Virtual Machine Interface for SPE and TRBE

Version 1.0

Document number	ARM DEN 0154
Document version	00bet0
Document confidentiality	Non-confidential
Document build information	f8e9be401

Important message

Beta release.

Beta quality status has a particular meaning to Arm of which the recipient must be aware. At this quality level the release will be sufficiently stable and committed for initial product development. The recipient can expect some changes to the Beta quality released material.

Virtual Machine Interface for SPE and TRBE version 1.0

Release information

Date	Version	Changes
2025/Apr/15	00bet0	• First release.

Arm Non-Confidential Document License (“License”)

This License is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This License may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this License and any translation, the terms of the English version of this License shall prevail.

The validity, construction and performance of this License shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

Contents

Virtual Machine Interface for SPE and TRBE version 1.0

Preface

Document status

Beta release.

Beta quality status has a particular meaning to Arm of which the recipient must be aware. At this quality level the release will be sufficiently stable and committed for initial product development.

The recipient can expect some changes to the Beta quality released material.

In case of any apparent discrepancy or missing information, please contact Arm Limited.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *Arm® Architecture Reference Manual, for A-profile architecture*. (ARM DDI 0487) Arm Limited.

About this book

I_{HBGWM} This document describes the *Virtual Machine Interface for SPE and TRBE version 1.0*.

Chapter 1 *Problem Description*
An *informative* section describing the need for an interface. Chapter 1 *Problem Description* also defines the goals of the *Virtual Machine Interface for SPE and TRBE version 1.0*.

Chapter 2 *Specification*
A *normative* section that describes the *Virtual Machine Interface for SPE and TRBE version 1.0* the mandatory aspects of the extension.

Chapter 3 *Programmers' Model*
Describes the registers that software uses to communicate the limitations of virtualization support across the *Virtual Machine Interface for SPE and TRBE version 1.0*.

The *Problem description* and *Specification* chapters use Rules-based writing.

D_{NDQCM} This document uses general names for FEAT_SPE or FEAT_TRBE System registers, System register fields, and instructions. Table 3.1 in Chapter 3 *Programmers' Model* provides the disambiguation of these general names.

This document uses terminology defined in Chapter 3.2 *Glossary*.

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration.
- Rule.
- Goal.
- Information.
- Software usage.

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

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

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiiii}

- *L* is a label that indicates the content class of the content item.
- *iiiii* is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behavior.

A Declaration is rendered with the label *D*.

Rule

A Rule is a statement that describes the behavior of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments or queries about our documentation, create a ticket at https://support.developer.arm.com. As part of the ticket, include:

- The title, *Virtual Machine Interface for SPE and TRBE version 1.0*.
- The number, ARM DEN 0154 00bet0.
- The page number(s) to which your comments apply.
- The rule identifier(s) to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Chapter 1
Problem Description

I_SHXBZ FEAT_SPE and FEAT_TRBE write telemetry to in-memory *telemetry buffers*:

- FEAT_SPE writes records for sampled operations to an in-memory *Profiling Buffer*.
- FEAT_TRBE writes trace data to an in-memory *trace buffer*.

The location of the buffer is specified through a pair of System registers, usually as virtual addresses. If the PE encounters an MMU fault when writing to the buffer, then the PE stops collecting the telemetry.

Using virtual addresses enables the operating system to allocate an arbitrarily-sized buffer.

The use of virtual addresses means that to prevent MMU faults, the buffer must be pinned and allocated by all stages of translation. That is:

- Every virtual address in the buffer must be mapped to a physical address.
- The mappings from virtual to physical addresses are not changed while debugging.
- The mappings must remain valid and allow writes to the buffer locations.

This applies for stage 1, stage 2, and GPT tables.

FEAT_TRBE and FEAT_SPE_nVM enable the operating system to use a *physically-addressed buffer* mode. For a guest virtual machine, this uses *intermediate* physical addresses. However, this is restricted to a single contiguous buffer in the physical or intermediate physical address space.

The use of an intermediate-physically addresssed buffer means the buffer must be pinned and allocated at stage 2.

Pinning and allocation can be particularly problematic when a hypervisor is used, because both stage 1 and stage 2 translations must be pinned and allocated. Hypervisors might otherwise implement lazy allocation of memory, to allow for oversubscription of virtual memory by guests.

To do this, the hypervisor must be able to determine which addresses in the guest physical address map are being used for the buffer. This is further complicated because:

- Software is not restricted in the order in which it writes to the SPE System registers, meaning that a guest operating system might configure the virtual address and limit of the Profiling Buffer before assigning and pinning the translation granules in the stage 1 translation.
- Other operating system threads could change a stage 1 mapping at any time.

Current solutions result in the hypervisor pinning and allocating *all* translation granules for a guest operating system. This is not an acceptable solution, particularly in cloud environments where techniques such as oversubscription by a hypervisor are used to ensure fair-share usage of resources by multiple tenants.

D_{VGCHG} This document uses the terms pinned and allocated as defined in Chapter 3.2 *Glossary*. In other documents, *pinned* is sometimes used to mean allocated.

G_{HRFPX} Define a programmers' model interface for SPE and TRBE that can used in production environments either on bare metal or with a hypervisor, without paravirtualization.

Chapter 2
Specification

D_{TGBCX} This document defines a programmers' model interface for SPE and TRBE called the *Virtual machine interface for SPE and TRBE*.

The *Virtual machine interface for SPE and TRBE* is defined to be used by an operating system that might be run under a hypervisor, but retains compatability with the bare metal interface. That is, additional programming contraints are defined that might be relied upon by a hypervisor, allowing the hypervisor to avoid certain use cases that would be difficult to virtualize.

The *Virtual machine interface for SPE and TRBE* has the following components:

- 2.1 *Accessing System registers*.
- 2.2 *Limiting the size of the buffer from a hypervisor*.
- 2.3 *Addressing modes*.

