Arm® Base System Architecture 1.2

Platform Design Document

Non-confidential

Document number: DEN0094E

Contents

Release information

Version 1.2 (01 Jul 2025)

- Deprecate the concept of BSA Levels.
- Assign IDs to information statements.
- Future requirements - Add RCiEP and iEP rules B_REP_1 and B_IEP_1 to future requirements checklist.
- Future requirements - Add PE security rules B_SEC_01 - B_SEC_05 to future requirements checklist.
- Future Requirements - Move SBSA rules S_PCIe_10 and S_PCIe_11 to BSA B_PCIe_10 and B_PCIe_11. (822).
- Future Requirements - FRS/DRS MSI/MSI-X interrupt requirement (750).
- Future requirements - Add PE context-aware breakpoint requirements (852).
- Move "Self-hosted debug for Armv9-A" from SBSA to BSA (928).
- Errata for BSA 1.1 - Update to SMMU_02 to clarify the constraints on stalling devices (817).
- Errata for BSA 1.1 - Relax B_MEM_04 to be a recommendation (810).
- Errata for BSA 1.1 - Clarify PCIe initial memory type and attributes (822).
- Errata for BSA 1.1 - Mark the PCIe integration rules that are not applicable to RCiEP or RCEC (846).
- Errata for BSA 1.1 - RCiEP and i-EP scope clarification (835).
- Errata for BSA 1.1 - Rework i-EP and RCiEP sections for commonality (836).
- Errata for BSA 1.1 - B_SEC_05 (FEAT_SPECRES) missing support for FEAT_SPECRES2 (918).
- Errata for BSA 1.1 - Replace Prefetchable and Non-Prefetchable PCIe memory with 64-bit and 32-bit (630).

Version 1.1 (20 Oct 2024)

- Main changes are adding BSA level 1 (existing rules) and future level 2, and Errata for BSA 1.0.
- Future Requirements - Require FEAT_LSE (Atomics) from Armv8.1 (598).
- Future Requirements - Recommend FEAT_LRCPC from Armv8.3+ (599).
- Move SBSA Appendix A (Support for Secure Firmware) to BSA (818).
- Errata for BSA 1.0 - Clean separation between BSA and SBSA (571).
- Errata for BSA 1.0 - Relax UART to be conditional if in use by OS or debugging or end user (756).
- Errata for BSA 1.0 - Clarify that systems with GICv4 are compliant with the GICv3 rules (576).
- Errata for BSA 1.0 - Do not recommend GICv3 + v2M for MSI (553).
- Errata for BSA 1.0 - Clarify "Legacy I/O" terminology (521).
- Errata for BSA 1.0 - Relax RB_PE_15 "standard algorithm" requirement to a recommendation (590).
- Errata for BSA 1.0 - Update spec references to PCIe 6.0 (588).
- Errata for BSA 1.0 - Removing "Arm Strongly Recommends" instances (611).
- Errata for BSA 1.0 - Recommendation on usage of 32bit BARs (620).
- Errata for BSA 1.0 - Clarification on the recommended algorithms for FEAT_PAuth (596).
- Errata for BSA 1.0 - Clarification that SVE2 is required for Armv9 standard systems (574).
- Errata for BSA 1.0 - Clarification on system memory view for PCIe DMA transactions and permitted address modifications (518).
- Errata for BSA 1.0 - 32-bit NP BAR support clarification (643).
- Errata for BSA 1.0 - Add SMMU DVM implementation note (631).
- Errata for BSA 1.0 - Remove LPIs as a permitted interrupt type for certain on-chip peripherals (Timer/WDOG/UART) (656).
- Errata for BSA 1.0 - Recommend Precision Time Measurement (PTM) for some environments (595).
- Errata for BSA 1.0 - Clarify RB_SMMU_06 and introduce System Physical Address Space (SPAS) (624).
- Errata for BSA 1.0 - Clarify RBB_SMMU_12 (811).
- Errata for BSA 1.0 - Remove CBSA reference (650).
- Errata for BSA 1.0 - MSI Interrupts only required if interrupts supported (697).
- Errata for BSA 1.0 - Remove PPI assignment rule from BSA checklist (694).
- Errata for BSA 1.0 - Remove the requirement for i-EP to not have wire-based interrupts (717).
- Errata for BSA 1.0 - Update B_SEC_01 for FEAT_CSV2_3 (710).

- Errata for BSA 1.0 - Update B_PE_16 to reference FEAT_MTE2 (708).
- Errata for BSA 1.0 - Recommend SM3 and SM4 crypto extensions in some markets (738).
- Errata for BSA 1.0 - RCiEP OS support update (691).
- Errata for BSA 1.0 - Add rationale and errata to RB_GIC_04 (741).
- Errata for BSA 1.0 - EP, RCiEP, and i-EP acceptable latency requirement relaxation (747).
- Errata for BSA 1.0 - FEAT_LSE and FEAT_LSE2 recommendations and implementation note (748).
- Errata for BSA 1.0 - Add note on OS behavior with non-permitted heterogenous PE differences (718).
- Errata for BSA 1.0 - Clarify that RCEC can be implemented as part of the RCiEP (760).
- Errata for BSA 1.0 - Clarify RB_PE_9 PMUv3 requirement (807).
- Errata for BSA 1.0 - P2P optionality prevents devices and PEs from having the same memory view of the system (676).
- Errata for BSA 1.0 - BSA timer rules clarifications (774).

Version 1.0C (06 Oct 2022)

- Errata for BSA 1.0 - Change "Arm Recommends" and "Arm Strongly Recommends" terminology (532).
- Errata for BSA 1.0 - BSA checklist update - remove SBSA specific rules (540).
- Errata for BSA 1.0 - Minor typos and formatting issues in Appendix E, F, H (549).
- Errata for BSA 1.0 - Clarification of always-on PE timers (543).

Version 1.0B (30 Mar 2022)

- Main changes are clarifying and re-writing Section E.8, PCIe I/O Coherency rules.
- Errata for BSA 1.0 - Replace RPCI_IC_01 - RPCI_IC_09 with RPCI_IC_11 - RPCI_IC_18 (446).
- Errata for BSA 1.0 - RB_PE_01 to allow Armv9-A (466).
- Errata for BSA 1.0 - R8_PE_03 (323).
- Errata for BSA 1.0 - RB_PE_09, RB_PE_10, C.2, RRE_INT_1, RRE_ORD_4 (339).
- Errata for BSA 1.0 - RB_PE_14 (463).
- Errata for BSA 1.0 - RPCI_IN_09 (441).
- Errata for BSA 1.0 - "PI011" typo (450).
- Errata for BSA 1.0 - Update PCIe related definitions (464).
- Use the term requester to represent devices capable of initiating memory transactions (339).

Version 1 (21 Sep 2020)

- Initial public 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 owner 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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

About this document

Terms and abbreviations

Term	Meaning
ABI	Application Binary Interface.
ACS	Access Control Services. A set of features intended to ensure that uncontrolled peer-to-peer transaction cannot occur. See PCIe specification [1] for more details.
AER	Advanced Error Reporting. A PCIe feature that enables software to isolate and analyze errors with fine granularity. See PCIe specification [1] for more details.
ARE	Affinity Routing Enable (GICv3 [2]).
Arm ARM	Arm Architecture Reference Manual. See [3].
ATS	Address Translation Services.
BBR	Base Boot Requirements. See [4].
Completer	An agent in a computing system that responds to and completes a memory transaction that was initiated by a Requester.
CSP	Cloud Service Provider.
CTI	Cross Trigger Interface, see [3].
DMA	Direct Memory Access.
DVM	Distributed Virtual Memory.
ECAM	Enhanced Configuration Access Mechanism.
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 cache.
LPI	Locality-specific Peripheral Interrupt (GICv3 [2]).
MMIO	Memory Mapped Input Output.
P2P or Peer-to-peer	See PCIe specification [1] for more details.
PCIe Host Bridge (PHB)	See PCIe specification [1] for more details.
PE	Processing Element, as defined in the Arm ARM.
PMU	Performance Monitor Unit.
PPI	Private Peripheral Interrupt.
PRI	Page Request Interface.
PTM	Precision Time Measurement. PCIe standard specified method for finding the relationship between a PTM primary clock and another clock in a device in the same hierarchy.
RCEC	Root Complex Event Collector. See PCIe specification [1] for more details.
RCiEP	Root Complex integrated End Point. See PCIe specification [1] for more details.

Term	Meaning
Requester	An agent in a computing system that is capable of initiating memory transactions.
Root Complex (RC)	See PCIe specification [1] for more details.
Root Port (RP)	See PCIe specification [1] for more details.
SBSA	Server Base System Architecture. See [5].
SGI	Software-Generated Interrupt.
SPI	Shared Peripheral Interrupt.
SR-IOV	Single Root I/O virtualization. This is a method for a PCIe device to be virtualized. See PCIe specification [1] for more details.
SRE	System Register interface Enable (GICv3 [2]).
SVM	Shared Virtual Memory.
System firmware data	System description data structures, for example ACPI or Flattened Device Tree.
TCG	Trusted Computing Group.
TPM	Trusted Platform Module is a technology, typically implemented through a Secure microcontroller, that can securely store artifacts used to authenticate a platform, or platform components. The TPM technical specification is maintained by the TCG consortium.
Trace unit	A logical component which generates a trace stream from a PE.
VM	Virtual Machine.

References

This section lists publications by Arm and by third parties.

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

[1] *PCI Express Base Specification Revision 6.3, version 1.0*. PCI-SIG.

[2] *IHI 0069 Arm® Architecture Specification, GIC architecture version 3.0 and version 4.0*. Arm Ltd.

[3] *DDI 0487 Arm® Architecture Reference Manual for A-profile architecture*. Arm Ltd.

[4] *DEN 0044 Arm® Base Boot Requirements*. Arm Ltd.

[5] *DEN 0029 Server Base System Architecture*. Arm Ltd.

[6] *Advanced Configuration and Power Interface Specification.* Industry.

[7] *DEN 0151 PC Base System Architecture*. Arm Ltd.

[8] *DEN 0022 Arm Power State Coordination Interface, System Software on ARM specification*. Arm Ltd.

[9] *DEN 0077 Arm Firmware Framework for Arm v8-A*. Arm Ltd.

[10] *PC16550D UART Datasheet*. National Semiconductor.

[11] *DDI 0183 Arm® PrimeCell® UART (PL011) Technical Reference Manual*. Arm Ltd.

[12] *PCI Local Bus Specification Revision 3.0*. PCI-SIG.

[13] *PCI-To-PCI Bridge Architecture Specification 1.2*. PCI-SIG.

[14] *Arm IHI 0062 Arm® System Memory Management Unit Architecture Specification*. Arm Ltd.

[15] *IHI 0070 Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.3*. Arm Ltd.

[16] *IHI 0048 Arm® Architecture Specification, GIC architecture version 2.0*. Arm Ltd.

[17] *IHI 0054 ARM® System Trace Macrocell Programmers Model Architecture*. Arm Limited.

[18] *DEN 0068 CoreSight Base System Architecture*. Arm Ltd.

[19] *DDI 0416 Arm® CoreSight Trace Memory Controller Technical Reference Manual*. Arm Limited.

[20] *ARM® Embedded Trace Router Architecture Specification*. Arm Limited.

[21] *ARM® CoreSight System-on-Chip SoC-600 Technical Reference Manual*. Arm Limited.

[22] *IHI 0029 ARM® CoreSight Architecture Specification*. Arm Limited.

[23] *DEN 0034 ARM® Debug and Trace Configuration and Usage Models*. Arm Limited.

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
- Rationale
- Implementation note
- 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 generally 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*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

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*.

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find offensive terms in this document, please contact terms@arm.com.

Feedback

Arm welcomes feedback on its documentation.

If you have any comments or suggestions for additions and improvements create a ticket at

https://support.developer.arm.com.

As part of the ticket include:

- The title (Arm Base System Architecture).
- The document ID and version (DEN0094E 1.2).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Background

Arm processors are used in a wide variety of system-on-chip products in many diverse markets. The constraints on products in these markets are inevitably very different, and it is impossible to produce a single product that meets all of the needs of the various markets.

The Arm architecture defines different architectural profiles: Application profile, Real-time profile, and Microcontroller profile. These profiles segment the solutions that are produced by Arm, and align with the varying functional requirements of target markets. The differences between products that are targeted at different profiles are substantial. This is because of the diverse functional requirements of the market segments.

However, even within an architectural profile, the wide-ranging use of a product means that there are frequent requests for features to be removed to save silicon area. This is relevant for products targeted at cost-sensitive markets. In these markets, the cost of customizing software to accommodate the loss of a feature is small compared to the overall cost saving of removing the feature itself.

In other markets, for example those which require an open platform with complex software, the savings that are gained from removing a hardware feature are outweighed by the cost of software development to support the different variants. Also, software development is often performed by third parties. The uncertainty about whether new features are widely deployed can be a substantial barrier to the adoption of those features.

The Arm Application profile must balance these two competing business pressures. It offers a wide range of features, for example Advanced SIMD, SVE, SME, virtualization support, and TrustZone system security technology, to tackle an increasing range of problems. It also provides the flexibility to reduce silicon space by removing hardware features in cost-sensitive implementations.

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 (BSA) specifications are part of Arm's strategy of addressing this variability.

2 Introduction

This document specifies a hardware system architecture, based on Arm 64-bit architecture, that system software, for example operating systems, hypervisors, and firmware can rely on. This document addresses PE features and key aspects of system architecture.

The primary goal of this document is to ensure standard system architecture to enable generic OS images to install, boot, and run on compliant hardware. A driver-based model for advanced platform capabilities beyond basic system configuration and boot is required. However, such a driver model is outside the scope of this document. Fully discoverable and describable peripherals aid the implementation of this type of driver model.

Arm does not mandate compliance with this specification. However, Arm anticipates that OEMs, ODMs, cloud service providers and software providers will require compliance to maximize Out of Box software compatibility and reliability.

Implementations that are consistent with Base System Architecture can include additional features that are not included in the definition of BSA. However, software that is written for a specific version of BSA, must run, unaltered, on implementations that include this type of additional functionality.

Note

This is intended to avoid approaches, like software emulation of functionality that is critical to the performance of software using the BSA. It is not intended to act as a restriction of legitimate exploration of the power, performance, or area tradeoffs that characterize different products, nor to restrict the use of trapping within a virtualization system.

Software that runs on a system including an Arm core inevitably includes code that is system specific. This type of code is typically partitioned from the rest of the system software in the form of Firmware, Hardware Abstraction Layers, Board Support Packages, drivers and similar constructs. This document refers to these types of constructs as *Hardware Specific Software*. The Arm Base Boot Requirements (Arm BBR) specification [4] describes boot requirements for operating systems that require the use of UEFI, ACPI[6], Device tree and SMBIOS.

This specification uses the phrase software that is consistent with the Base System Architecture to indicate software that is designed to be portable between different implementations that are consistent with the Base System Architecture. Boot Software that is consistent with the Base System Architecture might use features that are not included in this specification. Such usage, however, is conditional upon checking that the platform supports the features, for example by using hardware ID registers or system firmware data.

The list of rules to be implemented for each BSA version is available in Section 3.14. All rules and views of a version, that are described in that version's checklist, are required to be implemented to be compliant with a version.

An implementation is consistent with a version of the Base System Architecture if it implements all of the functionality required for that version at a performance that is appropriate for the target uses of that version. This means that all functionality of BSA is expected to perform well when exploited by software.

2.1 Approach and scope

This document is structured as BSA and supplements:

Section 3 Base System Architecture

The Base System Architecture (BSA), Section 3, describes the requirements and run-time features of a base system required to install, boot, and run an operating system.

Server BSA [5]

The Server Base System Architecture (SBSA) supplement [5] describes the requirements and features required for Server OS install, boot and functionality. SBSA provides a consistent target for software developers, driving the alignment of server market capabilities forward over time.

PC-BSA [7]

The PC Base System Architecture (PC-BSA) supplement [7] describes the requirements and features required for PC OS boot and functionality. PC-BSA provides a consistent target for software developers, driving the alignment of PC market capabilities forward over time.

Appendixes

The Appendixes provide additional implementation requirements and guidance.

2.2 How to interpret this document

The sections and rules in this document can be visualized this way.

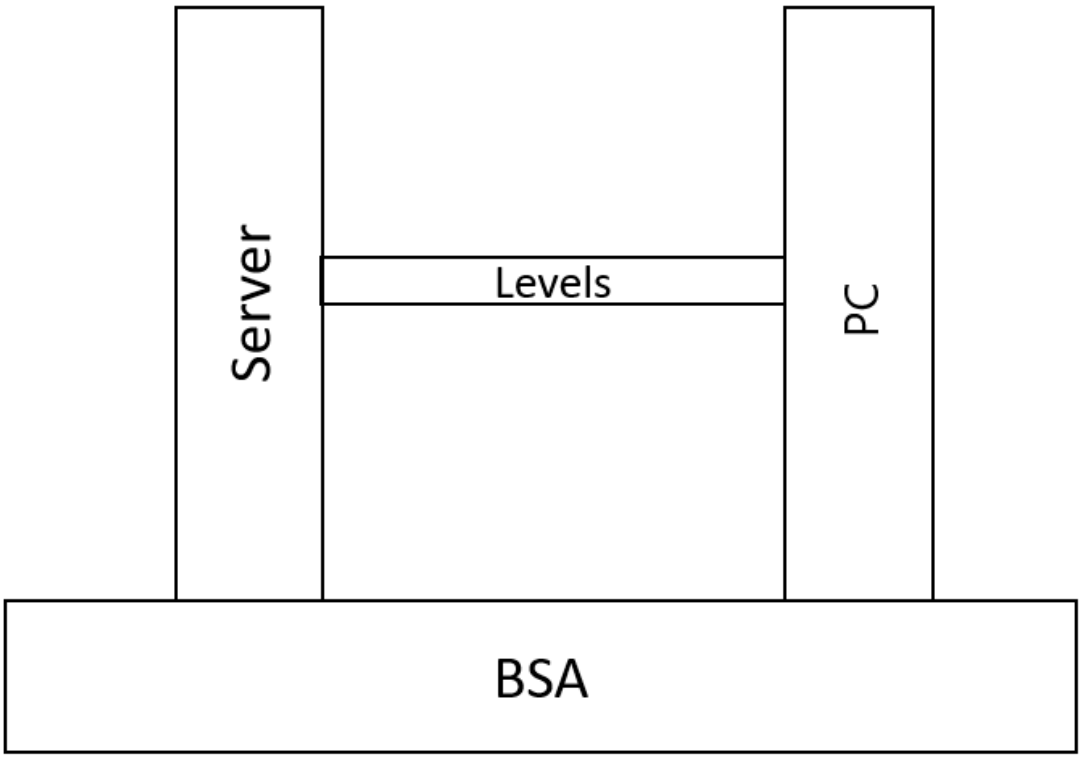

Figure 1: Target markets and Arm BSA

Standard OS boot: The rules in BSA, Section 3, must be implemented to support a Standard OS boot.

Server SoC designers: The rules in SBSA, [5] for the target level of compliance must be implemented.

PC SoC designers: The rules in PC-BSA, [7] for the target level of compliance must be implemented.

PCIe: If devices are presented to software as PCIe devices, they must be compliant to rules mentioned in Section E and Section 3.13.

2.3 Design patterns

This section outlines design patterns that provide the rationale behind key requirements expressed within this specification.

2.3.1 Non-secure operating systems

Operating systems can run on bare-metal or under virtualization. Fundamental to this is that the software view, the operating system has of the system, is invariant.

This specification aims to ensure that operating systems have access to a known minimum amount of functionality that they can rely on. This specification describes requirements on hardware and obligations on hypervisors and operating systems to achieve this minimum amount of functionality.

Non-secure devices offer much of the key functionality of a base system and are expected to be under the control of the operating system. It is expected that devices will have device specific drivers in the operating system to manage and control the device, however there are certain design patterns that are anticipated.

Interrupts from devices will use an interrupt controller that is compliant with Arm GIC architecture, with no intervening logic. The interrupt controller is critical to the operating system, so it must be a standard architecture with no device specific code to handle the acknowledgment and/or routing of interrupts. This means that systems must not have non-standard components for combining interrupts or converting between MSI and wired interrupts.

2.3.1.1 Memory allocation and DMA

Allocating memory for a device is carried out by the device driver interfacing with the operating system. Most operating systems support allocating memory that is contiguous in (intermediate) physical address space. However, it is impossible for operating systems to always guarantee this type of allocation. This is because the memory space becomes fragmented over time. Although certain devices will always prefer memory to be allocated either contiguously or in large chunks, it is expected that devices should be able to handle memory that is non-contiguous in (I)PA space in 4kB chunks.

It is typical for an OS-level device driver to pin pages that are targeted by DMA, to avoid DMA page faults that are in general fatal to the DMA transfer. Pinning ensures pages are both present and immune to page-out or removal by other threads. Depending on the driver and OS, DMA memory might be allocated or managed by the driver and pinned for the lifetime of the driver, or might be user space pages that are pinned before the DMA transfer begins and unpinned upon its completion.

Some devices can withstand DMA page faults, for example using the PCIe ATS and PRI protocols. A typical usage is for this type of a device to be mapped directly into the address space of an application. The application programs the DMA directly, using virtual addresses and avoids the cost of passing requests through a kernel driver. The resulting DMA might lead to a page fault, which is dealt with in a common OS service in a similar way to a CPU page fault. After resolving the fault, the service instructs the device to retry the transfer. Just as for a CPU page fault, the procedure is transparent to the application software aside from a small delay.

There are two design patterns that are anticipated for devices to work with non-contiguous memory:

- The device has a device specific structure, for example an MMU or scatter-gather tables, under the control of the device's device driver that handles the mapping. In this design pattern the MMU or other structure is not required to be standard. The MMU and device specific structure is under the control of device specific software. Whether a device can tolerate page faults on memory that it has been assigned is device specific, and the responsibility of the driver to allocate memory accordingly. If the device requires stage 2 SMMU functionality, as described in Non-secure hypervisors section, then this must be an Arm SMMU-compliant stage 2 implementation.
- The device is behind a stage 1 Arm SMMU that the operating system configures to map contiguous virtual memory addresses to discontiguous physical allocations. The device driver then uses virtual

addresses for device DMA. In this design pattern, the stage 1 SMMU functionality must be an Arm SMMU compliant implementation so that it can be managed by standard operating system code.

2.3.2 Hypervisors

Arm expects that the use of hypervisors in the non-secure space will become common place in client, edge, and server devices. Figure 2 shows the various means to host an OS through virtualization as compared to a baseline of bare-metal hosting.

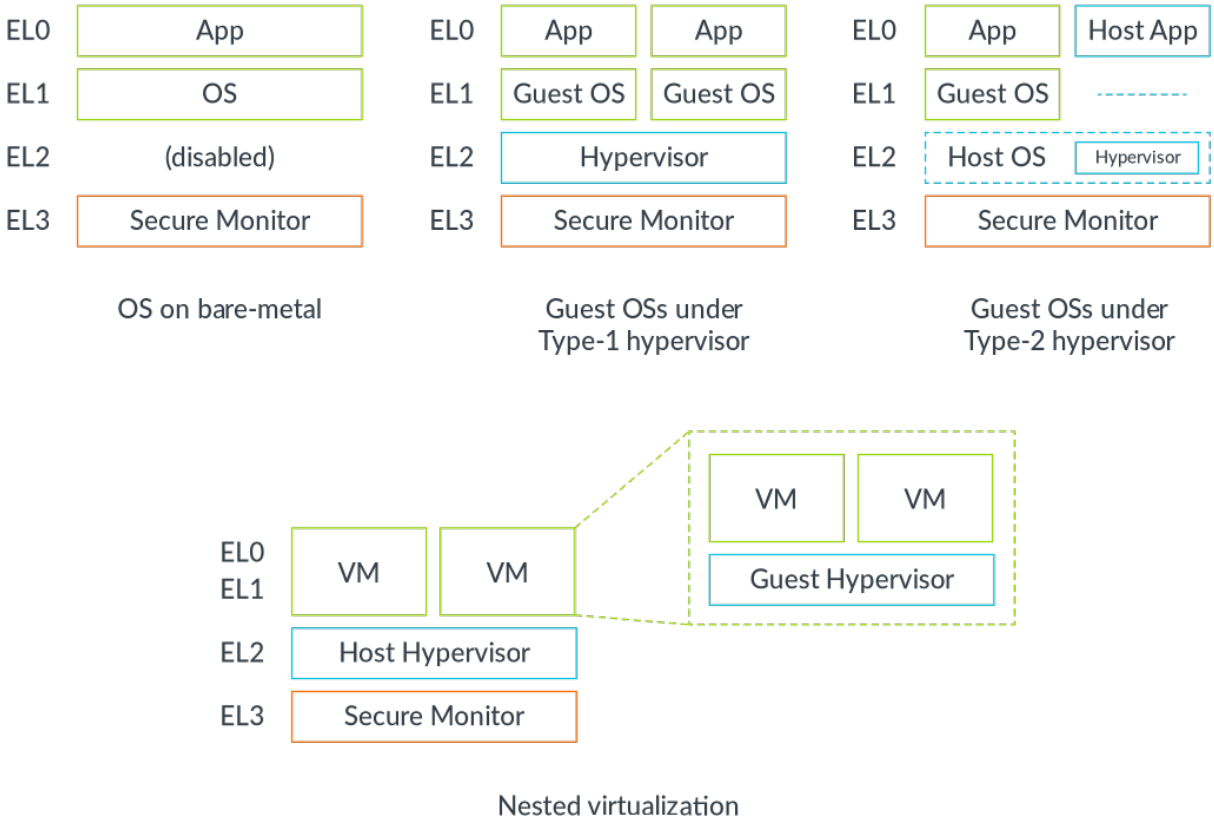

Figure 2: Nested virtualization

As previously outlined, the software view the OS has of the system must be agnostic to the hosting approach. This means the VM that the hypervisor exposes to the main operating system must meet certain minimum functional requirements that the operating system can rely on. For example, a minimum number of hardware breakpoints and watchpoints, a minimum number of performance monitors, a PSCI [8] interface for power management, a GIC interface for interrupts and so forth.

This specification covers the hardware requirements for this as well as the implied obligations on the hypervisor.

Providing these requirements enables software systems to be composed from operating systems and hypervisors from different providers.

Hypervisors use stage 2 translation table to create Virtual Machines (VMs). For example, one VM to run the main operating system and then other VMs to run security tasks. The stage 2 translation table might also be used to enforce certain security policies on the main operating system, such as only allowing certain areas of

memory to be executable when the policy engine in a security VM is satisfied. In client and edge devices, Arm does not expect these hypervisors to use the translation capability of the stage 2 translation table. Arm only expects these hypervisors to use the protection properties.

Hardware implementations must expect to support 4KB granules at stage 2, although it is expected that to reduce the performance overhead of virtualization, operating systems and hypervisors will work to keep the page size at stage 2 as large as possible for as much memory as possible.

In client devices, it is expected that most Non-secure devices will be assigned to the main operating system.

In general, to keep the other VMs and the hypervisor secure, all DMA must be behind a stage 2 Arm SMMU. The hypervisors will want to keep device specific code to a minimum. This means that stage 2 SMMU functionality must be compliant with the Arm architected SMMU.

2.3.3 Platform security functionality

TrustZone is an environment where Silicon providers and OEMs implement key platform-specific security functionality. This functionality is often a complex mix of security hardware and various software components, for example Trusted operating systems and applications, coming together to offer rich functionality, surfaced to the main operating system through APIs.

