ARM® Compute Subsystem SCP

Version: 1.0

Message Interface Protocols

ARM DUI 0922B (ID062615)

ARM Compute Subsystem SCP
Message Interface Protocols

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
27 April 2015	A	Non-Confidential	First release
01 May 2015	B	Non-Confidential	Claify platforms to which this document applies

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

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

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM’s trademark usage guidelines at http://www.arm.com/about/trademark-usage-guidelines.php

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
ARM Compute Subsystem SCP Message Interface Protocols

Preface

This preface introduces the *ARM Compute Subsystem SCP Messaging Interface Protocols*. It contains the following sections:

- *About this book* on page vi
- *Feedback* on page viii.

About this book

This book is for the ARM *Compute Subsystem* (CSS) *System Control Processor* (SCP). A CSS is a reference IP subsystem from ARM. The Juno *ARM Development Platform* (ADP) is an example CSS-based platform

This book describes the *System Control and Power Interface* (SCPI) and *Boot Over MHU* (BOM) Message Interface Protocols that can be used for message passing between the SCP and the *Application Processor* (AP) on ARM CSS-based platforms. It also describes the *Message Handling Unit* (*MHU) Transport Layer* (MTL), a low-level protocol that handles communication between sender and receiver.

Intended audience

This book has been written for software developers who are interacting with an SCP within an ARM Compute Subsystem, from a firmware or operating system level. For example, for the purposes of power control.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this chapter for an introduction to the ARM CSS System Control Processor.

Chapter 2 ***CSS Message Handling Unit (MHU) Transport Layer***

Read this chapter for a description of the MHU Transport Layer.

Chapter 3 ***CSS System Control and Power Interface (SCPI)***

Read this chapter for details of the SCPI protocol.

Chapter 4 ***CSS Boot Over MHU (BOM) Protocol***

Read this chapter for a description of the BOM protocol.

Appendix A ***Juno ARM Development Platform(ADP) Implementation Details***

Read this appendix for details of the Juno-specific variants to the protocols.

Appendix B ***Revisions***

Read this appendix for a list of changes to the documentation.

Glossary

The *ARM® Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM® Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See *ARM® Glossary* http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html.

Typographical conventions

The following table describes the typographical conventions:

Typographical conventions

Style	Purpose
italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *`italic`*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists publications by ARM and by third parties.

See *Infocenter* `http://infocenter.arm.com`, for access to ARM documentation.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *Juno ARM® Development Platform SoC Technical Reference Manual* (ARM DDI 0515).

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- The title.
- The number, ARM DUI 0922B.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1
Introduction

The following is an introduction to the ARM CSS SCP:

- *About the System Control Processor* on page 1-2.

1.1 About the System Control Processor

An ARM *System Control Processor* (SCP) is a dedicated subsystem that exists alongside one or more application processors within an ARM CSS. Its main purpose is to initialize and control components both within the SoC, and outside the SoC, offloading these tasks from the application processors.

The key features of the SCP are as follows:

- Boot and system start-up and security integrity.
- Initial configuration and subsequent reset.
- Managing clocks, voltage regulators and associated operating performance points, to support *Dynamic Voltage and Frequency Scaling* (DVFS).
- Power state management for the power regions within the SoC.
- Handling hardware wake-up requests from components such as timers and interrupts.
- Responsible for maintaining and enforcing consistency between device states within the system.
- Sensor control and management

The services that are provided by the SCP are exposed to the AP software using message interfaces.

These message interfaces use areas of shared memory and the CSS MHU peripheral, which is used as a messaging signaling mechanism.

The MHU provides a mechanism to assert interrupt signals between the SCP and the application processor

Figure 1-1 shows a diagram of the communication between the SCP and AP.

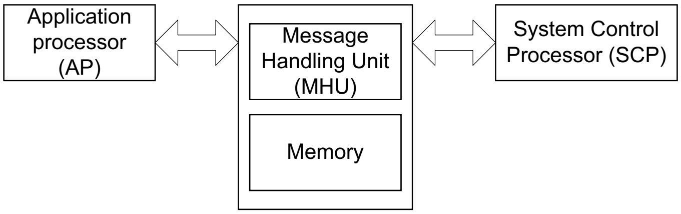
Application processor (AP)
Message Handling Unit (MHU)
Memory
System Control Processor (SCP)

Figure 1-1 SCP to AP communication

Chapter 2
CSS Message Handling Unit (MHU) Transport Layer

The following topics describe the MHU Transport Layer (MTL) protocol, as used in an ARM CSS:

- *Physical and virtual channels* on page 2-2.
- *Communication flow* on page 2-4.

2.1 Physical and virtual channels

Communication between the AP and SCP is achieved on and ARM CSS-based platforms by using both shared memory, and a hardware peripheral called the MHU. The *MHU Transport Layer* (MTL) protocol defines the concepts of both *physical* and *virtual* channels, that are used together to support data transfer and request signaling for one or more messaging interfaces:

- *Physical channels*.
- *Virtual channels*.

2.1.1 Physical channels

The CSS MHU peripheral provides physical channels that are used for communication between the SCP and the AP. These channels are called physical channels because their implementation is fixed in hardware. Each physical channel is unidirectional (SCP to AP or AP to SCP) and is either fully accessible or restricted to Secure access only..

The exact number of available physical channels, and their properties, varies depending on the specific CSS implementation.

A physical channel comprises:

- A 32-bit STAT (STATUS) register:
  - Read only. Writes ignored.
  - If any of the bits become set through writes to the corresponding SET register, an interrupt is asserted on the receiver.
- A 32-bit SET register:
  - Write only. Read as zero.
  - Sets bits in the associated STAT register.
- A 32-bit CLEAR register:
  - Write only. Read as zero.
  - Clears bits in the associated STAT register.
- An interrupt line to the receiver that the channel direction defines. For example, if the channel direction is “AP to SCP” the interrupt line is connected to the SCP.

The SCP regards each bit in the STAT register as a *slot* that can be mapped to a virtual channel.

Note

Slot 31 is reserved because the MHU hardware uses bit [31] to indicate a Non-secure access attempt.

The total number of available slots is therefore 31 [30:0].

The state of the slot bits in the STAT register act as a signaling method between the sender and receiver. If the sender wants to transmit a message across a virtual channel, it must first check the state of the slot bit associated with that channel. If the slot bit is clear, it is safe for the sender to initiate a new communication. If the bit is set, the associated virtual channel is in use. The sender must wait for the slot to clear before it writes a new message.

2.1.2 Virtual channels

Virtual channels are a software construct that are designed to add flexibility to the communication system.

A virtual channel defines:

- A shared region of memory into which protocol-specific data is written.
- The protocol that is used to communicate on the virtual channel, for example, SCPI. Both the AP and SCP must respect this association.
- A mapping to a physical channel that is used for signaling (the slot within the registers of the physical channel that the virtual channel is associated with).

The size and location of the shared memory region are IMPLEMENTATION DEFINED. However, the region must be readable by both the SCP and the AP. If the protocol associated with the virtual channel is bidirectional, the region must also be writable by both the SCP and the AP. If suitable arbitration is implemented in platform software, a single region of memory can also be shared between multiple virtual channels.

Figure 2-1 shows a physical channel, represented by the SET, CLEAR, and STATUS registers. The physical channel is supporting three virtual channels which occupy slots 0, 1, and 30. Three regions of shared memory are reserved, one per virtual channel.

The independent virtual channels share the single receiver-side interrupt line of the physical channel that they are associated with. Different entities within the same software domain can access the virtual channels concurrently. This shared access enables greater flexibility.

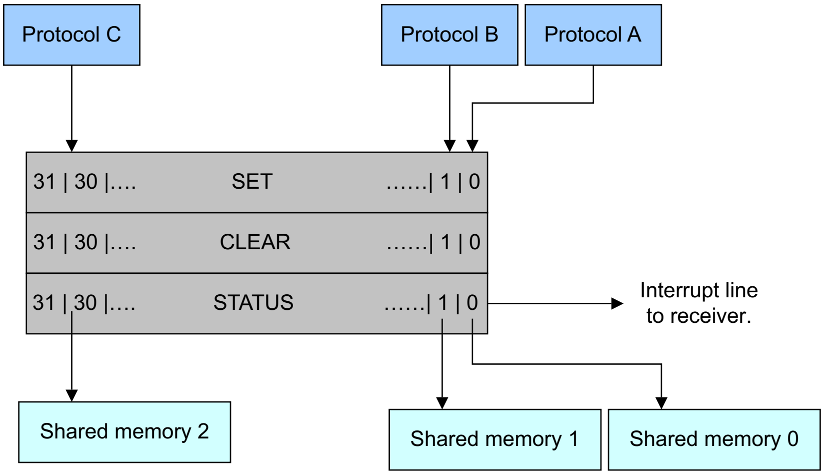

Figure 2-1 Protocols using MTL channels

2.2 Communication flow

In the examples that follow, the configuration in Figure 2-1 on page 2-3 applies. The examples assume that the sender uses the virtual channel that is associated with slot 0 (Protocol A, Shared Memory 0)..

