



OPEN

Compute Project

Open Boot and Management Framework Interface Consolidation Protocol (OBMF-ICP)

Version 1.0.0
October 7th, 2026

Editors:

Janusz Jurski – Intel Corporation

Kasper Wszolek – NVIDIA Corporation

Contributors:

NVIDIA Corporation: Mariusz Oriol, Mateusz Duchalski, Donald Shannon, Rouslan Dimitrov, Adrian Ambrozewicz, Leticia Sakata

Intel Corporation: Matthew Schnoor, Yi Zeng

Microsoft Corporation: Bharat Pillilli

Dell Technologies: Tim Lambert

Arm Limited: Samer El-Haj-Mahmoud, Stuart Yoder

Google: Kasun Athukorala, Rom Lemarchand, Ofer Yehielli

Code Construct: Jeremy Kerr, Andrew Jeffery

AMI: Rama Bisa

Renesas Electronics America Inc.: Michael Joehren

Marvell: Satananda Burla

AMD: Supreeth Venkatesh, Mahesh Kurapati

NXP: Priyanka Jain

Version History

Date	Version #	Author	Description
7 OCT 2026	1.0.0	Kasper Wszolek Janusz Jurski Ofer Yehielli Mariusz Oriol	Initial version

Table of Contents

1. License	8
1.1. Open Web Foundation (OWF) CLA	8
2. Compliance with OCP Tenets	10
2.1. Openness	10
2.2. Efficiency	10
2.3. Impact	10
2.4. Scale	10
2.5. Sustainability	10
3. OBMF-ICP Introduction	11
3.1. Goals	11
3.2. Requirements	11
3.3. Assumptions	13
3.4. Trademarks	14
3.5. References	14
3.6. Linux and OpenBMC Interoperability	15
3.7. Acronyms	15
4. OBMF-ICP Architecture Overview	16
4.1. OBMF-ICP Protocol Stack Overview	16
4.2. OBMF-ICP Component Roles	17
4.3. Typical Use Case Example	20
5. OBMF-ICP Protocol Definition	23
5.1. OBMF-ICP Message Format	25
5.2. OBMF-ICP Status Codes	26
5.3. OBMF MMIO Channel	27
5.3.1. Short Read Request	28
5.3.2. Short Read Response	28
5.3.3. Short Write Request	28
5.3.4. Short Write Response	28
5.3.5. Long Read Request	29
5.3.6. Long Read Response	29
5.3.7. Long Write Request	29
5.3.8. Long Write Response	29
5.4. OBMF GPIO Channel	30
5.4.1. GPIO Channel Request	31
5.4.2. GPIO Channel Response	32
5.5. OBMF Serial Channel	33
5.5.1. Serial Channel Request	35
5.5.2. Serial Channel Response	35
5.6. OBMF I2C Controller Channel	37
5.6.1. I2C Controller Channel Request	38
5.6.2. I2C Controller Channel Response	39
5.7. OBMF I2C Target Channel	40
5.7.1. I2C Target Channel Request	42
5.7.2. I2C Target Channel Response	42

5.8.	OBFM I3C Controller Channel	42
5.8.1.	I3C Controller Channel Request	44
5.8.2.	I3C Controller Channel Response	46
5.9.	OBFM IPMI Channel	47
5.9.1.	IPMI Channel Request	48
5.9.2.	IPMI Channel Response	49
5.10.	OBFM SPI Controller Channel	49
5.10.1.	SPI Controller Channel Request	50
5.10.2.	SPI Controller Channel Response	51
5.11.	OBFM I/O Channel	52
5.11.1.	Sequential Port I/O Read Transaction 8-bit	53
5.11.2.	Sequential Port I/O Read 8-bit Request	53
5.11.3.	Sequential Port I/O Read 8-bit Response	53
5.11.4.	Sequential Port I/O Port Write 8-bit Request	53
5.11.5.	Sequential Port I/O Port Write 8-bit Response	54
5.11.6.	Sequential Port I/O Read Transaction 16-bit	54
5.11.7.	Sequential Port I/O Write Transaction 16-bit	54
5.11.8.	Sequential Port I/O Read Transaction 32-bit	54
5.11.9.	Sequential Port I/O Write Transaction 32-bit	54
5.11.10.	Fixed Port I/O Read Transaction 8-bit	54
5.11.11.	Fixed Port I/O Write Transaction 8-bit	55
5.11.12.	Fixed Port I/O Read Transaction 16-bit	55
5.11.13.	Fixed Port I/O Write Transaction 16-bit	55
5.11.14.	Fixed Port I/O Read Transaction 32-bit	55
5.11.15.	Fixed Port I/O Write Transaction 32-bit	55
5.12.	OBFM Memory Device Channel	55
5.12.1.	Memory Device Channel Request	57
5.12.2.	Memory Device Channel Response	58
5.13.	OBFM TPM Channel	58
5.14.	OBFM RTC Channel	58
5.15.	OBFM-ICP Control Channel – Channel 0	58
5.15.1.	Channel Specific Status and Control Registers	66
5.15.2.	Channel Configuration Data	69
6.	Security Considerations.....	79
6.1.	Aggregation of Legacy Interfaces and Physical Protection	79
6.2.	Leveraging Existing Security Layers	80
6.3.	Channel-Specific Security Protections	80
6.4.	Device Authentication and Origin Validation (Informative)	80
6.5.	Channel Isolation	80
6.6.	Transport Guarantees	81

Table of Figures

Figure 1: OBMF-ICP Protocol Stack.....	16
Figure 2: Specifications related to OBMF-ICP	17
Figure 3: Typical OBMF-ICP Host and OBMF-ICP Device Roles	18
Figure 4: Concept of OBMF-ICP Channels and Data Structures	19
Figure 5: Sample Request and Response Transaction Uses	20
Figure 6 Example of use cases for various OBMF Channel Types	21
Figure 7 Example of OBMF Channels Discovery and Configuration	23
Figure 8 Multiple OBMF-ICP Channel Types	24
Figure 9 OBMF Runtime and Manufacturing Configuration support	25
Figure 10 Example of GPIO channel use case with OBMF Bridge.	30
Figure 11 GPIO Persistent State across Reset	31
Figure 12 Example of Serial channel use case with OBMF Bridge.	34
Figure 13 Example Serial Channel flow diagram	36
Figure 14 Example of I2C Controller channel use case with OBMF Bridge.	37
Figure 15 Example of I2C Target channel use case with OBMF Bridge.	40
Figure 16 Conceptual view of how I2C Writes are handled by the OBMF Target Channel.	41
Figure 17 Example of I3C Controller channel use case with OBMF Bridge.	43
Figure 18 Example of IPMI channel use case with OBMF Bridge.	48
Figure 19 Example of SPI channel use case with OBMF Bridge.	50
Figure 20 Example of Host SoC Boot and Configuration Storage access	56
Figure 21 Example of Generic Memory Device access	56
Figure 22 OBMF Manufacturing and Runtime Configuration Flow	61