This functionality varies across platforms and is often differentiating, including examples such as key stores, biometric unlock, Secure video paths and electronic payment.

Arm has introduced Secure-EL2 (S-EL2) to the A-profile architecture, in Armv8.4, to help manage the increasing complexity of TrustZone Software and improve the overall system security.

Before S-EL2, all software running in S-EL1 had access to all of the Secure and Non-secure address spaces. On many systems that is a significant amount of code, possibly from multiple providers, with fundamental access to the system and therefore an increasing vulnerability.

2.3.3.1 Secure EL2

S-EL2 enables the introduction of a hypervisor that can control the Secure stage 2 translation table and therefore supports partitioning of the secure side software into different VMs. Each S-EL1 VM only has access to the address space that it needs to function. This introduces the security principle of hardware backed separation in TrustZone between Trusted operating systems.

Each secure side VM will be isolated from other secure side VMs. This means that each VM can reason about its security independent of the other VMs. This makes managing the security of TrustZone software significantly more scalable.

Another key advantage of S-EL2 is that the S-EL1 VMs can have their access of the non-secure address space limited. With the advent of ‘security VMs’ on the Non-secure side, this will be key to reasoning about the security of the Non-secure VMs without needing any knowledge of the Secure VMs. This is because S-EL2 will guarantee isolation.

Arm believes that the benefits of S-EL2 will best be realized when supported by standards. For example, the Firmware Framework for A-Profile systems [9] provides an ABI for communicating between VMs, whether the VMs are non-secure or secure.

Arm also believes that the client devices eco-system should move to firmware that is as standard as possible at EL3 and S-EL2. This will make it much easier to reason about the fundamentals of TrustZone security across the eco-system. Arm wants to build on the success of Arm Trusted Firmware being used as reference code for EL3 and include reference code for S-EL2.

Over time Arm hopes to encourage as much platform-specific code to be migrated out of EL3 and into a Secure-EL1 VM.

To support this standard firmware implementation this specification introduces a Secure timer, a Secure watchdog and a Secure UART when S-EL2 is present.

Specialist security hardware offers key functionality and protections in modern systems, for example hardware cryptography and key storage. This type of hardware will have drivers in TrustZone Software. APIs will surface core functionality through the standard firmware. Other functionality, for example Secure video playback, will be controlled by platform specific code in a S-EL1 VM and surfaced to the main operating system.

2.3.4 Memory partitioning

It is assumed that the device memory is partitioned into the Secure and Non-secure address spaces. There is no architectural support for moving memory from one address space to another. However, it is recognized that there can be system specific support for moving memory between address spaces, but great care needs to be taken to ensure memory remains coherent.

2.3.5 Peripheral subsystems

This specification requires implementation of standard UART and watchdog components for certain target markets. This is required to ease debugging and bring-up at early stages of boot, and to aid the implementation of standard firmware.

This specification requires key interface IP to implement register interfaces according to established standards. This is required to ease the development and portability of operating system specific device drivers

This specification provides specific requirements and guidance for successful integration of PCIe into Arm based systems.

3 Base System Architecture

The BSA describes the hardware features that are required to install, boot, and run an operating system on bare-metal or within a virtualization environment.

3.1 Approach

Arm BSA specifications are structured in terms of requirements for three software views of a system:

- **Operating system:** This view describes a base set of functionalities that an operating system, either running on bare-metal or under virtualization, can rely on.
- **Hypervisor:** This view describes a base set of functionalities that a Normal world hypervisor can rely on. It is consistent with this specification for a hypervisor to be hosted using nested virtualization.
- **Platform security functionality:** This view describes a base set of functionality that standard platform Secure firmware can rely on. Specifically, it provides requirements to support software running in Secure state.

The software views approach reflects that the software systems of today's devices are composed from software components that rely on these views and that each of these might be from different providers.

3.2 Scope

This document describes the features for a generic system based on the Arm 64-bit Architecture.

The rules applicable to a specific target market are described in the respective supplements:

- Server BSA [5]
- PC-BSA [7]

3.3 PE architecture

I_{XMTYC} The PEs that are referred to in this specification are those that are running the operating system or hypervisor, not PEs that are acting like devices.

3.3.1 Operating system

PEs in the base system are compliant with Armv8-A or Armv9-A [3] and the following is true:

$R_{B\_PE\_01}$ All PEs are architecturally symmetric except for the permitted exceptions listed in Section A.

I_{LTGZV} Mixing PEs based on Armv8 and Armv9 architectures in the same system is not allowed.

$R_{B\_PE\_02}$ The number of PEs must not exceed:

- Eight, when the interrupt controller is compliant to GICv2.
- 2^{28}, when the interrupt controller is compliant to GICv3 or higher.

I_{RPWMY} Some features required in this section represent a set of extensions or features, for example Advanced SIMD and floating-point support. In such cases, the Arm Architecture Reference Manual for A-profile architecture[3] describes the specific architecture extensions or features that are required.

$R_{B\_PE\_03}$ PEs must implement the Advanced SIMD (FEAT_AdvSIMD) and floating-point (FEAT_FP) support.

$R_{B\_PE\_04}$ PEs must support 4KB translation granules at stage 1.

$R_{B\_PE\_05}$ All PEs are coherent and in the same Inner Shareable domain.

$R_{B\_PE\_06}$ Where trade regulations allow, PEs must implement cryptography extension support for FEAT_AES, FEAT_SHA1 and FEAT_SHA256.

I_{TLHXL} It is recommended that FEAT_SM3 and FEAT_SM4 are also supported in hardware aimed at markets where they are used.

$R_{B\_PE\_07}$ PEs must implement little-endian support.

$R_{B\_PE\_08}$ PEs must implement EL1 and EL0 in the AArch64 Execution state.

$R_{B\_PE\_09}$ PEs must implement PMU extension version 3 (FEAT_PMUv3), and the base system must expose a minimum of four programmable PMU counters to the operating system.

$R_{B\_PE\_10}$ If the interrupt controller is compliant to GICv3 or higher, the PMU overflow signal from each PE must be wired to a unique PPI interrupt with no intervening logic.

$R_{B\_PE\_11}$ Each PE must implement a minimum of six breakpoints, two of which are context aware breakpoints (that is, can match virtual addresses, contextID, or VMID).

$R_{B\_PE\_12}$ Each PE implements a minimum of four synchronous watchpoints.

$R_{B\_PE\_13}$ PEs must implement the FEAT_CRC32 instructions.

$R_{B\_PE\_14}$ Implementation of SVE is optional. Implementation of SVE2 (FEAT_SVE2) is required for PEs that are based on the Armv9 architecture (see FEAT_SVE2 in [3]).

I_{BYZJZ} When SVE or SVE2 is implemented, it is recommended that the performance of well-optimized SVE or SVE2 code is no worse than code which uses the equivalent NEON instructions.

I_{YLSRX} SVE can only be supported on PEs that are based on Armv8 architecture. SVE2 can only be supported on PEs that are based on Armv9 architecture.

I_{VGRNL} Typical operating systems will not be able to take advantage of differences in maximum vector length among PEs.

I_{ZCSSW} If FEAT_PAuth (Pointer Authentication), mandatory from Armv8.3, is implemented, it is recommended that one of the standard algorithms defined by the Arm architecture [3] is implemented for address and generic authentication.

X_{NMKDY} Supporting an IMPLEMENTATION DEFINED algorithm is not recommended as it can adversely affect features that rely on architecture compatibility, for example software emulation and VM migration.

I_{BFLFZ} It is consistent with this specification to implement PEs with support for the AArch32 Execution state.

3.3.2 Hypervisor

The following additional requirements are applicable when standard hypervisor support is required:

$R_{B\_PE\_18}$ PEs must implement Non-secure EL2 in AArch64.

$R_{B\_PE\_19}$ PEs must support 4KB translation granules at stage 2.

$R_{B\_PE\_20}$ The translation granules supported at stage 2 must match those supported at stage 1.

$R_{B\_PE\_21}$ The base system must expose a minimum of two programmable PMU counters to a hypervisor.

I_{FRJMD} The combined requirements of operating system and hypervisor views is the implementation of a minimum of six programmable PMU counters.

$R_{B\_PE\_22}$ Two of the implemented breakpoints in each PE must be able to match on VMID. See B_PE_11.

3.3.3 Platform security functionality

The following additional requirements are needed to support software running in Secure state.

$R_{B\_PE\_23}$ PEs must implement EL3 in the AArch64 Execution state.

$R_{B\_PE\_24}$ PEs must implement Secure state.

I_{STMZB} From Armv8.4 onwards, if both EL2 and Secure state are implemented, then Secure EL2 must be implemented.

3.3.3.1 Expected usage of Secure state

I_{DKPVL} The base system is expected to use the PE EL3 and Secure state as a place to implement platform-specific firmware. The system might choose to implement further functionality in the Secure state, but this is beyond the scope of BSA specification.

I_{WJSQS} Arm does not expect PCI Express to be present in the Secure state. This is reflected in the GICv3 architecture, which does not support Secure LPI.

3.3.4 PE Architecture - future requirements

The following section lists a preview of new PE Architecture rules that will be required in a future version of this specification.

I_{JXYZL} When the Memory Tagging Extension is implemented, the implementation can be full, including instructions and checks, or just provide support for the instructions.

$R_{B\_PE\_16}$ If FEAT_MTE2 (Full Memory Tagging Extension) is implemented then:

- All general-purpose volatile host DRAM that can be used by an operating system for applications must support memory tagging.
- Dedicated memories for accelerators, or remote memory, or non-volatile memory do not need to support it. Firmware tables will indicate the memory ranges to the OS.

$R_{B\_PE\_17}$ If PEs implement the Scalable Vector Extension (SVE) and the Statistical Profiling Extension (SPE), the PEs must implement FEAT_SPEv1p1.

I_{XSJFL} Whether the instruction caches are implemented as VIPT or PIPT is IMPLEMENTATION DEFINED. Not all PEs are required to support the same instruction cache addressing scheme.

$R_{B\_PE\_25}$ All PEs must implement Large System Extensions (LSE) as indicated by ID_AA64ISAR0_EL1.Atomic = 0b0010. See FEAT_LSE in [3].

U_{QYMGJ} FEAT_LSE is a mandatory feature in Armv8.1 which introduces support for atomic instructions, thus enabling the use of atomic operations as a scalable method to synchronize access to shared resources from multiple clients. Implementations are encouraged to take advantage of this feature and ascertain that:

- Use of an atomic operation instead of exclusives for arbitrating access to a shared resource does not degrade performance. This should hold regardless of how contended the shared resource is.
- Use of an atomic operation instead of exclusives does not decrease the number of successful synchronization events per second that the system supports.

X_{XLHCJ} FEAT_LSE is a minimal boot requirement for some operating systems, such as Windows.

I_{DBKRP} It is recommended that all PEs implement Large System Extensions v2 (LSE2) as indicated by ID_AA64MMFR2_EL1.AT = 0b0001. See FEAT_LSE2 in [3].

X_{FNGTD} FEAT_LSE2 is used by software when available to improve performance for some use cases.

I_{HYRMQ} It is recommended that all PEs implement Load-Acquire RCpc instructions as indicated by ID_AA64ISAR1_EL1.LRCPC = b0001. See FEAT_LRCPC in [3].

R_{XRPZG} Each PE must implement a minimum of six breakpoints, with at least breakpoints 4 and 5 being context-aware breakpoints (that is, can match virtual addresses, contextID, or VMID).

I_{NBMNJ} Rule XRPZG replaces rule B_PE_11.

PE security requirements

PEs must implement the cache speculation side-channel attack mitigations that are introduced in Armv8.5:

$R_{B\_SEC\_01}$ PEs must implement the restrictions on speculation that are introduced in the Arm v8.5 extensions to the Arm architecture [3] and SCXTNUM_ELx registers as indicated by ID_AA64PFR0_EL1.CSV2==b0010 or ID_AA64PFR0_EL1.CSV2==b0011 and by ID_AA64PFR0_EL1.CSV3==b0001. See FEAT_CSV2 and FEAT_CSV3 in [3].

$R_{B\_SEC\_02}$ PEs implement the PSTATE/CPSR SSBS (Speculative Store Bypass Safe) bit and the instructions to manipulate it. See FEAT_SSBS in [3]. This is identified by ID_AA64PFR1_EL1.SSBS==b0010.

$R_{B\_SEC\_03}$ PEs implement the CSDB, SSBB and PSSBB barriers. [3].

$R_{B\_SEC\_04}$ PEs implement the FEAT_SB. This is indicated by ID_AA64ISAR1_EL1.SB== b0001.

$R_{B\_SEC\_05}$ PEs implement Enhanced Speculation Restriction FEAT_SPECRES or FEAT_SPECRES2 to restrict use of information gathered through control flow, data value prediction, or cache prefetch prediction, from affecting speculative execution. See FEAT_SPECRES and FEAT_SPECRES2 in [3].

I_{ZBXFJ} The mitigations described in B_SEC_01 to B_SEC_05 are introduced by the Armv8.5-A extension [3], and can be implemented on any prior version of Armv8.

3.4 Memory map

3.4.1 Operating system

I_{JZDLR} This specification does not require a standard memory map. Arm expects that the system memory map is described to system software by system firmware data.

X_{SCLSV} Systems will not necessarily fully populate all of the addressable memory space.

$R_{B\_MEM\_01}$ All memory accesses, whether they access memory space that is populated or not, must respond within finite time, to avoid the possibility of system deadlock.

I_{FFCQD} Compliant software must not make any assumptions about the memory map that might prejudice compliant hardware. For example, the full physical address space must be supported. There must be no dependence on memory or peripherals being located at certain physical locations.

$R_{B\_MEM\_02}$ Where a memory access is to an unpopulated part of the addressable memory space, accesses must be terminated in a manner that is presented to the PE as either a precise Data Abort, or as a system error interrupt, or an SPI, or LPI interrupt to be delivered to the GIC.

$R_{B\_MEM\_03}$ All Non-secure DMA requesters in a base system that are expected to be under the control of the operating system or hypervisor must be capable of addressing all of the Non-secure address space.

$I_{B\_MEM\_04}$ It is recommend that if the DMA requests from B_MEM_03 go through a SMMU then the requester is capable of addressing all of the Non-secure address space when the SMMU is turned off.

X_{FLXSP} A legacy OS, for example, one that does not support SMMUv3, might not support DMA remapping services for devices that are not capable of addressing the full system address space.

$R_{B\_MEM\_05}$ All PEs must be able to access all of the Non-secure address space.

$R_{B\_MEM\_06}$ Non-secure devices that cannot directly address all of the Non-secure address space must be placed behind a stage 1 SMMU that is compatible with the Arm SMMUv2 or SMMUv3 specification, that has an output address size large enough to address all of the Non-secure address space. See Section 3.7.

$R_{B\_MEM\_07}$ Where it is possible for the forward progress of a memory transaction to depend on a second memory access, the system must avoid deadlock if the memory access gets ordered behind the original transaction.

I_{WRTGF} The scenario described in B_MEM_07 can occur in PCI memory or I/O physical address space. For example, in the case of PCI, without an appropriate mitigation, a deadlock could arise if the memory access were also a PCI transaction, and therefore might be ordered behind the original PCI transaction. A system is permitted to resolve this dependency by terminating the memory accesses. A transaction that is terminated in this case might return any value, have any written data ignored, or be terminated with an error.

- For example, in line with the PCI ordering rules, the completion for a read of an SMMU or GIC table might be blocked behind an earlier inbound PCI transaction which the SMMU or GIC is blocking, until the table access completes.
- Because a system is permitted to avoid deadlock by terminating transactions in this way, system software must not allocate any structures that relate to a SMMU or GIC in PCIe address space or I/O address space.

3.4.2 Platform security functionality

The following additional requirements are needed to support software running in Secure state.

$R_{B\_MEM\_08}$ The system must provide some memory that is mapped in the Secure address space.

$R_{B\_MEM\_09}$ This Secure memory must not be aliased in the Non-secure address space.

I_{BXSCJ} The amount of Secure memory that is provided is platform-specific. This is because the intended use of the memory is for platform-specific firmware.

3.5 Interrupt controller

3.5.1 Operating system

$R_{B\_GIC\_01}$ A base system must present OSs and hypervisors with the interfaces defined by the one of the following:

- A Generic Interrupt Controller (GIC) v2 interrupt controller.
- A GICv2 interrupt controller with GICv2m extension.
- A GICv3, or higher, interrupt controller.

$R_{B\_GIC\_02}$ Table 2 shows the limitations and valid configurations allowed in a base system.

Table 2: GIC valid configurations

GIC Arch	Limitations: number of PE	PCIe MSI[X] Support
v2	8	No PCIe support
v2 + v2M	8	MSI[X] to SPI. See Section I
v3 without ITS	2^{28}	Valid only if no PCIe
v3 + ITS	2^{28}	Full MSI[X] support

I_{YHJCD} GICv3 in this context refers to GICv3 or higher architecture.

I_{FQFPG} It is possible for v2M to be used with GICv3 (without ITS). While this configuration is supported by operating systems, it can be problematic especially with virtualization, depending on the number of PCIe MSI[X] frames needed in the platform. Because of this, the configuration is not recommended and should be limited to platforms that include GICv3 and cannot support ITS, but still want to enable PCIe with MSI[X].

$R_{B\_GIC\_03}$ If the system includes PCI Express and GICv3 interrupt controller is supported, then the GICv3 interrupt controller must implement ITS and LPI.

$R_{B\_GIC\_04}$ If a GICv3 interrupt controller is supported, then the interrupt controller must be configured to support two Security states.

X_{BJQLV} GIC support for Secure state is required because PEs are required to implement Secure state in compliance to B_PE_23 and B_PE_24. For corresponding GICv2 requirements, see Section I.

I_{YZMBR}
- The Arm PL011 UART requires a level interrupt for legacy interrupt support.

For a base system implementing a GICv3 interrupt controller and PCI Express:

- It is recommended that MSI and MSI-X are mapped to LPI interrupts.
- It is permissible to build a system with no SPI for PCIe peripherals. However, Arm expects that the peripheral ecosystem will continue to rely on wired level interrupts, and expects that most systems to support SPI and LPI interrupts. PCI legacy INTA-INTD interrupts are wired interrupts. See Section E.6 for more details.
- A PCIe root complex requires a level interrupt for legacy interrupt support.

3.5.2 Platform firmware

$R_{B\_GIC\_05}$ The system shall implement at least eight Non-secure SGIs, assigned to interrupt IDs 0-7.

3.6 PPI assignments

$R_{B\_PPI\_00}$ A base system must map the interrupts shown in Table 3, Table 4 and Table 5 to PPI.

I_{QKQRP} The interrupt IDs in an implementation are allowed to be different from the recommended values that are given in the following tables. However, assignment of the PPIs within the base PPI INTID range of 16 - 31 is recommended. Failure to use these values might result in compatibility issues where software is expecting these values.

$R_{B\_PPI\_01}$

Table 3: PPI assignments for operating system

Recommended Interrupt ID	Interrupt	Description
30	Overflow interrupt from CNTP	Non-secure physical timer interrupt.
27	Overflow interrupt from CNTV	Virtual timer interrupt.
24	CTIIRQ	Cross Trigger Interface (CTI) interrupt.
23	Performance Monitors Interrupt	Indicates an overflow condition in the Performance monitoring unit.
22	COMMIRQ	Debug Communication Channel (DCC) interrupt.
21	PMBIRQ	Statistical profiling Interrupt, if Statistical Profiling Extensions are implemented.

I_{WVDDR} An implementation might need to reserve a PPI for trace buffer overflow.

$R_{B\_PPI\_02}$

Table 4: PPI assignments for Hypervisor

Recommended Interrupt ID	Interrupt	Description
28	Overflow interrupt from CNTHV	Non-secure EL2 virtual timer interrupt (if PEs are Armv8.1 or greater).
26	Overflow interrupt from CNTHP	Non-secure EL2 physical timer interrupt.
25	GIC Maintenance interrupt	The virtual PE interface list register overflow interrupt.

$R_{B\_PPI\_03}$

Table 5: PPI assignments for platform security functionality

Recommended Interrupt ID	Interrupt	Description
29	Overflow interrupt from CNTPS	Secure Physical timer interrupt.
20	CNTHPS	Secure EL2 physical timer interrupt (if Secure EL2 is implemented).
19	CNTHVS	Secure EL2 virtual timer interrupt (if Secure EL2 is implemented).

3.7 System MMU and device assignment

I_{KCXVJ} The base system might implement an IMPLEMENTATION DEFINED number of SMMU components. Arm expects that these components will be described by system firmware data along with a description of how to associate them with the devices they police.

I_{HGBYK} SMMUv3 is not backwards compatible with SMMUv2.

I_{LSFHM} If a system implements PCIe, it is recommended that the SMMU is compliant to SMMUv3 or higher.

I_{GNMVY} SMMUv2 does not have PCI Express ATS support. Standard PCI Express ATS support is included in SMMUv3.

$R_{B\_SMMU\_01}$ All the System MMUs presented to an OS or hypervisor must be compliant with the same architecture version.

$R_{B\_SMMU\_02}$ The SMMU must support the translation granule sizes that are supported by the PEs.

X_{QGMSR} A device behind an SMMU must be able to see the whole non-secure physical address space that is visible to PEs.

$R_{B\_SMMU\_06}$ This means that, if a system supports a System Physical Address Space (SPAS) of 52-bits, then:

- PEs are required to implement FEAT_LPA or FEAT_LPA2, and 52-bit physical address range (ID_AA64MMFR0_EL1.PARange = 0b0110),
- The SMMU must support a 52-bit output address size (SMMU_IDR5.OAS = 0b0110).

Otherwise, the SPAS is less than the smaller of ID_AA64MMFR0_EL1.PARange and SMMU_IDR5.OAS, and the machine must have all physical RAM and device mappings below this limit.

I_{TKSQQ} It is recommended that ID_AA64MMFR0_EL1.PARange and SMMU_IDR5.OAS match when describing the addressable memory in a system.

U_{FVJKL} Operating systems can discover the SPAS as advertized by the system firmware data. See [4].

I_{PPJRT} FEAT_LPA [3] and FEAT_LPA2 [3] expand the maximum physical address width from 48 to 52 bits. Using 52-bit PA with FEAT_LPA requires 64KB translation granules. FEAT_LPA2 extends that capability to systems with 16K and 4KB translation granules.

3.7.1 Operating system

I_{JRQMN} If a device is subject to stage 1 translation, allocation of memory attributes, and application of permission checks then this specification collectively refers to this translation, attribution, and permission checking as stage 1 policing. The act of stage 1 policing is called stage 1 System MMU functionality.

$R_{B\_SMMU\_07}$ Devices that operate across non-contiguously allocated memory require stage 1 System MMU functionality.

I_{SVMJD} Stage 1 System MMU functionality visible only to device specific drivers, is IMPLEMENTATION DEFINED.

$R_{B\_SMMU\_08}$ If Secure-EL2 is not implemented, stage 1 System MMU functionality that is made visible to an operating system must present the interface of a System MMU compatible with one of the following:

- The SMMUv2 specification, where each context bank must present a unique physical interrupt to the GIC.
- The Arm SMMUv3 specification or higher, where the integration of the System MMUs is compliant with rules SMMU_01 and SMMU_02 as specified in “SMMUv3 integration”, Section D.

$R_{B\_SMMU\_12}$ For any device that is behind an SMMU, all accesses output from the device must be presented to the SMMU regardless of the specified address.

I_{FRSVR} Software can either program stage 1 System MMUs to use the same page tables as the PE or build shadow page tables.

I_{TKGRM} MSIs from a PCI Express device are translated in the same way as any other writes from that device.

I_{YJKWT} Support for broadcast TLB maintenance operations is not required.

3.7.2 Hypervisor

I_{BSBXS} It is IMPLEMENTATION DEFINED whether any given device in a system supports the ability to be hardware virtualized, for example SR-IOV. Arm expects that devices that can be hardware virtualized have that property expressed either by system firmware data, or though hardware discoverability.

$R_{B\_SMMU\_16}$ If a device is assigned and passed through to an operating system under a hypervisor, then the memory transactions of the device must be subject to stage 2 translation, allocation of memory attributes, and application of permission checks, under the control of the hypervisor.

I_{HGSML} This specification collectively refers to this translation, attribution, and permission checking as *policing*. The act of policing is called stage 2 System MMU functionality.

$R_{B\_SMMU\_17}$ From a hardware perspective, this means that a base system supporting a protection hypervisor requires all non-secure DMA capable devices that will be assigned to a non-secure VM for direct control to be policed by stage 2 System MMU functionality.

$R_{B\_SMMU\_18}$ If Secure-EL2 is not implemented, stage 2 System MMU functionality must be provided by a System MMU compatible with the Arm SMMUv2 specification or Arm SMMUv3 specification.

$R_{B\_SMMU\_19}$ When stage 2 System MMU functionality is provided by a System MMU compatible with the Arm SMMUv2 specification, each context bank must present a unique physical interrupt to the GIC.

$R_{B\_SMMU\_21}$ When stage 2 System MMU functionality is provided by a System MMU compatible with the Arm SMMUv3 spec:

- The integration of the System MMUs is compliant with rules SMMU_01 and SMMU_02 as specified in “SMMUv3 integration”, Section D.

3.7.3 System MMU and device assignment - future requirements

The following section lists a preview of SMMU and device assignment rules that will be required in a future version of this specification.

$R_{B\_SMMU\_03}$ if PEs implement FEAT_LVA (ID_AA64MMFR2_EL1.VARange = 0b0001), the SMMU must support extended virtual addresses (SMMU_IDR5.VAX = 0b01).

I_{FVFVN} Armv8.4 introduces TLB Invalidation instructions which apply to a range of input addresses rather than just to a single address.

$R_{B\_SMMU\_04}$ If PEs use Armv8.4 and can issue TLB range invalidation instructions, the SMMU must support range invalidation.

$R_{B\_SMMU\_05}$ All DVM receivers visible to normal world software in a system must receive all DVM messages initiated by a DVM requester. This would require that the DVM capabilities of the SMMU and the interconnect are the same or a superset of the initiator (typically a PE).

U_{GHBPB} DVM is an important capability for enabling and optimizing virtualization features such as Shared Virtual Memory (SVM). Implementations are encouraged to ascertain that the system provides correct end-to-end support for DVM operations.

$R_{B\_SMMU\_09}$ If Secure-EL2 is implemented, stage 1 System MMU functionality that is made visible to an operating system must present the interface of a System MMU compatible with the Arm SMMUv3.2, or higher, architecture revision where:

- The integration of the System MMUs is compliant with rules SMMU_01 and SMMU_02 as specified in “SMMUv3 integration”,Section D.
- SMMU implementations must provide level 1 or level 2 support for page table resizing. It is recommended that the SMMU implements level 2. If the SMMU implementation provides level 2, then it is recommended that the PE also provides level 2.

X_{NRTPX} MPAM architecture requires that all requesters that can access an MPAM controlled resource must support passing MPAM ID information.

$R_{B\_SMMU\_11}$ Therefore, a SMMUv3.2, or higher, implementation must support the MPAM extension if the requests it serves access MPAM controlled resources.

X_{RZLNV} To facilitate page table sharing between PE and SMMU, the SMMU features must match the PE features:

$R_{B\_SMMU\_13}$ If PE supports 16-bit ASID, The SMMU must implement support for 16-bit ASID.

$R_{B\_SMMU\_14}$ The SMMU will support little-endian for translation table walks, and at a minimum must match the endianness support of the PEs.

$R_{B\_SMMU\_23}$ If PE supports 16-bit VMID, the SMMU must implement support for 16-bit VMID.

