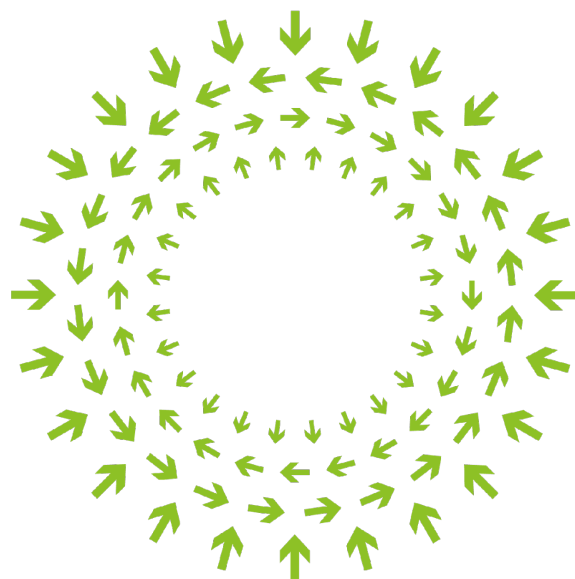




OPEN

Compute Project

Foundation Chiplet System Architecture

1.1.0

Effective October 2026

Authors:
Arm Ltd.
Secure-IC, a Cadence Company

Contents

Foundation Chiplet System Architecture

Version History	vii
License	viii
Open Web Foundation (OWF) CLA	viii
Acknowledgements	ix
Trademarks	x
Compliance with OCP Tenets	xi
Openness	xi
Efficiency	xi
Impact	xi
Scale	xi
Sustainability	xi
Rules-based writing	xii
Content item identifiers	xii
Content item rendering	xii
Content item classes	xii
Additional reading	xiii
1 Introduction	1
1.1 Layered architecture	2
1.2 Purpose	3
1.2.1 Document structure	3
1.3 Using this specification	4
1.3.1 System designer considerations	4
1.3.2 Standalone chiplet designer considerations	4
1.3.3 Chiplet types and classification	4
1.3.4 Interface types and classification	5
1.4 Limitations of scope	6
1.5 Basic definitions	7
2 Rule Compliance Levels	9
2.1 Overview	10
2.1.1 Level 0	10
2.1.2 Level 1	10
2.1.3 Full Compliance	10
2.2 Rule Identifiers by Level	11
3 System types	17
3.1 Compute and Hub System	18
3.1.1 System requirements	20
3.2 Compute Tile System	21
3.2.1 System requirements	23
4 Chiplet types	24
4.1 Compute 1 Chiplet	25

4.1.1	Chiplet requirements	26
4.2	Hub Chiplet	30
4.2.1	Chiplet requirements	31
4.3	Compute 2 Chiplet	36
4.3.1	Chiplet requirements	37
4.4	Fully Coherent Expansion Chiplet	42
4.4.1	Chiplet requirements	43
4.5	Fully Coherent Expansion (Remote Translation) Chiplet	48
4.5.1	Chiplet requirements	49
4.6	I/O Coherent Expansion (Translated) Chiplet	53
4.6.1	Chiplet requirements	54
4.7	I/O Coherent Expansion (Untranslated) Chiplet	58
4.7.1	Chiplet requirements	59
4.8	I/O Coherent Expansion (Remote Translation) Chiplet	62
4.8.1	Chiplet requirements	63
4.9	I/O Chiplet	67
4.9.1	Chiplet requirements	68
4.10	I/O Controller Chiplet	71
4.10.1	Chiplet requirements	72
5	Chiplet interfaces	75
5.1	Common requirements for chiplet interfaces	76
5.2	Chiplet activity states and interface availability	77
5.3	Hub Chiplet to Hub Chiplet Interface requirements	79
5.3.1	Boot Sequence Control Interface	79
5.3.2	Boot Code Image Load Interface	79
5.3.3	Coherent Memory Traffic Interface	80
5.3.4	Interrupt handling	80
5.3.5	Security	80
5.3.6	System control communication	81
5.3.7	Debug	81
5.4	Hub Chiplet to Compute Chiplet Interface requirements	82
5.4.1	Boot Sequence Control Interface	82
5.4.2	Boot Code Image Load Interface	82
5.4.3	Coherent Memory Traffic Interface	83
5.4.4	Interrupt handling	83
5.4.5	Security	83
5.4.6	System control communication	84
5.4.7	Debug	84
5.5	Compute 1 Chiplet to Compute 1 Chiplet Interface requirements	85
5.5.1	Boot Code Image Load Interface	85
5.5.2	Coherent Memory Traffic Interface	85
5.5.3	Interrupt handling	86
5.5.4	Security	86
5.5.5	System control communication	86
5.5.6	Debug	87
5.6	Compute 2 Chiplet to Compute 2 Chiplet Interface requirements	88
5.6.1	Boot Sequence Control Interface	88
5.6.2	Boot Code Image Load Interface	88
5.6.3	Coherent Memory Traffic Interface	89
5.6.4	Interrupt handling	89
5.6.5	Security	89
5.6.6	System control communication	90
5.6.7	Debug	90
5.7	Fully Coherent Expansion Chiplet Interface requirements	91

5.7.1	Boot Sequence Control Interface	91
5.7.2	Boot Code Image Load Interface	91
5.7.3	Coherent Memory Traffic Interface	92
5.7.4	Interrupt handling	92
5.7.5	Device Configuration Interface	92
5.7.6	Security	93
5.7.7	System control communication	93
5.7.8	Debug	93
5.8	Fully Coherent Expansion (Remote Translation) Chiplet Interface requirements	95
5.8.1	Coherent Memory Traffic Interface	95
5.8.2	I/O Coherent Memory Traffic Interface	95
5.8.3	Address Translation Interface	96
5.8.4	Interrupt handling	96
5.8.5	Device Configuration Interface	96
5.8.6	Security	97
5.8.7	System control communication	97
5.8.8	Debug	98
5.9	I/O Coherent Expansion (Translated) Chiplet Interface requirements	99
5.9.1	Boot Sequence Control Interface	99
5.9.2	Boot Code Image Load Interface	99
5.9.3	I/O Coherent Memory Traffic Interface	100
5.9.4	Non-Coherent Memory Traffic Interface	100
5.9.5	Interrupt handling	100
5.9.6	Device Configuration Interface	101
5.9.7	Security	101
5.9.8	System control communication	101
5.9.9	Debug	102
5.10	I/O Coherent Expansion (Untranslated) Chiplet Interface requirements	103
5.10.1	Boot Sequence Control Interface	103
5.10.2	Boot Code Image Load Interface	103
5.10.3	Untranslated I/O Coherent Memory Traffic Interface	104
5.10.4	Interrupt handling	104
5.10.5	Device Configuration Interface	104
5.10.6	Security	105
5.10.7	Debug	105
5.11	I/O Coherent Expansion (Remote Translation) Chiplet Interface requirements	106
5.11.1	I/O Coherent Memory Traffic Interface	106
5.11.2	Address Translation Interface	106
5.11.3	Non-Coherent Memory Traffic Interface	107
5.11.4	Interrupt handling	107
5.11.5	Device Configuration Interface	107
5.11.6	Security	108
5.11.7	System control communication	108
5.11.8	Debug	108
5.12	I/O Chiplet Interface requirements	110
5.12.1	Boot Sequence Control Interface	110
5.12.2	Boot Code Image Load Interface	110
5.12.3	Untranslated I/O Coherent Memory Traffic Interface	111
5.12.4	Non-Coherent Memory Traffic Interface	111
5.12.5	Interrupt handling	111
5.12.6	Device Configuration Interface	112
5.12.7	Security	112
5.12.8	Debug	112
5.13	I/O Controller Chiplet Interface requirements	113
5.13.1	Boot Sequence Control Interface	113

5.13.2	Boot Code Image Load Interface	113
5.13.3	Non-Coherent Memory Traffic Interface	114
5.13.4	Interrupt handling	114
5.13.5	Device Configuration Interface	114
5.13.6	Security	115
5.13.7	Debug	115
6	Common chiplet components	116
6.1	Chiplet system component boot	117
6.1.1	Chiplet system component boot stages	117
6.1.2	Chiplet system component boot sequence signaling	117
6.1.3	Chiplet system component boot sequence	120
6.2	System Control Framework	125
6.2.1	System Controller	125
6.2.2	System Control Agent	128
6.3	Trusted Security Agent	129
6.3.1	Boot sequencing of trusted security agents	130
6.3.2	Handling boot code images	130
6.3.3	Trusted security agent interrupts	131
7	Chiplet interface implementation	132
7.1	Hardware fault requirements for chiplet interfaces	133
7.2	Interface protocols	134
7.2.1	Boot Sequence Control Interface	134
7.2.2	Boot Code Image Load Interface	134
7.2.3	Coherent Memory Traffic Interface	134
7.2.4	Untranslated I/O Coherent Memory Traffic Interface	134
7.2.5	I/O Coherent Memory Traffic Interface	134
7.2.6	Non-Coherent Memory Traffic Interface	134
7.2.7	Device Configuration Interface	135
7.2.8	Address Translation Interface	136
7.2.9	Interrupt handling	136
7.2.10	Trusted Security Agent Communication Interface	136
7.2.11	System Control Interface	136
7.2.12	Debug	136
7.3	Interface transports	137
7.3.1	Boot Sequence Control Interface	137
7.3.2	Boot Code Image Load Interface	137
7.3.3	Coherent Memory Traffic Interface	137
7.3.4	Untranslated I/O Coherent Memory Traffic Interface	137
7.3.5	I/O Coherent Memory Traffic Interface	138
7.3.6	Non-Coherent Memory Traffic Interface	138
7.3.7	Device Configuration Interface	138
7.3.8	Address Translation Interface	140
7.3.9	Interrupt handling	140
7.3.10	Trusted Security Agent Communication Interface	140
7.3.11	System Control Interface	141
7.3.12	Debug	141
8	Implementation constraints	142
8.1	Interoperability of chiplets	143
8.1.1	Memory system	143
8.2	Chiplet system scalability	144
8.2.1	Memory system	144

9	Security of chiplet systems	145
9.1	Chiplet system security model	147
9.1.1	Assumptions	147
9.1.2	Stakeholders	149
9.1.3	Security domains and trust boundaries	149
9.1.4	Chiplet system security platform and Root of Trust framework	151
9.1.5	Chiplet system lifecycle	154
9.1.6	Chiplet system topology models	155
9.1.7	Adversarial models	158
9.1.8	Chiplet system security goals	160
9.2	Threats and mitigations	161
9.2.1	Use model	161
9.2.2	Risk assessment method	161
9.2.3	Threat analysis	163
9.2.4	Implementation of mitigations	183
	Glossary	189

Version History

Date	Version #	Author	Description
19 Nov 25	1.0.0 Draft 0	Arm Ltd.	First release.
22 Jan 26	1.0.0 Draft 1	Arm Ltd.	Updated System Control Description. Relaxed Rules on UCle. Relaxed Rules on CHI-C2C.
27 Jan 26	1.0.0 Draft 2	Arm Ltd.	Added trademarks. Corrected document style.
10 Feb 26	1.0.0	Arm Ltd.	Release of 1.0.0.
23 Jul 26	1.1.0 Alpha 0	Arm Ltd.	Release of 1.1.0 Alpha 0. New security section. Expanded trademarks list. Expanded additional reading list. New section on implementation constraints. Added interconnect interoperability and snoop-filter capacity requirements. Updated boot sequencing of common components. Specified I3C transports for boot images and security-agent communication. Added memory-mapped debug access and interface availability rules. Clarified memory serialization and MMU placement requirements. Required PCIe Endpoint presentation for remote-translation devices. Added device-configuration protocol and transport entries. Removed the Compute 1 system-control-agent requirement. Limited mandatory Hub boot-image storage to the primary Hub. Strengthened boot-image transfer capabilities at Full compliance.
01 Sep 26	1.1.0 RC 0	Arm Ltd.	Release of 1.1.0 RC 0. Added acknowledgements section. Incorporated feedback from reviewers of security section.
09 Oct 26	1.1.0	Arm Ltd.	Release of 1.1.0.

License

Open Web Foundation (OWF) CLA

Contributions to this Specification are made under the terms and conditions set forth in **Modified Open Web Foundation Agreement 0.9 (OWFa 0.9)**. (As of October 16, 2024) (“Contribution License”) by:

Arm Ltd.

Secure-IC, a Cadence Company

Usage of this Specification is governed by the terms and conditions set forth in **Modified OWFa 0.9 Final Specification Agreement (FSA)** (As of October 16, 2024) (“**Specification License**”).

You can review the applicable Specification License(s) referenced above by the contributors to this Specification on the OCP website at <https://www.opencompute.org/contributions/templates-agreements>.

For actual executed copies of either agreement, please contact OCP directly.

Notes:

The above license does not apply to the Appendix or Appendices. The information in the Appendix or Appendices is for reference only and non-normative in nature.

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Acknowledgements

The Contributors of this Specification would like to acknowledge the following members of the Foundation Chiplet System Architecture (FCSA) workstream in the Open Chiplet Economy (OCE) subproject of the Open Compute Project (OCP) Server Project for their feedback:

- Cadence Design Systems, Inc.
- Giridhar Mandyam, AMD
- Guillaume Leterrier, Valeo
- PUFsecurity
- Robert Pelt, AMD
- Sohrab Aftabjahani, Intel

Trademarks

AMBA® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

AMD® is a registered trademark of Advanced Micro Devices, Inc.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cadence® is a registered trademark of Cadence Design Systems, Inc.

CXL® is a registered trademark of Compute Express Link Consortium, Inc.

GlobalPlatform® is a registered trademark of GlobalPlatform, Inc. in the US and/or elsewhere.

HDMI® is a registered trademark of HDMI Licensing Administrator, Inc.

I3C® is a registered trademark of MIPI Alliance, Inc.

Intel™ is a trademark of Intel Corporation.

Microsoft® is a registered trademark of the Microsoft group of companies.

MIPI CSI-2® is a registered trademark of MIPI Alliance, Inc.

PCIe® and PCI Express® are registered trademarks of PCI-SIG.

PSA Certified™ is a trademark of GlobalPlatform, Inc.

PUFsecurity™ is a trademark of PUFsecurity in Taiwan and/or in other countries.

RISC-V® is a registered trademark of RISC-V International.

Trusted Computing Group® is a registered trademark of the Trusted Computing Group (TCG).

UCIe™ is a trademark of UCIe (Universal Chiplet Interconnect Express).

Valeo® is a registered trademark of Valeo SE.

All other brands or product names are the property of their respective holders.

Compliance with OCP Tenets

This section describes how this specification complies with the OCP tenets.

For a full explanation, see [OCP Tenets Explained](#).

Openness

The FCSA is licensed under the Open Web Foundation Agreement (OWFa) version 0.9. Feedback and contributions are encouraged from the OCP community, and the specification is reviewed and developed under the supervision of the public Foundation Chiplet System Architecture workstream, operating under the Open Chiplet Economy Sub-Project.

Efficiency

The FCSA enhances efficiency by encouraging and facilitating the reuse of silicon dies in multiple systems and packages, bringing economies of scale to system-on-chip (SoC) and system-in-package (SiP) development and manufacturing processes, and reducing the cost and time to market for custom semiconductor solutions.

Impact

The FCSA provides a new baseline for the chiplet market, empowering the industry to build interoperable chiplets across domains. The FCSA defines common system topologies, standard chiplet types, and functional and interface requirements between chiplets to provide a common foundation that maximizes interoperability and reduces fragmentation across the industry.

Scale

Chiplet systems enable common dies to be used in different configurations and topologies to scale performance while retaining an acceptable yield. The FCSA supports features required in large-scale deployments, including error reporting, manageability, and advanced security technologies.

Sustainability

The FCSA embraces sustainability by reducing development and manufacturing costs for advanced custom semiconductor systems.

Rules-based writing

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

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

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

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

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

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

Content item identifiers

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

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

Content item rendering

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

- L is a label that indicates the content class of the content item.
- $iiii$ is the identifier of the content item.

Content item classes

Declaration

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

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

A Declaration does not describe behavior.

A Declaration is rendered with the label D .

Rule

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

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

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

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

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

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

A Goal is rendered with the label *G*.

Information

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

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

Rationale

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

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

Implementation note

An Implementation note provides guidance on implementation of the specification.

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

Software usage

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

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

Additional reading

- [1] *Open Compute Project*. See <https://www.opencompute.org/>
- [2] *Foundation Chiplet System Architecture*. See <https://www.opencompute.org/wiki/Server/OCE/FCSA>
- [3] *Open Chiplet Economy*. See <https://www.opencompute.org/community/open-chiplet-economy>
- [4] *AMBA 5 CHI Architecture Specification*. (ARM IHI 0050) Arm.
- [5] *Arm CoreSight Architecture Specification, v3.0*. (ARM IHI 0029F) Arm.
- [6] *PCI Express Base Specification*. PCI-SIG.
- [7] *Universal Chiplet Interconnect Express (UCIe) Specification Revision 1.1*. UCIe.
- [8] *MIPI Alliance Specification for I3C*. MIPI.
- [9] *Arm AMBA CHI Chip-to-Chip Specification*. (ARM AES 0071) Arm.

- [10] *Arm System Control and Management Interface Platform Design Document*. (ARM DEN 0056) Arm.
- [11] *RISC-V Platform Management Interface Specification (RPMI)*. RISC-V International.
- [12] *Bunch of Wires (BoW) PHY Specification*. Open Compute Project (OCP).
- [13] *Root of Trust Definitions and Requirements, v1.1*. GlobalPlatform.
- [14] *PSA Certified Platform Security Model*. GlobalPlatform.
- [15] *Caliptra: A Datacenter System on a Chip (SoC) Root of Trust (RoT)*. See <https://chipsalliance.github.io/Caliptra/doc/Caliptra.html>
- [16] *OCP S.A.F.E. Program*. See <https://www.opencompute.org/community/ocp-safe-program>
- [17] *Seven properties of Highly Secure Devices*. Microsoft.
- [18] *IDE and TDISP: An Overview of PCIe Technology Security Features*. See <https://pcisig.com/blog/ide-and-tdisp-overview-pcie%C2%AE-technology-security-features>
- [19] *NIST Special Publication 800-30 Revision 1: Guide for Conducting Risk Assessments*. NIST.
- [20] *NIST Special Publication 800-218: Secure Software Development Framework (SSDF) Version 1.1*. NIST.
- [21] *NIST Special Publication 800-161 Revision 1: Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations*. NIST.
- [22] *DICE Architectures*. Trusted Computing Group.
- [23] *Security Protocol and Data Model (SPDM) Specification*. (DSP0274) DMTF.
- [24] *NIST Special Publication 800-193: Platform Firmware Resiliency Guidelines*. NIST.
- [25] *AMBA AXI Protocol Specification*. (ARM IHI 0022) Arm.
- [26] *Compute Express Link (CXL) Specification*. CXL.
- [27] *IEEE Standard Test Access Port and Boundary-Scan Architecture*. (ISBN 0-7381-2945-3) IEEE.

1

Introduction

Designers use chiplets instead of traditional monolithic dies to overcome reticle limits, improve yield, reuse designs, enable multiple product configurations, and optimize technology choices for cost and applicability.

Chiplets can partition monolithic designs or aggregate additional functionality from devices that traditionally use separate packages into a shared package. The Foundation Chiplet System Architecture (FCSA) applies to a monolithic system design that is partitioned across chiplets and in which chiplet interfaces replace on-die connections. For partitioned systems, the FCSA defines common partitioning approaches. Mature system architectures already exist for aggregating peripheral devices that use established standards, for example, Peripheral Component Interconnect Express (PCIe), into a shared package. The FCSA adds limited additional value to this aggregation approach. Both partitioning and aggregation are important for building chiplet-based systems.

This Foundation Chiplet System Architecture document specifies a hardware system architecture for a system that is distributed across multiple chiplets. A *system* is the hardware that supports a single system software image, for example, an operating system (OS).

The FCSA identifies high-level properties for system topologies and their constituent chiplet types. It specifies the chiplet types required to build these systems, their properties, and their interfaces. Separately, it specifies protocol and transport mechanisms for each interface requirement.

1.1 Layered architecture

This architecture specification is part of the layered system architecture framework shown in [Figure 1.1](#). Each higher layer extends the lower layers to form a more complete specification for a particular design or product profile.

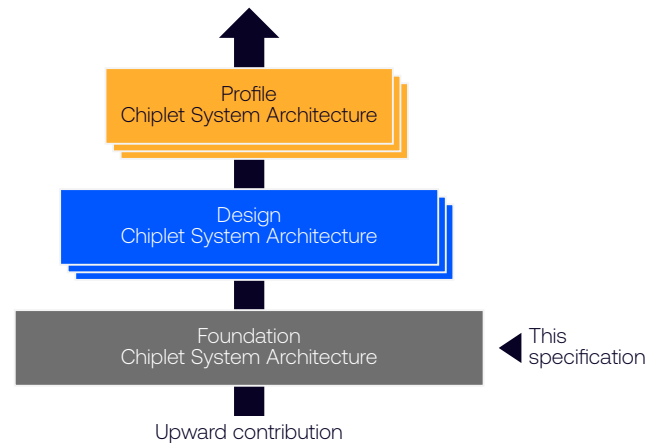


Figure 1.1: Layered architecture

This specification is in the foundation layer. The layers are:

Foundation

This specification.

The Foundation Chiplet System Architecture (FCSA) aligns the system architecture by specifying basic requirements and common concepts that are not specific to a design or product profile.

The FCSA is a contribution to the Open Compute Project (OCP) [1]. For more information or to contribute to the FCSA, see the FCSA workstream [2] of the Open Chiplet Economy OCP subproject [3].

Design

A design specification extends the FCSA with requirements that align the system architecture with a particular system design. For example, a chiplet system architecture can be specific to an instruction set architecture (ISA).

Profile

A profile specification extends a design specification for a particular product profile. A profile can define a market segment, a deployment environment, or a narrowly focused functional extension.

Higher-level specifications incorporate the contributions of lower layers and add requirements. Because this document is rules-based, the rule set of each higher-level specification is a superset of the rule sets in the lower-level specifications that it extends.

1.2 Purpose

This specification provides a foundation for a chiplet ecosystem based on the reuse of components and design effort in chiplet-based systems, including the reuse of complete chiplets.

This ecosystem requires an agreed classification of chiplet types to ensure that features and interfaces are compatible. This classification provides a foundation for more layers of standardization, including protocols, physical interface definitions, and reusable chiplet products.

During chiplet design, designers make implementation-specific choices about how to implement system functionality, for example, interrupts or direct memory access (DMA). When these choices are arbitrary and nondifferentiating, the FCSA provides a consistent implementation. This consistency enables chiplet compatibility without limiting differentiation.

1.2.1 Document structure

The FCSA specifies chiplet system compositions, chiplet types, and interface definitions.

The following figure shows the relationship between the components specified in the FCSA.

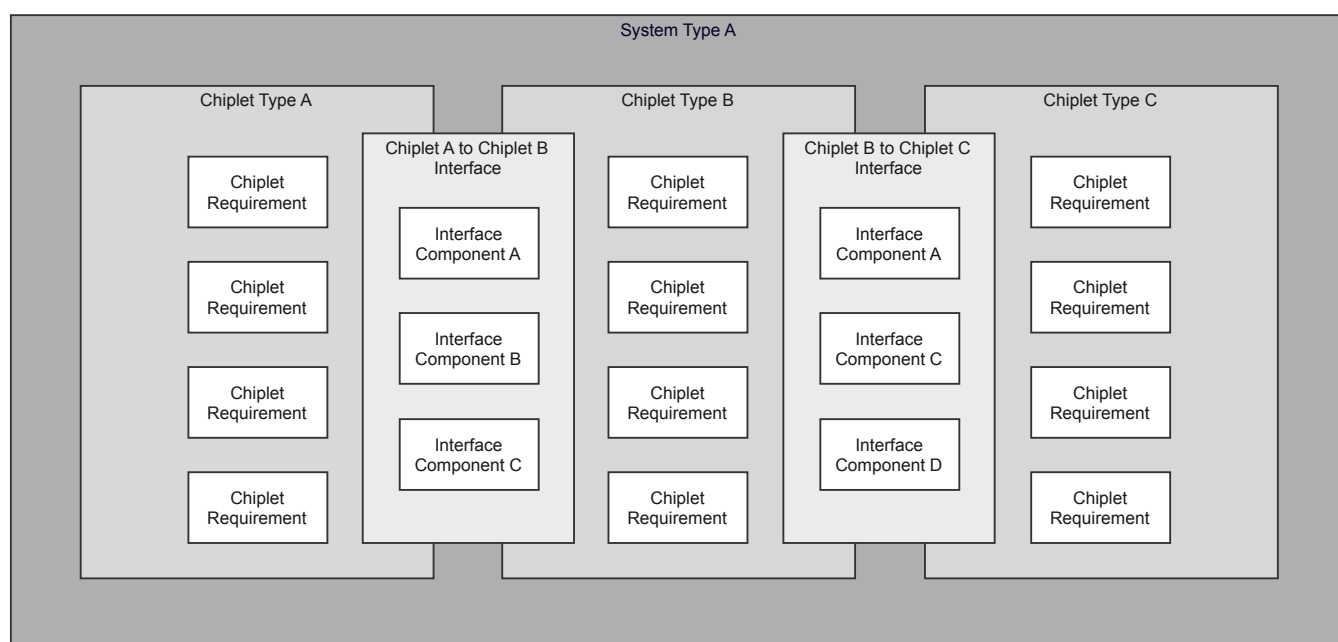


Figure 1.2: FCSA component structure

Each chiplet in a system has a defined type. The FCSA specifies a valid composition of chiplet types for the system.

A chiplet type specifies the properties that fulfill the requirements of the chiplet within the system. These properties typically affect the interface requirements between the chiplet and other chiplets in the system.

A chiplet interface specifies the properties of the connections between two chiplets of the same or different types. A chiplet interface contains multiple interface components. Each component describes the properties of an interface requirement.

Where possible, a chiplet interface component maps to an implementation that specifies a protocol, a transport mechanism, and any other requirements. The FCSA specifies protocols and transport mechanisms when appropriate industry standards exist. Otherwise, it maps the interface component as TO BE DEFINED.

1.3 Using this specification

The sections that are relevant to you depend on your role. For example, you might be a system designer who partitions or creates a chiplet system. You might instead be a standalone chiplet designer within the same organization as the system designer or working for a third party.

Although the FCSA does not cover all solutions for connecting chiplets, it covers key functional interactions and interfaces between chiplets.

1.3.1 System designer considerations

A system designer typically creates a system in which some chiplets are reusable across product configurations or product generations. The system designer might source some chiplets from standalone chiplet designers.

For a system designer:

- Identify the system type that best matches the requirements. Consider:
 - How chiplet partitioning aligns with system requirements
 - Product configurations enabled by chiplet combinations, including:
 - * Different product scales, for example, different amounts of compute capacity
 - * Different features, for example, different acceleration engines
- Review the component chiplet requirements to identify the number and types of chiplets that best match the partitioned functionality. Include the chiplets required for a minimally functional system and any other required or optional chiplet types. Other required types can provide scale, and optional types can add differentiating functionality. Consider:
 - The effects of chiplet size on reticle limits and yield
 - Technology cost and applicability
 - Potential for reuse from previous designs or in future designs
 - Separation of specialist functionality
 - Availability of third-party chiplets as products or design services

Use these and other factors to select the number and types of chiplets in the system.

Use the selected chiplet types as a base specification for the system and chiplet designs, or cite them in a Request for Quote (RFQ) for a third-party chiplet designer.

1.3.2 Standalone chiplet designer considerations

A standalone chiplet designer creates a chiplet for use in other systems. The chiplet can be designed for a specific system or for integration into multiple systems.

For a chiplet designer:

- Use the system types to understand how the intended chiplet functionality integrates with different systems.
- Identify the chiplet requirements from internal specifications or external RFQ requirements.
- Select the appropriate chiplet type based on its capabilities and interface requirements, or as specified by the RFQ.

Use the selected chiplet type as a base specification for the chiplet design.

This process helps ensure that the standalone chiplet design includes the required functionality and interfaces to connect to a system that follows the FCSA.

1.3.3 Chiplet types and classification

An FCSA chiplet type defines requirements for on-chiplet functionality and a mandatory set of interfaces. The interface set does not prevent a chiplet from including additional interfaces. These requirements ensure that the chiplet can form a functional system with other chiplets of the system type.

A chiplet with additional capabilities can still be classified as a chiplet type that requires less functionality if it contains the required functions. Therefore, a chiplet can be classified as more than one type. Integration-specific configuration of the interfaces might be necessary to manage certain functions and ensure correct behavior when the chiplet is used as the specified type.

Supporting multiple chiplet types can increase the flexibility of a chiplet and enable its use in more than one system. However, this approach can be suboptimal in a specific deployment because functionality for chiplet types that are not in use remains unused. This unused functionality can waste silicon area and power.

1.3.4 Interface types and classification

An FCSA chiplet interface type defines requirements that an implementation must meet. The chiplet types being connected determine these requirements. The requirements ensure that the defined chiplet types can interact and form a functional system.

An interface can interact with multiple chiplet types if the interface and chiplets support the required functions. In such cases, a single interface can be classified as more than one type. The interface might require configuration during manufacturing or at runtime to support a connection to a specific chiplet type.

Supporting multiple interface types increases the flexibility of both the interface and the chiplet. However, this approach can be suboptimal in a specific deployment because functionality for unused interface types can waste silicon area and power.

1.4 Limitations of scope

Hardware that is not tied to a single system is outside the scope of the FCSA. A system that complies with the FCSA can use functions provided by this hardware. However, the FCSA does not specify rules for the hardware. Examples include:

- Aggregations of off-die devices and accelerators that a device driver model discovers and abstracts, for example, industry-standard peripheral devices
- A chiplet that is fully abstracted from the system, for example, a memory chiplet standardized by the JEDEC Solid State Technology Association (JEDEC)

Integration between separate systems is outside the scope. For example, a server compute chiplet and a Data Processing Unit (DPU) chiplet can each form a separate system. The FCSA can apply separately to each system.

The following figure shows the scope for single and composite chiplets.

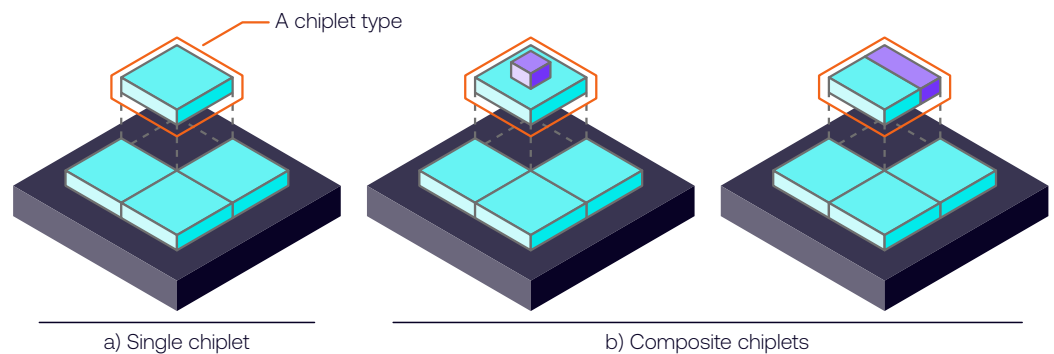


Figure 1.3: Examples of single and composite chiplets

The FCSA defines some chiplet types. When a chiplet is a composite structure of multiple dies, the type definition applies to the entire composite structure that is integrated as a unit into the system, as shown in [Figure 1.3](#). The component dies function only in combination and are likely to be co-designed. Individual component dies are outside the scope of the FCSA. Examples include component dies that provide only cache memory, an interface physical layer (PHY), or analog circuitry and that have closely coupled dependencies on other chiplets in the composite.

1.5 Basic definitions

The following terms and concepts are used throughout the FCSA.

System

The hardware that supports a single system software image of an operating system or hypervisor.

Application PE

A Processing Element (PE) that runs the operating system.

NOTE: Other processing elements might exist in the system but not run the main system software. Examples include processing elements that run security or system control firmware. In FCSA terminology, these processing elements are not Application PEs.

Fully Coherent Processing Device

A device whose primary function is to transform, analyze, generate, encode, decode, or otherwise process data, for example, an accelerator for a particular type of computation. The device contains coherent caches and makes coherent accesses to the rest of the system.

I/O Coherent Processing Device

A device whose primary function is to transform, analyze, generate, encode, decode, or otherwise process data, for example, an accelerator for a particular type of computation. The device supports I/O coherency. It has no coherent cache and makes coherent accesses to the rest of the system.

I/O Coherent I/O Device

A device whose primary function is to provide I/O. The device consists of a controller and an interface that supports I/O coherency. Neither the device nor devices connected to the interface have a coherent cache. The device makes coherent accesses to the rest of the system.

Non-Coherent I/O Device

A device whose primary function is to provide I/O. The device consists of a controller and an interface that accesses only Non-cacheable system memory.

System Addressable Memory

All agents in the system share a common physical address map. The FCSA refers to this map as *System Addressable Memory*. The FCSA assumes that all Application PEs in the system share the same operating system or hypervisor.

System Main Memory

Memory that is both:

- Contained in a chiplet, either integrated or directly connected
- Allocatable by the OS or hypervisor

This definition also aligns with the Advanced Microprocessor Bus Architecture (AMBA) Coherent Hub Interface (CHI) [4] definition: “the memory that holds the data value of an address location when no cached copies of that location exist.”

Typical examples include external dynamic random-access memory (DRAM) and high-bandwidth memory (HBM).

Cacheable system main memory

Locations in *System Main Memory* that are allowed to be allocated in a cache.

Non-cacheable system main memory

Locations in *System Main Memory* that are not allowed to be allocated in a cache.

Boot Code Image

An image of firmware for one of the boot stages of a [Trusted Security Agent](#), [System Controller](#), or Application PE.

System Debug Agent

A software or hardware component that facilitates debugging of the system as defined in this section. A *private* debug agent differs because its scope is limited to a subsystem.

2

Rule Compliance Levels

I_NDLSV This section describes the levels of compliance within the FCSA and how they can be applied.

2.1 Overview

The Foundation Chiplet System Architecture (FCSA) embeds the notion of levels of compliance. Each level adds functionality over and above a previous level. Unless explicitly stated, all specification items that belong to level N apply to levels greater than N.

A chiplet or system can claim compliance with a level when it adheres to all the rules for that level and all lower levels.

Individual rules are marked with their levels, as in the example below:

R_{YYYY}

This is an example of an FCSA rule that is assigned to level N.

(*LEVEL-N*)

Rules without an indicated level needs to adhered to if the chiplet or system wants to reach full compliance to the FCSA.

2.1.1 Level 0

This level means the chiplet meets the basic functional and interface rules for the chiplet or system type.

A list of the level 0 rule identifiers that appear in the FCSA is provided in [Table 2.1](#).

2.1.2 Level 1

In addition to the requirements in level 0, for compliance to level 1 the design meets additional requirements appropriate for the chiplet type or system.

A list of the level 1 rule identifiers that appear in the FCSA is provided in [Table 2.2](#).

2.1.3 Full Compliance

This indicates full alignment to the FCSA requirements for the specified chiplet or system.

A list of the level Full rule identifiers that appear in the FCSA is provided in [Table 2.3](#).

2.2 Rule Identifiers by Level

Table 2.1: Level 0 Rule Identifiers in the FCSA

3 System types							
3.1 Compute and Hub System							
LYHKS	TFJSM	LCJNF	CJZHL	ZCFRY	PPLNQ	MDPNF	FZVCL
DXYWQ	TZJGV	ZKZHH					
3.2 Compute Tile System							
TKPTF	MVWBD	JHWVM	DSKBW	ZFTDV	BFSJV	ZDVGZ	CCHPT
TTPHX	WTRKM						
4 Chiplet types							
4.1 Compute 1 Chiplet							
JWRCG	BPRJK	RCYDP	XTDHG	PTZSC	NCHRQ	KNKSV	NRJZH
NGZDK	ZHHPH	HQXHS	XGXZR	MCLJS	CQDKD	GHRFJ	ZYTGO
TGSXZ	MJNJG	GVVMS	ZDSHM	DZFBZ	RNSKY	PGGRB	XSFFH
SFPHM	MDGWP	RCRRQ	WDJHV	DKSBZ	MRRGT		
4.2 Hub Chiplet							
MZLJX	CNQZH	LCQGC	SVWVP	HBYND	MKYNH	WMTVP	MBQZT
MLQLD	ZTCKN	PCJMB	FXFNL	JDYJY	PMKFN	YHQDR	XDGOV
TZRSV	RXLZG	MTBPW	VJFWY	KZFYG	RGMKW	GNJLK	GSSWV
FTXZM	ZPLGW	VNGDP	NJQDV	NXDVW	CKXYX	ZFWWC	PYRRZ
BTGJS	ZCPHB	LDSVP	GYSJB	SXNDJ	TVKHG	CJMRD	WTGSM
YJBND	KYYNQ	DZGQK					
4.3 Compute 2 Chiplet							
VKGJK	SJNGN	NDHDT	KNHMM	CGKQW	MDFWD	WLHVZ	JZSRC
DVNHD	QGGWQ	LGXLN	XNYDJ	PHHQM	ZVZZN	VYJFL	ZCLCN
DMSQX	QSXVN	XCVGN	ZXWKH	DZDHQ	PBYQY	VXGYS	CLYNB
GPSSB	TXVCM	THPPG	NYZVM	YNMLX	SGZXK	VVDFFD	NLGDR
WVMSK	PGTMQ	YQMHG	FDRRV	CNDVJ	FWCJG	KGFNL	GNZNV
ZDBLR	XFSNZ	NSRSQ	ZHVKB	XDDFR	XYVWT	LPTTQ	
4.4 Fully Coherent Expansion Chiplet							

SHXTT	HXGDG	GHKCL	MKCTS	HMZJQ	YDWVV	NBXWR	FMBVG
JNCKH	VCPQR	FJDNP	SPTFW	FQPML	QDMPX	LYVBX	LVJXD
LFYRB	XVWQF	WHJVN	QQTXS	RDGJV	PBNCH	FDDHQ	FXKVK
ZRPHF	QZSYG	PWHNJ	RNNVV	ZNXXR	RJMNL	XLWKP	FFWWY
BMPMM	ZDSMN	YSNYC	XRKQP	KHQGY	VJGGP		

4.5 Fully Coherent Expansion (Remote Translation) Chiplet

KGJQZ	VZRNQ	ZKKWN	VGSNV	KSDBW	MHJBP	GGJVV	WDGDV
KLQNT	WNHGS	SBTVJ	VHJYG	SVMSC	FFDXH	XMDFB	LMWRD
DDBSL	PJVLK	RVBSS	CWKWQ	NYNCY	MBSCP	XVZTS	JSDXW
BVKJT	XYXHC	PGHRF	XBJSZ				

4.6 I/O Coherent Expansion (Translated) Chiplet

YTTPM	MPKFL	LGNWX	RWJXM	NJQMS	FHVQW	XPQFB	DFJJB
LRYFY	NKBDT	GPNPF	JKNBM	YLRSX	TXQKB	PGQRS	DVDHL
XCBFQ	HSWKR	BGYTR	XHLRJ	RDSRT	ZHDSR	LPFHQ	SXYNL
SDNYR	TWRHN	ZHTQD	PHQVC	LCWDK	JTTCQ	ZKGSN	HQHCV
QGKMH	RBRHD	DDQVP					

4.7 I/O Coherent Expansion (Untranslated) Chiplet

YWWFY	GNBSW	PZFMT	WFLXB	JHQPD	BNCHJ	DGKSX	HKCGS
YHDSL	DPLFM	NDGMQ	RGMBN	NPHFD	KFPMT	PQGLG	RBSQV
KMTCS	JMJYS	HHJCC	MGPTM	PJLCZ	MLLGS	HHNSD	NJLVJ
WSWRW							

4.8 I/O Coherent Expansion (Remote Translation) Chiplet

MCTLV	JFVWW	LYDWT	LVZJJ	YKSJV	DFKTT	XKHNG	RXVFP
SJJRK	VGYHF	GXVLS	JPCFZ	MGPMV	MJDDS	VLFCJ	GNNND
JWZKZ	SSKPN	XCbfd	RHJMG	PWMCT	VVWXF	MTJGB	WXTJK
NTMSD	SSBPB	VDGNN	DZVCK				

4.9 I/O Chiplet

FKNWS	WWGKL	HCYCZ	CFTXC	FHZFP	ZQPNC	JHMFZ	KWXTX
CPTMG	ZMHCY	RTPGY	YRKBX	JSKVP	RRCYS	MGCKS	LCHML
XJFYN	WPDRB	BNDMC	YFNFR	WYPZL	GWHVL	LMZZV	BXGXT

4.10 I/O Controller Chiplet

FFNKP	LTWZF	JWNXJ	RDQYR	DSBQD	WBVMW	XWTVX	BNQNG
JRSRT	RZGDN	QWLMP	LZRDM	MWPHD	DWLLK	QMDXK	MDBRR
KZQVC	PDQHN	GRYYV	JDSWQ	NHPSD	JHKVK	YNBXR	DJFVK
VVBCX							

5 Chiplet interfaces

5.3 Hub Chiplet to Hub Chiplet Interface requirements

KNVZY	SYWLF	SYDRX	LQZMW	DFVLC	CQVLD	DHBHJ	TPSJG
DKMDM	RVKPZ	NJPCS	PMLBM	BMRTC	VWWPY		

5.4 Hub Chiplet to Compute Chiplet Interface requirements

JQQPR	YQMLC	FVFQQ	GHLHT	CLWWD	GMZTH	NNQCD	FXRMW
SZZGL	YGWKV	BHDVN	HJZYS	HZCKZ	CZKJG		

5.5 Compute 1 Chiplet to Compute 1 Chiplet Interface requirements

LPBNG	KYGBK	SDKFD	JSDGJ	KFDDX	YYBCT	LPQPB	SSDSP
FZBGN	SCNVZ	MFVBR	MRDBH				

5.6 Compute 2 Chiplet to Compute 2 Chiplet Interface requirements

DNJVQ	SVNXJ	RBNSS	CZLQD	NLNCD	JRLRN	TXHMX	LLTTW
GZMKP	RTCSK	PDJZG	NLZFW	QLHKH	NCKXH		

5.7 Fully Coherent Expansion Chiplet Interface requirements

LNKBT	ZTXLR	VLQPR	RBMCH	WGSPV	PFZGF	GJBYM	JZNWW
JZKYZ	RVDHX	ZCQQH	GHGXC	GQTVW	LJRBQ	FZVDD	BHWTC

5.8 Fully Coherent Expansion (Remote Translation) Chiplet Interface requirements

CMRXD	TGCLP	QSHDG	SXWZM	VBVKL	LHSRL	LWJBR	HYTDS
GLVZD	LXTVM	GRXQH	CSFZB	GGTMJ	QBXFV	GQNNR	XXTQB
PCWPV	DXKQF						

5.9 I/O Coherent Expansion (Translated) Chiplet Interface requirements

MWPML	VTDRN	HCYKF	HXNJH	RGQKJ	YHSGY	JSFTT	BMJZP
RNYKY	XRDCX	ZGDGY	JJKSQ	BHRHJ	NTZHB	SQLJT	CYHBV
TKMBQ							

5.10 I/O Coherent Expansion (Untranslated) Chiplet Interface requirements

NTXZD	XGDKV	CBWZD	SZXHJ	QBQMD	DTDLR	CDBPS	RYVZX
QTJWG	BSQPC	MCCDT	PVDQS	PDJPV	JJPDD	ZGBHJ	

5.11 I/O Coherent Expansion (Remote Translation) Chiplet Interface requirements

HHPXJ	YQSDV	MTLNR	SVYMX	NMQCH	ZLKGN	QRJPN	RVRZS
LFDBC	ZTFJZ	BNLPG	LGJSW	VQBC T	FQQRQ	QXZDW	VPNZQ
SDPSD	ZKCGC						

5.12 I/O Chiplet Interface requirements

CTTSY	NHLLR	LHRDK	WLLKW	XKHXM	DVBZV	RSJDZ	DFQYV
YLRCM	HJDFT	RVTJM	KBGTC	HXWGL	HHJRZ	TDQTB	HLPGX

5.13 I/O Controller Chiplet Interface requirements

ZBNTF	DJCGG	DDDSW	YZHWP	YJGHH	LSJMB	PCWSQ	ZQV FY
YRHHX	HLXNC	LKCWK	XZPFB	RNDMS	NRHNZ		

6 Common chiplet components

6.2 System Control Framework

TMGXQ	NHBRK	CSHRH	ZGHZS	FCBMQ	NQZFH	FLDNS	
-------	-------	-------	-------	-------	-------	-------	--

6.3 Trusted Security Agent

FFKWC	MWLGW	ZYJRM	RCPHY	JMCGT			
-------	-------	-------	-------	-------	--	--	--

Table 2.2: Level 1 Rule Identifiers in the FCSA

5 Chiplet interfaces

5.3 Hub Chiplet to Hub Chiplet Interface requirements

ZPXZW	KMPRP
-------	-------

5.4 Hub Chiplet to Compute Chiplet Interface requirements

NQBSN	YWSTJ
-------	-------

5.5 Compute 1 Chiplet to Compute 1 Chiplet Interface requirements

SZTCH	RSGWB
-------	-------

5.6 Compute 2 Chiplet to Compute 2 Chiplet Interface requirements

JRSXS RLQYC

5.7 Fully Coherent Expansion Chiplet Interface requirements

PRKPY WPZNH

5.9 I/O Coherent Expansion (Translated) Chiplet Interface requirements

CNCGH JLGGN

5.10 I/O Coherent Expansion (Untranslated) Chiplet Interface requirements

WBSBR WWJWH

5.12 I/O Chiplet Interface requirements

WGMMT GBYYK

5.13 I/O Controller Chiplet Interface requirements

RDVNB JHJVP

7 Chiplet interface implementation

7.2 Interface protocols

JCLGK ZKQNK WVPQS ZNCXC TZRTL

8 Implementation constraints

8.1 Interoperability of chiplets

QLFLC CCBNN

Table 2.3: Level FULL Rule Identifiers in the FCSA

4 Chiplet types

4.5 Fully Coherent Expansion (Remote Translation) Chiplet

BKXFJ LFGWL KWBLT JNZML

4.8 I/O Coherent Expansion (Remote Translation) Chiplet

HBQJS CFCZN JNYGP

6 Common chiplet components

6.2 System Control Framework

SYGXX	WFBVB	CQFTX	PWYBJ	XKQHN	SLGFF	TDNWN	BTYLD
FLQLR	HRPJW	KGYFQ	FSDRF	KZCTP	KLDXX	WZZBY	TVTBS
GGJPQ							

6.3 Trusted Security Agent

HRPTP	NPXGM	KYGJF	QCDJQ	SJFCT	XFSTQ	LFMVS	PQYSK
DQZZW	LDZYP	PCHHY					

7 Chiplet interface implementation

7.2 Interface protocols

JFNFZ	KJVCD	DPWJP	DDXMP	TQRTF	NJGKB	ZFDJY	LZXRR
VLQZG	RJNQG	CWKYG	TDLVS	PXBTX	LXCQZ	KMLLD	

7.3 Interface transports

ZGBTM	CLTBQ	JRHDK	ZZXCL	QDDHW	NCZJN	BCQKB	KBDRB
CJBHK	YNCSL	JDCPH	MHKZW	QZPSP	DDCFS	CTJXM	ZRGXC
HSZXD	BGHPZ	VTPDZ	XYZYJ	GTDWN	HKNHZ	LNKLH	

8 Implementation constraints

8.2 Chiplet system scalability

TPXLG

3

System types

This chapter describes the topology of each system type. It shows how the system is partitioned into chiplets and identifies the chiplet types that the system contains.