I_{ZKNJW} The *Virtual machine interface for SPE and TRBE* does not require new architecture features, and can be used on any implementation that includes FEAT_FGT and FEAT_SPE or FEAT_TRBE, as applicable.

However, it is expected that software will need to be written to follow the *Virtual machine interface for SPE and TRBE*, irrespective of whether it is running directly on hardware, or under a hypervisor.

S_{DMWTS} When using the *Virtual machine interface for SPE and TRBE* under virtualization:

- EL2 software uses traps to catch writes to BUFLIMITR_EL1.
- On a trapped EL1 write to BUFLIMITR_EL1 that sets BUFLIMITR_EL1.E to `0b1`, enabling the telemetry buffer, EL2 software allocates and pins the translation granules in the telemetry buffer at stage 2.

  For a guest managing stage 1 of the address translation, if the Effective value of BUFLIMITR_EL1.nVM is `0b0`, then EL2 software walks the stage 1 translation tables to discover the VA to IPA mapping.

For a guest hypervisor managing stage 2 of the address translation, EL2 software walks the guest stage 2 translation tables to discover the IPA to PA mapping.

- On a trapped EL1 write to BUFLIMITR_EL1 that clears BUFLIMITR_EL1.E to `0b0`, disabling the telemetry buffer, EL2 software can unpin those translation granules if needed, or allow this to happen later.

These rules apply equally to a guest hypervisor executing at EL1 under nested virtualization. If a guest hypervisor sets MDCR_EL2.E2xB to select EL2 as the owning Exception level, then it is acting as a guest operating system and is managing stage 1 of the address translation for the telemetry buffer. Otherwise, it is managing stage 2 of the address translation.

For this approach to work:

1. Before writing to the BUFLIMITR_EL1 register to enable the telemetry buffer, the guest must:
   - Have configured the telemetry buffer controls. For a guest hypervisor, this includes the MDCR_EL2.E2xB control.
   - Not allocate a buffer larger than that supported by EL2 software. See 2.2 *Limiting the size of the buffer from a hypervisor*.
   - For a guest managing stage 1 of the address translation, ensure one of the following applies:
     - The VA to IPA mapping is defined.
     - The Effective value of BUFLIMITR_EL1.nVM is `0b1`.

     For a guest hypervisor managing stage 2 of the address translation, ensure the IPA to PA mapping is defined.
   - Ensure all translation granules in the telemetry buffer are writable.
2. After writing to the BUFLIMITR_EL1 register to enable the telemetry buffer, and before disabling, the guest must:
   - Not change the telemetry buffer controls. See 2.1 *Accessing System registers*.
   - Not change the VA to IPA mapping or IPA to PA mapping, as applicable.
   - Execute a `TSB CSYNC` or `PSB CSYNC` instruction before disabling.

Because a guest operating system or hypervisor might not be aware that it is executing under virtualization, all operating systems or hypervisors must implement these rules regardless of which Exception level the operating system or hypervisor is executing at.

Note: If FEAT_SPE_nVM is not implemented, then the Effective value of PMBLIMITR_EL1.nVM is `0b0`.

S_{DJJTD} Because S_{DMWTS} requires that the hypervisor walk the stage 1 translation tables of the guest, the hypervisor must have read access to the stage 1 translation tables.

If FEAT_RME is implemented and Realm state is being debugged, then the RMM is responsible for walking the stage 1 translation tables. This is because the owning Security state is Realm state and the translation tables for the telemetry buffer are in Realm memory, even if the telemetry buffer itself is in Non-secure memory. For some systems, this might mean that FEAT_SPE or FEAT_TRBE cannot be used in Realm state.

See also:

- *Arm® Architecture Reference Manual, for A-profile architecture* [1].

2.1 Accessing System registers

S_{FDXJJ} When PMBLIMITR_EL1.E is `0b1`, meaning the Profiling Buffer is enabled, software must behave as if the PE can do all of the following:

- Ignore writes to the Profiling Buffer controls, other than a write to PMBLIMITR_EL1.E that disables the Profiling Buffer. The Statistical Profiling Unit registers affected are:
  - PMBPTR_EL1.
  - PMBLIMITR_EL1.
  - PMBSR_EL1.
  - If FEAT_SPE_nVM is implemented, PMBMAR_EL1.
- Prefetch and cache address translations for addresses in the Profiling Buffer.

I_{RNKGP} S_{FDXJJ} permits a hypervisor to emulate a virtual machine that ignores writes to Profiling Buffer controls and prefetches address translations.

I_{NQPRT} A future version of the architecture might permit the PE to behave as described in S_{FDXJJ}, including when the owning translation regime is out of context.

I_{XDXTN} When TRBLIMITR_EL1.E is `0b1`, meaning the trace buffer is enabled, FEAT_TRBE permits the PE to do all of the following:

- Ignore writes to the trace buffer controls, other than a write to TRBLIMITR_EL1.E that disables the trace buffer. For Self-hosted mode, the Trace Buffer Unit registers affected are:
  - TRBPTR_EL1.
  - TRBBASER_EL1.
  - TRBLIMITR_EL1.
  - TRBTRG_EL1.
  - TRBSR_EL1.
  - TRBMAR_EL1.
- Prefetch and cache address translations for addresses in the trace buffer.

S_{QVZNN} If the owning Exception level is EL1 and EL2 is implemented and enabled in the owning Security state, then hypervisor software executing at EL2 can use S_{FDXJJ} and I_{XDXTN} to also cache the translations. For instance, to allocate the stage 2 translations for the telemetry buffer on a trap of BUFLIMITR_EL1 that sets BUFLIMITR_EL1.E to `0b1`.

S_{KFKZQ} To avoid loss of telemetry, software must not change the translation table entries, stage 1, or stage 2 mappings for the telemetry buffer while the buffer is enabled.

