Server Base System Architecture

Document number:	ARM-DEN-0029A Version 3.1
Date of Issue:	27th February 2017
Author:	Architecture and Technology Group
Confidentiality:	Non-Confidential

Proprietary Notice

ARM Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

Release Information

The following releases of this document have been made:

Date	Issue	Confidentiality	Change
11 February 2016	Initial	Non-Confidential Restricted access	Initial Release
27 February 2017	A	Non-Confidential	Change of Proprietary Notice Addition of release history No other changes

Contents

1 ABOUT THIS DOCUMENT

1.1 References

This document refers to the following documents:

Reference	Doc No	Title
[1]	ARM DDI 0406	ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition
[2]	ARM IHI 0048	ARM® Architecture Specification, GIC architecture version 2.0
[3]	ARM DDI 0183	ARM® PrimeCell® UART (PL011) Technical Reference Manual
[4]	ARM IHI 0067	ARM® System Memory Management Unit Architecture Specification, 64KB Translation Granule Supplement
[5]	ARM IHI 0062	ARM® System Memory Management Unit Architecture Specification
[6]	ARM DDI 0487	ARM® Architecture Reference Manual ARMv8, for the ARMv8-A architecture profile
[7]	ARM DEN 0044	Server Base Boot Requirements, System Software on ARM® Platforms
[8]	ARM IHI 0069	ARM® Architecture Specification, GIC architecture version 3.0 and version 4.0

1.2 Terms and abbreviations

This document uses the following terms and abbreviations:

Term	Meaning
Base Server System	A system compliant with the Server Base System Architecture
SBSA	Server Base System Architecture
ARM ARM	ARM Architecture Reference Manual; see [1] and [6].
GIC	Generic Interrupt Controller
VM	Virtual Machine
PE	Processing Element, as defined in the *ARM ARM*. Typically a single hardware thread of a PE.
PMU	Performance Monitor Unit
I/O Coherent	A device is I/O Coherent with the PE caches if its transactions snoop the PE caches for cacheable regions of memory. The PE does not snoop the device's cache.
SGI	Software Generated Interrupt
SPI	Shared Peripheral Interrupt

PPI	Private Peripheral Interrupt
LPI	Locality-specific Peripheral Interrupt (GICv3)
SRE	System Register interface Enable (GICv3)
ARE	Affinity Routing Enable (GICv3)
System firmware data	System description data structures such as ACPI or FDT
SBBR	Server Base Boot Requirements

1.3 Feedback

ARM welcomes feedback on its documentation.

1.3.1 Feedback on this manual

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

- The title.
- The document and version number, ARM-DEN-0029 v3.0.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

2 BACKGROUND

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

The ARM architecture profiles, **A**pplication, **R**eal-time, and **M**icrocontroller, exist in part to segment the solutions produced by ARM and to describe the characteristics of particular target markets. The differences between products targeted at different profiles are substantial due to 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, where 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, such as those which require an open platform with complex software, the savings gained from removing a hardware feature are outweighed by the cost of software development to support the different variants. In addition, software development is often performed by third parties, and the uncertainty about whether new features are widely deployed can be a substantial brake to the adoption of those features.

The ARM Application profile must balance these two competing business pressures. It offers a wide range of features, such as Advanced SIMD and floating point support, and TrustZone system security technology, to tackle an increasing range of problems, while allowing features to be removed from implementations where they are not needed and where silicon area and cost savings are an issue.

ARM processors are built into a large variety of systems. Aspects of this system functionality are crucial to the fundamental function of system software.

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

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

3 INTRODUCTION

This document specifies a hardware system architecture, based on ARM 64-bit architecture, which server system software, such as operating systems, hypervisors and firmware can rely on. It addresses PE features and key aspects of system architecture.

The primary goal is to ensure enough standard system architecture to enable a suitably-built single OS image to run on all hardware compliant with this specification. A driver-based model for advanced platform capabilities beyond basic system configuration and boot is required, however that is outside the scope of this document. Fully discoverable and describable peripherals aid the implementation of such a driver model.

This specification also specifies features that firmware can rely on, allowing for some commonality in firmware implementation across platforms.

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

This specification embeds the notion of levels of functionality. Level 0 is the first level; level 1 adds functionality on top of level 0. Unless explicitly stated, all specification items belonging to level N apply to levels greater than N.

An implementation is consistent with a level of the Server Base System Architecture if it implements all of the functionality of that level at performance levels appropriate for the target uses of that level. This means that all functionality of a level can be exploited by software without unexpectedly poor performance.

Note: This is intended to avoid approaches such as software emulation of functionality that is critical to the performance of software using the SBSA. 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.

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

Software running on a system including an ARM core inevitably includes code that is system-specific. Such 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 such constructs as *Hardware Specific Software*. The ARM Server Base Boot Requirements (SBBR) specification [7] describes firmware requirements for an ARM server system. Where this specification refers to system firmware data, it refers to firmware specified in the SBBR.

This specification uses the phrase *software consistent with the Server Base System Architecture* to indicate software that is designed to be portable between different implementations that are consistent with the Server Base System Architecture. Software that is consistent with the Server Base System Architecture does not depend on the presence of hardware features that are not mandated in this specification. However, software might use

features that are not included in this specification, after checking that the platform supports the features, for example by using hardware ID registers or system firmware data.

4 SBSA

4.1 Level 0

4.1.1 PE Architecture

The PEs referred to in this specification are those that are running the operating system or hypervisor, not PEs that are acting as devices.

PEs in the base server system are compliant with ARMv8 and the following is true:

- The number of PEs in the system does not exceed eight.

  Note: This restriction is due to GICv2 limitations and will be removed in a future level.
- PEs implement Advanced SIMD extensions.
- Whether the Instruction Caches are implemented as VIPT or PIPT is IMPLEMENTATION DEFINED.

  Note: Not all PEs are required to support the same Instruction Cache addressing scheme.
- PEs shall implement 16-bit ASID support.
- PEs shall support 4KB and 64KB translation granules at stage 1 and stage 2.
- All PEs are coherent and in the same Inner Shareable domain.
- Where export restrictions allow, PEs should implement cryptography extensions.
- PEs shall implement little-endian support.
- PEs shall implement EL2.
- PEs shall implement AArch64 at all Exception levels.
- The PMU overflow signal from each PE must be wired to a unique PPI or SPI interrupt with no intervening logic.
- Each PE implements a minimum of four programmable PMU counters.
- Each PE implements a minimum of four synchronous watchpoints.
- Each PE implements a minimum of four breakpoints, two of which must be able to match virtual address, contextID or VMID.
- All PEs are architecturally symmetric except for the permitted exceptions laid out in APPENDIX C: *Permitted Architectural Difference between PEs*.