2.2.1 Sending a message

1. The sender polls STAT [0] to determine if the virtual channel associated with slot 0 is ready to accept a message. The sender must wait until STAT[0] becomes clear to ensure that any previous commands have been received and processed.

2. The sender writes a message to the shared memory area of the receiver. The content and size of the message are protocol-specific. The sender must then ensure that the written data is visible to the receiver. This visibility is achieved, where appropriate, though the use of barriers, or their equivalent, before proceeding to the next step.

3. The sender writes 1 to SET [0]. This write signals to the receiver that a new message is pending on the virtual channel for slot 0.

These steps cause an interrupt to be asserted at the receiving end of the physical channel. This interrupt triggers the flow that is described in *Receiving a message*.

2.2.2 Receiving a message

1. An interrupt is raised at the receiving end of the physical channel. The recipient reads the STAT register to determine which slot bit has been set, and which virtual channel has a pending message.

2. The recipient can now access the message data in the shared memory area that is associated with the virtual channel. This message data is valid until the channel is next released (as described in the following step). When the channel is released, the next incoming message overwrites the existing data in the shared memory area.

   ARM recommends that the recipient creates a copy of the message data outside of the shared memory area so that the message can be processed asynchronously and the channel can be released as soon as possible.

3. The receiver writes 1 to CLEAR[0]. The write clears the corresponding slot bit in the STAT register. This action indicates to the sender that the command was successfully received, though not necessarily processed, and that the virtual channel is ready to accept a new message.

Chapter 3
CSS System Control and Power Interface (SCPI)

The SCPI is one of the primary interfaces to the SCP in an ARM CSS-based platform. It is used to access many of the services that are exposed to the AP. The SCP is expected to be idle and waiting for SCPI commands for most of the time after the system boot process completes.

An SCPI message consists of a compulsory header and an optional payload. The header and payload are written into the shared memory area that is associated with the virtual channel upon which the message is transmitted.

Either the SCP or the AP can send a message at any point in time. SCPI communication is asynchronous and bidirectional.

This chapter contains the following sections:

3.1 SCPI Message header

The header is a 64-bit structure that is defined as:

Table 3-1 SCPI Command header

Bits	Name	Description
[6:0]	Command ID	ID that identifies the command
[7]	Set ID	0 = Standard, 1 = Extended
[15:8]	Sender ID	Sender ID to match a reply. The value is sender-specific.
[24:16]	Payload Size	Size in bytes
[31:25]	RESERVED	
[63:32]	Status	Status indicating the result of a command. The status values are: **0** SCPI_OK - Success **1** SCPI_E_PARAM – Invalid parameter(s) **2** SCPI_E_ALIGN – Invalid alignment **3** SCPI_E_SIZE – Invalid size **4** SCPI_E_HANDLER – Invalid handler or callback **5** SCPI_E_ACCESS – Invalid access or permission denied **6** SCPI_E_RANGE – Value out of range **7** SCPI_E_TIMEOUT – Timeout has occurred **8** SCPI_E_NOMEM – Invalid memory area or pointer **9** SCPI_E_PWRSTATE – Invalid power state **10** SCPI_E_SUPPORT – Feature not supported or disabled **11** SCPI_E_DEVICE– Device error **12** SCPI_E_BUSY – Device is busy The status field value is only relevant when the AP is receiving a message from the SCP. Any status value set by the AP is ignored by the SCP.

The header is written into the first 64 bits of the shared memory area. The payload follows immediately afterwards. The SCPI protocol enforces a maximum payload size of 512 bytes. The protocol uses bits [24:16] to store the size value. However, there can also be an IMPLEMENTATION DEFINED limit on the payload size that is lower than 512 bytes. In this case, where a response is specified, exceeding the maximum payload size results in a response of SCPI_E_SIZE from the recipient.

For commands that elicit a response from the recipient, the Command ID and Set ID fields remain the same in the response, except for the size and the status. The Payload Size field (and Sender ID field if appropriate) must be updated in the response. If the SCP is sending the response to the AP, then the Status field is also updated.

The Sender ID field is a field that can be used to associate responses with requests. When the AP sends a message to the SCP and sets a Sender ID value, the SCP includes the Sender ID value in the response to the AP. This enables platform software on the AP to differentiate the response from other incoming messages.

If SCP sends a message to the AP that is not a response to a command, for example, a periodic temperature sensor reading, it identifies itself with a Sender ID of 0. However, the SCPI protocol does not explicitly reserve ID 0 for this purpose. AP software is free to leave the Sender ID as 0 for its commands if it does not require this functionality.

3.1.1 Channel ownership

Only one software entity (OS, firmware, or bootloader) owns a virtual channel at any point in time.

The owner of the channel is responsible for arbitration between multiple requests, for example, from cores, or threads, through any lock mechanism appropriate.

3.1.2 Endianness

All multi-byte values are little-endian.

3.2 SCPI commands

This section contains a comprehensive list of the standard supported SCPI command set. IMPLEMENTATION DEFINED commands can be added to the “extended set”. If these commands are used, the Set ID field of the header must be set to 1 if a command from the extended set is sent.

3.2.1 SCP Ready

At the end of the SCP boot sequence, when the SCP RAM Firmware image has been transferred, the SCP must leave the BL0 Firmware executing from ROM, and pass control to SCP RAM Firmware, executing from SRAM. The SCP uses the `Ready` command to inform the AP that it can now accept requests to prevent application cores from sending SCPI requests before the SCP RAM Firmware is ready.

——— **Note** ———

The SCP initiates this command.

———————————

Header details

Command ID	`0x01`
Set ID	0 (Standard)

AP to SCP payload

None.

SCP to AP payload

None.

3.2.2 Get SCP capability

This command described the SCP capabilities. AP software can use this command to query the supported version of the SCPI protocol, the event definitions, enabled command sets, and enabled commands.

Header details

Command ID	`0x02`
Set ID	0 (Standard)

AP to SCP payload

None.

SCP to AP payload

Table 3-2 Payload

Offset	Description
0	SCPI protocol version
4	Payload Size Limits
8	Platform version
12	Commands enabled 0
16	Commands enabled 1
20	Commands enabled 2
24	Commands enabled 3

Table 3-3 SCPI protocol version

Bits	Name	Description
[15:0]	SCPI Minor version	Implemented minor version of the SCPI protocol. Changes in the Minor version number do not break compatibility with previous versions based on the same Major version number.
[31:16]	SCPI Major version	Implemented major version of the SCPI protocol. Changes in the Major version number can break compatibility with previous versions.

Table 3-4 Payload size limits

Bits	Name	
[8:0]	AP Payload Size Limit	The maximum size for an SCPI payload in the AP to SCP direction.
[15:9]	RESERVED	-
[24:16]	SCP Payload Size Limit	The maximum size for an SCPI payload in the SCP to AP direction.
[31:25]	RESERVED	-

Table 3-5 Platform version

Bits	Name	Description
[31:0]	Version	IMPLEMENTATION DEFINED identifier for the platform revision.

Table 3-6 Commands enabled 0

Bits	Name	Description
[0]	Extended Set Enabled	When a platform extends the standard SCPI command set by implementing command IDs `0x80` to `0xFF`, this bit is set to 1. **0** Disabled **1** Enabled
[31:1]	Standard Command Set	Bitmap for the standard commands available (IDs `0x01` to `0x1F`). **0** Disabled **1** Enabled

Table 3-7 Commands enabled 1

Bits	Name	Description
[31:0]	Standard Command Set	Bitmap for the standard commands available (IDs `0x20` to `0x3F`). **0** Disabled **1** Enabled

Table 3-8 Commands enabled 2

Bits	Name	Description
[31:0]	Standard Command Set	Bitmap for the standard commands available (IDs `0x40` to `0x5F`). **0** Disabled **1** Enabled

Table 3-9 Commands enabled 3

Bits	Name	Description
[31:0]	Standard Command Set	Bitmap for the standard commands available (IDs `0x60` to `0x7F`). **0** Disabled **1** Enabled

3.2.3 Set CSS Power State

This command is used by the AP to change the power state of clusters, and the *Compute Sub-system* (CSS). The values of the *Cluster ID* and *CPU ID* fields depend on the particular CSS hardware configuration that has been implemented. The number of clusters and cores can vary.