$R_{B\_SMMU\_20}$ If Secure EL2 is implemented, stage 2 System MMU functionality that is made visible to a hypervisor must present the interface of a System MMU compatible with the Arm SMMUv3.2, or higher, architecture revision where:

- The integration of the System MMUs is compliant with rules SMMU_01 and SMMU_02 as specified in “SMMUv3 integration”,Section D.
- SMMU implementations must provide level 1 or level 2 support for page table resizing. It is recommended that the SMMU implements level 2. If the SMMU implementation provides level 2, then it is recommended that the PE also provides level 2.

I_{SCHPZ} Software can either program stage 2 System MMUs to use the same translation tables as the PE or build shadow translation tables.

I_{JYTPF} For systems that implement SMMU architecture revision 3.2, if the PEs do not implement secure EL2, the SMMUs are not required to implement Secure stage 2.

I_{DVFSJ} It is IMPLEMENTATION DEFINED whether any given Secure device in a base architecture system supports the ability to be hardware virtualized. It is expected that devices that can be hardware virtualized have that property expressed either by system firmware data, or through hardware discoverability.

$R_{B\_SMMU\_24}$ If Secure-EL2 is implemented, all secure DMA capable devices that can be assigned to a Secure VM must be policed by stage 2 Secure SMMU functionality.

$R_{B\_SMMU\_25}$ If Secure-EL2 is implemented, stage 2 Secure MMU functionality must be provided by a System MMU compatible with the Arm SMMUv3.2, or higher, architecture revision where:

- The integration of the System MMUs is compliant with rules SMMU_01 and SMMU_02 as specified in “SMMUv3 integration”,Section D.
- SMMU implementations must provide level 1 or level 2 support for page table resizing.

3.8 Clock and timer subsystem

3.8.1 Operating system

$R_{B\_TIME\_01}$ The base system must include the system counter of the Generic Timer, as specified in the Arm ARM [3].

$R_{B\_TIME\_02}$ The system counter of the Generic Timer must run at a minimum frequency of 10MHz.

$R_{B\_TIME\_03}$ The system counter of the Generic Timer must not roll over inside a 10-year period.

$R_{B\_TIME\_04}$ The architecture of the system counter of the Generic Timer mandates that the counter must be at least 56 bits, and at most 64 bits. From Armv8.4, for systems that implement counter scaling, the minimum becomes 64 bits.

I_{WLTBZ} As part of the Generic Timer subsystem. the Generic Timer system counter also exports its count value, or an equivalent encoded value, through the system to the timers in the PEs.

$R_{B\_TIME\_05}$ The count exposed by the Generic Timer system counter must be available to the PE timers when PE timers are active.

I_{DJQBG} The local PE timers have a programmable count value. When the value expires it generates a Private Peripheral Interrupt for the associated PE.

I_{HKNFT} The local PE timers can be built so that they are always on. This property is described in the system firmware data.

I_{ZDCPR} A PE timer is always on from the perspective of the operating system if it can count, generate interrupts and wake the PE regardless of the PE’s power state.

$R_{B\_TIME\_06}$ Unless all the local PE timers are always on, the base system must implement a system wakeup timer that can be used when PE timers are powered down.

$R_{B\_TIME\_07}$ If the system includes an operating-system-visible system wakeup timer, it must be in the form of the memory mapped timer that is described in the Armv8 ARM [3].

$R_{B\_TIME\_08}$ If the system includes an operating-system-visible system wakeup timer, it must generate an interrupt on timer expiry that must be wired to the GIC as an SPI. This system wakeup timer can also be used to wake up PEs. See Section 3.9.

X_{YJTGT} It is recognized that in large system a shared resource like the system wakeup timer can create a system bottleneck. This is because access to it must be arbitrated through a system-wide lock. It is anticipated that this can be dealt with by the platform by having the firmware tables describe the PE timers as always on and remove the need for the an operating-system-visible system wakeup timer.

$R_{B\_TIME\_09}$ The platform will either implement hardware always-on PE timers or use the platform firmware to save and restore the PE timers in a performant way.

I_{GVPHT} The term 'platform firmware' in B_TIME_09 refers to any firmware executing on a PE or on a system control processor, which has sufficient privilege to save and restore the PE timers with or without hardware assistance. The term 'performant' in B_TIME_09 implies that the cost and latency of saving and restoring the PE timers should not be detrimental to the overall requirements of the market segment of the product. B_TIME_09 does not preclude platform-specific mechanisms from assisting the PE timers in providing always on behavior from the perspective of the operating system, including propagating the wakeup interrupt.

I_{WYFCQ} The wakeup timer does not require a virtual timer to be implemented and it is permissible for the virtual offset register to Read-as-Zero. Writes to the virtual offset register in CNTCTLBase frame are ignored. The timer is not required to have a CNTEL0Base frame.

$R_{B\_TIME\_10}$ If the system includes an operating-system-visible system wakeup timer, this memory-mapped timer must be mapped onto Non-secure address space. This is called the *Non-secure system wakeup timer*.

Table 6 summarizes which address space the register frames must be mapped to.

Table 6: Generic counter and timer memory mappings

Register Frame	
CNTControlBase	Secure
CNTReadBase	Not required
CNTCTLBase	Non-secure and Secure
CNTBaseN	Non-secure

I_{TYTGM} The Generic Timer register frames in Table 6 are described in the Arm ARM [3].

3.9 Wakeup semantics

X_{XMFMS} Systems implement many different power domains and power states. It is important for the OS or hypervisor, or both, to understand the relationship between these power domains and the facilities a system has for waking PEs from various low power states.

I_{DCZSM} A key component in controlling the entry to and exit from low-power states is the IMPLEMENTATION DEFINED power controller. The power controller controls the application of power to the various power domains. On entry to low-power states hardware-specific software will program the power controller to take the correct action. On exit from a low-power state, hardware-specific software might need to reprogram the power controller. Hardware-specific software is required to save and restore system state when entering and exiting some low-power states.

I_{XWVGL} This specification defines two classes of wakeup methods: interrupts, and always-on power domain wake events.

The first class of wakeup methods are interrupts. This specification defines interrupts that wake PEs as wakeup interrupts.

A wakeup interrupt is any interrupt that is any one of the following:

- An SPI that directly targets a PE.
- An SGI.
- A PPI.

- An LPI

For an interrupt to be a wakeup interrupt, it must also be enabled in the distributor.

$R_{B\_WAK\_01}$ A PE must wake in response to a wakeup interrupt, independent of the state of its PSTATE interrupt mask bits, which are the A, I, and F bits, and of the wakeup interrupt priority.

I_{FYLNP} Typically, a wakeup signal is exported from the GIC to the power controller to initiate the PE wakeup.

There are some power states where a PE will not wake on an interrupt. It is the responsibility of system software to ensure there are no wakeup interrupts targeting a PE entering these states. See Table 7.

I_{GMWJS} The local PE timers are an important source of interrupts that can wake the PE. However, the local PE timer might be powered down in some low-power states. This is because the local PE timer might be in the same power domain as the PE. In low-power states where the local PE timer is powered down, system software can use an SGI from other running PEs to wake the PE, or system software can configure the system wakeup timer to send a wakeup interrupt to the PE to wake it.

In some very deep low power states, the GIC will be powered down. To wake from these states, another class of wakeup methods can be used: always-on power domain wake events.

$R_{B\_WAK\_02}$ If the system supports a low-power state where the GIC is powered down, then there must be an IMPLEMENTATION DEFINED way to program the power controller to wake a PE on expiry of the system wakeup timer or the generic watchdog. In this scenario, the system wakeup timer or generic watchdog is still required to send its interrupt.

I_{TQSGD} There might be other IMPLEMENTATION DEFINED always-on power domain wakeup events that can wake PEs from deep low-power states, for example PCI Express wakeup events and Wake-on-LAN.

I_{SRNTZ} See Section 3.10 for a description of the power state semantics that the system must comply with.

$R_{B\_WAK\_03}$ Whenever a PE is woken from a sleep or off state the OS or hypervisor must be presented with an interrupt so that the PE software can determine which device requested the wakeup.

$R_{B\_WAK\_04}$ The interrupt must be pending in the GIC at the point that control is handed back to the OS or hypervisor from the system-specific software performing the state restore.

$R_{B\_WAK\_05}$ This interrupt must behave like any other: a device sends an interrupt to the GIC, and the GIC sends the interrupt to the OS or hypervisor. The OS or hypervisor is not required to communicate with a system-specific interrupt controller.

$R_{B\_WAK\_06}$ If the wakeup event is an edge, then the system must ensure that this edge is not lost. The system must ensure that the edge wakes the system and is subsequently delivered to the GIC without losing the edge.

I_{XLLXJ} An example of an expected chain of events would be:

1. Wakeup event occurs, for example GPIO or wake-on-LAN.
2. The power controller responds by powering on the necessary resources that include the PE and the GIC.
3. The PE comes out of reset and hardware-specific software restores state, including the GIC.
4. An interrupt is presented to the GIC representing the wakeup event. In many situations this might be exactly the same signal as the wakeup event.
5. The system must ensure that, by the time the hardware-specific restore software has delegated to the OS or hypervisor, the interrupt is pending in the GIC.
6. The OS or hypervisor can respond to the interrupt.

3.10 Power state semantics

I_{NYFCQ} This specification does not require a given hierarchy of power domains, but there are some rules and semantics that must be followed.

Figure 3 shows an example block diagram showing a possible hierarchy of power domains. Note that there are other examples that conform to this specification that are not subsets of the system in the diagram.

$R_{B\_WAK\_07}$ In order for either the OS or hypervisor, or both, to be able to reason about wakeup events and to know which timers will be available to wake the PE, all PEs must be in a state that is consistent with one of the semantics described in Table 7 and Table 8.

I_{CDVWN} All PEs do not need to be in the same state. Arm expects that the semantics of the power states that a system supports will be described by system firmware data. Table 9 describes the power state semantics in a set of component-specific rules.

$R_{B\_WAK\_08}$ System MMUs and GICv3 make use of tables in memory in the power states where GIC is 'On'. For this type of state, system memory must be available and will respond to requests without requiring intervention from software running on the PEs.

I_{SVJXJ} Hardware-specific software is required to save and restore system state when entering and exiting low-power states.

X_{JTMKC} It is highly likely that many systems will support very low-power states where most system logic is powered down and the system memory is in self-refresh, but the OS retains control over future wakeup. This is reflected in power state semantic E.

I_{YPNLP} In this state, the GIC can be powered off after system software has saved its state. In this state, wakeup signals go straight to the system power controller and do not require use of the GIC to wake the PEs. The system power controller is system specific. When in a power state of semantic E, the system power controller wakes an IMPLEMENTATION DEFINED PE, or set of PEs, when the system wakeup timer expires. Other system-specific events might also cause wakeup from this state, for example a PCI Express wakeup event. The events that will cause wakeup from this state are expected to be discoverable from system firmware data.

$R_{B\_WAK\_09}$ When the system is in a state where the GIC is powered down, devices must not send messaged interrupts to the GIC. See Table 8.

I_{QSMNJ}

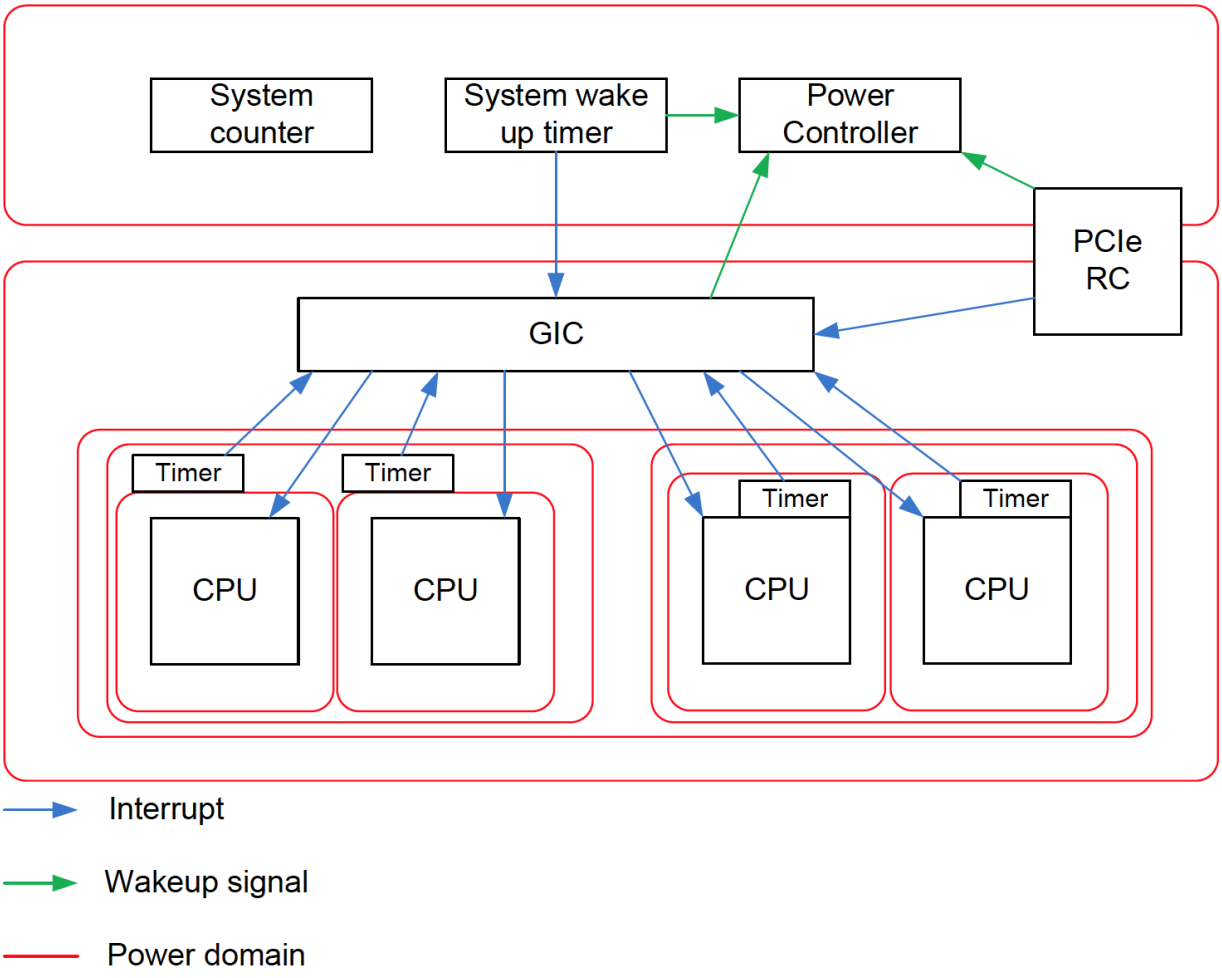

Figure 3: Example system block diagram showing power domains and timer hierarchy

Table 7: PE power states

PE State	Description
Run	The PE is powered up and running code.
Idle_standby	The PE is in STANDBYWFI state, but remains powered up. There is full state retention, and no state saving, or restoration are required. Execution automatically resumes after any interrupt or external debug request (EDBGRQ). Debug registers are accessible.
Idle_retention	The PE is in STANDBYWFI state, but remains powered up. There is full state retention, and no state saving, or restoration are required. Execution automatically resumes after any interrupt or external debug request (EDBGRQ). Debug registers are not accessible.
Sleep	The PE is powered down but hardware will wake the PE autonomously, for example, on receiving a wakeup interrupt. No PE state is retained. State must be explicitly saved. The woken PE starts execution at the reset vector, and then hardware-specific software restores state.

PE State	Description
Off	The PE is powered down and is not required to be woken by interrupts. The only way to wake the PE is by explicitly requesting the power controller, for example, from system software running on another PE, or an external source like a poweron_reset. This state can be used to support hot remove of PE. No PE state is retained.

I_{KTDGP} The Timers column in Table 8 applies to system wake timers, system counters and generic watchdogs.

$R_{B\_WAK\_10}$

Table 8: Power state semantics

Semantic	PE and GIC PE interface	PE timers	GIC distributor	Timers	Note
A	Run	On	On	On	-
B	Idle	On	On	On	PE will resume execution on receipt of any interrupt.
C	Sleep	On	On	On	PE will wake on receipt of a wakeup interrupt.
D	Sleep	Off	On	On	PE will wake on receipt of a wakeup interrupt, but local timer is off.
E	Sleep	Off	Off	On	PE will wake from system timer wakeup event or other system specific events.
F	Off	Off	On	On	Some, but not all, PEs are in Off state.
G	Off	Off	Off	Off	All PEs in Off state.
H	Sleep	On	Off	On	PE will wake from PE timer, system timer wakeup event or other system specific events.
I	Idle	Off	On	On	PE will resume execution on receipt of any interrupt, but the local timer is off.

$R_{B\_WAK\_11}$

Table 9: Component power state semantics

PE and GIC PE Interface	Individual PEs and their associated GIC PE interface can be in Run, Idle, Sleep or Off state.
PE timers	Must be On if the associated PE is in the Run state. Might be On or Off if the PE is in Idle or Sleep state. Must be Off if the PE is in the Off state.
GIC Distributor	Must be On if any PE is in the Run or Idle state. Might be On or Off if all PEs are in either the Sleep or Off state, with at least one PE in the Sleep state. Must be Off If all PEs are in the Off state.
System wakeup timers and system counter and generic watchdog	Must be On if any PE is not in the Off state. Must be Off if all PEs are in the Off state.

3.11 Watchdogs

$R_{B\_WD\_00}$ Implementation of a Watchdog is OPTIONAL. If implemented, the following rules apply: B_WD_01, B_WD_02, B_WD_03, B_WD_04, and B_WD_05.

$R_{B\_WD\_01}$ The Generic Watchdog must be implemented as specified in Section C.

$R_{B\_WD\_02}$ The watchdog must have both its register frames mapped onto Non-secure address space. This watchdog is referred to as the Non-secure watchdog.

$R_{B\_WD\_03}$ Watchdog Signal 0 is routed as an SPI to the GIC and it is expected this will be configured as a Non-secure EL2 interrupt, directly targeting a single PE.

I_{LCSWL} In this context, *platform* means any entity that is more privileged than the code running at Non-secure EL2. Examples of the platform component that services Watchdog Signal 1 are: EL3 system firmware, or a system control processor, or dedicated reset control hardware.

$R_{B\_WD\_04}$ Watchdog Signal 1 must be routed to the platform.

$R_{B\_WD\_05}$ The action taken on the raising of Watchdog Signal 1 is platform-specific.

I_{QRDNJ} Only directly targeted SPI are required to wake a PE; see Section 3.9 for more information. Programming the watchdog SPI to be directly targeted ensures delivery of the interrupt independent of PE power states. However, it is possible to use a 1 of N SPI to deliver the interrupt, if one of the target PEs is running.

3.12 Peripheral subsystems

$R_{B\_PER\_01}$ If the system has a USB2.0 host controller peripheral it must conform to EHCI v1.1 or later.

$R_{B\_PER\_02}$ If the system has a USB3.0 host controller peripheral it must conform to XHCI v1.0 or later.

$R_{B\_PER\_03}$ If the system has a SATA host controller peripheral it must conform to AHCI v1.3 or later.

$R_{B\_PER\_04}$ Peripheral subsystems which do not conform to rues B_PER_01, B_PER_02 or B_PER_03 are permitted, if those peripherals are not required to boot and install an OS.

I_{FPJBF} A base system is recommended to have a UART for the purpose of system development and bring up.

$R_{B\_PER\_05}$ If the base system includes a UART that is:

- OS enumerable for use as OS console or debug port, or
- user accessible serial port

then the UART must be one of:

- The Generic UART as specified in Section B.
- A fully 16550 compatible UART [10].

$R_{B\_PER\_06}$ The UART interrupt output is connected to the GIC as an SPI.

$R_{B\_PER\_07}$ The UART must be mapped onto Non-secure address space. This is called the Non-secure UART.

$R_{B\_PER\_09}$ The memory attributes of DMA traffic must be one of the following:

- Inner Write-Back, Outer Write-Back, Inner Shareable.
- Inner Non-cacheable, Outer Non-cacheable.
- A device type.

$R_{B\_PER\_10}$ I/O coherent DMA traffic must have the attribute - Inner Write-Back, Outer Write-Back, Inner Shareable.

3.12.1 Platform security functionality

$R_{B\_PER\_11}$ If a TCG TPM based security model is supported, the base system needs to provide a TPM implementation that is compliant to TPM Library Specification, Family 2.0 [12].

3.12.2 PCIe integration

$R_{B\_PER\_08}$ If the system has a PCI Express root complex then it must comply with the rules in Section E.

$R_{B\_PER\_12}$ To ensure standard software support, a device claiming to follow the PCI Express specification [1] must follow all the rules in the PCIe specification [1] which are software-visible.

I_{MCTKP} PCI Express integration is covered Section E. Device assignment requirements are covered in Section E.7

3.12.2.1 PCIe integration - Future requirements

The following section lists a preview of new PCIe integration rules that will be required in a future version of this specification. These rules might be integrated into Section E in the future.

R_{BJLPB} If Function Readiness Status (FRS) is supported in a Root Port, then the Root Port must support MSI or MSI-X to generate interrupts.

I_{ZYCFN} Rule BJLPB relate to PCI_MSI_1 and PCI_MSI_2 in Section E.

$R_{B\_PCIe\_10}$ If Steering Tags are supported by the system, the STE.DCP (Directed Cache Prefetch) control bit in the System MMU must be honoured to ensure the feature can be enabled and disabled architecturally.

$R_{B\_PCIe\_11}$ The mechanisms for supporting Steering Tags are IMPLEMENTATION DEFINED, but must have the following properties:

- Steering Tags must be treated as cache allocation hints and must only apply to Normal Cacheable memory transactions in the shareability domain of the I/O Subsystem.
- Steering Tags must not alter coherency guarantees.
- Steering Tags are permitted to be ignored.
- Steering Tag value properties for the platform must be discoverable by software as advertised by firmware

I_{WMWSM} A Steering Tag value of 0 indicates no Steering Tag preference and is treated as if no Steering Tag is provided.

I_{JWHKH} Rules B_PCIe_10 and B_PCIe_11 relate to KNMPH in Section E.

3.13 Presenting an on-chip peripheral as PCIe device - future requirements

The following section lists a preview of new rules that will be required in a future version of this specification.

I_{JSVYQ} There are two options for presenting on-chip peripherals as a PCIe device.

- Option 1: Root Complex Integrated Endpoint (RCiEP)
- Option 2: Integrated Endpoint (i-EP)

$R_{B\_REP\_1}$ The rules to implement an RCiEP device are described in Section F.1, Section F.2, and Section G.1

$R_{B\_IEP\_1}$ The rules to implement an i-EP device are described in Section F.1, Section F.3, and Section G.2

3.13.1 Option 1: Root Complex Integrated Endpoint (RCiEP)

I_{GMNTP} In this option, the on-chip peripheral appears to software during enumeration as a Root Complex Integrated EndPoint (RCiEP) that is connected to the root bus behind a host bridge. Figure 4, Figure 5, and Figure 6 show how the peripheral is visible to software with this option. These examples are not an exhaustive list of RCiEP and RCEC configurations. These are examples of designs and combinations that adhere to this specification and the PCIe specification [1].

I_{ZCKKT} In this option, a Root Complex Event Collector (RCEC) must be present if any of the following is true:

- Any of the functions in the RCiEP must implement Advanced Error Reporting (AER) or PCIe baseline error reporting. See Section 6.2 of the PCIe specification [1] for details on baseline and Advanced error reporting [1].
- The RCiEP must implement PCIe PME event signaling. See Section 6.1.6 of the PCIe specification [1] for details on PCIe PME support.

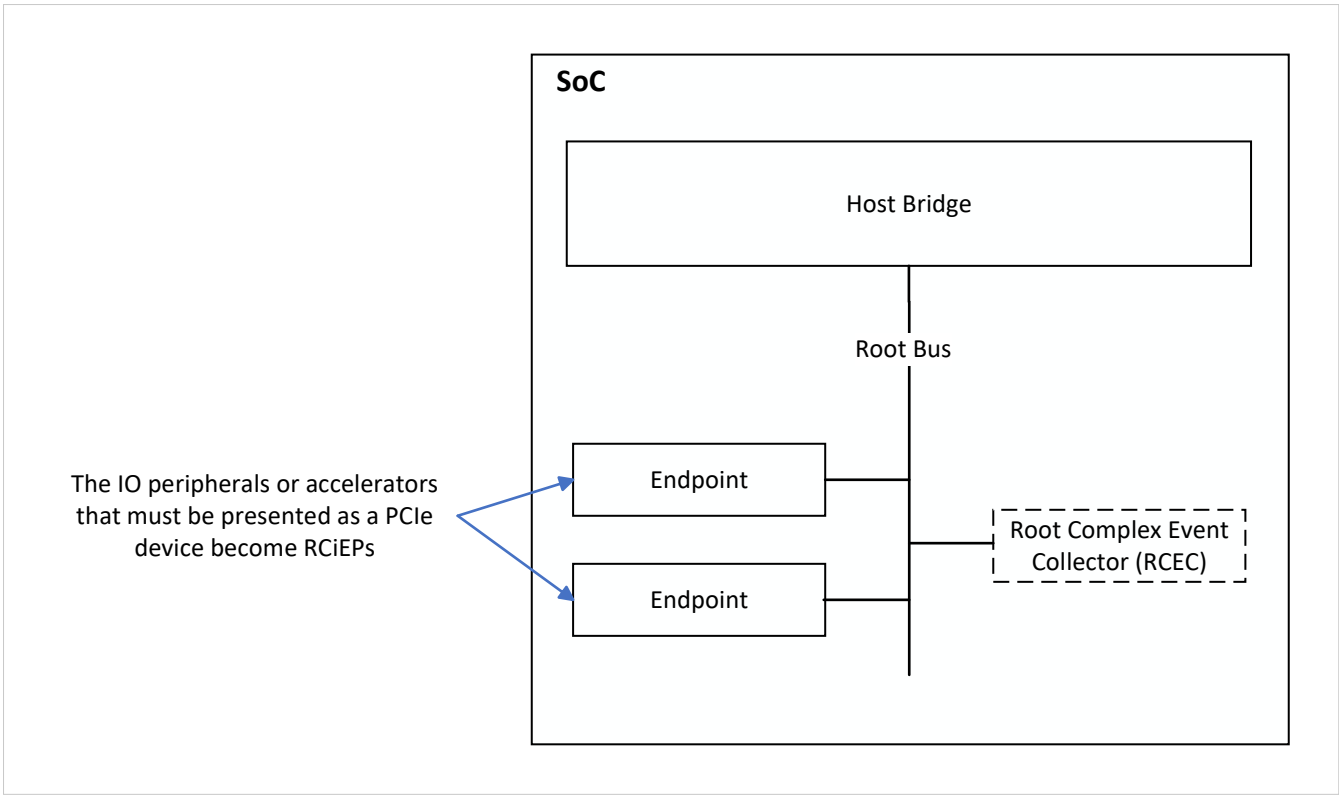

Figure 4: Peripheral presented as an RCiEP behind a host bridge with RCEC on the same Bus

See Section 7.9.10.2 of the PCIe specification [1] for details on RCEC and RCiEP association.