Note: It is consistent with this specification to implement PEs with EL3 and with support for the AArch32 Execution state.

4.1.2 Interrupt Controller

The base server system shall implement a GICv2 interrupt controller.

The system shall implement at least eight Non-secure Software Generated Interrupts, assigned to interrupt IDs 0-7.

4.1.3 Memory Map

This specification does not mandate a standard memory map. It is expected that the system memory map is described to system software by system firmware data.

To enable EL2 hypervisors to use a 64KB translation granule at stage 2 MMU translation, the base server system shall ensure that all memory and peripherals can be mapped using 64KB stage 2 pages and must not require the

use of 4KB pages at stage 2. It is expected therefore that peripherals that are to be assigned to different virtual machines will be situated within different 64KB regions of memory.

Systems will not necessarily fully populate all of the addressable memory space. All memory accesses, whether they access memory space that is populated or not, shall respond within finite time, so as to avoid the possibility of system deadlock. 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 a precise Data Abort or causes a system error interrupt, or causes an SPI interrupt to be delivered to the GIC.

Note: 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.

Note: The ARM implementation of GICv2, the GIC-400 product, needs special address bus wiring in a 64KB translation granule system. See Appendix F: GIC-400 and 64KB Translation Granule.

4.1.4 I/O Virtualization

It is implementation-specific whether any given device in a base server architecture system supports the ability to be hardware virtualized. It is expected that devices that can be hardware virtualized have that property expressed by system firmware data.

If a device is virtualized 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. This specification collectively refers to this translation, attribution, and permission checking as *policing*. The act of policing is referred to as stage 2 System MMU functionality.

This stage 2 System MMU functionality must be provided by a System MMU compatible with the ARM SMMUv1 with support for a 64KB translation granule specification, where:

- Support for stage1 policing is not required.

Note: Support for broadcast TLB maintenance operations is not required, and compliant software must maintain the MMU TLBs using the software interface.

SMMUv1 with support for a 64KB translation granule does not have support for PCI Express ATS; support for PCI Express ATS will be system-specific.

The base server system might instance an IMPLEMENTATION DEFINED number of SMMU components. It is expected that these components will be described by system firmware data along with a description of how to associate them with the devices they police.

Note: This is consistent with the ARM MMU-401 implementation. Software can either program stage 2 System MMUs to use the same page tables as the PE or build shadow page tables. Standard PCI Express ATS support, which is included in SMMUv3, is introduced in a later level of this specification.

4.1.5 Clock and Timer Subsystem

The base server system shall include the system counter of the Generic Timer as specified in the ARM ARM.

The system counter of the Generic Timer shall run at a minimum frequency of 10MHz and at a maximum frequency of 400MHz.

The architecture of the counter mandates that it shall be at least 56 bits, and at most 64 bits.

Note: The counter shall be sized and programmed to ensure that rollover never occurs in practical situations. The timers and watchdogs that use the counter as a timebase rely on the counter not rolling over.

The Generic Timer system counter also exports its count value, or an equivalent encoded value, through the system to the timers in the PEs as part of the Generic Timer subsystem. This count must be available to the PE Timers when they are active, which is when the PEs are in power states where the PE timer is required to be on.

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

The local PE timers can be built so that they are *always on*. This property is described in the system firmware data. Unless all of the local PE timers are always on, the base server system shall implement a system-specific system wakeup timer that can be used when PE Timers are powered down. On timer expiry, the system wakeup timer shall generate a level interrupt that shall be wired to the GIC as an SPI. Additionally, the system wakeup timer can be used to wake up PEs. See 4.1.6.

4.1.6 Wakeup semantics

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 it has for waking PEs from various low power states.

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 may 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.

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.

In addition, for an interrupt to be a wakeup interrupt, it shall be enabled in the distributor. A PE shall wake in response to a wakeup interrupt, independent of the state of its CPSR interrupt mask bits, which are the A, I, and F bits, and of the wakeup interrupt priority.

Note: Typically, a wakeup signal is exported from the GIC to the power controller to initiate the PE wakeup.

Note: 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.

The local PE timers are an important source of interrupts and can wake the PE. However the local PE timer may be powered down in some low-power states, as it 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 it 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, there is another class of wakeup methods that can be used; always-on power domain wake events.

If the system supports a low-power state where the GIC is powered down, there is an IMPLEMENTATION DEFINED way to program the power controller to wake a PE on expiry of the system wakeup timer. In this scenario, the system wakeup timer is still required to send its interrupt.

There may be other IMPLEMENTATION DEFINED always-on power domain wakeup events that can wake PEs from deep low-power states, such as PCI Express wakeup events and Wake-on-LAN.
See 4.1.7 for a description of the power state semantics that the system must comply with.

4.1.7 Power State Semantics

This specification does not mandate a given hierarchy of power domains, but there are some rules and semantics that must be followed.

Figure 1 is 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.

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 1: PE Power States and Table 2: Power State Semantics. Note that all PEs do not need to be in the same state. It is expected that the semantics of the power states that a system supports will be described by system firmware data. Table 3: Component Power State Semantics describes the power state semantics in a set of component-specific rules.

System MMUs and, in the future, GICv3, make use of tables in memory in the power states where GIC is ‘On’, system memory shall be available and will respond to requests without requiring intervention from software running on the PEs.

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

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. 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 may also cause wakeup from this state, such as a PCI Express wakeup event. The events that will cause wakeup from this state are expected to be discoverable from system firmware data.

When the system is in a state where the GIC is powered down devices must not send messaged interrupts to the GIC.