The SCP does not reply to this command to avoid generating interrupts which must be handled by AP cores. To do so could interfere with a requested power state transition.

This command is processed from the lowest level domain upwards (Core→ Cluster→ CSS) and the SCP automatically acts to enforce any power state dependencies. For example, if a request to turn on a core is received, and the cluster that the core belongs to is powered off, the SCP will power on that cluster automatically, regardless of the requested Cluster Power State.

The Cluster Power State is only used when the last powered core in a cluster is being powered off. This is so that the SCP can determine whether the intent is to keep the cluster ON or to switch it to one of the available low-power modes. Likewise, the CSS Power State parameter is only processed when the last cluster is being powered off.

——— **Note** ———

During a power down sequence, the core being powered off must enter `WFI` to complete the sequence. Failing to enter `WFI` leads to undefined behavior. This behavior is a hardware restriction that is imposed to ensure that the core has reached a consistent state and that any transactions have been completed.

Header details

Command ID `0x03`

Set ID 0 (Standard)

AP to SCP payload

Table 3-10 AP to SCP payload

Offset	Description
0	Power State Descriptor

Table 3-11 Power state descriptor

Bits	Name	Description
[3:0]	CPU ID	A number which identifies the core in its cluster. This field supports up to eight cores in a cluster. The actual number of cores is platform-specific and can vary between clusters in the same system.
[7:4]	Cluster ID	Cluster ID is a number which identifies the cluster in the system.
[11:8]	CPU Power State	One of the supported power states.
[15:12]	Cluster power state	One of the supported power states.
[19:16]	CSS Power state	One of the supported power states.
[31:20]	RESERVED	-

SCP to AP payload

None.

3.2.4 Get CSS Power state

Returns the current state of each cluster and its cores in the CSS. The power state of the CSS is not returned. The implication is that it is ON if an AP core was able to send the command to the SCP.

There is no lock mechanism preventing an asynchronous wake-up event from happening while the SCP is processing this command.

The size of the payload is variable depending on the number of clusters that the system supports. The AP software must read the size field of the SCP to AP header, and handle the payload appropriately.

Header details

Command ID `0x04`

Set ID 0 (Standard)

AP to SCP payload

None.

SCP to AP payload

Table 3-12 SCP to AP payload

Offset	Description
0	Power State

Power state

The payload returns one or more copies of this structure. The actual count depends on the number of clusters in the system.

Table 3-13 Power state descriptor

Bits	Name	Description
[3:0]	Cluster ID	Cluster ID of the current entry.
[7:4]	Cluster Power State	Current cluster power state.
[15:8]	CPU Power State	Current power state of each core belonging to Cluster ID. Each bit corresponds to a core, with bit 0 assigned to CPU 0.

3.2.5 Set System Power State

This command sets the power state but acts at a system level.

All cores must be OFF (apart from the last core sending the command) before the SCP can shut down, reboot, or reset the system.

If there is more than one core still running, the SCP waits for an IMPLEMENTATION DEFINED time before abandoning the request. It assumes the system was not properly prepared for shutdown.

Header details

Command ID `0x05`

Set ID 0 (Standard).

AP to SCP payload

Table 3-14 AP to SCP payload

Offset	Description
0	System State

Table 3-15 System state

Bits	Name	Description
[7:0]	System Power State	One of the following power states: **0** Shutdown **1** Reboot – Board level reset **2** Reset – SoC level reset

SCP to AP payload

None.

3.2.6 Set CPU Timer

This command sets a timer to wake up a given core.

The SCP uses this timer when the target core is powered down.

If the timestamp is in the past when the core is being powered down, the core is reset. This timer is a *one-shot* timer and must be re-enabled after being triggered.

If the target core is powered down and a wake-up occurs (for example, an IRQ interrupt) before the timer expires, the timer is canceled. The timestamp is based on the generic counters that are shared between the SCP and AP which run at the REFCLK rate.

Header details

Command ID `0x06`

Set ID 0 (Standard)

AP to SCP payload

Table 3-16 AP to SCP payload

Offset	Description
0	Timestamp (LSB)
4	Timestamp (MSB)
8	CPU identifier

Table 3-17 Timestamp (LSB)

Bits	Name	Description
[31:0]	Timestamp (LSB)	Bytes 0-3 of the timestamp (in REFCLK ticks).

Table 3-18 Timestamp (MSB)

Bits	Name	Description
[31:0]	Timestamp (MSB)	Bytes 4-7 of the timestamp (in REFCLK ticks).

Table 3-19 CPU Identifier

Bits	Name	Description
[3:0]	CPU ID	CPU ID is a number which identifies the core in its cluster.
[7:4]	Cluster ID	Cluster ID is a number which identifies the cluster in the system.

SCP to AP payload

None.

3.2.7 Cancel CPU Timer

This command cancels any outstanding timer that can wake a given core.

If no outstanding timer is set on the given core, the command is ignored.

If this command is received before the target core is powered down, it clears the last `Set CPU Timer` request. No timer will be used on the next power down request.

Header details

Command ID `0x07`

Set ID 0 (Standard)

AP to SCP payload

Table 3-20 AP to SCP payload

Offset	Description
0	CPU identifier

Table 3-21 CPU Identifier

Bits	Name	Description
[3:0]	CPU ID	CPU ID is a number which identifies the core in its cluster.
[7:4]	Cluster ID	Cluster ID is a number which identifies the cluster in the system.

SCP to AP payload

None.

3.2.8 Get DVFS Capability

Returns the number of power domains with DVFS support in the system.

Header details

Command ID `0x08`

Set ID 0 (Standard)

AP to SCP payload

None.

SCP to AP payload

Table 3-22 SCP to AP payload

Offset	Description
0	Power domains

Table 3-23 Power domains

Bits	Name	Description
[7:0]	Count	Number of power domains with support for DVFS.

3.2.9 Get DVFS Info

This command returns the DVFS capabilities of a given power domain.

The size of this command depends on the number of *Operating Point*s (OPPs) supported by the power domain. The Frequency and Voltage tuple repeats for the number of operating points that are defined on the Header.

Header details

Command ID `0x09`

Set ID 0 (Standard)

AP to SCP payload

Table 3-24 AP to SCP payload

Offset	Description
0	Power Domain ID

Table 3-25 Power Domain ID

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being queried.

SCP to AP payload

Table 3-26 SCP to AP payload

Offset	Description
0	Domain information
4	Operating Point Tuple - Frequency
8	Operating Point Tuple - Voltage

Table 3-27 Domain information

Bits	Name	Description
[7:0]	Power Domain ID	ID of the power domain queried.
[15:8]	Number of Operating Points	Number of discrete operating points that are supported.
[31:16]	Latency	Worst case latency when switching operating points (in microseconds).

Table 3-28 Operating Point Tuple - Frequency

Bits	Name	Description
[31:0]	Frequency	Frequency in Hertz (Hz)

Table 3-29 Operating Point Tuple - Voltage

Bits	Name	Description
[31:0]	Voltage	Voltage in millivolts (mV)

3.2.10 Set DVFS

Sets the Operating Point of a given Power Domain.

`Operating Point Index` is an index (starting from 0) to the Operating Point List. See, *Get DVFS Info* on page 3-11 and *Get DVFS Capability* on page 3-11.

Header details

Command ID `0x0A`

Set ID 0 (Standard)

AP to SCP payload

Table 3-30 AP to SCP payload

Offset	Description
0	Domain and Index

Table 3-31 Domain and Index

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being set.
[15:8]	Operating Point Index	An index in the Operating Point List. See, *Get DVFS Info* on page 3-11.

SCP to AP payload

None.

3.2.11 Get DVFS

Returns the current Operating Point for the given Power Domain. This command returns only an index (starting from 0) to the Operating Point List. See, *Get DVFS Info* on page 3-11.

Header details

Command ID `0x0B`

Set ID 0 (Standard)

AP to SCP payload

Table 3-32 AP to SCP payload

Offset	Description
0	Power Domain

Table 3-33 Power Domain

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being queried

SCP to AP payload

Table 3-34 SCP to AP payload

Offset	Description
0	Index

Table 3-35 Index

Bits	Name	Description
[7:0]	Operating Point Index	An index in the Operating Point List. See, *Get DVFS Info* on page 3-11.

3.2.12 Get DVFS Statistics

This command is used to return the DVFS statistics for a given power domain. The payload contains a header followed by a 64-bit entry for each operating point.

——— **Note** ———

The SCP to AP payload size of this command depends on the number of operating points that the given power domain supports.

——— **Note** ———

After sending the response message, the SCP resets its statistics counters for the given domain and will begin gathering fresh data.