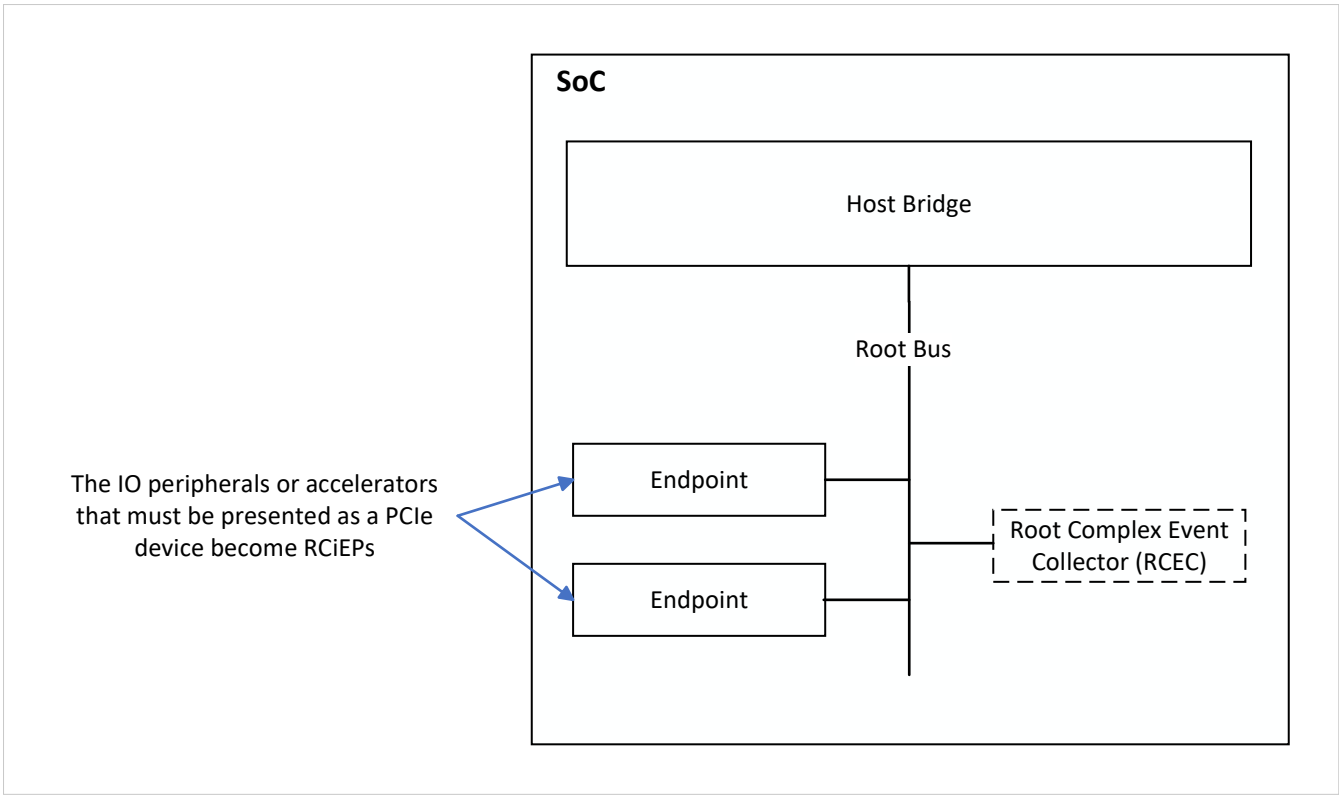

Figure 5: Peripheral presented as an RCiEP behind a host bridge with RCEC on the same Device

See Section 7.9.10.2 of the PCIe specification [1] for details on RCEC and RCiEP association.

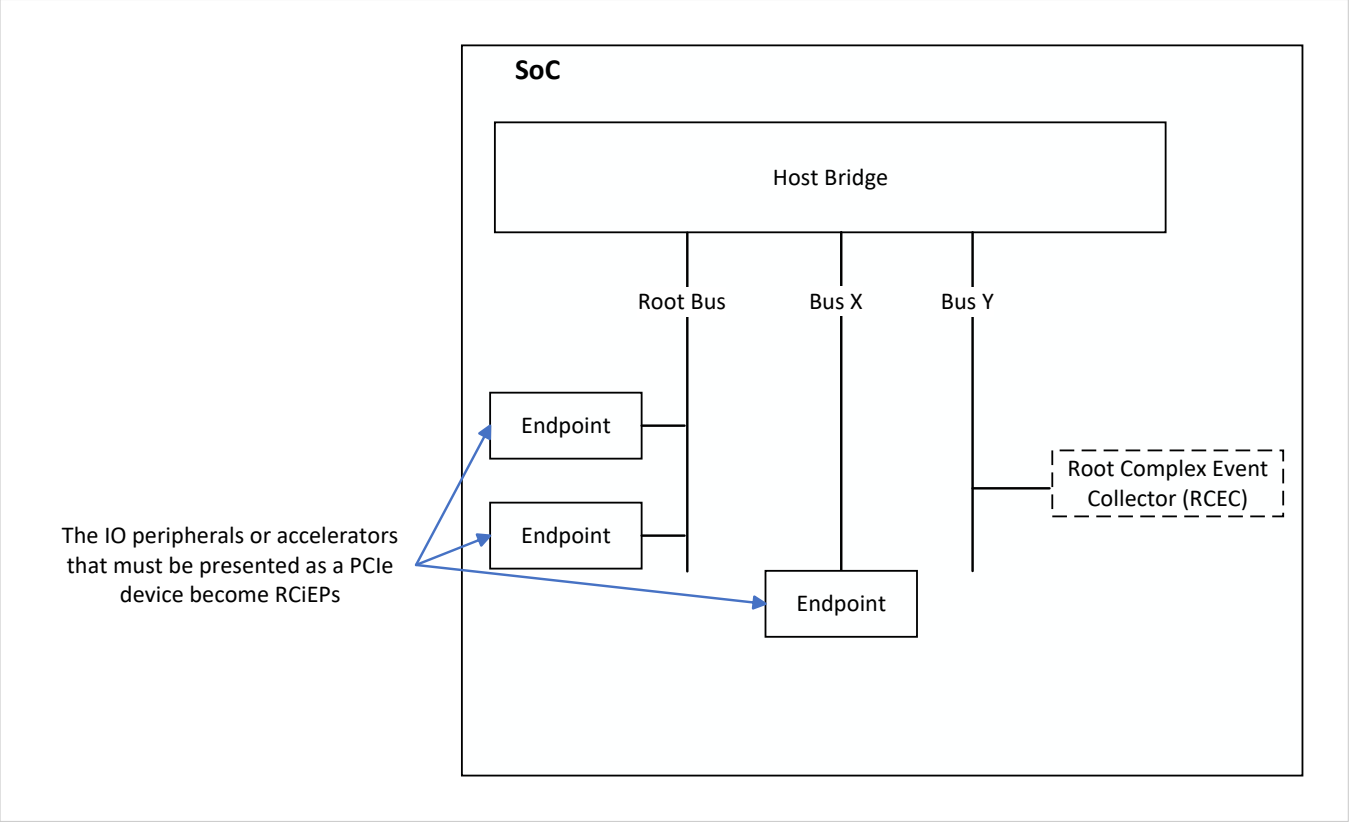

Figure 6: Peripherals presented as RCiEPs behind a host bridge with RCEC on a separate Bus

See Section 7.9.10.3 of the PCIe specification [1] for details on RCEC and RCiEP association.

3.13.2 Option 2: Integrated Endpoint (i-EP)

In this option, the on-chip peripheral appears to software during enumeration as an endpoint that is behind a Root Port, which in turn is behind a host bridge. Figure 7 shows how the peripheral or accelerator is visible to software with this option. In this document, the acronym i-EP describes this combination of Root Port and the integrated Endpoint.

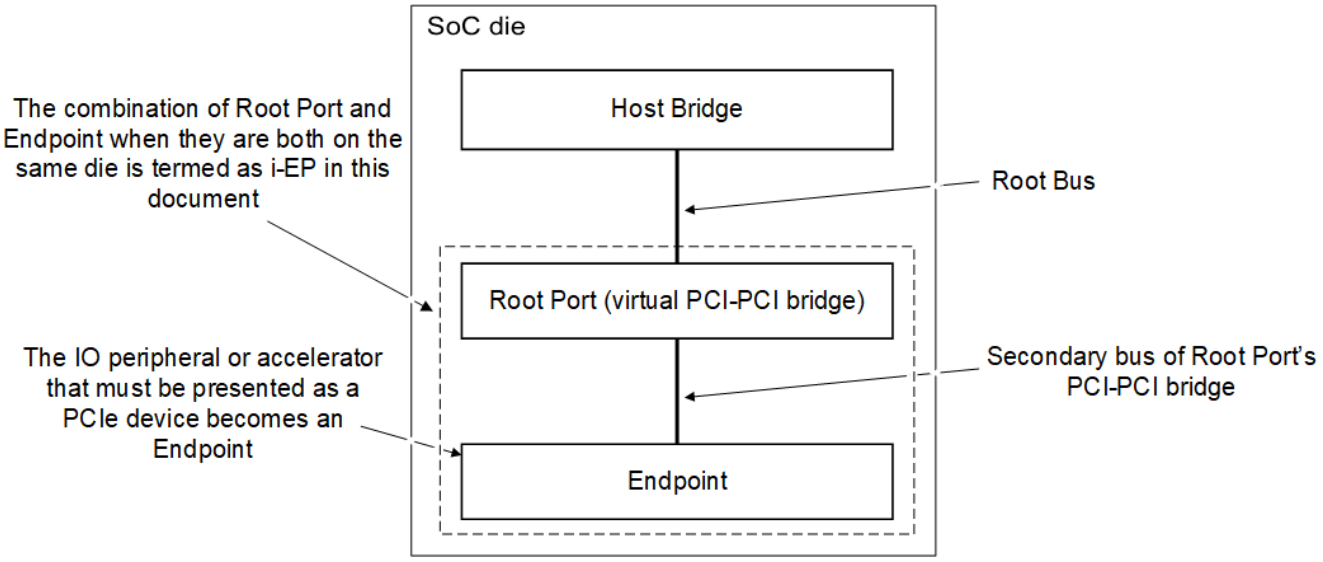

Figure 7: Peripheral presented as an Endpoint behind a Root Port

3.13.3 Comparison between RCiEP and iEP

Table 10 lists the key differences between the RCiEP and i-EP options.

Table 10: Comparison between RCiEP and i-EP options

Functional Area	RCiEP option	i-EP option
Enumeration	OS support is available.	OS support is available.
RAS: AER based error logging and reporting support.	Requires RCEC. OS support is available.	OS support is available.
Power management: PCIe PME event reporting and logging.	Requires RCEC or mechanism to generate a firmware "wake" notification targeting the Device object that generated the PME.	OS support is available.
Link related registers	Not required.	Required by PCIe specification in Root Port and the endpoint.

I_{ZCCKW} AER or PCIe baseline error reporting is necessary only if the RCiEP or the RCEC detects errors that are defined by the PCIe specification. See Section 6.2.7 in the PCIe specification for the list of errors.[1]

At the time of writing of this specification, RCEC OS support for PME is only available on some OS variants.

3.14 Base System Architecture - checklist

This section lists the minimum hardware requirements required to install, boot, and run an operating system on bare-metal or within a virtualization environment.

PE	Rule ID	Specification version first introduced
Operating system		
	B_PE_01	1.0
	B_PE_02	1.0
	B_PE_03	1.0
	B_PE_04	1.0
	B_PE_05	1.0
	B_PE_06	1.0
	B_PE_07	1.0
	B_PE_08	1.0
	B_PE_09	1.0
	B_PE_10	1.0
	B_PE_11	1.0
	B_PE_12	1.0
	B_PE_13	1.0
	B_PE_14	1.0
Hypervisor		
	B_PE_18	1.0
	B_PE_19	1.0
	B_PE_20	1.0
	B_PE_21	1.0
	B_PE_22	1.0
Platform security		
	B_PE_23	1.0
	B_PE_24	1.0

Memory map	Rule ID	Specification version first introduced
Operating system		
	B_MEM_01	1.0
	B_MEM_02	1.0
	B_MEM_03	1.0

Memory map	Rule ID	Specification version first introduced
	B_MEM_05	1.0
	B_MEM_06	1.0
	B_MEM_07	1.0
Platform security		
	B_MEM_08	1.0
	B_MEM_09	1.0

Interrupts	Rule ID	Specification version first introduced
Operating system		
	B_GIC_01	1.0
	B_GIC_02	1.0
	B_GIC_03	1.0
	B_GIC_04	1.0
	B_GIC_05	1.0
	B_PPI_00	1.0

SMMU	Rule ID	Specification version first introduced
Operating system		
	B_SMMU_01	1.0
	B_SMMU_02	1.0
	B_SMMU_06	1.0
	B_SMMU_07	1.0
	B_SMMU_08	1.0
	B_SMMU_12	1.0
Hypervisor		
	B_SMMU_16	1.0
	B_SMMU_17	1.0
	B_SMMU_18	1.0
	B_SMMU_19	1.0
	B_SMMU_21	1.0

Timer subsystem	Rule ID	Specification version first introduced
Operating system		
	B_TIME_01	1.0
	B_TIME_02	1.0
	B_TIME_03	1.0
	B_TIME_04	1.0
	B_TIME_05	1.0
	B_TIME_06	1.0
	B_TIME_07	1.0
	B_TIME_08	1.0
	B_TIME_09	1.0
	B_TIME_10	1.0

Power and wakeup	Rule ID	Specification version first introduced
Operating system		
	B_WAK_01	1.0
	B_WAK_02	1.0
	B_WAK_03	1.0
	B_WAK_04	1.0
	B_WAK_05	1.0
	B_WAK_06	1.0
	B_WAK_07	1.0
	B_WAK_08	1.0
	B_WAK_10	1.0
	B_WAK_11	1.0

Watchdogs	Rule ID	Specification version first introduced
Operating system		
	B_WD_00	1.0

Peripherals	Rule ID	Specification version first introduced
Operating system		
	B_PER_01	1.0
	B_PER_02	1.0
	B_PER_03	1.0
	B_PER_04	1.0
	B_PER_05	1.0
	B_PER_06	1.0
	B_PER_07	1.0
	B_PER_08	1.0
	B_PER_09	1.0
	B_PER_10	1.0
	B_PER_12	1.0
Platform security		
	B_PER_11	1.0

3.14.1 BSA future requirements checklist

In addition to the BSA rules in Section 3.14, the following rules are required.

PE	Rule ID
Operating system	
	B_PE_16
	B_PE_17
	B_PE_25
	XRPZG
	B_SEC_01
	B_SEC_02
	B_SEC_03
	B_SEC_04
	B_SEC_05

SMMU	Rule ID
	B_SMMU_03
	B_SMMU_04

SMMU	Rule ID
	B_SMMU_05
	B_SMMU_09
	B_SMMU_11
	B_SMMU_13
	B_SMMU_14
	B_SMMU_20
	B_SMMU_23
	B_SMMU_24
	B_SMMU_25

Peripherals	Rule ID
Operating system	
	B_REP_1
	B_IEP_1
	BJLPB
	B_PCIe_10
	B_PCIe_11

A Heterogenous systems

This section outlines the requirements for systems composed of heterogeneous PEs, such as big.LITTLE.

A.1 Implementation, identification, and revision differences

Table 22 shows the permitted differences in architected registers between PEs in a single base system. These variations are not expected to be perceived as architectural differences by an operating system or other supervisory software.

The permitted differences column lists the bit fields for a register that can vary from PE to PE. Where a bit field is not listed, the value must be the same across all PEs in the system.

X_{NYDXM} An architectural difference between PEs, in any aspect not supported in Table 22, can lead to unexpected software behavior. Typically, an Operating System will disable a feature if it is not supported by all PEs in the system. Some Operating Systems might fail to online PEs if its architectural features differ, in any aspect not supported in Table 22, from other PEs in the system.

Table 22: Permitted architectural differences

Description	Short-Form	Permitted Differences
AArch64 Memory Features Register	ID_AA64MMFR0_EL1	Bits [3:0] describing the supported physical address range.
Main ID Register	MIDR_EL1	Part number [15:4], Revision [3:0], Variant [23:20].
Virtualization Processor ID Register	VPIDR_EL2	Same fields as MIDR_EL1, writable by hypervisor.
Multiprocessor ID Register	MPIDR_EL1	Bits [39:32] and Bits [24:0]. Affinity fields and MT bit.
Virtualization Multiprocessor ID Register	VMPIDR_EL2	Same fields as MPIDR, writable by hypervisor.
Cache type register	CTR_EL0	Bits [15:14] Level 1 Instruction Cache Policy.
Revision ID Register	REVIDR_EL1	Specific to implementation indicates implementation specific Revisions/ECOs. All bits can vary.
Cache level ID register	CLIDR_EL1	All bits, each PE can have a unique cache hierarchy.
If Armv8.3-CCIDX is not implemented: currentCache Size ID Register	CCSIDR_EL1	Sets [27:13], Data cache associativity [12:3]. Caches on different PEs can be different sizes. Line Size [2:0] must be the same on each PE as software relies on this property.

Description	Short-Form	Permitted Differences
If Armv8.3-CCIDX is implemented: Current Cache Size ID Register	CCSIDR_EL1	The number of Sets [55:32] and the associativity of Caches [23:3] can be different on each PE. Line Size [2:0] must be the same on each PE as software relies on this property.
If Armv8.3-CCIDX is implemented: Current Cache Size ID Register 2	CCSIDR2_EL1	The number of Sets [23:0] can be different on each PE.
Auxiliary Control Register	ACTLR_EL{1,2,3}	Specific to implementation, all bits can vary.
Auxiliary Fault Status Registers	AFSR{0,1}_EL{1,2,3}	Specific to implementation, all bits can vary.

B Generic UART

B.1 About

This specification of the Arm generic UART is designed to offer a basic facility for software bring up. This specification describes the registers and behavior that is required for system software to use the UART to receive and transmit data. This specification does not describe registers that are needed to configure the UART. This is because these registers are considered hardware-specific and will be set up by hardware-specific software. This specification does not cover the physical interface of the UART to the outside world, as this is system specific.

The registers that are described in this specification are a subset of the Arm PL011 r1p5 UART. An instance of the PL011 r1p5 UART will be compliant with this specification.

An implementation of the Generic UART must provide a transmit FIFO and a receive FIFO. Both FIFOs must have the same number of entries, and this number must be at least 32. The generic UART does not support DMA Features, Modem control features, Hardware flow control features, or IrDA SIR features.

The generic UART uses 8-bit words, equivalent to UARTLCR_H.WLEN ==b11.

The basic use model for the FIFO allows software polling to manage flow, but this specification also requires an interrupt from the UART to allow for interrupt-driven use of the UART.

Table 23 identifies the minimum register set used for SW management of the UART.

B.2 Generic UART register frame

The Generic UART is specified as a set of 32-bit registers. However, it is required that implementations support accesses to these registers using read and writes accesses of various sizes. The required access sizes are included in Table 23. The base address of each access, independent of access size, must be the same as the base address of the register being accessed.

If an access size not listed in the table is used, the results are IMPLEMENTATION DEFINED.

The Generic UART is little-endian.

Table 23: Base UART register set

Offset	Name	Description	Permitted access sizes/bits
0x000-0x003	UARTDR Data Register	A 32-bit read/write register. **Bits [7:0]** An 8-bit data register used to access the Tx and Rx FIFOs. **Bits [11:8]** 4 bits of error status used to detect frame errors. Read-only. **Bits [31:12]** - Reserved. (Ref Section 3.3.1 - PL011TRM [11])	Read: 16,32 Write: 8,16,32

Offset	Name	Description	Permitted access sizes/bits
0x004-0x007	UARTRSR/UARTECR Receive status and error clear register	A 32-bit read/write register - a write clears the bits **Bits [3:0]** Four bits of error status, used to detect frame errors as in the UARTDR register, except it allows clearing of these bits. **Bits [31:4]** - Reserved. (Ref Section 3.3.2 - PL011TRM [11])	Read: 16,32 Write: 8,16,32
0x018-0x01c	UARTFR Flag Register	A 32-bit read-only register. **Bits [2:0]** - Reserved **Bits [7:3]** Bits indicate state of UART and FIFOs, with operation as PL011. **Bits [15:8]** - Reserved. (Ref Section 3.3.3 - PL011TRM [11])	Read: 8, 16, 32
0x03c-0x03f	UARTRIS Raw Interrupt Status Register	A 32 bit read-only register. **Bits [3:0]** - Reserved. **Bits [10:4]** Bits used indicate state of Interrupts. **Bits [31:11]** - Reserved. (Ref Section 3.3.11 - PL011TRM [11])	Read: 16, 32
0x040-0x043	UARTMIS Masked Interrupt Status Register	A 32 bit read-only register. **Bits [3:0]** - Reserved. **Bits [10:4]** Bits used indicate state of Interrupts. **Bits [31:11]** - Reserved. (Ref Section 3.3.12 - PL011TRM [11])	Read: 16, 32
0x038-0x03b	UARTIMSC Mask Set/Clear Register	A 32-bit read/write register showing the current mask status. **Bits [3:0]** Write as Ones. **Bits[10:4]** Bits used to set or clear the mask bits assigned to the corresponding interrupts: 1 = mask 0 = unmask. **Bits [31:11]** Reserved, preserve value. Setting the mask bit to 1 enables the interrupt. (Ref Section 3.3.10 - PL011TRM [11])	Read: 16, 32 Write: 16, 32

Offset	Name	Description	Permitted access sizes/bits
0x044-0x047	UARTICR Interrupt Clear Register	A 32-bit write-only register. **Bits[3:0]** Reserved **Bits [10:4]** Bits used to clear the interrupts whose status is indicated in UARTRIS. **Bits[31:11]** - Reserved (Ref Section 3.3.13 - PL011TRM [11])	Write: 16, 32

B.3 Interrupts

The UARTINTR interrupt output must be connected to the GIC.

B.4 Control and setup

Hardware-specific software is required to set up the UART into a state where the above specification can be met and the UART can be used.

This setup is equivalent to the following PL011 state:

```
UARTLCR_H.WLEN == b11 // 8-bit word
UARTLCR_H.FEN == b1 // FIFO enabled
UARTCR.RXE == b1 // receive enabled
UARTCR.TXE == b1 // transmit enabled
UARTCR.UARTEN == b1 // UART enabled
```

B.5 Operation

The base UART operation complies with the subset of features implemented of the PL011 Primecell UART, the operation of which can be found in sections 2.4.1, 2.4.2, 2.4.3, and 2.4.5 of the ARM® PrimeCell® UART (PL011) Technical Reference Manual [11]. Operations of the IrDA SIR, modem, hardware flow control, and DMA are not supported.

C Generic Watchdog

C.1 About

The Generic Watchdog aids the detection of errant system behavior. If the Generic Watchdog is not refreshed periodically, it will raise a signal, which is typically wired to an interrupt. If this watchdog remains un-refreshed, it will raise a second signal which can be used to interrupt higher-privileged software or cause a PE reset.

The Generic Watchdog has two register frames, one that contains the refresh register and one for control of the watchdog.

C.2 Watchdog operation

The basic function of the Generic Watchdog is to count for a fixed period of time, during which it expects to be refreshed by the system indicating normal operation. If a refresh occurs within the watch period, the period is refreshed to the start. If the refresh does not occur then the watch period expires, and a signal is raised and a second watch period is begun.

The initial signal is typically wired to an interrupt and alerts the system. The system can attempt to take corrective action that includes refreshing the watchdog within the second watch period. If the refresh is successful, the system returns to the previous normal operation. If it fails, then the second watch period expires and a second signal is generated. The signal is fed to a higher agent as an interrupt or reset for it to take executive action.

The Watchdog uses the Generic Timer system counter as the timebase against which the decision to trigger an interrupt is made.

Note

The Arm ARM states that the system counter measures the passing of real-time. This counter is sometimes called the physical counter.

The Watchdog is based on a 64-bit compare value and comparator. When the generic timer system count value is greater than the compare value, a timeout refresh is triggered.

The compare value can either be loaded directly or indirectly on an explicit refresh or timeout refresh.

When the watchdog is refreshed explicitly, the compare value is loaded with the sum of the zero-extended watchdog offset register and the current generic timer system count value.

Revision 1 of watchdog increases the length the watchdog offset register to 48 bit. The operation of the watchdog remains the same. Software can determine which version of the watchdog is implemented through the watchdog interface identification register (W_IID).

When the watchdog is refreshed through a timeout, the compare value is loaded with the sum of the zero-extended watchdog offset register and the current generic timer system count value. See below for exceptions.

An explicit watchdog refresh occurs when one of a number of different events occur:

- The Watchdog Refresh Register is written.
- The Watchdog Offset Register is written.
- The Watchdog Control and Status register is written.

In the case of an explicit refresh, the Watchdog Signals are cleared. A timeout refresh does not clear the Watchdog Signals.

The watchdog has the following output signals:

- Watchdog Signal 0 (WS0).
- Watchdog Signal 1 (WS1).

If WS0 is asserted and a timeout refresh occurs, then the compare value must retain its current value. This means that the compare value records the time that WS1 is asserted.

If both watchdog signals are deasserted and a timeout refresh occurs, WS0 is asserted.

If WS0 is asserted, the architectural state of the watchdog is not reset.

If WS0 is asserted and a timeout refresh occurs, WS1 is asserted.

WS0 and WS1 remain asserted until an explicit refresh or watchdog Cold reset occurs.

System firmware must record the WS1 assert event. The mechanism to report WS1 assertion event to the Operating System is IMPLEMENTATION DEFINED.

WS0 and WS1 are deasserted when the watchdog is disabled.

The status of WS0 and WS1 can be read in the Watchdog Control and Status Register.

Watchdog Cold reset must only occur as part of the watchdog powering-up sequence. On a Cold reset, certain Generic Watchdog register values are reset to a known state.

The following pseudo-code describes the Generic Watchdog behavior.

```
TimeoutRefresh = (SystemCounter[63:0] > CompareValue[63:0])
If WatchdogColdReset
        WatchdogEnable = DISABLED
Endif
If LoadNewCompareValue
        CompareValue = new_value
ElseIf ExplicitRefresh == TRUE or (TimeoutRefresh == TRUE and WS0 == FALSE)
        CompareValue = SystemCounter[63:0] + ZeroExtend(WatchdogOffsetValue
            ↪[47:0])
Endif
If WatchdogEnable == DISABLED
        WS0 = FALSE
        WS1 = FALSE
ElseIf ExplicitRefresh == TRUE
        WS0 = FALSE
        WS1 = FALSE
ElseIf TimeoutRefresh == TRUE
        If WS0 == FALSE
                WS0 = TRUE
        Else
                WS1 = TRUE
        Endif
Endif
```

The Generic Watchdog must be disabled when the System Counter is being updated, or the results are UNPREDICTABLE.

C.3 Register summary

This section gives a summary of the registers, relative to the base address of the relevant frames.

All registers are 32 bits in size and should be accessed using 32-bit reads and writes. If an access size other than 32 bits is used, the results are IMPLEMENTATION DEFINED. There are two register frames, one for a refresh register, and the other containing the status and setup registers.

The Generic Watchdog is little-endian. Table 24 shows the refresh frame.

Table 24: Refresh frame

Offset	Name	Description
0x000-0x003	WRR	Watchdog refresh register. A write to this location causes the watchdog to refresh and start a new watch period. A read has no effect and returns 0.
0x004-0xFCB	-	Reserved.
0xFCC-0xFCF	W_IID	See Section C.4.2
0xFD0-0xFFF	-	IMPLEMENTATION DEFINED.

Table 25 shows the watchdog control frame.