List of Tables

Table 1: Key Requirements of the OBMF-ICP Protocol	11
Table 2: Acronyms	15
Table 3: OBMF-ICP Message Format	26
Table 4: Generic Status Codes	26
Table 5: OBMF MMIO Channel	27
Table 6: MMIO Channel Roles	27
Table 7: MMIO Channel-specific Status Codes	27
Table 8: Short Read Request	28
Table 9: Short Read Response	28
Table 10: Short Write Request	28
Table 11: Short Write Response	28
Table 12: Long Read Request	29
Table 13: Long Read Response	29
Table 14: Long Write Request	29
Table 15: Long Write Response	29
Table 16: GPIO Channel Roles	30
Table 17: GPIO Channel-specific Status Codes	30
Table 18: OBMF-ICP GPIO Channel Request message	31
Table 19: OBMF-ICP Get GPIO Values Channel Response message	33
Table 20: Serial Channel Roles	34
Table 21: Serial Channel-specific Status Codes	34
Table 22: OBMF-ICP Serial Channel Request message	35
Table 23: OBMF-ICP Serial Channel Response message	35
Table 24: I2C Controller Channel Roles	37
Table 25: I2C Controller Channel-specific Status Codes	37
Table 26: I2C Controller Channel Request	38
Table 27: SMBus Operations Encoding Using OBMF I2C Controller Commands	38
Table 28: I2C Controller Channel Response	39
Table 29: I2C Target Channel Roles	41
Table 30: I2C Target Channel-specific Status Codes	41
Table 31: I2C Target Channel Request	42
Table 32: I2C Target Channel Response	42
Table 33: I3C Controller Channel Roles	43
Table 34: I3C Channel-specific Status Codes	43
Table 35: OBMF-ICP I3C Channel Request message	44
Table 36: I3C_CMD_EXECUTE_SEQUENCE Operations Array	45
Table 37: I3C_TRANSFER_CCC	45
Table 38: I3C_TRANSFER_CCC_W_DEFINING_BYTE	45
Table 39: I3C_TRANSFER_PRIVATE_SDR	45
Table 40: OBMF-ICP I3C Channel Response message	46
Table 41: Operation Result Descriptor Structure (for Execute Sequence)	46
Table 42: Discovered Device Structure for DAA	47
Table 43: Counter Entry Structure (for Get Stats)	47
Table 44: IPMI Channel Roles	48
Table 45: IPMI Channel-specific Status Codes	48
Table 46: OBMF-ICP IPMI Channel Request message	49
Table 47: OBMF-ICP IPMI Channel Response message	49
Table 48: SPI Channel Roles	49
Table 49: SPI Channel-specific Status Codes	49
Table 50: OBMF-ICP SPI Channel Request message	51

Table 51: OBMF-ICP SPI Channel Response message	51
Table 52: OBMF I/O Channel	52
Table 53: IO Channel Roles	52
Table 54: IO Channel-specific Status Codes	52
Table 55: OBMF-ICP I/O Read 8-bit Request message	53
Table 56: OBMF-ICP I/O Read 8-bit Response message	53
Table 57: OBMF-ICP I/O Port Write 8-bit Request message	53
Table 58: OBMF-ICP I/O Port Write 8-bit Response message	54
Table 59: Memory Device Channel Roles	57
Table 60: Memory Device Channel-specific Status Codes	57
Table 61: OBMF-ICP Memory Device Channel Request message	57
Table 62: OBMF-ICP Memory Device Channel Response message	58
Table 63: Channel 0 Tables Convention	61
Table 64: Channel 0 MMIO Register Table	62
Table 65: Channel Configuration Space Common Header	63
Table 66: MMIO Channel Control and Status Registers	66
Table 67: GPIO Channel Control and Status Registers	66
Table 68: Serial Channel Control and Status Registers	66
Table 69: I2C Controller Channel Control and Status Registers	67
Table 70: I2C Target Channel Control and Status Registers	68
Table 71: I3C Controller Channel Control and Status Registers	68
Table 72: IPMI Channel Control and Status Registers	68
Table 73: SPI Controller Channel Control and Status Registers	69
Table 74: Memory Device Channel Control and Status Registers	69
Table 75: MMIO Channel Configuration Data	69
Table 76: GPIO Channel Configuration Data	70
Table 77: GPIO Configuration Entry	70
Table 78: Serial Channel Configuration Data	71
Table 79: I2C Controller Channel Configuration Data	72
Table 80: I2C Target Channel Configuration Data	72
Table 81: I3C Controller Channel Configuration Data	73
Table 82: IBI_TARGET_CONFIG_ARRAY	73
Table 83: IPMI Channel Configuration Data	74
Table 84: SPI Controller Channel Configuration Data	74
Table 85: IO Channel Configuration Data	77
Table 86: I/O RANGE_CFG	78
Table 87: MEM_DEV_CFG	78

1. License

THE UPDATED DEFAULT CONTRIBUTOR LICENSE AGREEMENT (CLA) IS [OWFa 0.9](#). PLEASE VERIFY THE CORRECT CLA/FSA IS USED AND EXECUTED FOR THIS CONTRIBUTION.

1.1. 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:

- Advanced Micro Devices
- AMI
- Arm Limited
- ASPEED Technology, Inc.
- Code Construct
- Dell Technologies
- Google Inc.
- Hewlett Packard Enterprise
- Intel Corporation
- Marvell Technology
- Microsoft Corporation
- Nuvoton Technology Israel Ltd.
- NVIDIA
- NXP
- Renesas Electronics America Inc.

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.

2. Compliance with OCP Tenets

2.1. Openness

OBMF-ICP promotes an open, vendor-agnostic protocol for communication between host processors, BMCs, and other platform devices. The specification enables broad implementation across Host SoC vendors, DC-SCM designs, PCIe/CXL cards, GPUs, NICs and other devices while leveraging existing OCP and DMTF standards.