This includes issuing TLB maintenance operations that might invalidate prefetched translations, meaning any benefit of prefetching is lost.

S_{LRLKW} Where FEAT_SPE or FEAT_TRBE permits CONSTRAINED UNPREDICTABLE behavior by hardware due to misprogramming of the telemetry unit by software, a hypervisor can emulate any of the permitted behaviors if a guest misprograms the telemetry unit in the same way.

FEAT_SPE or FEAT_TRBE only permit the CONSTRAINED UNPREDICTABLE behavior when the telemetry unit attempts to write telemetry to the telemetry buffer, or when debugging is enabled and allowed. However, the *Virtual machine interface for SPE and TRBE* allows the hypervisor to emulate any of the permitted behaviors immediately on the guest writing `0b1` to BUFLIMITR_EL1.E providing debugging is not stopped. That is, when BUFSR_EL1.S is `0b0`.

The hypervisor must not corrupt guest memory.

Example

Restrictions on programming the Trace Buffer Unit [1, D6.2.2, L.a] permits the telemetry unit to do any of the following, if the current write pointer is not valid when the telemetry unit attempts to write to the trace buffer, along with other CONSTRAINED UNPREDICTABLE options.

Restrictions on the current write pointer [1, D17.7.1, L.a] permits the telemetry unit to do any of the following, if the current write pointer is not valid when profiling is enabled, that is, when the telemetry buffer is enabled and all the other conditions in *Enabling profiling* [1, D17.4, L.a] are met.

- Write to any Location that is writable by a privileged access in the owning translation regime.
- Generate a telemetry buffer management event.

Under the *Virtual machine interface for SPE and TRBE*, the hypervisor is permitted to emulate any of the same CONSTRAINED UNPREDICTABLE behaviors if the current write pointer is not valid and BUFSR_EL1.S is `0b0` when the guest writes `0b1` to BUFLIMITR_EL1.E to enable the telemetry buffer. However, the hypervisor must not corrupt guest memory, and generate the telemetry buffer management event in this case.

In this example, *not valid* means:

- For FEAT_SPE, software has not followed the rules on setting the current write pointer described in *Restrictions on the current write pointer*.
- For FEAT_TRBE, the current write pointer has an *out-of-range value*, or a *misaligned value* that is not a *restart value*. These are terms defined in *Restrictions on programming the Trace Buffer Unit*.

S_{JFLTP} The hypervisor needs to be robust against the guest using invalid stage 1 translations when the Effective value of BUFLIMITR_EL1.nVM is `0b0`.

For example, if not all translation granules for the telemetry buffer have been allocated by the guest when it sets BUFLIMITR_EL1.E to `0b1`. The hypervisor cannot create stage 2 translations for these translation granules.

The hypervisor needs to be robust against the guest changing or invalidating stage 1 translations when the Effective value of BUFLIMITR_EL1.nVM is `0b0`.

That is, virtualization of the telemetry unit is not permitted to use a translation that has been invalidated, providing the invalidation has been fully synchronized using the correct software sequences. For example, this prohibits the hypervisor from setting EL2 as the owning Exception level and configuring the telemetry buffer to virtual addresses in the EL2&0 translation regime that are mapped to the same physical addresses as the virtual addresses used by the guest for telemetry buffer when the guest set BUFLIMITR_EL1.E to `0b1`.

If the guest does use an invalid stage 1 translation, or change or invalidate a stage 1 translation after enabling the telemetry buffer, then a telemetry buffer management fault might be generated. For example, because there is no valid stage 2 translation for the new stage 1 mapping. For the invalidating cases, this is as if the new stage 1 translation is not observed.

This is acceptable under the *Virtual machine interface for SPE and TRBE* because the guest violated its constraints under S_{DMWTS}.

The hypervisor needs to be robust against the guest incorrectly programming the telemetry buffer pointers. For example, setting the current write pointer to an address above the limit pointer.

Note: If FEAT_SPE_nVM is not implemented, then the Effective value of PMBLIMITR_EL1.nVM is `0b0`.

2.2 Limiting the size of the buffer from a hypervisor

D_{FPSHT} BUFIDR_EL1.MaxBuffSize is reserved to identify the largest supported telemetry buffer size to software.

R_{JDXTL} BUFIDR_EL1.MaxBuffSize reads as `0x0000`.

I_{RSPMS} A value of zero for BUFIDR_EL1.MaxBuffSize means there is no limit.

S_{BFBSJ} R_{JDXTL} defines the hardware behavior.

When FEAT_FGT is implemented, EL2 software can configure HDFGRTR_EL2 to trap reads of BUFIDR_EL1 and return a value for BUFIDR_EL1.MaxBuffSize that encodes the largest telemetry buffer size it will permit. EL2 software must be aware that a guest might cache this value.

When FEAT_FGT is not implemented, EL2 and EL1 software must use a paravirtualization mechanism, such as a device tree entry, to report the largest supported telemetry buffer size.

I_{JLYNY} There is no hardware support to virtualize the value of BUFIDR_EL1.MaxBuffSize and avoid the trap.

S_{XGVLY} Software treats BUFIDR_EL1.MaxBuffSize as the largest telemetry buffer available on the current PE. This is expected to be the same for all PEs in a multiprocessor system, although this is only a software convention.

This means that in a multiprocessor system, the hypervisor is normally able to allocate as many buffers of MaxBuffSize as there are PEs available to the guest.

However, BUFIDR_EL1.MaxBuffSize is a hint, not a guarantee.

There might be reasons why a hypervisor fails to honor the BUFIDR_EL1.MaxBuffSize value after publishing it in a register, or can honor larger buffer sizes if requested. For example:

- If memory resources become over-allocated and every PE in the system requests a BUFIDR_EL1.MaxBuffSize buffer, then the hypervisor might not be able to honor all the requests.
- If only a single PE in the system is requesting a telemetry buffer, then the hypervisor might be able to honor a larger request.