Header details

Command ID `0x0C`

Set ID 0 (Standard)

AP to SCP payload

Table 3-36 AP to SCP payload

Offset	Description
0	Power Domain

Table 3-37 Power Domain

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being queried.

SCP to AP

Table 3-38 SCP to AP payload

Offset	Description
0	Operating Point Count
4	Switch Count
8	Start Timestamp (LSB)
12	Start Timestamp (MSB)
16	Current Timestamp (LSB)
20	Current Timestamp (MSB)
24	Residency[n] (LSB)
28	Residency[n] (MSB)

Table 3-39 Operating Point Count

Bits	Name	Description
[7:0]	Number of Operating Points	Number of operating points that this power domain supports.
[31:8]	RESERVED	-

Table 3-40 Switch Count

Bits	Name	Description
[31:0]	Number of Switches	Total number of switches between operating points.

Table 3-41 Start Timestamp (LSB)

Bits	Name	Description
[31:0]	Start Timestamp (LSB)	This timestamp indicates when the SCP started collecting the data.

Table 3-42 Start Timestamp (MSB)

Bits	Name	Description
[31:0]	Start Timestamp (MSB)	This timestamp indicates when the SCP started collecting the data.

Table 3-43 Current Timestamp (LSB)

Bits	Name	Description
[31:0]	Current Timestamp (LSB)	This timestamp indicates the last time the SCP collected the data, giving the AP the time period that is associated with the statistics.

Table 3-44 Current Timestamp (MSB)

Bits	Name	Description
[31:0]	Current Timestamp (MSB)	This timestamp indicates the last time the SCP collected the data, giving the AP the time period that is associated with the statistics.

Table 3-45 Residency[n] (LSB)

Bits	Name	Description
[31:0]	Residency (LSB)	The duration (in REFCLK ticks) that the power domain spent in the DVFS operating point with ID = n.

Table 3-46 Residency[n] (MSB)

Bits	Name	Description
[31:0]	Residency (MSB)	The duration (in REFCLK ticks) that the power domain spent in the DVFS operating point with ID = n.

3.2.13 Get Clocks Capability

Returns the number of clocks in the system.

Header details

Command ID	`0x0D`
Set ID	0 (Standard)

AP to SCP payload

None.

SCP to AP

Table 3-47 SCP to AP payload

Offset	Description
0	Clocks

Table 3-48 Clocks

Bits	Name	Description
[31:0]	Count	Number of clocks.

3.2.14 Get Clock Info

Returns the details of a specific clock.

Header details

Command ID `0x0E`

Set ID 0 (Standard)

AP to SCP payload

Table 3-49 AP to SCP payload

Offset	Description
0	Clock ID

Table 3-50 Clock ID

Bits	Name	Description
[15:0]	Clock ID	Clock being queried.

SCP to AP payload

Table 3-51 SCP to AP payload

Offset	Description
0	Clock ID
2	Flags
4	Minimum rate
8	Maximum rate
12	Clock name

Table 3-52 Clock ID

Bits	Name	Description
[15:0]	Clock ID	Clock being queried.

Table 3-53 Flags

Bits	Name	Description
[0]	Readable	When 1, indicates that the SCP accepts a Get request.
[1]	Writable	When 1, indicates that the SCP accepts a Set request.
[15:2]	RESERVED	-

Table 3-54 Minimum rate

Bits	Name	Description
[31:0]	Minimum rate	Minimum frequency in Hertz (Hz)

Table 3-55 Maximum rate

Bits	Name	Description
[31:0]	Maximum rate	Maximum frequency in Hertz (Hz)

Table 3-56 Clock Name

Name	Description
Clock Name	C string of up to 20 single-byte characters (including the null terminator): `[a - Z] + [a - Z	0 - 9	_]*`

3.2.15 Set Clock Value

Sets a clock frequency value. If the device supports it, setting the Frequency value to zero disables the clock,

Header details

Command ID `0x0F`

Set ID 0 (Standard)

AP to SCP payload

Table 3-57 AP to SCP payload

Offset	Description
0	Clock ID
4	Value

Table 3-58 Clock ID

Bits	Name	Description
[15:0]	Clock ID	Clock being set.
[31:16]	RESERVED	-

Table 3-59 Value

Bits	Name	Description
[31:0]	Frequency	Frequency in Hertz (Hz)

SCP to AP payload

None.

3.2.16 Get Clock Value

Gets the clock frequency value in Hertz (Hz).

Header details

Command ID `0x10`

Set ID 0 (Standard)

AP to SCP payload

Table 3-60 AP to SCP payload

Offset	Description
0	Clock ID

Table 3-61 Clock ID

Bits	Name	Description
[15:0]	Clock ID	Clock being queried.

SCP to AP payload

Table 3-62 SCP to AP payload

Offset	Description
0	Value

Table 3-63 Value

Bits	Name	Description
[31:0]	Frequency	Frequency in Hertz (Hz).

3.2.17 Get Power Supply Capability

Returns the number of manageable power supplies or voltage regulators in the system.

Header details

Command ID `0x11`

Set ID 0 (Standard)

AP to SCP payload

None.

SCP to AP payload

Table 3-64 SCP to AP payload

Offset	Description
0	Power supplies

Table 3-65 Power Supplies

Bits	Name	Description
[15:0]	Count	Number of power supplies.

3.2.18 Get Power Supply Info

Returns the configuration of a specific power supply.

Header details

Command ID `0x12`

Set ID 0 (Standard)

AP to SCP payload

Table 3-66 AP to SCP payload

Offset	Description
0	Power Supply ID

Table 3-67 Power supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power supply being queried.

SCP to AP payload

Table 3-68 SCP to AP payload

Offset	Description
0	Power Supply ID
2	Flags
4	Minimum voltage
8	Maximum voltage
12	Power Supply name

Table 3-69 Power Supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power Supply being queried.

Table 3-70 Flags

Bits	Name	Description	
[0]	Readable	**0**	The power supply value cannot be read.
		1	The power supply value can be read with the Get Power Supply command.
[1]	Writable	**0**	The power supply value cannot be written.
		1	The power supply value can be modified with the Set Power Supply command.
[15:2]	RESERVED	-	

Table 3-71 Minimum Voltage

Bits	Name	Description
[31:0]	Minimum voltage	Minimum voltage in millivolts (mV)

Table 3-72 Maximum Voltage

Bits	Name	Description
[31:0]	Maximum voltage	Maximum voltage in millivolts (mV)

Table 3-73 Power Supply Name

Name	Description
Power Supply Name	C string of up to 20 single-byte characters (including the null terminator): `[a - Z] + [a - Z	0 - 9	_]*`

3.2.19 Set Power Supply

Sets a power supply voltage value.

Note

If the device supports being switched off, setting the voltage value to 0 disables the power supply.

Header details

Command ID `0x13`

Set ID 0 (Standard)

AP to SCP payload

Table 3-74 AP to SCP payload

Offset	Description
0	Power Supply ID
4	Voltage

Table 3-75 Power Supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power Supply being set
[31:16]	RESERVED	-

Table 3-76 Voltage

Bits	Name	Description
[31:0]	Voltage	Voltage to set the power supply to in mV

SCP to AP payload

None.

3.2.20 Get Power Supply

Gets the voltage of the power supply.

Header details

Command ID `0x14`

Set ID 0 (Standard)

AP to SCP payload

Table 3-77 AP to SCP payload

Offset	Description
0	Power Supply ID

Table 3-78 Power Supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power Supply being set

SCP to AP payload

Table 3-79 SCP to AP payload

Offset	Description
0	Voltage

Table 3-80 Voltage

Bits	Name	Description
[31:0]	Voltage	Voltage of the power supply in mV. **Note** This returns the target voltage that the power supply is set to. It does not return the measured voltage.

3.2.21 Get Sensor Capability

Returns the number of sensor devices in the system. For example, sensors for temperature, voltage, and power.

Header details

Command ID `0x15`

Set ID 0 (Standard)

AP to SCP payload

None.

SCP to AP payload

Table 3-81 SCP to AP payload

Offset	Description
0	Sensors

Table 3-82 Sensors

Bits	Name	Description
[15:0]	Count	Number of sensors

3.2.22 Get Sensor Info

Returns detailed configuration information for a specific sensor.