2.2. Efficiency

OBMF-ICP improves platform efficiency by consolidating multiple management and control interfaces over a common protocol and, where applicable, a single physical connection like OBMF over USB Binding. It reduces interface complexity for use cases such as GPIO, I2C, TPM, RTC, UART, SPI, SSIF/IPMI, and other management functions.

2.3. Impact

OBMF-ICP enables industry-wide interoperability by standardizing boot, configuration, and runtime communication between host-side devices and management controllers. Its impact comes from reducing fragmentation, supporting reusable implementations, and enabling adoption across DC-SCM and non-DC-SCM platforms.

2.4. Scale

OBMF-ICP is designed for scalable deployment across diverse platform architectures, including highly integrated AI platforms. By providing a single, robust, and scalable interface that aggregates legacy interfaces, OBMF-ICP reduces platform complexity and accelerates AI hardware scalability.

2.5. Sustainability

OBMF-ICP supports sustainable platform design by reducing the number of dedicated interfaces and enabling easier module reuse. By leveraging existing OCP, DMTF, USB standards, OBMF-ICP reduces integration, maintenance and reuse overhead, avoids duplicated solutions, and improves long-term maintainability.

3. OBMF-ICP Introduction

3.1. Goals

OBMF-ICP specification has been driven by the need to standardize the protocol for use during boot, configuration, and runtime communication between Host Processor or other devices on the Host Processor Module (HPM) and Baseboard Management Controller (BMC) on the DC-SCM Module. But OBMF-ICP is not tied to DC-SCM and can be used in other circumstances when consolidation of other interfaces is necessary.

OBMF-ICP protocol shall meet the following goals:

- Host Processor (Host SoC) vendor agnostic
- Applicable to non-DC-SCM use cases
- Reusable for other devices e.g. PCIe Cards, CXL Cards, etc.
- Should be physical interface agnostic protocol (e.g., MCTP/PLDM)
- Recommended physical interface choices for DC-SCM use cases
- Supporting adoption model for vendor specific features (vendor extensions)
- Leveraging OCP standards & DMTF standards
- Align to existing security standards and requirements (no new OBMF-ICP security solutions)

OBMF-ICP allows establishing multiple communication channels, each of them transporting a different use case over a single physical connection. Use case examples are: GPIO, I2C, TPM, RTC, UART, SPI, SSIF/IPMI, BIOS POST code reporting and others. OBMF-ICP can be used to consolidate these various types of traffic.

Cryptographic content protection, such as encryption, is out of scope of OBMF-ICP specification. Every protocol that requires protection, such as TPM, should provide their own protections against the targeted threats (such as man-in-the-middle attacks) or use pre-existing mechanisms, such as wrapping the OBMF-ICP communication in a SPDH session.

3.2. Requirements

The following are the key requirements of the OBMF-ICP protocol:

Table 1: Key Requirements of the OBMF-ICP Protocol

No.	Title	Description
1	Vendor agnostic OBMF-ICP definition	Any implementation of OBMF-ICP Device or OBMF-ICP Host shall operate correctly with any vendor's implementation without vendor-specific protocol adaptations, ensuring compatibility across different processor architectures.

No.	Title	Description
2	Backward-compatible support for OBMF-ICP Devices	OBMF-ICP Host is responsible for OBMF-ICP specification version discovery and shall implement backward-compatible driver to the specific OBMF-ICP. If the OBMF-ICP version is not supported, OBMF-ICP Host shall reject such a device.
3	Applicable to non-DC-SCM use cases	OBMF-ICP shall function correctly when deployed in a non-DC-SCM system design, without requiring DC-SCM-specific hardware.
4	Reusable for other devices	OBMF-ICP shall be applicable across multiple device types e.g. GPUs, PCIe Cards, CXL Cards, etc., ensuring consistency and minimizing redundant implementations.
5	Physical interface agnostic protocol	OBMF-ICP shall define a protocol that is independent of the underlying physical interface.
6	Transport layer agnostic	OBMF-ICP protocol shall be defined independently of the transport layer, with specific bindings defined for each supported transport (e.g., MCTP via DMTF specifications).
7	Recommended physical interface choices for DC-SCM	OBMF-ICP should provide recommended physical interface options specifically tailored for DC-SCM implementations (e.g., USB, I3C), while maintaining broader interface agnosticism.
8	Supporting adoption model for vendor specific features (vendor extensions)	OBMF-ICP shall support a structured adoption model for vendor-specific extensions, allowing proprietary features to coexist with standard functionality.
9	Leveraging OCP standards & DMTF standards	OBMF-ICP shall leverage existing OCP and DMTF standards, with clearly defined dependencies and integration points to ensure alignment and interoperability.
10	Align to existing security standards and requirements	OBMF-ICP shall align with existing industry security standards, such as SPDH, to ensure secure communication and system integrity.
11	Support Multi-Node and Multi-Partitions architectures	OBMF-ICP shall support multi-node and multi-partition system architectures in alignment with DC-SCM definitions of Multi-Node and Multi-Partitions.
12	Request-response model	OBMF-ICP shall employ a request-response communication model to ensure synchronous interaction between applications and components.
13	Asynchronous notification model	Specific OBMF-ICP channel definitions MAY implement channel-specific asynchronous notification model that will not require any response e.g. GPIO Channel.
14	Multiple communication channels between different applications	OBMF-ICP shall support multiple independent communication channels between different applications operating on the same endpoint. Delays in one application shall not block communications in other channels. Application/channels examples: - TPM - UART - GPIO
15	One outstanding request per channel	OBMF-ICP shall allow only one outstanding request per communication channel to simplify message management and ensure predictable behavior.
16	Low complexity allowing HW-offload options.	OBMF-ICP shall be designed with low complexity to enable hardware offload and efficient implementation in constrained environments.
17	Prioritization of channels	OBMF-ICP shall allow prioritization of communication channels. The enforcement of prioritization policies by OBMF-ICP Device is implementation-specific.