The system types define the sets of chiplets required to create viable systems. A system can be scaled or extended by including more instances of mandatory or additional chiplet types.

The system types in the FCSA support independent design of each chiplet type. When combined, chiplets of these types provide the functions required for the system to operate.

The system types are:

- [Compute & Hub System](#)
- [Compute Tile System](#)

3.1 Compute and Hub System

I_{LNZRX}

The [Compute & Hub System](#) type provides many-core, general-purpose compute, coherent memory, and application-specific acceleration across multiple chiplets:

- One or more [Compute 1 Chiplet](#) instances contain the Application PEs.
- A [Hub Chiplet](#) provides system main memory, system cache, and I/O.
- The following chiplet types can add application-specific system functions:
 - [Fully Coherent Expansion Chiplet](#)
 - [I/O Coherent Expansion \(Translated\) Chiplet](#)
 - [I/O Coherent Expansion \(Untranslated\) Chiplet](#)
 - [I/O Coherent Expansion \(Remote Translation\) Chiplet](#)
 - [Fully Coherent Expansion \(Remote Translation\) Chiplet](#)
- The following chiplet types can provide I/O functions on separate chiplets:
 - [I/O Chiplet](#)
 - [I/O Controller Chiplet](#)
- The system can connect to industry-standard PCIe or Compute Express Link (CXL) devices through an appropriately placed interface.

The following figure shows a simplified system example that includes a Memory Management Unit (MMU) and system-level cache (SLC).

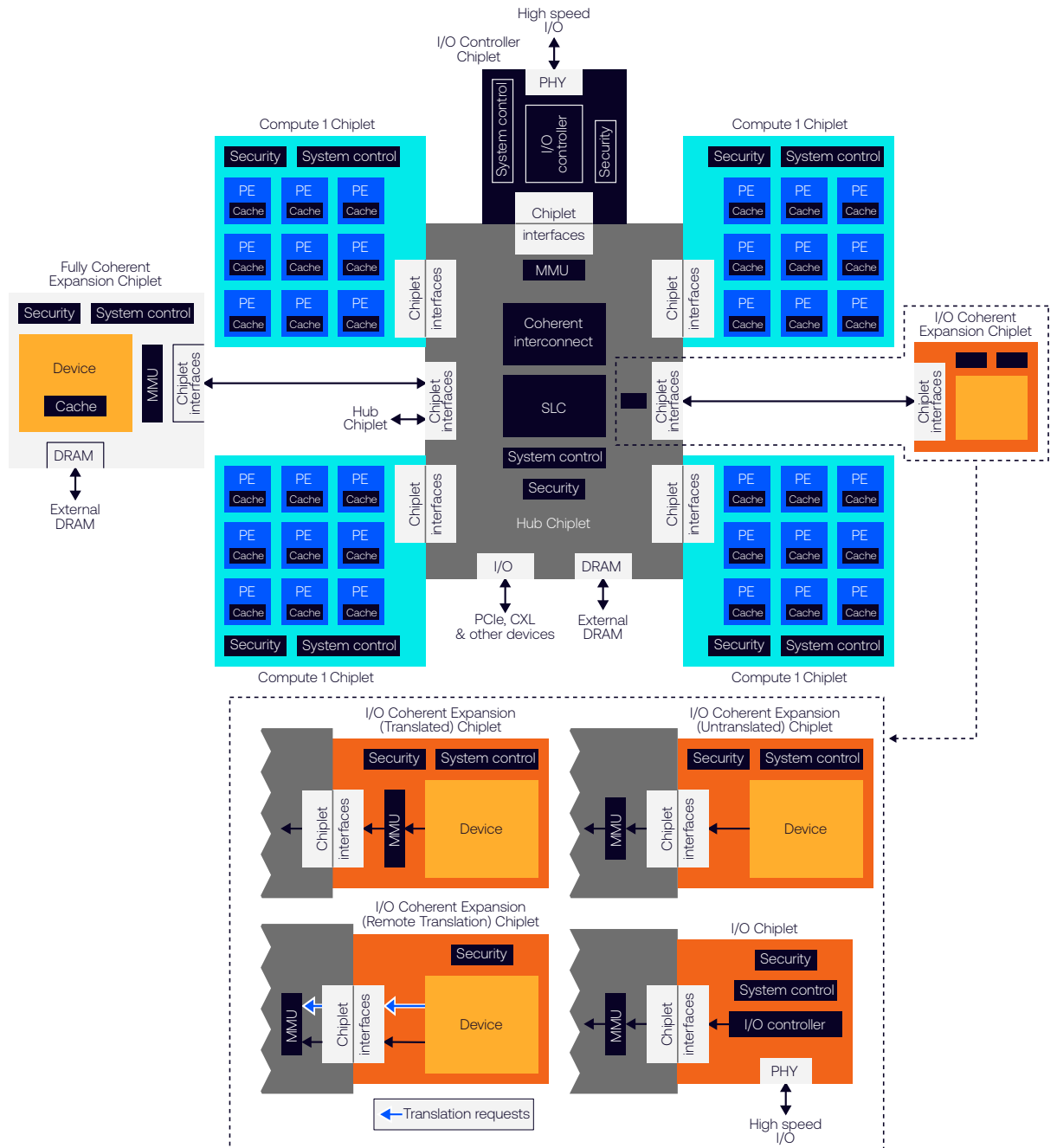


Figure 3.1: Example Compute & Hub System chiplet configuration

3.1.1 System requirements

All constituent chiplets collectively and consistently satisfy these system-level requirements.

3.1.1.1 Composition of chiplet types

R _{LYHKS}	A Compute & Hub System contains one or more Compute 1 Chiplets .	(LEVEL-0)
R _{TFJSM}	A Compute & Hub System contains one or more Hub Chiplets .	(LEVEL-0)
R _{LCJNF}	A Compute & Hub System contains zero or more Fully Coherent Expansion Chiplets .	(LEVEL-0)
R _{CJZHL}	A Compute & Hub System contains zero or more I/O Coherent Expansion (Translated) Chiplets .	(LEVEL-0)
R _{ZCFRY}	A Compute & Hub System contains zero or more I/O Coherent Expansion (Untranslated) Chiplets .	(LEVEL-0)
R _{PPLNQ}	A Compute & Hub System contains zero or more I/O Coherent Expansion (Remote Translation) Chiplets .	(LEVEL-0)
R _{MDPNF}	A Compute & Hub System contains zero or more Fully Coherent Expansion (Remote Translation) Chiplets .	(LEVEL-0)
R _{FZVCL}	A Compute & Hub System contains zero or more I/O Chiplets .	(LEVEL-0)
R _{DXYWQ}	A Compute & Hub System contains zero or more I/O Controller Chiplets .	(LEVEL-0)

3.1.1.2 Application PE

R _{TZJGV}	All Application PEs in a Compute & Hub System are coherent with respect to one shared region of system memory.	(LEVEL-0)
--------------------	--	-----------

3.1.1.3 System memory

R _{ZKZHH}	All Application PEs in a Compute & Hub System can access all system-addressable memory.	(LEVEL-0)
--------------------	---	-----------

3.2 Compute Tile System

I_{RWZCX}

The [Compute Tile System](#) type provides many-core, general-purpose compute, coherent memory, and application-specific acceleration across multiple chiplets:

- One or more [Compute 2 Chiplet](#) instances contain the Application PEs, system main memory, system cache, and I/O.
- The following chiplet types can add application-specific system functions:
 - [Fully Coherent Expansion Chiplet](#)
 - [I/O Coherent Expansion \(Translated\) Chiplet](#)
 - [I/O Coherent Expansion \(Untranslated\) Chiplet](#)
 - [I/O Coherent Expansion \(Remote Translation\) Chiplet](#)
 - [Fully Coherent Expansion \(Remote Translation\) Chiplet](#)
- The following chiplet types can provide I/O functions on separate chiplets:
 - [I/O Chiplet](#)
 - [I/O Controller Chiplet](#)
- The system can connect to industry-standard PCIe or CXL devices through an appropriately placed interface.

The following figure shows a simplified system example.

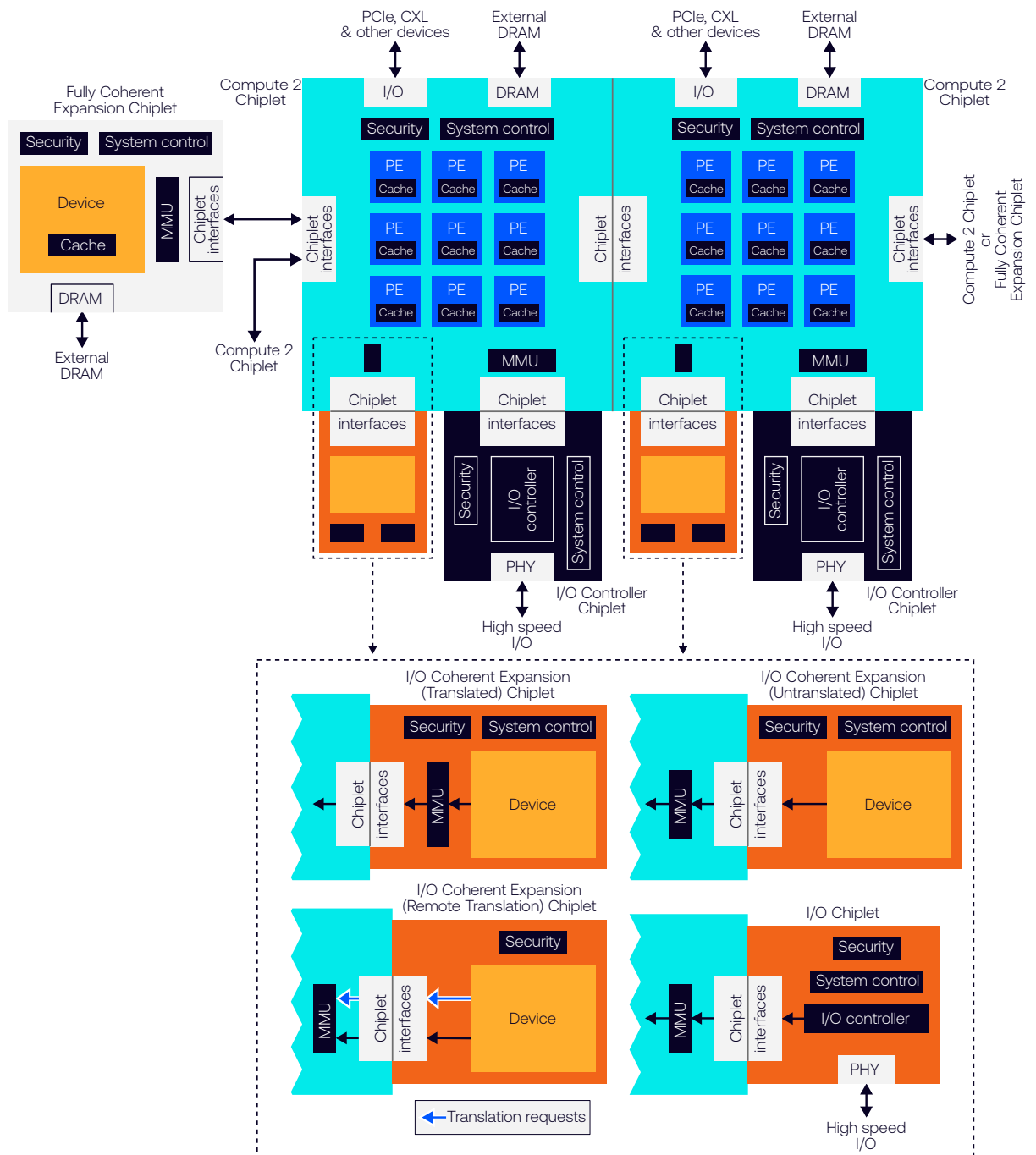


Figure 3.2: Example Compute Tile System chiplet configuration

3.2.1 System requirements

All constituent chiplets collectively and consistently satisfy these system-level requirements.

3.2.1.1 Composition of chiplet types

R _{TKPTF}	A Compute Tile System contains one or more Compute 2 Chiplets .	(LEVEL-0)
R _{MVWBD}	A Compute Tile System contains zero or more Fully Coherent Expansion Chiplets .	(LEVEL-0)
R _{JHWVM}	A Compute Tile System contains zero or more I/O Coherent Expansion (Translated) Chiplets .	(LEVEL-0)
R _{DSKBW}	A Compute Tile System contains zero or more I/O Coherent Expansion (Untranslated) Chiplets .	(LEVEL-0)
R _{ZFTDV}	A Compute Tile System contains zero or more I/O Coherent Expansion (Remote Translation) Chiplets .	(LEVEL-0)
R _{BFSJV}	A Compute Tile System contains zero or more Fully Coherent Expansion (Remote Translation) Chiplets .	(LEVEL-0)
R _{ZDVGZ}	A Compute Tile System contains zero or more I/O Chiplets .	(LEVEL-0)
R _{CCHPT}	A Compute Tile System contains zero or more I/O Controller Chiplets .	(LEVEL-0)

3.2.1.2 Application PE

R _{TTPHX}	All Application PEs in a Compute Tile System are coherent with respect to one shared region of system memory.	(LEVEL-0)
--------------------	---	-----------

3.2.1.3 System memory

R _{WTRKM}	All Application PEs in a Compute Tile System can access all system-addressable memory.	(LEVEL-0)
--------------------	--	-----------

4

Chiplet types

This chapter defines the following chiplet types:

- [Compute 1 Chiplet](#)
- [Hub Chiplet](#)
- [Compute 2 Chiplet](#)
- [Fully Coherent Expansion Chiplet](#)
- [Fully Coherent Expansion \(Remote Translation\) Chiplet](#)
- [I/O Coherent Expansion \(Translated\) Chiplet](#)
- [I/O Coherent Expansion \(Untranslated\) Chiplet](#)
- [I/O Coherent Expansion \(Remote Translation\) Chiplet](#)
- [I/O Chiplet](#)
- [I/O Controller Chiplet](#)

4.1 Compute 1 Chiplet

$\mathbb{I}_{\text{VGHWL}}$ The **Compute 1 Chiplet** type contains one or more Application PEs. It is not required to contain system main memory or I/O peripherals because a **Hub Chiplet** provides these resources. The chiplet might also connect to another **Compute 1 Chiplet**.

Figure 4.1 shows simplified examples of a **Compute 1 Chiplet**.

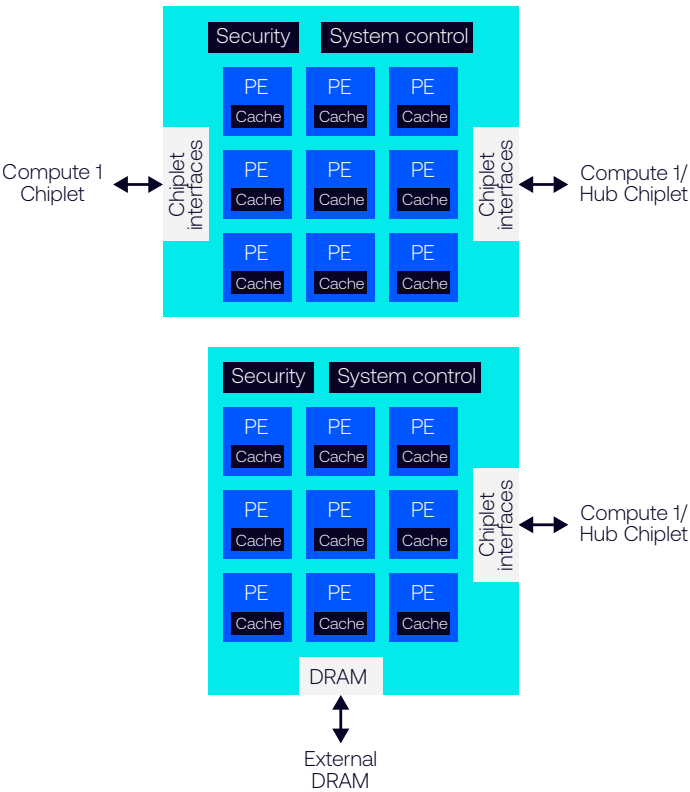


Figure 4.1: Compute 1 Chiplet examples

The **Compute 1 Chiplet** is a constituent of the **Compute & Hub System** system type, which specifies additional system-level rules that apply to this chiplet type.

4.1.1 Chiplet requirements

The following requirements define the [Compute 1 Chiplet](#) type.

4.1.1.1 Chiplet-to-chiplet interfaces

R _{JWRCG}	A Compute 1 Chiplet interfaces with one of the following mutually exclusive chiplet types: • A Hub Chiplet • A Compute 1 Chiplet	(LEVEL-0)
R _{BPRJK}	A Compute 1 Chiplet interfaces with zero or more Compute 1 Chiplets .	(LEVEL-0)
R _{RCYDP}	A Compute 1 Chiplet interfaces with zero or exactly one Hub Chiplet .	(LEVEL-0)
I _{WRFKN}	See Compute 1 Chiplet to Compute 1 Chiplet Interface Requirements .	
I _{NBZQX}	See Hub Chiplet to Compute Chiplet Interface Requirements .	

4.1.1.2 Application PE

R _{XTDHG}	A Compute 1 Chiplet contains one or more Application PEs.	(LEVEL-0)
R _{PTZSC}	All Application PEs in a Compute 1 Chiplet are fully coherent.	(LEVEL-0)

4.1.1.3 Fully coherent processing device

R _{NCHRQ}	A Compute 1 Chiplet does not contain any fully coherent processing devices.	(LEVEL-0)
--------------------	---	-----------

4.1.1.4 I/O coherent processing device

R _{KNKSV}	A Compute 1 Chiplet does not contain any I/O coherent processing devices.	(LEVEL-0)
--------------------	---	-----------

4.1.1.5 I/O coherent I/O device

R _{NRJZH}	A Compute 1 Chiplet does not contain any I/O coherent I/O devices.	(LEVEL-0)
--------------------	--	-----------

4.1.1.6 Non-coherent I/O device

R _{NGZDK}	A Compute 1 Chiplet does not contain any non-coherent I/O devices.	(LEVEL-0)
--------------------	--	-----------

4.1.1.7 System memory

R _{ZHHPH}	A Compute 1 Chiplet might contain system main memory, either integrated or directly connected.	(LEVEL-0)
R _{HQXHS}	The system main memory in or directly connected to a Compute 1 Chiplet has associated serialization functionality and might have associated hardware coherency functionality. This functionality is contained in the Compute 1 Chiplet .	(LEVEL-0)
R _{XGXZR}	The system main memory in or directly connected to a Compute 1 Chiplet receives accesses from agents in the same chiplet or other chiplets.	(LEVEL-0)

R_{MCLJS} Any system addressable memory in the system is available through a [Compute 1 Chiplet](#) to agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{CQDKD} Any coherent caches in a [Compute 1 Chiplet](#) receive coherency protocol accesses from any coherent agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{GHRFJ} Any device shared memory in or directly connected to a [Compute 1 Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

4.1.1.7.1 MMU location

I_{PNRLI} Agents in a [Compute 1 Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain.

R_{ZYTQO} PEs in a [Compute 1 Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet. (LEVEL-0)

4.1.1.8 Interrupt handling

I_{BFBQR} This section concerns interrupts for Application PEs that run the main operating system.

A [Compute 1 Chiplet](#) can be a source and destination of interrupts. Therefore, the chiplet can:

- Generate an interrupt internally for a PE in the chiplet
- Generate an interrupt internally for a PE in another chiplet
- Receive an interrupt from another chiplet for a PE in the chiplet

R_{TGSXZ} A [Compute 1 Chiplet](#) contains an interrupt controller component. (LEVEL-0)

I_{CTFCS} The interrupt controller component in the [Compute 1 Chiplet](#) receives and forwards interrupts as follows:

- If the destination is in the same chiplet, the component forwards the interrupt to the target PE.
- If the destination is in another chiplet, the component forwards the interrupt to the interrupt controller component in the chiplet that contains the target PE.

[Figure 4.2](#) shows a simplified interrupt path between chiplets that both contain an interrupt controller component.

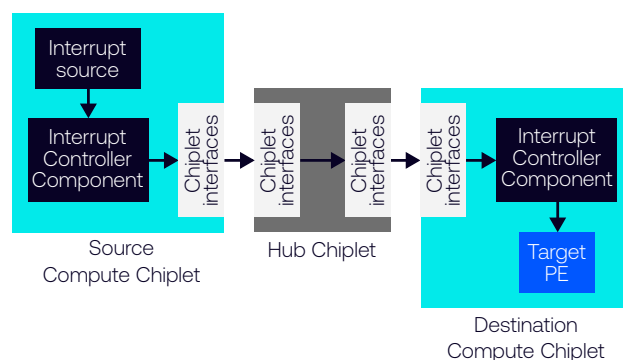


Figure 4.2: Example interrupt communication between two compute chiplets

R_{MJNUG} The [Compute 1 Chiplet](#) interrupt controller component receives interrupts from within the same chiplet. (LEVEL-0)

R_{GVVMS} The [Compute 1 Chiplet](#) interrupt controller component receives interrupts for the PEs it contains, forwarded from other chiplets.

(LEVEL-0)

R_{ZDSHM} Where the target PE of an interrupt is in another chiplet, the [Compute 1 Chiplet](#) interrupt controller component forwards the interrupt to the interrupt controller component in the chiplet that contains the target PE.

(LEVEL-0)

R_{DZFZV} Where the target PE of an interrupt is in the same chiplet, the [Compute 1 Chiplet](#) interrupt controller component forwards interrupts to the destination PE.

(LEVEL-0)

4.1.1.9 Chiplet system control

R_{RNSKY} A [Compute 1 Chiplet](#) contains a system controller (see [System Controller](#)).

(LEVEL-0)

R_{PGGRB} The [system controller](#) in a [Compute 1 Chiplet](#) is a secondary system controller.

(LEVEL-0)

4.1.1.10 Security

R_{XSFHF} A [Compute 1 Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{SFPHM} A [trusted security agent](#) in a [Compute 1 Chiplet](#) is a secondary trusted security agent.

(LEVEL-0)

4.1.1.11 Debug

I_{KNFPR} A [Compute 1 Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.1.1.11.1 Debug access

R_{MDGWP} An external debugger can access the debug resources of components in a [Compute 1 Chiplet](#) through a Debug Access Port (DAP) in the chiplet.

(LEVEL-0)

R_{RCRRQ} The debug resources of components in a [Compute 1 Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets.

(LEVEL-0)

R_{WDJHV} System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [Compute 1 Chiplet](#) by using memory-mapped accesses.

(LEVEL-0)

I_{PVSHB} The [Trusted Security Agent](#) in a [Compute 1 Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{DKSBZ} A [Compute 1 Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity.

(LEVEL-0)

4.1.1.12 System

I_{HRXSM}	A unique identifier is necessary for each Compute 1 Chiplet because a system might contain more than one instance of the chiplet.
R_{MRRGT}	All chiplets in the system can access the identification register in a Compute 1 Chiplet . The value in the register uniquely identifies the chiplet instance within the system.

(LEVEL-0)

4.2 Hub Chiplet

I_{CTPLV}

The **Hub Chiplet** type has direct access to system main memory and I/O peripherals. It provides this access to connected compute chiplets. Other connected chiplets can provide fully coherent processing by devices other than Application PEs, I/O coherent expansion, or I/O.

A **Hub Chiplet** provides security and system management functions. If a system contains multiple instances of a **Hub Chiplet**, one is the primary chiplet and coordinates these functions.

Figure 4.3 shows a simplified example of a **Hub Chiplet**.

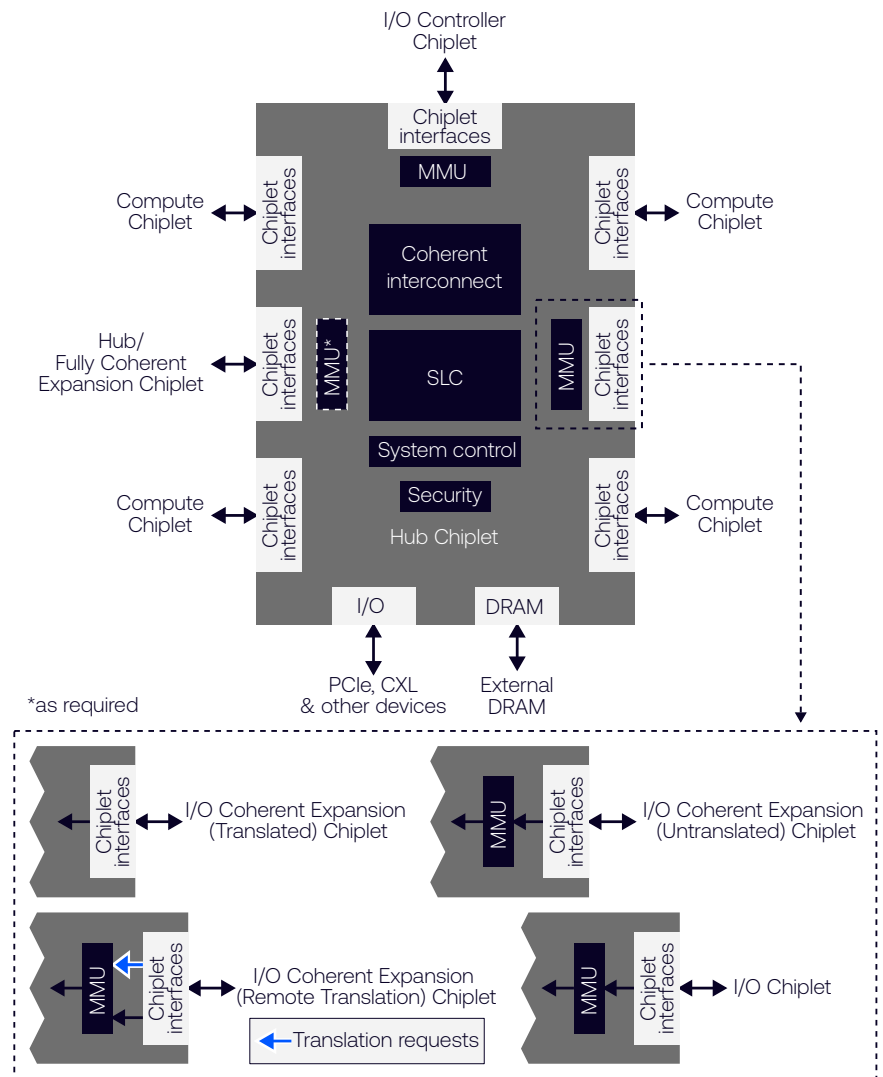


Figure 4.3: Hub Chiplet example

The **Hub Chiplet** is a constituent of the **Compute & Hub System** system type, which specifies additional system-level rules that apply to this chiplet type.

4.2.1 Chiplet requirements

The following requirements define the [Hub Chiplet](#) type.

4.2.1.1 Chiplet-to-chiplet interfaces

R_{MZLJX}	A Hub Chiplet interfaces with one or more Compute 1 Chiplets .	(LEVEL-0)
R_{CNQZH}	A Hub Chiplet interfaces with zero or more Compute 2 Chiplets .	(LEVEL-0)
R_{LCQGC}	A Hub Chiplet interfaces with zero or more Hub Chiplets .	(LEVEL-0)
R_{SVWVP}	A Hub Chiplet interfaces with zero or more Fully Coherent Expansion Chiplets .	(LEVEL-0)
R_{HBYND}	A Hub Chiplet interfaces with zero or more Fully Coherent Expansion (Remote Translation) Chiplets .	(LEVEL-0)
R_{MKYNH}	A Hub Chiplet interfaces with zero or more I/O Coherent Expansion (Translated) Chiplets .	(LEVEL-0)
R_{WMTVP}	A Hub Chiplet interfaces with zero or more I/O Coherent Expansion (Untranslated) Chiplets .	(LEVEL-0)
R_{MBQZT}	A Hub Chiplet interfaces with zero or more I/O Coherent Expansion (Remote Translation) Chiplets .	(LEVEL-0)
R_{MLQLD}	A Hub Chiplet interfaces with zero or more I/O Chiplets .	(LEVEL-0)
R_{ZTCKN}	A Hub Chiplet interfaces with zero or more I/O Controller Chiplets .	(LEVEL-0)
I_{QPRCW}	See Hub Chiplet to Compute Chiplet Interface Requirements .	
I_{SDZJJ}	See Hub Chiplet to Hub Chiplet Interface Requirements .	
I_{NDKNP}	See Fully Coherent Expansion Chiplet Interface Requirements .	
I_{HSQOH}	See I/O Coherent Expansion (Translated) Chiplet Interface Requirements .	
I_{CXHLL}	See I/O Coherent Expansion (Untranslated) Chiplet Interface Requirements .	
I_{KKNFJ}	See I/O Coherent Expansion (Remote Translation) Chiplet Interface Requirements .	
I_{FQDZM}	See Fully Coherent Expansion (Remote Translation) Chiplet Interface Requirements .	
I_{YBZWX}	See I/O Chiplet Interface Requirements .	
I_{HMTNQ}	See I/O Controller Chiplet Interface Requirements .	

4.2.1.2 Application PE

R_{PCJMB}	A Hub Chiplet does not contain any Application PEs.	(LEVEL-0)
-----------------------------------	---	-----------------------------

4.2.1.3 Fully coherent processing device

R_{FXFNL}	A Hub Chiplet contains zero or more fully coherent processing devices.	(LEVEL-0)
-----------------------------------	--	-----------------------------

4.2.1.4 I/O coherent processing device

R_{JDYJY}	A Hub Chiplet contains zero or more I/O coherent processing devices.	(LEVEL-0)
-----------------------------------	--	-----------------------------

4.2.1.5 I/O coherent I/O device

R_{PMKFN} A [Hub Chiplet](#) contains zero or more I/O coherent I/O devices. (LEVEL-0)

4.2.1.6 Non-coherent I/O device

R_{YHQDR} A [Hub Chiplet](#) contains zero or more non-coherent I/O devices. (LEVEL-0)

4.2.1.7 System memory

R_{XDGVQ} A [Hub Chiplet](#) contains system main memory, either integrated or directly connected. (LEVEL-0)

R_{TZRSV} The system main memory in or directly connected to a [Hub Chiplet](#) has associated serialization functionality and might have associated hardware coherency functionality. This functionality is contained in the [Hub Chiplet](#). (LEVEL-0)

R_{RXLZG} The system main memory in or directly connected to a [Hub Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

R_{MTBPW} Any system addressable memory in the system is available through a [Hub Chiplet](#) to agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{VJFWY} Any coherent caches in a [Hub Chiplet](#) receive coherency protocol accesses from any coherent agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{KZFYG} Any device shared memory in or directly connected to a [Hub Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

4.2.1.7.1 MMU location

$I_{HXR BV}$ Agents in a [Hub Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain.

R_{RGMKW} I/O coherent or non-coherent agents in a [Hub Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet. (LEVEL-0)

R_{GNJLK} Fully coherent agents in a [Hub Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet. (LEVEL-0)

4.2.1.8 Interrupt handling

I_{MWPTG} This section concerns interrupts for Application PEs that run the main operating system.

A [Hub Chiplet](#) can be a source of interrupts, or convey interrupts from other chiplets. Therefore, the chiplet can:

- Generate an interrupt internally for a PE in another chiplet
- Receive an interrupt for a PE in another chiplet from a chiplet that does not have an interrupt controller component

R_{GSSWV} A [Hub Chiplet](#) contains an interrupt controller component. (LEVEL-0)

I_{PPCWK}

The interrupt controller component in the [Hub Chiplet](#) receives interrupts and forwards them to the interrupt controller component in the chiplet that contains the target PE.

[Figure 4.4](#) shows a simplified interrupt path between chiplets that both contain an interrupt controller component.

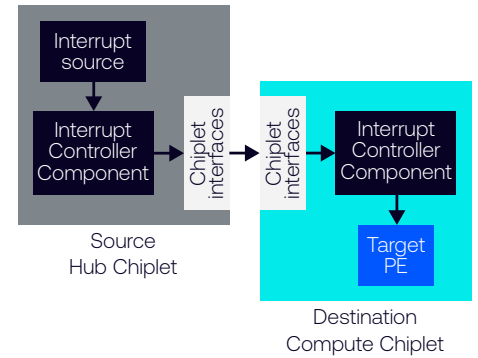


Figure 4.4: Example interrupt communication between a hub and a compute chiplet

R_{FTXZM}

The [Hub Chiplet](#) interrupt controller component receives interrupts from within the same chiplet.

(LEVEL-0)

R_{ZPLGW}

The [Hub Chiplet](#) interrupt controller component receives interrupts from other chiplets in the system that do not contain an interrupt controller component.

(LEVEL-0)

R_{VNGDP}

The [Hub Chiplet](#) interrupt controller component forwards interrupts to the interrupt controller component in the chiplet of the target PE.

(LEVEL-0)

4.2.1.9 Non-volatile memory

R_{NJQDV}

When a [Hub Chiplet](#) is designated as the primary chiplet (see [Hub Chiplet system requirements](#)), it contains non-volatile memory, either integrated or directly connected, that can hold boot code images for the trusted security agents, system controllers, and Application PEs in the system.

(LEVEL-0)

R_{NXDVW}

When a [Hub Chiplet](#) is not designated as the primary chiplet (see [Hub Chiplet system requirements](#)), it might contain non-volatile memory, either integrated or directly connected, that can hold boot code images for the trusted security agents, system controllers, and Application PEs in the system.

(LEVEL-0)

4.2.1.10 Chiplet system control

R_{CKXYX}

A [Hub Chiplet](#) contains a system controller (see [System Controller](#)).

(LEVEL-0)

R_{ZFWWC}

When a [Hub Chiplet](#) is designated as the primary chiplet (see [Hub Chiplet system requirements](#)), the [system controller](#) it contains is the primary system controller.

(LEVEL-0)

R_{PYRRZ}

When a [Hub Chiplet](#) is not designated as the primary chiplet (see [Hub Chiplet system requirements](#)), the [system controller](#) it contains is a secondary system controller.

(LEVEL-0)

R_{BTGJS}

A [Hub Chiplet](#) contains one or more system control agents (see [System Control Agent](#)).

(LEVEL-0)

4.2.1.11 Security

R_{ZCPHB}

A [Hub Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.
- Secure storage of immutable boot code.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{LDSVP}

When a [Hub Chiplet](#) is designated as the primary chiplet (see [Hub Chiplet system requirements](#)), a single [trusted security agent](#) it contains is the primary trusted security agent and any others are secondary trusted security agents.

(LEVEL-0)

R_{GYSJB}

When a [Hub Chiplet](#) is not designated as the primary chiplet (see [Hub Chiplet system requirements](#)), all of the [trusted security agents](#) it contains are secondary trusted security agents.

(LEVEL-0)

4.2.1.12 Debug

I_{DJTMK}

A [Hub Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.2.1.12.1 Debug access

R_{SXNDJ}

An external debugger can access the debug resources of components in a [Hub Chiplet](#) through a Debug Access Port (DAP) in the chiplet.

(LEVEL-0)

R_{TVKHG}

The debug resources of components in a [Hub Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets.

(LEVEL-0)

R_{CJMRD}

System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [Hub Chiplet](#) by using memory-mapped accesses.

(LEVEL-0)

I_{CCXBP}

The [Trusted Security Agent](#) in a [Hub Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{WTGSM}

A [Hub Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity.

(LEVEL-0)

4.2.1.13 System

I_{WMGRV}

A unique identifier is necessary for each [Hub Chiplet](#) because a system might contain more than one instance of the chiplet.

R_{YJBND}

All chiplets in the system can access the identification register in a [Hub Chiplet](#). The value in the register uniquely identifies the chiplet instance within the system.

(LEVEL-0)

R_{KYYNQ}

There is a single chiplet in the system that is designated as the *primary* chiplet and it is an instance of a [Hub Chiplet](#).

(LEVEL-0)

R_{DZGQK} When there are two or more instances of a [Hub Chiplet](#) in the system, only one of them is designated as the *primary* chiplet. A [Hub Chiplet](#) contains an IMPLEMENTATION DEFINED mechanism that determines if it is the *primary* or a *secondary* chiplet.

(LEVEL-0)

U_{ZCTBJ} An example mechanism for determining the primary chiplet is an input signal that is tied off at the package.

4.3 Compute 2 Chiplet

I_{XBBNM}

The **Compute 2 Chiplet** type contains one or more Application PEs and has direct access to system main memory and I/O peripherals. Connected chiplets can provide additional functions, including another **Compute 2 Chiplet**, fully coherent processing by devices other than Application PEs, I/O coherent expansion, or I/O.

A **Compute 2 Chiplet** provides security and system management functions. If a system contains multiple instances of a **Compute 2 Chiplet**, one is the primary chiplet and coordinates these functions.

Figure 4.5 shows a simplified example of a **Compute 2 Chiplet**.

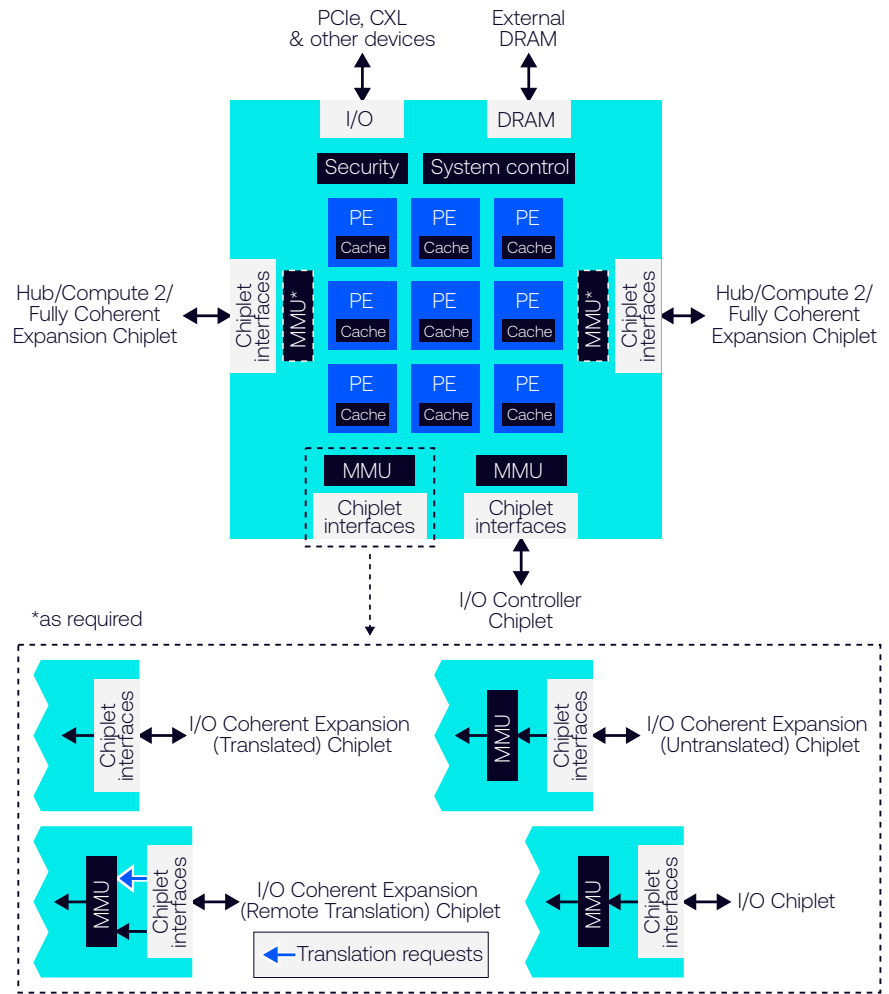


Figure 4.5: Compute 2 Chiplet example

The **Compute 2 Chiplet** is a constituent of the **Compute Tile System** system type, which specifies additional system-level rules that apply to this chiplet type.

4.3.1 Chiplet requirements

The following requirements define the [Compute 2 Chiplet](#) type.

4.3.1.1 Chiplet-to-chiplet interfaces

R_{VKGJK}	A Compute 2 Chiplet interfaces with one of the following mutually exclusive chiplet types: • A Hub Chiplet • A Compute 2 Chiplet	(<i>LEVEL-0</i>)
R_{SJNGN}	A Compute 2 Chiplet interfaces with zero or exactly one Hub Chiplet .	(<i>LEVEL-0</i>)
R_{NDHDT}	A Compute 2 Chiplet interfaces with zero or more Compute 2 Chiplets .	(<i>LEVEL-0</i>)
R_{KNHMM}	A Compute 2 Chiplet interfaces with zero or more Fully Coherent Expansion Chiplets .	(<i>LEVEL-0</i>)
R_{CGKQW}	A Compute 2 Chiplet interfaces with zero or more Fully Coherent Expansion (Remote Translation) Chiplets .	(<i>LEVEL-0</i>)
R_{MDFWD}	A Compute 2 Chiplet interfaces with zero or more I/O Coherent Expansion (Translated) Chiplets .	(<i>LEVEL-0</i>)
R_{WLHVZ}	A Compute 2 Chiplet interfaces with zero or more I/O Coherent Expansion (Untranslated) Chiplets .	(<i>LEVEL-0</i>)
R_{JZSRC}	A Compute 2 Chiplet interfaces with zero or more I/O Coherent Expansion (Remote Translation) Chiplets .	(<i>LEVEL-0</i>)
R_{DVNHD}	A Compute 2 Chiplet interfaces with zero or more I/O Chiplets .	(<i>LEVEL-0</i>)
R_{QGGWQ}	A Compute 2 Chiplet interfaces with zero or more I/O Controller Chiplets .	(<i>LEVEL-0</i>)
I_{ZQGLZ}	See Hub Chiplet to Compute Chiplet Interface Requirements .	
I_{VXZQN}	See Compute 2 Chiplet to Compute 2 Chiplet Interface Requirements .	
I_{LVBGP}	See Fully Coherent Expansion Chiplet Interface Requirements .	
I_{SPPT}	See Fully Coherent Expansion Chiplet (Remote Translation) Interface Requirements .	
I_{XQGLB}	See I/O Coherent Expansion (Translated) Chiplet Interface Requirements .	
I_{HJRVL}	See I/O Coherent Expansion (Untranslated) Chiplet Interface Requirements .	
I_{DBDNY}	See I/O Coherent Expansion (Remote Translation) Chiplet Interface Requirements .	
I_{MHXYJ}	See I/O Chiplet Interface Requirements .	
I_{LNRCG}	See I/O Controller Chiplet Interface Requirements .	

4.3.1.2 Application PE

R_{LGXLN}	A Compute 2 Chiplet contains one or more Application PEs.	(<i>LEVEL-0</i>)
R_{XNYDJ}	All Application PEs in a Compute 2 Chiplet are fully coherent.	(<i>LEVEL-0</i>)

4.3.1.3 Fully coherent processing device

R_{PHHQM} A [Compute 2 Chiplet](#) contains zero or more fully coherent processing devices. (LEVEL-0)

4.3.1.4 I/O coherent processing device

R_{ZVZZN} A [Compute 2 Chiplet](#) contains zero or more I/O coherent processing devices. (LEVEL-0)

4.3.1.5 I/O coherent I/O device

R_{VYJFL} A [Compute 2 Chiplet](#) contains zero or more I/O coherent I/O devices. (LEVEL-0)

4.3.1.6 Non-coherent I/O device

R_{ZCLCN} A [Compute 2 Chiplet](#) contains zero or more non-coherent I/O devices. (LEVEL-0)

4.3.1.7 System memory

R_{DMSQX} A [Compute 2 Chiplet](#) contains system main memory, either integrated or directly connected. (LEVEL-0)

R_{QSXVN} The system main memory in or directly connected to a [Compute 2 Chiplet](#) has associated serialization functionality and might have associated hardware coherency functionality. This functionality is contained in the [Compute 2 Chiplet](#). (LEVEL-0)

R_{XCVGN} The system main memory in or directly connected to a [Compute 2 Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

R_{ZXWKH} Any system addressable memory in the system is available through a [Compute 2 Chiplet](#) to agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{DZDHQ} Any coherent caches in a [Compute 2 Chiplet](#) receive coherency protocol accesses from any coherent agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{PBYQY} Any device shared memory in or directly connected to a [Compute 2 Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

4.3.1.7.1 MMU location

R_{VXGYS} Agents in a [Compute 2 Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain. (LEVEL-0)

R_{CLYNB} PEs in a [Compute 2 Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet. (LEVEL-0)

R_{GPSSB} I/O coherent or non-coherent agents in a [Compute 2 Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet. (LEVEL-0)

4.3.1.8 Interrupt handling

I_KNLT

This section concerns interrupts for Application PEs that run the main operating system.

A **Compute 2 Chiplet** can be a source and destination of interrupts. Therefore, the chiplet can:

- Generate an interrupt internally for a PE in the chiplet
- Generate an interrupt internally for a PE in another chiplet
- Receive an interrupt from another chiplet for a PE in the chiplet
- Receive an interrupt from another chiplet for a PE in another chiplet

R_TXVCM

A **Compute 2 Chiplet** contains an interrupt controller component.

(LEVEL-0)

I_FRVDP

The interrupt controller component in the **Compute 2 Chiplet** receives and forwards interrupts as follows:

- If the destination is in the same chiplet, the component forwards the interrupt to the target PE.
- If the destination is in another chiplet, the component forwards the interrupt to the interrupt controller component in the chiplet that contains the target PE.

Figure 4.6 shows a simplified interrupt path between chiplets that both contain an interrupt controller component.

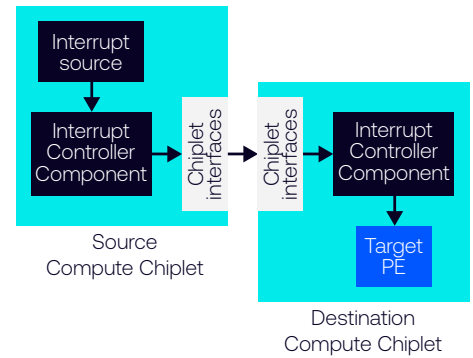


Figure 4.6: Example interrupt communication between two compute chiplets

R_THPPG

The **Compute 2 Chiplet** interrupt controller component receives interrupts from within the same chiplet.

(LEVEL-0)

R_NYZVM

The **Compute 2 Chiplet** interrupt controller component receives interrupts from other chiplets in the system that do not contain an interrupt controller component.

(LEVEL-0)

R_YNMLX

The **Compute 2 Chiplet** interrupt controller component receives interrupts for the PEs it contains, forwarded from other chiplets.

(LEVEL-0)

R_SGZXX

The **Compute 2 Chiplet** interrupt controller component forwards interrupts to a PE, where the target PE is in the same chiplet.

(LEVEL-0)

R_VVDFD

Where the target PE of an interrupt is in another chiplet, the **Compute 2 Chiplet** interrupt controller component forwards the interrupt to the interrupt controller component in the chiplet that contains the target PE.

(LEVEL-0)

4.3.1.9 Non-volatile memory

R_{NLGDR} When a [Compute 2 Chiplet](#) is designated as the primary chiplet (see [Compute 2 Chiplet system requirements](#)), it contains non-volatile memory, either integrated or directly connected, that can hold boot code images for the trusted security agents, system controllers, and Application PEs in the system. (LEVEL-0)

R_{WVMSK} When a [Compute 2 Chiplet](#) is not designated as the primary chiplet (see [Compute 2 Chiplet system requirements](#)), it might contain non-volatile memory, either integrated or directly connected, that can hold boot code images for the trusted security agents, system controllers, and Application PEs in the system. (LEVEL-0)

4.3.1.10 Chiplet system control

R_{PGMTQ} A [Compute 2 Chiplet](#) contains a system controller (see [System Controller](#)). (LEVEL-0)

R_{YQMHG} When a [Compute 2 Chiplet](#) is designated as the primary chiplet (see [Compute 2 Chiplet system requirements](#)), the [system controller](#) it contains is the primary system controller. (LEVEL-0)

R_{FDRRV} When a [Compute 2 Chiplet](#) is not designated as the primary chiplet (see [Compute 2 Chiplet system requirements](#)), the [system controller](#) it contains is a secondary system controller. (LEVEL-0)

R_{CNDVJ} A [Compute 2 Chiplet](#) contains one or more system control agents (see [System Control Agent](#)). (LEVEL-0)

4.3.1.11 Security

R_{FWCJG} A [Compute 2 Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.
- Secure storage of immutable boot code.