Header details

Command ID `0x16`

Set ID 0 (Standard)

AP to SCP payload

Table 3-83 AP to SCP payload

Offset	Description
0	Sensor ID

Table 3-84 Sensor ID

Bits	Name	Description
[15:0]	Sensor ID	Sensor being queried

SCP to AP payload

Table 3-85 SCP to AP payload

Offset	Description
0	Sensor ID
2	Sensor class
3	Sensor triggers
4	Sensor name

Table 3-86 Sensor ID

Bits	Name	Description
[15:0]	Sensor ID	Sensor being queried

Table 3-87 Sensor Class

Bits	Name	Description
[7:0]	Sensor class	One of the following sensor classes: **0** Temperature **1** Voltage **2** Current **3** Power

Table 3-88 Sensor Triggers

Bits	Name	Description
[7:0]	Trigger types	Indicates the types of trigger that the sensor supports: **0** None supported **1** Periodic trigger **2** Bounds trigger **3** Bounds and periodic (not concurrently)

Table 3-89 Sensor Name

Name	Description
Sensor name	C string of up to 20 single-byte characters (including the null terminator): `[a - Z] + [a - Z	0 - 9	_]*`

3.2.23 Get Sensor Value

Reads a sensor value.

Header details

Command ID `0x17`

Set ID 0 (Standard)

AP to SCP payload

Table 3-90 AP to SCP payload

Offset	Description
0	Sensor ID

Table 3-91 Sensor ID

Bits	Name	Description
[15:0]	Sensor ID	The sensor being read

SCP to AP payload

Table 3-92 SCP to AP payload

Offset	Description
0	Sensor value

Table 3-93 Sensor Value

Bits	Name	Description
[31:0]	Value	The range and unit are specific to the sensor being read.

3.2.24 Config Periodic Sensor Readings

Configures automatic periodic reading of the sensor. The SCP sends an `Async Sensor Value` command when the sensor takes a reading.

Note

The periods a sensor supports are sensor and platform specific.

Periodic sensor readings and sensor bounds are mutually exclusive at the level of individual sensors. It is therefore not possible to enable both at once for a given sensor. It is possible to have both types of reading enabled within the system, if the previous, per-sensor constraint is respected.

The type of asynchronous sensor reading that is enabled for a given sensor is the last type that was requested. For example, enabling periodic readings and then enabling sensor bounds result in only bounds-based readings from the sensor.

Header details

Command ID	`0x18`
Set ID	0 (Standard)

AP to SCP payload

Table 3-94 AP to SCP payload

Offset	Description
0	Sensor ID
2	Recurrence
4	Period

Table 3-95 Sensor ID

Bits	Name	Description
[15:0]	Sensor ID	Sensor being configured for periodic readings.

Table 3-96 Recurrence

Bits	Name	Description
[15:0]	Recurrence	Specifies how many periodic readings are taken before stopping. The following values are special: **0x0000** Stop sensor readings **0xFFFF** Continuous sensor readings

Table 3-97 Period

Bits	Name	Description
[31:0]	Period	Period between readings in REFCLK ticks.

SCP to AP payload

None.

3.2.25 Config Sensor Bounds

Configures the upper and lower bounds on a sensor, and a set of trigger conditions that generate an `Async Sensor Value` command if their conditions are satisfied.

——— **Note** ———

The sensor device must support self-monitoring or limits because the SCP does not poll the sensor to determine if the trigger conditions have been met.

A sensor advertises its support for bounds-based readings when its configuration is queried. See, *Get Sensor Info* on page 3-26 for more information.

If a sensor is configured using the `Config Sensor Bounds` command by mistake, when it does not support bounds, the trigger conditions will never be met. No `Async Sensor Value` commands are issued for the misconfigured sensor.

———————————

Periodic sensor readings and sensor bounds are mutually exclusive at the level of individual sensors. It is not possible to enable both at once for a given sensor. It is possible to enable both types of reading within the system, if any previous per-sensor constraint is respected.

The type of asynchronous sensor reading that is enabled for a given sensor is the last type that is requested. For example, if periodic readings and then sensor bounds are enabled, only bounds-based readings from the sensor are taken.

Header details

Command ID `0x19`

Set ID 0 (Standard)

AP to SCP payload

Table 3-98 AP to SCP payload

Offset	Description
0	Sensor ID
2	Triggers
4	Lower Limit
8	Upper Limit

Table 3-99 Sensor ID

Bits	Name	Description
[15:0]	Sensor ID	Sensor being configured for bounds-based readings.

Table 3-100 Triggers

Bits	Name	Description
[11:0]	Recurrence	Specifies how many triggers can fire before SCP stops sending sensor readings. The following values are special: **0x0000** Remove configured triggers and stop sending sensor readings. **0xFFFF** Continue accepting trigger events and sending continuous sensor readings
[12]	Trigger Above Upper Limit	Send a reading when the sensor crosses the upper limit (increasing): **0** Disabled **1** Enabled

Table 3-100 Triggers (continued)

Bits	Name	Description
[13]	Trigger Below Upper Limit	Send a reading when the sensor crosses the upper limit (decreasing). **0** Disabled **1** Enabled
[14]	Trigger Above Lower Limit	Send a reading when the sensor crosses the lower limit (increasing). **0** Disabled **1** Enabled
[15]	Trigger Below Lower Limit	Send a reading when the sensor crosses the lower limit (decreasing). **0** Disabled **1** Enabled

Table 3-101 Lower limit

Bits	Name	Description
[31:0]	Value	The lower limit for triggers. If the sensor detects that its reading has crossed this threshold in either direction, it evaluates the trigger conditions and fires enabled triggers. This causes the SCP to send an `Async Sensor Value` message to the AP. The units for this value are specific to the sensor being configured.

Table 3-102 Upper limit

Bits	Name	Description
[31:0]	Value	The upper limit for triggers. If the sensor detects that its reading has crossed this threshold in either direction, it evaluates the trigger conditions and fires enabled triggers. This causes the SCP to send an `Async Sensor Value` message to the AP. The units for this value are specific to the sensor being configured.

SCP to AP payload

None.

3.2.26 Async Sensor Value

Sends a sensor reading from the SCP to the AP when a sensor with periodic readings or bounds-based triggers enabled has new data available.

Note

The SCP initializes this command.

Header details

Command ID `0x1A`

Set ID 0 (Standard)

AP to SCP payload

None.

SCP to AP payload

Table 3-103 SCP to AP payload

Offset	Description
0	Metadata
4	Sensor Value

Table 3-104 Metadata

Bits	Name	Description
[15:0]	Sensor ID	Sensor from which the value was read.
[31:16]	Reading ID	Sequential number that is incremented at each reading.

Table 3-105 Sensor Value

Bits	Name	Description
[31:0]	Value	The range and units are sensor-specific.

3.2.27 Set Device Power State

Set the power state of a given peripheral device. This command is distinct from the CSS Power State equivalents. It is designed for configuration of peripheral devices that are not part of the CSS.

Header details

Command ID `0x1B`

Set ID 0 (Standard)

AP to SCP payload

Table 3-106 SCP to AP payload

Offset	Description
0	Device

Table 3-107 Device

Bits	Name	Description
[15:0]	Device ID	Identifier of the device being set.
[23:16]	Power State	Target device power state.

SCP to AP payload

None.

3.2.28 Get Device Power State

Get the power state of a given peripheral device. This command is distinct from the CSS Power State equivalents. It is designed for configuration of peripheral devices that are not part of the CSS.

Header details

Command ID `0x1C`

Set ID 0 (Standard)

AP to SCP payload

Table 3-108 AP to SCP payload

Offset	Description
0	Device ID

Table 3-109 Device ID

Bits	Name	Description
[15:0]	Device ID	Identifier of the device being queried.

SCP to AP payload

Table 3-110 SCP to AP payload

Offset	Description
0	Power State

Table 3-111 Power State

Bits	Name	Description
[7:0]	Power State	Current device power state:

Chapter 4
CSS Boot Over MHU (BOM) Protocol

The following topics describe the *Boot Over MHU* (BOM) protocol:

- *About the BOM protocol* on page 4-2.
- *Boot protocol flow* on page 4-4.

4.1 About the BOM protocol

The BOM protocol in ARM CSS-based platforms is used by the AP within the CSS to transfer a RAM firmware image to SCP during the boot process.

The protocol is based on two secure physical channels, one in each direction, two virtual channels and a shared memory area of 16 bytes. Multi-byte data must be considered little-endian.

- 4 bytes for the command header.
- 8 bytes for the payload.
- 4 reserved bytes.

Because the protocol shares a single area of memory between the two virtual channels, arbitration is required to prevent the SCP from overwriting data that is written by the AP, or the AP overwriting data that is written by the SCP.

All operations affecting the shared memory are considered synchronous. After writing to memory, both the SCP and the AP must wait for the appropriate response to their command before writing again.

Boot Over MHU supports two commands:

Info Contains information about the firmware image being transferred, such as the total size, and an Adler32 checksum value.

Data Used to instruct SCP to copy across one or more portions of the firmware image into its internal RAM.