No.	Title	Description
18	Latency constraints and mitigations for time-sensitive channels	OBMF-ICP does not define timing and latency requirements for time sensitive protocols like UART and TPM. It is system integrator responsibility to select the physical media that will be able to meet the protocol constraints.
19	No changes required to Host BIOS/OS/Drivers	OBMF-ICP shall not require modifications to host BIOS, operating systems, or drivers for interfaces it consolidates, ensuring seamless integration. Note: OBMF use models allow for bridging to native interfaces through discrete or embedded bridge in the Host SoC.
20	Leveraging existing security standards	OBMF-ICP shall not define native security solutions. Existing solutions such as SPDH shall be used to provide secure channels when needed.
21	Leveraging existing security protection for TPM traffic	If secure communication is required for the TPM channel, OBMF-ICP shall rely on TCG TPM Communication over SPDH Secure Session to ensure traffic protection.
22	Leveraging existing device driver models	OBMF-ICP specification should align with existing OpenBMC and Linux device driver models to maximize legacy software stack reuse.
23	Predefined standard channels	OBMF-ICP specification should define a list of standardized channels such as UART, GPIO, I2C etc. and allow for extensions with vendor-specific channels.
24	Channel presence enforcement by Device	If the OBMF-ICP Device needs to consume a channel, it shall indicate this on a per-channel basis using the DEVICE_ROLE capability in the Channel 0 discovery structure.
25	OBMF-ICP supports up to 64kB message size	OBMF-ICP shall allow exchanging application messages of size up to 64 kB.
26	Device-specific message size	OBMF-ICP Device may limit the maximum supported message size using transports-specific method to report supported message size.
27	Device always exposes Channel 0	OBMF-ICP Device shall always expose the Control Channel 0 discovery structure.
28	Host shall not expose Channel 0	The OBMF-ICP Host (BMC) shall not expose Channel 0.
29	Mandatory MMIO Short Read/Write for Channel 0 discovery	Channel 0 shall always be accessed using MMIO Short Read and Short Write transactions.
30	Minimum Request and Response payload size for Channel 0 access	OBMF-ICP Device shall support MMIO Short Read and Short Write with a minimum Request and Response payload size of 64 bytes in order to comply with OBMF-ICP and enable Channel 0 discovery. This 64-byte minimum is specific to Channel 0 / MMIO Short transactions and is not a general minimum for all channels.

3.3. Assumptions

The following are the key assumptions about the environment or use cases:

1. The underlying OBMF-ICP transport shall guarantee message delivery, that is, OBMF-ICP messages shall not be dropped nor duplicated.
2. The underlying OBMF-ICP transport shall guarantee in-order delivery, that is, OBMF-ICP messages are never reordered.
3. The underlying OBMF-ICP transport shall guarantee message integrity.
4. OBMF-ICP does not define priority differentiation mechanisms. If needed, the underlying OBMF-ICP transport layer can be used to differentiate priorities.

Note that MCTP alone (DSP0236 and related standard protocols) does not provide the above guarantees. For this reason, OBMF-ICP over MCTP Binding needs to define reliable delivery methods.

3.4. Trademarks

Names and brands may be claimed as trademarks by their respective companies. I2C® is a trademark of NXP Semiconductor. PCIe® and PCI Express® are the registered trademarks of PCI-SIG. I3C® is a trademark of MIPI Alliance, Inc.

3.5. References

References to other standards:

- SMBus Management Interface Forum. System Management Bus (SMBus) Specification. System Management Interface Forum, Inc, Version 2.0, August 3rd, 2000
- MIPI alliance Specification for I3C BasicSM Version 1.1.1. – Released July 2021
- USB Implementers Forum. Universal Serial Bus Specification, Revision 2.0, April 27th, 2000
- PCI-SIG PCI Express Base Specification Revision 6.3, January 15, 2025
- DMTF Standard. Management Component Transport Protocol (MCTP) SMBus/I2C Transport Binding Specification, Rev 1.2.0, Apr 6, 2020
- DMTF Standard. DSP0283, Management Component Transport Protocol (MCTP) Universal Serial Bus (USB) Transport Binding Specification, Rev 1.1.0, May 14th, 2026
- DMTF Standard. DSP0233, Management Component Transport Protocol (MCTP) I3C Transport Binding Specification, Rev 1.0.1, Mar 25, 2024
- DMTF Standard. DSP0238, Management Component Transport Protocol (MCTP) PCIe® VDM Transport Binding Specification, Rev 1.4.0, Feb 28, 2026
- TCG TPM Communication over SPDM Secure Session, Version 1.0, December 12, 2025
- Intelligent Platform Management Specification Second Generation, Version 2.0, Revision 1.1, October 1st, 2013
- TCG PC Client Platform TPM Profile (PTP) Specification for TPM 2.0, Version 1.07, March 23, 2026

References to other OBMF-specific documents (all documents summarized on <https://github.com/opencomputeproject/ocp-obmf/wiki>):

- DMTF Standard. DSP0295, OBMF-ICP over Management Component Transport Protocol (MCTP) Binding Specification, https://www.dmtf.org/sites/default/files/DSP0295_1.0.0WIP80.pdf
- OBMF-ICP over USB Class Specification. <https://www.opencompute.org/documents/obmf-icp-over-usb-1-0-pdf>

3.6. Linux and OpenBMC Interoperability

The OBMF specification defines optimized channel types and configuration options whose naming conventions, flag definitions, and behavioral semantics are intentionally aligned with existing Linux Kernel subsystem APIs including GPIO, I2C, I3C, SPI, TTY/UART, IPMI, etc. for the sole purpose of maximizing interoperability and compliance with the Linux Kernel, OpenBMC, and related open-source BMC software stacks. Any similarities to Linux Kernel interface constants, bit field definitions, or behavioral descriptions are driven exclusively by this interoperability objective. The OBMF specification is an independent work describing hardware and firmware behavior of communication protocols; it does not incorporate copyrightable expression from the Linux Kernel source code.