Table 25: Watchdog control frame

Offset	Name	Description
0x000-0x003	WCS	Watchdog control and status register. A read/write register containing a watchdog enable bit, and bits indicating the current status of the watchdog signals.
0x004-0x007	-	Reserved.
0x008-0x00B	WOR [31:0]	Watchdog offset register. A read/write register containing the lower 32 bits of the unsigned watchdog countdown timer value.
0x00C-0x00F	WOR [63:32]	Watchdog offset register: **Bits [31:16]** Reserved. Read all zeros, write has no effect. **Bits [15:0]** Read/write upper 16 bits of the watchdog countdown timer.
0x010-0x013	WCV [31:0]	Watchdog compare value. Read/write registers
0x014-0x017	WCV [63:32]	containing the current value in the watchdog compare register.
0x018-0xFCB	-	Reserved.
0xFCC-0xFCF	W_IIDR	See Section C.4.2
0xFD0-0xFFF	-	IMPLEMENTATION DEFINED

Note

In the previous revision of the Generic Watchdog, revision 0, the Watchdog offset register was 32 bit, and offset 0x00C-0x00F was Reserved.

C.4 Register descriptions

C.4.1 Watchdog Control and Status Register

The format of the Watchdog Control and Status Register is:

Bits [31:3]

Reserved. Read all zeros, write has no effect.

Bits [2:1] - Watchdog Signal Status bits

A read of these bits indicates the current state of the watchdog signals; bit [2] reflects the status of WS1 and bit [1] reflects the status of WS0.

A write to these bits has no effect.

Bit [0] - Watchdog Enable bit

A write of 1 to this bit enables the Watchdog, a 0 disables the Watchdog.

A read of these bits indicates the current state of the Watchdog enable.

The watchdog enable bit resets to 0 on watchdog Cold reset.

C.4.2 Watchdog Interface Identification Register

W_IIDR is a 32-bit read-only register. The format of the register is:

ProductID, bits [31:20]

An IMPLEMENTATION DEFINED product identifier.

Architecture version, bits [19:16]

Revision field for the Generic Watchdog architecture. The value of this field depends on the Generic Watchdog architecture version:

- 0x1 for Generic Watchdog v1.

Revision, bits [15:12]

An IMPLEMENTATION DEFINED revision number for the component.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the Generic Watchdog:

Bits [11:8] The JEP106 continuation code of the implementer.

Bit [7] Always 0.

Bits [6:0] The JEP106 identity code of the implementer.

D SMMUv3 integration

This appendix details rules about the integration of a SMMUv3 SMMU into system.

I_{QQQBL} The system is permitted to include any number of SMMUs.

R_{SMMU_01} All SMMU translation table walks and all SMMU accesses to SMMU memory structures and queues are I/O coherent (SMMU_IDR0.COHACC == 1).

I_{KKBHX} SMMUv3 supports two distinct page table fault models: stall on fault, and terminate on fault. Care must be taken when designing a system to use the stall on fault model.

R_{SMMU_02} The system must be constructed such that for any device for which stalling is supported the act of the SMMU stalling on a fault from a device must not

- Stall the progress of any other device or PE that is not under the control of the same operating system as the stalling device.
- Stall the progress of any other device or PE where this might lead to system deadlock or instability.

I_{KCZXQ} The SMMUv3 spec requires that PCIe root complex must not use the stall model due to potential deadlock.

I_{BQCTQ} See Section E.12 for requirements on PCIe PASID support.

I_{GGPMS} See Section H for requirements on how DeviceID and StreamID should be assigned and how ITS groups should be used.

E PCI Express integration

E.1 Configuration space

I_{GCYHY} This section contains rules that apply to all PCI Express functions, unless stated otherwise.

I_{VGLFZ} The following rules do not apply to RCiEP or RCEC: PCI_IN_04, PCI_IN_05, PCI_IN_06, PCI_IN_08, PCI_IN_09, PCI_IN_10, PCI_IN_11, PCI_IN_12, PCI_IN_13, PCI_IN_17, PCI_IN_18, and PCI_IN_19.

$R_{PCI\_IN\_01}$ Systems must map memory space to PCI Express Configuration Space, using the PCI Express Enhanced Configuration Access Mechanism (ECAM). For more information about ECAM, see *PCI Express Base Specification Revision 3.1*[1].

I_{XWXRJ} The ECAM maps Configuration Space to a contiguous region of memory address space, using bit slices of the memory address to map on to the PCI Express Configuration Space address fields. This mapping is shown in Table 26.

Table 26: Enhanced Configuration Address Mapping

Memory Address bits	PCI Express Configuration Space address field
(20 + n - 1):20	Bus Number 1 <= n <= 8.
19:15	Device Number.
14:12	Function Number.
11:8	Extended Register Number.
7:2	Register Number.
1:0	Byte.

To ensure that the enumeration process works correctly, a combination of PCIe host bridge (PHB) and PCIe Root Port must meet the following requirements:

$R_{PCI\_IN\_02}$ Once boot firmware hands control over to the operating system, application processor accesses to ECAM regions must work with no additional programming. The accesses must not require any OS visible programming.

$R_{PCI\_IN\_03}$ The Configuration Space of all the devices, Root Ports, Root Complex Integrated Endpoints, and switches behind a PHB must be in a single ECAM region.

$R_{PCI\_IN\_04}$ The configuration space of all the Endpoints and Switches in a Root port's hierarchy must be in the same ECAM space as the root port.

$R_{PCI\_IN\_05}$ Root Port must appear as a PCI-PCI bridge to software (See Section 7.1 [1]). This implies that Root Port:

- must have all registers that are part of the Type 1 header, as specified in Section 7.5.1.3 of the PCIe specification [1].
- must have all the capabilities required by PCIe specification for a Root Port. This includes the PCI express capability structure (See Section 7.5.3 [1]).
- registers must follow the access attributes (RW/RO and so on) specified in the PCIe specification [1].

$R_{PCI\_IN\_06}$ PHB, in conjunction with Root Port, must recognize transactions that are coming in from application PEs as PCIe configuration transactions if the transaction address is within the ECAM range mapped to the Root Port, or the hierarchy that originates at that Root Port. This must be done by mapping the address of the incoming

memory transaction to the PCIe Configuration address space, as described in Table 26 (See Section 7.2.2 [1]).

$R_{PCI\_IN\_07}$ PHB in conjunction with Root Port must return all 1s as read response data for Configuration read requests to non-existent functions and devices on the root bus, that is the primary bus of the Root Port. No error must be reported to software by the Root Port unless explicitly enabled to do so (See Section 2.3.2 [1]).

$R_{PCI\_IN\_08}$ PHB, in conjunction with Root Port, must return all 1s as read response data for Configuration read requests that get an unsupported request response from downstream Endpoints or switches. No error must be reported to software by the Root Port unless explicitly enabled to do so (See Section 2.3.2 [1]).

$R_{PCI\_IN\_09}$ PHB in conjunction with Root Port must return all 1s as read response data for Configuration read requests that arrive at the Root Port targeting downstream functions when the Root port link is in DL_Down status (See Section 2.9.1 [1]). Note that this includes the case when the link is in L3 and the downstream device is in D3cold. No error must be reported to software by the Root Port unless explicitly enabled to do so (See Section 2.3.2 [1]).

I_{WQBFR} Accesses to the Root Port's own ECAM space while the link is in DL_down status must function correctly.

$R_{PCI\_IN\_10}$ PHB in conjunction with Root Port must send out Configuration transactions that are intended for the subordinate bus range of the Root Port as Type 1 Configuration transactions to downstream devices and switches. Subordinate bus range is between secondary bus number, exclusive, and the subordinate bus number, inclusive. (See Section 3.2.2.3.1 [12]).

$R_{PCI\_IN\_11}$ PHB in conjunction with Root Port must send out Configuration transactions that are intended for the secondary bus of the Root Port as Type 0 Configuration transactions to devices and switches downstream (See Section 3.2.2.3.1 [12]).

$R_{PCI\_IN\_12}$ PHB in conjunction with Root Port must Recognize and consume Configuration transactions intended for the Root Port Configuration space and, read or write the appropriate Root Port Configuration register (See Section 3.2.2.3.1 [12]).

$R_{PCI\_IN\_13}$ PHB in conjunction with Root Port must recognize transactions received on the primary side of the Root Port PCI-PCI bridge, targeting 32-bit memory or 64-bit memory spaces of devices and switches that are on the secondary side of the bridge:

- Where the address falls within the 32-bit memory or 64-bit memory windows specified in the type 1 header registers, the transactions must be forwarded to the secondary side (See Section 6.26.2 [1], Chapter 4 [13]).
- Where the address of the request does not fall within the 32-bit memory or 64-bit memory windows that are specified in the type 1 header registers, the Root Port must respond with unsupported request (See Section 6.26.2 [1], Chapter 4 [13]).

$R_{PCI\_IN\_14}$ PHB in conjunction with Root Port must return all 1s data to the requester PE as the response for a configuration read if all of the following are true:

- CRS software visibility is disabled or not present.
- CRS response was received for the request the first time it was issued by the Root Complex. The Root Complex then tried to make the request return valid data by re-issuing the request an implementation defined number of times, but CRS was the response received for all such re-issues.

$R_{PCI\_IN\_15}$ PHB in conjunction with Root Port must return all 1s data to the requester PE as the response for a configuration read if all of the following are true:

- CRS software visibility is enabled.
- The Configuration read is not targeting the Vendor ID register.
- CRS response was received for the request the first time it was issued by the Root Complex. The Root Complex then tried to make the request return valid data by re-issuing the request an implementation defined number of times, but CRS was the response received for all such re-issues.

$R_{PCI\_IN\_16}$ PHB in conjunction with the Root Port must return all 1s data to the requester PE as the response for a configuration read if all of the following are true:

- Target bus number of the request is not within the secondary bus to subordinate bus range of any of the Root Ports.
- Target Bus, Device and Function (BDF) of the request does not match BDF of any on-chip functions.
- Target BDF of the request does not match the BDF of any of the Root Ports

$R_{PCI\_IN\_17}$ The Root port must comply with the following (as per section 6.13 of [1]).

If the target bus number of a configuration request is equal to the Root Port's secondary bus number, then:

- If ARI forwarding is disabled and the target device number of the request==0, then the access is to the secondary bus of the Root Port, and the access is forwarded downstream as type 0 configuration request.
- If ARI forwarding is disabled and target device number of the request > 0, then the access is terminated and all 1s data is returned to the requester PE.
- If ARI forwarding is enabled, then the access is to the secondary bus of the Root Port, and the access is forwarded downstream as type 0 configuration request regardless of the target device number of the request.

$R_{PCI\_IN\_18}$ The Root Port must comply with the byte enable rules that are specified in the PCIe specification (See Section 2.2.5 [1]) and must support 1 byte, 2 byte and 4 byte Configuration read and write requests.

$R_{PCI\_IN\_19}$ All registers present in the Root Port PCIe configuration space must follow the rules as defined in section 7.2 of the PCIe specification [1].

$R_{PCI\_IN\_20}$ Any vendor specific data in the PCIe configuration space must be presented by one of the following capabilities, as defined in the PCIe specification [1] :

- Vendor Specific Capability
- Vendor Specific Extended Capability (VSEC)
- Designated Vendor Specific Extended Capability (DVSEC)

I_{HKXFD} It is recommended that if the Root Port is the requester of a PCIe transaction, then the requester ID of the transaction is formed using the bus, device, and function numbers of the Root Port.

I_{FWKPJ} It is recommended that completer ID for PCIe completions generated by the Root Port, for example, Completion for a memory read request made earlier from an Endpoint to memory, is formed using the bus, device ,and function numbers of the Root Port.

I_{WJWFF} It is recommended that the Root Port supports Configuration Request Retry Status (CRS) software visibility feature (Section 2.3.2 [1]). If CRS software visibility is supported, then it must be according to the CRS completion handling rules that are specified in the PCIe specification (Section 2.3.2 [1]).

I_{KRTMM} The system can implement multiple ECAM regions. The base address of each ECAM region within the system memory map is IMPLEMENTATION DEFINED and is expected to be discoverable from system firmware data.

I_{PHYPD} Whether a system supports non-PE agents accessing ECAM regions is system specific.

I_{YLNHP} Alternative Routing-ID Interpretation (ARI) is permitted. For buses with an ARI device the ECAM field [19:12] is interpreted as the 8-bit function number.

E.2 PCI Express memory space

I_{DCBYL} It is system specific whether a system supports mapping PCI Express memory space as cacheable.

$R_{PCI\_MM\_01}$ All systems must support mapping PCI Express memory space as device memory.

$R_{PCI\_MM\_02}$ All systems must support mapping PCI Express memory space as non-cacheable memory.

$R_{PCI\_MM\_03}$ When PCI Express memory space is mapped as normal memory, the system must support unaligned accesses to that region.

I_{LZXGS} PCI Type 1 headers, used in PCI-to-PCI bridges, and therefore in root ports and switches, have to be programmed with the address space resources claimed by the given bridge.

$R_{PCI\_MM\_04}$ Systems compliant to this specification must support 32-bit programming of 32-bit BARs on such endpoints. This can be achieved in two ways:

Method 1: PE physical address space can be reserved below 4GB, whilst maintaining a one-to-one mapping between PE physical address space and 32-bit BAR memory address space.

Method 2: It is also possible to use a fixed offset translation scheme that creates a fixed offset indirection between PE physical address space, and PCI memory. This allows a window in PE physical address space that is above 4GB to be mirrored in PCI memory space below 4GB. This requires support in the PHB. Furthermore, firmware must program the PHB with the fixed offset, and to supply this information to the OS [4].

Caution should be used with fixed offset translation schemes. Some devices might have software compatibility issues with accessing 64-bit addresses.

Method 1 is recommended, because this method eases peer-to-peer support in 32-bit BAR memory.

I_{GZDQW} It is recommended that systems and devices always use 64-bit BARs where possible to reduce contention for the limited 32-bit address space.

X_{NXDZE} The 32-bit address space should be reserved for devices that require 32-bit BARs. Examples include boot devices and graphics devices.

E.3 PCI Express device view of memory

X_{LBHKY} Transactions originating from a PCI Express device (*PCI Express DMA transactions*) might be presented to an SMMU for address translation and permission policing.

$R_{PCI\_MM\_05}$ Except for address transformations allowed by the SMMU architecture specification, PCI Express devices must have the same view of system physical address space as the PEs.

I_{XMPTG} Rule PCI_MM_05 does not apply to PCI Express memory space in systems that do not support peer-to-peer transactions. However, if peer-to-peer is supported then PCI_MM_05 also applies to transactions targeting PCI Express memory space.

$R_{PCI\_MM\_06}$ In a system where PCI Express DMA transactions are not presented to an SMMU, the address field of the PCI Express transaction must not be modified except for the truncation of leading zeros to conform with the physical address width of the system.

$R_{PCI\_MM\_07}$ In a system where PCI Express DMA transactions are presented to an SMMU, the address field of the PCI Express transaction must not be modified except as explicitly allowed by the SMMU architecture specification.

I_{MLJFZ} For accesses from a PE to the PCI memory space in a PCIe endpoint, offset based translation is permissible, as described in Method 2 of Section E.2.

I_{FGNBK} In order to be compliant with PCI_MM_05, all peer-to-peer transactions targeting a PCIe endpoint with offset based translation enabled must be routable through the host, for example by supporting ACS.

E.4 Message Signaled Interrupts

$R_{PCI\_MSI\_1}$ Support for Message Signaled Interrupts (MSI/MSI-X) is required for PCI Express devices which generate interrupts.

I_{XNXYS} MSI and MSI-X are edge-triggered interrupts that are delivered as a memory write transaction.

I_{PSWKQ} Arm introduced standard support for MSI(-X) in the GICv2m architecture, this support is extended in GICv3.

$R_{PCI\_MSI\_2}$ The intended use model is that each unique MSI(-X) must trigger an interrupt with a unique ID and the MSI(-X) must target GIC registers requiring no hardware specific software to service the interrupt.

E.5 GICv3 support for MSI(-X)

I_{LBHGV} GICv3 adds a new class of interrupt, LPI, to address MSI(-X). LPI can be targeted to a single PE.

In GICv3, SPI can be targeted at a single PE or can be “1 of N”, where the interrupt will be delivered to any one of the PEs in the system currently powered up.

In GICv3, SPI are still limited in scale, but an implementation can support thousands of LPIs.

In GICv3, MSI(-X) can target SPI or LPI.

A single GICD_SETSPI_NSR register is supported for MSI targeting SPI. This is a compatibility break with GICv2m, and does not support I/O virtualization.

GICv3 provides the GITS_TRANSLATER register for MSI targeting LPI. This register uses a Device_ID to uniquely identify the originating device to fully support I/O virtualization, and is backed by memory-based tables to support flexible re-targeting of interrupts.

E.6 Legacy interrupts

$R_{PCI\_LI\_01}$ PCI Express legacy Interrupt messages must be converted to an SPI.

$R_{PCI\_LI\_02}$ A unique SPI ID must be allocated to each of the legacy interrupt lines of a PHB. It is permissible to share SPI IDs across PCI host bridges.

$R_{PCI\_LI\_03}$ Each legacy interrupt SPI must be programmed as level-sensitive in the appropriate GIC_ICFGR.

I_{JKHDR} The exact SPI IDs that are allocated are IMPLEMENTATION DEFINED.

$R_{PCI\_LI\_04}$ IMPLEMENTATION DEFINED registers must not be used to deliver these messages, only registers defined in the PCI Express specification and the Arm GIC specification.

E.7 System MMU and Device Assignment

$R_{PCI\_SM\_01}$ Hardware support for function or virtual function assignment to a VM or user space driver is optional, but if required must use a System MMU compliant with the Arm System MMU specification [15].

I_{XSYKQ} Individual Base System Architectures require certain versions of the SMMU to be used at particular levels of the specification.

I_{FYFCM} Each function or virtual function that can be assigned to a VM or to a user space driver is associated with a SMMU context. The programming of this association is IMPLEMENTATION DEFINED and is expected to be described by system firmware data.

I_{WDVLJ} SMMU does not support PCI Express ATS until SMMUv3, and as such ATS support is system-specific in systems that do not have a SMMUv3 or later.

$R_{PCI\_SM\_02}$ Functions intended for VM assignment, or assignment to a user space driver must implement function level reset.

E.8 I/O Coherency

I_{YQJXR} I/O interfaces such as PCI Express do not support the coherency operations necessary to maintain a fully coherent cache in the I/O device. I/O coherency refers to the mechanism that an I/O subsystem uses to enable I/O components to coherently access cacheable memory. In the Arm architecture, memory types and

attributes are applied to each transaction and determine the coherence properties of the transaction. See “Memory types and attributes” in [3]. Some memory types and attributes also have cache allocation hints associated with them. See “Cacheability, cache allocation hints, and cache transient hints” in [15].

$R_{PCI\_IC\_11}$ If an I/O subsystem supports I/O coherency, it must be in the same Inner Shareability domain as the PEs.

X_{LPKNT} The system might need to snoop PE caches in order to maintain coherency for I/O accesses. For accesses to Normal memory that use the Outer Shareable attribute, shareability applies across both the Inner Shareability and Outer Shareability domains.

I_{WTLPK} All observers in an Inner Shareability domain are always members of the same Outer Shareability domain. See “Shareable Normal memory” in [15].

I_{MZFWQ} Prior revisions of the BSA required I/O subsystems that support PCI Express components to also support I/O Coherency. This requirement was relaxed for the BSA and might be strengthened in domain-specific specifications like the Server BSA [5].

I_{JVNRX} If an I/O subsystem does not support I/O coherency, it can be in a different Inner or Outer Shareability domain as the PEs and the system is not required to snoop PE caches for I/O accesses.

E.8.1 Memory type and attribute assignment

I_{JKQVB} Memory type and attribute assignment for PCI Express DMA transactions is conceptually described in several sequential steps. [Figure 8] shows this flow. This description does not mandate the order in which an implementation chooses to apply the memory type and attribute assignment steps as long as the observed result is equivalent with the sequential description for the supported system configurations.

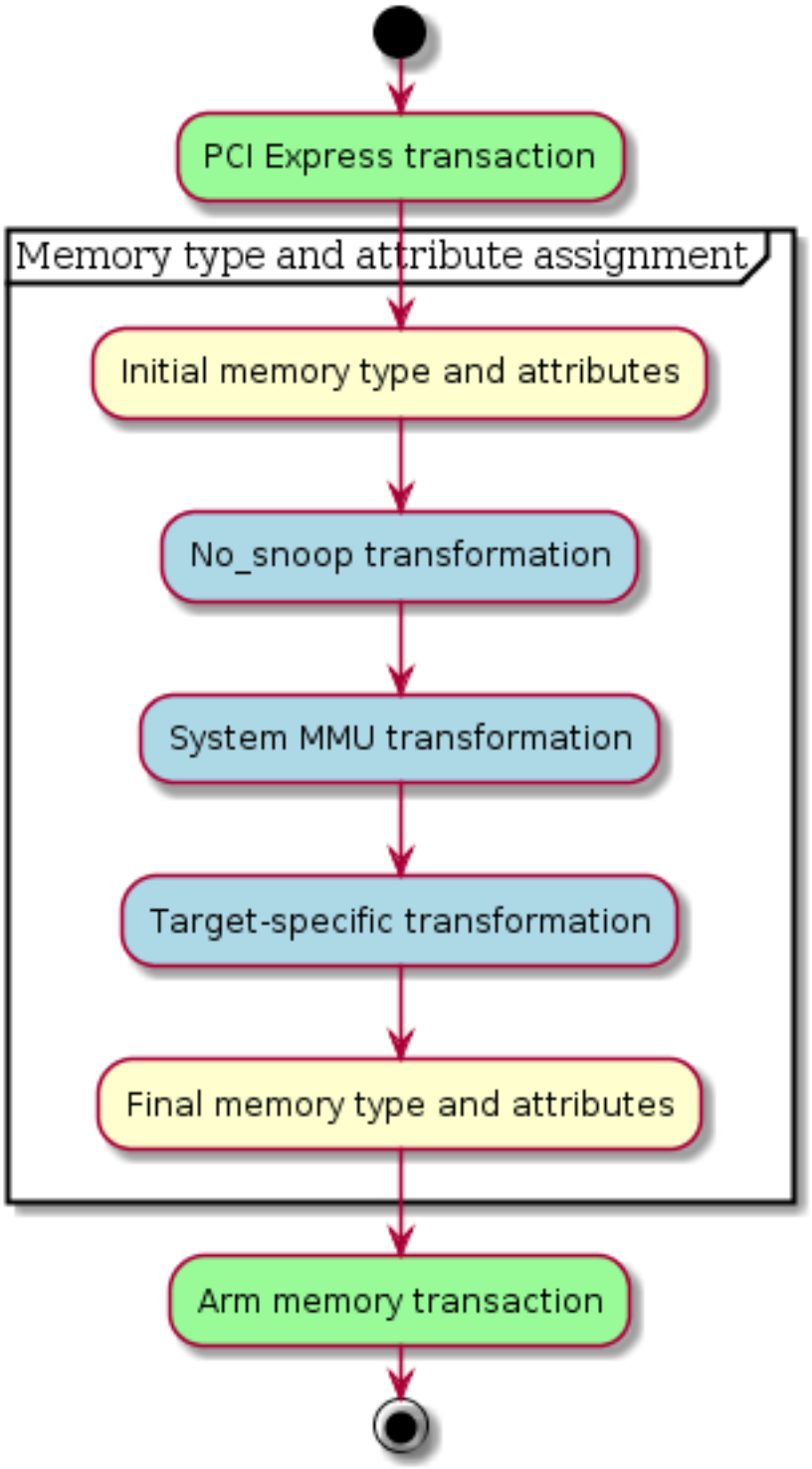

Figure 8: Memory type and attribute assignment for PCI Express DMA transactions

$R_{PCI\_IC\_12}$ Until final memory type and attributes are known, the transaction must follow PCI Express transaction rules and must not be collapsed/combined/merged or otherwise altered based on an intermediate memory type or attribute.

$R_{PCI\_IC\_13}$

PCI Express Translated transactions can contain ATS Memory Attributes (AMA). An implementation must provide a mode of operation where AMA values, if presented in the PCI Express transaction, are ignored and do not alter the memory type, memory attributes, or cache allocation hints of a transaction. Any other interpretation of AMA is IMPLEMENTATION DEFINED. Arm might provide guidance in the future with respect to AMA.

E.8.1.1 Initial memory type and attributes

$R_{PCI\_IC\_14}$ The initial memory attributes associated with a PCI Express transaction must be Normal Inner and Outer Write-back Cacheable Outer Shareable.

I_{KNMPH} PCI Express transactions contain TLP Processing Hints (TPH). TPH is comprised of two fields: PH (Processing Hint), and ST (Steering Tag), as well as an indication that the hints are valid and whether Steering Tag is 8-bits or 16-bits. The interpretation of these fields is IMPLEMENTATION DEFINED. It is recommended that when the hints are valid, regardless of the value in the PH and ST fields, the initial cache allocation hints are changed to Read-Allocate, Write-Allocate. Refer to RB_PCIe_10 and RB_PCIe_11 for requirements when ST is implemented.

E.8.1.2 No_snoop transformation

$R_{PCI\_IC\_15}$ PCI Express transactions contain a No_snoop attribute. If the No_snoop attribute is set in the PCI Express transaction, the memory attributes associated with the transaction must be replaced with Normal Inner and Outer Non-cacheable.

I_{VFKCC} Because all data access to Non-cacheable locations are data coherent to all observers, Non-cacheable locations are always treated as Outer Shareable. See “Shareable Normal Memory” in [15].

I_{BMYJT} A transformation based on the No_snoop attribute is specified in contemporary versions of the SMMUv3 architecture [15]. The System MMU transformation can cause the No_snoop Transformation specified in this specification to not be observable when a System MMU is present. Future revisions of the System MMU architecture might introduce a modal behavior that affects the System MMU interpretation of the No_snoop attribute and the observability of the No_snoop Transformation specified in this specification.

E.8.1.3 System MMU transformation

$R_{PCI\_IC\_16}$ If there is a System MMU in the path of the transaction, the memory attributes associated with the transaction must be provided as input to the System MMU. The memory attributes of the transaction must be replaced with the output of the System MMU. See [15] for System MMU operation.

E.8.1.4 Target-specific transformation

$R_{PCI\_IC\_17}$ For some address targets and platform configurations, an IMPLEMENTATION DEFINED mechanism might need to be provided to override or ignore the memory type and attributes associated with the transaction in order to guarantee the required properties of the target:

- Message Signaled Interrupts (MSIs) must be recorded according to PCI Express transaction rules. They must remain ordered behind prior writes, must not merge, and must not allocate in caches.
- If PCI Express peer-to-peer functionality is supported, PCI Express transactions that target PCI Express memory-mapped I/O space must not violate PCI Express transaction rules or the properties required by the target PCI Express BAR region.
- Other special address regions can be defined by the platform.

X_{ZZHBB} Some address targets, such as those associated with peripherals or memory mapped I/O ranges might require special transaction properties. Hardware and/or firmware mechanisms might need to be provided to ensure correct functionality for these targets across the supported platform configurations. These mechanisms are particularly needed if the System MMU is not present, bypassed, or otherwise not involved in assigning memory types and attributes (as is the case for ATS Translated transactions).

E.8.1.5 Final memory attributes and coherence properties of Arm memory transactions

I_{NDTLS} If the final attributes associated with the transaction match the attributes used by the PE to access the memory, PCI Express transactions are coherent with accesses from the PEs. See “Memory types and attributes” in [15].