The AP always initiates the sequence by sending a single `Info` command, followed by a varying number of `Data` commands.

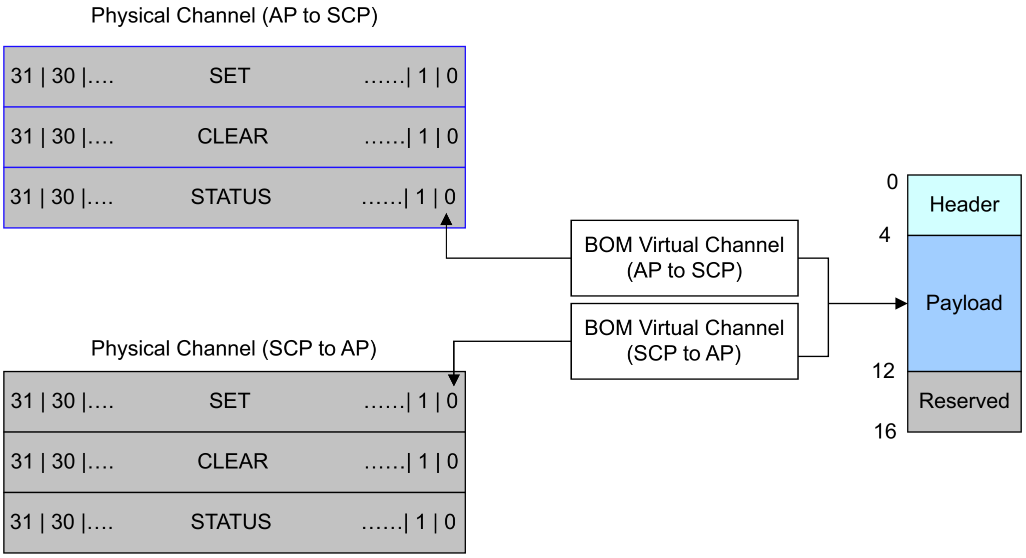

Figure 4-1 BOM protocol

The firmware image can be copied across completely using a single Data command and setting the Block Size to be the total image size. It can also be copied by using multiple Data commands with a smaller Block Size value. The results are equivalent and the choice is left to the AP platform software.

Note

The block data itself is not copied into the shared memory area of the channel. Instead, its location is indicated in the command. This approach reduces the number of internal copies on the AP firmware.

4.1.1 Error handling

The SCP responds to each command with a status value which uses zero to indicate success or non-zero values to indicate failure. In contrast to the SCPI protocol, the meanings of the return codes are not consistent across all commands. Refer to the description of individual commands when interpreting non-zero return codes.

When a failure occurs the SCP requires the previous, failed command to be reissued. A platform-specific limit exists for the maximum number of failed commands that the SCP tolerates. The exception to this rule is if checksum validation fails after the SCP receives the final Data command. In such a case, regardless of the number of failed commands that are seen, the SCP sends a response message with the status code set to `Corrupted Image`.

Once the limit is exceeded, the boot process is considered to have failed and a platform-specific response is carried out. The BOM protocol does not enforce any particular course of action. However, some suitable possibilities are:

- Resetting the platform.
- Restarting the complete boot process.
- Negotiating an alternative boot method.

4.1.2 Validation

The SCP ROM firmware implements an Adler-32 checksum algorithm that is used to validate the RAM firmware image once it has been transferred. The checksum is only used to ensure that the image has been received without any corruption by hardware or software and must not be considered a form of authentication. If the platform requires authentication, the AP secure firmware must authenticate the SCP RAM firmware image before sending it to the SCP.

4.2 Boot protocol flow

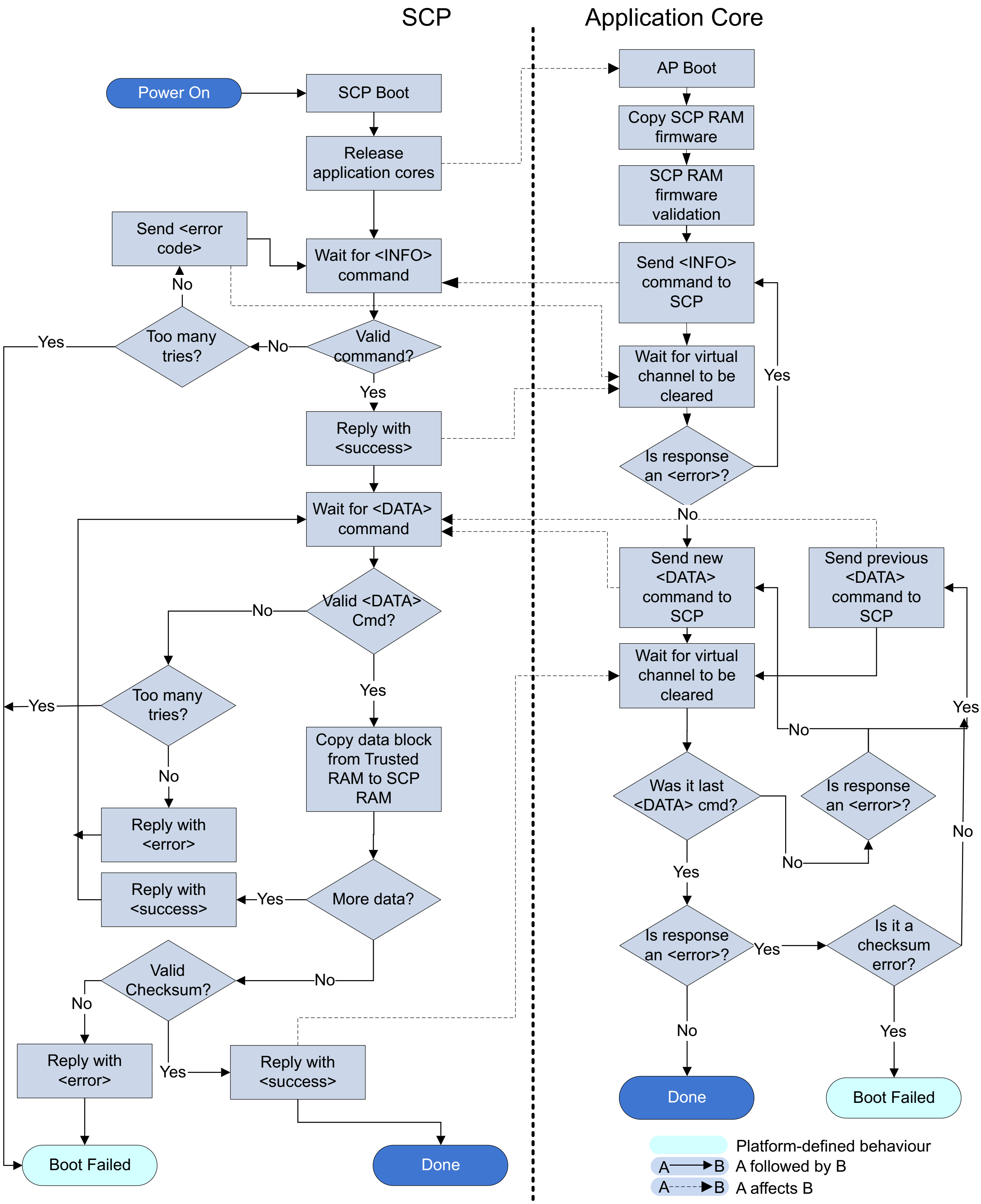

Figure 4-2 Boot protocol flow

4.2.1 Command Header

The command header common to the Info and Data commands, is written at the beginning of the shared memory that is dedicated for the BOM protocol, followed by a single command as a payload.

Table 4-1 Command Header

Bits	Name	Description
[7:0]	Command ID	ID that identifies the command: **0x00** Info **0x01** Data **0x02 - 0xFF** Invalid
[31:8]	RESERVED	-

4.2.2 Command Set

This section contains the following topics:

- *Info*
- *Data* on page 4-6.
- *Invalid commands* on page 4-7.

Info

The `Info` command contains information about the firmware image to be transferred, the size of the image, and its Adler-32 checksum.

The AP to SCP Payload contains:

Table 4-2 AP to SCP payload

Offset	Description
0	Image size
4	Image hash

Table 4-3 Image Size (4 bytes)

Bits	Name	Description
[31:0]	Image size	Image size in bytes. The value must be a multiple of 4.

Table 4-4 Image Hash (4 bytes)

Bits	Name	Description
[31:0]	Image hash	Adler-32 checksum.

The SCP to AP status response is a 32 bits parameter with the following values:

Table 4-5 SCP to AP status response

Bits	Name	Description	
[31:0]	Status	**0**	Success
		1	Unexpected command
		2	Image size is not a multiple of 4 bytes
		3	Image size exceeds a platform-specific maximum size.