For more information about HES, see [Trusted Security Agent](#). (LEVEL-0)

R_{KGFNL} When a [Compute 2 Chiplet](#) is designated as the primary chiplet (see [Compute 2 Chiplet system requirements](#)), a single [trusted security agent](#) it contains is the primary trusted security agent and any others are secondary trusted security agents. (LEVEL-0)

R_{GNZNV} When a [Compute 2 Chiplet](#) is not designated as the primary chiplet (see [Compute 2 Chiplet system requirements](#)), all of the [trusted security agents](#) it contains are secondary trusted security agents. (LEVEL-0)

4.3.1.12 Debug

I_{ZNNHD} A [Compute 2 Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.3.1.12.1 Debug access

R_{ZDBLR} An external debugger can access the debug resources of components in a [Compute 2 Chiplet](#) through a Debug Access Port (DAP) in the chiplet. (LEVEL-0)

R_{XFSNZ} The debug resources of components in a [Compute 2 Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets. (LEVEL-0)

R _{NSRSQ}	System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a Compute 2 Chiplet by using memory-mapped accesses.	(LEVEL-0)
I _{KPDJB}	The Trusted Security Agent in a Compute 2 Chiplet might restrict debug access to enforce the hardware security lifecycle state of the chiplet. <i>Authenticated debug</i> provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].	
R _{ZHVKB}	A Compute 2 Chiplet might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an <i>Authenticated debug</i> capability that enables debug access after the trusted security agent verifies identity.	(LEVEL-0)

4.3.1.13 System

I _{DBTLF}	A unique identifier is necessary for each Compute 2 Chiplet because a system might contain more than one instance of the chiplet.	
R _{XDDFR}	All chiplets in the system can access the identification register in a Compute 2 Chiplet . The value in the register uniquely identifies the chiplet instance within the system.	(LEVEL-0)
R _{XYVWT}	There is a single chiplet in the system that is designated as the <i>primary</i> chiplet. When there are one or more instances of a Hub Chiplet in the system a single instance of a Hub Chiplet is designated as the <i>primary</i> chiplet. When there are zero instances of a Hub Chiplet in the system a single instance of a Compute 2 Chiplet is designated as the <i>primary</i> chiplet.	(LEVEL-0)
R _{LPTTQ}	When there are zero instances of a Hub Chiplet and two or more instances of a Compute 2 Chiplet in the system, only one of the instances of a Compute 2 Chiplet is designated as <i>primary</i> . A Compute 2 Chiplet contains an IMPLEMENTATION DEFINED mechanism that determines if it is a <i>primary</i> or a <i>secondary</i> chiplet.	(LEVEL-0)
U _{TYRVJ}	An example mechanism for determining the primary chiplet is an input signal that is tied off at the package.	

4.4 Fully Coherent Expansion Chiplet

TLNDW

The **Fully Coherent Expansion Chiplet** type provides the system with fully coherent application-specific acceleration and access to system main memory. The chiplet might include interfaces to external DRAM and to other fully coherent, I/O coherent, or non-coherent chiplet types.

Figure 4.7 shows simplified examples of a **Fully Coherent Expansion Chiplet**.

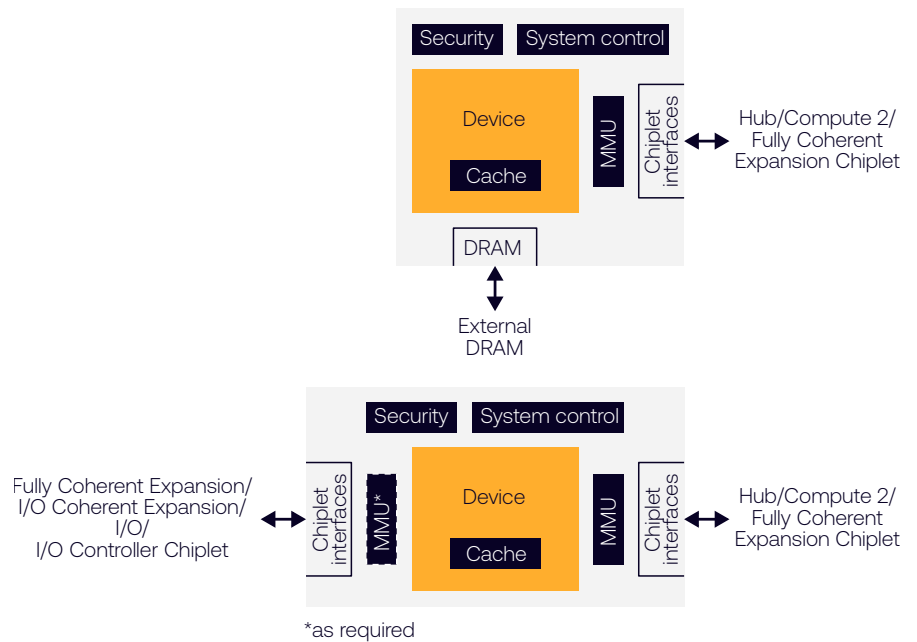


Figure 4.7: Fully Coherent Expansion Chiplet examples

A **Fully Coherent Expansion Chiplet** is a constituent of the following system types, which specify additional system-level rules that apply to this chiplet type:

- **Compute & Hub System**
- **Compute Tile System**

4.4.1 Chiplet requirements

The following requirements define the [Fully Coherent Expansion Chiplet](#) type.

4.4.1.1 Chiplet-to-chiplet interfaces

R_{SHXTT}	A Fully Coherent Expansion Chiplet interfaces with one of the following mutually exclusive chiplet types: • A Hub Chiplet • A Compute 2 Chiplet • A Fully Coherent Expansion Chiplet	(<i>LEVEL-0</i>)
R_{HXGDG}	A Fully Coherent Expansion Chiplet interfaces with zero or exactly one Hub Chiplet .	(<i>LEVEL-0</i>)
R_{GHKCL}	A Fully Coherent Expansion Chiplet interfaces with zero or exactly one Compute 2 Chiplet .	(<i>LEVEL-0</i>)
R_{MKCTS}	A Fully Coherent Expansion Chiplet interfaces with zero or more Fully Coherent Expansion Chiplets .	(<i>LEVEL-0</i>)
R_{HMZJQ}	A Fully Coherent Expansion Chiplet interfaces with zero or more Fully Coherent Expansion (Remote Translation) Chiplets .	(<i>LEVEL-0</i>)
R_{YDWVV}	A Fully Coherent Expansion Chiplet interfaces with zero or more I/O Coherent Expansion (Translated) Chiplets .	(<i>LEVEL-0</i>)
R_{NBXWR}	A Fully Coherent Expansion Chiplet interfaces with zero or more I/O Coherent Expansion (Untranslated) Chiplets .	(<i>LEVEL-0</i>)
R_{FMBVG}	A Fully Coherent Expansion Chiplet interfaces with zero or more I/O Coherent Expansion (Remote Translation) Chiplets .	(<i>LEVEL-0</i>)
R_{JNCKH}	A Fully Coherent Expansion Chiplet interfaces with zero or more I/O Chiplets .	(<i>LEVEL-0</i>)
R_{VCPQR}	A Fully Coherent Expansion Chiplet interfaces with zero or more I/O Controller Chiplets .	(<i>LEVEL-0</i>)
I_{YVR SB}	See Fully Coherent Expansion Chiplet Interface Requirements .	
I_{GPV MH}	See Fully Coherent Expansion Chiplet (Remote Translation) Interface Requirements .	
I_{HVW GS}	See I/O Coherent Expansion (Translated) Chiplet Interface Requirements .	
I_{TRF HV}	See I/O Coherent Expansion (Untranslated) Chiplet Interface Requirements .	
I_{SVF PP}	See I/O Coherent Expansion (Remote Translation) Chiplet Interface Requirements .	
I_{HVJ YJ}	See I/O Chiplet Interface Requirements .	
I_{MQP DR}	See I/O Controller Chiplet Interface Requirements .	

4.4.1.2 Application PE

R_{FJDN P}	A Fully Coherent Expansion Chiplet does not contain any Application PEs.	(<i>LEVEL-0</i>)
------------------------------------	--	--------------------

4.4.1.3 Fully coherent processing device

R_{SPTFW}	A Fully Coherent Expansion Chiplet contains one or more fully coherent processing devices.	(<i>LEVEL-0</i>)
-----------------------------------	--	--------------------

4.4.1.4 I/O coherent processing device

R_{FQPM} A [Fully Coherent Expansion Chiplet](#) contains zero or more I/O coherent processing devices. (LEVEL-0)

4.4.1.5 I/O coherent I/O device

R_{QDMPX} A [Fully Coherent Expansion Chiplet](#) contains zero or more I/O coherent I/O devices. (LEVEL-0)

4.4.1.6 Non-coherent I/O device

R_{LYVBX} A [Fully Coherent Expansion Chiplet](#) contains zero or more non-coherent I/O devices. (LEVEL-0)

4.4.1.7 System memory

R_{LVJXD} A [Fully Coherent Expansion Chiplet](#) might contain system main memory, either integrated or directly connected. (LEVEL-0)

R_{LFYRB} The system main memory in or directly connected to a [Fully Coherent Expansion Chiplet](#) has associated serialization functionality and might have associated hardware coherency functionality. This functionality is contained in the [Fully Coherent Expansion Chiplet](#). (LEVEL-0)

R_{XVWQF} The system main memory in or directly connected to a [Fully Coherent Expansion Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

R_{WHJVN} Any system addressable memory in the system is available through a [Fully Coherent Expansion Chiplet](#) to agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{QQTXS} Any coherent caches in a [Fully Coherent Expansion Chiplet](#) receive coherency protocol accesses from any coherent agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{RDGJV} Any device shared memory in or directly connected to a [Fully Coherent Expansion Chiplet](#) receives accesses from agents in the same chiplet or other chiplets. (LEVEL-0)

4.4.1.7.1 MMU location

I_{RSWCG} Agents in a [Fully Coherent Expansion Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in [Figure 4.8](#).

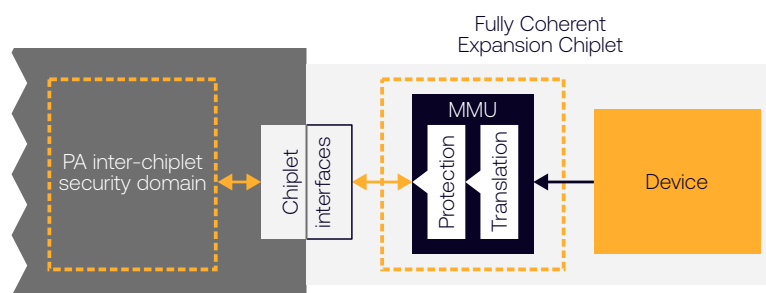


Figure 4.8: Fully Coherent Expansion Chiplet MMU location

R_{PBNCH} I/O coherent or non-coherent agents in a **Fully Coherent Expansion Chiplet** access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet.

(LEVEL-0)

R_{FDDHQ} Fully coherent agents in a **Fully Coherent Expansion Chiplet** access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet.

(LEVEL-0)

4.4.1.8 Interrupt handling

I_{FJGYD} This section concerns interrupts for Application PEs that run the main operating system.

A **Fully Coherent Expansion Chiplet** can be a source of interrupts, or convey interrupts from other chiplets. Therefore, the chiplet can:

- Generate an interrupt internally for a PE in another chiplet
- Receive an interrupt from another chiplet for a PE in another chiplet

R_{FXKVK} A **Fully Coherent Expansion Chiplet** can contain an interrupt controller component.

(LEVEL-0)

I_{DYPHT} The chiplet routes interrupts as follows:

- If the chiplet has an interrupt controller component, the interrupt controller component in the **Fully Coherent Expansion Chiplet** receives interrupts and forwards them to the interrupt controller component in the chiplet that contains the target PE.
- If the chiplet does not have an interrupt controller component, it forwards interrupts as MSIs to an interrupt controller component in a connected chiplet.

Figure 4.9 shows a simplified interrupt path between chiplets that both contain an interrupt controller component.

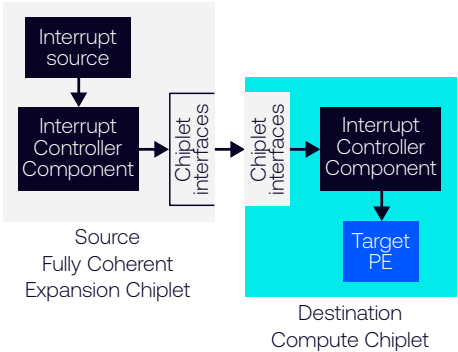


Figure 4.9: Example interrupt communication between fully coherent expansion and compute chiplets

R_{ZRPHF} Where a **Fully Coherent Expansion Chiplet** contains an interrupt controller component, the interrupt controller component receives interrupts from within the same chiplet.

(LEVEL-0)

R_{QZSYG} Where a **Fully Coherent Expansion Chiplet** contains an interrupt controller component, it can receive interrupts from other chiplets that do not contain an interrupt controller component.

(LEVEL-0)

R_{PWHNJ} Where a **Fully Coherent Expansion Chiplet** contains an interrupt controller component, the interrupt controller component forwards interrupts to the interrupt controller component in the chiplet of the target PE.

(LEVEL-0)

R_{RNNV} Where a [Fully Coherent Expansion Chiplet](#) does not contain an interrupt controller component, it transmits interrupts as MSIs to an interrupt controller component in a connected chiplet. (LEVEL-0)

4.4.1.9 Chiplet system control

R_{ZNXXR} A [Fully Coherent Expansion Chiplet](#) contains a system controller (see [System Controller](#)). (LEVEL-0)

R_{RJMNL} The [system controller](#) in a [Fully Coherent Expansion Chiplet](#) is a secondary system controller. (LEVEL-0)

R_{XLWKP} A [Fully Coherent Expansion Chiplet](#) contains one or more system control agents (see [System Control Agent](#)). (LEVEL-0)

4.4.1.10 Security

R_{FFWWY} A [Fully Coherent Expansion Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{BMPMM} A [trusted security agent](#) in a [Fully Coherent Expansion Chiplet](#) is a secondary trusted security agent. (LEVEL-0)

4.4.1.11 Debug

I_{TCZKV} A [Fully Coherent Expansion Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.4.1.11.1 Debug access

R_{ZDSMN} An external debugger can access the debug resources of components in a [Fully Coherent Expansion Chiplet](#) through a Debug Access Port (DAP) in the chiplet. (LEVEL-0)

R_{YSNYC} The debug resources of components in a [Fully Coherent Expansion Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets. (LEVEL-0)

R_{XRKQP} System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [Fully Coherent Expansion Chiplet](#) by using memory-mapped accesses. (LEVEL-0)

I_{MLBHG} The [Trusted Security Agent](#) in a [Fully Coherent Expansion Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{KHQGY} A [Fully Coherent Expansion Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity. (LEVEL-0)

4.4.1.12 System

I _{TRCQC}	A unique identifier is necessary for each Fully Coherent Expansion Chiplet because a system might contain more than one instance of the chiplet.
R _{VJGGP}	All chiplets in the system can access the identification register in a Fully Coherent Expansion Chiplet. The value in the register uniquely identifies the chiplet instance within the system. (LEVEL-0)

4.5 Fully Coherent Expansion (Remote Translation) Chiplet

I_xBDDVG

The **Fully Coherent Expansion (Remote Translation) Chiplet** type provides the system with application-specific acceleration. The chiplet might include an interface to external memory.

This chiplet type supports system-specific expansion with complex fully coherent processing devices. The trust boundary with the connected host chiplet is at the chiplet interface, for example, when a system integrates third-party solutions. The chiplet is required to communicate by using physical addresses to support generic integration with coherent protocols.

Remote address translation operates as follows:

1. The device in the chiplet requests an address translation from an MMU in the host chiplet and stores the responses locally.
2. The device applies the received address translation to its transactions. It presents the transactions to the host chiplet on the chiplet interface with physical addresses.
3. The device receives address translation invalidations and discards the locally stored translation information.

This chiplet uses the following interfaces and protocols:

- A configuration interface to program the device
- An interface to request address translations
- An interface to send Message Signaled Interrupts (MSIs)
- A DMA protocol for data transfer

Figure 4.10 shows a simplified example of a **Fully Coherent Expansion (Remote Translation) Chiplet**.

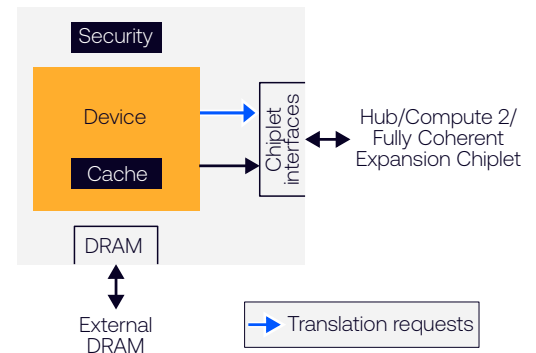


Figure 4.10: Fully Coherent Expansion (Remote Translation) Chiplet example

A **Fully Coherent Expansion (Remote Translation) Chiplet** is a constituent of the following system types, which specify additional system-level rules that apply to this chiplet type:

- **Compute & Hub System**
- **Compute Tile System**

4.5.1 Chiplet requirements

The following requirements define the [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) type.

4.5.1.1 Chiplet-to-chiplet interfaces

R_{KGJQZ} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with one of the following mutually exclusive chiplet types:

- A [Hub Chiplet](#)
- A [Compute 2 Chiplet](#)
- A [Fully Coherent Expansion Chiplet](#)

(LEVEL-0)

R_{VZRNQ} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [Hub Chiplet](#).

(LEVEL-0)

R_{ZKKWN} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [Compute 2 Chiplet](#).

(LEVEL-0)

R_{VGSNV} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [Fully Coherent Expansion Chiplet](#).

(LEVEL-0)

I_{FRDFT} See [Fully Coherent Expansion \(Remote Translation\) Chiplet Interface Requirements](#).

4.5.1.2 Application PE

R_{KSDBW} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) does not contain any Application PEs.

(LEVEL-0)

4.5.1.3 Fully coherent processing device

R_{MHJBP} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains one or more fully coherent processing devices.

(LEVEL-0)

R_{BKXFJ} Any fully coherent processing device in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is presented to the system as a PCIe Endpoint [6].

(LEVEL-FULL)

4.5.1.4 I/O coherent processing device

R_{GGJVV} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains zero or more I/O coherent processing devices.

(LEVEL-0)

R_{LFOWL} Any I/O coherent processing device in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is presented to the system as a PCIe Endpoint [6].

(LEVEL-FULL)

4.5.1.5 I/O coherent I/O device

R_{WDGDV} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains zero or more I/O coherent I/O devices.

(LEVEL-0)

R_{KWBLT} Any I/O coherent I/O device in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is presented to the system as a PCIe Endpoint [6].

(LEVEL-FULL)

4.5.1.6 Non-coherent I/O device

R _{KLQNT}	A Fully Coherent Expansion (Remote Translation) Chiplet contains zero or more non-coherent I/O devices. (LEVEL-0)
R _{JNZML}	Any non-coherent I/O device in a Fully Coherent Expansion (Remote Translation) Chiplet is presented to the system as a PCIe Endpoint [6]. (LEVEL-FULL)

4.5.1.7 System memory

R _{WNHGS}	A Fully Coherent Expansion (Remote Translation) Chiplet might contain system main memory, either integrated or directly connected. (LEVEL-0)
R _{SBTVJ}	The system main memory in or directly connected to a Fully Coherent Expansion (Remote Translation) Chiplet has associated serialization functionality and might have associated hardware coherency functionality. This functionality is contained in the Fully Coherent Expansion (Remote Translation) Chiplet . (LEVEL-0)
R _{VHJYG}	A Fully Coherent Expansion (Remote Translation) Chiplet contains one or more fully coherent agents. (LEVEL-0)
R _{SVMSC}	A Fully Coherent Expansion (Remote Translation) Chiplet might contain caches that support coherency protocol accesses by coherent agents in other chiplets in the system. (LEVEL-0)
R _{FFDXH}	System software accesses memory-mapped configuration of devices in a Fully Coherent Expansion (Remote Translation) Chiplet . (LEVEL-0)

4.5.1.7.1 MMU location

I _{TJMMT}	Agents in a Fully Coherent Expansion (Remote Translation) Chiplet access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is not contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in Figure 4.11 .
--------------------	---

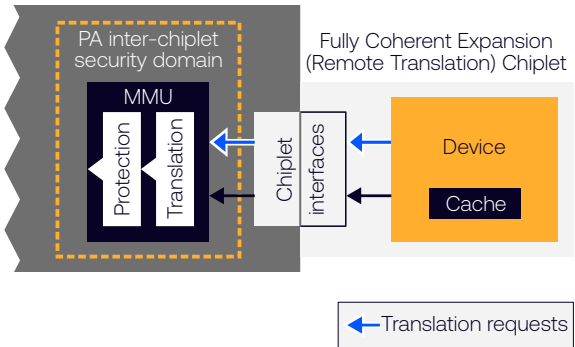


Figure 4.11: Fully Coherent Expansion (Remote Translation) Chiplet MMU location

R _{XMDFB}	I/O coherent or non-coherent agents in a Fully Coherent Expansion (Remote Translation) Chiplet access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is not contained in the chiplet. (LEVEL-0)
--------------------	--

R_{LMWRD} Fully coherent agents in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is not contained in the chiplet.

(LEVEL-0)

R_{DDBSL} Any agents in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) that request address translation information do so from an MMU. That MMU is not in the chiplet.

(LEVEL-0)

4.5.1.8 Interrupt handling

I_{LKYKB} This section concerns interrupts for Application PEs that run the main operating system.

A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) can be a source of interrupts. The chiplet forwards these interrupts as MSIs.

R_{PJVLK} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) transmits interrupts as MSIs to a connected chiplet.

(LEVEL-0)

4.5.1.9 Chiplet system control

R_{RVBSS} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains a system controller (see [System Controller](#)).

(LEVEL-0)

R_{CWKWQ} The [system controller](#) in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is a secondary system controller.

(LEVEL-0)

R_{NYNCY} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains one or more system control agents (see [System Control Agent](#)).

(LEVEL-0)

4.5.1.10 Security

R_{MBSCP} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{XVZTS} A [trusted security agent](#) in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is a secondary trusted security agent.

(LEVEL-0)

4.5.1.11 Debug

I_{NPJVH} A [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.5.1.11.1 Debug access

R_{JSDXW} An external debugger can access the debug resources of components in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) through a Debug Access Port (DAP) in the chiplet.

(LEVEL-0)

R_{BVKJT} The debug resources of components in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets.

(LEVEL-0)

R _{XYXHC}	System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a Fully Coherent Expansion (Remote Translation) Chiplet by using memory-mapped accesses.	(LEVEL-0)
I _{GXGWX}	The Trusted Security Agent in a Fully Coherent Expansion (Remote Translation) Chiplet might restrict debug access to enforce the hardware security lifecycle state of the chiplet. <i>Authenticated debug</i> provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].	
R _{PGHRF}	A Fully Coherent Expansion (Remote Translation) Chiplet might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an <i>Authenticated debug</i> capability that enables debug access after the trusted security agent verifies identity.	(LEVEL-0)

4.5.1.12 System

I _{LKJSJ}	A unique identifier is necessary for each Fully Coherent Expansion (Remote Translation) Chiplet because a system might contain more than one instance of the chiplet.	
R _{XBJSZ}	All chiplets in the system can access the identification register in a Fully Coherent Expansion (Remote Translation) Chiplet . The value in the register uniquely identifies the chiplet instance within the system.	(LEVEL-0)

4.6 I/O Coherent Expansion (Translated) Chiplet

I_{NKFQB}

The **I/O Coherent Expansion (Translated) Chiplet** type provides the system with application-specific acceleration and I/O coherent I/O. The chiplet might include interfaces to external DRAM for private use by the application-specific acceleration and to other I/O coherent or non-coherent chiplet types.

Devices in the chiplet access coherent system memory by using physical addresses (PAs). A memory management unit (MMU) in the chiplet generates the PAs.

Figure 4.12 shows simplified examples of a **I/O Coherent Expansion (Translated) Chiplet**.

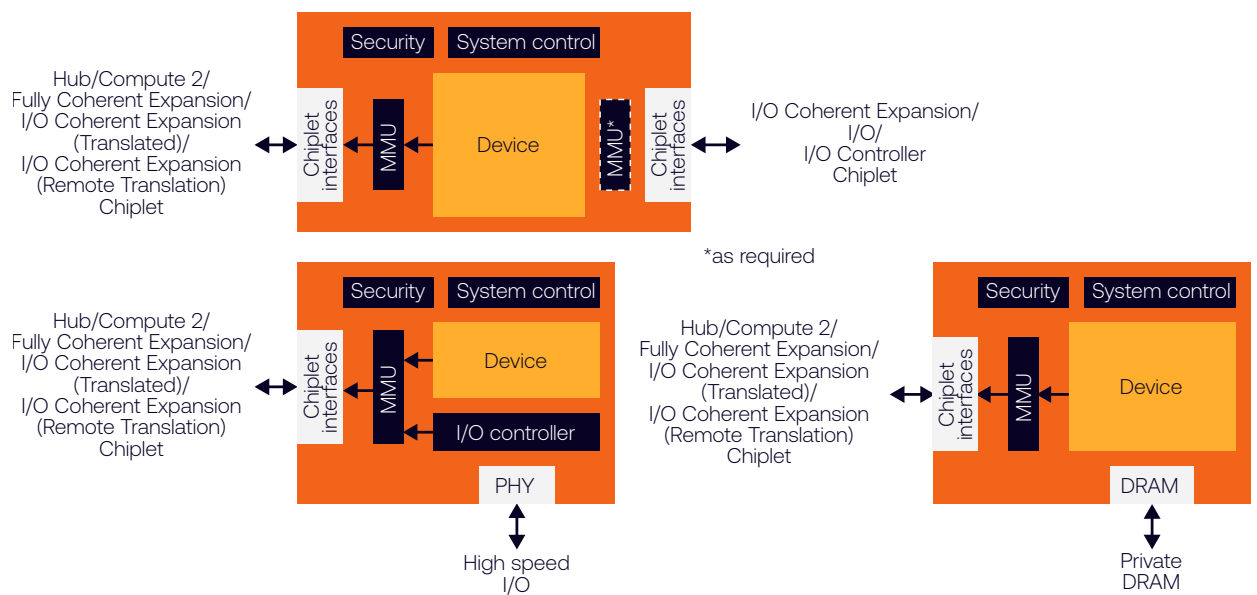


Figure 4.12: I/O Coherent Expansion (Translated) Chiplet examples

A **I/O Coherent Expansion (Translated) Chiplet** is a constituent of the following system types, which specify additional system-level rules that apply to this chiplet type:

- **Compute & Hub System**
- **Compute Tile System**

4.6.1 Chiplet requirements

The following requirements define the [I/O Coherent Expansion \(Translated\) Chiplet](#) type.

4.6.1.1 Chiplet-to-chiplet interfaces

R_{YTTM} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with one of the following mutually exclusive chiplet types:

- A [Hub Chiplet](#)
- A [Compute 2 Chiplet](#)
- A [Fully Coherent Expansion Chiplet](#)
- An [I/O Coherent Expansion \(Translated\) Chiplet](#)

(LEVEL-0)

R_{MPKFL} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or exactly one [Hub Chiplet](#).

(LEVEL-0)

R_{LGNWX} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or exactly one [Compute 2 Chiplet](#).

(LEVEL-0)

R_{RWJXM} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or exactly one [Fully Coherent Expansion Chiplet](#).

(LEVEL-0)

R_{NJQMS} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or more [I/O Coherent Expansion \(Translated\) Chiplets](#).

(LEVEL-0)

R_{FHVQW} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or more [I/O Coherent Expansion \(Untranslated\) Chiplets](#).

(LEVEL-0)

R_{XPQFB} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or more [I/O Coherent Expansion \(Remote Translation\) Chiplets](#).

(LEVEL-0)

R_{DFJJB} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or more [I/O Chiplets](#).

(LEVEL-0)

R_{LRYYF} A [I/O Coherent Expansion \(Translated\) Chiplet](#) interfaces with zero or more [I/O Controller Chiplets](#).

(LEVEL-0)

I_{HKZRJ} See [Fully Coherent Expansion Chiplet Interface Requirements](#).

I_{MTRFG} See [I/O Coherent Expansion \(Translated\) Chiplet Interface Requirements](#).

I_{LYJSR} See [I/O Coherent Expansion \(Untranslated\) Chiplet Interface Requirements](#).

I_{FNWCB} See [I/O Coherent Expansion \(Remote Translation\) Chiplet Interface Requirements](#).

I_{SGTPD} See [I/O Chiplet Interface Requirements](#).

I_{DBYZT} See [I/O Controller Chiplet Interface Requirements](#).

4.6.1.2 Application PE

R_{NKBDT} A [I/O Coherent Expansion \(Translated\) Chiplet](#) does not contain any Application PEs.

(LEVEL-0)

4.6.1.3 Fully coherent processing device

R_{GPNPF} A [I/O Coherent Expansion \(Translated\) Chiplet](#) does not contain any fully coherent processing devices.

(LEVEL-0)

4.6.1.4 I/O coherent processing device

R_{JKNBM} A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains one or more I/O coherent processing devices. (LEVEL-0)

4.6.1.5 I/O coherent I/O device

R_{YLRSX} A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains zero or more I/O coherent I/O devices. (LEVEL-0)

4.6.1.6 Non-coherent I/O device

R_{TXQKB} A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains zero or more non-coherent I/O devices. (LEVEL-0)

4.6.1.7 System memory

R_{PGQRS} A [I/O Coherent Expansion \(Translated\) Chiplet](#) might contain Non-cacheable system main memory, either integrated or directly connected. (LEVEL-0)

R_{DVDHL} The system main memory in or directly connected to a [I/O Coherent Expansion \(Translated\) Chiplet](#) has associated serialization functionality. This functionality is contained in the [I/O Coherent Expansion \(Translated\) Chiplet](#). (LEVEL-0)

R_{XCBFQ} A [I/O Coherent Expansion \(Translated\) Chiplet](#) does not contain any fully coherent agents. (LEVEL-0)

R_{HSWKR} Any system addressable memory in the system is available through a [I/O Coherent Expansion \(Translated\) Chiplet](#) to agents within the chiplet or in other chiplets in the system. (LEVEL-0)

R_{BGYTR} A [I/O Coherent Expansion \(Translated\) Chiplet](#) does not contain any caches that support coherency protocol accesses by coherent agents in other chiplets in the system. (LEVEL-0)

4.6.1.7.1 MMU location

I_{LWGQZ} Agents in a [I/O Coherent Expansion \(Translated\) Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in [Figure 4.13](#).

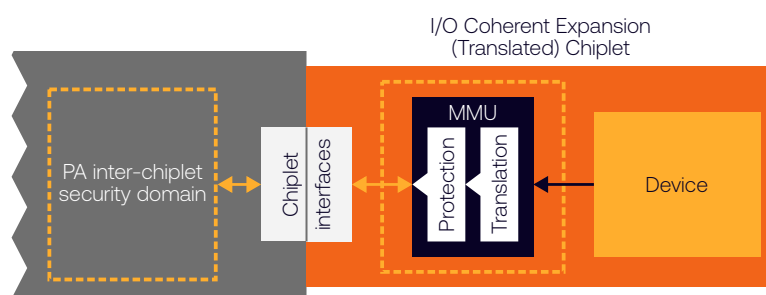


Figure 4.13: I/O Coherent Expansion (Translated) Chiplet MMU location

R_{XHLRJ} I/O coherent or non-coherent agents in a [I/O Coherent Expansion \(Translated\) Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is contained in the chiplet. (LEVEL-0)

4.6.1.8 Interrupt handling

I_{HFHBL}

This section concerns interrupts for Application PEs that run the main operating system.

A [I/O Coherent Expansion \(Translated\) Chiplet](#) can be a source of interrupts, or convey interrupts from other chiplets. Therefore, the chiplet can:

- Generate an interrupt internally for a PE in another chiplet
- Receive an interrupt from another chiplet for a PE in another chiplet

R_{RDSRT}

A [I/O Coherent Expansion \(Translated\) Chiplet](#) can contain an interrupt controller component.

(LEVEL-0)

I_{TFVYT}

The chiplet routes interrupts as follows:

- If the chiplet has an interrupt controller component, that component receives interrupts and forwards them to the interrupt controller component in the chiplet that contains the target PE.
- If the chiplet does not have an interrupt controller component, it forwards interrupts as MSIs to an interrupt controller component in a connected chiplet.

R_{ZHDSR}

Where a [I/O Coherent Expansion \(Translated\) Chiplet](#) contains an interrupt controller component, the interrupt controller component receives interrupts from within the same chiplet.

(LEVEL-0)

R_{LPFHQ}

Where a [I/O Coherent Expansion \(Translated\) Chiplet](#) contains an interrupt controller component, it can receive interrupts from other chiplets that do not contain an interrupt controller component.

(LEVEL-0)

R_{SXYNL}

Where a [I/O Coherent Expansion \(Translated\) Chiplet](#) contains an interrupt controller component, the interrupt controller component forwards interrupts to the interrupt controller component in the chiplet of the target PE.

(LEVEL-0)

R_{SDNYR}

Where a [I/O Coherent Expansion \(Translated\) Chiplet](#) does not contain an interrupt controller component, it transmits interrupts as MSIs to an interrupt controller component in a connected chiplet.

(LEVEL-0)

4.6.1.9 Chiplet system control

R_{TWRHN}

A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains a system controller (see [System Controller](#)).

(LEVEL-0)

R_{ZHTQD}

The [system controller](#) in a [I/O Coherent Expansion \(Translated\) Chiplet](#) is a secondary system controller.

(LEVEL-0)

R_{PHQVC}

A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains one or more system control agents (see [System Control Agent](#)).

(LEVEL-0)

4.6.1.10 Security

R_{LCWDK}

A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{JTTCQ}

A [trusted security agent](#) in a [I/O Coherent Expansion \(Translated\) Chiplet](#) is a secondary trusted security agent.

(LEVEL-0)

4.6.1.11 Debug

I_{QDJFZ} A [I/O Coherent Expansion \(Translated\) Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.6.1.11.1 Debug access

R_{ZKGSN} An external debugger can access the debug resources of components in a [I/O Coherent Expansion \(Translated\) Chiplet](#) through a Debug Access Port (DAP) in the chiplet. (LEVEL-0)

R_{HQHCN} The debug resources of components in a [I/O Coherent Expansion \(Translated\) Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets. (LEVEL-0)

R_{QGKMH} System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [I/O Coherent Expansion \(Translated\) Chiplet](#) by using memory-mapped accesses. (LEVEL-0)

I_{RGVSH} The [Trusted Security Agent](#) in a [I/O Coherent Expansion \(Translated\) Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{RBRHD} A [I/O Coherent Expansion \(Translated\) Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity. (LEVEL-0)

4.6.1.12 System

I_{TCKLP} A unique identifier is necessary for each [I/O Coherent Expansion \(Translated\) Chiplet](#) because a system might contain more than one instance of the chiplet.

R_{DDQVP} All chiplets in the system can access the identification register in a [I/O Coherent Expansion \(Translated\) Chiplet](#). The value in the register uniquely identifies the chiplet instance within the system. (LEVEL-0)

4.7 I/O Coherent Expansion (Untranslated) Chiplet

I_{TLKDT}

The **I/O Coherent Expansion (Untranslated) Chiplet** type provides the system with application-specific acceleration and I/O coherent I/O. The chiplet might include an interface to external DRAM for private use by the application-specific acceleration and interfaces to other I/O coherent or non-coherent chiplet types.

The chiplet provides system-specific expansion for existing I/O coherent processing devices with minimal additional complexity. Attached chiplets manage system control, interrupt handling, and address translation.

Figure 4.14 shows simplified examples of a **I/O Coherent Expansion (Untranslated) Chiplet**.

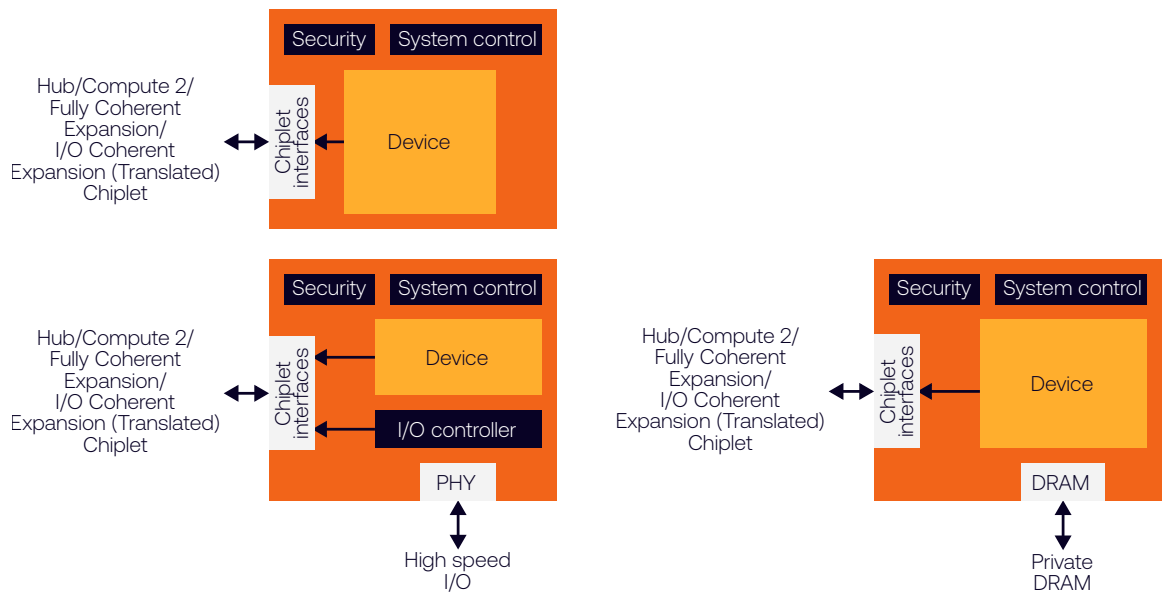


Figure 4.14: I/O Coherent Expansion (Untranslated) Chiplet examples

A **I/O Coherent Expansion (Untranslated) Chiplet** is a constituent of the following system types, which specify additional system-level rules that apply to this chiplet type:

- **Compute & Hub System**
- **Compute Tile System**

4.7.1 Chiplet requirements

The following requirements define the [I/O Coherent Expansion \(Untranslated\) Chiplet](#) type.

4.7.1.1 Chiplet-to-chiplet interfaces

[R_{YWWFY}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) interfaces with one of the following mutually exclusive chiplet types:

- A [Hub Chiplet](#)
- A [Compute 2 Chiplet](#)
- A [Fully Coherent Expansion Chiplet](#)
- An [I/O Coherent Expansion \(Translated\) Chiplet](#)

(*LEVEL-0*)

[R_{GNBSW}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) interfaces with zero or exactly one [Hub Chiplet](#).

(*LEVEL-0*)

[R_{PZFMT}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) interfaces with zero or exactly one [Compute 2 Chiplet](#).

(*LEVEL-0*)

[R_{WFLXB}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) interfaces with zero or exactly one [Fully Coherent Expansion Chiplet](#).

(*LEVEL-0*)

[R_{JHQPD}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) interfaces with zero or exactly one [I/O Coherent Expansion \(Translated\) Chiplet](#).

(*LEVEL-0*)

[I_{JKFCD}](#) See [Fully Coherent Expansion Chiplet Interface Requirements](#).

[I_{DWDHZ}](#) See [I/O Coherent Expansion \(Untranslated\) Chiplet Interface Requirements](#).

[I_{NCBJR}](#) See [I/O Coherent Expansion \(Translated\) Chiplet Interface Requirements](#).

4.7.1.2 Application PE

[R_{BENCHJ}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) does not contain any Application PEs.

(*LEVEL-0*)

4.7.1.3 Fully coherent processing device

[R_{DGKSY}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) does not contain any fully coherent processing devices.

(*LEVEL-0*)

4.7.1.4 I/O coherent processing device

[R_{HKCSG}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) contains one or more I/O coherent processing devices.

(*LEVEL-0*)

4.7.1.5 I/O coherent I/O device

[R_{YHDSL}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) contains zero or more I/O coherent I/O devices.

(*LEVEL-0*)

4.7.1.6 Non-coherent I/O device

[R_{DPLFM}](#) A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) contains zero or more non-coherent I/O devices.

(*LEVEL-0*)

4.7.1.7 System memory

R_{NDGMO} A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) does not contain system main memory, either integrated or directly connected.

(LEVEL-0)

R_{RGMEN} A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) does not contain any fully coherent agents.

(LEVEL-0)

R_{NPHFD} A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) does not contain any caches that support coherency protocol accesses by coherent agents in other chiplets in the system.

(LEVEL-0)

R_{KFPMT} System software accesses memory-mapped configuration of devices in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#).

(LEVEL-0)

4.7.1.7.1 MMU location

I_{JPSMF} Agents in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is not contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in [Figure 4.15](#).

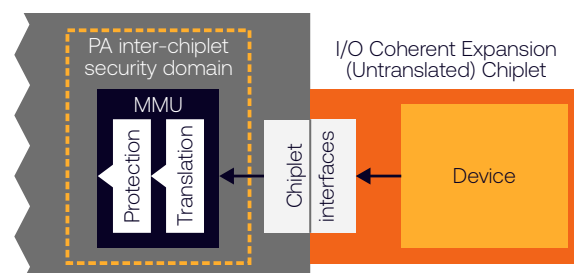


Figure 4.15: I/O Coherent Expansion (Untranslated) Chiplet MMU location

R_{PQGLG} I/O coherent or non-coherent agents in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is not contained in the chiplet.

(LEVEL-0)

4.7.1.8 Interrupt handling

I_{JCLGP} This section concerns interrupts for Application PEs that run the main operating system.

The components on a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) can generate interrupts. The chiplet forwards these interrupts as MSIs.

R_{RBSQV} A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) transmits interrupts as MSIs to a connected chiplet.

(LEVEL-0)

4.7.1.9 Chiplet system control

R_{KMTCS} A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) contains one or more system control agents (see [System Control Agent](#)).

(LEVEL-0)

R_{JMJYS} The primary [system controller](#) controls the Power Management Integrated Circuit (PMIC) associated with a [I/O Coherent Expansion \(Untranslated\) Chiplet](#).

(LEVEL-0)

4.7.1.10 Security

R_{HHJCC}

A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{MGP_{TM}}

A [trusted security agent](#) in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) is a secondary trusted security agent.

(LEVEL-0)

4.7.1.11 Debug

I_{ZQBPJ}

A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.7.1.11.1 Debug access

R_{PJLCZ}

An external debugger can access the debug resources of components in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) through a Debug Access Port (DAP) in the chiplet.

(LEVEL-0)

R_{MLLS}

The debug resources of components in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets.

(LEVEL-0)

R_{HHNSD}

System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) by using memory-mapped accesses.

(LEVEL-0)

I_{RHYFK}

The [Trusted Security Agent](#) in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{NJLVJ}

A [I/O Coherent Expansion \(Untranslated\) Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity.

(LEVEL-0)

4.7.1.12 System

I_{VFM_{RV}}

A unique identifier is necessary for each [I/O Coherent Expansion \(Untranslated\) Chiplet](#) because a system might contain more than one instance of the chiplet.

R_{WSRW}

All chiplets in the system can access the identification register in a [I/O Coherent Expansion \(Untranslated\) Chiplet](#). The value in the register uniquely identifies the chiplet instance within the system.

(LEVEL-0)

4.8.1 Chiplet requirements

The following requirements define the [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) type.

4.8.1.1 Chiplet-to-chiplet interfaces

R_{MCTLV} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with one of the following mutually exclusive chiplet types:

- A [Hub Chiplet](#)
- A [Compute 2 Chiplet](#)
- A [Fully Coherent Expansion Chiplet](#)
- An [I/O Coherent Expansion \(Translated\) Chiplet](#)

(*LEVEL-0*)

R_{JFVWW} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [Hub Chiplet](#).

(*LEVEL-0*)

R_{LYDWT} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [Compute 2 Chiplet](#).

(*LEVEL-0*)

R_{LVZJJ} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [Fully Coherent Expansion Chiplet](#).

(*LEVEL-0*)

R_{YKSJV} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) interfaces with zero or exactly one [I/O Coherent Expansion \(Translated\) Chiplet](#).

(*LEVEL-0*)

I_{KMCHP} See [I/O Coherent Expansion \(Remote Translation\) Chiplet Interface Requirements](#).

4.8.1.2 Application PE

R_{DFKTT} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) does not contain any Application PEs.

(*LEVEL-0*)

4.8.1.3 Fully coherent processing device

R_{XKHNG} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) does not contain any fully coherent processing devices.

(*LEVEL-0*)

4.8.1.4 I/O coherent processing device

R_{RXVEP} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains one or more I/O coherent processing devices.

(*LEVEL-0*)

R_{HBQJS} Any I/O coherent processing device in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is presented to the system as a PCIe Endpoint [6].

(*LEVEL-FULL*)

4.8.1.5 I/O coherent I/O device

R_{SJJRK} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains zero or more I/O coherent I/O devices.

(*LEVEL-0*)

R_{CF CZN} Any I/O coherent I/O device in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is presented to the system as a PCIe Endpoint [6].

(*LEVEL-FULL*)

4.8.1.6 Non-coherent I/O device

R_{VGYHF} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains zero or more non-coherent I/O devices. (LEVEL-0)

R_{JNYGP} Any non-coherent I/O device in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is presented to the system as a PCIe Endpoint [6]. (LEVEL-FULL)

4.8.1.7 System memory

R_{GXVLS} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) might contain Non-cacheable system main memory, either integrated or directly connected. (LEVEL-0)

R_{JPCFZ} The system main memory in or directly connected to a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) has associated serialization functionality. This functionality is contained in the [I/O Coherent Expansion \(Remote Translation\) Chiplet](#). (LEVEL-0)

R_{MGPMV} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) does not contain any fully coherent agents. (LEVEL-0)

R_{MJDDS} A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) does not contain any caches that support coherency protocol accesses by coherent agents in other chiplets in the system. (LEVEL-0)

R_{VLFQJ} System software accesses memory-mapped configuration of devices in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#). (LEVEL-0)

4.8.1.7.1 MMU location

I_{CKFKS} Agents in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is not contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in [Figure 4.17](#).

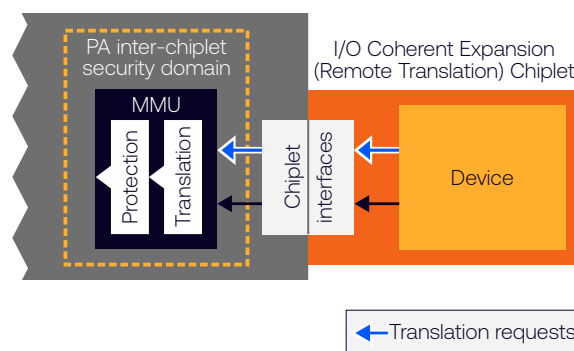


Figure 4.17: I/O Coherent Expansion (Remote Translation) Chiplet MMU location

R_{GNNND} I/O coherent or non-coherent agents in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is not contained in the chiplet. (LEVEL-0)

R_{JWZKZ} Any agents in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) that request address translation information do so from an MMU. That MMU is not in the chiplet. (LEVEL-0)

4.8.1.8 Interrupt handling

I_{JQSTB}

This section concerns interrupts for Application PEs that run the main operating system.