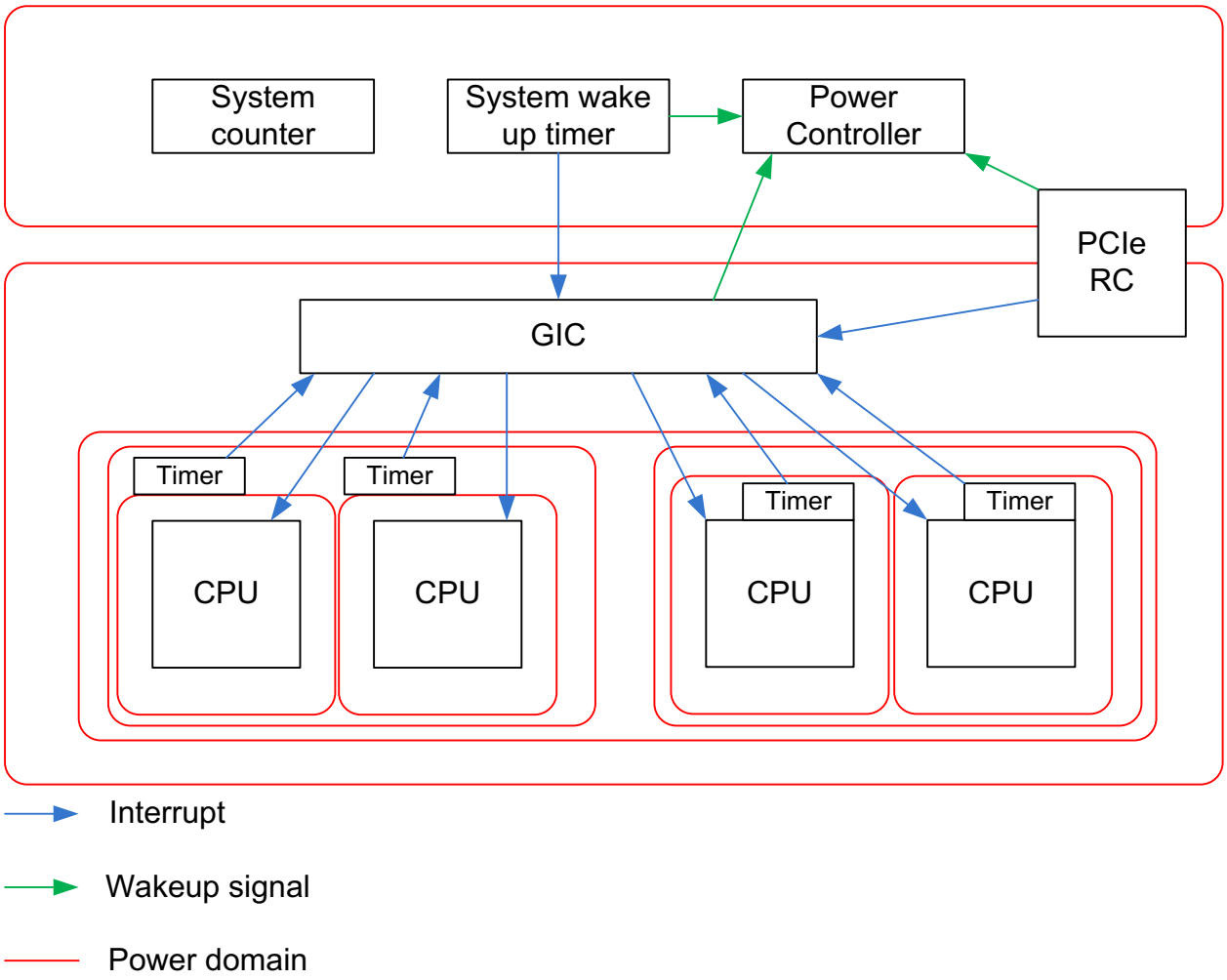

Figure 1: Example system block diagram showing power domains and timer hierarchy

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.

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 such as a poweron_reset. This state can be used when the system software explicitly decides to remove the PE from active service, giving the hardware opportunity for more aggressive power saving. No PE state is retained.

Table 1: PE Power States

Semantic	PE	PE timers	GIC	System wakeup timers and system counter	Note
A	Run	On	On	On	-
B	Idle	On	On	On	The PE will resume execution on receipt of any interrupt.
C	Sleep	On	On	On	The PE will wake on receipt of a wakeup interrupt.
D	Sleep	Off	On	On	The PE will wake on receipt of a wakeup interrupt, but the local timer is off.
E	Sleep	Off	Off	On	The PE will wake as the result of a 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 are in Off state.
H	Sleep	On	Off	On	The PE will wake from as the result of a PE timer event, a system timer wakeup event, or other system-specific events.
I	Idle	Off	On	On	The PE will resume execution on receipt of any interrupt, but the local timer is off.

Table 2: Power State Semantics

Component	Semantics
PE	Individual PEs 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	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 wake up timers and system	Must be On if any PE is not in the Off state. Must be Off if all PEs are in the Off state.

counter	

Table 3: Component Power State Semantics

4.1.8 Peripheral Subsystems

If the system has a USB2.0 host controller peripheral it must conform to EHCI v1.1 or later.

If the system has a USB3.0 host controller peripheral it must conform to XHCI v1.0 or later.

If the system has a SATA host controller peripheral it must conform to AHCI v1.3 or later.

Peripheral subsystems which do not conform to the above are permitted, provided that they are not required to boot and install an OS.

4.2 Level 1

4.2.1 PE Architecture

In addition to the level 0 requirements, the following must be true of the PEs in the base server system:

- Each PE must implement a minimum of six programmable PMU counters.
- Each PE must implement a minimum of six breakpoints, two of which must be able to match virtual address, contextID or VMID.

4.2.2 Interrupt Controller

If the base server system includes PCI Express then the base server system must implement a GICv2m interrupt controller. The system must implement at least one Non-secure MSI frame with a minimum of 32 SPIs. There is no requirement to support a Secure MSI frame.

Note that a GICv2m interrupt controller is a GICv2 interrupt controller with additional register frames specified in the GICv2m specification, see APPENDIX E: GICv2m Architecture, for standardized support of PCI Express MSI and MSI-X.

If the base server system does not include a PCI Express root complex then the base server system must implement a GICv2 interrupt controller.

4.2.3 Clock and Timer Subsystem

Level 0 of the SBSA requires a system-specific system timer unless all of the local PE timers are always on. Level 1 of the SBSA supersedes this requirement and requires that unless all of the local PE timers are always on that there is a system wakeup timer in the form of the memory mapped timer described in the ARMv8 ARM [6].

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.

In systems that implement EL3, the memory mapped timer (the CNTBaseN frame and associated CNTCTLBase frame) must be mapped into the Non-secure address space.

Table 4 : Generic counter and timer memory mappings shows where the various counter and timer frames are mapped in systems with and without EL3.

Register Frame	System without EL3	System with EL3
CNTControlBase	Non-secure	Secure
CNTReadBase	Non-secure	Either
CNTCTLBase	Non-secure	Non-secure and Secure
CNTBaseN	Non-secure	Non-secure

Table 4 : Generic counter and timer memory mappings

4.2.3.1 Summary of the required registers of the CNTControlBase frame

Offset	Name	Type	Description
0x000	CNTCR	RW	Counter Control Register
0x004	CNTSR	RO	Counter Status Register
0x008	CNTCV[31:0]	RW	Counter Count Value Register
0x00C	CNTCV[63:32]	RW	Counter Count Value Register
0x010-0x01C	-	RES0	Reserved
0x020	CNTFID0	RO OR RW	Frequency modes table, and end marker.
0x020+4n	CNTFID*n*	RO OR RW	CNTFID0 is the base frequency, and each CNTFID*n* is an alternative frequency. For more information see ARM ARM.
(0x024+4n)-0x0BC	-	RES0	Reserved
0x0C0-0X0FC	-	IMPLEMENTATION DEFINED	Reserved for IMPLEMENTATION DEFINED registers
0x100-0xFCC	-	RES0	Reserved
0xFD0-0xFFC	CounterID<n>	RO	Counter ID registers 0-11