Data

The AP to SCP `Data` command transfers blocks of the SCP RAM firmware image from the AP to the SCP. The command is sent one or more times until the whole SCP RAM firmware image is sent to the SCP.

The AP to SCP Payload contains:

Table 4-6 AP to SCP payload

Offset	Description
0	Trusted RAM offset
4	Block size

Table 4-7 Trusted RAM offset (4 bytes)

Bits	Name	Description
[31:0]	Offset	Address offset of the image block from the base of the Trusted RAM. The value must be 4-byte aligned.

Table 4-8 Block size (4 bytes)

Bits	Name	Description
[31:0]	Block size	Size in bytes. Value must be a multiple of four.

The SCP to AP status response is a 32 bits parameter with the following values:

Table 4-9 SCP to AP status response

Bits	Name	Description	
[31:0]	Status	**0**	Success
		1	Unexpected command
		2	Block size is not a multiple of 4 bytes
		3	Offset is not 4 bytes aligned.
		4	Offset is outside Trusted RAM
		5	Data overflow (Offset + Block Size is outside Trusted RAM)
		6	Total image size exceeds maximum size
		7	Corrupted image[a]

a. The SCP only validates the image on the last data block transfer.

Invalid commands

The SCP responds to any invalid command with Status = 1:

Table 4-10 Invalid commands

Bits	Name	Description	
[31:0]	Status	**1**	Unexpected command

Appendix A
Juno ARM Development Platform(ADP) Implementation Details

Previous chapters have covered the SCP Message Interface Protocols in a generic manner that is applicable to all current ARM Compute Subsystems, without reference to configurations and values that are specific to any CSS or platform.

For the Juno ADP the specific protocol constraints and implementation details are provided in the following sections:

- *MHU Transport Layer (MTL) Configuration* on page A-2.
- *System Control and Power Interface (SCPI) header format* on page A-4.
- *SCPI commands* on page A-5.
- *Boot Over MHU (BOM) Protocol* on page A-16.

A.1 MHU Transport Layer (MTL) Configuration

MTL Configuration contains the following topics:

- *Physical channels*.
- *Virtual channels*.

A.1.1 Physical channels

On Juno, there are six physical MHU channels:

- AP to SCP Secure.
- SCP to AP Secure.
- AP to SCP High-priority.
- SCP to AP High-priority.
- AP to SCP Low-priority.
- SCP to AP Low-priority.

Refer to the *Juno ARM® Development Platform SoC Technical Reference Manual* for more details.

A.1.2 Virtual channels

This section covers three areas:

- *Secure channels*.
- *Low priority channels* on page A-3.
- *High priority channels* on page A-3.

Secure channels

The secure virtual channels use the following physical channels:

- AP to SCP Secure
- SCP to AP Secure

They implement the following protocols:

- Slot 0: Boot protocol
  - SCP to AP shared memory: `0x04000080` - `0x0400017F`
  - AP to SCP shared memory: `0x04000180` - `0x0400027F`
  - Implemented in the SCP ROM firmware only.
  - Only available to Trusted Firmware during the boot sequence.
- Slot 0: SCPI protocol
  - SCP to AP shared memory: `0x4000080` - `0x0400017F`
  - AP to SCP shared memory: `0x4000180` - `0x0400027F`
  - Implemented in the SCP RAM firmware only.
  - The shared memory region and physical channel slot are the same as for the boot protocol. This is possible because the two protocols are never used at the same time.
- Slots 1-30:
  - Unused.

Low priority channels

The low-priority virtual channels use the following physical channels:

- AP to SCP Low-priority
- SCP to AP Low-priority

They implement the following protocols:

- Slot 0: SCPI protocol
  - SCP to AP shared memory: `0x2E000000` - `0x2E0000FF`
  - AP to SCP shared memory: `0x2E000100` - `0x2E0001FF`
- Slots 1-30:
  - Unused.

High priority channels

The high-priority virtual channels use the following physical channels:

- AP to SCP High-priority
- SCP to AP High-priority

They implement the following protocols:

- Slot 0: SCPI protocol
  - SCP to AP shared memory: `0x2E000200` - `0x2E0002FF`
  - AP to SCP shared memory: `0x2E000300` - `0x2E0003FF`
- Slots 1-30:
  - Unused.

A.1.3 Physical channel ownership

On Juno the following ownership applies to the physical channels:

- Secure channels:
  - ARM® Trusted Firmware.
- High and Low-priority channels
  - UEFI (during system boot)
  - operating system (after system boot)

A.2 System Control and Power Interface (SCPI) header format

The header is a 64-bit structure that is defined as:

Table A-1 SCPI Command header

Bits	Name	Description
[6:0]	Command ID	ID that identifies the command
[7]	Set ID	0 = Standard, 1 = Extended
[15:8]	Sender ID	Sender ID to match a reply. The value is sender-specific.
[24:16]	Payload Size	Size in bytes up to a maximum of 256.
[31:25]	RESERVED	
[63:32]	Status	Status indicating the success of a command. The status field is only used by the SCP when it is sending a message to the AP. **0** SCPI_OK - Success **1** SCPI_E_PARAM – Invalid parameter(s) **2** SCPI_E_ALIGN – Invalid alignment **3** SCPI_E_SIZE – Invalid size **4** SCPI_E_HANDLER – Invalid handler or callback **5** SCPI_E_ACCESS – Invalid access or permission denied **6** SCPI_E_RANGE – Value out of range **7** SCPI_E_TIMEOUT – Timeout has occurred **8** SCPI_E_NOMEM – Invalid memory area or pointer **9** SCPI_E_PWRSTATE – Invalid power state **10** SCPI_E_SUPPORT – Feature not supported or disabled **11** SCPI_E_DEVICE– Device error **12** SCPI_E_BUSY – Device is busy

Note

Shaded entries are Juno-specific variants.

A.3 SCPI commands

This section provides information on the SCPI command set as implemented on the Juno platform. Each command is listed along with its default attributes, as set by the reference SCP firmware. Additional constraints on the payload data are shown where applicable.

Note

Where payload structures are shown it is the shaded fields that give Juno-specific constraints. Fields without shading are unchanged from the generic SCPI command set.

A.3.1 SCP Ready

Default Attributes Secure channel or channels only.

There are no additional restrictions on the payload of this command in Juno.

A.3.2 Get SCP capability

Default Attributes None.

There are no additional restrictions on the payload of this command in Juno.

A.3.3 Set CSS Power State

Default Attributes Secure channel or channels only.

Table A-2 Power state descriptor

Bits	Name	Description
[3:0]	CPU ID	CPU ID is a number which identifies the core in its cluster. **0** CPU0 **1** CPU1 **2** CPU2 (LITTLE cluster only) **3** CPU3 (LITTLE cluster only)
[7:4]	Cluster ID	Cluster ID is a number which identifies the cluster in the system. **0** big **1** LITTLE
[11:8]	CPU Power State	**0** ON **1** RESERVED **2** RESERVED **3** OFF

Table A-2 Power state descriptor (continued)

Bits	Name	Description
[15:12]	Cluster power state	**0** ON **1** RESERVED **2** RESERVED **3** OFF
[19:16]	CSS Power state	One of the supported power states. **0** ON. All subsystems ON **1** Sleep0 - The SYSTOP power domain is placed in memory retention and DDR enters Deep Power Down state **2** RESERVED **3** RESERVED
[31:20]	RESERVED	-

A.3.4 Get CSS Power state

Default Attributes None.

Table A-3 Power state

Bits	Name	Description
[3:0]	Cluster ID	Cluster ID of the current entry.
[7:4]	Cluster Power State	Current cluster power state. **0** ON **1** RESERVED **2** RESERVED **3** OFF
[15:8]	CPUs power state	Current power state of each core belonging to Cluster ID. Each bit corresponds to a core, with bit 0 assigned to CPU 0. **0** Off **1** On

A.3.5 Set System Power State

Default Attributes Secure channel or channels only.

There are no additional restrictions on the payload of this command in Juno.

A.3.6 Set CPU Timer

Default Attributes Secure channel or channels only.

There are no additional restrictions on the payload of this command in Juno.

Note

The REFCLK rate on Juno is 50MHz.

A.3.7 Cancel CPU Timer

Default Attributes Secure channel or channels only.

There are no additional restrictions on the payload of this command in Juno.

A.3.8 Get DVFS Capability

Default Attributes None.

There are no additional restrictions on the payload of this command in Juno.

A.3.9 Get DVFS Info

Default Attributes None.

Table A-4 Power Domain ID

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being queried. **0** VBIG **1** VLITTLE **2** VGPU

A.3.10 Set DVFS

Default Attributes None.