The components on a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) can generate interrupts. The chiplet forwards these interrupts as MSIs.

R_{SSKPN}

A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) transmits interrupts as MSIs to a connected chiplet. (LEVEL-0)

4.8.1.9 Chiplet system control

R_{XCBFD}

A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains a system controller (see [System Controller](#)). (LEVEL-0)

R_{RHJMG}

The [system controller](#) in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is a secondary system controller. (LEVEL-0)

R_{PWMCT}

A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains one or more system control agents (see [System Control Agent](#)). (LEVEL-0)

4.8.1.10 Security

R_{VVWXF}

A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{MTJGB}

A [trusted security agent](#) in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is a secondary trusted security agent. (LEVEL-0)

4.8.1.11 Debug

I_{JGNHX}

A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.8.1.11.1 Debug access

R_{WXTJK}

An external debugger can access the debug resources of components in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) through a Debug Access Port (DAP) in the chiplet. (LEVEL-0)

R_{NTMSD}

The debug resources of components in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets. (LEVEL-0)

R_{SSBPB}

System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) by using memory-mapped accesses. (LEVEL-0)

I_{ZBRXS}

The [Trusted Security Agent](#) in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{VDGNN}

A [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity.

(LEVEL-0)

4.8.1.12 System

I_{QRVVW}

A unique identifier is necessary for each [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) because a system might contain more than one instance of the chiplet.

R_{DZVCK}

All chiplets in the system can access the identification register in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#). The value in the register uniquely identifies the chiplet instance within the system.

(LEVEL-0)

4.9 I/O Chiplet

I_{GXMN}

The **I/O Chiplet** type provides the system with I/O coherent I/O.

This chiplet provides system-specific expansion for I/O coherent I/O devices with minimal additional complexity. Attached chiplets manage system control, interrupt handling, and address translation.

Figure 4.18 shows a simplified example of a **I/O Chiplet**.

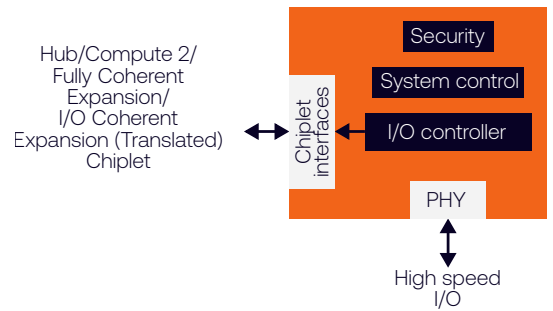


Figure 4.18: I/O Chiplet example

A **I/O Chiplet** is a constituent of the following system types, which specify additional system-level rules that apply to this chiplet type:

- **Compute & Hub System**
- **Compute Tile System**

4.9.1 Chiplet requirements

The following requirements define the [I/O Chiplet](#) type.

4.9.1.1 Chiplet-to-chiplet interfaces

R_{FKNWS}	A I/O Chiplet interfaces with one of the following mutually exclusive chiplet types: • A Hub Chiplet • A Compute 2 Chiplet • A Fully Coherent Expansion Chiplet • An I/O Coherent Expansion (Translated) Chiplet	(LEVEL-0)
R_{WWGKL}	A I/O Chiplet interfaces with zero or exactly one Hub Chiplet .	(LEVEL-0)
R_{HCYCZ}	A I/O Chiplet interfaces with zero or exactly one Compute 2 Chiplet .	(LEVEL-0)
R_{CFTXC}	A I/O Chiplet interfaces with zero or exactly one Fully Coherent Expansion Chiplet .	(LEVEL-0)
R_{FHZFP}	A I/O Chiplet interfaces with zero or exactly one I/O Coherent Expansion (Translated) Chiplet .	(LEVEL-0)
I_{DVHCD}	See Fully Coherent Expansion Chiplet Interface Requirements .	
I_{KWXNR}	See I/O Chiplet Interface Requirements .	
I_{NJNHQ}	See I/O Coherent Expansion (Translated) Chiplet Interface Requirements .	

4.9.1.2 Application PE

R_{ZQPNC}	A I/O Chiplet does not contain any Application PEs.	(LEVEL-0)
--------------------	---	-----------

4.9.1.3 Fully coherent processing device

R_{JHMFZ}	A I/O Chiplet does not contain any fully coherent processing devices.	(LEVEL-0)
--------------------	---	-----------

4.9.1.4 I/O coherent processing device

R_{KWXTX}	A I/O Chiplet does not contain any I/O coherent processing devices.	(LEVEL-0)
--------------------	---	-----------

4.9.1.5 I/O coherent I/O device

R_{CPTMG}	A I/O Chiplet contains one or more I/O coherent I/O devices.	(LEVEL-0)
--------------------	--	-----------

4.9.1.6 Non-coherent I/O device

$R_{\text{ZMH CY}}$	A I/O Chiplet contains zero or more non-coherent I/O devices.	(LEVEL-0)
---------------------	---	-----------

4.9.1.7 System memory

R_{RTPGY}	A I/O Chiplet might contain system main memory, either integrated or directly connected.	(LEVEL-0)
$R_{\text{YRK BX}}$	A I/O Chiplet does not contain any fully coherent agents.	(LEVEL-0)

R_{JSKVP} A [I/O Chiplet](#) does not contain any caches that support coherency protocol accesses by coherent agents in other chiplets in the system. (LEVEL-0)

R_{RRSYS} System software accesses memory-mapped configuration of devices in a [I/O Chiplet](#). (LEVEL-0)

4.9.1.7.1 MMU location

I_{NFQMM} Agents in a [I/O Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is not contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in [Figure 4.19](#).

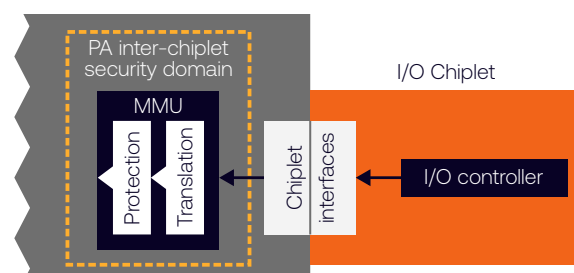


Figure 4.19: I/O Chiplet MMU location

R_{MGCKS} I/O coherent or non-coherent agents in a [I/O Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is not contained in the chiplet. (LEVEL-0)

4.9.1.8 Interrupt handling

I_{GPSWK} This section concerns interrupts for Application PEs that run the main operating system. The components on a [I/O Chiplet](#) can generate interrupts. The chiplet forwards these interrupts as MSIs.

4.9.1.9 Chiplet system control

R_{LCHML} A [I/O Chiplet](#) contains one or more system control agents (see [System Control Agent](#)). (LEVEL-0)

R_{XJFYN} The primary [system controller](#) controls the Power Management Integrated Circuit (PMIC) associated with a [I/O Chiplet](#). (LEVEL-0)

4.9.1.10 Security

R_{WPDRE} A [I/O Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#). (LEVEL-0)

R_{BNDMC} A [trusted security agent](#) in a [I/O Chiplet](#) is a secondary trusted security agent. (LEVEL-0)

4.9.1.11 Debug

I_{HBMGP} A [I/O Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.9.1.11.1 Debug access

R_{YFNFR} An external debugger can access the debug resources of components in a [I/O Chiplet](#) through a Debug Access Port (DAP) in the chiplet.

(LEVEL-0)

R_{WYPZL} The debug resources of components in a [I/O Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets.

(LEVEL-0)

R_{GWHVL} System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [I/O Chiplet](#) by using memory-mapped accesses.

(LEVEL-0)

I_{WSNCQ} The [Trusted Security Agent](#) in a [I/O Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{LMZZV} A [I/O Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity.

(LEVEL-0)

4.9.1.12 System

I_{MQGLT} A unique identifier is necessary for each [I/O Chiplet](#) because a system might contain more than one instance of the chiplet.

R_{BXGXT} All chiplets in the system can access the identification register in a [I/O Chiplet](#). The value in the register uniquely identifies the chiplet instance within the system.

(LEVEL-0)

4.10 I/O Controller Chiplet

I_{DGSTG}

The **I/O Controller Chiplet** type provides high-speed I/O to the system, including physical layers (PHYs) for double data rate synchronous dynamic random-access memory (DDR SDRAM), Universal Serial Bus (USB), MIPI Camera Serial Interface 2 (MIPI CSI-2), and High-Definition Multimedia Interface (HDMI). The chiplet contains no coherent agents.

Figure 4.20 shows a simplified example of a **I/O Controller Chiplet**.

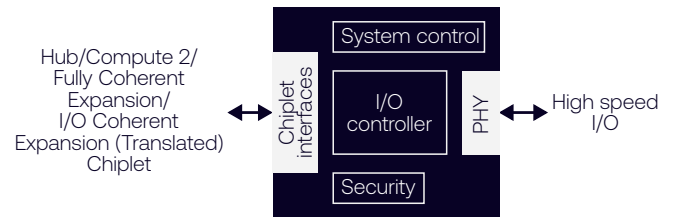


Figure 4.20: I/O Controller Chiplet example

A **I/O Controller Chiplet** is a constituent of the following system types, which specify additional system-level rules that apply to this chiplet type:

- **Compute & Hub System**
- **Compute Tile System**

4.10.1 Chiplet requirements

The following requirements define the [I/O Controller Chiplet](#) type.

4.10.1.1 Chiplet-to-chiplet interfaces

R_{FFNKP}	A I/O Controller Chiplet interfaces with one of the following mutually exclusive chiplet types: • A Hub Chiplet • A Compute 2 Chiplet • A Fully Coherent Expansion Chiplet • An I/O Coherent Expansion (Translated) Chiplet	(<i>LEVEL-0</i>)
R_{LTWZF}	A I/O Controller Chiplet interfaces with zero or exactly one Hub Chiplet .	(<i>LEVEL-0</i>)
R_{JWNXJ}	A I/O Controller Chiplet interfaces with zero or exactly one Compute 2 Chiplet .	(<i>LEVEL-0</i>)
R_{RDQYR}	A I/O Controller Chiplet interfaces with zero or exactly one Fully Coherent Expansion Chiplet .	(<i>LEVEL-0</i>)
R_{DSBQD}	A I/O Controller Chiplet interfaces with zero or exactly one I/O Coherent Expansion (Translated) Chiplet .	(<i>LEVEL-0</i>)
I_{CJMRS}	See Fully Coherent Expansion Chiplet Interface Requirements .	
I_{CPZPP}	See I/O Controller Chiplet Interface Requirements .	
I_{WCJZH}	See I/O Coherent Expansion (Translated) Chiplet Interface Requirements .	

4.10.1.2 Application PE

R_{WBVMW}	A I/O Controller Chiplet does not contain any Application PEs.	(<i>LEVEL-0</i>)
-----------------------------------	--	--------------------

4.10.1.3 Fully coherent processing device

R_{XWTVX}	A I/O Controller Chiplet does not contain any fully coherent processing devices.	(<i>LEVEL-0</i>)
-----------------------------------	--	--------------------

4.10.1.4 I/O coherent processing device

R_{BNQNG}	A I/O Controller Chiplet does not contain any I/O coherent processing devices.	(<i>LEVEL-0</i>)
-----------------------------------	--	--------------------

4.10.1.5 I/O coherent I/O device

R_{JRSRT}	A I/O Controller Chiplet does not contain any I/O coherent I/O devices.	(<i>LEVEL-0</i>)
-----------------------------------	---	--------------------

4.10.1.6 Non-coherent I/O device

R_{RZGDN}	A I/O Controller Chiplet contains one or more non-coherent I/O devices.	(<i>LEVEL-0</i>)
-----------------------------------	---	--------------------

4.10.1.7 System memory

R_{QWLMP}	A I/O Controller Chiplet might contain system main memory, either integrated or directly connected.	(<i>LEVEL-0</i>)
R_{LZRDM}	A I/O Controller Chiplet does not contain any fully coherent agents.	(<i>LEVEL-0</i>)

R_{MWPHD} A [I/O Controller Chiplet](#) does not contain any I/O coherent agents. (LEVEL-0)

R_{DWLLK} A [I/O Controller Chiplet](#) does not contain any caches that support coherency protocol accesses by coherent agents in other chiplets in the system. (LEVEL-0)

4.10.1.7.1 MMU location

I_{RPPRQ} Agents in a [I/O Controller Chiplet](#) access system addressable memory. All agents in the system access memory through a Memory Management Unit (MMU) that provides address translation and memory access protection. The MMU is not contained in the chiplet and is located inside the Physical Address (PA) inter-chiplet security domain as shown in [Figure 4.21](#).

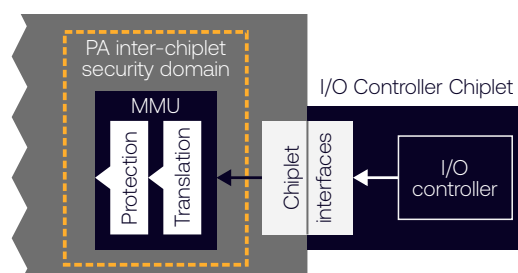


Figure 4.21: I/O Controller Chiplet MMU location

R_{QMDXK} I/O coherent or non-coherent agents in a [I/O Controller Chiplet](#) access system memory through an MMU that provides memory translation and memory access protection functions. That MMU is not contained in the chiplet. (LEVEL-0)

4.10.1.8 Interrupt handling

I_{ZVJZG} This section concerns interrupts for Application PEs that run the main operating system.

The components on a [I/O Controller Chiplet](#) can generate interrupts. The chiplet forwards these interrupts as MSIs.

R_{MDBRR} A [I/O Controller Chiplet](#) transmits interrupts as MSIs to a connected chiplet. (LEVEL-0)

4.10.1.9 Chiplet system control

R_{KZQVC} A [I/O Controller Chiplet](#) contains one or more system control agents (see [System Control Agent](#)). (LEVEL-0)

R_{PDQHN} The primary [system controller](#) controls the Power Management Integrated Circuit (PMIC) associated with a [I/O Controller Chiplet](#). (LEVEL-0)

4.10.1.10 Security

R_{GRYYV} A [I/O Controller Chiplet](#) contains one or more trusted security agents (see [Trusted Security Agent](#)) that collectively implement, at a minimum, the following Hardware Enforced Security (HES) functions:

- Secure storage of a chiplet identifier.
- Enforcement of the hardware security lifecycle of the chiplet, including moderation of invasive subsystems, for example, debug. A trusted security agent might manage the security lifecycle or delegate that management to another trusted security agent.

For more information about HES, see [Trusted Security Agent](#).

(LEVEL-0)

R_{JDSWQ} A [trusted security agent](#) in a [I/O Controller Chiplet](#) is a secondary trusted security agent. (LEVEL-0)

4.10.1.11 Debug

I_{NRYYB} A [I/O Controller Chiplet](#) contains debug resources associated with its components, including any Application PEs, trusted security agents, and system controllers.

4.10.1.11.1 Debug access

R_{NHPSD} An external debugger can access the debug resources of components in a [I/O Controller Chiplet](#) through a Debug Access Port (DAP) in the chiplet. (LEVEL-0)

R_{JHKVK} The debug resources of components in a [I/O Controller Chiplet](#) receive memory-mapped accesses from system debug agents in the same chiplet or other chiplets. (LEVEL-0)

R_{YNBXR} System debug agents in the same chiplet or other chiplets can access any debug resources in the system through a [I/O Controller Chiplet](#) by using memory-mapped accesses. (LEVEL-0)

I_{YKMDL} The [Trusted Security Agent](#) in a [I/O Controller Chiplet](#) might restrict debug access to enforce the hardware security lifecycle state of the chiplet. *Authenticated debug* provides access to chiplet debug features after the trusted security agent verifies identity. For more information, see the Arm CoreSight Architecture [5].

R_{DJFVK} A [I/O Controller Chiplet](#) might restrict debug access according to the hardware security lifecycle state of the chiplet. The chiplet might provide an *Authenticated debug* capability that enables debug access after the [trusted security agent](#) verifies identity. (LEVEL-0)

4.10.1.12 System

I_{KRZNG} A unique identifier is necessary for each [I/O Controller Chiplet](#) because a system might contain more than one instance of the chiplet.

R_{VVBCX} All chiplets in the system can access the identification register in a [I/O Controller Chiplet](#). The value in the register uniquely identifies the chiplet instance within the system. (LEVEL-0)

5

Chiplet interfaces

This chapter defines the interface requirements between different chiplet types.

This chapter defines the following interface types:

- [Hub Chiplet to Hub Chiplet Interface](#)
- [Hub Chiplet to Compute Chiplet Interface](#)
- [Compute 1 Chiplet to Compute 1 Chiplet Interface](#)
- [Compute 2 Chiplet to Compute 2 Chiplet Interface](#)
- [Fully Coherent Expansion Chiplet Interface](#)
- [Fully Coherent Expansion \(Remote Translation\) Chiplet Interface](#)
- [I/O Coherent Expansion \(Translated\) Chiplet Interface](#)
- [I/O Coherent Expansion \(Untranslated\) Chiplet Interface](#)
- [I/O Coherent Expansion \(Remote Translation\) Chiplet Interface](#)
- [I/O Chiplet Interface](#)
- [I/O Controller Chiplet Interface](#)

5.1 Common requirements for chiplet interfaces

U_{WMYQD}

The interface requirements are grouped by pairs of chiplet types. Some requirements are common to multiple groups.

Some interface requirement sets are supersets or subsets of other sets. Therefore, an interface between one combination of chiplet types might also meet the requirements for a different combination.

Associating chiplet types with an interface specification does not impose an additional requirement or restriction on the interface. A chiplet interface might comply with more than one interface specification in this chapter. If the chiplet type requirements are compatible, the same interface might connect to multiple chiplet types.

5.2 Chiplet activity states and interface availability

I_{SMSPG}

During the operational lifecycle of a system, chiplet activity states and interface availability change. This FCSA defines chiplet activity states to support the interface requirements. [Figure 5.1](#) shows an illustrative sequence of these states. Each interface description identifies the states in which that interface is available.

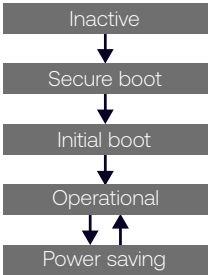


Figure 5.1: Chiplet activity states

Inactive

The chiplet is inactive. All chiplet interfaces are unavailable.

Secure boot

The security lifecycle states of the trusted security agents are synchronized. The chiplet interfaces between trusted security agents are available.

Initial boot

Boot code images are loaded into the chiplets. This process enables the security components, system control components, and Application PEs to boot or advance beyond the initial immutable boot code. The interfaces that transfer boot code images are available. Interfaces that initialize basic system functionality, such as the system counter, are also available.

Operational

The system has reached the point in the initialization sequence at which the high-capacity, relatively high-power chiplet interfaces are available.

Power saving

Some chiplet interfaces are unavailable to save power. Some communications use relatively low-power or dedicated interfaces instead.

[Table 5.1](#) summarizes interface availability by chiplet activity state.

Table 5.1: Summary of chiplet activity states and interface availability

Interface	<i>Inactive</i>	<i>Secure boot</i>	<i>Initial boot</i>	<i>Operational</i>	<i>Power saving</i>
Boot Code Image Load	Unavailable	Available	Available	Available	Unavailable
Coherent Memory Traffic	Unavailable	Unavailable	Unavailable	Available	Unavailable
Untranslated I/O Coherent Memory Traffic	Unavailable	Unavailable	Unavailable	Available	Unavailable
I/O Coherent Memory Traffic	Unavailable	Unavailable	Unavailable	Available	Unavailable
Non-Coherent Memory Traffic	Unavailable	Unavailable	Unavailable	Available	Unavailable
Device Configuration	Unavailable	Unavailable	Unavailable	Available	Unavailable
Address Translation	Unavailable	Unavailable	Unavailable	Available	Unavailable
Interrupts	Unavailable	Unavailable	Unavailable	Available	Unavailable
Trusted Security Agent Communication	Unavailable	Available	Available	Available	Available
System Control	Unavailable	Available	Available	Available	Available
Debug Access	Unavailable	Available	Available	Available	Available

5.3 Hub Chiplet to Hub Chiplet Interface requirements

I_{PKTVB} This section specifies the requirements of the interface between a [Hub Chiplet](#) and another [Hub Chiplet](#). [Figure 5.2](#) shows the interfaces between the two chiplets.

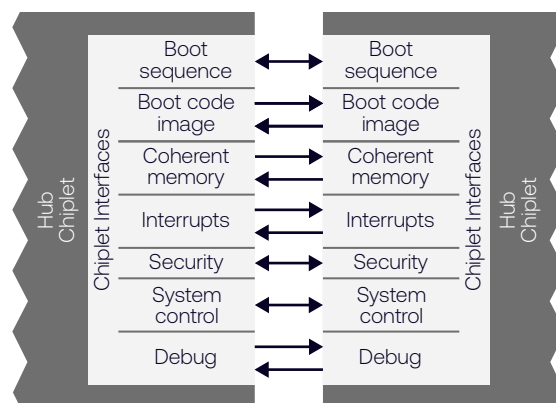


Figure 5.2: Hub Chiplet to Hub Chiplet interfaces

5.3.1 Boot Sequence Control Interface

I_{GFNJB} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{KNVZY} A [Boot Sequence Control Interface](#) directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).

(LEVEL-0)

R_{SYWLF} The [Boot Sequence Control Interface](#) between a [Hub Chiplet](#) and another [Hub Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.

(LEVEL-0)

U_{HQHGJ} For example, a [Boot Sequence Control Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{GFMJD} For implementation details, see [Boot Sequence Control Interface](#).

5.3.2 Boot Code Image Load Interface

I_{GNTLV} A [Boot Code Image Load Interface](#) transports boot code images between two directly connected chiplets. The source chiplet loads the images from non-volatile memory that it accesses directly or through connected chiplets. The interface transports the images to the destination chiplet and to other chiplets connected to the destination chiplet.

R_{ZPXZW} For a [Hub Chiplet](#) and another [Hub Chiplet](#) that are directly connected, the system implementation designates one chiplet as the source of boot code images and the other as the destination. A [Boot Code Image Load Interface](#) transports the images from the source chiplet to memory in the destination chiplet or in other chiplets connected to the destination chiplet.

(LEVEL-1)

R_{KMPRP} A [Boot Code Image Load Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-1)

U_{CGLNT} For example, a [Boot Code Image Load Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational chiplet activity states* but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{SHCBY} For implementation details, see [Boot Code Image Load Interface](#).

5.3.3 Coherent Memory Traffic Interface

I_{XPNJG} A [Coherent Memory Traffic Interface](#) enables coherent agents in a [Hub Chiplet](#) and chiplets connected to it to access system addressable memory and coherent caches in another [Hub Chiplet](#) and chiplets connected to it.

R_{SYDRX} A [Coherent Memory Traffic Interface](#) directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) and transports cache-coherent memory traffic and non-coherent memory traffic in both directions.

(LEVEL-0)

R_{LQZMW} A [Coherent Memory Traffic Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{JDHFK} For example, a [Coherent Memory Traffic Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{VZCKZ} For implementation details, see [Coherent Memory Traffic Interface](#).

5.3.4 Interrupt handling

I_{ZXQMY} This section concerns interrupts that target application PEs running the main operating system.

R_{DEVLC} An [Interrupt Interface](#) directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) and transports interrupts in both directions.

(LEVEL-0)

R_{CQVLD} An [Interrupt Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{JLQMC} For example, an [Interrupt Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{ZFVHB} For implementation details, see [Interrupts](#).

5.3.5 Security

R_{DHBBJ} A [Trusted Security Agent Communication Interface](#) directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) and transports messages between the [trusted security agents](#) in the chiplets.

(LEVEL-0)

R_{TPSJG} A [Trusted Security Agent Communication Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) can support the delegation of security lifecycle management and other secure communication between [trusted security agents](#).

(LEVEL-0)

R_{DKMDM} A [Trusted Security Agent Communication Interface](#) that directly connects a [Hub Chiplet](#) and another [Hub Chiplet](#) uses encryption, integrity protection, and replay protection.

(LEVEL-0)

R _{RVKPZ}	A Trusted Security Agent Communication Interface that directly connects a Hub Chiplet and another Hub Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (<i>LEVEL-0</i>)
U _{FZSXT}	For example, a Trusted Security Agent Communication Interface that directly connects a Hub Chiplet and another Hub Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{DVYVQ}	For implementation details, see Trusted Security Agent Communication Interface .

5.3.6 System control communication

R _{NJPCS}	A System Control Interface directly connects a Hub Chiplet and another Hub Chiplet and transports messages between the system controllers in the chiplets. (<i>LEVEL-0</i>)
R _{PMLBM}	A System Control Interface that directly connects a Hub Chiplet and another Hub Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (<i>LEVEL-0</i>)
U _{FQPCB}	For example, a System Control Interface that directly connects a Hub Chiplet and another Hub Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{VFZZK}	For implementation details, see System Controller Communication .

5.3.7 Debug

5.3.7.1 Debug Access

R _{BMRTC}	A Debug Access Interface directly connects a Hub Chiplet and another Hub Chiplet and enables access between debug components in the two chiplets. (<i>LEVEL-0</i>)
R _{VWWPY}	A Debug Access Interface that directly connects a Hub Chiplet and another Hub Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (<i>LEVEL-0</i>)
U _{PWKLP}	For example, a Debug Access Interface that directly connects a Hub Chiplet and another Hub Chiplet might be available in the <i>Operational</i> chiplet activity state but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{BVCMK}	For implementation details, see Debug Access Interface .

5.4 Hub Chiplet to Compute Chiplet Interface requirements

I_{KPSBM} In this section, the term Compute refers to the chiplet to which a [Hub Chiplet](#) connects. The requirements in this section apply to the interface between a [Hub Chiplet](#) and the compute chiplet. The compute chiplet is a [Compute 1 Chiplet](#) or a [Compute 2 Chiplet](#).

I_{WSTSX} [Figure 5.3](#) shows the interfaces between the two chiplets.

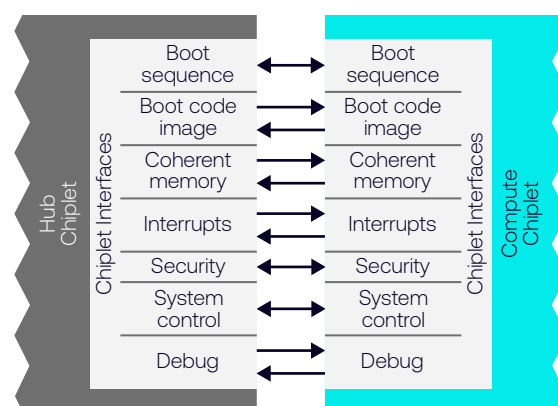


Figure 5.3: Hub Chiplet to Compute Chiplet interfaces

5.4.1 Boot Sequence Control Interface

I_{ZCBKP} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{JQQPR} A [Boot Sequence Control Interface](#) directly connects a [Hub Chiplet](#) and a compute chiplet and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).

(LEVEL-0)

R_{YQMLC} The [Boot Sequence Control Interface](#) between a [Hub Chiplet](#) and a compute chiplet is available when the chiplets share a [chiplet activity state](#) in which the interface is available.

(LEVEL-0)

U_{HPWKX} For example, a [Boot Sequence Control Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{JXYJZ} For implementation details, see [Boot Sequence Control Interface](#).

5.4.2 Boot Code Image Load Interface

I_{ZSFTB} Non-volatile memory is accessible to a [Hub Chiplet](#). A [Boot Code Image Load Interface](#) transports boot code images from the non-volatile memory to each chiplet in the system.

R_{NQBSN} For a [Hub Chiplet](#) and a compute chiplet that are directly connected, the system implementation designates one chiplet as the source of boot code images and the other as the destination. A [Boot Code Image Load Interface](#) transports the images from the source chiplet to memory in the destination chiplet or in other chiplets connected to the destination chiplet.

(LEVEL-1)

R_{YWSTJ} A [Boot Code Image Load Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-1)

U_{CCDDY} For example, a [Boot Code Image Load Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet might be available in the *Secure boot*, *Initial boot*, and *Operational chiplet activity states* but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{WVMBX} For implementation details, see [Boot Code Image Load Interface](#).

5.4.3 Coherent Memory Traffic Interface

I_{GRRLM} A [Coherent Memory Traffic Interface](#) enables coherent agents in a [Hub Chiplet](#) and chiplets connected to it to access system addressable memory and coherent caches in a compute chiplet and chiplets connected to it.

I_{MKFMT} A [Coherent Memory Traffic Interface](#) enables coherent agents in a compute chiplet and chiplets connected to it to access system addressable memory and coherent caches in a [Hub Chiplet](#) and chiplets connected to it.

R_{FVFQQ} A [Coherent Memory Traffic Interface](#) directly connects a [Hub Chiplet](#) and a compute chiplet and transports cache-coherent memory traffic and non-coherent memory traffic in both directions.

(LEVEL-0)

R_{GHLHT} A [Coherent Memory Traffic Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{FWCYL} For example, a [Coherent Memory Traffic Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{GTCPJ} For implementation details, see [Coherent Memory Traffic Interface](#).

5.4.4 Interrupt handling

I_{FGYNK} This section concerns interrupts that target application PEs running the main operating system.

R_{CLWWD} An [Interrupt Interface](#) directly connects a [Hub Chiplet](#) and a compute chiplet and transports interrupts in both directions.

(LEVEL-0)

R_{GMZTH} An [Interrupt Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{YVRLS} For example, an [Interrupt Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{PDGKJ} For implementation details, see [Interrupts](#).

5.4.5 Security

R_{NNQCD} A [Trusted Security Agent Communication Interface](#) directly connects a [Hub Chiplet](#) and a compute chiplet and transports messages between the [trusted security agents](#) in the chiplets.

(LEVEL-0)

R_{FXRMW} A [Trusted Security Agent Communication Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet can support the delegation of security lifecycle management and other secure communication between [trusted security agents](#).

(LEVEL-0)

R_{SZZGL} A [Trusted Security Agent Communication Interface](#) that directly connects a [Hub Chiplet](#) and a compute chiplet uses encryption, integrity protection, and replay protection.

(LEVEL-0)

R _{YGWKV}	A Trusted Security Agent Communication Interface that directly connects a Hub Chiplet and a compute chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{FDWDK}	For example, a Trusted Security Agent Communication Interface that directly connects a Hub Chiplet and a compute chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving chiplet activity states</i> but unavailable in the <i>Inactive</i> chiplet activity state.
I _{GWHRV}	For implementation details, see Trusted Security Agent Communication Interface .

5.4.6 System control communication

R _{BHDVN}	A System Control Interface directly connects a Hub Chiplet and a compute chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{HJZYS}	A System Control Interface that directly connects a Hub Chiplet and a compute chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{QBZLG}	For example, a System Control Interface that directly connects a Hub Chiplet and a compute chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving chiplet activity states</i> but unavailable in the <i>Inactive</i> chiplet activity state.
I _{FVKJS}	For implementation details, see System Controller Communication .

5.4.7 Debug

5.4.7.1 Debug Access

R _{HZCKZ}	A Debug Access Interface directly connects a Hub Chiplet and a compute chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{CZKJG}	A Debug Access Interface that directly connects a Hub Chiplet and a compute chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{KWXVD}	For example, a Debug Access Interface that directly connects a Hub Chiplet and a compute chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{BJKDZ}	For implementation details, see Debug Access Interface .

5.5 Compute 1 Chiplet to Compute 1 Chiplet Interface requirements

I_{FGEFYM}

This section specifies the requirements of the interface between a [Compute 1 Chiplet](#) and another [Compute 1 Chiplet](#).

[Figure 5.4](#) shows the interfaces between the two chiplets.

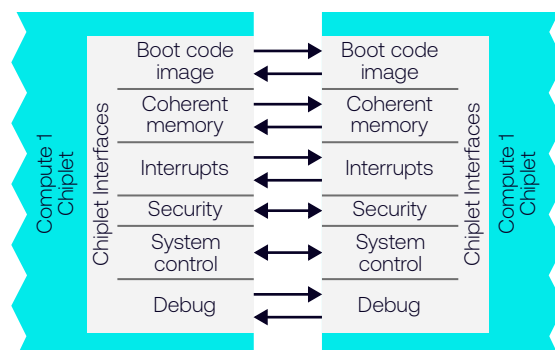


Figure 5.4: Compute 1 Chiplet to Compute 1 Chiplet interfaces

5.5.1 Boot Code Image Load Interface

I_{FZTRR}

A [Boot Code Image Load Interface](#) transports boot code images between two directly connected chiplets. The source chiplet loads the images from non-volatile memory that it accesses directly or through connected chiplets. The interface transports the images to the destination chiplet and to other chiplets connected to the destination chiplet.

R_{SZTCH}

For a [Compute 1 Chiplet](#) and another [Compute 1 Chiplet](#) that are directly connected, the system implementation designates one chiplet as the source of boot code images and the other as the destination. A [Boot Code Image Load Interface](#) transports the images from the source chiplet to memory in the destination chiplet or in other chiplets connected to the destination chiplet.

(LEVEL-1)

R_{RSGWB}

A [Boot Code Image Load Interface](#) that directly connects a [Compute 1 Chiplet](#) and another [Compute 1 Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-1)

U_{LMNCN}

For example, a [Boot Code Image Load Interface](#) that directly connects a [Compute 1 Chiplet](#) and another [Compute 1 Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational chiplet activity states* but unavailable in the *Inactive* and *Power saving chiplet activity states*.

I_{CZBJL}

For implementation details, see [Boot Code Image Load Interface](#).

5.5.2 Coherent Memory Traffic Interface

I_{QHDGN}

A [Coherent Memory Traffic Interface](#) enables coherent agents in a [Compute 1 Chiplet](#) and chiplets connected to it to access system addressable memory and coherent caches in another [Compute 1 Chiplet](#) and chiplets connected to it.

R_{LPBNG}

A [Coherent Memory Traffic Interface](#) directly connects a [Compute 1 Chiplet](#) and another [Compute 1 Chiplet](#) and transports cache-coherent memory traffic and non-coherent memory traffic in both directions.

(LEVEL-0)

R_{KYGBK}

A [Coherent Memory Traffic Interface](#) that directly connects a [Compute 1 Chiplet](#) and another [Compute 1 Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U _{DWPQF}	For example, a Coherent Memory Traffic Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{WRXSD}	For implementation details, see Coherent Memory Traffic Interface .

5.5.3 Interrupt handling

I _{PXSQW}	This section concerns interrupts that target application PEs running the main operating system.
R _{SDKFD}	An Interrupt Interface directly connects a Compute 1 Chiplet and another Compute 1 Chiplet and transports interrupts in both directions. (LEVEL-0)
R _{JSDGJ}	An Interrupt Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{WDTFC}	For example, an Interrupt Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{DTHZB}	For implementation details, see Interrupts .

5.5.4 Security

R _{KFDDX}	A Trusted Security Agent Communication Interface directly connects a Compute 1 Chiplet and another Compute 1 Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{YYBCT}	A Trusted Security Agent Communication Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)
R _{LPQPB}	A Trusted Security Agent Communication Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{SSDSP}	A Trusted Security Agent Communication Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{QTZGF}	For example, a Trusted Security Agent Communication Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{SYDWB}	For implementation details, see Trusted Security Agent Communication Interface .

5.5.5 System control communication

R _{FZBGN}	A System Control Interface directly connects a Compute 1 Chiplet and another Compute 1 Chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{SCNVZ}	A System Control Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)

U _{NMJSN}	For example, a System Control Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving chiplet activity states</i> but unavailable in the <i>Inactive</i> chiplet activity state.
I _{LNBDP}	For implementation details, see System Controller Communication .

5.5.6 Debug

5.5.6.1 Debug Access

R _{MFVBR}	A Debug Access Interface directly connects a Compute 1 Chiplet and another Compute 1 Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{MRDBH}	A Debug Access Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{LNVCZ}	For example, a Debug Access Interface that directly connects a Compute 1 Chiplet and another Compute 1 Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{GBJSZ}	For implementation details, see Debug Access Interface .

5.6 Compute 2 Chiplet to Compute 2 Chiplet Interface requirements

I_{HRJYK} This section specifies the requirements of the interface between a [Compute 2 Chiplet](#) and another [Compute 2 Chiplet](#).

[Figure 5.5](#) shows the interfaces between the two chiplets.

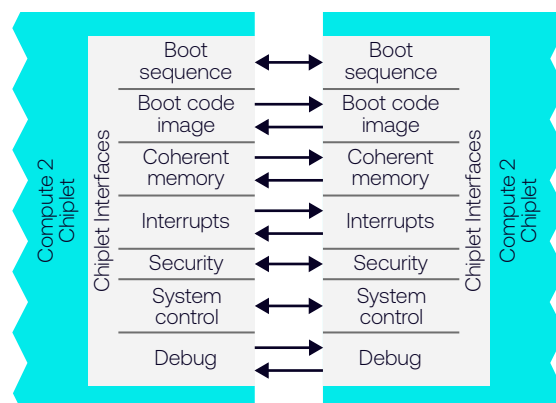


Figure 5.5: Compute 2 Chiplet to Compute 2 Chiplet interfaces

5.6.1 Boot Sequence Control Interface

I_{LQPGR} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{DNJVQ} A [Boot Sequence Control Interface](#) directly connects a [Compute 2 Chiplet](#) and another [Compute 2 Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).
(LEVEL-0)

R_{SVNXJ} The [Boot Sequence Control Interface](#) between a [Compute 2 Chiplet](#) and another [Compute 2 Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.
(LEVEL-0)

U_{YVNCQ} For example, a [Boot Sequence Control Interface](#) that directly connects a [Compute 2 Chiplet](#) and another [Compute 2 Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{RBCDJ} For implementation details, see [Boot Sequence Control Interface](#).

5.6.2 Boot Code Image Load Interface

I_{SBNTD} A [Boot Code Image Load Interface](#) transports boot code images between two directly connected chiplets. The source chiplet loads the images from non-volatile memory that it accesses directly or through connected chiplets. The interface transports the images to the destination chiplet and to other chiplets connected to the destination chiplet.

R_{JRSXS} For a [Compute 2 Chiplet](#) and another [Compute 2 Chiplet](#) that are directly connected, the system implementation designates one chiplet as the source of boot code images and the other as the destination. A [Boot Code Image Load Interface](#) transports the images from the source chiplet to memory in the destination chiplet or in other chiplets connected to the destination chiplet.
(LEVEL-1)

R _{RLQYC}	A Boot Code Image Load Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-1)
U _{VSRWY}	For example, a Boot Code Image Load Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , and <i>Operational chiplet activity states</i> but unavailable in the <i>Inactive</i> and <i>Power saving</i> chiplet activity states.
I _{XFJML}	For implementation details, see Boot Code Image Load Interface .

5.6.3 Coherent Memory Traffic Interface

I _{KBDPD}	A Coherent Memory Traffic Interface enables coherent agents in a Compute 2 Chiplet and chiplets connected to it to access system addressable memory and coherent caches in another Compute 2 Chiplet and chiplets connected to it.
R _{RBNSS}	A Coherent Memory Traffic Interface directly connects a Compute 2 Chiplet and another Compute 2 Chiplet and transports cache-coherent memory traffic and non-coherent memory traffic in both directions. (LEVEL-0)
R _{CZLQD}	A Coherent Memory Traffic Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{KGYZK}	For example, a Coherent Memory Traffic Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{MNYKS}	For implementation details, see Coherent Memory Traffic Interface .

5.6.4 Interrupt handling

I _{TWRML}	This section concerns interrupts that target application PEs running the main operating system.
R _{NLNCD}	An Interrupt Interface directly connects a Compute 2 Chiplet and another Compute 2 Chiplet and transports interrupts in both directions. (LEVEL-0)
R _{JRLRN}	An Interrupt Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{ZLJZD}	For example, an Interrupt Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{ZWVHK}	For implementation details, see Interrupts .

5.6.5 Security

R _{TXHMX}	A Trusted Security Agent Communication Interface directly connects a Compute 2 Chiplet and another Compute 2 Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{LLTTW}	A Trusted Security Agent Communication Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)

R _{GZMKP}	A Trusted Security Agent Communication Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{RTCSK}	A Trusted Security Agent Communication Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{NCQLX}	For example, a Trusted Security Agent Communication Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{PRZWQ}	For implementation details, see Trusted Security Agent Communication Interface .

5.6.6 System control communication

R _{PDJZG}	A System Control Interface directly connects a Compute 2 Chiplet and another Compute 2 Chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{NLZFW}	A System Control Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{GVNWK}	For example, a System Control Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{GFBQC}	For implementation details, see System Controller Communication .

5.6.7 Debug

5.6.7.1 Debug Access

R _{QLHKK}	A Debug Access Interface directly connects a Compute 2 Chiplet and another Compute 2 Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{NCKXH}	A Debug Access Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{NCHFQ}	For example, a Debug Access Interface that directly connects a Compute 2 Chiplet and another Compute 2 Chiplet might be available in the <i>Operational</i> chiplet activity state but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{YGPRL}	For implementation details, see Debug Access Interface .

5.7 Fully Coherent Expansion Chiplet Interface requirements

I_{DJFXG} In this section, the term Host refers to the chiplet to which a [Fully Coherent Expansion Chiplet](#) connects. The requirements in this section apply to the interface between a [Fully Coherent Expansion Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), or another [Fully Coherent Expansion Chiplet](#).

I_{FYFCP} [Figure 5.6](#) shows the interfaces between the two chiplets.

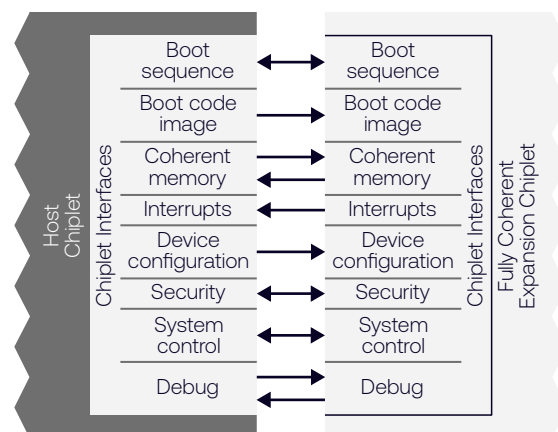


Figure 5.6: Fully Coherent Expansion Chiplet interfaces

5.7.1 Boot Sequence Control Interface

I_{LXXQS} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{LNKBT} A [Boot Sequence Control Interface](#) directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).

(LEVEL-0)

R_{ZTXLR} The [Boot Sequence Control Interface](#) between a host chiplet and a [Fully Coherent Expansion Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.

(LEVEL-0)

U_{LWGVN} For example, a [Boot Sequence Control Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{KLDZN} For implementation details, see [Boot Sequence Control Interface](#).

5.7.2 Boot Code Image Load Interface

I_{YDTNL} Non-volatile memory is accessible to a host chiplet. A [Boot Code Image Load Interface](#) transports boot code images from the non-volatile memory to each chiplet in the system.

R_{PRKPY} A [Boot Code Image Load Interface](#) directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) and can transport boot code images from a host chiplet to memory locations in a [Fully Coherent Expansion Chiplet](#) or in other chiplets connected to that chiplet.

(LEVEL-1)

R_{WPZNH} A [Boot Code Image Load Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-1)

U_{LQCLJ} For example, a [Boot Code Image Load Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational chiplet activity states* but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{RDPDB} For implementation details, see [Boot Code Image Load Interface](#).

5.7.3 Coherent Memory Traffic Interface

I_{GRSRV} A [Coherent Memory Traffic Interface](#) enables coherent agents in a host chiplet and chiplets connected to it to access system addressable memory and coherent caches in a [Fully Coherent Expansion Chiplet](#) and chiplets connected to it.

I_{BTYCX} A [Coherent Memory Traffic Interface](#) enables coherent agents in a [Fully Coherent Expansion Chiplet](#) and chiplets connected to it to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.

R_{VLQPR} A [Coherent Memory Traffic Interface](#) directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) and transports cache-coherent memory traffic and non-coherent memory traffic in both directions.

(LEVEL-0)

R_{RBMCH} A [Coherent Memory Traffic Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{ZYKQY} For example, a [Coherent Memory Traffic Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{BHBCN} For implementation details, see [Coherent Memory Traffic Interface](#).

5.7.4 Interrupt handling

I_{MDKRR} This section concerns interrupts that target application PEs running the main operating system.

R_{WGSPV} An [Interrupt Interface](#) directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) and transports interrupts in both directions.

(LEVEL-0)

R_{PFZGF} An [Interrupt Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{XZQWV} For example, an [Interrupt Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{QGBCN} For implementation details, see [Interrupts](#).

5.7.5 Device Configuration Interface

R_{GJBYM} A [Device Configuration Interface](#) directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) and enables host software to configure one or more devices in a [Fully Coherent Expansion Chiplet](#).

(LEVEL-0)

R_{JZNWW} A [Device Configuration Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U _{CGBPH}	For example, a Device Configuration Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{RVVGW}	For implementation details, see Device Configuration Interface .

5.7.6 Security

R _{JZKYZ}	A Trusted Security Agent Communication Interface directly connects a host chiplet and a Fully Coherent Expansion Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{RVDHX}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)
R _{ZCQOH}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{GHGXC}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{MBYQJ}	For example, a Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{VYLSW}	For implementation details, see Trusted Security Agent Communication Interface .

5.7.7 System control communication

R _{GQTVW}	A System Control Interface directly connects a host chiplet and a Fully Coherent Expansion Chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{LJRBQ}	A System Control Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{QMLZC}	For example, a System Control Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{HQGGR}	For implementation details, see System Controller Communication .

5.7.8 Debug

5.7.8.1 Debug Access

R _{FZVDD}	A Debug Access Interface directly connects a host chiplet and a Fully Coherent Expansion Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{BHWTC}	A Debug Access Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)

U _{GBYYY}	For example, a Debug Access Interface that directly connects a host chiplet and a Fully Coherent Expansion Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{RDPLD}	For implementation details, see Debug Access Interface .

5.8 Fully Coherent Expansion (Remote Translation) Chiplet Interface requirements

I_{HHFHD} In this section, the term Host refers to the chiplet to which a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) connects. The requirements in this section apply to the interface between a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), or a [Fully Coherent Expansion Chiplet](#).

I_{BHKFR} [Figure 5.7](#) shows the interfaces between the two chiplets.

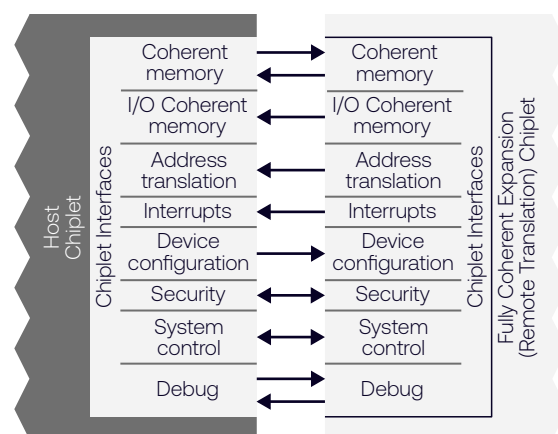


Figure 5.7: Fully Coherent Expansion (Remote Translation) Chiplet interfaces

5.8.1 Coherent Memory Traffic Interface

I_{GPSBW} [Coherent Memory Traffic Interface](#) enables coherent agents in a host chiplet and chiplets connected to it to access system addressable memory and coherent caches in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) and chiplets connected to it.

I_{MNQGD} A [Coherent Memory Traffic Interface](#) enables coherent agents in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) and chiplets connected to it to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.

R_{CMRXD} A [Coherent Memory Traffic Interface](#) directly connects a host chiplet and a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) and transports cache-coherent memory traffic and non-coherent memory traffic in both directions.