These are expected to be rare conditions. However, both guest and hypervisor must be able to cope with this condition.

S_{LJDDM} The number of bits available for BUFIDR_EL1.MaxBuffSize means that the suggested encoding is as a 5-bit exponent `E` and a 9-bit mantissa `M`, encoding the buffer size as follows:

```
if IsZero(E) then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))
```

With this encoding:

- A value of `0x0001` encodes the smallest maximum buffer size, 4KB.
- A value of `0x3FFF` encodes the largest maximum buffer size, 4,092TB.
- Only multiples of 4KB and not all values can be encoded. For example, only multiples of 8KB can be encoded between 4MB (`0x0400`) and 8MB (`0x0600`), and so on. Every multiple of 4KB up to 4MB, and every whole number of megabytes, gigabytes, or terabytes, up to 1,023, can be encoded.

Very large trace buffers are not uncommon, or an off-chip consumer might be memory-mapped into a large physical address range.

S_{JHSSQ} If the guest attempts to allocate a telemetry buffer that is larger than that specified by BUFIDR_EL1.MaxBuffSize, then, on a trapped write of BUFLIMITR_EL1 that sets BUFLIMITR_EL1.E to `0b1`, EL2 software immediately injects a buffer management event, by setting BUFSR_EL1 appropriately before returning to the guest. To allow this, the *Buffer size error telemetry buffer management event* is defined.

If the guest attempts to allocate a telemetry buffer that is not larger than that specified by BUFIDR_EL1.MaxBuffSize, but the hypervisor is unable to allocate the buffer (see S_{XGVLY}), then, on a trapped write of BUFLIMITR_EL1 that sets BUFLIMITR_EL1.E to `0b1`, EL2 software injects a telemetry buffer management event as described above.

On a Buffer size error telemetry buffer management event, the guest either retries with a smaller buffer size, or fails the operation. However, the guest must limit retries of the failed operation, and eventually fail.

I_{PLLFF} S_{JHSSQ} describes a simple approach to handling these cases. Injecting a telemetry buffer management event is the preferred approach, because it helps the guest to debug and/or react to this issue.

It is expected that the case where the hypervisor is unable to allocate the buffer will be uncommon. However, as this might not be the case in practice, the *Virtual machine interface for SPE and TRBE* leaves open the possibility of alternative handling, by the hypervisor, the guest, or both.

I_{QZJSG} S_{BFBSJ} and S_{JHSSQ} describe a form of architectural paravirtualization. This is unusual in the Arm architecture, but not unique.

I_{RLRWN} EL2 software could also allocate as many translation granules as it previously advertised as supported in BUFIDR_EL1.MaxBuffSize at the start of the telemetry buffer, marking other translation granules as no access or Translation fault at stage 2.

FEAT_TRBE supports a circular buffer mode, and so has all of a *Base*, *Current*, and *Limit* pointer. This means the hypervisor treats the buffer as starting at TRBPTR_EL1, wrapping from the address specified by TRBLIMITR_EL1.LIMIT to TRBBASER_EL1, and ending at TRBPTR_EL1, rather than starting at TRBBASER_EL1 and ending at the address specified by TRBLIMITR_EL1.LIMIT.

FEAT_SPE does not support a circular buffer mode, and so has only a *Current* and *Limit* pointer. The telemetry buffer starts at PMBPTR_EL1 and extends to the address specified by PMBLIMITR_EL1.LIMIT.

This allows for a degraded experience, as one of two outcomes might occur, including a possible loss of telemetry:

1. The PE takes some other exception before the buffer is filled. For example, a context switch timer interrupt.

   In this case, the guest will flush any outstanding data to the buffer, disable the buffer, and swap in another process. When the process being debugged is swapped in again, the guest will reenable the buffer. However, when it does so BUFPTR_EL1 has moved on, meaning the new limit can also be moved on.

   If the buffer that is allocated is still reasonably large, this might continue indefinitely with no loss of telemetry.

2. A stage 2 fault is generated when the buffer is exceeded. Telemetry might be lost.

Because of the possibility of losing telemetry, and the complexities involved with implementing such a scheme, this approach is not recommended.

2.2.1 Buffer size error buffer management event

D_{NZJXZ} When BUFSR_EL1.EC is `0b000000`, *Other buffer management event*, the value `0b000100` in BUFSR_EL1.BSC is reserved to encode a *Buffer size error* telemetry buffer management event.

S_{XCVLS} D_{NZJXZ} allows a hypervisor to emulate the following behavior if a guest operating system attempts to write a value to BUFLIMITR_EL1, such that BUFLIMITR_EL1.E is `0b1` and the telemetry buffer size exceeds the value published in BUFIDR_EL1.MaxBuffSize:

If BUFSR_EL1.S is `0b0`, then a Buffer size error telemetry buffer management event is generated:

- BUFSR_EL1.S is set to `0b1`.
- For FEAT_TRBE only, TRBSR_EL1.IRQ is set to `0b1`.
- BUFSR_EL1.EC is set to `0b000000`.
- The BSC field of BUFSR_EL1.MSS is set as follows:
  - BUFSR_EL1.BSC is set to `0b000100`, Buffer size error.
- The other fields in BUFSR_EL1 are unchanged.

For FEAT_TRBE, the telemetry buffer size is the difference between the addresses specified by TRBBASER_EL1 and TRBLIMITR_EL1.LIMIT.

For FEAT_SPE, the telemetry buffer size is the difference between the addresses specified by PMBPTR_EL1 and PMBLIMITR_EL1.LIMIT.

Note: The telemetry unit never generates the Buffer size error telemetry buffer management event.

2.3 Addressing modes

I_{NPSDC} FEAT_SPE_nVM adds a register field, PMBIDR_EL1.AddrMode, that identifies which of the addressing modes are implemented. This includes a value reserved for *Physical address mode only*.

