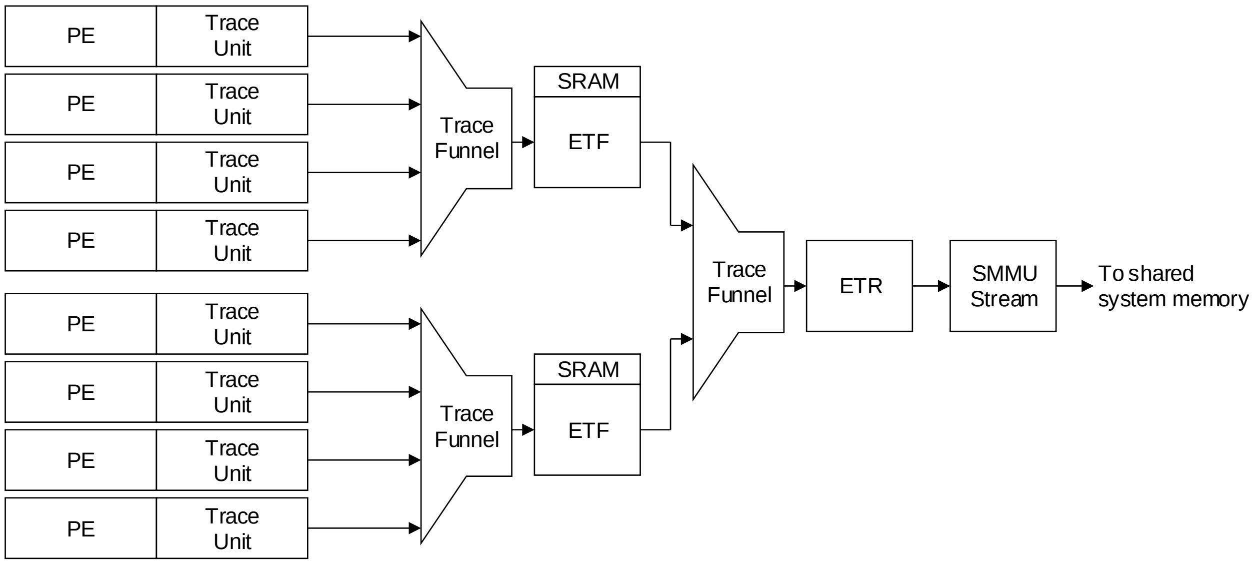

Figure 3: Example 3, with ETFs and a shared ETR

8.2 TC Level 1 examples

8.2.1 Example 4

Example 4 provides an ETR for each PE, allowing trace from each trace unit to be captured independently.

For each of the zones in the example, the components in the zone do not overlap the same 64KB page as any component in another zone. The ETR implements the scatter-gather mechanism. An optional SMMU can be provided for Stage 2 translations, however this might be a shared SMMU.

Table 7: Example 4 compliance

	Compliance Level
Trace Generation (TG)	0
Trace Capture (TC)	1

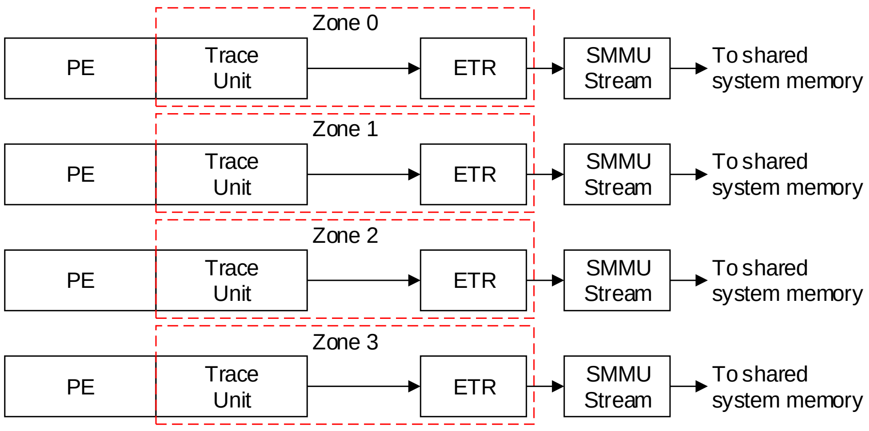

Figure 4: Example 4, with an ETR for each PE

8.2.2 Example 5

Example 5 provides both an ETF and ETR for each PE, allowing trace from each trace unit to be captured independently. This system provides capture in shared system memory while using the dedicated memory of the ETF as an inline buffer.

For each of the zones in the example, the components in the zone do not overlap the same 64KB page as any component in another zone. The ETR implements the scatter-gather mechanism. An optional SMMU can be provided for Stage 2 translations, however this might be a shared SMMU.