(LEVEL-0)

R_{TGCLP} A [Coherent Memory Traffic Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{GXKGF} For example, a [Coherent Memory Traffic Interface](#) that directly connects a host chiplet and a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{MFGBW} For implementation details, see [Coherent Memory Traffic Interface](#).

5.8.2 I/O Coherent Memory Traffic Interface

I _{ZKZJZ}	An I/O Coherent Memory Traffic Interface enables I/O coherent agents in a Fully Coherent Expansion (Remote Translation) Chiplet to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.
R _{QSHDG}	An I/O Coherent Memory Traffic Interface directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet and transports I/O coherent memory traffic from a Fully Coherent Expansion (Remote Translation) Chiplet to a host chiplet.
	(LEVEL-0)
R _{SXWZM}	An I/O Coherent Memory Traffic Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
	(LEVEL-0)
U _{QDKVT}	For example, an I/O Coherent Memory Traffic Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{GTCRB}	For implementation details, see I/O Coherent Memory Traffic Interface .

5.8.3 Address Translation Interface

R _{VBVKL}	An Address Translation Interface directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet and enables one or more devices in a Fully Coherent Expansion (Remote Translation) Chiplet to access address translations through an MMU in a host chiplet.
	(LEVEL-0)
R _{LHSRL}	An Address Translation Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
	(LEVEL-0)
U _{NWGWD}	For example, an Address Translation Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{YBZRH}	For implementation details, see Address Translation Interface .

5.8.4 Interrupt handling

I _{ZHFTY}	This section concerns interrupts that target application PEs running the main operating system.
I _{BJLXZ}	A Fully Coherent Expansion (Remote Translation) Chiplet might contain components that generate interrupts.
R _{LWJBR}	An Interrupt Interface transports interrupts as MSIs from a host chiplet to a Fully Coherent Expansion (Remote Translation) Chiplet .
	(LEVEL-0)
R _{HYTDS}	An Interrupt Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
	(LEVEL-0)
U _{BDRFG}	For example, an Interrupt Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{CYSDK}	For implementation details, see Interrupts .

5.8.5 Device Configuration Interface

R _{GLVZD}	A Device Configuration Interface directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet and enables host software to configure one or more devices in a Fully Coherent Expansion (Remote Translation) Chiplet . (LEVEL-0)
R _{LXTVM}	A Device Configuration Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{BYXHS}	For example, a Device Configuration Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{YYTZY}	For implementation details, see Device Configuration Interface .

5.8.6 Security

R _{GRXQH}	A Trusted Security Agent Communication Interface directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{CSFZB}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)
R _{GGMJ}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{QBXFV}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{WGCL}	For example, a Trusted Security Agent Communication Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{TYFDG}	For implementation details, see Trusted Security Agent Communication Interface .

5.8.7 System control communication

R _{GQNNR}	A System Control Interface directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{XXTOB}	A System Control Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{MWCKT}	For example, a System Control Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{DKNSC}	For implementation details, see System Controller Communication .

5.8.8 Debug

5.8.8.1 Debug Access

R_{PCWPV}	A Debug Access Interface directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R_{DXKQF}	A Debug Access Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U_{MHLTV}	For example, a Debug Access Interface that directly connects a host chiplet and a Fully Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i>, <i>Secure boot</i>, <i>Initial boot</i>, and <i>Power saving</i> chiplet activity states.
I_{WKZWQ}	For implementation details, see Debug Access Interface.

5.9 I/O Coherent Expansion (Translated) Chiplet Interface requirements

I_{QJSSD} In this section, the term Host refers to the chiplet to which an [I/O Coherent Expansion \(Translated\) Chiplet](#) connects. The requirements in this section apply to the interface between an [I/O Coherent Expansion \(Translated\) Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), a [Fully Coherent Expansion Chiplet](#), or another [I/O Coherent Expansion \(Translated\) Chiplet](#).

I_{JZCPB} [Figure 5.8](#) shows the interfaces between the two chiplets.

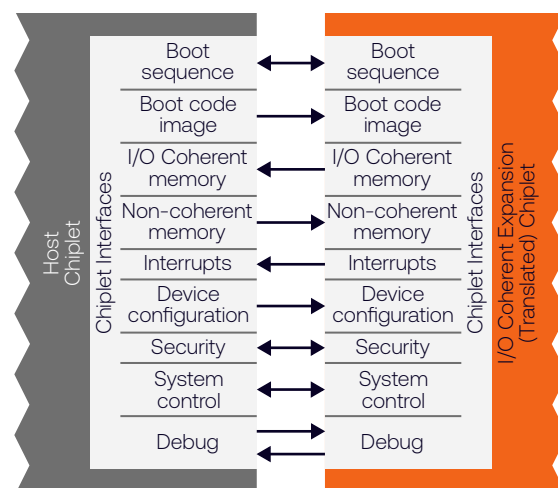


Figure 5.8: I/O Coherent Expansion (Translated) Chiplet interfaces

5.9.1 Boot Sequence Control Interface

I_{QVKPH} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{MWPML} A [Boot Sequence Control Interface](#) directly connects a host chiplet and an [I/O Coherent Expansion \(Translated\) Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).
(LEVEL-0)

R_{VTDRN} The [Boot Sequence Control Interface](#) between a host chiplet and an [I/O Coherent Expansion \(Translated\) Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.
(LEVEL-0)

U_{XZFBH} For example, a [Boot Sequence Control Interface](#) that directly connects a host chiplet and an [I/O Coherent Expansion \(Translated\) Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{MVYQS} For implementation details, see [Boot Sequence Control Interface](#).

5.9.2 Boot Code Image Load Interface

I_{SLXGZ} Non-volatile memory is accessible to a host chiplet. A [Boot Code Image Load Interface](#) transports boot code images from the non-volatile memory to each chiplet in the system.

R_{CNCGH} A [Boot Code Image Load Interface](#) directly connects a host chiplet and an [I/O Coherent Expansion \(Translated\) Chiplet](#) and can transport boot code images from a host chiplet to memory locations in an [I/O Coherent Expansion \(Translated\) Chiplet](#) or in other chiplets connected to that chiplet.
(LEVEL-1)

R _{JLGGN}	A Boot Code Image Load Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (<i>LEVEL-1</i>)
U _{VRZYT}	For example, a Boot Code Image Load Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , and <i>Operational chiplet activity states</i> but unavailable in the <i>Inactive</i> and <i>Power saving</i> chiplet activity states.
I _{FZGMS}	For implementation details, see Boot Code Image Load Interface .

5.9.3 I/O Coherent Memory Traffic Interface

I _{LCPND}	An I/O Coherent Memory Traffic Interface enables I/O coherent agents in an I/O Coherent Expansion (Translated) Chiplet to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.
R _{HCKYF}	An I/O Coherent Memory Traffic Interface directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet and transports I/O coherent memory traffic from an I/O Coherent Expansion (Translated) Chiplet to a host chiplet. (<i>LEVEL-0</i>)
R _{HXNJH}	An I/O Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (<i>LEVEL-0</i>)
U _{XVPJB}	For example, an I/O Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{GWWSK}	For implementation details, see I/O Coherent Memory Traffic Interface .

5.9.4 Non-Coherent Memory Traffic Interface

I _{BFSDB}	A Non-Coherent Memory Traffic Interface enables agents in a host chiplet to access system addressable memory in an I/O Coherent Expansion (Translated) Chiplet and chiplets connected to it.
R _{RGQKJ}	A Non-Coherent Memory Traffic Interface directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet and transports non-coherent memory traffic from a host chiplet to an I/O Coherent Expansion (Translated) Chiplet . (<i>LEVEL-0</i>)
R _{YHSGY}	A Non-Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (<i>LEVEL-0</i>)
U _{STBSZ}	For example, a Non-Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{KRTRC}	For implementation details, see Non-Coherent Memory Traffic Interface .

5.9.5 Interrupt handling

I _{JCQGS}	This section concerns interrupts that target application PEs running the main operating system.
I _{YZDGH}	An I/O Coherent Expansion (Translated) Chiplet might contain components that generate interrupts.
R _{JSFTT}	An Interrupt Interface between a host chiplet and an I/O Coherent Expansion (Translated) Chiplet transports interrupts as MSIs. (<i>LEVEL-0</i>)

R _{BMJZP}	An Interrupt Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{DCVYN}	For example, an Interrupt Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{WVWQ}	For implementation details, see Interrupts .

5.9.6 Device Configuration Interface

R _{RNYKY}	A Device Configuration Interface directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet and enables host software to configure one or more devices in an I/O Coherent Expansion (Translated) Chiplet . (LEVEL-0)
I _{GKSGZ}	For implementation details, see Device Configuration Interface .

5.9.7 Security

R _{XRDCX}	A Trusted Security Agent Communication Interface directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{ZGDGY}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)
R _{JJKSQ}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{BHRHJ}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{JXQZC}	For example, a Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{GXDBS}	For implementation details, see Trusted Security Agent Communication Interface .

5.9.8 System control communication

R _{NTZHB}	A System Control Interface directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{SQJLT}	A System Control Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{BGPYB}	For example, a System Control Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{ZRKPF}	For implementation details, see System Controller Communication .

5.9.9 Debug

5.9.9.1 Debug Access

R _{CYHBV}	A Debug Access Interface directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{TKMBQ}	A Debug Access Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{JZPWW}	For example, a Debug Access Interface that directly connects a host chiplet and an I/O Coherent Expansion (Translated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i>, <i>Secure boot</i>, <i>Initial boot</i>, and <i>Power saving</i> chiplet activity states.
I _{WZPWM}	For implementation details, see Debug Access Interface.

5.10 I/O Coherent Expansion (Untranslated) Chiplet Interface requirements

I_{GZLLQ} In this section, the term Host refers to the chiplet to which an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) connects. The requirements in this section apply to the interface between an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), a [Fully Coherent Expansion Chiplet](#), an [I/O Coherent Expansion \(Translated\) Chiplet](#), or another [I/O Coherent Expansion \(Untranslated\) Chiplet](#).

I_{QGNZM} [Figure 5.9](#) shows the interfaces between the two chiplets.

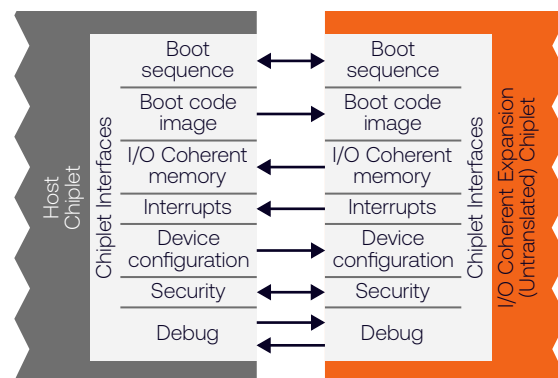


Figure 5.9: I/O Coherent Expansion (Untranslated) Chiplet interfaces

5.10.1 Boot Sequence Control Interface

I_{XXWKF} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{NTXZD} A [Boot Sequence Control Interface](#) directly connects a host chiplet and an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).
(*LEVEL-0*)

R_{XGDKV} The [Boot Sequence Control Interface](#) between a host chiplet and an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.
(*LEVEL-0*)

U_{LWTTY} For example, a [Boot Sequence Control Interface](#) that directly connects a host chiplet and an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{KTTWQ} For implementation details, see [Boot Sequence Control Interface](#).

5.10.2 Boot Code Image Load Interface

I_{MHJVS} Non-volatile memory is accessible to a host chiplet. A [Boot Code Image Load Interface](#) transports boot code images from the non-volatile memory to each chiplet in the system.

R_{WBSBR} A [Boot Code Image Load Interface](#) directly connects a host chiplet and an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) and can transport boot code images from a host chiplet to memory locations in an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) or in other chiplets connected to that chiplet.
(*LEVEL-1*)

R _{WWJWH}	A Boot Code Image Load Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-1)
U _{NMRMC}	For example, a Boot Code Image Load Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , and <i>Operational chiplet activity states</i> but unavailable in the <i>Inactive</i> and <i>Power saving</i> chiplet activity states.
I _{DXGYQ}	For implementation details, see Boot Code Image Load Interface .

5.10.3 Untranslated I/O Coherent Memory Traffic Interface

I _{CPDTB}	A Untranslated I/O Coherent Memory Traffic Interface enables I/O coherent agents in an I/O Coherent Expansion (Untranslated) Chiplet to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.
R _{CBWZD}	A Untranslated I/O Coherent Memory Traffic Interface directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet and transports I/O coherent memory traffic from an I/O Coherent Expansion (Untranslated) Chiplet to a host chiplet. (LEVEL-0)
R _{SZXHJ}	The Untranslated I/O Coherent Memory Traffic Interface from an I/O Coherent Expansion (Untranslated) Chiplet to a host chiplet is untranslated. (LEVEL-0)
R _{QBQMD}	A Untranslated I/O Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{WCRTK}	For example, a Untranslated I/O Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{TPZJH}	For implementation details, see I/O Coherent Memory Traffic Interface .

5.10.4 Interrupt handling

I _{SDGFD}	This section concerns interrupts that target application PEs running the main operating system.
I _{PDLGP}	An I/O Coherent Expansion (Untranslated) Chiplet might contain components that generate interrupts.
R _{DTDLR}	An Interrupt Interface between a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet transports interrupts as MSIs from an I/O Coherent Expansion (Untranslated) Chiplet through an MMU in the host chiplet. (LEVEL-0)
R _{CDBPS}	An Interrupt Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{MVZLP}	For example, an Interrupt Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{BWHKL}	For implementation details, see Interrupts .

5.10.5 Device Configuration Interface

R _{RYVZX}	A Device Configuration Interface directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet and enables host software to configure one or more devices in an I/O Coherent Expansion (Untranslated) Chiplet . (LEVEL-0)
R _{QTJWG}	A Device Configuration Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{FKBXQ}	For example, a Device Configuration Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{SHJVQ}	For implementation details, see Device Configuration Interface .

5.10.6 Security

R _{BSQPC}	A Trusted Security Agent Communication Interface directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{MCCDT}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)
R _{PVDQS}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{PDJPV}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{VYRSP}	For example, a Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{ZDMFP}	For implementation details, see Trusted Security Agent Communication Interface .

5.10.7 Debug

5.10.7.1 Debug Access

R _{JJPDD}	A Debug Access Interface directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{ZGBHJ}	A Debug Access Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{NYFME}	For example, a Debug Access Interface that directly connects a host chiplet and an I/O Coherent Expansion (Untranslated) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{VYWHN}	For implementation details, see Debug Access Interface .

5.11 I/O Coherent Expansion (Remote Translation) Chiplet Interface requirements

I_{JLVTG} In this section, the term Host refers to the chiplet to which an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) connects. The requirements in this section apply to the interface between an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), a [Fully Coherent Expansion Chiplet](#), or an [I/O Coherent Expansion \(Translated\) Chiplet](#).

I_{WLPLN} [Figure 5.10](#) shows the interfaces between the two chiplets.

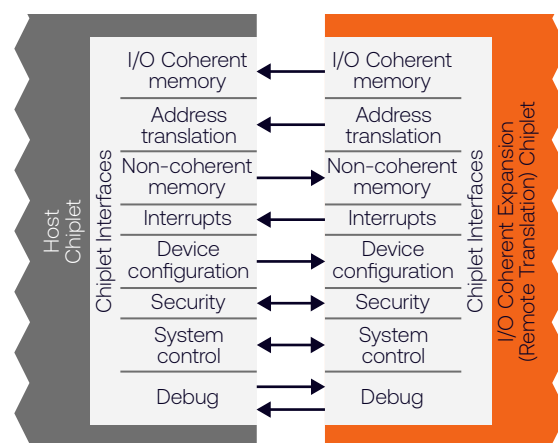


Figure 5.10: I/O Coherent Expansion (Remote Translation) Chiplet interfaces

5.11.1 I/O Coherent Memory Traffic Interface

I_{HHGDJ} An [I/O Coherent Memory Traffic Interface](#) enables I/O coherent agents in an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.

R_{HHPXJ} An [I/O Coherent Memory Traffic Interface](#) directly connects a host chiplet and an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) and transports I/O coherent memory traffic from an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) to a host chiplet.

(LEVEL-0)

R_{YQSDV} An [I/O Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{BHWWB} For example, an [I/O Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{XQBRP} For implementation details, see [I/O Coherent Memory Traffic Interface](#).

5.11.2 Address Translation Interface

R_{MTLNR} An [Address Translation Interface](#) directly connects a host chiplet and an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) and enables one or more devices in an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) to access address translations through an MMU in a host chiplet.

(LEVEL-0)

R _{SVYMX}	An Address Translation Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{MMWGY}	For example, an Address Translation Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{RRBJG}	For implementation details, see Address Translation Interface .

5.11.3 Non-Coherent Memory Traffic Interface

I _{GVFGW}	A Non-Coherent Memory Traffic Interface enables agents in a host chiplet to access system addressable memory in an I/O Coherent Expansion (Remote Translation) Chiplet and chiplets connected to it.
R _{NMQCH}	A Non-Coherent Memory Traffic Interface directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet and transports non-coherent memory traffic from a host chiplet to an I/O Coherent Expansion (Remote Translation) Chiplet . (LEVEL-0)
R _{ZLKGN}	A Non-Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{GTXWW}	For example, a Non-Coherent Memory Traffic Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{PRZTK}	For implementation details, see Non-Coherent Memory Traffic Interface .

5.11.4 Interrupt handling

I _{HHCLY}	This section concerns interrupts that target application PEs running the main operating system.
I _{HMLRC}	An I/O Coherent Expansion (Remote Translation) Chiplet might contain components that generate interrupts.
R _{QRJPN}	An Interrupt Interface transports interrupts as MSIs from a host chiplet to an I/O Coherent Expansion (Remote Translation) Chiplet . (LEVEL-0)
R _{RVRZS}	An Interrupt Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{YXMXV}	For example, an Interrupt Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{KTFNX}	For implementation details, see Interrupts .

5.11.5 Device Configuration Interface

R _{LFDBC}	A Device Configuration Interface directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet and enables host software to configure one or more devices in an I/O Coherent Expansion (Remote Translation) Chiplet . (LEVEL-0)
--------------------	--

R _{ZTFJZ}	A Device Configuration Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{VCDYR}	For example, a Device Configuration Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i> , <i>Secure boot</i> , <i>Initial boot</i> , and <i>Power saving</i> chiplet activity states.
I _{WSFJW}	For implementation details, see Device Configuration Interface .

5.11.6 Security

R _{BNLPG}	A Trusted Security Agent Communication Interface directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet and transports messages between the trusted security agents in the chiplets. (LEVEL-0)
R _{LGJSW}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet can support the delegation of security lifecycle management and other secure communication between trusted security agents . (LEVEL-0)
R _{VQBCT}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet uses encryption, integrity protection, and replay protection. (LEVEL-0)
R _{FQQRQ}	A Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{LGDTC}	For example, a Trusted Security Agent Communication Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{PHDNM}	For implementation details, see Trusted Security Agent Communication Interface .

5.11.7 System control communication

R _{QXZDW}	A System Control Interface directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet and transports messages between the system controllers in the chiplets. (LEVEL-0)
R _{VPNZQ}	A System Control Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{XWMWN}	For example, a System Control Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Secure boot</i> , <i>Initial boot</i> , <i>Operational</i> , and <i>Power saving</i> chiplet activity states but unavailable in the <i>Inactive</i> chiplet activity state.
I _{YKBDB}	For implementation details, see System Controller Communication .

5.11.8 Debug

5.11.8.1 Debug Access

R _{SDPSD}	A Debug Access Interface directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet and enables access between debug components in the two chiplets. (LEVEL-0)
R _{ZKCGC}	A Debug Access Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet is available when both chiplets are in a chiplet activity state in which the interface functions are operative. (LEVEL-0)
U _{ZVXKZ}	For example, a Debug Access Interface that directly connects a host chiplet and an I/O Coherent Expansion (Remote Translation) Chiplet might be available in the <i>Operational chiplet activity state</i> but unavailable in the <i>Inactive</i>, <i>Secure boot</i>, <i>Initial boot</i>, and <i>Power saving</i> chiplet activity states.
I _{BSCMY}	For implementation details, see Debug Access Interface.

5.12 I/O Chiplet Interface requirements

I_{XSVHF} In this section, the term Host refers to the chiplet to which an [I/O Chiplet](#) connects. The requirements in this section apply to the interface between an [I/O Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), a [Fully Coherent Expansion Chiplet](#), an [I/O Coherent Expansion \(Translated\) Chiplet](#), an [I/O Coherent Expansion \(Untranslated\) Chiplet](#), or another [I/O Chiplet](#).

I_{WDCXN} [Figure 5.11](#) shows the interfaces between the two chiplets.

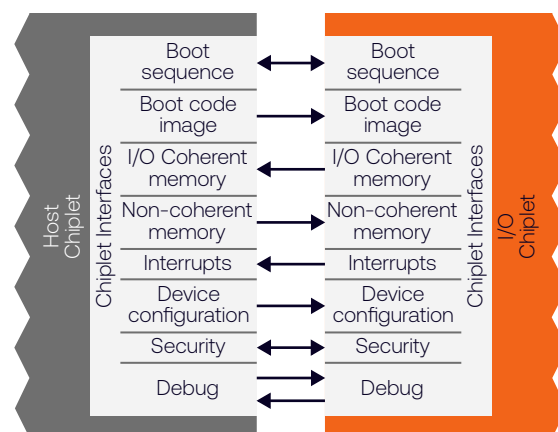


Figure 5.11: I/O Chiplet interfaces

5.12.1 Boot Sequence Control Interface

I_{KTPZV} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{CTTSY} A [Boot Sequence Control Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).

(LEVEL-0)

R_{NHLLR} The [Boot Sequence Control Interface](#) between a host chiplet and an [I/O Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.

(LEVEL-0)

U_{BDLZX} For example, a [Boot Sequence Control Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{LHKDS} For implementation details, see [Boot Sequence Control Interface](#).

5.12.2 Boot Code Image Load Interface

I_{HFCCM} Non-volatile memory is accessible to a host chiplet. A [Boot Code Image Load Interface](#) transports boot code images from the non-volatile memory to each chiplet in the system.

R_{WGMT} A [Boot Code Image Load Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and can transport boot code images from a host chiplet to memory locations in an [I/O Chiplet](#) or in other chiplets connected to that chiplet.

(LEVEL-1)

R_{GBYYK} A [Boot Code Image Load Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-1)

I_{QPMHQ} For example, a [Boot Code Image Load Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational chiplet activity states* but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{FGHMK} For implementation details, see [Boot Code Image Load Interface](#).

5.12.3 Untranslated I/O Coherent Memory Traffic Interface

I_{QBXCG} A [Untranslated I/O Coherent Memory Traffic Interface](#) enables I/O coherent agents in an [I/O Chiplet](#) to access system addressable memory and coherent caches in a host chiplet and chiplets connected to it.

R_{LHRDK} A [Untranslated I/O Coherent Memory Traffic Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and transports I/O coherent memory traffic from an [I/O Chiplet](#) to a host chiplet.

(LEVEL-0)

R_{WLLKW} The [Untranslated I/O Coherent Memory Traffic Interface](#) from an [I/O Chiplet](#) to a host chiplet is untranslated.

(LEVEL-0)

R_{XKHXM} A [Untranslated I/O Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{FYVBW} For example, a [Untranslated I/O Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{CFKLR} For implementation details, see [I/O Coherent Memory Traffic Interface](#).

5.12.4 Non-Coherent Memory Traffic Interface

I_{PCVDS} A [Non-Coherent Memory Traffic Interface](#) enables agents in a host chiplet to access system addressable memory in an [I/O Chiplet](#) and chiplets connected to it.

R_{DVBZV} A [Non-Coherent Memory Traffic Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and transports non-coherent memory traffic from a host chiplet to an [I/O Chiplet](#).

(LEVEL-0)

R_{RSJDZ} A [Non-Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{NRNFZ} For example, a [Non-Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{STLWC} For implementation details, see [Non-Coherent Memory Traffic Interface](#).

5.12.5 Interrupt handling

I_{PBRMC} This section concerns interrupts that target application PEs running the main operating system.

I_{XYFWM} An [I/O Chiplet](#) might contain components that generate interrupts.

R_{DFQYV} An [Interrupt Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{FRMKJ} For example, an [Interrupt Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{PWGNR} For implementation details, see [Interrupts](#).

5.12.6 Device Configuration Interface

R_{YLRCM} A [Device Configuration Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and enables host software to configure one or more devices in an [I/O Chiplet](#).
(LEVEL-0)

R_{HJDFI} A [Device Configuration Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
(LEVEL-0)

U_{NVBYX} For example, a [Device Configuration Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{DKNMD} For implementation details, see [Device Configuration Interface](#).

5.12.7 Security

R_{RVTJM} A [Trusted Security Agent Communication Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and transports messages between the [trusted security agents](#) in the chiplets.
(LEVEL-0)

R_{KBGTC} A [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) can support the delegation of security lifecycle management and other secure communication between [trusted security agents](#).
(LEVEL-0)

R_{HXWGL} A [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) uses encryption, integrity protection, and replay protection.
(LEVEL-0)

R_{HHJRZ} A [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
(LEVEL-0)

U_{RVPVJ} For example, a [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Secure boot*, *Initial boot*, *Operational*, and *Power saving chiplet activity states* but unavailable in the *Inactive* chiplet activity state.

I_{DPMFL} For implementation details, see [Trusted Security Agent Communication Interface](#).

5.12.8 Debug

5.12.8.1 Debug Access

R_{TDQTB} A [Debug Access Interface](#) directly connects a host chiplet and an [I/O Chiplet](#) and enables access between debug components in the two chiplets.
(LEVEL-0)

R_{HLPGX} A [Debug Access Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
(LEVEL-0)

U_{JQVLK} For example, a [Debug Access Interface](#) that directly connects a host chiplet and an [I/O Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{GCSVJ} For implementation details, see [Debug Access Interface](#).

5.13 I/O Controller Chiplet Interface requirements

I_{TCHHQ} In this section, the term Host refers to the chiplet to which an [I/O Controller Chiplet](#) connects. The requirements in this section apply to the interface between an [I/O Controller Chiplet](#) and the host chiplet. The host chiplet is a [Hub Chiplet](#), a [Compute 2 Chiplet](#), a [Fully Coherent Expansion Chiplet](#), an [I/O Coherent Expansion \(Translated\) Chiplet](#), an [I/O Coherent Expansion \(Untranslated\) Chiplet](#), an [I/O Chiplet](#), or another [I/O Controller Chiplet](#).

I_{XTPKB} [Figure 5.12](#) shows the interfaces between the two chiplets.

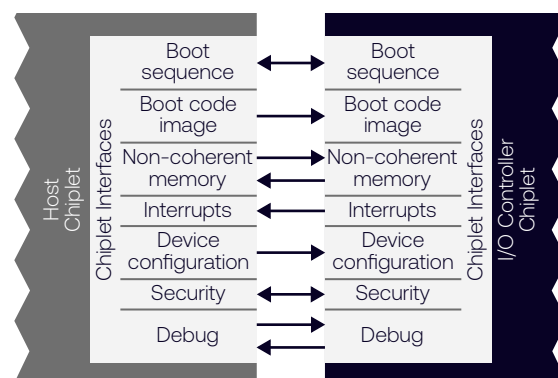


Figure 5.12: I/O Controller Chiplet interfaces

5.13.1 Boot Sequence Control Interface

I_{VBNCF} A [Boot Sequence Control Interface](#) between two directly connected chiplets transports boot-sequence control signals between the [trusted security agent](#) and [system controller](#) components of each chiplet. See [Chiplet System Component Boot Sequence](#) for more information.

R_{ZBNTF} A [Boot Sequence Control Interface](#) directly connects a host chiplet and an [I/O Controller Chiplet](#) and transports boot-sequence control signals between their [trusted security agents](#) and [system controllers](#).

(LEVEL-0)

R_{DJCGG} The [Boot Sequence Control Interface](#) between a host chiplet and an [I/O Controller Chiplet](#) is available when the chiplets share a [chiplet activity state](#) in which the interface is available.

(LEVEL-0)

U_{MQPKM} For example, a [Boot Sequence Control Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational* [chiplet activity states](#) but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{HPTTS} For implementation details, see [Boot Sequence Control Interface](#).

5.13.2 Boot Code Image Load Interface

I_{JBQQV} Non-volatile memory is accessible to a host chiplet. A [Boot Code Image Load Interface](#) transports boot code images from the non-volatile memory to each chiplet in the system.

R_{RDVNB} A [Boot Code Image Load Interface](#) directly connects a host chiplet and an [I/O Controller Chiplet](#) and can transport boot code images from a host chiplet to memory locations in an [I/O Controller Chiplet](#) or in other chiplets connected to that chiplet.

(LEVEL-1)

R_{JHJVP} A [Boot Code Image Load Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-1)

U_{FWTNS} For example, a [Boot Code Image Load Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Secure boot*, *Initial boot*, and *Operational chiplet activity states* but unavailable in the *Inactive* and *Power saving* chiplet activity states.

I_{FNCRV} For implementation details, see [Boot Code Image Load Interface](#).

5.13.3 Non-Coherent Memory Traffic Interface

I_{YRMRK} A [Non-Coherent Memory Traffic Interface](#) enables agents in a host chiplet to access system addressable memory in an [I/O Controller Chiplet](#) and chiplets connected to it.

I_{HNKGV} A [Non-Coherent Memory Traffic Interface](#) enables agents in an [I/O Controller Chiplet](#) to access system addressable memory in a host chiplet and chiplets connected to it.

R_{DDDSW} A [Non-Coherent Memory Traffic Interface](#) directly connects a host chiplet and an [I/O Controller Chiplet](#) and transports non-coherent memory traffic in both directions.

(LEVEL-0)

R_{YZHWP} A [Non-Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{QFCFD} For example, a [Non-Coherent Memory Traffic Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{NXZRC} For implementation details, see [Non-Coherent Memory Traffic Interface](#).

5.13.4 Interrupt handling

I_{CZKKR} This section concerns interrupts that target application PEs running the main operating system.

I_{VNLNT} An [I/O Controller Chiplet](#) might contain components that generate interrupts.

R_{YJGHH} An [Interrupt Interface](#) between a host chiplet and an [I/O Controller Chiplet](#) transports interrupts as MSIs from an [I/O Controller Chiplet](#) through an MMU in the host chiplet.

(LEVEL-0)

R_{LSJMB} An [Interrupt Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{VWNRD} For example, an [Interrupt Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{KFHHG} For implementation details, see [Interrupts](#).

5.13.5 Device Configuration Interface

R_{PCWSQ} A [Device Configuration Interface](#) directly connects a host chiplet and an [I/O Controller Chiplet](#) and enables host software to configure one or more devices in an [I/O Controller Chiplet](#).

(LEVEL-0)

R_{ZQVFY} A [Device Configuration Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.

(LEVEL-0)

U_{KQWJV} For example, a [Device Configuration Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{RFLNG} For implementation details, see [Device Configuration Interface](#).

5.13.6 Security

R_{YRHHX} A [Trusted Security Agent Communication Interface](#) directly connects a host chiplet and an [I/O Controller Chiplet](#) and transports messages between the [trusted security agents](#) in the chiplets.
(LEVEL-0)

R_{HLXNC} A [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) can support the delegation of security lifecycle management and other secure communication between [trusted security agents](#).
(LEVEL-0)

R_{LKCWK} A [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) uses encryption, integrity protection, and replay protection.
(LEVEL-0)

R_{XZPFB} A [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
(LEVEL-0)

U_{MYXRS} For example, a [Trusted Security Agent Communication Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Secure boot*, *Initial boot*, *Operational*, and *Power saving chiplet activity states* but unavailable in the *Inactive* chiplet activity state.

I_{RPXCH} For implementation details, see [Trusted Security Agent Communication Interface](#).

5.13.7 Debug

5.13.7.1 Debug Access

R_{RNDMS} A [Debug Access Interface](#) directly connects a host chiplet and an [I/O Controller Chiplet](#) and enables access between debug components in the two chiplets.
(LEVEL-0)

R_{NRHNZ} A [Debug Access Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) is available when both chiplets are in a chiplet activity state in which the interface functions are operative.
(LEVEL-0)

U_{CXCvZ} For example, a [Debug Access Interface](#) that directly connects a host chiplet and an [I/O Controller Chiplet](#) might be available in the *Operational chiplet activity state* but unavailable in the *Inactive*, *Secure boot*, *Initial boot*, and *Power saving* chiplet activity states.

I_{LRVTY} For implementation details, see [Debug Access Interface](#).

6

Common chiplet components

Some chiplet components are common to the specifications of several FCSA chiplet types. This chapter describes these components and the parts of the system [boot sequence](#) that involve them.

The components are:

- [System Control Framework](#)
- [Trusted Security Agent](#)

6.1 Chiplet system component boot

6.1.1 Chiplet system component boot stages

I_{ZDPY}

The [Trusted Security Agent](#) and [System Controller](#) components run firmware that provides their functionality to the system.

These components typically boot in stages. The following figure shows the boot stages that this section defines. It uses the abbreviations read-only memory (ROM), one-time programmable (OTP) memory, and nonvolatile memory (NVM).

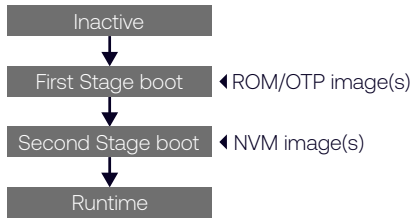


Figure 6.1: Chiplet system component boot stages

D_{WRSGM}

The *Inactive* system component boot stage is the component state before initialization.

D_{YWTMR}

The *First Stage* boot stage is the period during which the component executes its first-stage bootloader. In typical implementations, the component executes this bootloader from read-only memory (ROM) or one-time programmable (OTP) code. This stage is independent of similar components in other chiplets.

At or before this stage completes, communication is established between [Trusted Security Agent](#) components and between [System Controller](#) components, as required, to control the boot sequence. This communication between chiplets uses the [Boot Sequence Control Interface](#).

At or before this stage completes, a component might also establish communication with similar components in the same or other chiplets. This communication uses the [Trusted Security Agent Communication Interface](#) or the [System Control Interface](#).

A component might use the [Boot Code Image Load Interface](#) to transfer a *Second Stage* boot code image to a component in another chiplet. Alternatively, the target component might load the image itself.

An alternative implementation reduces this stage to a minimal form in which no ROM or OTP code is run. The [Boot Sequence Control Interface](#) is established without software.

D_{NLTLL}

The *Second Stage* boot stage is the period during which the component executes its second-stage bootloader. The component itself, or another component in the same or a different chiplet, loads this mutable code from nonvolatile memory.

During this stage, the component might reconfigure its environment and communicate with other components in the same or other chiplets. This communication uses the [Trusted Security Agent Communication Interface](#) or the [System Control Interface](#).

D_{SNSFZ}

The *Runtime* boot stage begins after the component completes its boot process.

6.1.2 Chiplet system component boot sequence signaling

I_{BKKSF}

The [Boot Sequence Control Interface](#) is a set of signals that a *Manager* system component and a *Subordinate* system component use to coordinate the boot sequence.

The [Boot Sequence Control Interface](#) supports multiple models for loading subordinate *Second Stage* boot code images from nonvolatile memory (NVM). The models use the same cross-chiplet wiring.

The supported models are:

- **Distributed:** The manager system component fetches its own *Second Stage* boot code image and the image for the subordinate system component from a single NVM. The manager loads the subordinate image into the subordinate system component through the *Boot Code Image Load Interface*.
 - The [Boot Sequence Control Interface](#) is used to ensure that the subordinate chiplet is ready to receive a boot code image and to control when the image is executed.
- **Shared:** The manager system component and the subordinate system component each fetch their own *Second Stage* boot code image from a single shared NVM.
 - The [Boot Sequence Control Interface](#) is used to manage access to the shared NVM and to control when boot code images are executed.
- **Independent:** The manager system component and the subordinate system component each load their own *Second Stage* boot code image in parallel from separate NVMs.
 - The [Boot Sequence Control Interface](#) is used as a synchronization point to indicate when the chiplets are ready to communicate.

D_{NRPDJ}

The [Boot Sequence Control Interface](#) signals are:

- **Ready to proceed:** indicates that either:
 - The subordinate system component is ready to load its *Second Stage* boot code image.
 - The subordinate system component has successfully loaded the image and is ready to execute it.
- **Proceed:** indicates that either:
 - The subordinate system component can proceed to load its *Second Stage* boot code image.
 - The subordinate system component can execute its *Second Stage* boot code image.
- **Error:** indicates that the subordinate system component has encountered an error when loading or verifying its *Second Stage* boot code image.

X_{LMBBY}

A subordinate system component might encounter an error when loading its *Second Stage* boot code image. The *Error* signal reports the error to the manager component that controls the boot sequence. An error might result from a failure to load the image or verify its integrity or expected contents.

I_{ZTLMW}

If required, the manager system component uses the [Boot Sequence Control Interface](#) to control image loading and execution in separate steps.

[Figure 6.2](#) shows an example of the *distributed* model in which the manager system component loads the *Second Stage* boot code image into the subordinate system component.

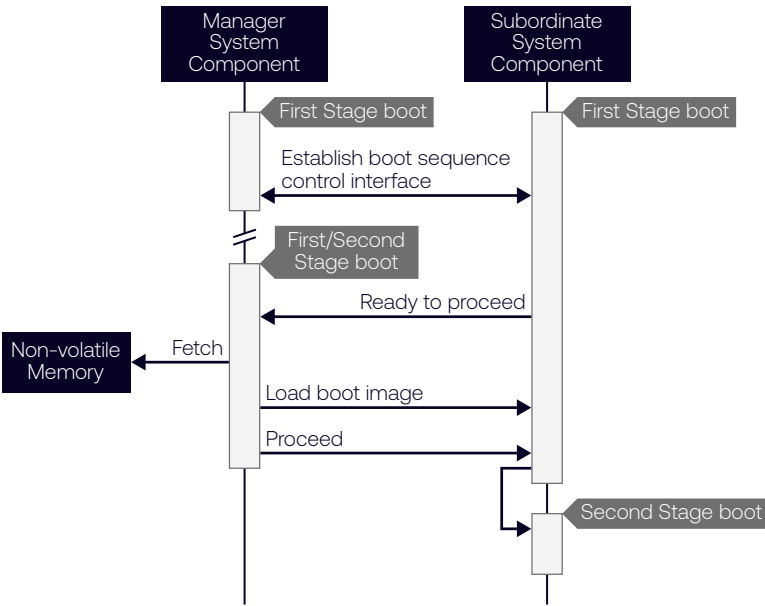


Figure 6.2: Subordinate system component boot sequence: distributed model

I_{RYRZM}

Figure 6.3 shows an example of the *shared* model in which the subordinate system component loads its *Second Stage* boot code image from an NVM that it shares with the manager system component.

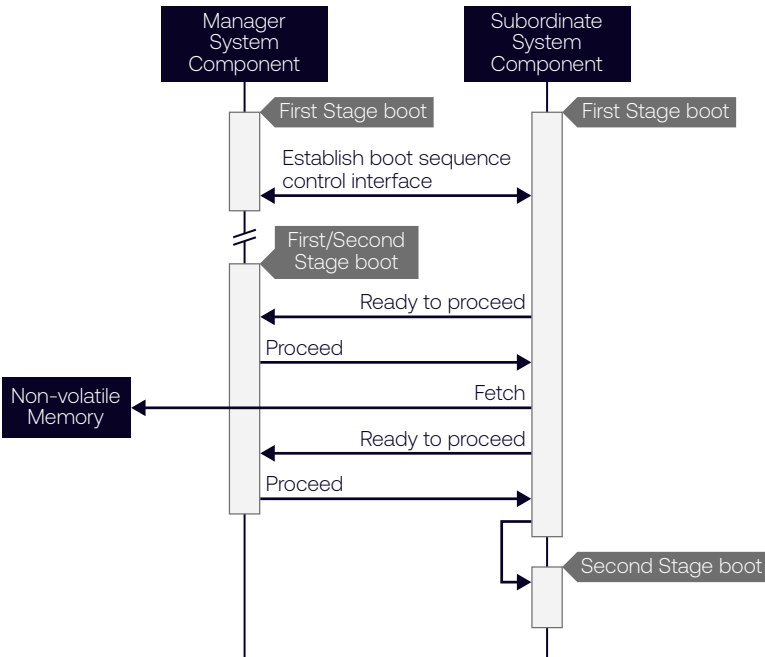


Figure 6.3: Subordinate system component boot sequence: shared model

I_{HKRXG}

Figure 6.4 shows an example of the *independent* model in which the manager and subordinate system components load their *Second Stage* boot code images from different NVMs. The manager system component can use the [Boot Sequence Control Interface](#) to control when the subordinate system component executes its image. Alternatively, the manager can use the interface as another synchronization point.

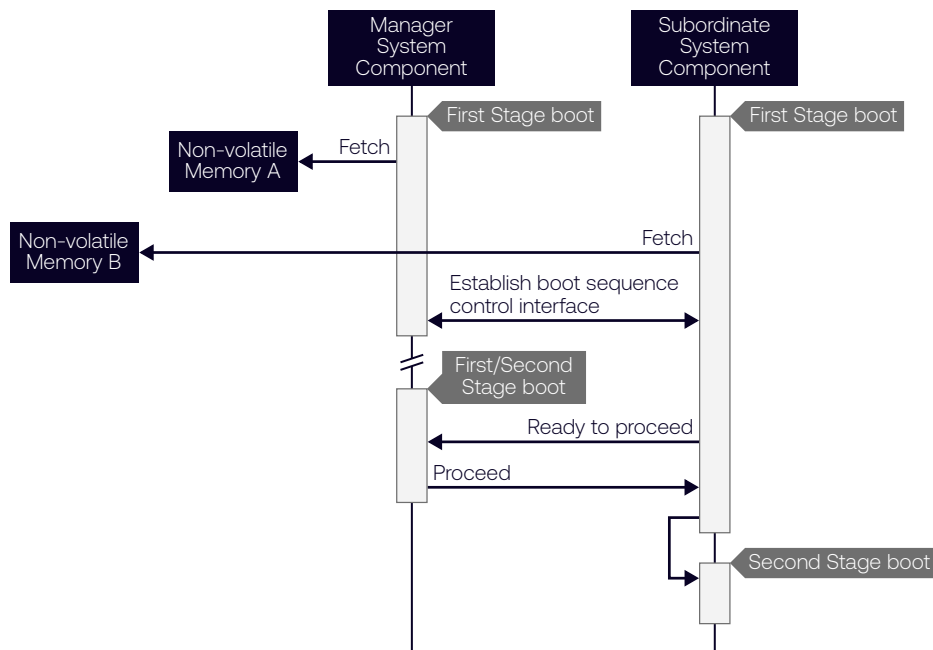


Figure 6.4: Subordinate system component boot sequence: independent model

I_{XHDXK}

Figure 6.5 shows an example in which the manager loads the *Second Stage* boot code image into the subordinate, but the load fails.

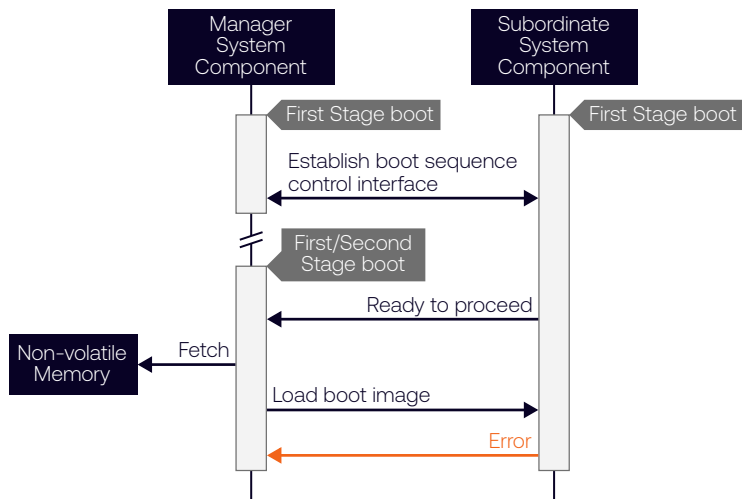


Figure 6.5: Subordinate system component boot sequence with error

I_{ZXVCS}

The control that the manager system component has over the *Second Stage* boot of the subordinate system component applies only to that stage. This control does not apply to subsequent stages in a multistage boot flow.

6.1.3 Chiplet system component boot sequence

I_{WYHNB}

A chiplet contains a [Trusted Security Agent](#) and might also contain a [System Controller](#).
In a chiplet system:

- One trusted security agent is designated as the primary trusted security agent. All other trusted security agents are secondary trusted security agents. The following connections apply:
 - A [Boot Sequence Control Interface](#) connects the primary trusted security agent to each secondary trusted security agent.
 - A *communication channel* connects the primary trusted security agent to each secondary trusted security agent.
- One system controller is designated as the primary system controller. All other system controllers are secondary system controllers. The following connections apply:
 - A [Boot Sequence Control Interface](#) connects the primary trusted security agent to the primary system controller.
 - A [Boot Sequence Control Interface](#) connects each secondary system controller to either the primary trusted security agent or the primary system controller.
 - A *communication channel* connects the primary system controller to each secondary system controller.

U_{XZXKG}

The specifications for the [Trusted Security Agent](#) and [System Controller](#) components define the order of their initial boot stages and the transfer of boot code images.

[Figure 6.6](#), [Figure 6.7](#), and [Figure 6.8](#) show three example initial boot sequences for trusted security agents and system controllers. They depict initialization control and boot code image transfers:

- [Figure 6.6](#) shows the primary trusted security agent loading boot code images from NVM, then transferring them to the secondary trusted security agents and system controllers.
- [Figure 6.7](#) shows:
 - The primary trusted security agent loading the boot code images for the trusted security agents and primary system controller from NVM and controlling their order
 - The primary system controller loading the boot code images for the secondary system controllers from NVM and controlling their order
- [Figure 6.8](#) shows each component loading its own boot code image from NVM, with order controlled by the primary trusted security agent.

The examples do not represent all system designs. The component rules specify the required boot-order points.

Each example identifies the chiplet interfaces used for its operations.