Without hardware support for FEAT_SPE_nVM, emulating *physical address mode* in a hypervisor requires the hypervisor to set the Owning Exception level to EL2 and map the buffer in the EL2 virtual address space as an alias for the EL1 intermediate physical address space buffer. In addition, when using *physical address mode*, the operating system must allocate a buffer that is contiguous in its output address space. For a guest operating system running under a hypervisor, this is the intermediate physical address space.

However, if FEAT_SPE_nVM is not implemented, then when the guest operating system attempts to write to PMBMAR_EL1, the access will be UNDEFINED rather than be trapped to EL2.

For this reason, the PMBIDR_EL1.AddrMode field is architecturally defined as valid only when FEAT_SPE_nVM is implemented. That is, software only uses *physical address mode* when both PMBIDR_EL1.AddrMode defines that it is available and ID_AA64DFR2_EL1.SPE_nVM defines that PMBIDR_EL1.AddrMode is valid. However, the field might be used under virtualization to describe a cut-down variant of *physical address mode* with no PMBMAR_EL1 register using fixed memory attributes.

When FEAT_SPE_nVM is supported, EL2 can support use of *physical address mode* in much the same way as it supports *virtual address mode*. See S_{DMWTS}.

An existing guest operating system that is not aware of this ID field will always try to use *virtual address mode* for FEAT_SPE. For this reason, supporting only *physical address mode* is provided as an option should future implementation of virtualization require it, but this is not recommended. For example, it might be a lower-cost alternative to virtualization of *virtual address mode*. Full details of such support are outside the scope of this document.

I_{YBKCC} FEAT_TRBEv1p1 adds a register field, TRBIDR_EL1.AddrMode, that identifies which of the addressing modes are implemented. This includes values reserved for *Virtual address mode only* and *Physical address mode only*.

A hypervisor might only provide support for *Virtual address mode only* or *Physical address mode only*. In particular, *Physical address mode only* could be emulated by EL2 being the Owning Exception level, and aliasing the buffer in the EL2 address space. Such an emulation removes the need for the hypervisor to walk to the stage 1 translation tables each time the buffer is enabled.

However, by using *physical address mode*, the operating system must allocate a buffer that is contiguous in its output address space. For a guest operating system running under a hypervisor, this is the Intermediate Physical Address (IPA) space.

An existing guest operating system that is not aware of this ID field might try to use either mode for FEAT_TRBE. For this reason, supporting only one of the *virtual* and *physical address modes* is provided as an option should future implementations of virtualization require it, but this is not recommended. For example, it might be a lower-cost alternative to full virtualization. Full details of such support are outside the scope of this document.

Chapter 3
Programmers' Model

Table 3.1 provides the disambiguation of the general System register, System register field, and instruction names used in this document.

Table 3.1: Disambiguation of general names

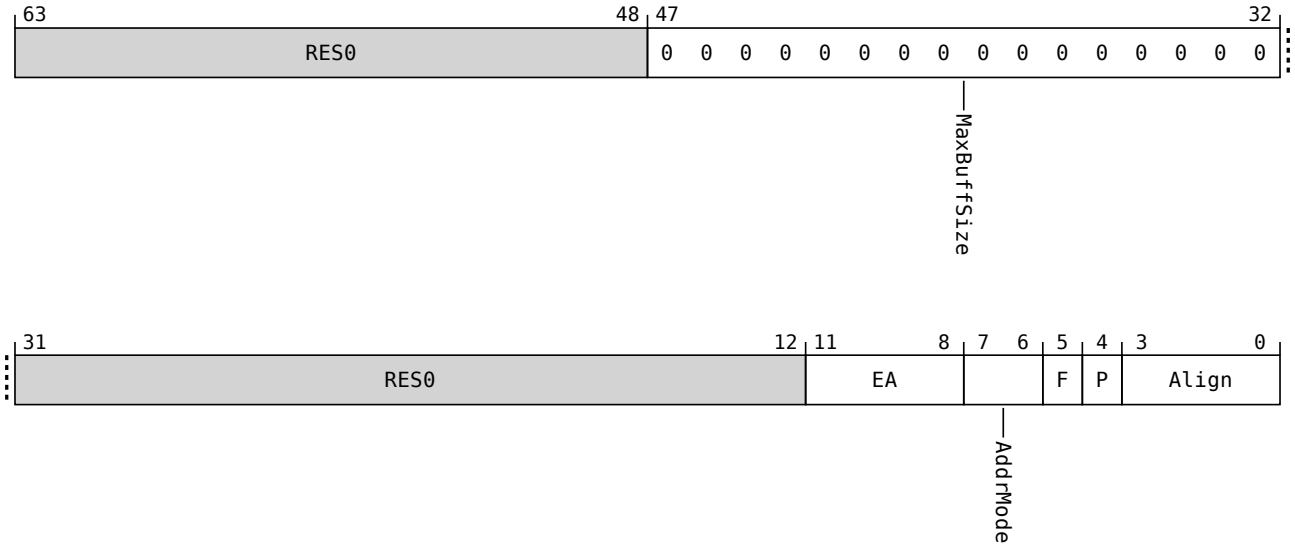
General form	SPE	TRBE
BUFSR_EL1	PMBSR_EL1	TRBSR_EL1
BUFPTR_EL1	PMBPTR_EL1	TRBPTR_EL1
BUFIDR_EL1	PMBIDR_EL1	TRBIDR_EL1
BUFLIMITR_EL1	PMBLIMITR_EL1	TRBLIMITR_EL1
MDCR_EL2.E2xB	MDCR_EL2.E2PB	MDCR_EL2.E2TB
MDCR_EL3.NSxB	MDCR_EL3.NSPB	MDCR_EL2.NSTB
MDCR_EL3.NSxBE	MDCR_EL3.NSPBE	MDCR_EL2.NSTBE

3.1 New and modified AArch64 System register fields

3.1.1 PMBIDR_EL1, Profiling Buffer ID Register

The PMBIDR_EL1 characteristics are:

Field descriptions



MaxBuffSize, bits [47:32]

Maximum supported Profiling Buffer size. Reserved for software use.