3.7. Acronyms

For the purposes of the OBMF-ICP specification, the following acronyms apply:

Table 2: Acronyms

Acronym	Definition
ASIC	Application Specific Integrated Circuit
BMC	Baseboard Management Controller
CCC	I3C® Common Command Code
CXL	Compute Express Link
DC-SCM	Data Center Secure Control Module
DDA	I3C® Dynamic Address Assignment
DMTF	Distributed Management Task Force see DMTF.org
ENTDAA	I3C® Enter Dynamic Address Assignment
ESPI	Enhanced Serial Peripheral Interface
FRU	Field Replaceable Unit
GPIO	General Purpose Input/Output
HDR	I3C® High Data Rate
I/O	Input / Output
HPM	Host Processor Module
IBI	I3C® In-Band Interrupt (IBI)
IPMI	Intelligent Platform Management Interface protocol
I2C	Inter-Integrated Circuit - two wire serial protocol
I3C	MIPI Alliance Improved Inter-Integrated Circuit – two wire serial protocol
MMIO	Memory Mapped I/O
MCTP	Management Component Transport Protocol is an open standard defined by DMTF
OCP	Open Compute Project
PLDM	Platform Level Data Model see DMTF.org
PCIe	PCI Express®
RSTACT	I3C® Reset Action CCC
SDR	I3C® Standard Data Rate
SETDASA	I3C® Set Dynamic Address from Static Address

<i>SPI</i>	<i>Serial Peripheral Interface</i>
<i>SPDM</i>	<i>Security Protocols and Data Models as defined by DMTF.org</i>
<i>SSIF</i>	<i>SMBus System Interface</i>
<i>TPM</i>	<i>Trusted Platform Module</i>
<i>UART</i>	<i>Universal Asynchronous Receiver-Transmitter</i>
<i>VDM</i>	<i>Vendor Defined Message</i>

4. OBMF-ICP Architecture Overview

4.1. OBMF-ICP Protocol Stack Overview

This specification defines the OBMF-ICP protocol that is part of a protocol stack as shown in Figure 1:

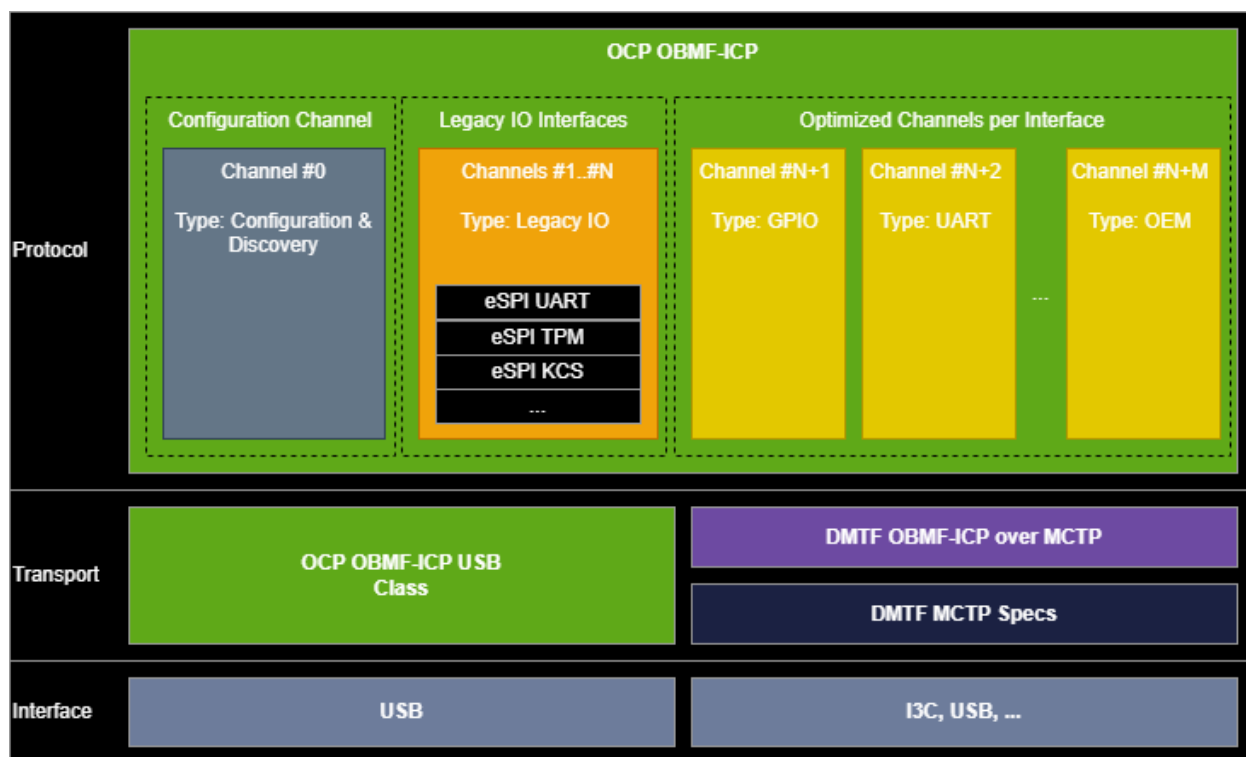


Figure 1: OBMF-ICP Protocol Stack

This specification defines message formats and encoding of the OBMF-ICP layer and relies on the underlying transport specifications, such as:

- OBMF-ICP USB Class specification (please see <https://www.opencompute.org/documents/obmf-icp-over-usb-1-0-pdf> for access) – defines how OBMF-ICP messages are carried over USB bus,

- OBMF-ICP over MCTP protocol stack (please see <https://github.com/opencomputeproject/ocp-obmf/wiki> for access) – multiple specifications define how OBMF-ICP messages are carried over any MCTP-capable bus,
- collateral RDL files (please see at <https://github.com/opencomputeproject/ocp-obmf/wiki> access) – define specific structures that are being modeled to support specific use cases, such as UART, TPM, VW (Virtual Wires), etc.

Figure 2 shows the specification dependencies. Documents published by OCP OBMF workstream as part of HW Management Project are marked with the appropriate symbol:

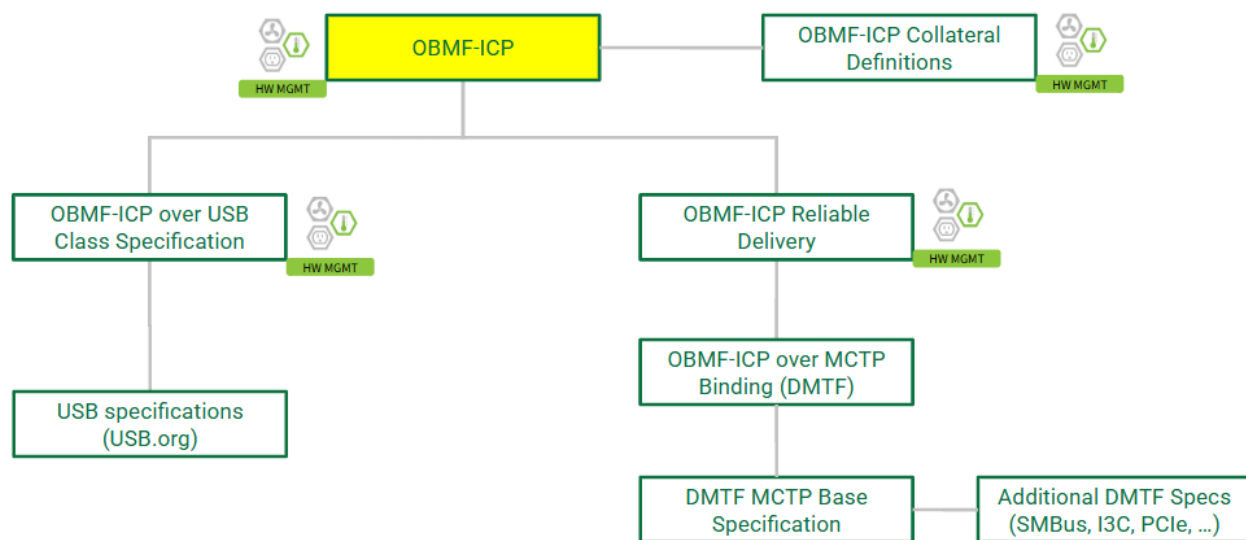


Figure 2: Specifications related to OBMF-ICP

4.2. OBMF-ICP Component Roles

OBMF-ICP is designed to enable communication between two entities that are traditionally connected using a multitude of legacy interfaces. OBMF-ICP replaces all these legacy interfaces with a new protocol.

The two communicating entities will be referred to as:

- **OBMF-ICP Host** – the platform management controller (BMC). The OBMF-ICP Host is always the BMC
- **OBMF-ICP Device** – a component managed by the OBMF-ICP Host (BMC), such as a Host SoC, a Bridge, a NIC, or a GPU

Figure 3 shows the typical platform topology with OBMF-ICP Host and OBMF-ICP Device roles:

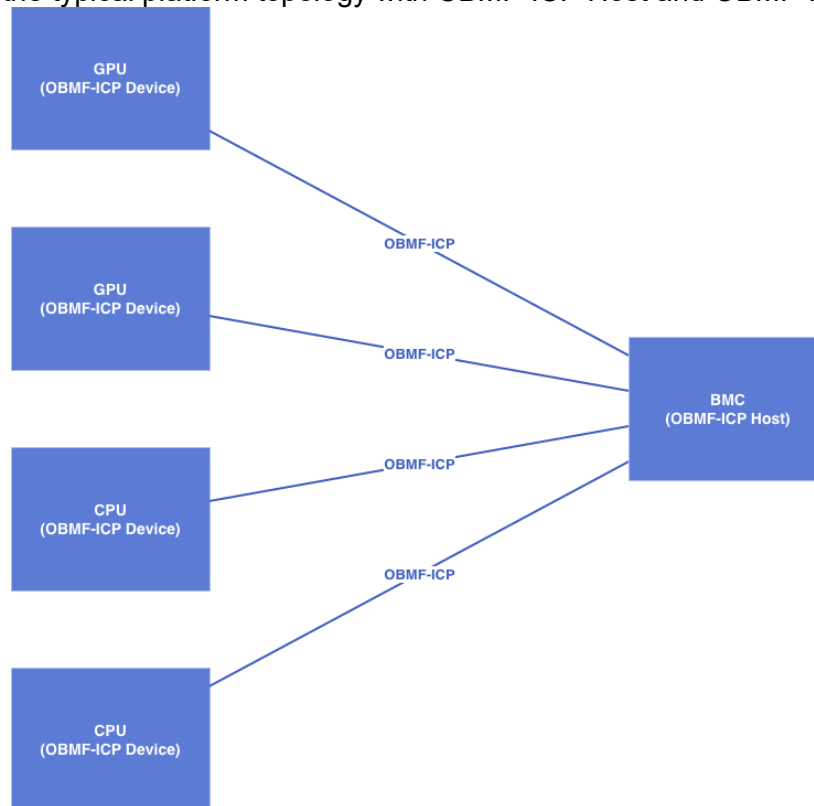


Figure 3: Typical OBMF-ICP Host and OBMF-ICP Device Roles

The legacy interfaces are now replaced with OBMF-ICP Channels. For example, there may be an OBMF-ICP Channel to provide Real-Time Clock (RTC) access using I/O Channel: a processor can ask the BMC to provide the current RTC information by reading data from a well-defined RTC data structure maintained by the BMC. For each channel, OBMF-ICP distinguishes two roles:

- **OBMF-ICP Producer** – for a given channel, the endpoint that produces and exposes the channel's data or service to the OBMF-ICP Consumer
- **OBMF-ICP Consumer** – for a given channel, the endpoint that consumes the channel's data or service provided by the OBMF-ICP Producer

In the example about RTC service above, BMC would be the OBMF-ICP Producer while the processor would be the OBMF-ICP Consumer. Note that at the same time the roles may be reversed for a different channel – for example, a processor may be an OBMF-ICP Producer of FRU information, and BMC may be requesting FRU information as an OBMF-ICP Consumer.

Channel 0 – mandatory channel that must be supported by the OBMF Device:

Channel 0 is used to communicate the list of other expected channels and any other important discovery information. Channel 0 is always exposed by the OBMF-ICP Device. The OBMF-ICP Host (BMC) never exposes Channel 0, as illustrated in Figure 4. If the OBMF-ICP Device needs to consume one or more channels (that is, act as the OBMF-

ICP Consumer for those channels), it shall advertise this on a per-channel basis using the `DEVICE_ROLE` set to Consumer in the Channel 0 discovery structure, thereby informing the OBMF-ICP Host which channels the Host is expected to produce and support for Device to Consume.

Channels 1 – N use two primary types for accessing interfaces behind OBMF:

1. MMIO channel type – set of standard data structure definitions (please see at <https://github.com/opencomputeproject/ocp-obmf/wiki> access) that typical implementations would follow. A specific vendor may define and publish (preferably using the same RDL format) additional extensions. The published definitions (standard and vendor-defined extensions) allow vendors (typically BMC HW/FW vendors) to implement the proper support of necessary services.
2. Interface-optimized channels – set of OBMF-ICP standardized and optimized channels for efficient access of specific interfaces like GPIO, I2C/SMBus, UART, SSIF/IPMI, SPI, and other future/OEM-defined channels.