I_{MQZJZ} If the final attributes associated with the transaction are mismatched with the attributes used by the PE to access the memory, PCI Express transactions are not guaranteed to be coherent with accesses from the PEs and software might need to use Cache Maintenance Operations (CMOs) to maintain coherency. See “Mismatched memory attributes” in [15].

I_{NJYQT} Some system implementations can guarantee coherency even for mismatched memory attributes.

$R_{PCI\_IC\_18}$ The shareability domain of an I/O subsystem that supports PCI Express must not contain any caches outside the shareability domain of the PEs.

X_{ZXNGH} Software-managed coherency requires that there are no caches that are accessible to PCI Express transactions but inaccessible to the PEs. Cache Maintenance Operations (CMOs) issued by a PE operate only within the shareability domain of the PEs.

E.9 Legacy I/O

I_{VSKGJ} The specification does not require support for legacy I/O transactions nor support for PCI Express I/O space BARs (see “I/O Transactions” and “Base Address Registers” in [1]). It is recommended to support legacy I/O transactions in order to maintain compatibility with the widest number of devices.

$R_{PCI\_IO\_01}$ If an implementation supports legacy I/O, it is supported using a one to one mapping between legacy I/O space and a window in the host physical address space. However, such schemes must not require a kernel driver to be set up; any necessary initialization must be performed before OS boot.

E.10 Integrated end points

X_{DQPYC} Feedback from OS vendors has indicated that they have seen many ‘almost PCI Express’ integrated endpoints. This leads to a bad experience and either no OS support for the endpoint or painful bespoke support.

$R_{PCI\_IEP\_1}$ Anything claiming to follow the PCI Express specification must follow all the specification that is software-visible to ensure standard, quality software support.

E.11 Peer-to-peer

$R_{PCI\_PP\_01}$ It is system-specific whether peer-to-peer traffic through the system is supported.

$R_{PCI\_PP\_02}$ Systems must not deadlock if PCI express devices attempt peer-to-peer transactions, even if the system does not support peer-to-peer traffic. This rule is needed to uphold the principle that a virtual machine and its assigned devices should not deadlock the system for other virtual machines or the hypervisor.

$R_{PCI\_PP\_03}$ In a system where the PCIe hierarchy allows peer-to-peer transactions, the root ports in an Arm based SoC must implement PCIe access control service (ACS) features.

$R_{PCI\_PP\_04}$ For Root ports this means that the following must be supported:

1. ACS Source Validation. (V)
2. ACS Translation Blocking. (B)
3. ACS P2P Request Redirect (R).
4. ACS P2P Completion Redirect (C).
5. ACS Upstream Forwarding (U).
6. The root port must support redirected request validation by querying an Arm architecture compliant SMMU to get the final target physical address and access permission information.

7. The root port must support ACS violation error detection, logging and reporting. Logging and reporting must be through the usage of AER mechanism.

I_{KMCPM} ACS P2P egress control is optional.

$R_{PCI\_PP\_05}$ If the Root port supports peer-to-peer traffic with other root ports, then it must support the following:

- Validation of the peer-to-peer transactions before sending it to the destination root port using the same mechanism as ACS redirected request validation. Any ACS violation error generated because of the request validation should be reported using the standard ACS violation error detection, logging and reporting mechanism specified in the PCIe specification [1].
- If the root port supports Address Translation services and peer-to-peer traffic with other root ports, then it must support ACS direct translated P2P (T).

$I_{PCI\_PP\_06}$ Since isolation between IO devices can be broken by the presence of any peer-to-peer capable entity in a PCIe hierarchy, the following is recommended for an Arm based system:

- All PCIe switches should support the following features:
  1. ACS Source Validation (V).
  2. ACS Translation Blocking (B).
  3. ACS P2P Request Redirect (R).
  4. ACS P2P Completion Redirect (C).
  5. ACS Upstream Forwarding (U).
  6. ACS Direct Translated P2P (T).
  7. ACS violation error detection, logging, and reporting as specified in the PCIe specification for ACS [1].
  8. Use of AER capability for logging and reporting ACS violation errors
- All multi-function devices, SR-IOV and non-SR-IOV (including functions in RCiEP and i-EP), that are capable of peer-to-peer traffic between different functions should support the following features:
  1. ACS P2P Request Redirect (R).
  2. ACS P2P Completion Redirect (C).
  3. ACS Direct Translated P2P (T)
  4. The device must support ACS violation error detection, logging, and reporting as specified in the PCIe specification for ACS [1].

E.12 PASID support

I_{MQVCB} SMMUv3 included optional support for PCIe PASID.

$R_{PCI\_PAS\_1}$ If the system supports PCIe PASID, then at least 16 bits of PASID must be supported. This support must be full system support, from the root complex through to the SMMUv3 and any end points for which PASID support is required.

E.13 PCIe Precision Time Measurement

$R_{PCI\_PTM\_1}$ Any system that implements PCIe *Precision Time Measurement* (PTM) [1] must use the Arm architecture defined System Counter [3] as PTM primary time source at the PTM root(s).

I_{SZTLV} Systems that are used in environments where precise coordination between the system time and a PCIe device time is necessary are recommended to support PTM on the corresponding PCIe Root Port(s).

F RCiEP and I-EP

This section describes on-chip peripherals that appear to software during enumeration as either a Root Complex Integrated EndPoint (RCiEP) or an Integrated Endpoint (i-EP). See Section 3.13 for an introduction on RCiEP and i-EP.

F.1 Rules Common for RCiEP and I-EP

This section describes the rules that are applicable to both Root Complex Integrated EndPoints and Integrated Endpoints.

$R_{RI\_CRS\_1}$ When a i-EP, RCiEP, or RCEC function is temporarily unable to process a configuration request following a reset and the reset is one of the valid reset conditions defined in Section 2.3.1 of the PCIe specification [1], the following are response options:

- Respond as defined in the PCIe specification [1] if the following are true:
  - Configuration Retry Status (CRS) visibility is present and is enabled.
  - The request is a configuration read to an address that includes the two bytes of the Vendor ID field.
- Otherwise, do not respond to the request until the function is ready.

$R_{RI\_BAR\_1}$ All BAR registers in an RCiEP, i-EP Endpoint, or i-EP Root Port must be writeable and readable as per the PCIe specification [1].

$R_{RI\_BAR\_2}$ Dynamic re-programming of these BAR registers must be allowed.

$R_{RI\_BAR\_3}$ A RCiEP, i-EP Endpoint, or i-EP Root Port must not support I/O space claimed through BARs.

$R_{RI\_INT\_1}$ If the RCiEP or i-EP supports interrupt generation, the RCiEP or i-EP must support MSI or MSI-X interrupt generation.

$R_{RI\_ORD\_1}$ The RCiEP or i-EP must obey PCIe ordering rules for the configuration and BAR mapped memory spaces when accessed in the inbound direction, towards the RCiEP or i-EP.

$R_{RI\_ORD\_2}$ PCIe ordering rules must be obeyed while sending out completions for configuration space and BAR mapped memory space accesses.

$R_{RI\_ORD\_3}$ If the RCiEP or i-EP uses the PCIe producer-consumer model for the interaction with software or peer devices, then the following must be ensured by the RCiEP or i-EP in collaboration with rest of the system:

- Write requests from the i-EP are observed by other agents in the order required for the producer-consumer model to work.
- A read request from the i-EP must not overtake previously issued write requests from the same i-EP if there is a Read after Write dependency between the Read and previously issued write or writes.

X_{TVYZJ} RCiEPs and i-EP Endpoints must interact with the SMMU in the same way that an external Endpoint does. The rules to achieve this are:

$R_{RI\_SMU\_1}$ PCIe ATS capability must be supported if the RCiEP or i-EP has a software visible cache for address translations.

$R_{RI\_SMU\_2}$ PCIe PRI mechanism must be supported if the RCiEP or i-EP Endpoint requires memory pages dynamically.

$R_{RI\_SMU\_3}$ If the RCiEP or i-EP Endpoint supports PASIDs, the PASID is used as SubStreamID as specified in the SMMU architecture specification.

$R_{RI\_SMU\_4}$ RCiEPs and i-EP Endpoints must use its BDF to generate StreamID using rules that are described in Section H.

$R_{RI\_RST\_1}$ RCiEPs and i-EP Endpoints must have Function Level Reset (FLR) support.

$R_{RI\_PWR\_1}$ RCiEPs, i-EP Endpoints, and i-EP Root Ports must have D state support and must have PCI Power management capability as specified in the PCIe specification [1].

F.2 RCiEP

This section describes the rules that are applicable when an on-chip peripheral is presented as an RCiEP. See Section 3.13 for an introduction on RCiEP.

I_{MQZDK} In this option, the on-chip peripheral appears to software during enumeration as a Root Complex Integrated EndPoint (RCiEP) that is connected to the root bus behind a host bridge. See Section 3.13 for an introduction on RCiEP.

R_{JKZMT} The RCiEP and RCEC must obey all the rules that are specified in Section E unless otherwise specified in this section.

$R_{RE\_PCI\_1}$ A RCiEP must obey all the rules that are specified in Section 1.3.2.3 of the PCIe 5.0 specification [1].

$R_{RE\_PCI\_2}$ The RCEC must obey all the rules that are specified in Section 1.3.4 of the PCIe 5.0 specification [1]. This includes for example:

- A Root Complex can contain more than one RCEC.
- Each RCEC must be associated with one or more RCiEPs.
- Each RCiEP must be associated with at most one RCEC.

I_{MTXLP} Association of RCECs to RCiEPs is defined in Sections 7.9.10.2 and 7.9.10.3 of the PCIe specification [1].

$R_{RE\_CFG\_1}$ Ability to recognize read or write requests coming in from PEs as PCIe configuration requests if the requests address is within the ECAM range that is allocated to the host bridge of RCiEP.

$R_{RE\_CFG\_2}$ An IMPLEMENTATION DEFINED mechanism for providing the bus and device number to each RCiEP.

$R_{RE\_CFG\_3}$ An IMPLEMENTATION DEFINED mechanism for providing the bus and device number to each RCEC.

I_{XYMQM} It is recommended that RCiEP Endpoint does not have any wire-based interrupts.

$R_{RE\_ORD\_4}$ The Transactions Pending bit must be cleared only after all non-posted requests, such as outstanding reads, have received responses.

$R_{RE\_PWR\_2}$ RCiEP must support PME messages for wake up signaling if the RCiEP needs to have a wake-up notification mechanism.

$R_{RE\_PWR\_3}$ PM_PME wake messages must be logged in the Root Complex Event collector that is associated with the RCiEP.

I_{RPTWW} The RCiEP is not expected to use Aux_Current.

$R_{RE\_ACS\_1}$ ACS capability must be present in the RCiEP if the RCiEP is a multi-function device and supports peer to peer traffic between its functions. It must comply with the PCIe specification on specific ACS access controls that must be supported.

$R_{RE\_ACS\_2}$ If the RCiEP has ACS capability, then it must have AER capability for reporting ACS violation errors.

$R_{RE\_ACS\_3}$ RCiEP requests that target peer Endpoints have to be mediated by the SMMU before proceeding to the target.

F.3 I-EP

This section describes the rules that are applicable when an on-chip devices is presented as an i-EP.

I_{JHJLT} In this option, the on-chip peripheral appears to software during enumeration as an endpoint that is behind a Root Port, which in turn is behind a host bridge. See Section 3.13 for an introduction on i-EP.

R_{HVZJY} The i-EP Root Port and Endpoint must obey all the rules that are specified in Section E unless otherwise specified in this section.

$R_{IE\_CFG\_1}$ Ability to recognize read or write requests coming in from PEs as PCIe configuration requests if the request's address is within the ECAM range that is allocated to the host bridge of i-EP.

$R_{IE\_CFG\_2}$ An IMPLEMENTATION DEFINED mechanism for providing the bus and device number to each i-EP.

$R_{IE\_CFG\_3}$ The i-EP Root Port must comply with all the enumeration related requirements that are given in Section E.1.

$R_{IE\_CFG\_4}$ The i-EP Endpoint must be able to capture the bus number for each of its functions.

I_{TBTCQ} If bus number consumption is a concern, then it is recommended that each i-EP has its own segment group.

$R_{IE\_ORD\_4}$ The Transactions Pending bit must be cleared only after all outstanding reads, atomic requests and write requests have received responses.

$R_{IE\_RST\_2}$ For i-EP, the Root Port must provide the ability to do a hot reset of the Endpoint using the Secondary Bus Reset bit in bridge Control Register.

$R_{IE\_RST\_3}$ For i-EP, the Root Port must provide the ability to hold the Endpoint in hot reset if the Link Disable bit in the Link Control Register is set. See the PCIe specification [1] for details on Link Disable, Secondary Bus Reset and hot reset.

$R_{IE\_PWR\_2}$ i-EP Endpoint must support PME messages for wake up signaling if the Endpoint needs to have a wake-up notification mechanism.

$R_{IE\_PWR\_3}$ PM_PME wake messages must be logged in the Root Complex Event collector that is associated with the Root Port for i-EP option.

I_{YZTHF} The i-EP Endpoint is not expected to use Aux_Current.

$R_{IE\_ACS\_1}$ ACS capability must be present in the i-EP endpoint functions if the i-EP Endpoint is a multi-function device and supports peer to peer traffic between its functions. It must comply with the PCIe specification on specific ACS access controls that must be supported. If the i-EP Endpoint has ACS capability, then it must have AER capability for reporting ACS violation errors.

$R_{IE\_ACS\_2}$ The i-EP Root Port must have ACS capability if the i-EP Endpoint can send transactions to a peer endpoint. It must comply with the PCIe specification [1] on specific ACS access controls that must be supported. If the i-EP Root Port has ACS capability, then it must have AER capability for reporting ACS violation errors.

G RCiEP and I-EP Registers

This section will describe the PCI register recommendations for Root Complex Integrated EndPoints (RCiEP) or an Integrated Endpoints (i-EP).

G.1 RCiEP capabilities and registers

A RCiEP implementation requires the following register implementation.

$R_{RE\_REG\_1}$ All type 0 header registers must be implemented for RCiEP and RCEC.

The registers are

Register	Register	Register
Device ID	Vendor ID	Status
Class Code	Revision ID	BIST
Header Type	Latency Timer	Cache Line Size
Base Address Registers	Command	Cardbus CIS Pointer
Subsystem ID	Subsystem Vendor ID	Capabilities Pointer
Exp ROM Base Address	Interrupt Pin	Interrupt Line
Max_Lat	Min_Gnt	

$R_{RE\_REG\_2}$ All registers of the PCI power management capability must be implemented for RCiEP and RCEC. The registers must be implemented as described in Table 28.

Table 28: Power management capability registers

Register	Requirement
Power Management Capabilities (PMC)	see Section G.1.1.6
Next Capability Pointer	Implement as described in the PCIe specification.
Capability ID	Implement as described in the PCIe specification.
Data	see Section G.1.1.8
Power Management Control/Status (PMCSR)	see Section G.1.1.7

$R_{RE\_REG\_3}$ The registers that are specified in Section 7.5.3 of the PCIe specification [1] for all devices must be implemented for RCiEP. The registers must be implemented as described in Table 29.

Table 29: PCI Express Capability registers required for RCiEP

Register	Requirement
PCI Express Capabilities Register	see Section G.1.1.1
Next Cap Pointer	Implement as described in the PCIe specification.
PCI Express CAP ID	Implement as described in the PCIe specification.
Device Capabilities	see Section G.1.1.2
Device Status	Implement as described in the PCIe specification.
Device Control	see Section G.1.1.3
Device Capabilities 2	see Section G.1.1.4
Device Status 2	Implement as described in the PCIe specification.
Device Control 2	see Section G.1.1.5

$R_{RE\_REC\_1}$ The registers that are specified in Section 7.5.3 of the PCIe specification [1] for Root Complex Event Collectors must be implemented for RCEC. The registers must be implemented as described in Table 30.

Table 30: PCI Express Capability registers

Register	Requirement
PCI Express Capabilities Register	see Section G.1.1.1
Next Cap Pointer	Implement as described in the PCIe specification.
PCI Express CAP ID	Implement as described in the PCIe specification.
Device Capabilities	see Section G.1.1.2
Device Status	Implement as described in the PCIe specification.
Device Control	see Section G.1.1.3
Root Capabilities	Implement as described in the PCIe specification.
Root Control	Implement as described in the PCIe specification.
Root Status	Implement as described in the PCIe specification.
Device Capabilities 2	see Section G.1.1.4
Device Status 2	Implement as described in the PCIe specification.

Register	Requirement
Device Control 2	see Section G.1.1.5

$R_{RE\_REC\_2}$ All registers of the Root Complex Event Collector Endpoint Association Extended Capability must be implemented by the RCEC. The registers must be implemented as shown in Table 31.

Table 31: RCEC Endpoint Association Extended Capability registers

Register	Requirement
PCI Express Extended Capability Header	Implement as described in the PCIe specification.
Association Bitmap for RCiEPs	Implement as described in the PCIe specification.
RCEC Associated Bus Numbers	Implement as described in the PCIe specification.

G.1.1 Register bit field rules for the RCiEP option

For all registers that are described in this section, unless otherwise specified, HW implementation and usage of each field must behave as described by the PCIe specification [1].

The attributes of all register fields must be as described in the PCIe specification [1].

Any bit or field that is specified as HWIGNORE has the following properties:

- The bit or field is a don't care for HW and the value of the bit or field will be ignored by hardware.
- The attributes must be as described in the PCIe specification [1].

G.1.1.1 PCI Express Capabilities Register

RCiEP : **Device type** must be hardwired as RCiEP type. The rest of the fields in this register must be implemented as specified in the PCIe specification [1].

RCEC : **Device type** must be hardwired as RCEC type. The rest of the fields in this register must be implemented as specified in the PCIe specification [1].

G.1.1.2 Device Capabilities Register

Device Capabilities Register	Requirement
Role based error reporting	RCEC and RCiEP: Hardwired to 1
Endpoint L0s acceptable latency	See note below
L1 acceptable latency	See note below
Captured slot power limit scale	RCEC and RCiEP: Hardwired to 0
Captured slot power limit value	RCEC and RCiEP: Hardwired to 0
Max payload size	See note below
Phantom functions	RCEC and RCiEP: Recommendation is to hardwire this bit to 0.

Device Capabilities Register	Requirement
Extended tag field	Hardwired to 1

Note

For both the RCEC and RCiEP, the value reported in this field must be compliant with the PCIe specification [1].

G.1.1.3 Device Control Register

Device Control Register field	Requirement
Max_Rd_Request Size, Max payload size	RCEC and RCiEP: HWIGNORE.
Phantom functions Enable	RCEC and RCiEP: Recommended to be hardwired to 0.
Aux power PM enable	RCEC and RCiEP: Recommended to be hardwired to 0.
Enable relaxed ordering	RCiEP: If HW can set RO or an equivalent attribute for transactions, then this bit controls RO attribute setting. Otherwise hardwired to 0.
	RCEC: Hardwired to 0.
Enable No Snoop	RCiEP: This bit enables/disables HW's ability to set non-cacheable attribute for transactions.
	RCEC: Hardwired to 0.
Extended Tag field enable	RCiEP and RCEC:HWIGNORE.

G.1.1.4 Device capabilities 2 Register

TPH Completer Supported

- RCiEP: This field must be set to appropriate valid values if the RCiEP can sink transactions with TPH hints and/or extended TPH hints.
- RCEC: Recommendation is to set this bit to 0.

10-Bit Tag Requester Supported, 10-Bit Tag Completer Supported

- HWIGNORE for RCEC and RCiEP.

End-End TLP prefix supported, Max End-End TLP prefixes

- RCiEP: Value of these bit fields are IMPLEMENTATION DEFINED.
- RCEC: Recommendation is to set these bit fields to 0.

FRS Supported

- RCiEP and RCEC: The value of this bit is IMPLEMENTATION DEFINED, based on the capability of the RCEC HW and associated RCiEP HW. If the RCiEPs support FRS, RCEC must support FRS.

LTR Mechanism Supported

- RCiEP and RCEC: This bit's value is IMPLEMENTATION DEFINED, based on the capability of the RCEC HW and associated RCiEP HW. If the RCiEP's support LTR mechanism, then the RCEC must support LTR mechanism.

10-bit tag requester supported

- RCiEP and RCEC: Recommendation is that these bits are hardwired to 1.

10-bit tag completer supported, **Extended Fmt field supported**

- RCiEP and RCEC: It is recommended that this bit is set to 1.

G.1.1.5 Device control 2 Register

IDO Request Enable:

- RCiEP: If the RCiEP HW can set an attribute equivalent to that of IDO for requests, then this bit controls the setting of that attribute. Otherwise, this bit is hardwired to 0.
- RCEC: Hardwired to 0.

IDO Completion Enable:

- RCiEP: If the RCiEP HW is capable of setting an attribute equivalent to that of IDO for completions, then this bit controls the setting of that attribute. Otherwise, this bit is hardwired to 0.
- RCEC: Hardwired to 0.

G.1.1.6 Power Management Capabilities Register

For RCiEP and RCEC **Aux_Current** must be hardwired to 0 indicating that the RCiEP/RCEC is self powered.

G.1.1.7 Power Management Control/Status Register

For both RCiEP and RCEC it is recommended that the **Data_Select** and **Data_Scale** fields are hardwired to 0.

G.1.1.8 Data Register

For both the RCEC and RCiEP it is recommended that this register is not implemented.

G.2 I-EP capabilities and registers

$R_{IE\_REG\_1}$ All type 0 header registers must be implemented for i-EP Endpoint. The registers must be implemented as described in the PCIe specification [1].

The registers are:

Register	Register	Register
Device ID	Vendor ID	Status
Class Code	Revision ID	BIST
Header Type	Latency Timer	Cache Line Size
Base Address Registers	Command	Cardbus CIS Pointer
Subsystem ID	Subsystem Vendor ID	Capabilities Pointer
Exp ROM Base Address	Interrupt Pin	Interrupt Line
Max_Lat	Min_Gnt	

$R_{IE\_REG\_2}$ The registers mandated in Section 7.5.3 of the PCIe specification [1] for an endpoint that is not a RCiEP must be implemented for i-EP Endpoint. The registers must be implemented as described in Table 35.

Table 35: PCI Express Capability registers required for Endpoint

Register	Requirement
PCI Express Capabilities Register	see Section G.1.1.1
Next Cap Pointer	Implement as described in the PCIe specification.
PCI Express CAP ID	Implement as described in the PCIe specification.
Device Capabilities	see Section G.2.1.9
Device Status	Implement as described in the PCIe specification.
Device Control	see Section G.2.1.10
Link Capabilities	see Section G.2.1.13
Link Status	see Section G.2.1.15
Link Control	see Section G.2.1.14
Device Capabilities 2	see Section G.2.1.11
Device Status 2	Implement as described in the PCIe specification.
Device Control 2	see Section G.2.1.12
Link Capabilities 2	see Section G.2.1.16
Link Status 2	see Section G.2.1.17
Link Control 2	see Section G.2.1.18

$R_{IE\_REG\_3}$ All type 1 header registers must be implemented for i-EP Root Port. The registers must be implemented as described in the PCIe specification.

The registers to be implemented as described in the PCIe specification are

Register	Register	Register
Device ID	Vendor ID	Status
Class Code	Revision ID	BIST
Header Type	Primary Latency Timer	Cache Line Size
Base Address Register 0	Base Address Register 1	Secondary Latency Timer
Subordinate Bus Number	Secondary Bus Number	Primary Bus Number
Secondary Status	I/O Limit	I/O Base
Memory Limit	Memory Base	64-bit Memory Limit

Register	Register	Register
64-bit Memory Base	64-bit Base Upper 32 Bits	64-bit Limit Upper 32 Bits
I/O Base Limit 16 Bits	I/O Base Upper 16 Bits	Capabilities Pointer
Exp ROM Base Address	Interrupt Pin	Interrupt Line

The register which have specific requirements are described in Table 37.

Table 37: Type 1 header registers requirement

Register	Requirement
Command	see Section G.2.1.1
Bridge Control	see Section G.2.1.1

$R_{IE\_REG\_4}$ The registers mandated in Section 7.5.3 of the PCIe specification [1] for Root Ports must be implemented for Root Port. The registers must be implemented as described in Table 38.

Table 38: PCI Express Capability registers required for Root Port

Register	Requirement
PCI Express Capabilities Register	see Section G.2.1.8
Next Cap Pointer	Implement as described in the PCIe specification.
PCI Express CAP ID	Implement as described in the PCIe specification.
Device Capabilities	see Section G.2.1.9
Device Status	Implement as described in the PCIe specification.
Device Control	see Section G.2.1.10
Link Capabilities	see Section G.2.1.13
Link Status	see Section G.2.1.15
Link Control	see Section G.2.1.14
Slot Capabilities	see Section G.2.1.5
Slot Status	see Section G.2.1.7
Slot Control	see Section G.2.1.6
Root Capabilities	Implement as described in the PCIe specification.

Register	Requirement
Root Control	Implement as described in the PCIe specification.
Root Status	Implement as described in the PCIe specification.
Device Capabilities 2	see Section G.2.1.11
Device Status 2	Implement as described in the PCIe specification.
Device Control 2	see Section G.2.1.12
Link Capabilities 2	see Section G.2.1.16
Link Status 2	see Section G.2.1.17
Link Control 2	see Section G.2.1.18

$R_{IE\_REG\_5}$ All registers of the PCI Power Management Capability must be implemented for i-EP Endpoint and i-EP Root Port. The registers must be implemented as described in Table 39.

Table 39: Power management capability registers required for Root Port and Endpoint

Register	Requirement
Power Management Capabilities (PMC)	see Section G.2.1.2
Next Capability Pointer	Implement as described in the PCIe specification.
Capability ID	Implement as described in the PCIe specification.
Data	see Section G.2.1.4
Power Management Control/Status (PMCSR)	see Section G.2.1.3

G.2.0.1 Supported Link Speed Declaration in Link Capabilities 2

Depending on the supported link speeds declared, the PCIe specification [1] mandates that certain additional capabilities must be present.

$R_{IE\_REG\_6}$ If the supported link speeds declared in Link Capabilities 2 includes 8GT/s or higher speeds, then the Secondary PCI Express Extended Capability Structure must be available for software to read and write. This capability must be present in both Root Port and the Endpoint. This capability has the registers that are shown in Table 40.

Table 40: Registers in Secondary PCI Express Extended Capability

Register	Requirement
PCI Express Extended Capability Header	Implement as described in the PCIe specification.
Link Control 3 Register	see Section G.2.1.20

Register	Requirement
Lane Error Status Register	see Section G.2.1.25
Lane Equalization Control Register	see Section G.2.1.24

$R_{IE\_REG\_7}$ If the supported link speeds that are declared in Link Capabilities 2 includes speeds that are 16GT/s and higher, then Datalink Feature extended capability required for i-EP root port and end point.

Table 41: Registers in Datalink Feature extended capability

Register	Requirement
PCI Express Extended Capability Header	Implement as described in the PCIe specification.
Data Link Feature Capabilities Register	see Section G.2.1.25
Data Link Feature Status Register	see Section G.2.1.25

$R_{IE\_REG\_8}$ If the supported link speeds that are declared in Link Capabilities 2 includes speeds that are 16GT/s and higher, then Physical layer 16GT/s extended capability is required for that Root Port or Endpoint.