4.2.3.2 Summary of the required registers of the CNTReadBase frame

Offset	Name	Type	Description
0x000	CNTCV[31:0]	RO	Counter Count Value Register
0x004	CNTCV[63:32]	RO	Counter Count Value Register
0x008-0xFCC	-	RES0	Reserved
0xFD0-0xFFC	CounterID<n>	RO	Counter ID registers 0-11

4.2.3.3 Summary of the required registers of the CNTCTLBase frame

Offset	Name	Type	Security	Description
0x000	CNTFRQ	RW	Secure	Counter Frequency register
0x004	CNTNSAR	RW	Secure	Counter Non-secure Access register
0x008	CNTTIDR	RO	Both	Counter Timer ID register
0x00C-0x03F	-	RES0	Both	Reserved
0x040+4N	CNTACR<N>	RW	Configurable	Counter Access Control register N
0x060-0x07F	-	RES0	Both	Reserved
0x0C0-0x0FC	-	UNK/SBZP	Both	Reserved
0x100-0x7FC	-	-	Both	IMPLEMENTATION DEFINED
0x800-0xFBC	-	UNK/SBZP	Both	Reserved
0xFC0-0xFCF	-	-	Both	IMPLEMENTATION DEFINED
0xFD0-0xFFC	CounterID<n>	RO	Both	Counter ID registers 0-11

4.2.3.4 Summary of the required registers of the CNTBaseN frame

Offset	Name	Type	Description
0x000	CNTPCT[31:0]	RO	Physical Count register
0x004	CNTPCT[63:32]	RO	Physical Count register
0x010	CNTFRQ	RO	Counter Frequency register
0x020	CNTP_CVAL[31:0]	RW	Physical Timer Compare Value register
0x024	CNTP_CVAL[63:32]	RW	Physical Timer Compare Value register
0x028	CNTP_TVAL	RW	Physical Timer Value register
0x02C	CNTP_CTL	RW	Physical Timer Control register
0x040-0xFCF	-	RES0	Reserved
0xFD0-0xFFC	CounterID<n>	RO	Counter ID registers 0-11

4.2.4 Watchdogs

The base server system implements a Generic Watchdog as specified in APPENDIX A: Generic Watchdog. Watchdog Signal 0 is routed as an SPI to the GIC and it is expected this will be configured as an EL2 interrupt, directly targeting a single PE.

Watchdog Signal 1 shall be routed to the platform. In this context, *platform* means any entity that is more privileged than the code running at 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.

The action taken on the raising of Watchdog Signal 1 is platform-specific.

Note: Only directly-targeted SPI are required to wake a PE; see section 4.1.6 for further 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 as long as one of the target PEs is running.

4.2.5 Wakeup semantics

If the system supports a low-power state where the GIC is powered down, then there shall 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.

4.2.6 Requirements on power state semantics

The power state semantic table and the Component Power State Semantic table are extended to include the Generic Watchdog.

Semantic	PE	PE timers	GIC	System wake up timers, system counter and generic watchdog	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.

Table 5: Power State Semantics

PE	Individual PEs can be in Run, Idle, Sleep or Off state.
PE timers	Must be On if the associated PE is in the Run state. May 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	Must be On if any PE is in the Run or Idle state. Maybe 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 wake up 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.

Table 6: Component Power State Semantics

4.2.7 Peripheral Subsystems

For the purpose of system development and bring up, the base server system shall include a Generic UART. The Generic UART is specified in Appendix B. The UARTINTR interrupt output is connected to the GIC as an SPI.

If the system has a PCI Express root complex then it must comply with the rules in APPENDIX D: PCI Express Integration.

4.3 Level 2

4.3.1 PE Architecture

The maximum number of PEs is raised to 2^{28}. This reflects the maximum number of PEs GICv3 can support.

1. The PMU overflow signal from each PE must be wired to a unique PPI interrupt with no intervening logic.

4.3.2 Interrupt Controller

The GICv3 specification introduces support for systems with more than eight PEs, as well as improved support for larger numbers of interrupts.

A level 2 base server system shall implement a GICv3 interrupt controller.

If the base server system includes PCI Express then the GICv3 interrupt controller shall implement ITS and LPI.

Note: It is expected that MSI and MIS-X are mapped to LPI interrupts.

Note: It is permissible to build a system with no support for SPI, however ARM expects that the peripheral eco-system will continue to rely on wired level interrupts and expects most systems to support SPI as well as LPI interrupts.

Note: The ARM PL011 UART requires a level interrupt, as does a PCIe root complex, for legacy interrupt support.

It is optional for the interrupt controller to include GICv2 and GICv2m backward compatibility; ARE may be implemented as RAO/WI for both Security states and SRE as RAO/WI for all Exception levels. Any system that does not include this compatibility will not be able to run software that is compliant with level 0 or level 1.

See Appendix G: GICv2m compatibility in a GICv3 system on page 52 for how backward compatibility with GICv2m can be achieved in a GICv3 system.

Support for compatibility with level 0 and level 1 of this spec (and hence GICv2 and GICv2m) is deprecated.

Note: A level 2 system running in backwards-compatible mode is only able to use a maximum of 8 PEs.

4.3.2.1 PPI assignments

A level 2 base server system must comply with the PPI mapping laid out in: Table 7 - PPI assignments.

Interrupt ID	Interrupt	Description
30	Overflow interrupt from CNTP	Non-secure physical timer interrupt.
29	Overflow interrupt from CNTPS	Secure Physical timer interrupt.
28	Overflow interrupt from CNTHV	EL2 virtual timer (if PEs are v8.1 or greater)
27	Overflow interrupt from CNTV	Virtual timer interrupt.
26	Overflow interrupt from CNTHP	Hypervisor timer interrupt.
25	GIC Maintenance interrupt	The virtual PE interface list register overflow interrupt.
24	CTIIRQ	CTI (Cross Trigger Interface) interrupt.
23	Performance Monitors Interrupt	Indicates an overflow condition in the performance monitors unit.
22	COMMIRQ	DCC (comms channel) interrupt.
21	PMBIRQ	Statistical Profiling Interrupt (if Statistical Profiling Extensions implemented)
19-20	Reserved	Expansion space for future SBSA usage

Table 7 - PPI assignments

4.3.3 Memory Map

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 that causes a system error interrupt or an SPI or LPI interrupt to be delivered to the GIC.

4.3.4 Requirements on power state semantics

GICv3 introduces a new class of interrupt: LPI. A PE receiving any of the following types of interrupt shall wake up:

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

The power state semantic table and the component power state semantic table are modified to split the GIC requirements into those for the PE Interface and those for the distributor.