The concept of channels and associated MMIO data structures or dedicated channels is shown in Figure 4:

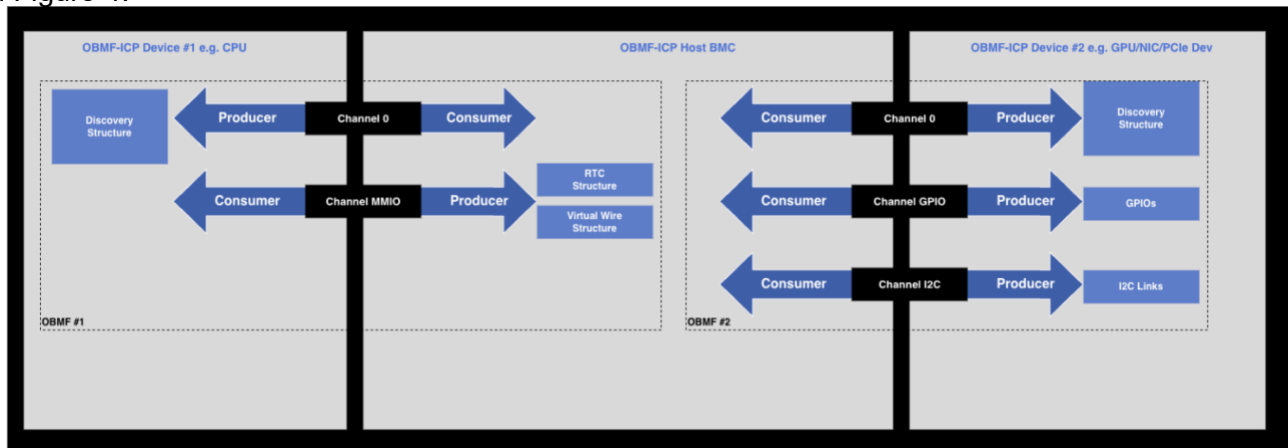


Figure 4: Concept of OBMF-ICP Channels and Data Structures

In order to enable data exchange, OBMF-ICP Protocol defines a standardized set of OBMF-ICP Request and Response transactions that is used to access OBMF-ICP Channels. For each transaction, there are two phases of communication, and we can define two roles of the communicating entities:

- **OBMF-ICP Requester** – the entity that initiates the read or write request
- **OBMF-ICP Responder** – the entity that handles the request and provides a response

In the majority of situations, it is the OBMF-ICP Consumer that is the OBMF-ICP Requester: initiates the requests to access the channels. However, some OBMF-ICP Channels also allow the OBMF-ICP Producer to initiate the request to the OBMF-ICP Consumer in order to notify about some asynchronous changes. For example, consider a BMC acting as OBMF-ICP Producer to provide Virtual Wire (VW) or GPIO channel service to a processor (OBMF-ICP Consumer). The processor may want to initiate Request transactions to get the state of an input

VW/GPIO. It may also initiate a Write transaction to update the output state of another wire. Additionally, BMC may also initiate a Request transaction to notify the processor about GPIO state changes. This example is shown in Figure 5:

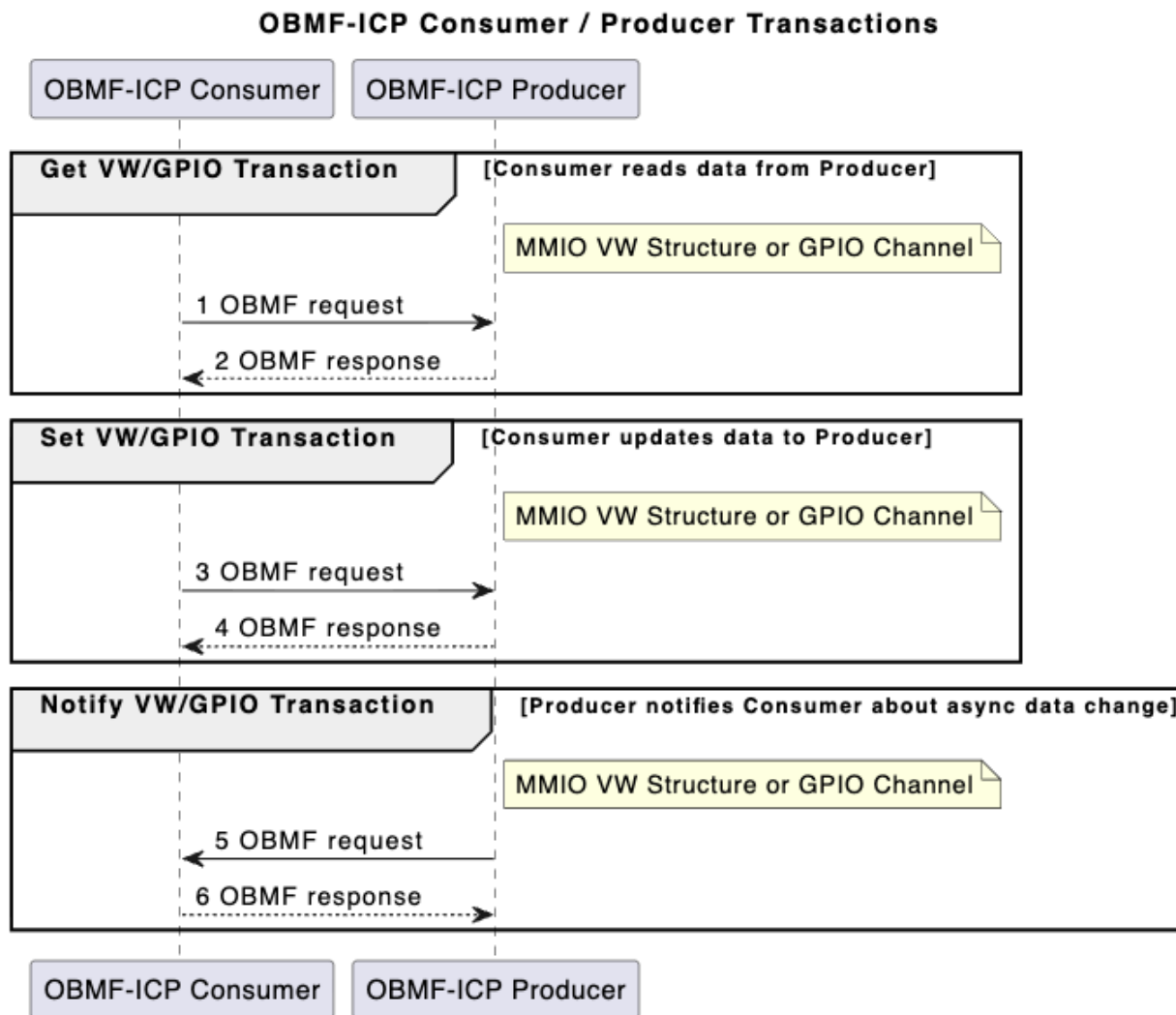


Figure 5: Sample Request and Response Transaction Uses

4.3. Typical Use Case Example

The OBFM-ICP specification supports flexible and any mix of channel use but primarily defines two main use cases for specific channel types.