Table 42: Registers in Physical Layer 16GT/s Extended Capability

Register	Requirement
PCI Express Extended Capability Header	Implement as described in the PCIe specification.
16.0 GT/s Capabilities Register	Implement as described in the PCIe specification.
16.0 GT/s Control Register	Implement as described in the PCIe specification.
16.0 GT/s Status Register	see Section G.2.1.23
16.0 GT/s Local Data Parity Mismatch Status Register	see Section G.2.1.26
16.0 GT/s First Retimer Data Parity Mismatch Status Register	see Section G.2.1.26
16.0 GT/s Second Retimer Data Parity Mismatch Status Register	see Section G.2.1.26

$R_{IE\_REG\_9}$ If the supported link speeds that are declared in Link Capabilities 2 includes speeds that are 16GT/s and higher, then Lane Margining at the Receiver Extended Capability is required for the Root Port and the Endpoint.

Table 43: Registers in Lane Margining at the Receiver Extended Capability

Register	Requirement
PCI Express Extended Capability Header	Implement as described in the PCIe specification.
Margining Port Status Register	Implement as described in the PCIe specification.
Margining Port Capabilities Register	see Section G.2.1.22
Margining Lane Status	see Section G.2.1.21
Margining Lane Control	see Section G.2.1.21

G.2.1 Register bit field rules for the i-EP option

For all registers that are described in this section, unless otherwise specified, HW implementation and usage of each field must behave as described by the PCIe specification [1]. Also, the attributes of all register fields must be as described in the PCIe specification [1].

Any bit or field specified as HWIGNORE has the following properties:

- The bit or field is a don't care for HW and the value of the bit or field will be ignored by hardware.
- The attributes must be as described in the PCIe specification [1].

G.2.1.1 Type 1 header registers

i-EP Root Port:

- Type 1 header registers.
  - In Bridge Control register, setting Secondary bus reset must cause a hot reset of the integrated Endpoint. See the PCIe specification for details on the effect of hot reset on the Endpoint.
  - if the BME bit in Command register is cleared, the i-EP Endpoint must not generate any memory read or write requests.

G.2.1.2 Power Management Capabilities Register

For both i-EP Root Port and Endpoint, **Aux_Current** must be hardwired to 0, this is to indicate that the i-EP Root Port/Endpoint is self powered.

G.2.1.3 Power Management Control/Status Register

For both the endpoint and the port, it is recommended that the **Data_Select** and **Data_Scale** fields are hardwired to 0.

G.2.1.4 Data Register

For both the endpoint and the port, it is recommended that this register is not implemented.

G.2.1.5 Slot Capabilities Register

All bits in this registers must be set to 0.

G.2.1.6 Slot Control Register

For the Root Port, **Data Link Layer State Changed Enable** bit must be implemented as per PCIe specification. All other bits in this register must be set to 0.

G.2.1.7 Slot Status Register

Slot Status Register field	Requirement
Data link layer state changed	i-EP Root Port: Implement as described in PCIe specification [1].
Presence detect state	i-EP Root Port: Implement as described in PCIe specification [1].
All other bits	i-EP Root Port: Always set to 0.

G.2.1.8 i-EP option: Rules for PCI Express Capabilities Register

For the i-EP Root Port:

- **Device type** must be hardwired as Root Port type.
- **Slot implemented** must be hardwired to 0b.

For the i-EP Endpoint:

- **Device type** must be hardwired as Endpoint type.

G.2.1.9 Device Capabilities Register

Device Capabilities Register	Requirement
Role based error reporting	i-EP Root Port and Endpoint: Hardwired to 1.
Endpoint L0s acceptable latency	The value reported in this field must be compliant with the PCIe specification.
Endpoint L1 acceptable latency	The value reported in this field must be compliant with the PCIe specification.
Captured slot power limit scale	i-EP Root Port and Endpoint: Hardwired to 0.
Captured slot power limit value	i-EP Root Port and Endpoint: Hardwired to 0.
Max payload size	i-EP Root Port and Endpoint: Value in this field is IMPLEMENTATION DEFINED and must be the same in both Root Port and Endpoint.
Phantom functions	i-EP Root Port and Endpoint: Recommendation is to Hardwire this bit to 0.
Function level reset capability	i-EP Root Port and Endpoint: Implement as described in the PCIe specification.
Extended Tag Field supported	i-EP Root Port and Endpoint: Hardwired to 1.

G.2.1.10 Device Control Register

Device Control Register field	Requirement
Max_Rd_Request Size, Max payload size	i-EP Root Port and Endpoint: HWIGNORE.
Phantom functions Enable	i-EP Root Port and Endpoint: Recommended to be hardwired to 0.
Aux power PM enable	i-EP Root Port and Endpoint: Recommended to be hardwired to 0.

Device Control Register field	Requirement
Enable relaxed ordering	i-EP Root Port: Recommendation is that HW does not use this bit. i-EP Endpoint: If HW can set RO or an equivalent attribute for transactions, then this bit controls RO attribute setting. Otherwise hardwired to 0.
Enable no Snoop	i-EP Root Port: HW must not use this bit. i-EP Endpoint: This bit enables/disables HW's ability to set non-cacheable attribute for transactions.
Extended Tag field enable	i-EP Root Port and Endpoint: HWIGNORE.
Initiate FLR	i-EP Root Port: Must be hardwired to 0. i-EP Endpoint: HW implementation and usage as described in the PCIe spec.

G.2.1.11 Device Capabilities 2 Register

Table 47: Device Capabilities 2 rules for the Root Port

Device Capabilities 2 Register field	Requirement
Completion timeout ranges supported	This field is hardwired to 0, if Root Port HW is not involved in transaction forwarding.
Completion timeout disable supported	This field is hardwired to 0, if Root Port HW is not involved in transaction forwarding.
ARI forwarding supported	Must be set to 1 f the Endpoint requires ARI mode.
AtomicOp routing supported	Must be set to 1 if the Endpoint needs to send atomic transactions to peer Endpoints.
32/64bit AtomicOp completer supported	Must be set to 1, if the Endpoint can generate 32 bit/64bit AtomicOps targeted towards main memory.
128bit CAS completer supported	Must be set to 1, if the Endpoint can generate 128Bit CAS atomic operations targeted towards main memory.
NO RO enabled PR-PR passing	Set to 1 only if the Endpoint sends/receives peer to peer transactions *and* needs to have RO bit set in such transactions with the restriction that posted requests must not pass older posted requests.
LTR mechanism supported	Must be the same value as that in Endpoint Device Capabilities register.

Device Capabilities 2 Register field	Requirement
TPH completer supported	Must be set to appropriate valid values if the Endpoint generates transactions with the equivalent of TPH hints and/or extended TPH hints.
LN system CLS	Must be set to appropriate values according to what the Endpoint needs in-terms of an LN completer.
10-bit tag requester supported	Hardwired to 1.
10-bit tag completer supported	Hardwired to 1.
OBFF supported	Must be set to the same value as that in Endpoint device capabilities 2 register. How the OBFF messaging/signaling implemented is IMPLEMENTATION DEFINED.
Extended Fmt field supported	Hardwired to 1.
End-End TLP prefix supported	See Note below
Max End-End TLP prefixes	See Note below
Emergency power reduction supported	Hardwired to 0
Emergency power reduction init required	Hardwired to 0
FRS supported	Set to 1 if the Endpoint generates FRS messages.

Note

Must be set to the same value as in the Endpoint Device Capabilities 2 register.

For i-EP Endpoint:

- **Emergency power reduction supported, Emergency power reduction initialization required** : It is recommended that these bits are set to 0.
- **End-End TLP prefix supported, Max End-End TLP prefixes** : Value of these bit fields are IMPLEMENTATION DEFINED based on the Endpoint capabilities for sinking TLP prefixes.
- **Extended Fmt field supported**. Hardwired to 1.
- **10-bit tag requester supported**. Hardwired to 1.

G.2.1.12 Device Control 2 Register

Table 48: Device control 2 rules for the Root Port

Device Control 2 register field	Requirement
End-End TLP prefix blocking	Implement as described in the PCIe specification. If this bit is set, then any transactions targeting the endpoint with End-End prefixes must be processed as described in the PCIe specification.

Device Control 2 register field	Requirement
Emergency power reduction request	Hardwired to 0. Not used/No effect for Root Port.
IDO completion enable	Recommendation is that HW does not use this bit.
IDO request enable	Recommendation is that HW does not use this bit.
LTR mechanism enable	If LTR is supported, and this bit is set then the HW must have a mechanism of reporting queueing and reporting LTR messages to the system from the Endpoint.
OBFF enable	If OBFF is supported and this bit is set, then the HW must have a mechanism of reporting OBFF states/cases to the Endpoint.
AtomicOp requester enable	If the endpoint dos not support AtomicOps as a completer, then this bit must be hardwired to 0. If the endpoint supports AtomicOps as a completer, and this bit is set, then AtomicOp requests from the host side, requested by PEs or by peer PCIe or peer non PCIe devices, will be accepted by the i-EP endpoint. If this bit is not set, then corresponding transaction will be given an error response.
AtomicOp egress blocking	If this bit is set, then either the Root Port or the Endpoint HW must discard any atomic requests that is received and must log an error in the *Root Port error* registers/Status register as appropriate.
Completion timeout value	If the Root Port HW is not involved in forwarding transactions, then this field is hardwired to 0.
Completion timeout disable	If Root Port HW is not involved in transaction forwarding, this bit is hardwired to 0.
ARI forwarding enable	Implement as described in the PCIe specification.
10 bit tag requester enable	HWIGNORE.

For the i-EP endpoint:

IDO Request Enable:

- If the i-EP Endpoint HW is capable of setting an attribute equivalent to that of IDO for requests, then this bit controls the setting of that attribute. Otherwise, this bit is hardwired to 0.

IDO Completion Enable:

- If the i-EP Endpoint HW is capable of setting an attribute equivalent to that of IDO for completions, then this bit controls the setting of that attribute. Otherwise, this bit is hardwired to 0.

10 bit tag requester enable

- HWIGNORE.

G.2.1.13 Link Capabilities Register

Link Capabilities Register field	Requirement
ASPM support	i-EP Root Port and Endpoint: Hardwired to 0
L1 exit latency	i-EP Root Port and Endpoint: Implement as described in the PCIe specification.
Clock power management	i-EP Root Port and Endpoint: Hardwired to 0
Surprise down error reporting capable	i-EP Root Port and Endpoint: Hardwired to 0
Max link speed, Max link width	i-EP Root Port and Endpoint: The value in these fields are implementation defined but it must obey the following conditions: * The value in these fields must be the same in both Endpoint and Root Port. * The value must be one of the encodings defined in the PCIe specification [1].
Port number	i-EP Root Port: Value is IMPLEMENTATION DEFINED. Must be unique for each Root Port. i-EP Endpoint: Value is IMPLEMENTATION DEFINED.

G.2.1.14 Link Control Register

Link Control Register field	Requirement
Common clock configuration	i-EP Root Port and Endpoint: HWIGNORE.
Extended synch	i-EP Root Port and Endpoint: HWIGNORE.
Enable clock power management	i-EP Root Port and Endpoint: HWIGNORE.
ASPM control	i-EP Root Port and Endpoint: HWIGNORE.
DRS signaling control	i-EP Root Port and Endpoint: Implement as per PCIe specification [1].
RCB control	i-EP Root Port and Endpoint: HWIGNORE.
ASPM optionality compliance	i-EP Root Port and Endpoint: Implement as described in PCIe specification.
Link disable	i-EP Root Port: If this bit is set, the Endpoint will be held in reset that is equivalent to hot reset. See the PCIe specification for details. i-EP Endpoint: Implement as described in the PCIe specification.
Retrain Link	i-EP Root Port: When a 1 is written to this bit, and the target link speed value is different from default in either the EP or the RP, then the current link speed field must be changed to the minimum of target link speed value of both EP and the RP. As per PCIe specification, read of this bit must always return 0.

Link Control Register field	Requirement
	i-EP Endpoint: Implement as described in the PCIe specification.

G.2.1.15 Link Status Register

Link Status Register field	Requirement
Current link speed	i-EP Root Port and Endpoint: The value in this field is the minimum of the target link speed value field of the Root Port and that of the Endpoint. The Root Port and end point link status registers must have the same value in this field.
Negotiated link width	i-EP Root Port and Endpoint: The value in this field is IMPLEMENTATION DEFINED, but the Root Port and endpoint Link Status registers must have the same value.
Slot clock configuration	i-EP Root Port and Endpoint: Hardwired to 0.
Link autonomous bandwidth mgnt status	i-EP Root Port: Set this bit to 1 on or before Data Link layer link active bit goes from 0 to 1. See the PCIe specification for details. i-EP Endpoint: Implement as described in the PCIe specification.
Data link layer link active	See note below.
Link training	i-EP Root Port: Set this bit to 1 whenever the retrain link bit is set, or an IMPLEMENTATION DEFINED amount of time after reset de-assertion. Clear this bit prior to Data Link Layer link active bit is set. This bit must be cleared only after equalization status bits are set for speeds where equalization is required. Default value is 0b. i-EP Endpoint: Implement as described in the PCIe specification.
Link Bandwidth Management Status	i-EP Root Port: Set this bit when the retrain link bit is set. The delay between retrain bit being set and this bit being set is IMPLEMENTATION DEFINED. i-EP Endpoint: Implement as described in the PCIe specification.

Note

i-EP Root Port and Endpoint: Hardwired to 0 if link speed is less than 5 GT/s. If speed is greater than or equal to 5GT/s, then this bit must be set to 1 for the following conditions:

- After reset deassertion, the delay between reset de-assertion and this bit going from 0 to 1 is IMPLEMENTATION DEFINED.
- After a 1 to 0 transition of the link disable bit occurs, the delay between link disable de-assertion and this bit going from 0 to 1 is IMPLEMENTATION DEFINED.
- After a 1 to 0 transition of the secondary bus reset bit, the delay between secondary bus reset de-assertion and this bit going from 0 to 1 is IMPLEMENTATION DEFINED.

This bit must be set to 0 for the following conditions:

- After reset assertion.
- After a 0 to 1 transition of the Link disable bit in Link Control Register.
- After a 0 to 1 transition of the secondary bus reset bit in bridge control.

See the PCIe specification [1] for details. i-EP Root Port and Endpoint must have the same value for this bit.

G.2.1.16 Link Capabilities 2 Register

Link Capabilities 2 Register field	Requirement
Supported link speeds vector	i-EP Root Port and Endpoint: Value in this field is IMPLEMENTATION DEFINED. The value must be compliant to PCIe specification [1]. Both Root Port and Endpoint must have the same value in this field.
Cross link supported	i-EP Root Port and Endpoint: Hardwired to 0
Lower SKP OS supported speeds vector	i-EP Root Port and Endpoint: Hardwired to 0
Lower SKP OS reception Supported speeds vector	i-EP Root Port and Endpoint: Hardwired to 0
Retimer presence detect supported	i-EP Root Port and Endpoint: Hardwired to 0
Two retimers presence detect supported	i-EP Root Port and Endpoint: Hardwired to 0
DRS supported	i-EP Root Port: If the Endpoint needs to support DRS or FRS, then this bit must be set for the Root Port as well. i-EP Endpoint: Implement as described in the PCIe specification.

G.2.1.17 Link Status 2 Register

Link Status 2 Register field	Requirement
Current de-emphasis level	i-EP Root Port and Endpoint: Value in this field is IMPLEMENTATION DEFINED and is present for only emulating the link.
Equalization 8.0 GT/s successful	i-EP Root Port and Endpoint: See note below
Equalization 8.0 GT/s phase 1 successful	i-EP Root Port and Endpoint: See note below
Equalization 8.0 GT/s phase 2 successful	i-EP Root Port and Endpoint: See note below
Equalization 8.0 GT/s phase 3 successful	i-EP Root Port and Endpoint: See note below

Link Status 2 Register field	Requirement
Link equalization request	i-EP Root Port and Endpoint: HW must never set this bit.
Retimer presence detected	i-EP Root Port and Endpoint: Hardwired to 0
Two retimers presence detected	i-EP Root Port and Endpoint: Hardwired to 0
Crosslink resolution	i-EP Root Port and Endpoint: Hardwired to 0
Downstream component presence	i-EP Root Port: Only 010, 100 and 101 are allowed. Also, when the link state goes from inactive to active, this field must change in value from 010 to 100/101. When the link state goes from active to inactive, this field must change from 100/101 to 010. See the PCIe specification for details. i-EP Endpoint: Implement as described in the PCIe specification.
DRS message received	i-EP Root Port and Endpoint: Implement as described in PCIe specification [1].

Note

This bit is set to 1 if the following conditions are true:

- The Data Link Layer Link Active bit has a 0 to 1 transition.
- Supported Link speeds includes 8GT/s and higher speeds.
- 32 GT/s speed is not supported or 32GT/s is supported and Equalization bypass to highest rate Disable is set in 32GT/s Control Register.

See Section 4.2.3 and 4.2.6.4.2 of the PCIe specification [1] for details or bypass mode. Note that this bit must go to 1 before the Link Active bit in Data Link Layer goes to 1.

This bit is set to 0 for an IMPLEMENTATION DEFINED amount of time if one of the following conditions is true:

- Link disable bit transitions from 1 to 0.
- Hot reset bit transitions from 1 to 0.

If Retrain Link bit is set to 1, target link speed is 8GT/s and perform equalization bit in Link Control 3 register is set, this bit is set to 0 for an IMPLEMENTATION DEFINED amount of time. The perform equalization bit must be cleared after this bit is set to 0.

The amount of time for which this bit is set to 0 must meet the following constraints:

- Large enough for software polling to succeed.
- Less than the delay between hot reset bit going to 0 and the Data Link Layer link active bit going to 1.
- Less than the delay between link disable bit going to 0 and the Data Link Layer link active bit going to 1.

The Endpoint and Root Port must have the same value for this bit. See the PCIe specification [1] for details.

G.2.1.18 Link Control 2 Register

Link Control 2 Register field	Requirement
Target link speed	i-EP Root Port and Endpoint: Implement as per PCIe specification. The Current link speed field in the Link Status Register in Root Port and Endpoint would be the minimum of Root Port's target link speed field and end point's target link speed field.
Enter compliance	i-EP Root Port: HWIGNORE. i-EP Endpoint: If this field is set to 1, then writes to Target link speed will take effect. See the PCIe specification for details.
Enter modified compliance	i-EP Root Port and Endpoint: HWIGNORE.
Selectable deemphasis	i-EP Root Port and Endpoint: HWIGNORE.
Transmit margin	i-EP Root Port and Endpoint: HWIGNORE.
Enter modified compliance	i-EP Root Port and Endpoint: HWIGNORE.
Compliance SOS	i-EP Root Port and Endpoint: HWIGNORE.
Compliance preset/deemphasis	i-EP Root Port and Endpoint: HWIGNORE.

G.2.1.19 Lane Equalization Control Register

For both the Root Port and Endpoint, all fields in this register must show values that are as per the encoding specified in the PCIe specification [1].

G.2.1.20 Link Control 3 Register

Link Control 3 Register field	Requirement
Perform Equalization	i-EP Root Port: See note on Equalization Successful bits in Section G.2.1.17 and Section G.2.1.23. i-EP endpoint: Implement as described in PCIe specification [1].
Enable Lower SKP OS Generation Vector	i-EP Root Port and Endpoint: Attributes as per PCIe specification. HWIGNORE.
Link Equalization Request Interrupt Enable	i-EP Root Port and Endpoint: HWIGNORE.

G.2.1.21 Margining Lane Control Register and Margining Lane Status Register

For both Root Port and endpoint, these two registers must emulate the behavior of the same registers in a Root port with real links. Only time margining needs to be supported and emulated. Healthy margin values must be presented to software, when software does time margining.

G.2.1.22 Margining Port Capabilities Register

For both Root Port and endpoint, the *margining uses Driver Software* field is set to 0.

G.2.1.23 16.0 GT/s Status Register

16.0 GT/s Status Register field	Requirement
Equalization 16.0 GT/s complete	i-EP Root Port and Endpoint: See note below
Equalization 16.0 GT/s phase 1 successful	i-EP Root Port and Endpoint: See note below
Equalization 16.0 GT/s phase 2 successful	i-EP Root Port and Endpoint: See note below
Equalization 16.0 GT/s phase 3 successful	i-EP Root Port and Endpoint: See note below
Link equalization request 16 GT/s	i-EP Root Port and Endpoint: HW must never set this bit.

Note

This bit is set to 1 if the following conditions are true:

- The Data Link Layer Link Active bit has a 0 to 1 transition.
- Supported Link speeds includes 16GT/s and higher speeds.
- 32 GT/s speed is not supported or 32GT/s is supported and Equalization bypass to highest rate Disable is set in 32GT/s Control Register.

See Section 4.2.3 and 4.2.6.4.2 of the PCIe specification [1] for details on bypass mode.

Note that this bit must go to 1 before the data link layer link active bit goes to 1.

This bit is set to 0 for an IMPLEMENTATION DEFINED amount of time if one of the following conditions is true:

- Link disable bit transitions from 1 to 0.
- Hot reset bit transitions from 1 to 0.

If Retrain link bit is set to 1, target link speed is 16GT/s or higher, and perform link equalization bit in link Control 3 register is set, this bit is set to 0 for an IMPLEMENTATION DEFINED amount of time. The perform equalization bit must be cleared after this bit is set to 0.

The amount of time for which this bit is set to 0 must meet the following constraints:

- Large enough for software polling to succeed.
- Less than the delay between hot reset bit going to 0 and the Data Link Layer link active bit going to 1.
- Less than the delay between link disable bit going to 0 and the Data Link Layer link active bit going to 1.

The endpoint and Root Port must have the same value for this bit. See the PCIe specification [1] for details.

G.2.1.24 i-EP option: Rules for 16.0 GT/s Lane Equalization Control Register

For both i-EP Root Port and Endpoint, this register, which is per lane, must have values in all fields which are within expected ranges as described in the PCIe specification [1].

G.2.1.25 Data Link Feature Capabilities and Data Link Feature Status

For i-EP Root Port:

- These registers must emulate the behavior of the same registers in a port with a real link.

G.2.1.26 16GT/s Registers and Lane Error Status Register

For both i-EP Root Port and endpoint, The following registers if present, must have all bits set to 0:

- 16.0 GT/s Local Data Parity Mismatch Status register
- 16.0 GT/s First Retimer Data Parity Mismatch Status register
- 16.0 GT/s Second Retimer Data Parity Mismatch Status register
- Lane Error Status Register

H DeviceID generation and ITS groups

H.1 ITS groups

H.1.1 Background

I_{FXFVQ} Every ITS block in the system is a member of a logical ITS group. Devices that send MSIs are also associated with an ITS group. Devices are only programmed to send MSIs to an ITS in their group. In the simplest case, the system contains one ITS group which contains all devices and ITS blocks. Devices are assigned DeviceID values within each ITS group. See Section H.2.

I_{FPCXC} The concept of ITS grouping means the system does not have to support the use of any ITS block from any device, which can ease system design.

H.1.2 Rules

I_{KHBVS} The system contains one or more ITS group(s).

R_{ITS_01} An ITS group can contain one or more ITS blocks.

R_{ITS_02} An ITS block is associated with one ITS group.

R_{ITS_03} A device that is expected to send an MSI is associated with one ITS group.

R_{ITS_04} Devices can be programmed to send MSIs to any ITS block within the group.

R_{ITS_05} If a device sends an MSI to an ITS block outside of its assigned group, the MSI write is illegal and does not trigger an interrupt that could appear to originate from a different device. See Section H.2.2 for permitted behavior of illegal MSI writes.

I_{YRLQM} The association of devices and ITS blocks to ITS groups is considered static by high-level software.

R_{ITS_06} An ITS group represents a DeviceID namespace independent of any other ITS group.

R_{ITS_07} All ITS blocks within an ITS group support a common DeviceID namespace size, a common input EventID namespace size and are capable of receiving an MSI from any device within the group.

R_{ITS_08} All ITS blocks within an ITS group observe the same DeviceID for any given device in the same ITS group.

I_{WHKPM} Rules ITS_07 and ITS_08 allow software to use ITS blocks sharing a common group interchangeably.

I_{JCCHC} System firmware data, for example, firmware tables like ACPI/FDT, describe the association of ITS blocks and devices with ITS groups to high-level software.

I_{VSKNR} It is recommended that the DeviceID namespace in each group is packed as densely as possible, and that it starts at 0.

I_{MJZDG} It is not required that all DeviceIDs be entirely contiguous but excessive fragmentation makes the software configuration of an ITS more difficult.

H.1.3 Examples of ITS groups

In Figure 9:

- ITS A serves two devices.
- ITS B and ITS C serve three devices. Any of these devices can send an MSI to either B or C.
- Two unrelated DeviceID namespaces exist. DeviceID 0 in Group 0 is different to DeviceID 0 in Group 1.
- A device in Group 0 can only trigger an MSI on its assigned ITS, A, and should not be configured to do otherwise. It cannot send an MSI to ITS B as it is in a different group to the device. If this is done, the MSI write might be ignored or aborted, but in any case, does not cause an interrupt that might appear to be valid.

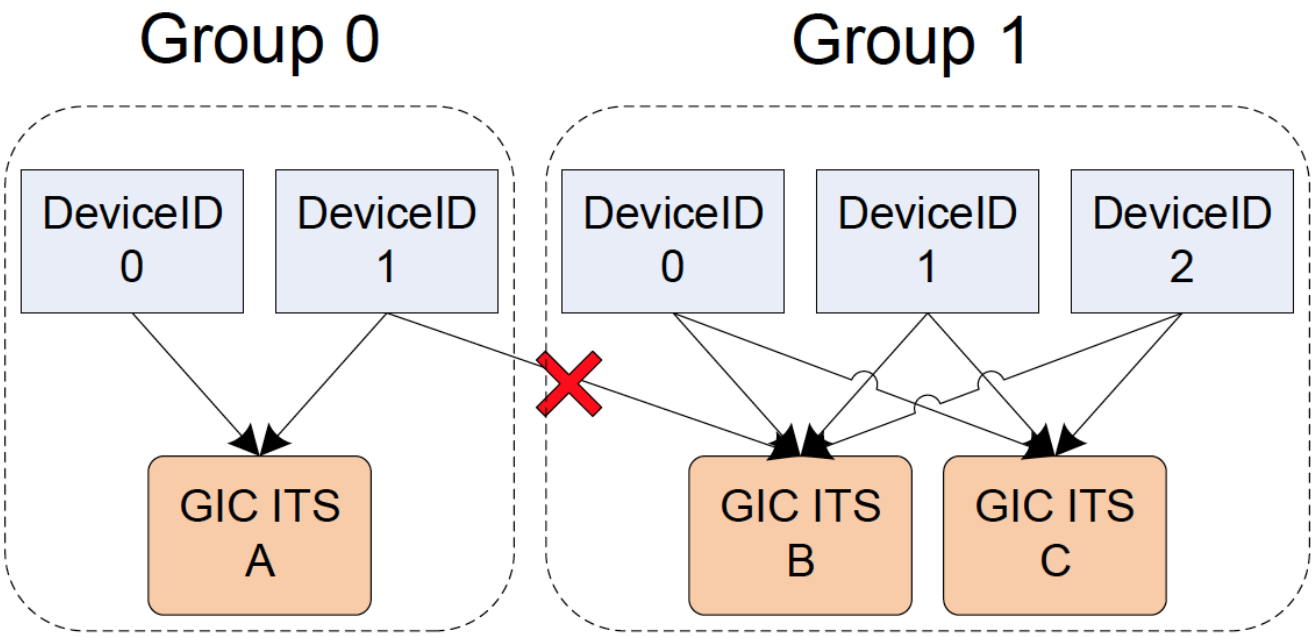

Figure 9: Device and ITS grouping

I_{MZSYJ} The properties that system-description structures convey to high-level software are:

- Identification of the two devices that are associated with Group 0, and the three associated with Group 1.
- ITS block A is in Group 0, B and C are in Group 1.
- For each MSI-capable requester device, which DeviceID in the group namespace the device has been assigned.