Reads as `0x0000`

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Note

Permitted values relate to the values an implementation is permitted to set this field to. A hypervisor might trap accesses to this register and use other values to describe limitations of its virtualization support to a guest operating system, as follows:

- MaxBuffSize bits[8:0] encodes a mantissa value, `M`.
- MaxBuffSize bits[13:9] encodes an exponent value, `E`.
- MaxBuffSize bits[15:14] are reserved.

The maximum buffer size, in bytes, is expressed using the following function:

```
if IsZero(E) then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))
```

For example:

- A value of `0x0001` means a maximum buffer size of 4KB.
- A value of `0x3FFF` means a maximum buffer size of 4092TB.

Access to this field is RO.

AddrMode, bits [7:6]

When FEAT_SPE_nVM is implemented:

Address Modes. Describes the addressing modes available for the Profiling Buffer.

AddrMode	Meaning
0b00	Only virtual address mode is supported.
0b01	Virtual and physical address modes are supported.
0b11	Reserved for software use under virtualization, to show that only physical address mode is supported.

Other values are reserved.

If the Effective value of PMSCR_EL2.EnVM is 1 and the value returned for PMBIDR_EL1.P is 0, then this field reads as 0b01. Otherwise, this field reads as 0b00.

Note

A hypervisor might trap accesses to this register to describe limitations of its virtualization support to a guest operating system.

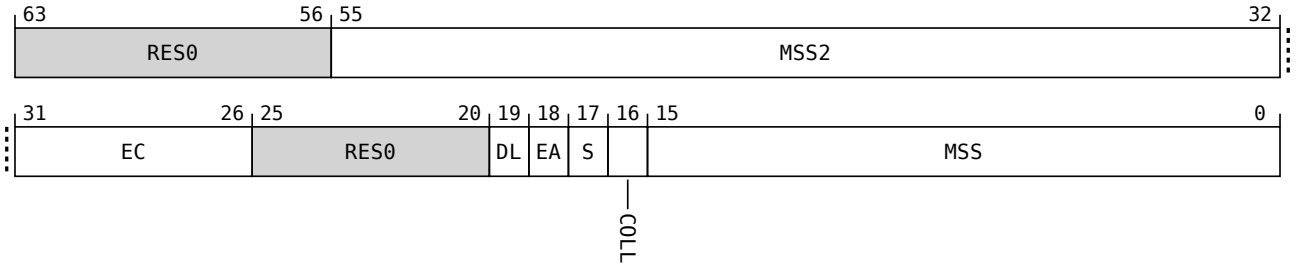
Otherwise:

Reserved, RES0.

3.1.2 PMBSR_EL1, Profiling Buffer Status/syndrome Register (EL1)

The PMBSR_EL1 characteristics are:

Field descriptions

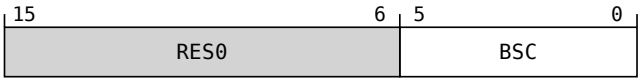


MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the Profiling Buffer management event.

The syndrome contents for each Profiling Buffer management event are described in the following sections.

MSS encoding for other Profiling Buffer management events



BSC, bits [5:0]

Profiling Buffer status code

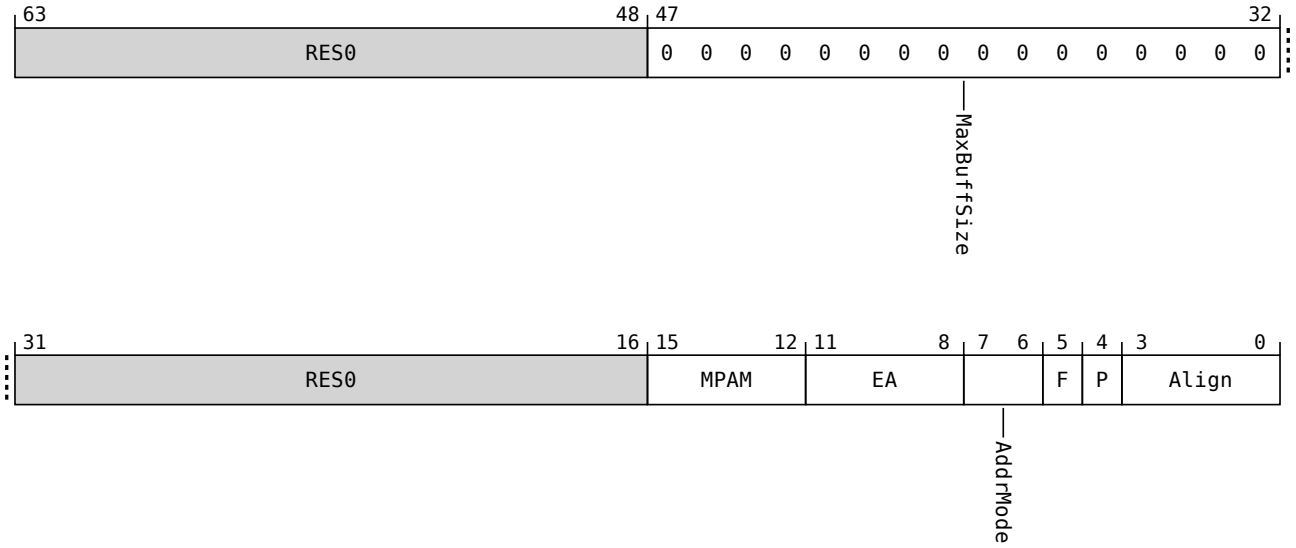
BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Profiling Buffer filled.
0b000100	Buffer size. The requested Profiling Buffer size was too large.