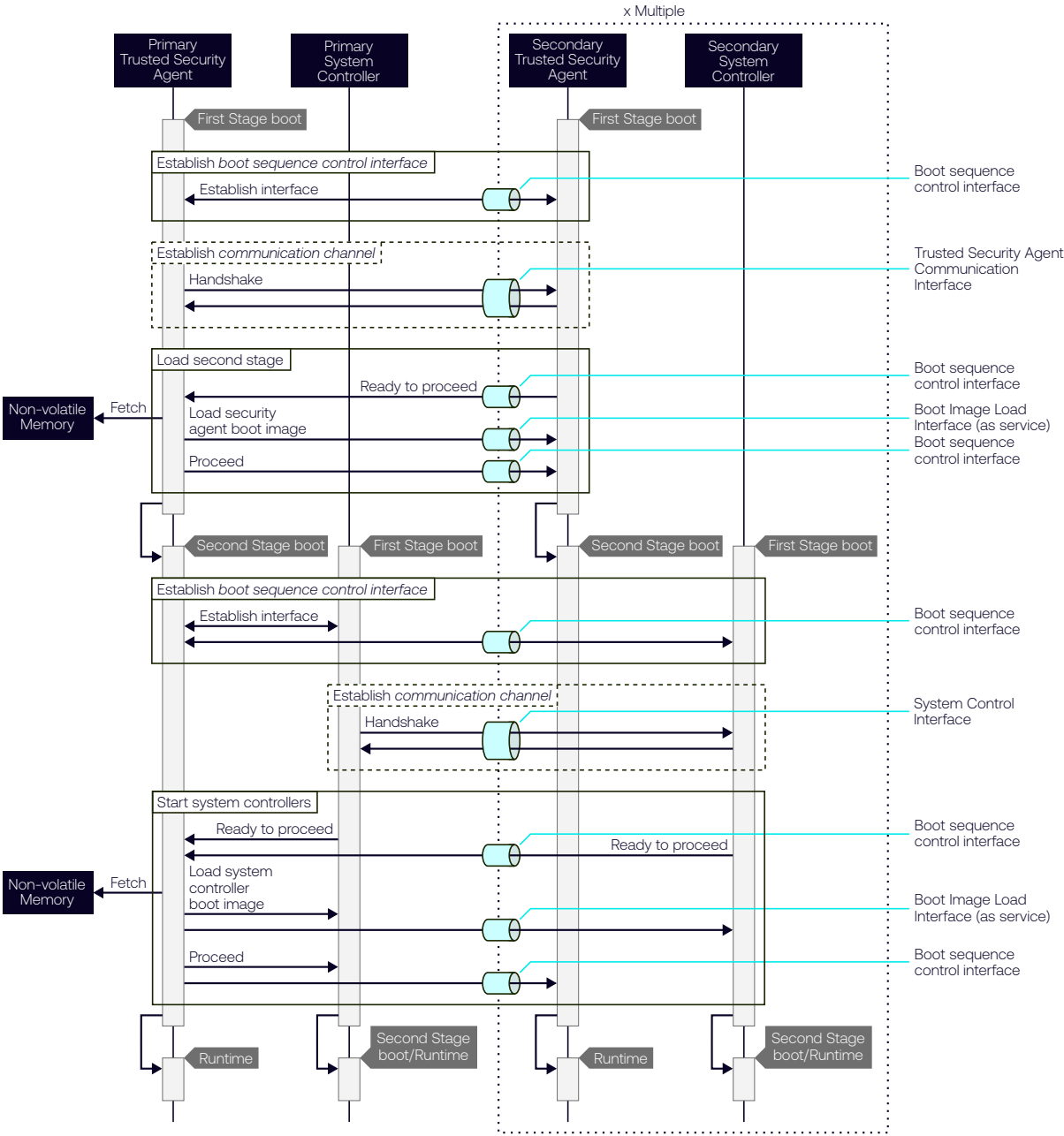


Figure 6.6: Example chiplet system component boot sequence 1

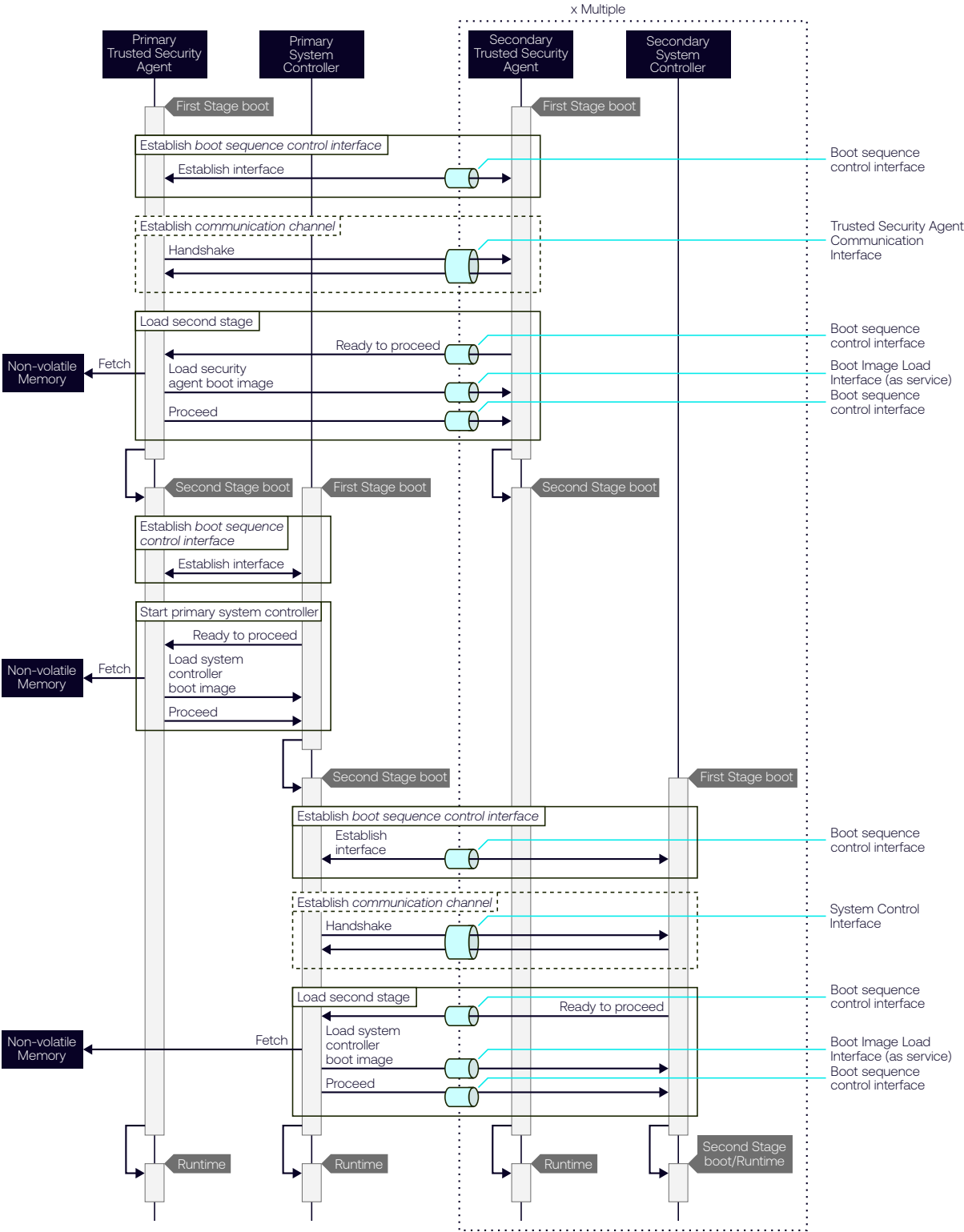


Figure 6.7: Example chiplet system component boot sequence 2

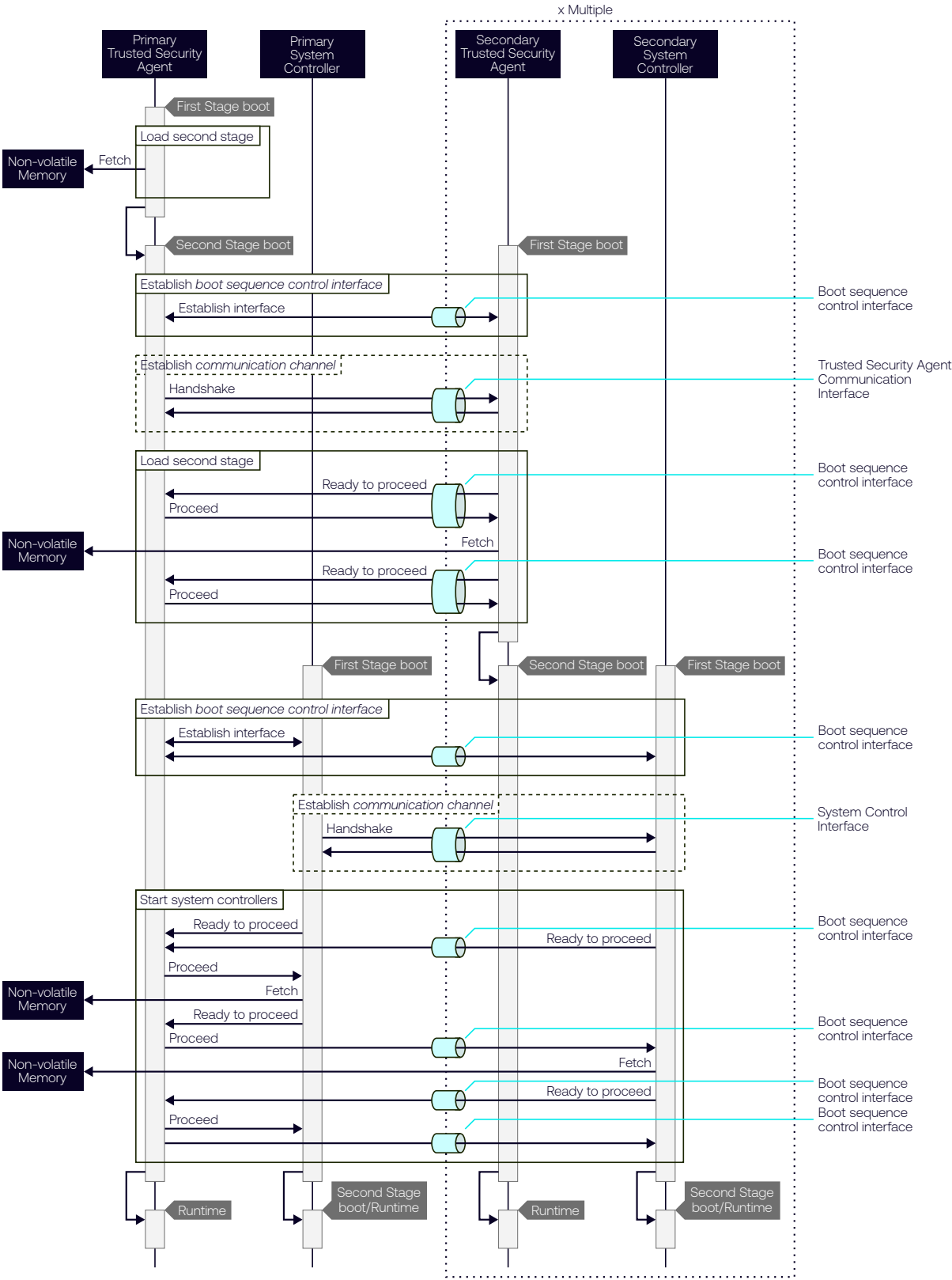


Figure 6.8: Example chiplet system component boot sequence 3

6.2 System Control Framework

The following figure shows an example System Control Framework.

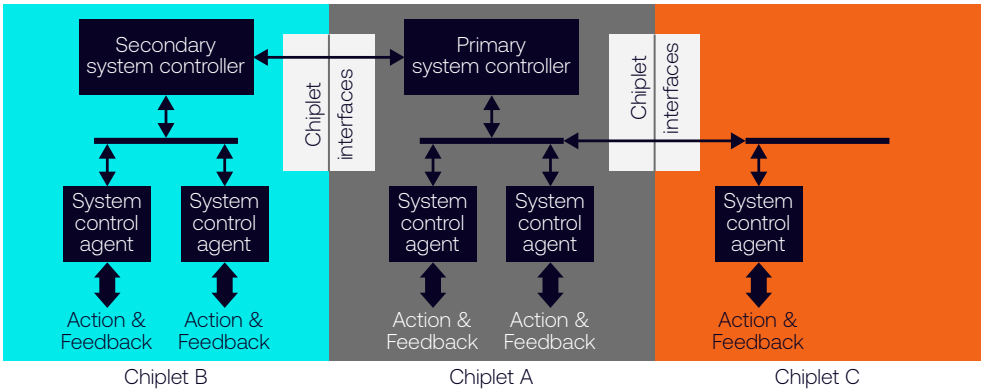


Figure 6.9: System Control Framework example

The System Control Framework performs low-level management tasks that are specific to a system-on-chip (SoC), including power and thermal management.

The figure shows system control functionality distributed across multiple components and chiplets. There are two types of components:

- [System Controller](#)
- [System Control Agent](#)

A chiplet typically contains one *system controller* component. A chiplet can contain multiple system controllers, and some chiplet types do not require one. One system controller is the primary system controller. The other controllers are secondary system controllers. The primary system controller coordinates their actions.

A system controller sets system control policy within its scope of influence. One or more *system control agents* implement this policy directly or locally.

6.2.1 System Controller

A System Controller is a component of the System Control Framework that makes system control policy decisions within a scope of influence in the system.

A system controller is a complex component that includes one or more processing elements (PEs) and supporting components. The system controller executes firmware and might include components for I/O and security.

There is only one *primary* system controller in the system. (LEVEL-0)

The primary system controller coordinates the actions of the secondary system controllers. (LEVEL-0)

If applicable, a *secondary system controller* is responsible for:

- Initialization of chiplet functions for which the system controller is responsible
- Chiplet resets
- Chiplet high-level clock control
- Chiplet performance management
- Chiplet power mode transitions
- Chiplet power budget management
- Chiplet temperature management

- Chiplet telemetry and sensor management
- Chiplet watchdog events

D_{CDNCB} A *primary system controller* has the responsibilities of a secondary system controller. If applicable, it is also responsible for:

- Initialization of system functions for which the system controller is responsible
- System power mode coordination
- System power budget management
- System temperature management
- System sensor and telemetry coordination
- System watchdog events

I_{SQYMV} Several of these functions require communication between system controllers. For example, system controllers communicate to synchronize initialization operations, receive budgets, negotiate power modes, and perform other functions.

R_{CSHRH} There is a *communication channel* between the primary system controller and each secondary system controller. If this channel connects two chiplets, it uses a [System Control Interface](#).

(LEVEL-0)

6.2.1.1 Boot sequencing of system controllers

R_{ZGHZS} There is *boot sequence signaling* between the primary trusted security agent and the primary system controller. If these signals pass between two chiplets, a [Boot Sequence Control Interface](#) is used. The primary trusted security agent is the manager of the interface, and the primary system controller is the subordinate.

(LEVEL-0)

R_{FCBMQ} There is *boot sequence signaling* between either the primary trusted security agent or the primary system controller and each secondary system controller.

If these signals pass between two chiplets:

- A [Boot Sequence Control Interface](#) is used.
- The designer of the chiplet that contains the primary trusted security agent and the primary system controller specifies which system component is the manager of the interface. The secondary system controller is the subordinate.

(LEVEL-0)

U_{LHJJX} Either the primary trusted security agent or the primary system controller manages the boot sequence of a secondary system controller. The designer of the chiplet that contains the primary components selects the manager. If the manager and the secondary system controller are in different chiplets, a single interface connects them. The secondary system controller accepts boot sequence control through the *boot sequence signaling* when required.

R_{SYGXX} The primary trusted security agent controls when the primary system controller proceeds with the *Second Stage* of boot.

(LEVEL-FULL)

R_{WFBVB} The *boot sequence signaling* between the primary trusted security agent and the primary system controller is established during the *First Stage* boot stage of the primary system controller.

(LEVEL-FULL)

R_{CQFTX} The *boot sequence signaling* between the manager and a secondary system controller is established during the *First Stage* boot stage of the secondary system controller.

(LEVEL-FULL)

R_{PWYBJ} The *communication channel* between the primary system controller and a secondary system controller might be established during the *First Stage* boot stage of the secondary system controller.

(LEVEL-FULL)

R _{XKQHN}	During its <i>First Stage</i> boot stage, the primary system controller asserts <i>Ready to proceed</i> using the <i>boot sequence signaling</i> . This signal indicates that the controller is ready to load the mutable boot code image that it executes in the <i>Second Stage</i> . (LEVEL-FULL)
R _{SLGFF}	When the primary system controller signals that it is ready for its <i>Second Stage</i> boot code image, the primary trusted security agent either: • Transfers the <i>Second Stage</i> boot code image to the primary system controller using the Boot Code Image Load Interface. • Indicates that the primary system controller can load the <i>Second Stage</i> boot code image by asserting the <i>Proceed</i> signal using the <i>boot sequence signaling</i>. (LEVEL-FULL)
R _{TDNWN}	After loading its <i>Second Stage</i> boot code image, the primary system controller asserts <i>Ready to proceed</i> using the <i>boot sequence signaling</i> to inform the primary trusted security agent. (LEVEL-FULL)
R _{BTYLD}	If the primary system controller encounters an error while loading its <i>Second Stage</i> boot code image, it asserts <i>Error</i> using the <i>boot sequence signaling</i> to inform the primary trusted security agent. (LEVEL-FULL)
R _{FLQLR}	After the primary system controller successfully loads its <i>Second Stage</i> boot code image, the primary trusted security agent asserts <i>Proceed</i> using the <i>boot sequence signaling</i> to enable the controller to execute the <i>Second Stage</i> boot. (LEVEL-FULL)
R _{HRPJW}	During its <i>First Stage</i> boot stage, a secondary system controller asserts <i>Ready to proceed</i> using the <i>boot sequence signaling</i> . This signal indicates that the controller is ready to load the mutable boot code image that it executes in the <i>Second Stage</i> . (LEVEL-FULL)
R _{KGYFQ}	When a secondary system controller signals that it is ready for its <i>Second Stage</i> boot code image, its manager either: • Transfers the <i>Second Stage</i> boot code image to the secondary system controller using the Boot Code Image Load Interface. • Indicates that the secondary system controller can load the <i>Second Stage</i> boot code image by asserting the <i>Proceed</i> signal using the <i>boot sequence signaling</i>. (LEVEL-FULL)
R _{FSDRF}	After loading its <i>Second Stage</i> boot code image, a secondary system controller asserts <i>Ready to proceed</i> using the <i>boot sequence signaling</i> to inform its manager. (LEVEL-FULL)
R _{KZCTP}	If a secondary system controller encounters an error while loading its <i>Second Stage</i> boot code image, it asserts <i>Error</i> using the <i>boot sequence signaling</i> to inform its manager. (LEVEL-FULL)
R _{KLDXX}	After a secondary system controller successfully loads its <i>Second Stage</i> boot code image, its manager asserts <i>Proceed</i> using the <i>boot sequence signaling</i> to enable the controller to execute the <i>Second Stage</i> boot. (LEVEL-FULL)
X _{PLCDX}	The primary trusted security agent controls the primary system controller. Either the primary trusted security agent or the primary system controller controls each secondary system controller. This arrangement provides these advantages: • Centralized arbitration of access to shared resources, including nonvolatile storage, is simpler to implement and defines clear expectations between chiplet types.

- Centralized determination of a potentially variable boot order for the trusted security agents and system controllers is simpler to implement. It supports the expected security platform boot state and defines clear expectations between chiplet types.

6.2.1.2 Handling boot code images

I _{YXCVN}	The primary system controller might require firmware loaded from a boot code image that the primary trusted security agent provides.
I _{HFXZQ}	A secondary system controller might require firmware loaded from a boot code image that the primary trusted security agent or primary system controller provides.
R _{WZZBY}	The primary trusted security agent is capable of transferring a boot code image to a system controller in another chiplet. The boot code image is first relayed through a secondary trusted security agent in the chiplet that contains the target system controller. The transfer between chiplets uses the Boot Code Image Load Interface , which the trusted security agents provide as a service. (LEVEL-FULL)
R _{TVTBS}	The primary system controller is capable of transferring a boot code image to a secondary system controller in another chiplet using the Boot Code Image Load Interface . (LEVEL-FULL)

6.2.1.3 System controller interrupts

I _{RPFDL}	A system controller receives interrupts that inform it about system events, messages from other controller entities or PEs, and actions that other system management components require. System controller interrupts do not follow the same path as interrupts targeted at Application PEs. An interrupt source sends them directly to the system controller. If the target system controller is on another chiplet, the interrupt is routed to the system controller on the current chiplet. That system controller forwards the interrupt as one or more messages on the System Control Interface to the target system controller.
R _{NQZFH}	A system controller receives interrupts directly from the source of the interrupt. (LEVEL-0)
R _{FLDNS}	System controller interrupts targeted at a system controller on another chiplet are sent to the system controller on the current chiplet. The receiving system controller then forwards them to the target system controller as one or more messages on the System Control Interface . (LEVEL-0)

6.2.2 System Control Agent

D _{XDMS}	A <i>system control agent</i> is a component of the System Control Framework that helps a system controller implement system control policy decisions. A system control agent is a fixed-function component that acts on hardware-level system controls in response to policy settings and direct requests from a system controller. For example, it can safely transition power modes or change clock-gating arrangements. It might include automatic functionality that enables real-time responses to feedback.
R _{GGJPQ}	The configuration of a system control agent is accessible to software through the system address map at the highest implemented security access level that the hardware supports. (LEVEL-FULL)

6.3 Trusted Security Agent

I_{GNQVC}

The requirements of each system design define its security guarantees. A *security platform* supports these guarantees. This collective term identifies all hardware and firmware components that deliver the security guarantees.

Certain security platform components are enforced in hardware. Together, these components provide the system *Hardware Enforced Security* (HES) functionality. HES functionality includes:

- Security platform boot state

The security platform boot state represents the security configurations of the security platform components. It also represents measurements and identification of all security platform firmware.

- Security platform attestation

Security platform attestation is a feature of complex devices with multiple security agents. Each security agent might implement its own secure boot process but does not itself provide a security boot state for the entire platform. Security platform attestation measures the critical boot stages of all platform security components. These measurements provide an overall platform security attestation.

- Hardware security lifecycle state

The security platform might be fully trusted or have a lower degree of trustworthiness. The *security lifecycle* reflects the transitions between these states, from manufacture through use to decommissioning.

The hardware security lifecycle state is enforced by:

- Maintaining hardware-provisioned parameters. These parameters are secrets and public identifiers that might help determine the trustworthiness of the platform and protect security platform assets.
- Restricting invasive subsystems, for example by ensuring that all debug interfaces are closed.
- Ensuring that only authorized security platform firmware is loaded. All parts of the security platform then require authorization from a hardware-enforced source to boot in a non-secure state.

- Cryptographic services necessary for security platform attestation, including the storage of security platform component identifiers, the derivation of keys, and the signing of measurements
- Shielded locations, including tamper-resistant storage
- Encryption of external memory

For more information about the security lifecycle state, refer to [Security of Chiplet Systems](#).

D_{PQHNP}

One or more *Trusted Security Agents* host HES functionality. These agents are isolated hardware subsystems dedicated to security. Their capabilities can range from providing secure storage for a unique identifier to hosting multiple security platform functions. An agent that hosts multiple functions has a private execution environment, trusted firmware, and a boot process.

D_{DXLCY}

A trusted security agent might manage its own security lifecycle and independently represent a Root of Trust (RoT). Because a trusted security agent is attestable, it might delegate its security lifecycle management to another trusted security agent.

I_{VDGJB}

A trusted security agent hosts system HES functionality. In a system design that uses chiplets, HES functionality is distributed across multiple chiplets. Each chiplet that requires or provides any system HES functionality contains one or more trusted security agents.

R_{FFKWC}

A single trusted security agent in the system is designated as the primary trusted security agent and acts as the Root of Trust (RoT). All other trusted security agents are designated as secondary trusted security agents.

(LEVEL-0)

R_{HRPTP}

The primary trusted security agent accepts responsibility for the security lifecycle management of all other trusted security agents in the system.

(LEVEL-FULL)

R _{NPXGM}	A secondary trusted security agent delegates its security lifecycle management to the primary trusted security agent. (LEVEL-FULL)
R _{MWLGW}	There is a <i>communication channel</i> between the primary trusted security agent and each secondary trusted security agent. If this channel connects two chiplets, it uses a Trusted Security Agent Communication Interface . (LEVEL-0)

6.3.1 Boot sequencing of trusted security agents

R _{ZYJRM}	There is <i>boot sequence signaling</i> between the primary trusted security agent and each secondary trusted security agent. If these signals pass between two chiplets, a Boot Sequence Control Interface is used. The primary trusted security agent is the manager of the interface, and the secondary trusted security agent is the subordinate. (LEVEL-0)
R _{KYGJF}	The <i>boot sequence signaling</i> between the primary trusted security agent and a secondary trusted security agent is established during the <i>First Stage</i> boot stage of the secondary trusted security agent. (LEVEL-FULL)
R _{QCDJQ}	The <i>communication channel</i> between the primary trusted security agent and a secondary trusted security agent might be established during the <i>First Stage</i> boot stage of the secondary trusted security agent. (LEVEL-FULL)
R _{SJFCT}	During its <i>First Stage</i> boot stage, a secondary trusted security agent asserts <i>Ready to proceed</i> using the <i>boot sequence signaling</i> . This signal indicates that the agent is ready to load the mutable boot code image that it executes in the <i>Second Stage</i> . (LEVEL-FULL)
R _{XFSTQ}	When a secondary trusted security agent signals that it is ready for its <i>Second Stage</i> boot code image, the primary trusted security agent either: • Transfers the <i>Second Stage</i> boot code image to the secondary trusted security agent using the Boot Code Image Load Interface. • Indicates that the secondary trusted security agent can load the <i>Second Stage</i> boot code image by asserting the <i>Proceed</i> signal using the <i>boot sequence signaling</i>. (LEVEL-FULL)
R _{LFMVS}	After loading its <i>Second Stage</i> boot code image, a secondary trusted security agent asserts <i>Ready to proceed</i> using the <i>boot sequence signaling</i> to inform the primary trusted security agent. (LEVEL-FULL)
R _{PQYSK}	If a secondary trusted security agent encounters an error while loading its <i>Second Stage</i> boot code image, it asserts <i>Error</i> using the <i>boot sequence signaling</i> to inform the primary trusted security agent. (LEVEL-FULL)
R _{DQZZW}	After a secondary trusted security agent successfully loads its <i>Second Stage</i> boot code image, the primary trusted security agent asserts <i>Proceed</i> using the <i>boot sequence signaling</i> to enable the agent to execute the <i>Second Stage</i> boot. (LEVEL-FULL)

6.3.2 Handling boot code images

I _{TPXGS}	The trusted security agent might provide a mechanism for securely transmitting and receiving boot code images between chiplets. A trusted security agent might also provide cryptographic services for processing system component boot code images, including decryption, measurement, and verification.
I _{MTWGV}	A secondary trusted security agent might require firmware loaded from a boot code image that the primary trusted security agent provides.

R_{LDZYP}	The primary trusted security agent is capable of transmitting a boot code image to a secondary trusted security agent in another chiplet through a Boot Code Image Load Interface .	(LEVEL-FULL)
R_{PCHHY}	A secondary trusted security agent is capable of receiving a boot code image from the primary trusted security agent in another chiplet using the Boot Code Image Load Interface .	(LEVEL-FULL)

6.3.3 Trusted security agent interrupts

I_{JRXKG}	A trusted security agent receives interrupts that inform it about system events, messages from other controller entities or PEs, and actions that other security management components require. Trusted security agent interrupts do not follow the same path as interrupts targeted at Application PEs. An interrupt source sends them directly to the trusted security agent. If the target trusted security agent is on another chiplet, the interrupt is routed to the trusted security agent on the current chiplet. That trusted security agent forwards the interrupt as one or more messages on the Trusted Security Agent Communication Interface to the target trusted security agent.	
R_{RCPHY}	A trusted security agent receives interrupts directly from the source of the interrupt.	(LEVEL-0)
R_{JMCCT}	Trusted security agent interrupts targeted at a trusted security agent on another chiplet are sent to the trusted security agent on the current chiplet. The receiving trusted security agent then forwards them to the target trusted security agent as one or more messages on the Trusted Security Agent Communication Interface .	(LEVEL-0)

7

Chiplet interface implementation

$\mathcal{I}_{\text{GMDKN}}$	This chapter maps the interfaces in the chiplet type descriptions to the protocols and transport mechanisms used in chiplet implementations.
$\mathcal{I}_{\text{VMGWH}}$	This chapter specifies interfaces by reference to available standards and architectures. The FCSA does not define new interface standards. If no applicable standard is available for an interface, its transport and protocol are TO BE DEFINED.
$\mathcal{I}_{\text{KPDWL}}$	The protocols and transports that the FCSA specifies for interfaces might have multiple versions. A chiplet can be used in a system only if each interface that connects it to another chiplet uses either: • The same protocol and transport versions on both chiplets • Different protocol and transport versions that are mutually compatible When planning to reuse a chiplet, the system or chiplet designer must understand the roadmap for each protocol and transport under consideration and plan for backward compatibility between versions.
$\mathcal{U}_{\text{QLGCD}}$	Multiple interfaces might share a single transport if they have common transport requirements and the shared transport satisfies all their requirements. A note identifies each interface with this type of requirement.
$\mathcal{I}_{\text{GNDXP}}$	Sharing a transport between multiple interfaces might affect interface performance, including latency, transaction ordering, and deadlock avoidance mechanisms.
$\mathcal{U}_{\text{WJVQV}}$	A system component with its own interface between chiplets might provide an interface as a service. A note identifies each interface that might be implemented in this way.

7.1 Hardware fault requirements for chiplet interfaces

I_{DQSRJ}	Systems that have functional safety, reliability, availability, serviceability, or similar requirements might use mechanisms that detect, measure, report, correct, or repair transient or permanent hardware faults in chiplet interfaces.
U_{KHZLN}	If an implementation-specific chiplet interface fault model defines system-level hardware fault requirements, the interface implementation specified by the FCSA meets those requirements.
U_{STTGM}	The protocol or transport of an interface specified by the FCSA might include features that help meet system-level hardware fault requirements. This consideration applies to interfaces that use the Universal Chiplet Interconnect Express (UCIe) [7] and I3C [8] transport mechanisms. The definition of any TO BE DEFINED interface functionality considers these features.

7.2 Interface protocols

This section specifies the protocols for each interface.

7.2.1 Boot Sequence Control Interface

- I_{DSLFLZ} System components, for example a [trusted security agent](#) or a [system controller](#), use control signals to sequence boot code image loading and execution. A [Boot Sequence Control Interface](#) carries these signals when the components are in different chiplets. See [Chiplet System Component Boot Sequence](#) for more information.
- R_{JFNFLZ} The protocol for the [Boot Sequence Control Interface](#) is TO BE DEFINED. It carries *Ready to proceed* and *Error* from the *Subordinate* to the *Manager* and *Proceed* from the *Manager* to the *Subordinate*. The implementation delivers each signal reliably to its receiving agent.
- (LEVEL-FULL)

7.2.2 Boot Code Image Load Interface

- I_{FXRVR} Certain system components require their boot code images to be loaded across chiplet boundaries.
- R_{KJVCN} The protocol for the [Boot Code Image Load Interface](#) is TO BE DEFINED.
- (LEVEL-FULL)
- U_{PDHSQ} A system component, for example a [trusted security agent](#) or a [system controller](#), might provide the [Boot Code Image Load Interface](#) as a service.

7.2.3 Coherent Memory Traffic Interface

- I_{SBKRM} A [Coherent Memory Traffic Interface](#) between two chiplets enables coherent agents in either chiplet to access system-addressable memory and caches in other chiplets. The interface also transports the relevant snoop and Distributed Virtual Memory (DVM) messages.
- R_{JCLGK} The protocol for the [Coherent Memory Traffic Interface](#) is AMBA CHI Chip-to-Chip (CHI C2C) [9].
- (LEVEL-1)

7.2.4 Untranslated I/O Coherent Memory Traffic Interface

- I_{MVZPN} An [Untranslated I/O Coherent Memory Traffic Interface](#) enables I/O-coherent agents in a chiplet to access system-addressable memory and caches in other chiplets.
- R_{DPWJP} The protocol for the [Untranslated I/O Coherent Memory Traffic Interface](#) is TO BE DEFINED.
- (LEVEL-FULL)

7.2.5 I/O Coherent Memory Traffic Interface

- I_{JTWJV} An [I/O Coherent Memory Traffic Interface](#) enables I/O-coherent agents in a chiplet to access system-addressable memory and caches in other chiplets.
- R_{ZKQNK} The protocol for the [I/O Coherent Memory Traffic Interface](#) is TO BE DEFINED.
- (LEVEL-1)
- U_{LVWDK} A protocol option for the [I/O Coherent Memory Traffic Interface](#) is AMBA CHI Chip-to-Chip (CHI C2C) [9].

7.2.6 Non-Coherent Memory Traffic Interface

- I_{RGHYT} A [Non-Coherent Memory Traffic Interface](#) between two chiplets enables agents in either chiplet to access system-addressable memory in other chiplets.
- R_{WVPQS} The protocol for the [Non-Coherent Memory Traffic Interface](#) is TO BE DEFINED.
- (LEVEL-1)

U_{HHHBY} A protocol option for the [Non-Coherent Memory Traffic Interface](#) is AMBA CHI Chip-to-Chip (CHI C2C) [9].

7.2.7 Device Configuration Interface

I_{JXTCR} A [Device Configuration Interface](#) enables host software to configure a device in an expansion chiplet through a memory-mapped interface.

7.2.7.1 Device Configuration Interface for a Fully Coherent Expansion Chiplet

R_{ZNCXC} The protocol for the [Device Configuration Interface](#) for a [Fully Coherent Expansion Chiplet](#) is TO BE DEFINED.
(*LEVEL-1*)

U_{VNFGK} A protocol option for the [Device Configuration Interface](#) for a [Fully Coherent Expansion Chiplet](#) is AMBA CHI Chip-to-Chip (CHI C2C) [9].

7.2.7.2 Device Configuration Interface for a Fully Coherent Expansion (Remote Translation) Chiplet

I_{WNPTV} A device in a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) might be presented to the system as a PCIe Endpoint [6].

R_{DDXMP} The protocol for the [Device Configuration Interface](#) for a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is PCIe [6].
(*LEVEL-FULL*)

7.2.7.3 Device Configuration Interface for an I/O Coherent Expansion (Translated) Chiplet

R_{TZRTL} The protocol for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Translated\) Chiplet](#) is TO BE DEFINED.
(*LEVEL-1*)

U_{MDRHJ} A protocol option for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Translated\) Chiplet](#) is AMBA CHI Chip-to-Chip (CHI C2C) [9].

7.2.7.4 Device Configuration Interface for an I/O Coherent Expansion (Untranslated) Chiplet

R_{TQRTF} The protocol for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) is TO BE DEFINED.
(*LEVEL-FULL*)

7.2.7.5 Device Configuration Interface for an I/O Coherent Expansion (Remote Translation) Chiplet

I_{KWQSZ} A device in a [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) might be presented to the system as a PCIe Endpoint [6].

R_{NJGKB} The protocol for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is PCIe [6].
(*LEVEL-FULL*)

7.2.7.6 Device Configuration Interface for an I/O Chiplet

R_{ZFDJY} The protocol for the [Device Configuration Interface](#) for an [I/O Chiplet](#) is TO BE DEFINED.
(*LEVEL-FULL*)

7.2.7.7 Device Configuration Interface for an I/O Controller Chiplet

R_{LZXRR} The protocol for the [Device Configuration Interface](#) for an [I/O Controller Chiplet](#) is TO BE DEFINED.
(*LEVEL-FULL*)

7.2.8 Address Translation Interface

- I_{JNHGF}** An [Address Translation Interface](#) enables a device in one chiplet to request address translations from a memory management unit (MMU) component in a directly connected chiplet.
- R_{VLQZG}** The protocol for the [Address Translation Interface](#) is TO BE DEFINED. (LEVEL-FULL)

7.2.9 Interrupt handling

- I_{YFJMM}** This section concerns interrupts targeted at application PEs that run the main operating system.
- R_{RJNQG}** Interrupts are sent as Message Signaled Interrupt (MSI) messages. (LEVEL-FULL)
- R_{CWKYG}** Interrupt controller components can also forward interrupts through memory-mapped accesses. (LEVEL-FULL)

7.2.10 Trusted Security Agent Communication Interface

- I_{WYXND}** The [trusted security agent](#) in each chiplet provides hardware-enforced security and might provide higher-level security functionality. System-level hardware-enforced security is distributed across these trusted security agents. All secondary trusted security agents in the system delegate their security lifecycle management to the primary trusted security agent.
- R_{TDLVS}** The protocol for the [Trusted Security Agent Communication Interface](#) is TO BE DEFINED. (LEVEL-FULL)
- U_{XDGJH}** If the [Boot Code Image Load Interface](#) is implemented as a service of the [trusted security agent](#), the [Trusted Security Agent Communication Interface](#) is shared with this function.

7.2.11 System Control Interface

- I_{CLDTS}** The [system controllers](#) in the chiplets perform various system control and power management functions. Several of these functions require communication between controllers.
- R_{PXBTX}** The protocol for the [System Control Interface](#) is TO BE DEFINED. (LEVEL-FULL)
- U_{VSLJT}** The protocol options for the [System Control Interface](#) include the System Control and Management Interface (SCMI) [10] and the RISC-V Platform Management Interface (RPMI) [11].

7.2.12 Debug

7.2.12.1 Debug Access Interface

- R_{LXCQZ}** The protocol that an external debugger uses to access the Debug Access Port (DAP) of a chiplet is TO BE DEFINED. (LEVEL-FULL)
- U_{YBJGL}** A CoreSight debugger can access a Debug Access Port (DAP) that implements the Arm Debug Interface (ADI) [5].
- R_{KMLLD}** The protocol that a system debug agent in one chiplet uses to access the debug resources of another chiplet is TO BE DEFINED. The protocol uses memory-mapped accesses. (LEVEL-FULL)

7.3 Interface transports

This section specifies the transports for each interface.

7.3.1 Boot Sequence Control Interface

I_{CCDXR}	System components, for example a trusted security agent or a system controller , use control signals to sequence boot code image loading and execution. A Boot Sequence Control Interface carries these signals when the components are in different chiplets. See Chiplet System Component Boot Sequence for more information.
R_{ZGBTM}	The transport for the Boot Sequence Control Interface between two chiplets consists of three dedicated wires that use return-to-zero signaling: • A signal wire driven by the <i>Subordinate</i> that asserts the <i>Ready to proceed</i> signal to the <i>Manager</i> • A signal wire driven by the <i>Subordinate</i> that asserts the <i>Error</i> signal to the <i>Manager</i> • A signal wire driven by the <i>Manager</i> that asserts the <i>Proceed</i> signal to the <i>Subordinate</i> (LEVEL-FULL)
U_{FBDNL}	The transport for the Boot Sequence Control Interface might be implemented as a capability of the Trusted Security Agent Communication Interface or the System Control Interface . In this implementation, the dedicated-wire transport is unused.

7.3.2 Boot Code Image Load Interface

I_{LHQXN}	The system loads boot code images for certain components, for example a trusted security agent or a system controller , into component memory before activating any interchiplet software communication protocol that operates with mutable code. The system might transfer the images from one chiplet to another.
R_{CLTBQ}	The transport for the Boot Code Image Load Interface is an I3C interface [8]. The transport is available to the transmitting and receiving components before any software communication protocols that operate with mutable code are active. The transport includes hardware that converts transport transactions to memory-mapped I/O (MMIO) accesses. (LEVEL-FULL)
R_{JRHDK}	When available, the transport for the Boot Code Image Load Interface might also use the Coherent Memory Traffic Interface , I/O Coherent Memory Traffic Interface , or Non-Coherent Memory Traffic Interface . (LEVEL-FULL)
U_{GSFJN}	A system component, for example a trusted security agent or a system controller , might provide the Boot Code Image Load Interface as a service.

7.3.3 Coherent Memory Traffic Interface

I_{QQWYS}	A Coherent Memory Traffic Interface between two chiplets enables coherent agents in either chiplet to access system-addressable memory and caches in other chiplets. The interface also transports the relevant snoop and Distributed Virtual Memory (DVM) messages.
R_{ZZXCL}	The transport for the Coherent Memory Traffic Interface is TO BE DEFINED. (LEVEL-FULL)
U_{LKFMS}	The transport options for the Coherent Memory Traffic Interface include the UCle v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (LEVEL-FULL)
U_{YVNJF}	A more recent protocol version can be used if it is backward compatible with the specified version. (LEVEL-FULL)

7.3.4 Untranslated I/O Coherent Memory Traffic Interface

I _{BZSFF}	An Untranslated I/O Coherent Memory Traffic Interface enables I/O-coherent agents in a chiplet to access system-addressable memory and caches in other chiplets.
R _{QDDHW}	The transport for the Untranslated I/O Coherent Memory Traffic Interface is TO BE DEFINED. (<i>LEVEL-FULL</i>)
U _{SGMNK}	The transport options for the Untranslated I/O Coherent Memory Traffic Interface include the UCle v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (<i>LEVEL-FULL</i>)
U _{CVVHX}	A more recent protocol version can be used if it is backward compatible with the specified version. (<i>LEVEL-FULL</i>)

7.3.5 I/O Coherent Memory Traffic Interface

I _{RKZPK}	An I/O Coherent Memory Traffic Interface enables I/O-coherent agents in a chiplet to access system-addressable memory and caches in other chiplets.
R _{NCZJN}	The transport for the I/O Coherent Memory Traffic Interface is TO BE DEFINED. (<i>LEVEL-FULL</i>)
U _{DSWDX}	The transport options for the I/O Coherent Memory Traffic Interface include the UCle v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (<i>LEVEL-FULL</i>)
U _{NFWLC}	A more recent protocol version can be used if it is backward compatible with the specified version. (<i>LEVEL-FULL</i>)

7.3.6 Non-Coherent Memory Traffic Interface

I _{JHFWG}	A Non-Coherent Memory Traffic Interface between two chiplets enables agents in either chiplet to access system-addressable memory in other chiplets.
R _{BCQKB}	The transport for the Non-Coherent Memory Traffic Interface is TO BE DEFINED. (<i>LEVEL-FULL</i>)
U _{DLZKL}	The transport options for the Non-Coherent Memory Traffic Interface include the UCle v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (<i>LEVEL-FULL</i>)
U _{JQBWY}	A more recent protocol version can be used if it is backward compatible with the specified version. (<i>LEVEL-FULL</i>)

7.3.7 Device Configuration Interface

I _{KBLTT}	A Device Configuration Interface enables host software to configure a device in an expansion chiplet through a memory-mapped interface.
--------------------	---

7.3.7.1 Device Configuration Interface for a Fully Coherent Expansion Chiplet

R _{KBDRB}	The transport for the Device Configuration Interface for a Fully Coherent Expansion Chiplet is TO BE DEFINED. (<i>LEVEL-FULL</i>)
U _{JPLBP}	The transport options for the Device Configuration Interface for a Fully Coherent Expansion Chiplet include the UCle v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (<i>LEVEL-FULL</i>)
U _{PKVXM}	A more recent protocol version can be used if it is backward compatible with the specified version. (<i>LEVEL-FULL</i>)

7.3.7.2 Device Configuration Interface for a Fully Coherent Expansion (Remote Translation) Chiplet

R_{CJBHK} The transport for the [Device Configuration Interface](#) for a [Fully Coherent Expansion \(Remote Translation\) Chiplet](#) is PCIe over UCIe v1.1 [7].
(LEVEL-FULL)

7.3.7.3 Device Configuration Interface for an I/O Coherent Expansion (Translated) Chiplet

R_{YNCSL} The transport for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Translated\) Chiplet](#) is TO BE DEFINED.
(LEVEL-FULL)

U_{XQHWQ} The transport options for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Translated\) Chiplet](#) include the UCIe v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol.
(LEVEL-FULL)

U_{MRQTT} A more recent protocol version can be used if it is backward compatible with the specified version.
(LEVEL-FULL)

7.3.7.4 Device Configuration Interface for an I/O Coherent Expansion (Untranslated) Chiplet

R_{JDCPH} The transport for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) is TO BE DEFINED.
(LEVEL-FULL)

U_{NBRSY} The transport options for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Untranslated\) Chiplet](#) include the UCIe v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol.
(LEVEL-FULL)

U_{WRTTH} A more recent protocol version can be used if it is backward compatible with the specified version.
(LEVEL-FULL)

7.3.7.5 Device Configuration Interface for an I/O Coherent Expansion (Remote Translation) Chiplet

R_{MHKZW} The transport for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) is TO BE DEFINED.
(LEVEL-FULL)

U_{SZTZW} The transport options for the [Device Configuration Interface](#) for an [I/O Coherent Expansion \(Remote Translation\) Chiplet](#) include the UCIe v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol.
(LEVEL-FULL)

U_{VZYTP} A more recent protocol version can be used if it is backward compatible with the specified version.
(LEVEL-FULL)

7.3.7.6 Device Configuration Interface for an I/O Chiplet

R_{QZPSP} The transport for the [Device Configuration Interface](#) for an [I/O Chiplet](#) is TO BE DEFINED.
(LEVEL-FULL)

U_{XCWST} The transport options for the [Device Configuration Interface](#) for an [I/O Chiplet](#) include the UCIe v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol.
(LEVEL-FULL)

U_{MCMHZ} A more recent protocol version can be used if it is backward compatible with the specified version.
(LEVEL-FULL)

7.3.7.7 Device Configuration Interface for an I/O Controller Chiplet

R _{DDCF5}	The transport for the Device Configuration Interface for an I/O Controller Chiplet is TO BE DEFINED. (<i>LEVEL-FULL</i>)
U _{QRDSB}	The transport options for the Device Configuration Interface for an I/O Controller Chiplet include the UCIE v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (<i>LEVEL-FULL</i>)
U _{MYTQF}	A more recent protocol version can be used if it is backward compatible with the specified version. (<i>LEVEL-FULL</i>)

7.3.8 Address Translation Interface

I _{NBQFR}	An Address Translation Interface enables a device in one chiplet to request address translations from an MMU component in a directly connected chiplet.
R _{CTJXM}	The transport for the Address Translation Interface is TO BE DEFINED. (<i>LEVEL-FULL</i>)
U _{QNVFW}	The transport options for the Address Translation Interface for an I/O Coherent Expansion (Remote Translation) Chiplet include the UCIE v1.1 [7] streaming protocol and the OCP Bunch of Wires (BoW) [12] protocol. (<i>LEVEL-FULL</i>)
U _{NXBZH}	A more recent protocol version can be used if it is backward compatible with the specified version. (<i>LEVEL-FULL</i>)

7.3.9 Interrupt handling

I _{CCVFN}	This section concerns interrupts targeted at application PEs that run the main operating system.
R _{ZRGXC}	Interrupts are sent as MSI messages. (<i>LEVEL-FULL</i>)
R _{HSZXD}	Interrupt controller components can also forward interrupts through memory-mapped accesses. (<i>LEVEL-FULL</i>)
R _{BGHPZ}	The chiplet type determines which of the following interfaces transports interrupts: • Coherent Memory Traffic Interface • I/O Coherent Memory Traffic Interface • Non-Coherent Memory Traffic Interface (<i>LEVEL-FULL</i>)

7.3.10 Trusted Security Agent Communication Interface

I _{DRPRG}	The trusted security agent in each chiplet provides hardware-enforced security and might provide higher-level security functionality. System-level hardware-enforced security is distributed across these trusted security agents. All secondary trusted security agents in the system delegate their security lifecycle management to the primary trusted security agent.
R _{VTPDZ}	The transport for the Trusted Security Agent Communication Interface is an I3C interface [8]. (<i>LEVEL-FULL</i>)
X _{LPGMG}	The I3C interface [8] is a standard interface suitable for transporting the Trusted Security Agent Communication Interface .
U _{PXGQL}	If the Boot Code Image Load Interface is implemented as a service of the trusted security agent , the Trusted Security Agent Communication Interface is shared with this function.
U _{SXWDL}	The Boot Sequence Control Interface might be implemented as a capability of the Trusted Security Agent Communication Interface .

7.3.11 System Control Interface

I _{SJSXS}	The system controllers and system control agents in the chiplets perform various system control and power management functions. Several of these functions require communication between controllers.
I _{FKMPZ}	Multiple transports between chiplets can carry communications between system controllers and system control agents. The transport depends on the chiplet activity state and which interfaces are available in that state.
R _{XYZYJ}	When two directly connected chiplets are both in the <i>Operational</i> chiplet activity state , the Coherent Memory Traffic Interface is available and transports the System Control Interface . (LEVEL-FULL)
R _{GTDWN}	When either or both of two directly connected chiplets are in the <i>Secure boot</i> , <i>Initial boot</i> , or <i>Power saving</i> chiplet activity state , the transport for the System Control Interface is TO BE DEFINED. (LEVEL-FULL)
X _{ZPTVP}	The I3C interface [8] is a standard interface suitable for transporting the System Control Interface .
U _{NLWKD}	If the Boot Code Image Load Interface is implemented as a service of the system controller , the System Control Interface is shared with this function.
U _{VHFLT}	The Boot Sequence Control Interface might be implemented as a capability of the System Control Interface .
U _{YNTTZ}	A dedicated WAKEUP wire signal might be required between chiplets that can enter power modes in which the System Control Interface is unavailable, for example the <i>Power saving</i> chiplet activity state .