Semantic	PE and GIC PE Interface	PE timers	GIC Distributor	System wakeup timers, system counter and generic watchdog	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.

Table 8: 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. May 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. Maybe 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.

Table 9: Component Power State Semantics

4.3.5 I/O Virtualization

Stage 2 System MMU functionality must be provided by a System MMU compatible with the ARM SMMUv2 spec where:

- Support for stage 1 policing is not required.

Note: Support for broadcast TLB maintenance operations is not required.

Note: This behavior is consistent with ARMs MMU-500 implementation.

4.3.6 Clock and Timer Subsystem

In systems that include the system wakeup time, the timer expiry interrupt is presented to the GIC as either an SPI or LPI.

It is recognized that in large system a shared resource like the system wakeup timer can create a system bottleneck, as access to it must be arbitrated through a system-wide lock. It is anticipated that this will be dealt with by the platform by having the firmware tables describe the PE timers as always on and remove the need for the system timer. The platform will either implement hardware always on PE timers or use the platform firmware to save and restore the PE timers in a performance scalable fashion.

Note: Systems compliant with Level 3 – firmware will have standard hardware that the firmware can use, see section 4.5.2.

4.3.7 Wakeup semantics

Whenever a PE is woken from a sleep or off state the OS or Hypervisor shall be presented with an interrupt so that it can determine which device requested the wakeup. 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.

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.

Note: 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.

An example of an expected chain of events would be:

1. Wakeup event occurs e.g. 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 system-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 system-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.

4.3.8 Watchdogs

The watchdog signal WS0 shall be presented to the GIC as SPI or an LPI interrupt.

4.3.9 Peripheral Systems

The UARTINTR output of the generic UART as described in Appendix B shall be connected to the GIC as an SPI or LPI.

4.4 Level 3

4.4.1 PE Architecture

In addition to the level 2 requirements, the following shall be true of the PEs in the base server system:

- Each PE shall implement the EL3 Exception level.
- PEs shall implement the CRC32 instructions.

4.4.2 Expected usage of Secure state

The Level 3 base server system is expected to use the PE EL3 and Secure state as a place to implement platform-specific firmware. The system may choose to implement further functionality in the Secure state, but that is outside the scope of SBSA Level 3.

Given this expected use SBSA Level 3 does not expect PCI express to be present in the Secure state; this assumption is reflected in the GICv3 architecture in that there is no support for Secure LPI.

4.4.3 Memory Map

All Non-secure on-chip masters in a base server 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. If the master goes through a SMMU then the master must be capable of addressing all of the Non-secure address space when the SMMU is turned off.

Non-secure off-chip devices that cannot directly address all of the Non-secure address space must be placed behind a stage 1 System MMU 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 4.4.5.

4.4.4 Interrupt Controller

The GICv3 interrupt controller shall support two Security states.

4.4.5 I/O Virtualization

Stage 2 System MMU functionality must be provided by a System MMU compatible with the ARM SMMUv2 specification where:

- Support for stage 1 policing is not required.
- Each context bank must present a unique physical interrupt to the GIC.

Or the Stage 2 System MMU functionality must be provided by a System MMU compatible with the ARM SMMUv3 spec where:

- Support for stage 1 policing is not required.
- The integration of the System MMUs is compliant with the specification in APPENDIX H: SMMUv3 Integration.

All the System MMUs in the system must the compliant with the same architecture version.

Note: System MMUv3 is not backwards compatible with System MMUv2 and as such any system implementing System MMUv3 MMUs is not strictly backwards compatible with level 2.

Note: Support for broadcast TLB maintenance operations is not required.

4.4.6 Clock and Timer Subsystem

If the system includes a system wakeup timer, this memory-mapped timer must be mapped on to Non-secure address space. This is now referred to as the *Non-secure system wakeup timer*. The following table summarizes which address space the register frames should be mapped on to.

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

4.4.7 Watchdogs

The watchdog required by level 2 must have both its register frames mapped on to Non-secure address space; this is referred to as the *Non-secure watchdog*.

Note: Only directly targeted SPI are required to wake a PE so programming the watchdog SPI to be directly targeted ensures delivery of the interrupt independent of PE power states. See section 4.2.6. However, it is possible to use a 1 of N SPI to deliver the interrupt as long as one of the target PEs is running.

4.4.8 Peripheral Subsystems

The Generic UART required by level 2 must be mapped on to Non-secure address space. This is referred to as the Non-secure Generic UART.

For systems that include PCI express, the PCI express integration appendix introduces an additional rule applicable to a level 3 system. See section 8.3, PCI Express device view of memory.

The memory attributes of DMA traffic must be one of the following:

- Inner writeback, outer writeback, Inner Shareable.
- Inner non-cacheable, outer non-cacheable.
- A device type.

I/O Coherent DMA traffic must have the attribute “Inner writeback, outer writeback, Inner Shareable”.

4.5 Level 3 – firmware

Level 3 – firmware is an optional additional set of requirements for a level 3 system. It is designed to give a base set of functionality that standard platform firmware can rely on. A system that is compliant with level 3 and not compliant with level 3 – firmware is still a fully compliant level 3 system. It has all the features required by the operating systems and hypervisors.

4.5.1 Memory Map

The system must provide some memory mapped in the Secure address space. The memory shall 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.

All Non-secure on-chip masters in a base server system that are expected to be used by the platform firmware must be capable of addressing all of the Non-secure address space. If the master goes through a SMMU then the master must be capable of addressing all of the Non-secure address space even when the SMMU is off.

4.5.2 Clock and Timer Subsystem

A system compatible with level 3- firmware must also include a Secure wakeup timer in the form of the memory mapped timer described in the ARMv8 ARM [6] This timer must be mapped into the Secure address space, and the timer expiry interrupt shall be presented to the GIC as an SPI. This timer is referred to as the Secure system wakeup timer.

The Secure wakeup timer does not require a virtual timer to be implemented and it is permissible for the virtual offset register to 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.

The following table summarizes which address space the register frames related to the Secure wakeup timer should be mapped on to.

Register Frame	
CNTControlBase	Secure
CNTReadBase	Not required
CNTCTLBase	Secure
CNTBaseN	Secure

CNTCTLBase may be shared amongst multiple timers, including various Secure and Non-secure timers. The SBSA specification does not require this.

Note: GICv3 does not support Secure LPI; therefore the Secure system timer interrupt shall not be delivered as LPI.

Note: It is recognized that in a large system, a shared resource like the system wakeup timer can create a system bottleneck, as access to it must be arbitrated through a system-wide lock. Level 3-firmware requires just a single timer so that standard firmware implementations have a guaranteed timer resource across platforms. It is anticipated that large PE systems will implement a more scalable solution such as one timer per PE.