All other values are reserved.

3.1.3 TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

Field descriptions



MaxBuffSize, bits [47:32]

Maximum supported trace buffer size. Reserved for software use.

Reads as `0x0000`

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Note

Permitted values relate to the values an implementation is permitted to set this field to. A hypervisor might trap accesses to this register and use other values to describe limitations of its virtualization support to a guest operating system, as follows:

- MaxBuffSize bits[8:0] encodes a mantissa value, `M`.
- MaxBuffSize bits[13:9] encodes an exponent value, `E`.
- MaxBuffSize bits[15:14] are reserved.

The maximum buffer size, in bytes, is expressed using the following function:

```
if IsZero(E) then UInt(M:Zeros(12)) else UInt('1':M:Zeros(UInt(E)+11))
```

For example:

- A value of `0x0001` means a maximum buffer size of 4KB.
- A value of `0x3FFF` means a maximum buffer size of 4092TB.

Access to this field is RO.

AddrMode, bits [7:6]

Address Modes. Describes the addressing modes available for the trace buffer.

AddrMode	Meaning
0b00	Virtual and physical address modes are supported.
0b01	Only virtual address mode is supported.
0b10	Reserved for software use under virtualization, to show that only physical address mode is supported.

Other values are reserved.

If the Effective value of TRFCR_EL2.DnVM is 1 and the value returned for TRBIDR_EL1.P is 0, then this field reads as 0b01. Otherwise, this field reads as 0b00.

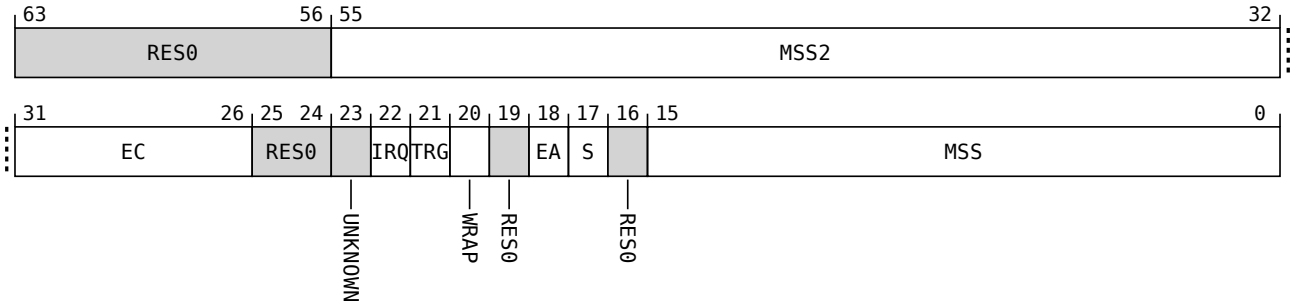
Note

A hypervisor might trap accesses to this register to describe limitations of its virtualization support to a guest operating system.

3.1.4 TRBSR_EL1, Trace Buffer Status/syndrome Register (EL1)

The TRBSR_EL1 characteristics are:

Field descriptions

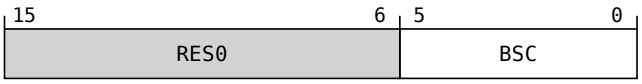


MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events



BSC, bits [5:0]

Trace buffer status code

BSC	Meaning	Applies when
0b000000	Collection not stopped, or access not allowed.	
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.	
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.	
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.	FEAT_TRBE_EXT is implemented
0b000100	Buffer size. The requested trace buffer size was too large.	

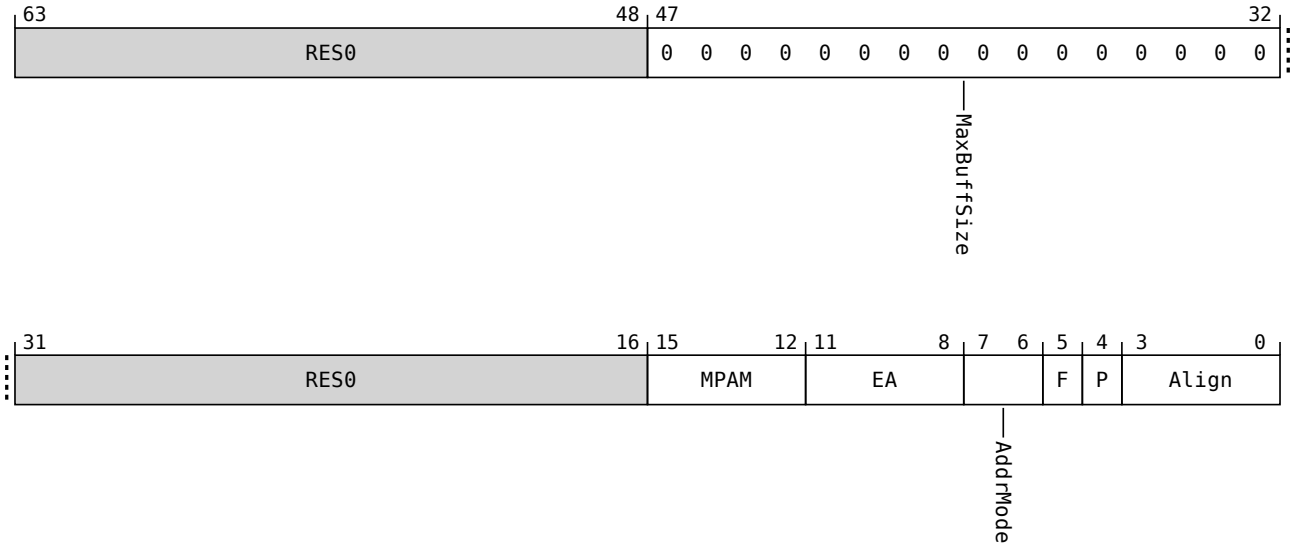
All other values are reserved.