H.2 Generation of DeviceID values

H.2.1 Background

$R_{ITS\_DEV\_1}$ Every device that is expected to send MSIs has a DeviceID associated with it.

I_{YHQWJ} This DeviceID is used to program the interrupt properties of MSIs originating from each device. The term “device” is used in the context of a logical programming interface used by one body of software.

I_{NQLJZ} Where a device is a client of an SMMU, that is, behind an SMMU, a granularity of source identification is typically chosen to distinguish the client device traffic from other clients. This allows the device to be assigned to a less-privileged software independent of other SMMU client devices.

$R_{ITS\_DEV\_2}$ The system designer assigns a requester a unique StreamID to device traffic input to the SMMU.

I_{GFHLX} The simplest way to achieve the same granularity of interrupt source differentiation and SMMU DMA differentiation is for the device’s DeviceID to be generated from the device’s SMMU StreamID. It is beneficial for high-level software and firmware system descriptions to ensure that this relationship is as simple as possible. DeviceID is derived from a StreamID 1:1 or with a simple linear offset.

$R_{ITS\_DEV\_3}$ When a device is not behind an SMMU, its DeviceID appears to high-level software as though it is assigned directly by the system designer.

I_{NLHTN} If a requester is a bridge from a different interconnect with an originator ID, like a PCIe RequesterID, and devices on that interconnect might need to send MSIs, the originator ID is used to generate a DeviceID. The function to generate the DeviceID should be an identity or a simple offset.

X_{BCDZS} The overall principle of DeviceID and StreamID mapping is that the relationship between one ID space, for example, a PCIe RequesterID namespace and a DeviceID, be easily described using linear span-and-offset operations.

X_{XFJZL} When an SMMU is used to allow devices to be programmed by possibly malicious software that is not the most privileged part of the system, devices that are not designed to directly trigger MSIs could be misused to direct a DMA write transaction at an ITS MSI target register.

$R_{ITS\_DEV\_4}$ The system must not allow this behavior to trigger an MSI that masquerades as originating from a different requester. The system must anticipate that PEs also have the potential to be misused in this manner.

I_{GGRYN} Exposing an ITS to a VM for legitimate MSI purposes can mean that the untrusted VM software is able to write to the ITS MSI target register from a PE.

H.2.2 Rules

$R_{ITS\_DEV\_5}$ Every device that is expected to originate MSIs is associated with a DeviceID.

$R_{ITS\_DEV\_6}$ DeviceID arrangement and system design prevents any mechanism that any software that is not the most privileged in the system, for example VM, or application, can exploit to trigger interrupts associated with a different body of software, for example, a different VM, or OS driver.

- A write to an ITS GITS_TRANSLATER from a PE, or from a device that is known at design time to not support genuine MSIs and is under control of software less privileged than the software controlling the ITS, is an illegal MSI write and must not be able to trigger an MSI appearing to have a DeviceID associated with a different device. See Section H.1.2, an MSI sent to an ITS in a group different to the originating device is also an illegal MSI write.
- An illegal MSI write is permitted to complete with WI semantics, or be terminated with an abort, or trigger an MSI having a DeviceID that does not alias any DeviceID of a legitimate source.

I_{NMTPW} Devices that are known at design time to only be controlled by the most privileged software in the system, such as those without an MMU/MPU, can be trusted not to send malicious writes to the ITS. For these devices no special steps are required to prevent malicious MSI writes. Devices that have the potential to be controlled by a VM cannot be trusted. Devices that are clients of an SMMU fall into the latter category.

$R_{ITS\_DEV\_7}$ If a device is a client of an SMMU, the associated DeviceID is derived from the SMMU's StreamID with an identity or simple offset function:

- The SMMU component must output the input StreamID unmodified so it can be used to derive the DeviceID downstream of the SMMU.
- If two devices have different StreamIDs, they must also have distinct DeviceIDs.
  - It is not permitted for >1 StreamID to be associated with 1 DeviceID.
  - It is not permitted for >1 DeviceID to be associated with 1 StreamID.
- The generic StreamID to DeviceID relationship is:
  - `DeviceID = zero_extend( SMMU_StreamID )+ Constant_Offset_A`
- A PCIe Root Complex behind an SMMU generates a StreamID on that SMMU from its RequesterID with this relationship:
  - `StreamID = zero_extend( RequesterID[N-1:0] )+ (1<<N)*Constant_B`
  - This StreamID is then used post-SMMU, as above, to generate a DeviceID.

I_{VTHVX} For rule ITS_DEV_7, Arm expects N above to be 16 bits, but this is not mandatory.

$R_{ITS\_DEV\_8}$ DeviceIDs derived from other kinds of system IDs are also created from an identity or simple offset function. For a Root Complex without an SMMU, the relationship is:

```
DeviceID = zero_extend( RequesterID[N-1:0] )+ (1<<N)*Constant_C
```

$R_{ITS\_DEV\_9}$ The relationships between a device, its StreamID and its DeviceID are considered static by OS or hypervisor software. If the mapping is not fixed by hardware, the relationship between a StreamID and a DeviceID must not change after system initialization, and OS drivers must not be required to set it up.

I_{WHTNW} Am recommends that all devices expected to originate MSIs have a DeviceID unique to their ITS group, even if the devices are not connected to an SMMU.

I_{FZRWH} Providing separate DeviceIDs for different devices can improve the efficiency of structure allocation in GIC driver software.

H.3 System description of DeviceID and ITS groups from firmware data

I_{VLNPJ} The properties of the GIC distributor, Redistributors and ITS blocks such as base addresses will be described to high-level software by system firmware data. Also, for any given device expected to send MSIs, system firmware data tables must ensure that:

- The DeviceID of the device can be determined, either:
  - Directly: A device is labeled with a DeviceID value.
  - Hierarchically indirect: If a device has a known ID on a sub-interconnect, the transformation between that interconnect ID and the DeviceID namespace is described in a manner that allows the DeviceID to be derived. This might comprise multiple transformations ascending a hierarchy, where a device is associated with intermediate IDs (such as a StreamID) which are ultimately used to generate a DeviceID.
    - Example: A PCIe Root Complex without an SMMU is described in terms of the DeviceID range output for its RequesterID range. The DeviceID of an endpoint served by the Root Complex is not directly provided, but is derived from the endpoint's RequesterID given the described mapping.
    - Example: A PCIe Root Complex with an SMMU is described in terms of the transformation of RequesterID range to SMMU input StreamID range and the transformation of StreamID range to DeviceID range.
- The device association with an ITS group can be determined and the ITS blocks within the group can be enumerated.

I_{WVHQB} More compact descriptions result by describing a range of DeviceIDs to allow DeviceIDs to be derived from a formula instead of directly describing individual DeviceIDs. This is especially pertinent for interconnects such as PCIe.

I_{DRBMW} PEs and other requesters that do not support MSIs are not described as being part of an ITS group; as they are not intended to invoke valid MSIs, there is no association to an ITS on which it is valid to invoke MSIs.

I_{MSZFP} The DeviceID and ITS group associations are not expected to be discoverable through a programming interface of hardware components and a system is not required to provide such an interface.

H.4 DeviceIDs from hot-plugged devices

I_{LCHZT} If a device is not physically present at system initialization time, values in the DeviceID namespace appropriate to the potential physical location of future devices must be reserved and associated with the device when it later becomes present, in a system-specific manner.

I_{ZHKSC} When a device is hot-plugged, it can be enumerated using an interconnect ID whose mapping to DeviceID was statically described in system description tables and its DeviceID derived from this existing mapping.

I_{PPLPF} If a new device's DeviceID cannot be derived from existing mappings in system description tables, the hot-plug mechanism, for example via firmware, must provide a means to determine the new device's DeviceID.

I_{JPPGT} These points also apply to a new device's SMMU StreamID.

I_{QJKLV} In current systems, hot-plug device that are capable of sending MSIs are most likely to be PCIe endpoints. When a system and PCIe-specific mechanism makes a new endpoint present, the existing indirect description of the Root Complex's DeviceID span is used to calculate the new DeviceID from the new RequesterID.

I_{JTKGJ} It is recommended that description of a sub-interconnect bridge, such as a PCIe Root Complex, includes all potential endpoints (on PCIe, up to 2^{16}) rather than limiting description to the endpoints present at boot time, if more client endpoints can later become present.

I GICv2m Architecture

I.1 Background

The purpose of this section is to provide a standardized means for ensuring that each individual PCI Express MSI and MSI-X message is presented to the operating system (OS) as a unique interrupt ID. It is designed to be used alongside an existing GICv2 implementation.

The standard abstraction that is expected to be given to the OS is the address of the MSI_SETSPI register and the set of SPIs that it can generate. It is expected that this abstraction will be represented by system firmware data.

I.2 About the GICv2m architecture

GICv2m provides an extension to GICv2 Generic Interrupt Controller Architecture, which enables MSIs to set GICv2 Shared Peripheral Interrupts (SPIs) to pending. This provides a similar mechanism to the message-based interrupt features added in GICv3.

The additional registers provided by GICv2m are specified as an additional memory-mapped Non-secure MSI register frame, described in Non-secure MSI register summary in Section I.7. This allows a GICv2m implementation to be built by adding a component that implements the additional registers to an existing GICv2-compatible interrupt controller. The additional component is connected to a subset of the SPI inputs to the GICv2 interrupt controller. When the additional component receives an MSI it generates an edge on the corresponding SPI input.

I.3 Security

GICv2m can optionally include Security Extensions to include support for Secure MSI or MSI-X. The GICv2m Security Extensions are optional even when the GIC Security Extensions are included. However, if the GICv2m Security Extensions are included the GIC Security Extensions are mandatory.

Table 57: GIC and GICv2m security extensions

GIC Security Extensions	GICv2m Security Extensions	Description
Not included	Not included	No support for Secure interrupts.
Included	Not included	MSI are Non-secure. Other interrupts can be Secure or Non-secure
Not included	Included	Not supported.
Included	Included	All interrupts can be Secure and Non-secure.

The inclusion of the GICv2m Security Extensions adds a further memory-mapped Secure MSI register frame, described in Section I.8.

I.4 Virtualization

To support virtualization, GICv2m supports the inclusion of an IMPLEMENTATION DEFINED number of instances of the Non-secure MSI register frame.

A hypervisor can allocate one or more instance to each guest operating system. Stage 2 translation tables

ensure each PCI Express function only has visibility of the Non-secure MSI register frames allocated to the operating system that is controlling the device. This ensures that a guest operating system is not able to program a PCI Express device to trigger MSI interrupts allocated to another guest operating system.

I.5 SPI allocation

GICv2m allows the allocation of SPIs to each of the register frames defined by the architecture.

Each instance of the Non-secure MSI register frame is allocated an IMPLEMENTATION DEFINED number of contiguous SPIs. For details of Non-secure MSI register frame instances, see Section I.4.

When GICv2m includes the Security Extensions, an additional IMPLEMENTATION DEFINED number of contiguous SPIs are allocated for Secure MSI.

The Secure MSI SPI range and the Non-secure MSI SPI range must not overlap, and are not required to be adjacent.

SPIs that are allocated to MSIs must only be controllable by the GICv2m MSI registers. This means that other interrupt sources must not share SPIs that are allocated as MSIs.

I.6 GICv2 programming

MSIs have edge-triggered properties. All SPIs that are allocated to MSIs must be programmed as edge-triggered in the appropriate GICv2 GICD_ICFGRn registers. For details of the GICD_ICFGRn registers see the *Arm Generic Interrupt Controller v2 Architecture Specification* [16].

In implementations that include the GICv2m Security Extensions, Secure system software must program the GIC so that:

- SPIs that are allocated to Secure MSI can be defined as Secure or Non-secure interrupts.
- SPIs that are allocated to Non-secure MSI must be defined as

Non-secure interrupts, unless the GIC has been configured to permit

Non-secure software to create and manage the interrupt.

When used with a processor that includes the Arm Security Extensions, this means that SPIs allocated to Secure MSI must be included in Group 0, and SPIs allocated to Non-secure MSI must be included in Group 1. This is achieved using the GICv2 GICD_IGROUPRn registers. Additionally, Non-secure software can be permitted to manage a Group 0 interrupt using the GICv2 GICD_NSACRn registers. For details of the GICD_IGROUPRn and GICD_NSACRn registers see the *Arm Generic Interrupt Controller v2 Architecture Specification* [16].

When using GICv2m, it is a programming error to incorrectly define the security of SPIs mapped to Non-secure MSI interrupts in GICv2. This will adversely affect the ability to port GICv2m-compatible software to GICv3.

I.7 Non-secure MSI register summary

This section summarizes the Non-secure MSI registers, relative to a base memory address. This register frame is present in all GICv2m implementations.

This register frame is 4KB in size, and all registers are 32 bits wide. It must be accessible using Non-secure accesses. All registers have similar behavior to equivalent registers in the GICv3 distributor.

Table 58: GICv2m Non-secure MSI register summary

Offset	Name	Description
0x000-0x007	-	Reserved.
0x008	MSI_TYPER	See Section I.9.1.
0x00C-0x03C	-	Reserved.
0x040	MSI_SETSPI_NS	See Section I.9.2.
0x044-0xFC8	-	Reserved.
0xFCC	MSI_IIDR	See Section I.9.3.
0xFD0-0xFFC	-	IMPLEMENTATION DEFINED.

I.8 Secure MSI register summary

This section summarizes the optional Secure MSI registers, relative to a base memory address. This register frame is only included in GICv2m implementations that include the optional GICv2m Security Extensions.

This register frame is 4KB in size, and all registers are 32 bits wide. It must only be accessible using Secure accesses. All registers have similar behavior to equivalent registers in the GICv3 distributor

Table 59: GICv2m Non-secure MSI register summary

Offset	Name	Description
0x000-0x007	-	Reserved.
0x008	MSI_TYPER	See Section I.9.1.
0x00C-0x03C	-	Reserved.
0x040	MSI_SETSPI_S	See Section I.9.2.
0x044-0xFC8	-	Reserved.
0xFCC	MSI_IIDR	See Section I.9.3.
0xFD0-0xFFC	-	IMPLEMENTATION DEFINED.

I.9 Register descriptions

All registers must support 32-bit word accesses. The MSI_SETSPI_S and MSI_SETSPI_NS registers must also support 16-bit writes to bits [15:0]. Whether other access sizes are permitted is IMPLEMENTATION DEFINED.

The GICv2m is little-endian.

I.9.1 MSI type register

MSI_TYPER is a 32-bit read-only register that provides information about the SPIs that are assigned to the MSI frame. For information about how SPIs are assigned to each frame, see Section I.5.

The format of the register is:

Bits [31:26]

Reserved, RES0.

Base SPI number, bits [25:16]

Returns the IMPLEMENTATION DEFINED ID of the lowest SPI assigned to the frame. SPI ID values must be in the range 32 to 1020.

Bits [15:10]

Reserved, RES0.

Number of SPIs, bits [9:0]

Returns the IMPLEMENTATION DEFINED number of contiguous SPIs assigned to the frame.

I.9.2 Set SPI register

MSI_SETSPI_NS and MSI_SETSPI_S are 32-bit write-only registers.

The format of the register is:

Bits [31:10]

Reserved, RES0.

SPI, bits [9:0]

On a write, an edge-triggered interrupt is generated to the GICv2 generic interrupt controller for an SPI with the ID identified by the value of this field. If the resulting value does not identify an SPI that is allocated to this frame, the write has no effect.

I.9.3 MSI Interface Identification Register

MSI_IIDR is a 32-bit read-only register.

The format of the register is:

ProductID, bits [31:20]

An IMPLEMENTATION DEFINED product identifier.

Architecture version, bits [19:16]

Revision field for the GICv2m architecture. The value of this field depends on the GICv2m architecture version:

- 0x0 for GICv2m v0.

Revision, bits [15:12]

An IMPLEMENTATION DEFINED revision number for the component.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the GICv2m:

Bits [11:8] The JEP106 continuation code of the implementer.

Bit [7] Always 0.

Bits [6:0] The JEP106 identity code of the implementer.

J GICv2m compatibility in a GICv3 system

A key difference between GICv3[2] and GICv2[16] is that GICv3 can support more than 8 PEs, which is the maximum supported by GICv2. To achieve this, there is a change in some of the architectural concepts. GICv3 has introduced a new type of interrupt, called LPI, which is designed to be more scalable. It also changes the routing semantics of SPI to enable them to scale to more than 8 PEs.

However, GICv3 supports full backwards compatibility with GICv2 when ARE==SRE==0.

GICv2m is an extension of the GICv2 architecture that adds register frames to support MSI(-X). This note explains how to achieve compatibility between a GICv3 hardware system and GICv2m software.

J.1 GICv2m-based hypervisor (GICv2m guests) or GICv2m OS without hypervisor

The GICv3 must be configured to have SRE==ARE==0 and can therefore only be used with 8 PEs or less, but is fully GICv2 compatible. To be GICv2m compatible, the hardware system must implement GICv2m register frames for MSI support.

J.2 GICv3-based hypervisor with GICv2m guest OS

The hypervisor runs with ARE==1 so can address > 8 PEs.

The GICv2m guests run with EL1.SRE==EL1.ARE==0 which aligns with GICv2 functionality. The guest must be restricted to eight PEs or fewer.

The GICv2m register frames are not needed for the guests though as long as the OS is using a suitable abstraction for MSI support. The expected abstraction for the MSI targets is the tuple of (register address, interrupt ID set).

It is expected that the firmware interface of the OS will hand over a set of these MSI registers to the OS, which in this case will be supplied by the hypervisor.

In this compatibility case, the hypervisor hands over the address of GITS_TRANSLATER and a set of IDs (the ID set need to be in the valid SPI range of 32-1019). The hypervisor creates a single interrupt translation table for all the devices that belong to the OS, and creates translations for the IDs handed to the OS to unique LPIs.

Whenever a device sends an MSI, the hypervisor receives the corresponding LPI. The hypervisor then posts the original ID to the guest. The target PE is chosen by the hypervisor based on the routing information the GICv2m guest programs into the SPI route register, which is trapped by the hypervisor.

K Support for secure firmware

X_{CXKKF} This section describes an additional set of recommendations for a base system that supports secure firmware. It is written to give a base set of functionality that platform firmware can rely on and decouple the operation of secure firmware and non-secure firmware by not having to arbitrate for shared resources, such as watchdog and timer.

K.1 Memory map

I_{VSMGM} The system should provide some memory mapped in the Secure address space. The memory must not be aliased in the Non-secure address space. The amount of Secure memory provided is platform-specific as the intended use of the memory is for platform-specific firmware.

K.2 Clock and timer subsystem

I_{CZTCQ} The system should also include a Secure wakeup timer in the form of the memory mapped timer described in the Armv8 ARM [3]. This timer must be mapped into the Secure address space, and the timer expiry interrupt must be presented to the GIC as an SPI. This timer is called the Secure system wakeup timer.

I_{XKRWC} The Secure wakeup timer does not require a virtual timer to be implemented. The virtual offset register can Read-As-Zero, where writes to the virtual offset register in CNTCTLBase frame are ignored. The timer is not required to have a CNTEL0Base frame.

I_{DQWWG} Table 60 summarizes which address space the register frames related to the Secure wakeup timer should be mapped onto.

Table 60: Secure wakeup timer register mapping

Register Frame	Address Space
CNTControlBase	Secure
CNTReadBase	Not required
CNTCTLBase	As specified in [3]
CNTBaseN	Secure

I_{VVGTC} CNTCTLBase might be shared among multiple timers, including various Secure and Non-secure timers. This specification does not require this.

I_{JMLPR} GICv3 does not support Secure LPI. Therefore, the Secure system timer interrupt must not be delivered as LPI.

I_{RMRTS} It is recognized that in a large system, a shared resource like the system wakeup timer can create a system bottleneck. This is because access to it must be arbitrated through a system-wide lock. This section requires just a single timer so that standard firmware implementations have a guaranteed timer resource across platforms. We anticipate that large PE systems will implement a more scalable solution like one timer for each PE.

K.3 Watchdog

I_{BSDPS} When Secure firmware is supported, the required behavior of watchdog signal 1 of the Non-secure watchdog is modified and is required to be routed as an SPI to the GIC. It is expected that this SPI be configured as an EL3 interrupt, directly targeting a single PE.

I_{MJDTX} A system should implement a second watchdog, and is referred to as the Secure watchdog. It must have both its register frames mapped in the Secure memory address space and must not be aliased to the Non-secure address space.

I_{WNZKR} Watchdog Signal 0 of the Secure watchdog must be routed as an SPI to the GIC and it is expected this will be configured as an EL3 interrupt, directly targeting a single PE.

I_{CCKWG} GICv3 does not support Secure LPI. The Secure watchdog interrupts must not be delivered as LPI.

I_{QQTBC} Only directly-targeted SPIs are required to wake a PE. Programming the watchdog SPI to be directly targeted ensures delivery of the interrupt independent of PE power states. However, it is possible to use a 1 of N SPI to deliver the interrupt, if one of the target PEs is running. See Section 3.9 for information about SPI waking a PE.

I_{ZPFQP} Watchdog Signal 1 of the Secure watchdog must be routed to the platform. In this context, platform means any entity that is more privileged than the code running at EL3. Examples of the platform component that services Watchdog Signal 1 are a system control processor, or dedicated reset control hardware.

I_{STXZP} The action that is taken on the raising of Watchdog Signal 1 of the Secure watchdog is platform-specific.

L Self-hosted debug for Armv9-A

L.1 Goals

This chapter specifies hardware requirements for Armv9-A architecture revision and higher [3], 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 section. 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.

L.2 Levels of functionality

The requirements are introduced through 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 Architecture if it implements all of the functionality of a given level, at performance levels 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.

L.3 Self-hosted debug capabilities

A suitably-built debug software framework running on hardware compliant with one or more versions 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.

L.4 External debug capabilities

This chapter does not specify any capabilities for external debug. However, it is anticipated that external debug scenarios might use the same hardware functions as self-hosted debug scenarios. 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 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.

These requirements do not specify any mechanism for software agents to arbitrate over the use of hardware functions.

L.5 PE Trace

L.5.1 Background

I_{FFJZR} 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 or Linux perf, with a history of executed instructions, branches, or function calls.

I_{NNSJL} 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.
- The methods of capturing the generated trace.

L.5.2 Embedded Trace Extension

I_{YBSJV} The levels of Embedded Trace Extension (ETE) provide the ability to generate a program trace and to set trace filtering trigger conditions.

L.5.3 ETE Level 1

I_{GVTTQ} ETE Level 1 provides basic program flow tracing capability for all PEs in the system, with tracing provided from all PEs concurrently.

R_{ETE_01} 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 ETE level.

R_{ETE_02} Each PE in the system must be provided with a trace unit compliant with the Embedded Trace Extension (FEAT_ETE) [3].

R_{ETE_03} The trace unit must support the following ETE features:

- Cycle counting with a cycle counter that is at least 12-bits in size.
- 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 two counters.
- The sequencer state machine.
- At least four resource selection pairs.

I_{YZTVQ} If a system provides control over the power to the trace unit, it is recommended that this control provides the means to conserve power when the trace unit is not used. This includes when transitioning between states where trace is used and not used.

R_{ETE_04} All trace units must share the same physical timestamp source.

R_{ETE_05} When TRFCR_EL1 and TRFCR_EL2 are used to select the time source, the selection of CoreSight time and Physical time must select the same time source. This means that there is no difference between CoreSight time and Physical time.

R_{ETE_06} The trace units for all PEs must be able to be enabled concurrently, and generate trace concurrently.

R_{ETE_07} Each PE in the system must implement the Trace Buffer Extension (FEAT_TRBE) [3].

R_{ETE_08} All the TRBE trace buffers must implement Flag Updates, or all TRBE trace buffers must not implement Flag Updates. TRBIDR_EL1.F must be the same for all TRBE trace buffers.

R_{ETE_09} All the TRBE trace buffers must implement the same minimum alignment constraints. TRBIDR_EL1.Align must be the same for all TRBE trace buffers.

R_{ETE_10} An implementation must reserve a PPI for the TRBE interrupt to PE.

I_{QLFWH} It is recommended that the TRBE and system interconnect infrastructure for capturing trace 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.

L.6 System Trace Macrocell

L.6.1 Background

I_{ZVXKW} Arm defines a System Trace Macrocell (STM) Architecture [17] 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_{JGBMQ} Providing an STM implementation in a system provides a memory-mapped programmers model for software instrumentation libraries to target consistently across SoCs.

L.6.2 Level 1

The Trace Generation requirements in Section L.6.2.1 define a central STM unit for use by instrumentation software running on all PEs in a system.

The Trace Capture requirements in Section L.6.2.2 define self-hosted capture of the trace from an STM. This capture method ensures the PE trace and STM trace are captured independently, ensuring trace buffers for PE tracing can be independently managed.

The Trace Generation requirements defined in Section L.6.2.1 are identical to STG Level 1 defined in [18]. The Trace Capture requirements defined in Section L.6.2.2 are identical to STC Level 2 defined in [18].

L.6.2.1 Trace Generation (STG)

R_{STM_01} The system must provide an STM unit compliant with the STMv1.1 architecture as defined in [17].

R_{STM_02} The STM unit must implement the following options:

- STPv2 trace protocol. See [17].
- 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_{STM_03} If any PE implements EL3, the STM unit must implement at least one stimulus port master which is only accessible by Secure software.

R_{STM_04} 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.

L.6.2.2 Trace Capture (STC)

I_{VDMFX} It is recommended that the system provides an independent trace buffer for the STM, based on shared system memory.

R_{STM_05} A trace buffer must be provided in the system to capture trace from the STM.

R_{STM_06} 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 [19].
- Shared system memory, based on a configuration of the Embedded Trace Router (ETR) Architecture, as defined in [20].

R_{STM_07} The trace buffer must be able to be located anywhere in normal memory accessible by the PE.

R_{STM_08} The trace buffer must support the Circular Buffer mode of operation.

R_{STM_09} The trace buffer for the STM must not allow trace from any PE trace unit to be captured.

R_{STM_10} 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. This is required in order 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_{STM_11} Where a page scattering mechanism is required, this must be based on one of the following:

- A System Memory Management Unit (SMMU) based on at least SMMUv1.
- The translation service provided by a CoreSight Address Translation Unit [21].

R_{STM_12} 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 Address Translation Unit [21]. 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 the CoreSight Address Translation Unit, and stage 2 is implemented using an SMMU.

R_{STM_13} 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_{STM_14} 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_{STM_15} 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. It is IMPLEMENTATION DEFINED whether the trace buffer is able to be located in Secure memory.

R_{STM_16} 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 [22].

R_{STM_17} 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_{STM_18} 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.

I_{WVSSC} 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, it is recommended that this control provides the means to conserve power when trace is not used. This includes when transitioning between states where trace is used and not used.

R_{STM_19} 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 [23].

R_{STM_20} 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_{STM_21} 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 [22], and use of the ATB Trigger Enable function in the STM.
- Use of a dedicated mechanism when a trigger occurs in the STM.

R_{STM_22} 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_{STM_23} 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_{STM_24} The trace buffer interrupt must be at least one of an SPI or LPI.

R_{STM_25} 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.

R_{STM_26} 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_{STM_27} The registers to control each trace buffer must be located in separate 64KB pages.

R_{STM_28} 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_{STM_29} 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_{MKFMS} Where the CoreSight Address Translation Unit is used, it is recommended that the CoreSight Address Translation Unit is located in a separate 64KB page from all other trace control components.