Table 8: Example 5 compliance

	Compliance Level
Trace Generation (TG)	0
Trace Capture (TC)	1

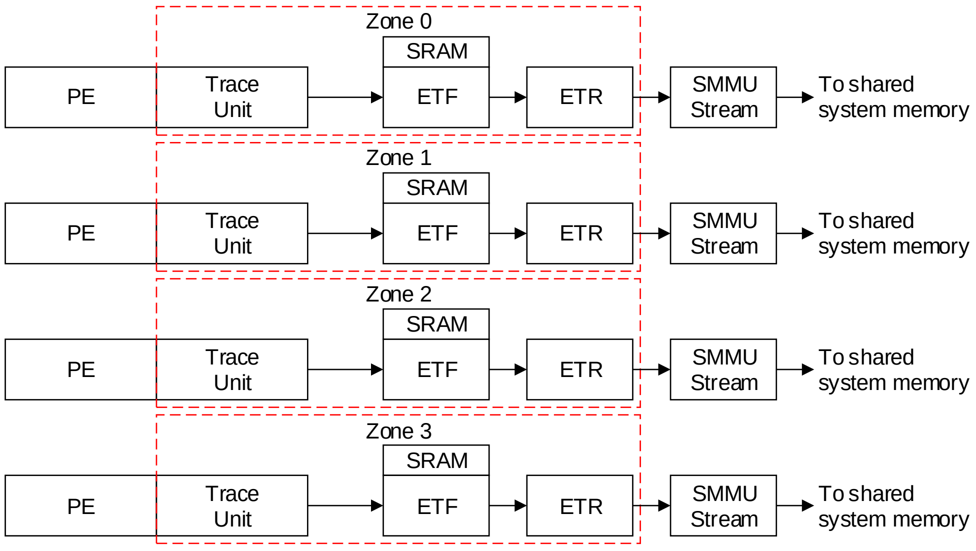

Figure 5: Example 5, with an ETF and ETR for each PE

8.2.3 Example 6

Example 6 provides an ETR for each PE allowing trace from each trace unit to be captured independently. This is similar to Example 4, but with the addition of a shared CoreSight Trace Port Interface Unit (TPIU) for capture by an external debugger. From a compliance perspective, Example 6 adds the replicator component for each trace unit which must obey the TC Level 1 rules for the components on the path to the ETR.

For each of the zones in the example, the components in the zone do not overlap the same 64KB page as any component in another zone. The ETR implements the scatter-gather mechanism. An optional SMMU can be provided for Stage 2 translations, however this might be a shared SMMU.

Table 9: Example 6 compliance

	Compliance Level
Trace Generation (TG)	0
Trace Capture (TC)	1

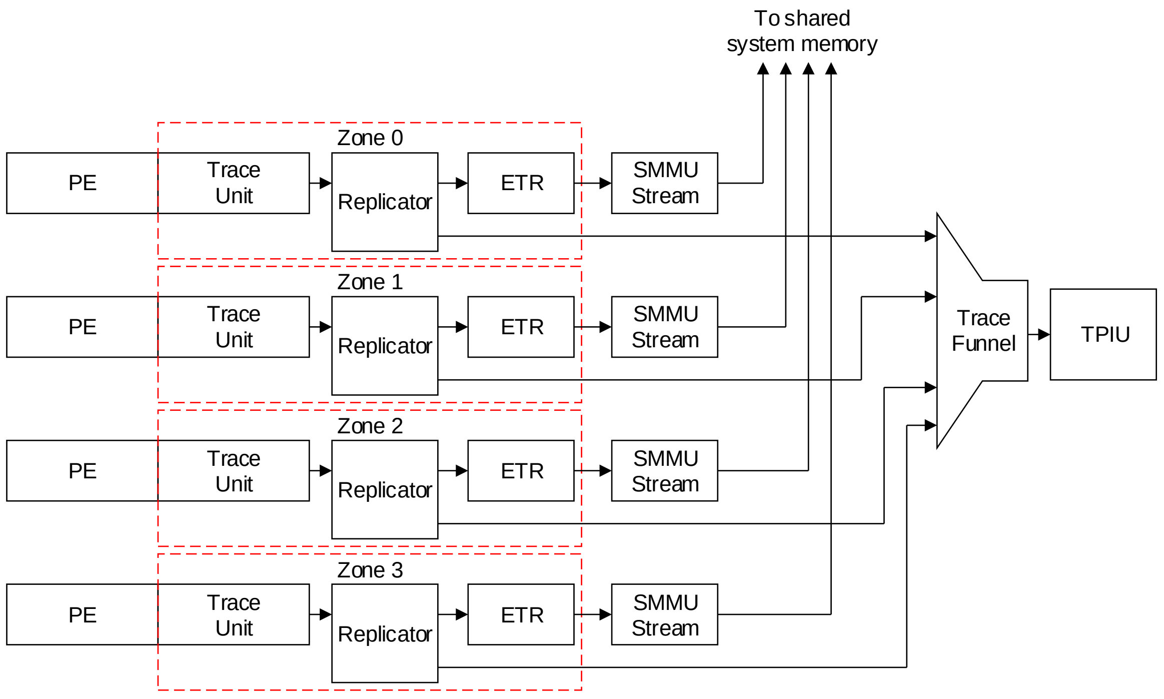

Figure 6: Example 6, with an ETR for each PE, plus a shared TPIU for external capture

8.3 Partial or non-compliance examples

8.3.1 Example N1

Example N1 provides a separate ETR for each PE. When using a CoreSight ATB trace funnel and replicator, the trace might be able to be sent to any ETR and therefore this example is not compliant with TC Level 1 and is only compliant with TC Level 0.

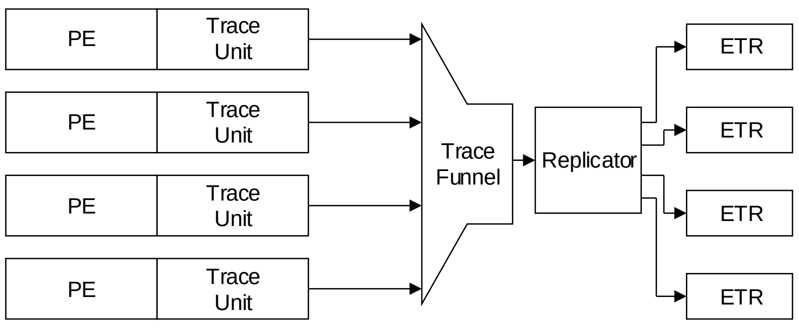

Figure 7: Example N1