Table A-5 Domain and Index

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being set. **0** VBIG **1** VLITTLE **2** VGPU
[15:8]	Operating Point Index	An index in the Operating Point List. See *Get DVFS Info*.

A.3.11 Get DVFS

Default Attributes None.

Table A-6 Power Domain ID

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being queried. **0** VBIG **1** VLITTLE **2** VGPU

A.3.12 Get DVFS Statistics

Default Attributes None.

Table A-7 Domain and Index

Bits	Name	Description
[7:0]	Power Domain ID	Power domain being queried. **0** VBIG **1** VLITTLE **2** VGPU

A.3.13 Get Clocks Capability

Default Attributes None.

There are no additional restrictions on the payload of this command in Juno.

A.3.14 Get Clock Info

Default Attributes None.

Table A-8 Clock ID

Bits	Name	Description
[7:0]	Clock ID	Clock being queried. **0** big cluster **1** LITTLE cluster **2** GPU **3** HDLCD_0 **4** HDLCD_1 **5** I2S **6** HDLCD REFCLK **7** HDLCD PXL_CLK_IN

A.3.15 Set Clock Value

Default Attributes None.

Table A-9 Clock ID

Bits	Name	Description
[7:0]	Clock ID	Clock being set. **0** big cluster **1** LITTLE cluster **2** GPU **3** HDLCD_0 **4** HDLCD_1 **5** I2S **6** HDLCD REFCLK **7** HDLCD PXL_CLK_IN

Note

Some of the clock devices are not writable. Use the *Get Clock Info* on page A-8 command to retrieve access permissions for each clock.

A.3.16 Get Clock Value

Default Attributes None.

Table A-10 Clock ID

Bits	Name	Description
[7:0]	Clock ID	Clock being queried. **0** big cluster **1** LITTLE cluster **2** GPU **3** HDLCD_0 **4** HDLCD_1 **5** I2S **6** HDLCD REFCLK **7** HDLCD PXL_CLK_IN

A.3.17 Get Power Supply Capability

Default Attributes None.

There are no additional restrictions on the payload of this command in Juno.

A.3.18 Get Power Supply Info

Default Attributes None.

Table A-11 Power Supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power supply being queried. **0** VSYS **1** VBIG **2** VLITTLE **3** VGPU

A.3.19 Set Power Supply

Default Attributes None.

Table A-12 Power Supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power supply being set. **0** VSYS **1** VBIG **2** VLITTLE **3** VGPU

Table A-13 Voltage

Bits	Name	Description
[31:0]	Voltage	Voltage to set the power supply to (in mV). On Juno, the maximum voltage for any power supply is 1100mV. Any request to set a power supply voltage above this level is rejected. Individual power supplies within the system enforce their own, component-specific limits which can be lower than the platform maximum.

Note

All power supply devices on Juno are read only.

A.3.20 Get Power Supply

Default Attributes None.

Table A-14 Power Supply ID

Bits	Name	Description
[15:0]	Power Supply ID	Power supply being queried. **0** VSYS **1** VBIG **2** VLITTLE **3** VGPU

A.3.21 Get Sensor Capability

Default Attributes None.

There are no additional restrictions on the payload of this command in Juno.

A.3.22 Get Sensor Info

Default Attributes None.

Table A-15 Sensor ID

Bits	Name	Description
[7:0]	Sensor ID	The sensor being queried. Juno R0 only **0** PMIC Temperature. **1** big Cluster Voltage. **2** LITTLE Cluster Voltage. **3** SoC Temperature. **4** SYSTOP Voltage. **5** SYSTOP Supply Voltage. **6** big Cluster Supply Voltage. **7** LITTLE Cluster Supply Voltage. **8** GPU Supply Voltage. Juno R1 only: **0** PMIC Temperature. **1** big Cluster Temperature. **2** big Cluster Voltage. **3** LITTLE Cluster Temperature. **4** LITTLE Cluster Voltage. **5** GPU Temperature (0). **6** GPU Temperature (1). **7** SoC Temperature. **8** SYSTOP Voltage. **9** SYSTOP Supply Voltage. **10** big Cluster Supply Voltage. **11** LITTLE Cluster Supply Voltage. **12** GPU Supply Voltage.

A.3.23 Get Sensor Value

Default Attributes None.

Table A-16 Sensor ID

Bits	Name	Description
[15:0]	Sensor ID	The sensor being read. Juno R0 only **0** PMIC Temperature. **1** big Cluster Voltage. **2** LITTLE Cluster Voltage. **3** SoC Temperature. **4** SYSTOP Voltage. **5** SYSTOP Supply Voltage. **6** big Cluster Supply Voltage. **7** LITTLE Cluster Supply Voltage. **8** GPU Supply Voltage. Juno R1 only: **0** PMIC Temperature. **1** big Cluster Temperature. **2** big Cluster Voltage. **3** LITTLE Cluster Temperature. **4** LITTLE Cluster Voltage. **5** GPU Temperature (0). **6** GPU Temperature (1). **7** SoC Temperature. **8** SYSTOP Voltage. **9** SYSTOP Supply Voltage. **10** big Cluster Supply Voltage. **11** LITTLE Cluster Supply Voltage. **12** GPU Supply Voltage.

A.3.24 Config Sensor Period Reading

Default Attributes None.

Table A-17 Sensor ID

Bits	Name	Description
[7:0]	Sensor ID	The sensor being configured. Juno R0 only **0** PMIC Temperature. **1** big Cluster Voltage. **2** LITTLE Cluster Voltage. **3** SoC Temperature. **4** SYSTOP Voltage. **5** SYSTOP Supply Voltage. **6** big Cluster Supply Voltage. **7** LITTLE Cluster Supply Voltage. **8** GPU Supply Voltage. Juno R1 only: **0** PMIC Temperature. **1** big Cluster Temperature. **2** big Cluster Voltage. **3** LITTLE Cluster Temperature. **4** LITTLE Cluster Voltage. **5** GPU Temperature (0). **6** GPU Temperature (1). **7** SoC Temperature. **8** SYSTOP Voltage. **9** SYSTOP Supply Voltage. **10** big Cluster Supply Voltage. **11** LITTLE Cluster Supply Voltage. **12** GPU Supply Voltage.

A.3.25 Config Sensor Bounds

Default Attributes None.

Table A-18 Sensor ID

Bits	Name	Description
[7:0]	Sensor ID	The sensor being configured. Juno R0 only **0** PMIC Temperature. **1** big Cluster Voltage. **2** LITTLE Cluster Voltage. **3** SoC Temperature. **4** SYSTOP Voltage. **5** SYSTOP Supply Voltage. **6** big Cluster Supply Voltage. **7** LITTLE Cluster Supply Voltage. **8** GPU Supply Voltage. Juno R1 only: **0** PMIC Temperature. **1** big Cluster Temperature. **2** big Cluster Voltage. **3** LITTLE Cluster Temperature. **4** LITTLE Cluster Voltage. **5** GPU Temperature (0). **6** GPU Temperature (1). **7** SoC Temperature. **8** SYSTOP Voltage. **9** SYSTOP Supply Voltage. **10** big Cluster Supply Voltage. **11** LITTLE Cluster Supply Voltage. **12** GPU Supply Voltage.

A.3.26 Async Sensor Value

Default Attributes None.

There are no additional restrictions on the payload of this command in Juno.

A.3.27 Set Device Power State

Default Attributes None.

Table A-19 Device

Bits	Name	Description
[15:0]	Device ID	Device ID. **0** DEBUGSYS **1** GPU
[23:16]	Power State	Device power state. **0** On **1** RESERVED **2** RESERVED **3** Off

A.3.28 Get Device Power State

Default Attributes None.

Table A-20 Device ID

Bits	Name	Description
[15:0]	Device ID	Identifier of the device being queried. **0** DEBUGSYS **1** GPU

Table A-21 Power State

Bits	Name	Description
[7:0]	Power State	Device power state: **0** On **1** RESERVED **2** RESERVED **3** Off **4 - 255** Device specific

A.4 Boot Over MHU (BOM) Protocol

The maximum size of the SCP RAM firmware image that can be transferred to the SCP is 128KB. A larger image exceeds the RAM capacity of the SCP and causes the SCP to respond with an error during the transfer process.

Appendix B
Revisions

This appendix describes the technical changes between released issues of this book.

Table B-1 Issue A

Change	Location	Affects
First release	-	-

Table B-2 Issue B

Change	Location	Affects
Clarify that document applies to ARM Compute Subsystems only	Throughout	All
128 bytes changed to 128KB	*Boot Over MHU (BOM) Protocol* on page A-16	BOM Protocol