7.3.12 Debug

7.3.12.1 Debug Access Interface

R _{HKNHZ}	The transport that an external debugger uses to access the Debug Access Port (DAP) of a chiplet is a Joint Test Action Group (JTAG) interface that complies with the Institute of Electrical and Electronics Engineers (IEEE) Standard 1149.1. (LEVEL-FULL)
U _{BGJJT}	Each chiplet provides debug access through its JTAG Test Access Port (TAP). A debugger can access each port individually, or multiple ports can form a daisy chain. In a daisy-chain configuration, all chiplets share the Test Clock (TCK) and Test Mode Select (TMS) signals. The Test Data In (TDI) and Test Data Out (TDO) signals connect the chiplets in series: the TDO of each chiplet connects to the TDI of the next chiplet.
R _{LNKLH}	A system debug agent in one chiplet uses memory-mapped accesses and, when available, the Coherent Memory Traffic Interface , I/O Coherent Memory Traffic Interface , or Non-Coherent Memory Traffic Interface to access the debug resources of another chiplet. (LEVEL-FULL)

8

Implementation constraints

This chapter describes implementation constraints that are not specific to individual chiplet types but support chiplet interoperability and successful integration into scalable systems.

This information helps system designers evaluate and integrate different chiplet designs. Some chiplets are co-designed for a specific chiplet system. Other chiplets are developed independently for use in different system architectures. Defining system-specific implementation constraints requires a thorough understanding of the system requirements, which is outside the scope of the FCSA. This chapter identifies the principal constraints that affect chiplet interoperability and system scalability.

This chapter contains the following sections:

- [Interoperability of chiplets](#)
- [Chiplet system scalability](#)

8.1 Interoperability of chiplets

The FCSA does not describe every aspect of the functionality that chiplets in a system implement. A complete description would enable all compliant implementations of that functionality to interoperate, but would exceed the scope of the FCSA. However, chiplets must implement functionality that relies on inter-chiplet interactions in an interoperable manner.

This section describes the constraints on implementations of system functions that must be consistent across chiplets to ensure correct operation. It also identifies implementation differences that can undermine interoperability. These constraints define the mutual expectations of chiplets that are designed for integration into a system.

8.1.1 Memory system

8.1.1.1 Transport of memory traffic

<code>I_{RBXPJ}</code>	A chiplet might transport memory traffic to another connected chiplet. Each interconnect fabric in a chiplet has a set of properties that define its transaction-handling capabilities. The Properties, Parameters, and Broadcast Signals chapter of AMBA CHI [4] lists these properties.
<code>I_{ZTTPQ}</code>	When a memory transaction is transferred between chiplets with different interconnect properties, the transaction might require adaptation to maintain protocol compliance. For example, adaptation can reduce a field, extend a field with zero padding, or generate an error response. The Protocol layer property exchange chapter of AMBA CHI Chip-to-Chip (CHI C2C) [9] lists the expected adaptations.
<code>R_{QLFLC}</code>	When a memory transaction is transported between chiplets with different interconnect properties, transaction handling is adapted as necessary to comply with the protocol. The adaptation occurs before the transaction is transferred over the chiplet memory traffic interface. (LEVEL-1)
<code>R_{CCBNN}</code>	If a memory transaction is transported between chiplets with different interconnect properties, the differences do not prevent successful transmission or reception. The differences also do not affect the ability to meet the performance requirements of related system functionality. Which interconnect properties are significant when they differ between chiplets is IMPLEMENTATION DEFINED. Examples include address-field and ID-field widths. (LEVEL-1)

8.2 Chiplet system scalability

Some details of how chiplets implement system functionality affect scalability. Compatibility of these implementation details across chiplets can improve the scalability of the system functionality.

This section identifies implementation details that affect system performance as chiplets are added to a design. System and chiplet designers can use these topics when they evaluate the scalability of a chiplet-based design.

This list is not exhaustive.

8.2.1 Memory system

8.2.1.1 Snoop filters

I_{BQTRV} A chiplet might contain one or more snoop filters. Other chiplets might contain requesters with coherent caches. When a snoop filter reaches capacity, an old entry is evicted from all caches to make space for a new entry.

R_{TPXLG} A snoop filter in a chiplet has sufficient capacity to meet the combined demand from all chiplets in the system without compromising expected system functionality.

(*LEVEL-FULL*)

9

Security of chiplet systems

A secure system operates as intended and protects the assets of the device manufacturer, owner, and user. This typically requires a compute platform that ensures authentic system hardware runs authorized firmware and software. For example, secure systems include:

- A hardware-enforced Trusted Execution Environment (TEE) that runs trusted firmware to isolate sensitive operations from the main operating system
- A hardware Root of Trust (RoT) implementation that uses a secure enclave to establish a chain of trust from power-on to application execution

Established techniques can secure a monolithic system-on-chip (SoC). Chiplet disaggregation exposes package-internal trust boundaries that are implicit in a monolithic SoC. System security becomes a composed state that depends on the identity, provisioning state, firmware state, configuration, lifecycle state, and communication paths of independently developed and integrated chiplets. A chiplet system therefore needs explicit mechanisms to establish, verify, and maintain its intended composition.

This specification uses the following security objective for the analysis:

SoC-equivalent security for chiplet systems: a chiplet system should provide the same level of security as an equivalent monolithic SoC implementation.

This analysis covers the system-level security differences that result from partitioning a system design into multiple chiplets in a system-in-package (SiP). These differences result from:

- Novel attack vectors
- Monolithic SoC security model assumptions invalidated by disaggregation into chiplets
- Supply-chain security assumptions invalidated by chiplet design and integration flows

This analysis considers secure chiplet systems at an abstract system-design level. It does not define a definitive set of security requirements for any specific product.

This analysis does not replace a product-specific security risk assessment. System and chiplet designers should use it as a basis for evaluating these threats against their product design and security needs.

This analysis has two main parts:

- A definition of the [chiplet system security model](#) used by the analysis
- A [threat analysis](#) with associated mitigations

9.1 Chiplet system security model

This section defines an abstract chiplet system security model. The model captures system-security aspects affected by chiplets without depending on a specific topology or implementation.

This model refers to a *Root of Trust* (RoT). A RoT is a minimal set of software, hardware, and data that is implicitly trusted in a chiplet or SoC. No lower-level software or hardware component can report that the RoT is authentic and unmodified. An *Initial RoT* is based on the first immutable code that runs after boot. The code can be physically immutable in ROM or provisioned, initialized, and locked by the chiplet or SoC manufacturer in one-time-programmable (OTP) memory. For a widely accepted definition, see *GlobalPlatform Root of Trust Definitions and Requirements* [13].

This model complements established secure system frameworks and designs, including the PSA Certified Platform Security Model [14], Caliptra [15], the OCP Security Appraisal Framework and Enablement (OCP S.A.F.E.) [16], and the Seven Properties of Highly Secure Devices [17].

A secure system typically uses the following established techniques:

- Secure management of hardware and software design and manufacturing processes
- A secure manufacturing process that provisions initial security credentials, for example a secret cryptographic identity and firmware verification keys
- Management of the system security lifecycle state, including clear differentiation between preproduction and postproduction states
- A locked post-provisioning system state that protects the credentials
- An implicitly trusted Initial RoT that anchors the chain of trust
- Authentication of mutable firmware and software before execution, initially by the Initial RoT
- Cryptographically verifiable attestation evidence that originates from the RoT
- Hardware controls that isolate the RoT and other security-critical subsystems
- Control of invasive access technologies, for example debug, within the RoT

A secure system can reliably produce verifiable reports of its identity and security state. The system identity can be linked to a security evaluation and certification that covers system design, manufacturing, provisioning, and testing of a representative instance. Verified attestation and certification enable the system to protect additional assets and operate as a trustworthy component in a larger system.

The components of this chiplet system security model are:

- The necessary [assumptions](#)
- The relevant [stakeholders](#)
- The [security domains and trust boundaries](#)
- The [chiplet system security platform and Root of Trust framework](#)
- The [chiplet system lifecycle](#)
- General [chiplet system topology models](#)
- [Adversarial models](#)
- The [chiplet system security goals](#)

9.1.1 Assumptions

9.1.1.1 Impact of chiplet disaggregation on security assumptions

Disaggregating a monolithic system into chiplets breaks several assumptions that underpin monolithic SoC security:

Platform hardware inseparability

In a monolithic SoC, the platform hardware is assumed to be inseparable.

Combined with production-line audits, this assumption provides confidence that a production system matches the security evaluation target.

For a chiplet system, this assumption is fragile because different entities can design, manufacture, and integrate the chiplets. Chiplet substitution during manufacturing, or perhaps after integration, might also be

possible. Because overall security depends on multiple independent entities, demonstrating that a production system matches the security evaluation target becomes harder.

The system needs evidence that the integrated chiplets, interposers, firmware images, manifests, and security-critical configuration match the intended composition.

Isolated die evaluation

For a monolithic SoC, the security of a die can be evaluated in isolation.

The security boundary of the target of evaluation lies entirely inside the SoC. The security evaluation does not depend on the security of external entities.

Chiplet disaggregation breaks this assumption because overall system security depends on multiple integrated chiplets, and potentially interposers.

Secure-by-design protection configuration

For a monolithic SoC, the configuration of protection mechanisms is assumed to be secure by design.

Mechanisms that set or query security-related hardware configurations are isolated from untrusted parts of the system. Analysis of the monolithic SoC design demonstrates this isolation.

In a chiplet system, security mechanism configuration can be distributed across multiple chiplets, so the interactions between those chiplets must also be considered.

9.1.1.2 Supply chain

The system supply chain can range from simple to highly complex. This analysis assumes that:

- Individual chiplet dies can be custom designs for the system or off-the-shelf components from a chiplet marketplace that are not designed for a specific SiP.
- System integration can occur in the same location or organization as chiplet manufacturing, or elsewhere by another entity.

Simpler supply chains can reduce some threats, but constraining the supply chain is not always possible or optimal.

9.1.1.3 Serviceability

This analysis does not assume support for field replacement, hot-plug, or rebinding of chiplets. It assumes that integration provisioning creates a stable authorized composition that the system verifies during later boots and operation.

Post-integration replacement of an individual chiplet would require a different lifecycle model. The system would reenter an integration-like service state and provision the updated package as a new authorized composition. This design would need a product-specific policy for replacement authorization, chiplet authentication, manifest update, revocation or retirement of the removed chiplet identity, rollback protection, key rotation or rederivation, and system attestation of the new composition.

This serviceability case is out of scope and differs from adversarial chiplet substitution before or during package integration. **T.SpoofChiplet** covers adversarial substitution.

Replaceable-chiplet systems are out of scope.

9.1.1.4 Established security techniques

This analysis assumes that chiplet systems apply established security techniques. The following examples provide context for controls discussed later in this chapter:

- Secure, measured boot for the RoT
- Secure firmware and software update
- RoT identity authentication and attestation
- Protection of sensitive data and key material in use, in transit, and at rest
- Authentication and integrity protection of security-relevant interfaces, with confidentiality protection where required

- Hardware protection of the RoT, including countermeasures against physical attacks that are in scope for the product
- Credential lifecycle management, including revocation, replacement, and, where appropriate, rotation or destruction

9.1.2 Stakeholders

A stakeholder is a person or group that places value in the system, or a part of it. The stakeholders identified for a chiplet system are listed in [Table 9.1](#).

Table 9.1: System security stakeholders

Stakeholder	Rationale
FCSA (or layered specification) developer	A breach involving an asset in a chiplet system specified by this architecture can damage the reputation and revenue of the developer.
Chiplet system designer	The designer specifies how the chiplet system must function. A breach involving a system asset can damage the reputation and revenue of the designer.
Chiplet manufacturer	A breach involving a chiplet product can affect a system that uses the product and damage the reputation and revenue of the manufacturer. A chiplet product includes the chiplet design, fabricated chiplet, and structures that support metrology or diagnostics.
Interposer manufacturer	A breach involving an interposer product can affect a system that uses the product and damage the reputation and revenue of the manufacturer.
Packaging provider	The provider assembles and packages the system to meet integrity and quality expectations. A breach involving a system asset can damage the reputation and revenue of the provider.
Original equipment manufacturer (OEM)	The chiplet system integrator constructs the system for the OEM, which vouches for correct operation after deployment. A breach involving a system asset can damage the reputation and revenue of the OEM.
End user	The end user entrusts private information to the system and expects its functions to remain available. A breach involving a system asset can reduce the trust of the end user in the system.

9.1.3 Security domains and trust boundaries

A *trust boundary* is a logical security perimeter that separates *security domains*. A *security domain* protects its *assets* by enforcing a security policy that controls access to and modification of those assets by entities and other security domains.

Security domains have no implicit global privilege ordering. A domain can access assets or services in another domain only if the domain that owns those assets or services explicitly authorizes the access through its interfaces and security policy. A security policy can define relative privilege for a specific asset or function, but that relationship does not apply to other assets or functions.

Security assets include information, system resources, services, and properties that stakeholders value. These properties include confidentiality, integrity, and availability.

A security domain can be implemented in software, hardware, or a combination of both. Security domains can arise through different mechanisms:

- Physical design

Examples include an integrated and isolated secure enclave or a physically separate chiplet.

- Programmable isolation mechanisms that boot firmware configures and makes immutable for the remainder of system runtime

For example, a TEE.

- Transient domains that a security policy authorizes another domain to construct, configure, and operate using a programmable hardware-based isolation mechanism

For example, a virtual machine (VM) within a hypervisor, or an application process within an operating system.

Disaggregation does not always change the intended access policy between security domains. However, when a trust boundary crosses chiplets, disaggregation changes the mechanisms and evidence required to enforce and assess that policy. These mechanisms can include evidence of chiplet identity and state, endpoint authorization, and inter-chiplet path protection.

The following figure shows a system with two chiplets, A and B, connected by a die-to-die interface. This system supports analysis of the security impact of chiplet disaggregation.

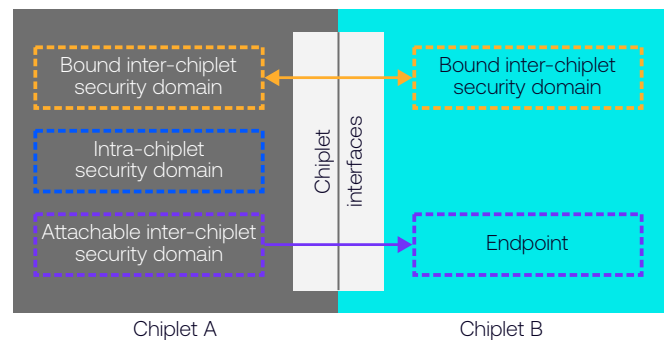


Figure 9.1: Chiplet system security domains

The trust boundaries define three types of security domain:

Intra-chiplet security domain of chiplet A.

An intra-chiplet security domain lies entirely within chiplet A. Examples include an integrated secure element and a security processor. If the security policy and enforcement of the domain have no security-relevant dependency on chiplet B or the die-to-die interface, disaggregation does not change the analysis of threats within the domain. These threats remain relevant to overall system security but are outside the scope of this analysis.

Bound inter-chiplet security domain that spans chiplet A and chiplet B.

A bound inter-chiplet security domain spans chiplets A and B for the lifetime of the domain. This domain type can arise when both chiplets provide compute cores connected coherently across the chiplet interface. Examples include memory address domains at different privilege levels, including a VM, OS kernel, or application process. An individual domain might be temporary, but equivalent access from chiplets A and B is inherent in this domain type. The security evaluation depends on both chiplets and their die-to-die interface.

Attachable inter-chiplet security domain that spans chiplet A and chiplet B.

This transient domain spans chiplets A and B, and access by chiplet B is temporary. This domain type can arise when components in chiplet B act as an endpoint, for example a device assigned to a confidential compute VM on chiplet A. The security evaluation depends on both chiplets and their die-to-die interface.

Inter-chiplet security domains that cross chiplet boundaries present specific security challenges. For example, chiplet A must determine whether chiplet B runs secure firmware, configures the isolation of the shared domain correctly, and disables debug access. These conditions are necessary to maintain the domain security properties.

For a bound inter-chiplet security domain, these checks are not limited to integration time. The System RoT must derive a composed security state from chiplet-local evidence, bind that state to system policy, and respond when a participating chiplet changes state in a way that invalidates a previous trust decision.

For an attachable inter-chiplet security domain, established designs and protocols can provide the required assurance. For example, the PCIe TEE Device Interface Security Protocol (TDISP) and Integrity and Data Encryption (IDE) capabilities enable a PCIe TEE device to be dynamically trusted and assigned to a VM, even when the connectors and bus might be exposed to an attacker [18].

Because established techniques exist for intra-chiplet and attachable inter-chiplet security domains, this analysis focuses on threats and mitigations for bound inter-chiplet security domains.

9.1.4 Chiplet system security platform and Root of Trust framework

A *System RoT* in a monolithic SoC provides security services that represent the security state of the entire system to external entities or internal entities under external control. For example, it can attest the security state of the system to an external verifier or a local verifier controlled by an external entity.

A chiplet system implements a *Security Platform* that provides the system-level security services expected from the System RoT in a monolithic SoC.

The following approach uses established hardware RoT techniques for chiplet identification, authentication, and attestation. This specification defines a *Chiplet System Security Platform Framework* with multiple components that work together to establish system trust. The following figure shows an example topology.

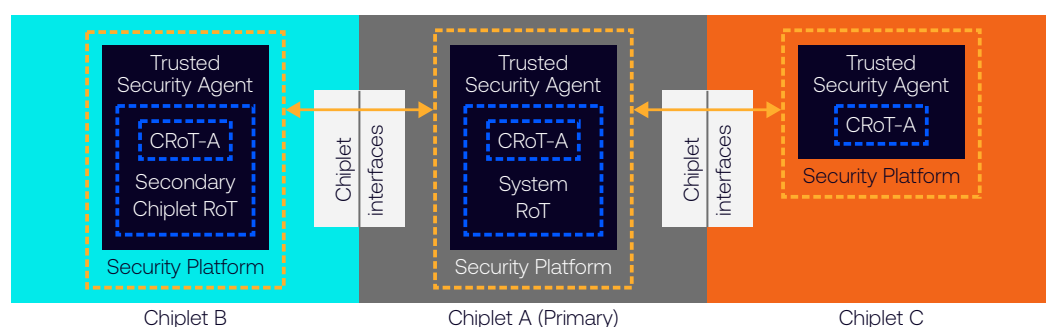


Figure 9.2: Example chiplet system security platform topology

The Security Platform spans the whole system. The *System RoT* in the primary chiplet forms the platform when it accepts verifiable local evidence from participating *Chiplet RoT for Attestation (CRoT-A)* instances. It also admits any *Secondary Chiplet RoT* instances that provide additional security services. The CRoT-A is the minimum evidence root for a participating chiplet. The framework requires cryptographically bound delegation, verifiable chiplet-local evidence, and an unambiguous system-level decision about whether the composed trust domain can exist. It does not require symmetric roles between chiplets. Each System RoT and Secondary Chiplet RoT role is built on a CRoT-A, contains one, or has its state evidenced by one.

The following subsections define the elements of the Chiplet System Security Platform Framework.

9.1.4.1 Security platform

The Security Platform is a bound inter-chiplet security domain that spans the whole system. It results from establishing trust and secure communication between the System RoT, all CRoT-As, and any Secondary Chiplet RoTs that provide additional security services.

The Security Platform provides the system-level security services expected from the RoT in a secure monolithic SoC. For example, it provides verifiable attestation reports on the system security state.

The Security Platform is formed only after the System RoT has accepted the participating chiplet identities, evidence, topology, and policy bindings. It is maintained only while those system-level security conditions satisfy the security policy.

9.1.4.2 System RoT

The *System RoT* of a SiP is similar to that of a monolithic SoC, but it must establish and maintain the security status of the composed system.

The System RoT is an intra-chiplet security domain in the primary chiplet that provides system-level identity and attestation services to relying parties. It can share the security domain of the CRoT-A in the primary chiplet or extend that CRoT-A. It can also be an independent RoT whose identity, firmware, lifecycle, configuration, and security state are evidenced by the CRoT-A. After successful chiplet verification, the System RoT establishes the Security Platform.

The System RoT makes system-scoped composition decisions. It determines the intended package composition and expected topology, authenticates participating chiplets, validates CRoT-A evidence, binds chiplet identities to the system manifest and policy, and decides whether the Security Platform can be formed or maintained.

The System RoT secures interfaces to Secondary Chiplet RoTs and CRoT-As that provide runtime security services, for example access to chiplet security configuration. It establishes or authorizes protected security control-plane associations and distributes or authorizes system-level policy. It applies the defined system response when a chiplet is absent, fails attestation, reports stale or revoked evidence, or enters a lower-assurance state.

The System RoT collects and authenticates CRoT-A attestations and combines them into the system-level attestation report.

The System RoT executes mutable firmware.

The external interface to the Security Platform is through the System RoT. With support from Secondary Chiplet RoTs and CRoT-As, the System RoT has the security properties of the equivalent System RoT in a monolithic SoC.

9.1.4.3 Secondary Chiplet RoT

A Secondary Chiplet RoT is an optional intra-chiplet security domain in a non-primary chiplet that provides functionality beyond the minimum CRoT-A function. For example, it might provide runtime trusted services, protected control-plane endpoints, local policy enforcement, or secure interfaces with the System RoT and other chiplet security components.

A Secondary Chiplet RoT can support the System RoT with additional chiplet-local security services, but is not responsible for system-level identity, system-level attestation, or the decision to form the Security Platform. Its admission into the composed system trust model depends on evidence from the CRoT-A of the chiplet.

A chiplet design might support multiple system topologies. Therefore, a Chiplet RoT might be capable of acting as the System RoT or a Secondary Chiplet RoT according to the system topology. Its implementation might be similar to that of the System RoT, but System RoT-specific capabilities are not exercised when it is designated as a Secondary Chiplet RoT.

A Secondary Chiplet RoT can share the security domain of the CRoT-A in the chiplet or extend that CRoT-A. It can also be an independent RoT whose identity, firmware, lifecycle, configuration, and security state are measured or attested by the CRoT-A.

A Secondary Chiplet RoT executes mutable firmware.

9.1.4.4 Chiplet RoT for Attestation (CRoT-A)

Each chiplet that participates in attested composition assurance contains a CRoT-A.

A CRoT-A is the minimum chiplet-local evidence root for the composed system. It is an intra-chiplet security domain that authenticates chiplet identity and security-critical state, and produces cryptographic evidence. It is responsible for chiplet-local security state and local enforcement for security-sensitive functions that cannot safely be delegated to another chiplet. A CRoT-A has the following properties:

- Custody of the unforgeable and provable chiplet identity
- A role as the Initial RoT for the chiplet, or an implementation rooted in that RoT
- Measurement of chiplet-local boot firmware, and verification when the boot model uses authenticated firmware loading
- Maintenance of the chiplet security lifecycle state
- Control and reporting of debug, test, scan, manufacturing, failure-analysis, and invasive access mechanisms
- Attestation of chiplet identity and security state, including firmware, lifecycle, debug, and security-critical configuration state
- Acceptance of policy and configuration commands, including commands that provision or manage security-related assets, only over an authenticated and authorized security control plane

The storage location or delivery path of mutable firmware does not determine whether it is trusted. How a CRoT-A establishes trust in its own firmware depends on its construction:

- A CRoT-A might provide its functionality without mutable firmware of its own. In this case, no additional mutable-firmware authentication is needed at this boundary, assuming the immutable hardware or firmware is designed and provisioned with appropriate security controls.
- A CRoT-A might load mutable firmware from local storage. In this case, the immutable part of the CRoT-A must authenticate the firmware before execution by using protected local reference information, for example a trust anchor or firmware digest. A trust anchor can be a firmware-signing public key or the public key of a certificate authority.
- A CRoT-A might receive mutable firmware from another component and authenticate the firmware itself before execution. This method has the same security requirement as the local-storage case. Only the firmware delivery path differs.
- A CRoT-A might receive mutable firmware from another component but be unable to authenticate the firmware image directly against protected local reference information. In this case, the immutable part of the CRoT-A must authenticate and authorize the supplying component. The immutable part must also establish that any security-relevant mutable state of that component is acceptable. The supplying component must authenticate the firmware and enforce the applicable rollback policy. The transfer mechanism must protect the image and bind the transfer to the intended recipient and image identity. These measures ensure that the CRoT-A receives the same image that the supplying component verified. The CRoT-A still requires a protected local trust anchor, credential, expected component identity, or shared secret. Each component must protect the credentials that it uses.

Equivalent requirements apply to security-critical configuration and other mutable assets that affect the CRoT-A. Before using a mutable asset, the CRoT-A must establish the authenticity, integrity, and applicable rollback state of that asset. The CRoT-A must also protect the reference information and any delegated verification relationship against the threats in the relevant adversarial models.

If the system design delegates control of mutable, security-critical CRoT-A state to the System RoT, the CRoT-A accepts changes only through authenticated and authorized interfaces. This state can include security lifecycle state, debug control, hardware security isolation, and communication protection. The CRoT-A still maintains chiplet-local policy and protects local assets when delegation to another chiplet would undermine the local trust boundary.

The isolated component that contains a CRoT-A might also contain the System RoT or a Secondary Chiplet RoT. Alternatively, it might support an independent System RoT or Secondary Chiplet RoT by loading, authenticating, measuring, or attesting the mutable firmware and critical state of that RoT. These additional services are distinct roles. They do not expand the minimum CRoT-A definition.

When a CRoT-A supports these additional roles, the services must be protected and evidenced as part of the composed system trust decision. CRoT-A evidence must cover the firmware and security-critical state of the supported RoT role. Interfaces that establish or report that state need the protection expected for security control-plane paths.

9.1.4.5 Establishing and maintaining composition assurance

At system boot, each participating CRoT-A:

- Establishes the chiplet identity
- Establishes trust in its firmware
- Determines the lifecycle and debug state
- Applies local policy to secrets and security-sensitive functions
- Prepares attestation evidence

The System RoT then:

- Discovers participating CRoT-A endpoints and admitted Secondary Chiplet RoT endpoints
- Challenges each CRoT-A
- Authenticates the evidence against provisioned trust anchors and certificate chains, and checks the applicable revocation state
- Validates the freshness of the evidence
- Appraises the reported measurements and security state against trusted reference values, the authorized rollback state, the system manifest, and security policy
- Derives a system-level boot or initialization state
- Checks that the observed chiplet set, roles, placement, and topology match the authorized composition

To form the Security Platform, the System RoT must not depend on services that become available only after the platform is formed. The System RoT must locally authenticate and appraise sufficient evidence to decide whether the Security Platform can be formed. If complete appraisal requires an external verifier that is unavailable during early boot, the security policy must keep the system in a restricted bootstrap state until the external appraisal succeeds.

If the evidence is accepted, the System RoT establishes or derives protected security control-plane associations with the participating chiplets, binds system policy to the accepted chiplet state, and authorizes formation of the Security Platform. During runtime, CRoT-As remain responsible for chiplet-local enforcement and local state reporting, while the System RoT maintains the system trust model and triggers revalidation on reset, warm reset, chiplet fault, debug unlock, firmware update, recovery, or any other state change that can affect the Security Platform.

The CRoT-As, System RoT, and Secondary Chiplet RoTs are implemented by Trusted Security Agent components. See [Trusted Security Agent](#).

9.1.5 Chiplet system lifecycle

[Figure 9.3](#) shows a typical chiplet system lifecycle. Each stage identifies the stakeholders involved in or directly affected by that stage. [Table 9.2](#) describes each stage. A comprehensive lifecycle model can also include states such as recoverable and nonrecoverable debug modes. However, this analysis focuses on the lifecycle stages most relevant to the security implications of chiplet disaggregation.



Figure 9.3: Chiplet system lifecycle

Table 9.2: Chiplet system lifecycle stages and stakeholders

Stage	Stakeholders	Description
Manufacturing	Intellectual property (IP) providers, chiplet manufacturer	Fabrication of the chiplets
Chiplet provisioning	IP providers, chiplet manufacturer, interposer manufacturer	Security provisioning of individual chiplets and any interposers
Integration provisioning	Chiplet system designer, packaging provider	Security provisioning of chiplets when they and any interposers are composed into a system and packaged
OEM provisioning	OEM	Security provisioning of the packaged chiplet system when deployed by the product OEM
Operation	IP providers, chiplet system designer, chiplet manufacturer, OEM	Initial boot, establishment of the Security Platform across multiple chiplets, and system operation
Disposal	IP providers, chiplet system designer, chiplet manufacturer, OEM	Revocation of security provisioning as part of a managed return material authorization (RMA) process or disposal of the chiplet system

The stages in [Table 9.2](#) describe system development, provisioning, deployment, operation, and disposal activities. They do not enumerate system security states. Each chiplet has its own lifecycle state. The System RoT evaluates attested chiplet-local lifecycle and other security evidence against system policy to derive and report a product-defined system security state. This reported state does not need to correspond directly to the state of any individual chiplet.

A key lifecycle change from a monolithic SoC is the division of provisioning into two stages: chiplet provisioning at the die manufacturing facility and integration provisioning at the SiP integration facility.

If different stakeholders perform die manufacturing and SiP integration, the chiplet supply chain presents a novel attack vector.

Product-specific RMA handling is part of the disposal stage rather than operational serviceability. This handling can transition a returned system into a diagnostic or disposition state before any failure analysis or debug access is authorized. Product policy must define a separate transition for field service, replacement, reprovisioning, or rebinding.

9.1.6 Chiplet system topology models

To identify and assess chiplet security risks, this analysis uses two simplified model systems. Attacks against more complex systems can be mapped to one of these models.

9.1.6.1 Two-chiplet A+B model

The following figure shows the two-chiplet A+B model.

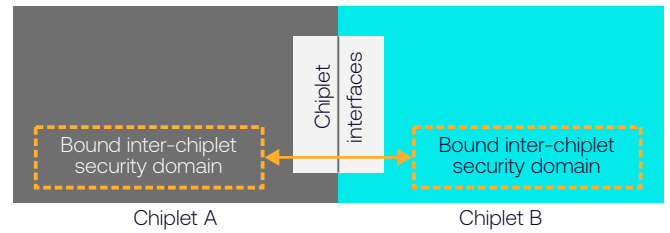


Figure 9.4: Two-chiplet A+B model

In this model, a bound inter-chiplet security domain spans chiplets A and B. Data interfaces connect the two chiplets.

9.1.6.2 Multi-chiplet A+B+E model

The following figure shows the multi-chiplet A+B+E model.

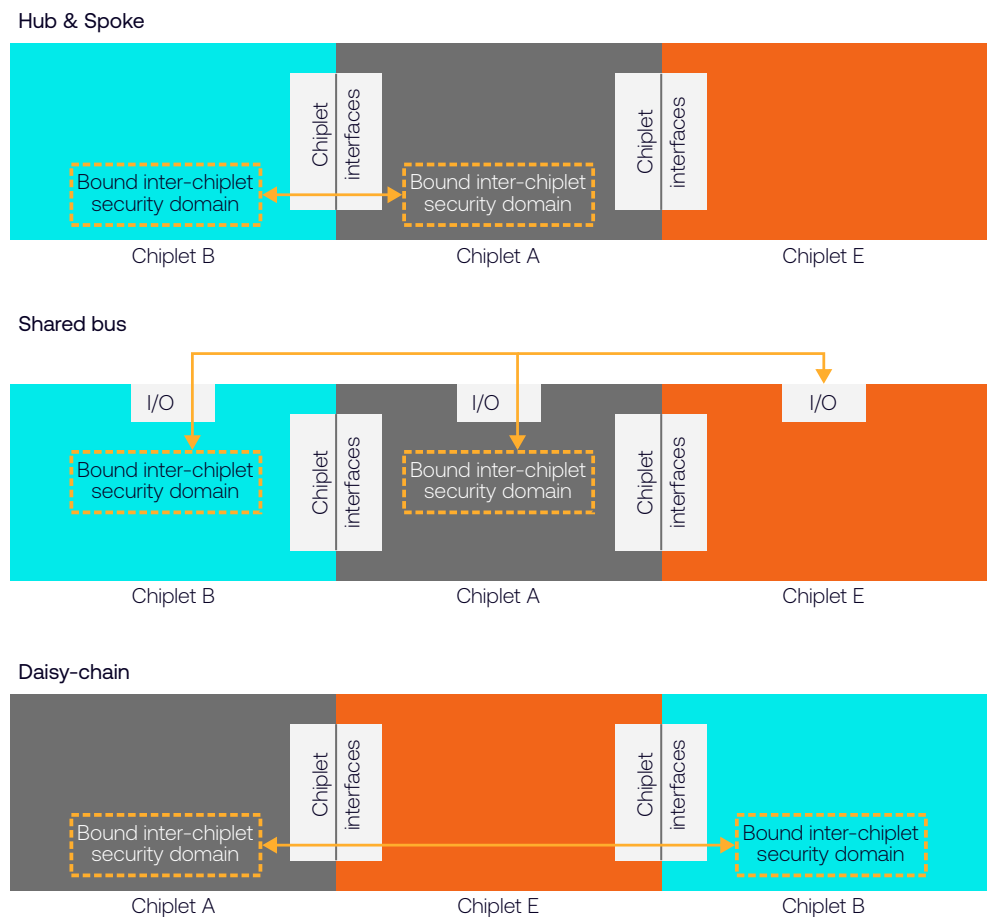


Figure 9.5: Multi-chiplet A+B+E model

This model extends the two-chiplet A+B model by adding chiplet E. Chiplet E is outside the bound inter-chiplet security domain that spans chiplets A and B.

For communication from chiplet A to chiplet B, the data interfaces have three possible topologies:

- Hub and spoke: chiplet B and chiplet E each have a dedicated connection to chiplet A that is inaccessible to the other chiplet.

- Shared bus: all three chiplets can communicate through a shared bus, for example, a common channel in the package or printed circuit board (PCB).
- Daisy-chain: chiplet B is not directly connected to chiplet A and communication from chiplet A to chiplet B must transit chiplet E.

9.1.6.3 Inter-chiplet path classes

The security properties required for an inter-chiplet path depend on the path function, physical exposure, and topology. A product-specific assessment should distinguish the following path classes:

Security control-plane paths

These paths are used for chiplet authentication, attestation, provisioning, boot sequencing, policy distribution, debug authorization, and trust-domain formation. Compromise of a security control-plane path can undermine the trust-establishment process for the system, even if the main-band data paths are physically less exposed or independently protected.

These paths generally require endpoint authentication, freshness, replay protection, integrity protection, and confidentiality where secrets or sensitive state are carried.

Main-band package data paths

These paths carry coherent or non-coherent functional data between chiplets inside the package. Cryptographic protection can be costly for high-bandwidth paths, so some products might rely on package topology, physical integration, or deployment assumptions instead.

These choices are residual-risk decisions. The assessment should identify the adversarial models that remain in scope.

Debug, test, and manufacturing paths

These paths include boundary scan, test access port (TAP)-controlled access, authenticated debug, mission-mode Design for Test (DFT), structural self-test, and mechanisms for bring-up, validation, and failure analysis. Different engineering teams might own these paths and apply different authorization models.

After secure boot, any chiplet-local transition into debug, test, scan, validation, reset, or another lower-assurance state constitutes a system-level security event if the chiplet participates in a bound inter-chiplet security domain.

A single chiplet can serve as the primary bridge to the external debug host and authenticate that host on behalf of the package. If the security policy of each chiplet permits delegation, other chiplets can delegate external debug-host authentication to the bridge chiplet. If a chiplet must authenticate the external debug host directly, authentication data between that chiplet and the debug host can transit through the bridge chiplet.

Memory-adjacent chiplet paths

High Bandwidth Memory (HBM), bonded low-power double data rate (LPDDR) memory, and other near-memory chiplet paths can be physically harder to access than traditional off-package memory, but physical accessibility is separate from security-domain binding.

Memory-adjacent chiplets still need analysis for logical misuse, reset or rollback behavior, stale encrypted-memory state, and incorrect binding of memory security state to a composed system identity.

When memory content requires confidentiality or integrity protection, an encryption or integrity endpoint that handles plaintext, keys, or integrity and replay metadata must be within the security domain that owns the content. A service admitted to and protected by that domain can also provide the endpoint. A memory PHY, interposer path, or memory device that handles only protected data can remain outside that domain if the product protects against replay and rollback.

Board-level and off-package paths

Inter-socket coherent traffic, board-level fabrics, and externally exposed trust extensions have a different physical exposure profile from package-contained links. Mitigations that are acceptable for package-contained trust extension might not be sufficient for board-level or off-package trust extension.

This assessment excludes the use of board-level or off-package paths to extend a security domain. However, it includes the security effects of an external debug, test, or manufacturing path that provides access to assets in an in-scope chiptlet or package security domain.

A product that uses a board-level or off-package path to extend a security domain still needs endpoint authentication, integrity and freshness protection, and confidentiality protection when required for the assets carried on the path.

9.1.7 Adversarial models

Adversarial models describe the adversaries that a designer must consider. This analysis defines the adversaries directly to keep the model self-contained. The definitions distinguish passive physical observation, active physical access, package-level invasive access, and supply-chain access.

The following adversarial models are used in this analysis:

AM.Remote

The adversary can access only data that does not require physical access to the system or the ability to run software on it. The adversary intercepts or provides data or requests through a network or other remote connection.

The adversary can, for example:

- Read system inputs and outputs by using external devices.
- Provide, forge, replay, or modify such inputs and outputs.
- Measure the timing of observable system operations, either during normal operation or in response to selected inputs, for example timing attacks on web servers. The measurement can reveal information about the internal state of the system.

AM.Software

The adversary can mount attacks from software running on the system.

In addition to **AM.Remote**, the adversary can, for example:

- Attempt software exploitation by running malicious software on the system.
- Exploit access to any memory-mapped configuration, monitoring infrastructure, or debug infrastructure of the system.
- Perform side-channel analysis that relies on software-exposed hardware features of the system.
- Perform software-induced glitching of resources with techniques including Rowhammer or RASpberry.

AM.PassivePhysical

The adversary has physical access to the system but can observe or measure only externally visible properties. The adversary does not attach hardware to the target, drive signals, modify the package, probe interposers, or make physically invasive modifications to chiptlets.

The adversary can, for example:

- Perform side-channel analysis that requires measurement of physical properties. Leakage sources include electromagnetic emissions, power consumption, photonic emissions, acoustic channels, and thermal leakage.
- Observe external interfaces, package-level emissions, or externally measurable timing without influencing the target operation.

AM.ActivePhysical

The adversary has physical access to the system and can perform active hardware attacks that do not require physically invasive access to chiptlet dies, package substrates, or interposers. These attacks can use reversible or irreversible modification outside the protected package boundary, externally accessible connectors, board-level interfaces, debug ports, or inserted components.

In addition to **AM.PassivePhysical**, the adversary can, for example:

- Connect malicious hardware to the system.
- Use externally reachable debug, test, service, or manufacturing interfaces.
- Gain access to the internal state of the system by interposing on wired connections between the system and external components, for example memory devices. The adversary might use this access to read, block, replay, or inject transactions.

AM.PackageInvasive

The adversary applies physically invasive techniques to the interconnections between chiplets, the package substrate, or the interposer.

In addition to **AM.ActivePhysical**, the adversary can, for example:

- Probe the packaging substrate to make contact with or disrupt physical communication channels between chiplets.
- Probe silicon or glass interposers to make contact with or disrupt physical communication channels between chiplets.

AM.ChipletInvasive

The adversary applies physically invasive techniques to the chiplets.

In addition to **AM.PackageInvasive**, the adversary can, for example:

- Apply chip decapsulation through laser or chemical etching followed by microphotography to reverse engineer the chiplet.
- Use a focused ion beam to perform gate-level modification.

AM.SupplyChain

The adversary has access to the supply chain that provides the component chiplets, chiplet-to-chiplet interposers, package redistribution layers (RDLs), and integration-time-provisioned firmware images assembled into the system. Individual chiplets, interposers, or firmware images can be substituted or modified.

For example, the adversary can:

- Interfere with the hardware provisioning process of chiplets to disrupt hardware-enforced security functionality.
- Substitute a tested and known-good chiplet with one that has failed testing.
- Substitute a chiplet with a counterfeit that can unexpectedly observe or disrupt system functions.
- Substitute a chiplet-to-chiplet interposer with a counterfeit that can unexpectedly observe or disrupt system functions.
- Substitute a firmware image with an unauthorized image that can unexpectedly observe or disrupt system functions.

Invasive silicon attacks (**AM.ChipletInvasive**) are not addressed in this analysis because they are equally applicable to a monolithic SoC.

Adversarial models **AM.Remote**, **AM.Software**, **AM.PassivePhysical**, **AM.ActivePhysical**, **AM.PackageInvasive**, and **AM.SupplyChain** are in scope for chiplet systems.

This analysis treats attacks against chiplet interposers (**AM.PackageInvasive**) as significantly less complex than attacks against the chiplet die (**AM.ChipletInvasive**). Interposer attacks are therefore in scope. For example, interfaces between elements on separate dies are easier targets where they cross an interposer.

For multi-vendor design and integration flows, the supply chain presents additional attack vectors (**AM.SupplyChain**) that might require more than supply-chain mitigation techniques and processes. These threats are relevant when considering system-level architectural security requirements.

9.1.8 Chiplet system security goals

The system-level security goals of a secure chiplet system are identical to those of a monolithic system. For a chiplet system, this model defines the following chiplet-specific security goals:

SG.Integrity: Integrity

Disaggregation of the system into chiplets does not enable an adversary to bypass the trust boundary of a bound inter-chiplet security domain.

SG.Confidentiality: Confidentiality

Disaggregation of the system into chiplets does not provide an adversary with easier access to assets within a bound inter-chiplet security domain.

SG.Authenticity: Authenticity

Disaggregation of the system into chiplets does not enable an adversary to substitute a chiplet, interposer, firmware image, manifest, or security-critical configuration without detection.

SG.VerifiableSystemState: Verifiable System State

The System RoT can derive and report a verifiable system security state from authenticated chiplet-local evidence. The reported state includes the intended system composition and the security-critical state of each chiplet whose behavior can affect a bound inter-chiplet security domain.

SG.CompositionalAssurance: Compositional Assurance

The system only forms or maintains a bound inter-chiplet security domain when the participating chiplets, interconnects, and provisioning state satisfy the system policy for that domain. A chiplet-local failure or downgrade that affects the domain is visible to the System RoT and has defined system-level consequences.

9.2 Threats and mitigations

9.2.1 Use model

The following analysis is intended to cover all security threats to chiplet-system assets. However, the threats and mitigations do not apply to all implementations in the same way. The chiplet or system designer must identify assets, interpret threats, and consider suggested mitigations in the context of the design. Design choices and the application domain affect mission-critical assets, threat applicability, risk, and the need for each mitigation.

The product mission, deployment environment, supply-chain model, and available product evidence can change asset criticality, threat applicability, Likelihood and Impact ratings, and residual Risk.

This threat analysis does not replace the requirements of an applicable product assurance or appraisal framework. A product assessed under OCP S.A.F.E. remains subject to the requirements of that framework. The product threat model should incorporate the chiplet-specific threats and assumptions identified in this analysis [16].

Product assumptions can modify the assessment only when supported by evidence for the product being assessed. For example, assumptions about controlled supply-chain practice, package-contained integration, operational facilities, exposed off-package paths, or serviceability can affect threat applicability, Likelihood, or Impact. The assessment must explain how the assumption is maintained and what security dependency it creates.

When using this analysis, the chiplet or system designer should:

- Adjust unmitigated Impact, Likelihood, and Risk ratings when justified by product evidence
- Select mitigations that are appropriate for the implementation constraints
- Adjust residual Impact, Likelihood, and Risk ratings when justified by product evidence, reflecting the selected mitigations and constraints

The mitigating actions are defined by the acronym *CAST*:

Control: Implement measures that reduce the likelihood, the impact, or both, and reduce the risk to an acceptable level.

Accept: Retain the threat because its risk is acceptable or because constraints or market needs prevent mitigation.

Suppress: Remove the feature or process that gives rise to the threat.

Transfer: Assign responsibility for mitigating the threat to a more capable or suitable party.

9.2.2 Risk assessment method

The risk ratings in this analysis use an approach derived from [19]. Each threat has a Likelihood and Impact that are evaluated on the 5-level scales in [Table 9.3](#) and [Table 9.4](#).

The ratings in this chapter are abstract-system defaults, not final product ratings. They assume that the affected chiplet, path, or asset is security-relevant to a bound inter-chiplet security domain. If a product-specific assessment shows narrower security scope, lower Impact values can be justified.

Residual Likelihood ratings are conservative for an abstract system. They assume correct implementation of named mitigations but do not include product-specific evidence. This evidence can include independent evaluation, tightly controlled supply-chain practices, formal verification of critical mechanisms, supplier security certification, or measured field performance. A product-specific assessment can justify a lower residual Likelihood when this evidence exists.

Table 9.3: Likelihood rating scale

Likelihood level	Definition
Very Low	An attacker is <i>highly unlikely</i> to attempt the attack, or the event is <i>highly unlikely</i> to occur. Adverse impacts are also <i>highly unlikely</i> .
Low	An attacker is <i>unlikely</i> to attempt the attack, the event is <i>unlikely</i> to occur, or adverse impacts are <i>unlikely</i> .
Medium	The event and adverse impacts are <i>somewhat likely</i> within the lifetime of the product. For example, a well-motivated and well-equipped adversary can cause the event.
High	The event and adverse impacts are <i>likely</i> within the lifetime of the product.
Very High	The event and adverse impacts are <i>almost certain</i> within a short time frame.

Table 9.4: Impact rating scale

Impact level	Definition	Example effects
Very Low	<i>Negligible</i> adverse impact.	Security behavior and functionality of the system are unaffected.
Low	<i>Limited</i> adverse impact on security functionality.	Degradation of secondary security functionality. Primary security features are unaffected.
Medium	<i>Serious</i> adverse impact on security functionality.	Compromise of a security domain and related security assets. Recovery and restoration of a secure system following the event are possible.
High	<i>Severe</i> adverse impact on system security.	Compromise of system-wide security domains and security assets, leading to exposure or exploitation of system assets. Recovery and restoration of a secure system following the event might be possible.
Very High	<i>Critical</i> adverse impact on system security.	Compromise of all system-wide security domains and critical security assets, leading to exposure or exploitation of all system assets and failure of security guarantees. It is not possible to recover or restore a secure system following the event.

Some mitigations reduce risk by limiting the consequence, duration, or scope of a successful attack instead of reducing Likelihood. Detection, attestation evidence, revalidation, revocation, degraded operation, recovery, or teardown can justify a lower residual Impact only under specific conditions. The product-specific policy must define the response, the product must validate the response, and the resulting state must be reported or otherwise available for assessment. Detection alone does not reduce Impact. Fail-closed or teardown behavior can preserve security by preventing reliance on an invalid trust decision, even if service is reduced until recovery or revalidation completes.

If the available evidence supports two possible Likelihood or Impact ratings, choose the higher rating.

The Likelihood and Impact ratings are combined using [Table 9.5](#) to determine the Overall Risk of a threat.