3.2 New and modified external TRBE register fields

3.2.1 TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

Field descriptions



MaxBuffSize, bits [47:32]

Maximum supported trace buffer size. Reserved for software use.

Reads as `0x0000`

The only permitted value is `0x0000`, indicating there is no limit to the maximum buffer size.

Access to this field is RO.

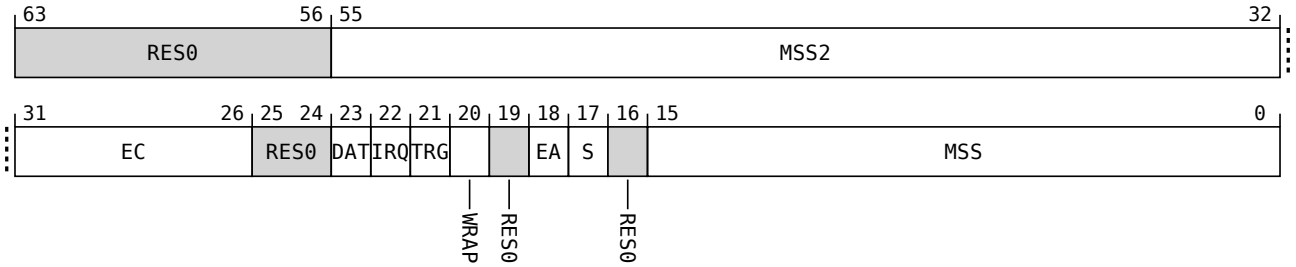
AddrMode, bits [7:6]

This field reads as an UNKNOWN value.

3.2.2 TRBSR_EL1, Trace Buffer Status/syndrome Register

The TRBSR_EL1 characteristics are:

Field descriptions

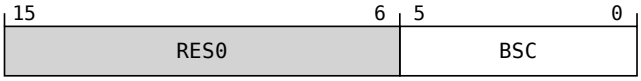


MSS, bits [15:0]

Management Event Specific Syndrome. Contains syndrome specific to the trace buffer management event.

The syndrome contents for each trace buffer management event are described in the following sections.

MSS encoding for other trace buffer management events



BSC, bits [5:0]

Trace buffer status code

BSC	Meaning
0b000000	Collection not stopped, or access not allowed.
0b000001	Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.
0b000010	Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.
0b000011	Manual Stop. Collection stopped because of a Manual Stop event. See TRBCR.ManStop for more information.
0b000100	Buffer size. The requested trace buffer size was too large.

All other values are reserved.

Glossary

Allocated

A translation granule is *allocated* by an operating system or hypervisor when the memory used to back the translation granule has been reserved by the operating system or hypervisor. However, the translation associated with the translation granule is not guaranteed to remain continuously valid and writable, unless the translation granule is also pinned.

Owning Exception level

The *owning Exception level* is defined by the following:

- For FEAT_SPE, the Profiling Buffer *owning Exception level* is defined by *The owning translation regime* [1, D17.7.2, L.a].
- For FEAT_TRBE, the trace buffer *owning Exception level* is defined by R_{SKVWG} in *The owning translation regime* [1, D6.3.5, L.a].

Owning Security state

The *owning Security state* is defined by the following:

- For FEAT_SPE, the Profiling Buffer *owning Security state* is defined by *The owning translation regime* [1, D17.7.2, L.a].
- For FEAT_TRBE, the trace buffer *owning Security state* is defined by R_{HBZNT} in *The owning translation regime* [1, D6.3.5, L.a].

Owning translation regime

The *owning translation regime* is defined by the following:

- For FEAT_SPE, the Profiling Buffer *owning translation regime* is defined by *The owning translation regime* [1, D17.7.2, L.a].
- For FEAT_TRBE, the trace buffer *owning translation regime* is defined by R_{DPGJG} in *The owning translation regime* [1, D6.3.5, L.a].

Pinned

A translation granule is *pinned* by an operating system or hypervisor when the translation associated with the translation granule is guaranteed to remain continuously valid and writable.

Statistical Profiling Extension

The Statistical Profiling Extension, *FEAT_SPE*, provides a non-invasive method of sampling software and hardware using randomized sampling of either architectural instructions, as defined by the instruction set architecture, or by microarchitectural operations.

Statistical Profiling physical addressing mode extension

The Statistical Profiling physical addressing mode extension, *FEAT_SPE_nVM*, enables use of a *physically-addressed buffer* mode for the Statistical Profiling buffer. FEAT_SPE_nVM is part of the Armv9.6 extensions.

Statistical Profiling Unit

The part of a PE that implements the Statistical Profiling Extension, FEAT_SPE.

Telemetry

Data generated by a telemetry unit in the PE and written to a telemetry buffer. That is, one of *profiling data* or *trace data*. For more information, see *telemetry buffer*.

Telemetry buffer

An in-memory buffer, defined by software, used to store telemetry.

- For FEAT_SPE, the Statistical Profiling Unit writes profiling data to an in-memory *Profiling Buffer*.
- For FEAT_TRBE, the Trace Buffer Unit writes trace data to an in-memory *trace buffer*.

Trace Buffer Extension

The Trace Buffer Extension, *FEAT_TRBE*, enables support for a Trace Buffer Unit within a PE. When the Trace Buffer Unit is enabled, program-flow trace generated by a trace unit is written directly to memory by the Trace Buffer Unit, rather than routing trace data to a trace sink.

Trace Buffer Extension version 1.1

The Trace Buffer Extension version 1.1, *FEAT_TRBEv1p1*, is part of the Armv9.6 extensions.

Trace Buffer Unit

The part of a PE that implements the Trace Buffer Extension, FEAT_TRBE.