4.5.3 Watchdogs

The required behavior of watchdog signal 1 of the Non-secure watchdog is modified in level 3– firmware 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.

A system compatible with level 3- firmware must 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.

Watchdog Signal 0 of the Secure watchdog shall 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.

Note: GICv3 does not support Secure LPI. The Secure watchdog interrupts shall not be delivered as LPI.

Note: Only directly targeted SPI 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 provided that one of the target PEs is running. See section 4.2.6 for information about SPI waking a PE.

Watchdog Signal 1 of the Secure watchdog shall 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.

The action taken on the raising of Watchdog Signal 1 of the Secure watchdog is platform-specific.

4.5.4 Peripheral Subsystems

A system compatible with level 3-firmware must provide a second generic UART, referred to as the Secure Generic UART, that can be configured to exist in the Secure memory address space. It must not be aliased in the Non-secure address space. The UARTINTR output of the Secure Generic UART shall be connected to the GIC as an SPI.

Note: GICv3 does not support Secure LPI. The Secure Generic UART interrupt shall not be delivered as LPI.

Systems that integrate PCI express should note that PCI express integration appendix introduces an additional rule applicable to a level 3 system, see 8.3 PCI Express device view of memory.

5 APPENDIX A: GENERIC WATCHDOG

5.1 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.

5.2 Watchdog Operation

The Generic Watchdog has the concept of a Cold reset and a Warm reset. On a Cold reset, certain register values are reset to a known state. Watchdog Cold reset must only occur as part of the watchdog powering-up sequence. On a Warm reset, the architectural state of the watchdog is not reset, but other logic such as the bus interface might be. This is to facilitate the PEs in the system going through a reset sequence, while the watchdog retains its state so it can be examined when the PEs are running.

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 referred to as 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.

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 following must occur:

- If the system is compliant to SBSA level 0 or level 1, then it is IMPLEMENTATION DEFINED whether the compare value is loaded with the sum of the zero-extended watchdog offset register and the current generic timer system count value, or whether it retains its current value.
- If the system is compliant to SBSA level 2 or higher, 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 and a timeout refresh occurs, WS1 is asserted.

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

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.

Note: The following pseudocode assumes that the compare value is not updated on a timeout refresh when WS0 == 1 and does not show the other permitted 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[31:0])
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 shall be disabled when the System Counter is being updated, or the results are UNPREDICTABLE.

Note: The watchdog offset register is 32 bits wide. This gives a maximum watch period of around 10s at a system counter frequency of 400MHz. If a larger watch period is required, the compare value can be programmed directly into the compare value register.

5.3 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 then 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 10 shows the 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_IIDR	See Watchdog Interface Identification Register on page 35.
0xFD0 – 0xFFF	-	IMPLEMENTATION DEFINED.

Table 10 Refresh Frame

Table 11 shows the 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	Watchdog offset register. A read/write register containing the unsigned 32 bit watchdog countdown timer value.
0x00C – 0x00F	-	Reserved.
0x010 – 0x013	WCV[31:0]	Watchdog compare value. Read/write registers containing the current value in the watchdog compare register.
0x014 – 0x017	WCV[63:32]	
0x018 – 0xFCB	-	Reserved.
0xFCC – 0xFCF	W_IIDR	See Watchdog Interface Identification Register on page 35.
0xFD0 – 0xFFF	-	IMPLEMENTATION DEFINED.

Table 11 Watchdog Control Frame

5.4 Register descriptions

5.4.1 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.

5.4.2 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:

- 0x0 for Generic Watchdog 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 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.

6 APPENDIX B: GENERIC UART

6.1 About

This specification of the ARM generic UART is designed to offer a basic facility for software bring up and as such specifies the registers and behavior required for system software to use the UART to receive and transmit data. This specification does not cover registers needed to configure the UART as these 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 specified 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.

The generic UART supports at least 32-entry separate transmit and receive byte FIFOs and 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 12 on page 38 identifies the minimum register set used for SW management of the UART.

6.2 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 12 Base UART Register Set. 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.

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)	Read: 16,32 Write: 8,16,32
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)	Read: 8, 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 used indicate state of UART and FIFOs, with operation as PL011. Bits [15:8] Reserved. (Ref Section 3.3.3 – PL011TRM)	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)	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)	Read: 16, 32

0x038 – 0x03b	UARTIMSC – Interrupt 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. (Ref Section 3.3.10 – PL011TRM)	Read: 16, 32 Write: 16, 32
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)	Write: 16, 32

Table 12 Base UART Register Set

6.3 Interrupts

The UARTINTR interrupt output shall be connected to the GIC.