Table 9.5: Overall risk determination

Likelihood	Impact				
	Very Low	Low	Medium	High	Very High
Very Low	Very Low	Very Low	Very Low	Low	Low
Low	Very Low	Very Low	Low	Low	Medium
Medium	Very Low	Low	Medium	Medium	High
High	Low	Low	Medium	High	Very High
Very High	Low	Medium	High	Very High	Very High

9.2.3 Threat analysis

Threats to a chiplet system are analyzed in one table per threat. Threats already relevant to a monolithic SoC-based system, and not affected by chiplet disaggregation, are not reproduced. The threats are:

- [T.ChipletBackdoor](#): A vulnerability is built into a chiplet
- [T.SpoofChiplet](#): A fake or substituted chiplet
- [T.Misconfiguration](#): A misconfigured chiplet
- [T.CredentialLeakage](#): Disclosure of provisioning secrets
- [T.Lifecycle](#): Insecure security lifecycle state
- [T.InvasiveAccess](#): Unauthorized invasive access
- [T.DFXAbuse](#): Malicious use of debug, test, and manufacturing access paths
- [T.FirmwareBackdoor](#): Malicious chiplet firmware
- [T.SpoofFirmware](#): Substituted chiplet firmware
- [T.FirmwareAbuse](#): Vulnerable chiplet firmware
- [T.FirmwareTamperingPhysical](#): Firmware modification after loading from storage through physical attack
- [T.FirmwareTamperingLogical](#): Firmware modification after loading from storage through logical attack
- [T.InterposerBackdoor](#): A vulnerability built into the interposer
- [T.SideChannel](#): Side-channel leakage of die-to-die communication
- [T.InterposerLeakage](#): Leakage of die-to-die communication by physical probe
- [T.InterposerTampering](#): Manipulation of die-to-die communication by physical probe
- [T.D2DLeakage](#): Leakage of die-to-die communication through an untrusted system component
- [T.D2DTampering](#): Manipulation of die-to-die communication through an untrusted system component
- [T.RoTLeakage](#): Disclosure of chiplet RoT assets
- [T.RoTTampering](#): Modification of chiplet RoT assets

These threats reflect the primary objectives of an attacker using chiplet separation as an attack vector. In the topology models defined earlier, these objectives are:

1. Cause chiplet A to trust chiplet B with an inter-chiplet security domain while the attacker accesses domain assets through chiplet B.
2. Use the die-to-die connection between chiplets A and B to access or manipulate assets within the inter-chiplet security domain.
3. Manipulate or disclose assets required for mitigation to bypass the mitigation and achieve objective 1 or 2.

Each threat analysis lists possible mitigations and their effects. Implementation notes are provided in the [Implementation of mitigations](#) section.

Threat: T.ChipletBackdoor: A vulnerability is built into a chiplet.

Description	In the two-chiplet A+B model, the attacker inserts a malicious hardware element or a deliberate vulnerability into the design of chiplet B, or exploits an unintended vulnerability discovered in chiplet B. The attacker might be the chiplet designer, an insider within the chiplet designer or provider, an external actor that modifies the design before manufacture, or someone with knowledge of a chiplet design vulnerability. The trojan component or vulnerability can give the attacker unauthorized influence over the inter-chiplet security domain, exposing assets in that domain to disclosure, tampering, or misuse.		
Adversarial models	SupplyChain.		
Security goals	Integrity, Confidentiality, Authenticity, Compositional Assurance.		
Unmitigated ...	Impact: Very high.	Likelihood: Medium.	Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to prevent unauthorized modifications to design, configuration data, and firmware. If components are sourced externally, especially off-the-shelf components from a chiplet marketplace, the system vendor must ensure that all suppliers operate secure development processes. M.ControlledSupplyChain: Tightly Controlled Supply Chain If chiplet design and manufacturing can remain in-house or within a tightly controlled supply chain, controlled supply-chain integration can <i>Suppress</i> this risk, reducing it to a level similar to monolithic SoC development.		
Residual ...	Impact: Very high.	Likelihood: Very low.	Risk: Low.

Threat: T.SpoofChiplet: A fake or substituted chiplet.

Description	In the two-chiplet A+B model, the attacker substitutes chiplet B in the supply chain, causing the integration of an attacker-controlled chiplet into the system. The attacker might operate at any point after chiplet manufacture to system integration. To be successful, the substituted chiplet must provide enough compatibility with chiplet B for the system to appear to function correctly. The spoof component can give the attacker an unauthorized point of control or observation inside the inter-chiplet security domain.
Adversarial models	SupplyChain.
Security goals	Integrity, Confidentiality, Authenticity, Compositional Assurance.
Unmitigated ...	Impact: Very high. Likelihood: Medium. Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce access to assets required to clone a chiplet and reduce opportunities to introduce a fake chiplet during integration. M.ControlledSupplyChain: Tightly Controlled Supply Chain If chiplet design and manufacturing can remain in-house, controlled supply-chain integration can <i>Suppress</i> this risk by removing reliance on external entities in the chain of custody from chiplet manufacture to package integration. In a supply chain that extends beyond a tightly controlled integration flow, chiplet authentication becomes an architectural control: M.ChipletIdentity: Chiplet Identity <i>Control</i> by security architecture: provide an unforgeable, secure cryptographic identity for each chiplet. The chiplet can prove that it is authentic using this credential. M.SystemManifest: System Manifest <i>Control</i> by security architecture: provision the System RoT with a system manifest that lists all chiplet identities in the system. The integrator can verify chiplet identity before system integration. At integration or boot, the System RoT can verify that each chiplet corresponds to the provisioned manifest. Secure chiplet identity and a system manifest provide cryptographic inseparability for the integrated system, replacing the physical inseparability of a monolithic SoC for the chiplets and interfaces covered by the manifest and attestation model. This does not by itself prove the absence of unmanaged interposer functionality, unmeasured passive elements, or paths outside the covered topology.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.Misconfiguration: A misconfigured chiplet.	
Description	The attacker misconfigures a chiplet during chiplet or integration provisioning. This action can introduce an attacker-controlled identity or credentials or cause the package to operate in a vulnerable state. The misconfiguration can give the attacker unauthorized access to sensitive functionality, or allow a chiplet to participate in the inter-chiplet security domain with attacker-controlled identity, credentials, or policy.
Adversarial models	SupplyChain.
Security goals	Integrity, Confidentiality, Verifiable System State, Compositional Assurance.
Unmitigated ...	Impact: Very high. Likelihood: Medium. Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce attacker access to provisioning data and processes. M.ControlledSupplyChain: Tightly Controlled Supply Chain If chiplet design and manufacturing can remain in-house or within a tightly controlled supply chain, controlled supply-chain integration can <i>Suppress</i> this risk by reducing opportunities for mismatch between chiplet and system provisioning. In a more distributed supply chain, chiplet attestation becomes an architectural control: M.ChipletAttestation: Chiplet Attestation <i>Control</i> by security architecture: attest the identity, lifecycle state, critical configuration state, and mutable firmware measurements of each chiplet. This gives the System RoT and external verifiers evidence of the state and configuration of each chiplet.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.CredentialLeakage: Disclosure of provisioning secrets.

Description	The attacker observes the configuration or provisioning of a chiplet during chiplet or integration provisioning. The credentials can give the attacker unauthorized access to sensitive functionality, for example provisioning or debug access, or let an attacker-controlled component spoof the chiplet or system identity. Leakage of credentials for any chiplet with access to an inter-chiplet security domain can compromise that domain.
Adversarial models	SupplyChain.
Security goals	Integrity, Confidentiality, Authenticity, Verifiable System State.
Unmitigated ...	Impact: Very high. Likelihood: Medium. Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require the use of secure development processes to reduce attacker access to provisioning systems. M.SecureProvisioning: Secure Provisioning <i>Control</i> by security architecture: use secure provisioning protocols for integration provisioning. Building on M.ChipletIdentity, this can provide confidentiality, integrity, and authenticity for provisioned assets and the target chiplet. M.ChipletAttestation: Chiplet Attestation <i>Control</i> by security architecture: chiplet attestation lets the System RoT, and external verifiers, verify that authorized firmware is running in a secure operational state.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.Lifecycle: Insecure security lifecycle state.

Description	The attacker causes the lifecycle states of the chiplets in the system to be inconsistent. In the two-chiplet A+B model, the attacker causes the system to operate in a production state while chiplet B operates in an insecure lifecycle state. The attacker might manipulate chiplet or system provisioning. This can allow an attacker to access the inter-chiplet security domain through chiplet B while the rest of the system continues to treat chiplet B as production-secure. Chiplet systems amplify this threat because different chiplets can occupy different lifecycle states while contributing to the same bound inter-chiplet security domain.
Adversarial models	SupplyChain.
Security goals	Integrity, Confidentiality, Verifiable System State, Compositional Assurance.
Unmitigated ...	Impact: Very high. Likelihood: Medium. Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce attacker access to provisioning systems and opportunities to introduce a chiplet in the wrong lifecycle state. M.ControlledSupplyChain: Tightly Controlled Supply Chain If it is possible to keep chiplet design and manufacturing in-house, controlled supply-chain integration can <i>Suppress</i> this risk, removing the need to trust external entities in the chain of custody from chiplet manufacture to package integration. When chiplet manufacturing and integration are split across organizations, management and attestation of the per-chiplet security lifecycle become architectural controls: M.ChipletLifecycle: Chiplet Lifecycle <i>Control</i> by security architecture: implement a secure lifecycle state within each chiplet. M.ChipletAttestation: Chiplet Attestation <i>Control</i> by security architecture: report chiplet lifecycle state to the System RoT. This gives the System RoT, and external verifiers, evidence of each chiplet lifecycle state. During integration provisioning, it verifies that chiplets are in a valid state for integration. During operation, it enables the System RoT to prevent use of production assets when any participating chiplet is in an insecure lifecycle state.
Residual ...	Impact: Very high. Likelihood: Low. Risk: Medium.

Threat: T.InvasiveAccess: Unauthorized invasive access.

Description	In the two-chiplet A+B model, the attacker makes unauthorized use of invasive access mechanisms on chiplet B, for example JTAG debug, to access protected resources. This gives the attacker access to an inter-chiplet security domain through chiplet B and enables exploitation of assets within the domain. This threat is also present in a monolithic SoC. In a chiplet system, chiplet A security also depends on whether chiplet B has enabled invasive access since system boot. This threat covers direct unauthorized invasive access. Malicious use of debug, test, manufacturing, or other Design for X (DFX) mechanisms after secure boot is covered separately by T.DFXAbuse.		
Adversarial models	ActivePhysical.		
Security goals	Integrity, Confidentiality, Verifiable System State, Compositional Assurance.		
Unmitigated ...	Impact: Very high.	Likelihood: High.	Risk: Very high.
Mitigating actions (suggested)	M.DebugControl: Debug Control <i>Control</i> by security architecture: require each chiplet to securely manage all local invasive access mechanisms. M.ChipletAttestation: Chiplet Attestation <i>Control</i> by security architecture: the CRoT-A gates chiplet-local invasive access mechanisms and includes their state in the chiplet attestation report. The System RoT authorizes or constrains use of mechanisms that can affect a bound inter-chiplet security domain according to system policy. Together these enable the CRoT-A and System RoT to provide the local gating, reporting, control, and authorization for invasive access expected from the RoT in a secure monolithic SoC.		
Residual ...	Impact: Very high.	Likelihood: Very low.	Risk: Low.

Threat: T.DFXAbuse: Malicious use of debug, test, and manufacturing access paths.

Description	In the two-chiplet A+B model, the attacker maliciously uses a debug, test, manufacturing, or other Design for X (DFX) mechanism on chiplet B as the system enters or after it has entered a secure operational state. Relevant mechanisms include boundary scan, test access port (TAP)-controlled access, authenticated debug, mission-mode Design for Test (DFT), self-test, bring-up, validation, and failure-analysis hooks. These mechanisms can expose chiplet-internal assets, manipulate security-critical configuration, trigger chiplet-local reset, perturb measured state after secure boot, or move a chiplet into a lower-assurance state. If the chiplet participates in a bound inter-chiplet security domain, a post-boot transition into debug, test, scan, validation, reset, or a lower-assurance state is a system security event. The rest of the system must not silently continue to rely on the previous trust decision.		
Adversarial models	Software, ActivePhysical, SupplyChain.		
Security goals	Integrity, Confidentiality, Verifiable System State, Compositional Assurance.		
Unmitigated ...	Impact: Very high.	Likelihood: High.	Risk: Very high.
Mitigating actions (suggested)	M.DebugControl: Debug Control <i>Control</i> by security architecture: enumerate, authorize, gate, and report debug, test, scan, manufacturing, failure-analysis, and invasive access mechanisms that can affect a bound inter-chiplet security domain. M.ChipletLifecycle: Chiplet Lifecycle <i>Control</i> by security architecture: prevent production assets from being used when a chiplet is in a lifecycle state that permits insecure access. M.ChipletAttestation: Chiplet Attestation <i>Control</i> by security architecture: report the current state and enabled-since-boot state of these mechanisms to the System RoT. M.SystemSecurityPolicy: System Security Policy <i>Control</i> by security architecture: define the system-level response to chiplet absence, attestation failure, debug or DFX enablement, reset, recovery, or lower-assurance state transitions. Together these enable the System RoT to fail closed, recover, or report degraded state when the previous trust decision is no longer valid.		
Residual ...	Impact: Very high.	Likelihood: Low.	Risk: Medium.

Threat: T.FirmwareBackdoor: Malicious chiplet firmware.

Description	In the two-chiplet A+B model, the attacker inserts malicious code into the firmware for chiplet B. The attacker might be the chiplet firmware developer, an insider within the chiplet designer or provider, or an external actor that modifies the firmware before it is provisioned to the chiplet. The malware can give the attacker unauthorized influence over the inter-chiplet security domain and the assets protected by that domain. A monolithic SoC has a similar firmware-integrity risk. In a chiplet system, the security of chiplet A depends on the correctness of the chiplet B firmware.
Adversarial models	SupplyChain.
Security goals	Integrity, Confidentiality, Authenticity, Compositional Assurance.
Unmitigated ...	Impact: Very high. Likelihood: Medium. Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce attacker access to chiplet firmware development systems. M.ControlledSupplyChain: Tightly Controlled Supply Chain If chiplet design and manufacturing can remain in-house, controlled supply-chain integration can <i>Suppress</i> this risk, reducing it to a level similar to monolithic SoC development. M.AuthorizedUpdate: Authorized Update When more complex supply chains are used, <i>Transfer</i> this risk to firmware update policy: authorized update prevents chiplet firmware updates without system-level authorization. This gives the integrator or OEM a separate control point for blocking, detecting, and revoking a malicious firmware version.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.SpoofFirmware: Substituted chiplet firmware.

Description	In the two-chiplet A+B model, the attacker causes chiplet B to update to attacker-controlled firmware. The system is assumed to implement secure boot and secure firmware update, which are baseline requirements for a secure system. However, the attacker might access the firmware signing system of the chiplet provider or obtain the firmware signing keys. The substituted firmware can give the attacker unauthorized influence over the inter-chiplet security domain and the assets protected by that domain. A monolithic SoC has a similar firmware-authenticity risk. In a chiplet system, the security of chiplet A depends on the authenticity of the chiplet B firmware.
Adversarial models	SupplyChain, Software.
Security goals	Integrity, Confidentiality, Authenticity, Verifiable System State.
Unmitigated ...	Impact: Very high. Likelihood: Medium. Risk: High.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require the use of secure development processes to reduce attacker access to chiplet firmware signing systems. M.ControlledSupplyChain: Tightly Controlled Supply Chain If it is possible to keep chiplet design and manufacturing in-house, controlled supply-chain integration can <i>Suppress</i> this risk, reducing the threat to a level similar to monolithic SoC development. M.AuthorizedUpdate: Authorized Update <i>Transfer</i> to firmware update policy: authorized update prevents chiplet B firmware updates without system-level authorization. A valid chiplet-vendor signature is not sufficient. M.FirmwareMeasurement: Firmware Measurement <i>Control</i> by security architecture: record measurements of chiplet B firmware and configuration data. M.ChipletAttestation: Chiplet Attestation If measurement is implemented within chiplet B, use chiplet attestation to report the measurements. This gives the System RoT and external verifiers evidence of the firmware state of each chiplet. If measurement is implemented within chiplet A, ensure that the firmware cannot be modified before execution by chiplet B. See T.FirmwareTamperingPhysical and T.FirmwareTamperingLogical.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.FirmwareAbuse: Vulnerable chiplet firmware.

Description	In the two-chiplet A+B model, the attacker exploits a vulnerability in the chiplet B firmware. The exploit can give the attacker access to, or control over, the inter-chiplet security domain and the assets protected by that domain. A monolithic SoC has a similar firmware-vulnerability risk. In a chiplet system, the security of chiplet A depends on the correctness of the chiplet B firmware.
Adversarial models	Software.
Security goals	Integrity, Confidentiality, Verifiable System State.
Unmitigated ...	Impact: Very high. Likelihood: High. Risk: Very high.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce vulnerabilities in deployed firmware and improve security-incident response readiness. M.FirmwareMeasurement: Firmware Measurement <i>Control</i> by security architecture: record measurements of chiplet B firmware and configuration data. M.ChipletAttestation: Chiplet Attestation Use chiplet attestation to report on the measurements from chiplet B. This enables the System RoT to report whether chiplet B is running an authorized, non-revoked firmware baseline. This does not prove that the firmware is free of vulnerabilities.
Residual ...	Impact: Very high. Likelihood: Low. Risk: Medium.

Threat: T.FirmwareTamperingPhysical: Firmware modification after loading from storage through physical attack.

Description	In the two-chiplet A+B model, the attacker modifies firmware or a security-critical boot or load-time asset used by chiplet B while it is being transferred from chiplet A. This threat assumes that chiplet A has access to the interface for retrieving the asset, and that chiplet B does not. The attacker might conduct an active physical attack on the connection between the chiplets. The modification can give the attacker access to, or control over, the inter-chiplet security domain and the assets protected by that domain. Physical attacks against the boot or asset-authentication flow can also include fault injection or voltage glitching. Whether these techniques are in scope depends on the product, attacker access, and implementation. This abstract analysis assumes that such attacks do not bypass the authentication, measurement, or local enforcement mechanisms unless the product-specific assessment brings them into scope. This threat is also present in a monolithic SoC, where it is mitigated by measurement and authentication of firmware and other security-critical boot or load-time assets after loading, and by protection from later modification. In a chiplet system, transfer of these assets between chiplets provides a new attack surface.
Adversarial models	PackageInvasive.
Security goals	Integrity, Confidentiality, Authenticity, Compositional Assurance.
Unmitigated ...	Impact: Very high. Likelihood: High. Risk: Very high.
Mitigating actions (suggested)	M.SecureChipletBoot: Secure Chiplet Boot <i>Control</i> by security architecture: secure boot on chiplet B removes the attacker opportunity created when firmware and security-critical boot or load-time assets are transferred to chiplet B before authentication. <i>Note:</i> This mitigation is required where the threat applies to CROt-A mutable firmware. When chiplet B does not implement full M.SecureChipletBoot, the System RoT on chiplet A can measure and authenticate firmware or a security-critical boot or load-time asset for chiplet B only if chiplet B has a trusted local mechanism ensuring that the authenticated asset is used, or that changes trigger revalidation before use. The transfer interface needs protection from physical attack. M.ChipletInterface: Chiplet Interface <i>Suppress</i> by integration design: select a harder-to-attack die-to-die connection technology. If chiplet B has no protected local reference information or mechanism, authentication by chiplet A cannot close the time-of-check-to-time-of-use gap between verification and execution. Chiplet B then needs M.SecureChipletBoot or an equivalent minimal immutable loader. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide cryptographic integrity for firmware and security-critical boot or load-time asset transfer to chiplet B.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.FirmwareTamperingLogical: Firmware modification after loading from storage through logical attack.

Description	In the multi-chiplet A+B+E model, the attacker modifies firmware or a security-critical boot or load-time asset used by chiplet B while it is being transferred from chiplet A. This threat assumes that chiplet A has access to the interface for retrieving the asset, and that chiplet B does not. The attacker might execute software on an untrusted component with access to the communication interface, possibly another chiplet E. The modification can give the attacker access to, or control over, the inter-chiplet security domain and the assets protected by that domain. This threat is also present in a monolithic SoC, where it is mitigated by measurement and authentication of firmware and other security-critical boot or load-time assets after loading, and by protection from later modification. In a chiplet system, transfer of these assets between chiplets provides a new attack surface.
Adversarial models	Software.
Security goals	Integrity, Confidentiality, Authenticity, Compositional Assurance.
Unmitigated ...	Impact: Very high. Likelihood: High. Risk: Very high.
Mitigating actions (suggested)	M.SecureChipletBoot: Secure Chiplet Boot <i>Control</i> by security architecture: secure boot on chiplet B removes the attacker opportunity created when firmware and security-critical boot or load-time assets are transferred to chiplet B before authentication. Note: This mitigation is required where the threat applies to CRoT-A mutable firmware. When chiplet B does not implement full M.SecureChipletBoot, the System RoT on chiplet A can measure and authenticate firmware or a security-critical boot or load-time asset for chiplet B only if chiplet B has a trusted local mechanism ensuring that the authenticated asset is used, or that changes trigger revalidation before use. The transfer interface needs protection from logical attack. M.ChipletTopology: Chiplet Topology <i>Suppress</i> by system design: isolate the die-to-die connections between A and B from access by chiplet E. If chiplet B has no protected local reference information or mechanism, authentication by chiplet A cannot close the time-of-check-to-time-of-use gap between verification and execution. Chiplet B then needs M.SecureChipletBoot or an equivalent minimal immutable loader. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide cryptographic integrity for firmware and security-critical boot or load-time asset transfer to chiplet B.
Residual ...	Impact: Very high. Likelihood: Very low. Risk: Low.

Threat: T.InterposerBackdoor: A vulnerability built into the interposer.

Description	The attacker inserts a malicious hardware element or deliberate vulnerability into the interposer design. The attacker might be the interposer designer, an insider within the interposer designer or provider, or an external actor that modifies the design before manufacture. The trojan component or vulnerability can expose communication between chiplets to disclosure, tampering, injection, replay, or loss. The attacker can use that access to observe or influence assets within the inter-chiplet security domain. The unmitigated Likelihood is Low because an attacker with supply-chain access is expected to prefer substituting a chiplet, firmware image, or provisioning asset over modifying the interposer. The Low rating does not imply that a successfully modified interposer would be technically ineffective.		
Adversarial models	SupplyChain.		
Security goals	Integrity, Confidentiality, Compositional Assurance.		
Unmitigated ...	Impact: High.	Likelihood: Low.	Risk: Low.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce the risk of unauthorized modifications to the interposer design. M.ControlledSupplyChain: Tightly Controlled Supply Chain If interposer design and manufacturing can remain in-house or within a tightly controlled supply chain, controlled supply-chain integration can <i>Suppress</i> this risk, reducing it to a level similar to monolithic SoC development. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide an authenticated, encrypted, and integrity and replay-protected communication protocol between chiplets. This prevents a malicious interposer from disclosing assets transferred across the connection, or from tampering with communication between chiplets in a way that compromises the inter-chiplet security domain.		
Residual ...	Impact: High.	Likelihood: Very low.	Risk: Low.

Threat: T.SideChannel: Side-channel leakage of die-to-die communication.

Description	The attacker uses micro-probing techniques on the package to perform passive, non-invasive side-channel attacks and extract sensitive information exchanged between chiplets. A monolithic SoC has related side-channel risks. Connections between chiplets can be larger scale and more structured than buses within a die, making them easier targets for passive probing. Side-channel attacks might use techniques including differential power analysis, timing analysis, electromagnetic or photonic emissions measurement, acoustic analysis, and thermal measurement. These attacks can expose sensitive assets exchanged over the connection. Relevant side channels can arise from explicit die-to-die protocol interfaces, but also from shared power, clock, package, thermal, electromagnetic, memory-adjacent, or monitoring structures whose behavior depends on security-sensitive chiplet activity. Attack complexity differs by inter-chiplet path class. For example, main-band package data paths and security control-plane paths can have different widths, speeds, physical exposure, and mitigation costs. Risk and mitigation requirements should be considered separately for each path class.		
Adversarial models	PassivePhysical.		
Security goals	Confidentiality.		
Unmitigated ...	Impact: High.	Likelihood: Medium.	Risk: Medium.
Mitigating actions (suggested)	M.ChipletInterface: Chiplet Interface <i>Suppress</i> by integration design: select a harder-to-attack die-to-die connection technology. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide an encrypted communication protocol between chiplets.		
Residual ...	Impact: High.	Likelihood: Very low.	Risk: Low.

Threat: T.InterposerLeakage: Leakage of die-to-die communication by physical probe.

Description	In the two-chiplet A+B model, the attacker uses a probe on the interposer connection between chiplet A and chiplet B to disclose the communication between the chiplets. Sensitive assets carried over the connection can therefore be exposed. Attack complexity differs by inter-chiplet path class. For example, main-band package data paths and security control-plane paths can have different widths, speeds, physical exposure, and mitigation costs. Risk and mitigation requirements should be considered separately for each path class.		
Adversarial models	PackageInvasive.		
Security goals	Confidentiality.		
Unmitigated ...	Impact: High.	Likelihood: Medium.	Risk: High.
Mitigating actions (suggested)	M.ChipletInterface: Chiplet Interface <i>Suppress</i> by integration design: select a harder-to-attack die-to-die connection technology. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide an encrypted communication protocol between chiplet A and chiplet B.		
Residual ...	Impact: High.	Likelihood: Very low.	Risk: Low.

Threat: T.InterposerTampering: Manipulation of die-to-die communication by physical probe.

Description	In the two-chiplet A+B model, the attacker uses a probe on the interposer connection between chiplet A and chiplet B to manipulate the communication. The attack can tamper with, inject, replay, or drop communication between chiplets, giving the attacker access to, or control over, assets within the security domain. Attack complexity differs by inter-chiplet path class. For example, main-band package data paths and security control-plane paths can have different widths, speeds, physical exposure, and mitigation costs. Risk and mitigation requirements should be considered separately for each path class.		
Adversarial models	PackageInvasive.		
Security goals	Integrity, Confidentiality.		
Unmitigated ...	Impact: High.	Likelihood: Low.	Risk: Low.
Mitigating actions (suggested)	M.ChipletInterface: Chiplet Interface <i>Suppress</i> by integration design: select a harder-to-attack die-to-die connection technology. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide an authenticated, integrity-protected, and replay-protected protocol between chiplet A and chiplet B. This prevents an interposer probe from tampering with communication between chiplets in a way that compromises the inter-chiplet security domain.		
Residual ...	Impact: High.	Likelihood: Very low.	Risk: Low.

Threat: T.D2DLeakage: Leakage of die-to-die communication via an untrusted system component.

Description	In the multi-chiplet A+B+E model, the attacker uses an untrusted component in the system to disclose messages transported over the connection between chiplet A and chiplet B. This might be software outside the inter-chiplet security domain on chiplet A or B, or another chiplet E outside the security domain. Security-domain assets carried over the connection can be exposed. The connection topology can vary across inter-chiplet path classes. It is appropriate to consider the risk and mitigation requirements separately for each path class.		
Adversarial models	Software.		
Security goals	Confidentiality.		
Unmitigated ...	Impact: High.	Likelihood: High.	Risk: High.
Mitigating actions (suggested)	M.ChipletTopology: Chiplet Topology <i>Suppress</i> by system design: isolate the die-to-die connections between A and B from access by chiplet E. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide an end-to-end encrypted communication protocol between the portions of the inter-chiplet security domain on chiplets A and B. The cryptographic communication keys must be derived or established with a mechanism that chiplet E cannot replicate. Therefore, chiplet E cannot decrypt the traffic or act as a adversary-in-the-middle attacker.		
Residual ...	Impact: High.	Likelihood: Very low.	Risk: Low.

Threat: T.D2DTampering: Manipulation of die-to-die communication via an untrusted system component.

Description	In the multi-chiplet A+B+E model, the attacker uses an untrusted component in the system to manipulate messages transported over the connection between chiplet A and chiplet B. This might be software outside the inter-chiplet security domain on chiplet A or B, or another chiplet E outside the security domain. The attack can tamper with, inject, replay, or drop communication between chiplets, giving the attacker access to, or control over, assets within the security domain. The connection topology can vary across inter-chiplet path classes. It is appropriate to consider the risk and mitigation requirements separately for each path class.		
Adversarial models	Software.		
Security goals	Integrity, Confidentiality.		
Unmitigated ...	Impact: High.	Likelihood: High.	Risk: High.
Mitigating actions (suggested)	M.ChipletTopology: Chiplet Topology <i>Suppress</i> by system design: isolate the die-to-die connections between A and B from access by chiplet E. M.ProtectedConnection: Protected Connection <i>Control</i> by security architecture: use a protected connection to provide an end-to-end authenticated, integrity-protected, and replay-protected protocol between the portions of the inter-chiplet security domain on chiplets A and B. The cryptographic communication keys must be derived or established with a mechanism that chiplet E cannot replicate. Therefore, chiplet E cannot decrypt the traffic or act as a adversary-in-the-middle attacker.		
Residual ...	Impact: High.	Likelihood: Very low.	Risk: Low.

Threat: T.RoTLeakage: Disclosure of chiplet RoT assets.			
Description	The attacker extracts or obtains confidential or security-sensitive assets used by the CRoT-A or associated RoT roles. These assets can include chiplet identity secrets, attestation keys, endorsement state, firmware measurement or verification roots, lifecycle and debug-state data, protected-connection key material, and integration-provisioned binding assets used to authenticate the chiplet or CRoT-A to the System RoT. Disclosure of these assets can let the attacker spoof this chiplet or falsify chiplet data about security-critical state.		
Adversarial models	Software, PassivePhysical, ActivePhysical, SupplyChain.		
Security goals	Confidentiality.		
Unmitigated ...	Impact: Very high.	Likelihood: High.	Risk: Very high.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce attacker access to provisioning data and processes. M.ChipletRootOfTrust: CRoT-A <i>Control</i> by security architecture: use a CRoT-A to provide an intra-chiplet security domain containing these assets, the services that use them, and the processes for establishing them. Strong implementation assurance for the CRoT-A, side-channel resistance, limited privilege for mutable firmware, and protection against in-scope physical access can further reduce the residual likelihood in a product-specific assessment.		
Residual ...	Impact: Very high.	Likelihood: Very low.	Risk: Low.

Threat: T.RoTTampering: Modification of chiplet RoT assets.

Description	The attacker modifies security-sensitive assets used by the CRoT-A or associated RoT roles. These assets can include chiplet identity secrets, attestation keys, endorsement state, firmware measurement or verification roots, lifecycle and debug-state data, protected-connection key material, and integration-provisioned binding, role, topology, or manifest state used to authenticate the chiplet or CRoT-A to the System RoT. The attacker can use the modified assets to spoof another chiplet, falsify chiplet data about security-critical state, or run attacker-provided firmware on the chiplet.		
Adversarial models	Software, ActivePhysical, SupplyChain.		
Security goals	Integrity.		
Unmitigated ...	Impact: Very high.	Likelihood: High.	Risk: Very high.
Mitigating actions (suggested)	M.SecureDevelopment: Secure Development <i>Transfer</i> to the supply chain: require secure development processes to reduce attacker access to provisioning data and processes. M.ChipletRootOfTrust: CRoT-A <i>Control</i> by security architecture: use a CRoT-A to provide an intra-chiplet security domain containing these assets, the services that use them, and the processes for establishing them. Strong implementation assurance for the CRoT-A, immutable enforcement of asset access policy, and robust anti-rollback protection can further reduce the residual likelihood in a product-specific assessment.		
Residual ...	Impact: Very high.	Likelihood: Very low.	Risk: Low.

9.2.4 Implementation of mitigations

This section provides implementation notes for mitigations to threats to chiplet systems.

Several mitigation descriptions refer to the following external security patterns and guidance for context:

- National Institute of Standards and Technology (NIST) Secure Software Development Framework (SSDF) and supply-chain risk-management guidance [20, 21]
- Trusted Computing Group (TCG) Device Identifier Composition Engine (DICE) and Caliptra patterns for device identity and attestation keys [15, 22]
- Security Protocol and Data Model (SPDM) protocols from DMTF (formerly the Distributed Management Task Force) for component authentication and attestation [23]
- NIST platform firmware resiliency guidance [24]

M.SecureDevelopment: Secure Development

As with a secure monolithic SoC, all organizations involved in system development are assumed to operate secure design and development processes. These processes aim to prevent unauthorized modifications to design, configuration data, and firmware. Secure development expectations cover both software and hardware design artifacts. They should align with recognized secure-development and supply-chain risk-management practices.

When vulnerabilities are discovered in deployed products, suppliers should provide patched firmware for affected systems.

If components are sourced externally, for example off-the-shelf components from a chiplet marketplace, the system developer and integrator must ensure that all suppliers operate secure development processes.

M.ControlledSupplyChain: Tightly Controlled Supply Chain

Keep control of chiplet and interposer design and manufacturing in-house, or within a tightly coupled and trusted supply chain.

When possible, this approach reduces or eliminates many of the supply-chain risks that are introduced or amplified by the disaggregation of the system.

When a product relies on controlled supply-chain processes to control substitution or counterfeiting risk, product evidence can include provenance or chain-of-custody records. These records link the delivered chiplet, interposer, or package to the expected supplier and integration process.

M.ChipletIdentity: Chiplet Identity

Provide a secure cryptographic identity for each chiplet.

During chiplet provisioning, each chiplet securely generates or receives a unique cryptographic identity. This identity follows the same general pattern as secure device identity and attestation keys in Root of Trust designs, including DICE and Caliptra. The identity must resist forging or spoofing:

- It is based on a secret that is generated inside the chiplet and never exported, or provisioned through an authorized process that protects the secret from disclosure outside the provisioning trust boundary.
- It cannot be extracted from the chiplet.
- It cannot be derived from data disclosed by the chiplet.
- It cannot be guessed.

For example, the underlying secret can be derived from on-chip physically unclonable function (PUF) material, provisioned one-time-programmable state, or related hardware-rooted secret-embedding technology. The product design must provide the required stability, uniqueness, confidentiality, and lifecycle controls.

The chiplet provider certifies the chiplet identity public key and publishes the certificate or provides it to the system integrator. This identity can be the basis for chiplet authentication, attestation of chiplet state, and confidentiality, integrity, and authenticity protection for provisioning assets bound to the intended chiplet.

When provenance or chain-of-custody evidence is part of the product assessment, the chiplet identity can provide the cryptographic binding point between that evidence and the chiplet admitted into the system.

The chiplet identity is implemented in the CRoT-A.

This follows the pattern used for the secure cryptographic identity in a secure System RoT.

M.ChipletAttestation: Chiplet Attestation

Implement cryptographic attestation of the identity, lifecycle state, critical configuration state, current and historical debug and DFX state, and mutable firmware measurements of each chiplet. The evidence should identify trusted computing base (TCB) firmware and security-critical configuration measurements separately from lifecycle and other runtime security-state claims. This mitigation builds on **M.ChipletIdentity** to provide the System RoT with verifiable evidence for each chiplet. Protocols including SPDm provide useful context for component authentication, signed measurements, and protected sessions. However, a product-specific chiplet design still needs to define the exact evidence claims and policy bindings required for its system composition.

The chiplet attestation is provided by the CRoT-A.

This enables the System RoT to derive a system-level boot or initialization state and make valid decisions about system security state, preventing access to protected assets when the system is not in a secure operational state.

Chiplet attestation evidence should include a freshness mechanism, for example a nonce or session challenge, so that stale evidence cannot be replayed. Evidence should also support anti-rollback decisions for firmware, lifecycle state, manifests, and security-critical configuration. The attestation scheme needs a defined process for revoking or replacing chiplet identity keys, endorsement certificates, firmware signing keys, and manifests.

Encapsulating all chiplet attestations into a system attestation report from the System RoT enables external verification of whole-system composition and state. All security-relevant chiplets should be represented in this composed attestation model. If a product authenticates only a subset of control-plane chiplets but other chiplets can still affect confidentiality, integrity, isolation, or system correctness, the partial coverage is a residual risk that should be explicit.

M.ChipletLifecycle: Chiplet Lifecycle

Each chiplet securely manages its security lifecycle state and applies appropriate security policies for each lifecycle state. The chiplet can securely report its lifecycle state to the System RoT.

The chiplet security lifecycle is implemented in the CRoT-A.

Consistent lifecycle states across all chiplets reduce the risk of an insecure operational state at the system level. Use **M.ChipletAttestation** to provide a verifiable report of chiplet lifecycle state to the System RoT.

Lifecycle policy can control credential authorization, use, rotation, revocation, and destruction. When a credential change affects the composed system, the System RoT must revalidate the affected evidence and policy bindings before forming or maintaining the Security Platform.

M.DebugControl: Debug Control

Each chiplet securely manages all debug, test, scan, manufacturing, failure-analysis, and invasive access mechanisms on the chiplet. Each chiplet can determine the currently enabled and enabled-since-boot state of each mechanism. These mechanisms include mission-mode Design for Test (DFT) and self-test mechanisms that can perturb security-critical state.

Local gating and management of these access mechanisms on the chiplet is implemented in the CRoT-A.

The System RoT authorizes, constrains, or responds to enablement of access mechanisms that can affect a bound inter-chiplet security domain, based on attested chiplet-local state and system policy. The System RoT, the CRoT-As, or both should authenticate external entities, for example the debug host. In the production lifecycle state, systems should also lock external debug interfaces by default.

This extends the secure SoC system requirement for control and reporting of invasive access mechanisms. Use **M.ChipletAttestation** to provide a verifiable chiplet-state report to the System RoT.

M.SystemManifest: System Manifest

During integration of the system, use a manifest of the intended chiplets to bind them together.

Provision the System RoT with a manifest listing all chiplet identities in the package. The manifest can also bind chiplet roles, chiplet count, placement, topology, connectivity expectations, the chiplet that acts as the System RoT, and any runtime asymmetry in security roles:

- The System RoT can use this manifest to authenticate all other chiplets during integration provisioning.
- The System RoT uses this manifest to verify that the system is unchanged at each boot.
- This manifest must be protected against modification after provisioning.
- If this manifest contains secret information, for example shared symmetric keys for authenticating chiplets, it must also be protected against disclosure outside of the System RoT.

The secure **M.ChipletIdentity** mitigation provides cryptographic inseparability for the system. It replaces the physical inseparability of a monolithic SoC for the chiplets, roles, and paths covered by the manifest. The manifest does not cover components or effects that are not represented in the system model, including unmanaged interposer behavior, passive extra elements, or paths outside the declared topology. The manifest scheme should define anti-rollback and revocation behavior so that an older or withdrawn composition cannot be reintroduced silently.

Where a product uses provenance or chain-of-custody evidence, the manifest can bind that evidence to the authorized chiplet identities, roles, package placement, and integration state used by the System RoT.

M.SecureProvisioning: Secure Provisioning

Use authenticated protocols for integration provisioning flows.

For a monolithic SoC, OEM provisioning can use the SoC security credentials established during wafer provisioning by the manufacturer. In a chiplet system, integration provisioning protocols can use the credentials of each chiplet. Building on **M.ChipletIdentity**, this approach can provide confidentiality, integrity, and authenticity for provisioned assets and the target chiplet.

The chiplet identity secret, or a credential derived from it, can verify that a provisioning package is intended for the target chiplet, or decrypt or authenticate a provisioning package specific to that chiplet.

Provisioning evidence can also support provenance or chain-of-custody claims when the product assessment relies on those claims to control substitution, counterfeiting, or unauthorized rebinding risk.

Asset provisioning can use a suitable interface on the target chiplet or the System RoT. Provisioning through the System RoT can combine **M.SystemManifest** and **M.ChipletAttestation** to establish a protected connection between the System RoT and the secure provisioning firmware of the chiplet.

If field replacement, reprovisioning, or rebinding is supported, the provisioning protocol and system policy need to define authentication, authorization, rollback protection, and revocation behavior for the replacement event.

M.AuthorizedUpdate: Authorized Update

Firmware updates must be coordinated at the system level. A system-level authorization credential must authorize each chiplet firmware update. Chiplet-vendor firmware-verification keys must also authenticate the firmware. The update policy should define anti-rollback checks and revocation behavior for firmware signing keys, update manifests, and withdrawn firmware versions. Platform firmware resiliency guidance provides useful context for update, recovery, and rollback policy.

M.FirmwareMeasurement: Firmware Measurement

Make and securely store cryptographic measurements of all mutable firmware and security-critical boot or load-time configuration data for a chiplet. The data can include policy blobs, manifests, boot parameters, measurement inputs, and derived configuration that affects isolation, protected connections, lifecycle, or interconnect policy. Measurements should identify the version or digest of the measured object precisely enough for the System RoT to make rollback and revocation decisions.

The CRoT-A securely stores these measurements.

Use **M.ChipletAttestation** to provide a verifiable report of chiplet firmware and configuration measurements to the System RoT.

M.SecureChipletBoot: Secure Chiplet Boot

Use a secure boot mechanism on each chiplet for the firmware and security-critical boot or load-time assets that are used on that chiplet. Prefer local authentication by the chiplet that uses the asset. If authentication is delegated to another component, the receiving chiplet still needs a protected local basis for trusting that component and the communication path used to deliver the verified asset. The chiplet hardware should prevent firmware rollback to an earlier security version.

This approach requires the immutable post-provisioning state of the chiplet to contain the firmware or asset verification key, the firmware or asset hash, or the credential or secret needed to authenticate the supplier and protected delivery path. The chiplet should also persist the highest firmware security version number, for example in one-time-programmable memory.

If a product uses a dedicated firmware image to recover a chiplet or the system, the secure boot mechanism must authenticate that image and enforce the same rollback requirements that apply to other boot firmware. The recovery policy should define the authorization requirements, lifecycle transitions, and representation of the recovered state in chiplet and system attestation. Designating an image as known-good does not remove these requirements.

M.ProtectedConnection: Protected Connection

In the two-chiplet A+B model, use an authenticated, encrypted, integrity-protected, and replay-protected protocol between chiplet A and chiplet B when the path carries security-critical traffic or assets that require confidentiality.

This protocol prevents disclosure of assets transferred across the connection. It also prevents communication tampering that can compromise the inter-chiplet security domain. The protocol protects against supply-chain, physical-probing, and logical attacks.

The need for protection, and the level of protection, depends on the [inter-chiplet path class](#), the criticality of the connection, the likelihood of a successful attack, and the attacker objective. For example:

- A security control-plane path between the System RoT and a CRoT-A or Secondary Chiplet RoT is more critical to protect than an application data path. Exploitation of the security control-plane path would enable an attacker to control the security configuration of the chiplet and undermine the protection of any inter-chiplet security domain.
- A slower and narrower security control-plane path over I3C is easier to intercept or deterministically manipulate than a wide, high-speed coherent memory interface.
- Main-band package data paths, off-package coherent paths, and memory-adjacent chiplet paths can require different protection choices because they have different performance costs and physical exposure.

M.ChipletIdentity and **M.ChipletAttestation** can form part of a protocol that establishes key-derivation keys from a boot-time chiplet attestation flow.

If a product relies on topology, packaging, or deployment assumptions instead of cryptographic protection for a path, the resulting residual risk should be stated against the relevant adversarial models.

M.ChipletInterface: Chiplet Interface

Use an attack-resistant die-to-die connection mechanism.

Different physical connection techniques present different levels of difficulty to an attacker with physical access:

- 2D package-substrate or redistribution-layer connections generally use the largest physical scale and can be easier to attack than other die-to-die connections.
- 2.5D silicon or glass interposers, bridges, and related package-level routing approaches generally use smaller features and shorter exposed paths than 2D substrate routing. These characteristics make a successful attack more difficult.
- Vertically stacked 3D die-to-die interfaces, including direct-bonded or hybrid-bonded interfaces, can be difficult to attack because the connections are dense, short, and internal to the stack.
- Edge-to-edge, bridge-based, and 3.5D-style integrations have physical exposure and attack difficulty that depend on the routing medium, feature size, package layer, and inspection or probing access. Assess these integrations separately from vertically stacked 3D interfaces.

M.ChipletTopology: Chiplet Topology

In the multi-chiplet A+B+E model, use isolated die-to-die connections between chiplets A and B. These connections do not pass through or connect to chiplet E. For example, avoid a daisy-chain topology in which the connection must pass through chiplet E.

This prevents chiplet E from eavesdropping on the connection or manipulating traffic.

M.SystemSecurityPolicy: System Security Policy

The System RoT defines the policy for forming, maintaining, degrading, and ending the Security Platform. This policy covers chiplet absence, attestation failure, stale or revoked evidence, lifecycle mismatch, debug or DFX enablement, firmware rollback, unexpected reset or recovery of a participating chiplet, and any product-defined transition to an RMA diagnostic or disposition state.

The default high-assurance behavior is to fail closed before trust-domain formation. If degraded operation is supported, the system must isolate the affected chiplet or path, prevent continued use of keys and claims that assumed the previous security state, and report the degraded state in system attestation.

Before policy authorizes diagnostic, failure-analysis, or debug access in an RMA state, it must revoke or make inaccessible data, credentials, and security bindings associated with the previous owner or user. The policy must report the resulting state. Protected manufacturer-root assets can remain available only if product policy requires them for authenticated return handling or a separately authorized reprovisioning process. They must not enable recovery of the prior owner context.

M.ChipletRootOfTrust: Chiplet RoT for Attestation (CRoT-A)

Implement an intra-chiplet security domain to protect the implementation of the **M.ChipletIdentity**, **M.ChipletAttestation**, **M.ChipletLifecycle**, **M.DebugControl**, **M.SecureChipletBoot**, and **M.ProtectedConnection** mitigations within each chiplet that participates in attested composition assurance.

This security domain provides confidentiality and integrity protection for the secrets, keys, and other data assets required for these mitigations. The security domain implements the services that provision and use these assets to provide the mitigations.

The CRoT-A enforces the asset access policy for these services, including protection against rollback or replacement of security-critical state where that state affects chiplet identity, attestation, lifecycle, debug control, secure boot, or protected-connection keys.

This domain within a chiplet is the Chiplet RoT for Attestation (CRoT-A). The CRoT-A acts as the minimum evidence root for reporting the security-critical state of the chiplet to the System RoT.

The implementation that contains the CRoT-A can also provide System RoT or Secondary Chiplet RoT services. Alternatively, these roles can extend the CRoT-A trust domain. They can also be independent RoTs whose identity, firmware, lifecycle, configuration, and security state are measured or attested by the CRoT-A. These additional roles can provide trusted services for the chiplet or give the System RoT control over chiplet security configuration. They remain distinct from the minimum CRoT-A role.

Glossary

AMBA

Advanced Microprocessor Bus Architecture - Arm Interconnect and related standards.

ATS

(PCIe) Address Translation Services, see [6].

AXI

Advanced eXtensible Interface, part of the AMBA family of protocols, see [25].

BoW

Bunch of Wires, an OCP PHY, transaction, and link-layer specification for chip and chiplet communications.

C2C

(AMBA) Chip-to-Chip Interface - for transporting AMBA protocols between chiplets. For more information, see [9].

CHI

(AMBA) Coherent Hub Interface, for more information see [4].

CXL

Compute Express Link, see [26].

DVM

Distributed Virtual Memory.

HES

Hardware Enforced Security.

I3C

Improved Inter-Integrated Circuit (Interface), see [8].

IEEE

Institute of Electrical and Electronics Engineers.

JTAG

Joint Test Action Group.

MHU

Message Handling Unit.

MMU

Memory Management Unit.

MSI

Message Signaled Interrupt.

OCP

Open Compute Project.

PA	Physical Address, an address that identifies a location in the physical memory map.
PCIe	Peripheral Component Interconnect Express, see [6].
PE	Processing Element, typically a CPU.
RDL	Redistribution Layer.
RFQ	Request for Quote.
RoT	Root of Trust.
RPMI	RISC-V Platform Management Interface Specification [11].
SCMI	System Control and Management Interface, see [10].
SoC	System-on-Chip.
TAP	Test Access Port, part of the JTAG / IEEE Std 1149.1™-2001 standard, see [27].
TCK	Test Clock, a standard TAP interface signal.
TDI	Test Data In, a standard TAP interface signal.
TDO	Test Data Out, a standard TAP interface signal.
TMS	Test Mode Select, a standard JTAG interface signal.
TO BE DEFINED	A feature for which there is an intent to define the implementation in a future version of the FCSA.
UCIe	Universal Chiplet Interconnect Express. For more information, see [7].