6.4 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
```

6.5 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 [3]. Operations of the IrDA SIR, modem, hardware flow control, and DMA are not supported.

7 APPENDIX C: PERMITTED ARCHITECTURAL DIFFERENCE BETWEEN PES

Table 13 shows the permitted differences in architected registers between PEs in a single base server system. The permitted differences column lists the bit fields for a register that may vary from PE to PE. Where a bit field is not listed, the value must be the same across all PEs in the system.

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 may vary.
Cache level ID register	CLIDR_EL1	All bits, each PE can have a unique cache hierarchy.
Cache Size ID Register	CCSIDR_EL1	Sets [27:13], Data cache associativity [12:3]. Caches on different PEs can be different sizes.
Auxiliary Control Register	ACTLR_EL{1,2,3}	Specific to implementation, all bits may vary.

Auxiliary Fault Status Registers	AFSR{0,1}_EL{1,2,3}	Specific to implementation, all bits may vary.

Table 13 Permitted architectural differences

8 APPENDIX D: PCI EXPRESS INTEGRATION

8.1 Configuration space

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 3.0*.

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 14.

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.

Table 14 Enhanced Configuration Address Mapping

The system may 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.

It is system-specific whether a system supports non-PE agents accessing ECAM regions.

Note: 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.

8.2 PCI Express Memory Space

It is system-specific whether a system supports mapping PCI Express memory space as cacheable.

All systems must support mapping PCI Express memory space as either device memory or non-cacheable memory. When PCI Express memory space is mapped as normal memory, the system must support unaligned accesses to that region.

8.3 PCI Express device view of memory

Transactions from a PCI express device will either directly address the memory system of the base server system or be presented to a SMMU for optional address translation and permission policing.

In systems that are compatible with level 3 or above of the SBSA, the addresses sent by PCI express devices must be presented to the memory system or SMMU unmodified. In a system where the PCI express does not use an SMMU, the PCI express devices have the same view of physical memory as the PEs. In a system with a SMMU for PCI express there are no transformations to addresses being sent by PCI express devices before they are presented as an input address to the SMMU.

8.4 Message signaled interrupts

Support for Message Signaled Interrupts (MSI/MSI-X) is required for PCI Express devices. MSI and MSI-X are edge-triggered interrupts that are delivered as a memory write transaction.

The system shall implement an interrupt controller compliant with the ARM Generic Interrupt Controller, each SBSA level specifies which version should be used.

Note: ARM introduced standard support for MSI(-X) in the GICv2m architecture, this support is extended in GICv3.

The intended use model is that each unique MSI(-X) shall trigger an interrupt with a unique ID and the MSI(-X) shall target GIC registers requiring no hardware specific software to service the interrupt.

8.4.1 GICv2m support for MSI(-X)

GICv2m has the MSI_SETSPI_NS register to support MSI(-X). If I/O virtualization of PCI Express drivers is to be supported in a GICv2m system, multiple versions of the register can exist in different memory pages, allowing different virtual machines to see different registers. Each register targets a unique set of SPIs.

8.4.2 GICv3 support for MSI(-X)

GICv2m is limited in scalability not only in terms of PE count, but also in the number of MSI(-X) that are supported. The architectural maximum number of SPI is 988.

GICv3 adds a new class of interrupt, LPI, to address this. 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 retargeting of interrupts.

8.5 Legacy interrupts

PCI Express legacy Interrupt messages must be converted to an SPI:

- Each of the 4 legacy interrupt lines must be allocated a unique SPI ID.
- The exact SPI IDs that are allocated are IMPLEMENTATION DEFINED.
- Each legacy interrupt SPI must be programmed as level-sensitive in the appropriate GIC_ICFGR.

IMPLEMENTATION DEFINED registers must not be used to deliver these messages, only registers defined in the PCI Express specification and the ARM GIC specification.

8.6 I/O Virtualization

Hardware support for I/O Virtualization is optional, but if required shall use a System MMU compliant with the ARM System MMU specification.

Each function, or virtual function, that requires hardware I/O virtualization is associated with a SMMU context. The programming of this association is IMPLEMENTATION DEFINED and is expected to be described by system firmware data.

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

Note: The Server Base System Architecture requires certain versions of the SMMU to be used at particular levels of the specification.

8.7 I/O Coherency

PCI Express transactions not marked as No_snoop accessing memory that the PE translation tables attribute as cacheable and shared are I/O Coherent with the PEs.

The PCI Express root complex is in the same Inner Shareable domain as the PEs.

I/O Coherency fundamentally means that no software coherency management is required on the PEs for the PCI Express root complex, and therefore devices, to get a coherent view of the PE memory.

This means that if a PCI Express device is accessing cached memory then the transactions from the PCI Express devices will snoop the PE caches.

PCI Express also allows PCI Express devices to mark transactions as No_snoop. The memory accessed by such transactions must have coherency managed by software.

The following summarize the attributes the transactions from the PCI Express root complex must have and how coherency is maintained.

8.7.1 PCI Express I/O Coherency without System MMU

In the case where there is not a System MMU translating transactions from the root complex, the system must be able to distinguish between addresses that are targeted at memory and devices. Transactions that are targeted at devices must be treated as device type accesses. They must be ordered, must not merge and must not allocate in caches. Transactions that are targeted at memory and that are marked No Snoop must be presented to the memory system as non-cached. Transactions that are targeted at memory and not marked as No_snoop must be presented as cached, shared.

The following table shows how coherency is managed for PCI Express transactions. If a memory page is marked as non-cached in the PE translation tables, all PCI Express transactions accessing that memory must be marked as No_snoop. Failure to do so may result in loss of coherency.

PE page table attribute	PCI Express transaction type	PCI Express transaction memory attributes	Coherency management
Cacheable, shared	Snoop	Cacheable, shared	Hardware
	No_snoop	Non-cached	Software
Cacheable, non-shared	Snoop	Cacheable, shared	Software
	No_snoop	Non-cached	Software
Non-cached	Snoop	Not allowed	Not allowed
	No_snoop	Non-cached	Hardware

Table 15 : PCI Express transaction types and I/O coherency

8.7.2 PCI Express I/O Coherency with System MMU

In the case where the system has a System MMU translating and attributing the transactions from the root complex, the PCI Express transactions must keep the memory attributes assigned by the System MMU. If the System MMU-assigned attribute is cacheable then it is IMPLEMENTATION DEFINED if No_snoop transactions replace the attribute with non-cached.

8.8 Legacy I/O

The specification does not specify a standard mechanism for supporting legacy I/O transactions. Software consistent with the Server Base System Architecture shall not support legacy I/O.

8.9 Integrated end points

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.

Anything claiming to follow the PCI Express specification must follow all the specification that is software-visible to ensure standard, quality software support.

8.10 Peer-to-peer

It is system-specific whether peer-to-peer traffic through the system is supported.

Systems compatible with level 3 or above of the SBSA 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.

8.11 PASID support

SMMUv3 included optional support for PCIe PASID. 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.

9 APPENDIX E: GICV2M ARCHITECTURE

9.1 Introduction

ARM is standardizing how PCI Express MSI and MSI-X interrupts are handled. The key part of this standardization is ensuring that each individual MSI(-X) message is presented to the operating system (OS) as a unique interrupt ID.

To facilitate this, using GICv2 interrupt controllers, ARM offers this specification to convert MSI(-X) writes to unique interrupts. 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.

9.2 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* on page 48. 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.

9.3 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.

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.

Table 16 GIC and GICv2m Security Extensions

The inclusion of the GICv2m Security Extensions adds a further memory-mapped Secure MSI register frame, described in Secure MSI register summary on page 48.

9.4 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.

9.5 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 *Virtualization* on page 47.

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.

9.6 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*.

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*.

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.

9.7 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.

Offset	Name	Description
0x000 – 0x007	-	Reserved.
0x008	MSI_TYPER	See *MSI Type Register* on page 49.
0x00C – 0x03C	-	RESERVED.
0x040	MSI_SETSPI_NS	See *Set SPI Register* on page 49.
0x044 – 0xFC8	-	Reserved.
0xFCC	MSI_IIDR	See *MSI Interface Identification Register* on page 50
0xFD0 – 0xFFC	-	IMPLEMENTATION DEFINED.

Table 17 GICv2m Non-secure MSI register summary

9.8 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.

Offset	Name	Description
0x000 – 0x007	-	Reserved.
0x008	MSI_TYPER	See *MSI Type Register* on page 49.
0x00C – 0x03C	-	Reserved.
0x040	MSI_SETSPI_S	See *Set SPI Register* on page 49.
0x044 – 0xFC8	-	Reserved.
0xFCC	MSI_IIDR	See *MSI Interface Identification Register* on page 50.
0xFD0 – 0xFFC	-	IMPLEMENTATION DEFINED.

Table 18 GICv2m Secure MSI register summary

9.9 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.

9.9.1 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 *SPI allocation* on page 47.

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.

9.9.2 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.

9.9.3 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.

9.10 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.

Offset	Name	Description
0x000 – 0x007	-	Reserved.
0x008	MSI_TYPER	See *MSI Type Register* on page 49.
0x00C – 0x03C	-	Reserved.
0x040	MSI_SETSPI_S	See *Set SPI Register* on page 49.
0x044 – 0xFC8	-	Reserved.
0xFCC	MSI_IIDR	See *MSI Interface Identification Register* on page 50.
0xFD0 – 0xFFC	-	IMPLEMENTATION DEFINED.

Table 19 GICv2m Secure MSI register summary

10 APPENDIX F: GIC-400 AND 64KB TRANSLATION GRANULE

The GICC register frame as defined in the GICv2 specification must be spread across two MMU pages in order that GICC_DIR is in a different page to the rest of GICC.

In a 64KB translation granule system this means that GICC needs to have its base at 4KB below a 64KB boundary.

The ARM implementation of GICv2, the GIC-400 product, aligns GICC to an 8KB boundary. An address wiring workaround is needed to use the GIC-400 in a 64KB translation granule system. The following is an example of how this may be done:

```
AddressGIC400[14:0] = {AddressSystem[18:16],AddressSystem[11:0]}
```

This has the effect of aliasing each 4KB of GIC registers 16 times in a 64KB page. System software can now be told that the last alias in a 64KB page is the GICC base, which conveniently runs into the first alias of the next 64KB page completing the GICC register frame.

11 APPENDIX G: GICV2M COMPATIBILITY IN A GICV3 SYSTEM

A key difference between GICv3 and GICv2 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 introduces 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.

11.1 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.

11.2 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.

12 APPENDIX H: SMMUV3 INTEGRATION

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

The system is permitted to include any number of SMMUs.

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

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.

The system must be constructed so 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.

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

See 8.11 PASID support for requirements on PCIe PASID support.

See 13 APPENDIX I: DeviceID generation and ITS groups for requirements on how DeviceID and StreamID should be assigned and how ITS groups should be used.

13 APPENDIX I: DEVICEID GENERATION AND ITS GROUPS

13.1 ITS groups

13.1.1 Introduction

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 13.2.

Note: 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.

13.1.2 Rules

- The system contains one or more ITS group(s).
- An ITS group may contain one or more ITS blocks.
- An ITS block is associated with one ITS group.
- A device that is expected to send an MSI is associated with one ITS group.
- Devices can be programmed to send MSIs to any ITS block within the group.
- 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 13.2.2 for permitted behavior of illegal MSI writes.

- The association of devices and ITS blocks to ITS groups is considered static by high-level software.
- An ITS group represents a DeviceID namespace independent of any other ITS group.
- 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.
- All ITS blocks within an ITS group observe the same DeviceID for any given device in the same ITS group.
    - Note: This rule, and the preceding rule, allows software to use ITS blocks sharing a common group interchangeably.
- System firmware data, for example, firmware tables such as ACPI/FDT, describe the association of ITS blocks and devices with ITS groups to high-level software.

ARM recommends that the DeviceID namespace in each group is as densely-packed as possible, and starts at 0 if possible.

Note: It is not required that all DeviceIDs be entirely contiguous but excessive fragmentation makes the software configuration of an ITS more difficult.

13.1.3 Examples of ITS groups

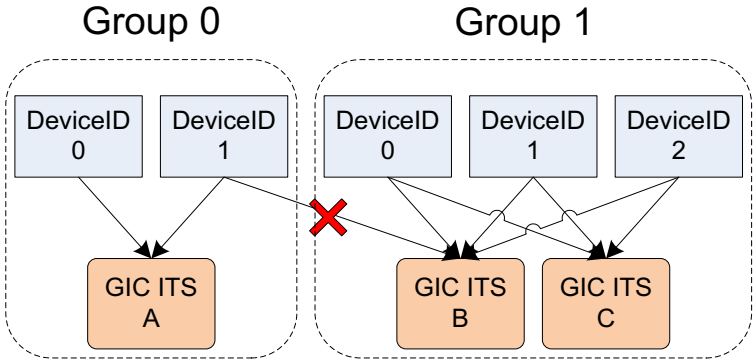

Figure 2: Device and ITS grouping

In Figure 2:

- ITS A serves two devices.
- ITS B and ITS C serve three devices; any of these three can send an MSI to either of 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.

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 master device, which DeviceID in the group's namespace the device has been assigned.

13.2 Generation of DeviceID values

13.2.1 Introduction

Every device master that is expected to send MSIs has a DeviceID associated with it. 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.

Where a device is a *client* of, that is, behind, an SMMU, a granularity of 'source' identification is typically chosen so that an individual device can be assigned to a less-privileged piece of software independent of neighboring SMMU client devices. The system designer assigns a master-unique StreamID to device traffic input to the SMMU. 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.

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.

If a master is a bridge from a different interconnect with an originator ID, such as 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.

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.

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. The system must not allow this behavior to trigger an MSI that masquerades as originating from a different master. The system must anticipate that PEs also have the potential to be misused in this manner. Exposing an ITS to a VM for legitimate MSI purposes can mean the untrusted VM software is able to write to the ITS MSI target register from a PE.

13.2.2 Rules

- Every device that is expected to originate MSIs is associated with a DeviceID.
- 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 13.1.2, 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.

  - Note: 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 (so no special steps are required to prevent malicious MSI writes), but a device that has the potential to be controlled by a VM cannot be trusted. Devices that are clients of an SMMU fall into the latter category.

- 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[15:0] ) + 0x10000*Constant_B`
    - This StreamID is then used post-SMMU, as above, to generate a DeviceID.
- 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[15:0] ) + 0x10000*Constant_C`
- The relationships between a device, its StreamID and its DeviceID are considered static by high-level software. If the mapping is not fixed by hardware, the relationship between a StreamID and a DeviceID must not change after system initialization.

ARM 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.
- **Note:** Providing separate DeviceIDs for different devices can improve the efficiency of structure allocation in GIC driver software.

13.3 System description of DeviceID and ITS groups from ACPI tables

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. In addition, for any given device expected to send MSIs, system firmware data tables must ensure that:

- The device's DeviceID 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's 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's association with an ITS group can be determined and the ITS blocks within the group can be enumerated.

Note: 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.

Note: PEs and other masters 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.

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.

13.4 DeviceIDs from hot-plugged devices

- 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.
- When a device is hot-plugged, it may 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.
- If a new device's DeviceID cannot be derived from existing mappings in system description tables, the hot-plug mechanism (e.g. via firmware) must provide a means to determine the new device's DeviceID.

Note: These points also apply to a new device's SMMU StreamID.

Note: In current systems, hot-plug device masters 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.

ARM recommends 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.