1. Host SoC to BMC interfaces – MMIO-based channels are intended for seamless enabling of OBMF-ICP. Use of OBMF Bridge (discrete or integrated) is possible integrate Host SoC native interfaces.
2. BMC to Devices (GPU, NIC, PCIe Cards, etc.) interfaces – Interface-optimized channels are intended for best BW optimization and highest efficiency

The specification does not restrict its application to only these scenarios - other use cases are also supported. To efficiently enable OBMF-ICP on both legacy CPUs and on new IO devices, that BMC needs to manage, this specification defines two types of OBMF-ICP interface consolidation. Refer to example below which outlines possible integration options for OBMF channels.

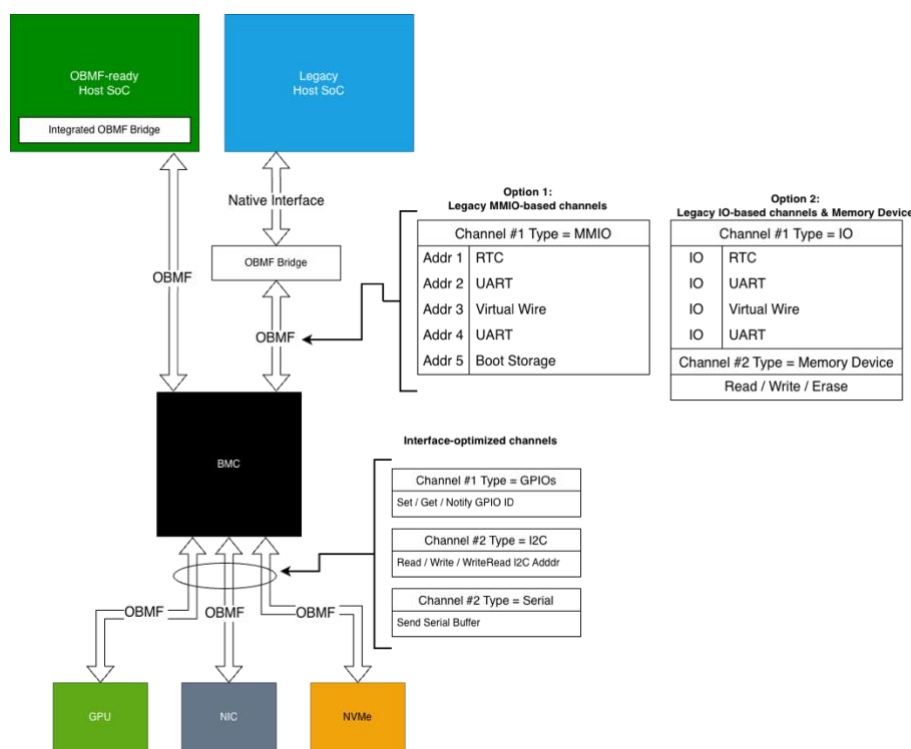


Figure 6 Example of use cases for various OBMF Channel Types

Precondition:

1. For OBMF-ICP MMIO channels OBMF Device vendor defines and publishes their own vendor-defined extensions required to be supported by:
 - a. Option 1: OBMF Host MMIO Channel or
 - b. Option 2: IO Ports and Memory Device Channels
2. BMC HW/FW (OBMF-ICP Host) implements services to meet the defined expectations by exposing Channels requested by Device.
3. OBMF Interface-optimized channel definitions shall align with the existing BMC FW software stack to minimize the number of required changes and new driver development.

4. OBMF Bridge for Host SoC integration (discrete or embedded) needs to be available early in the Host SoC initialization flow aligned to early use of native interfaces on the Host SoC.

System operation sequence:

1. BMC (OBMF-ICP Host) discovers Device (OBMF-ICP Device) capabilities by reading Channel 0 content.
2. An OBMF-ICP Device, such as a Host SoC or an OBMF Bridge connected to a Host SoC, uses the Channel 0 Device Role defined per channel to declare whether a specific channel is produced or consumed by the OBMF-ICP Device.
 - a. Channels Consumed by the Device: Device boot flow often depends on the Consumer channels to be available (e.g. Host SoC waits for Memory Device Consumer channel to start fetching the Boot Image). To allow for that, the OBMF-ICP Host enables the necessary OBMF-ICP Device channels discovered with the Consumer role. By enabling the Device Consumer channels through Channel 0, the OBMF-ICP Host notifies the Device that it can start using the requested channels as Consumer. This completes the contract between the OBMF-ICP Host and the OBMF-ICP Device in that regard. If it is not met, the Device boot will not proceed. The OBMF-ICP Device uses the available channels by sending OBMF-ICP requests on a specific OBMF-ICP channel to the OBMF-ICP Host.
 - b. Channels Produced by the Device: OBMF-ICP Host verifies support for channels produced by the OBMF-ICP Device and configures them accordingly. From this point, the OBMF-ICP Host can start using the channels by sending OBMF-ICP requests on a specific OBMF-ICP channel to the OBMF-ICP Device.
3. For the majority of use cases of OBMF-ICP Devices, such as a GPU, a NIC, or a general-purpose OBMF Bridge, the Device role is limited to the Producer role for all channels. These channels can be pre-configured in Manufacturing and always enabled for OBMF-ICP Host usage. The OBMF-ICP Host discovers the supported channels, can modify the runtime configuration, and starts using the channels by sending OBMF-ICP requests on a specific OBMF-ICP channel to the OBMF-ICP Device.

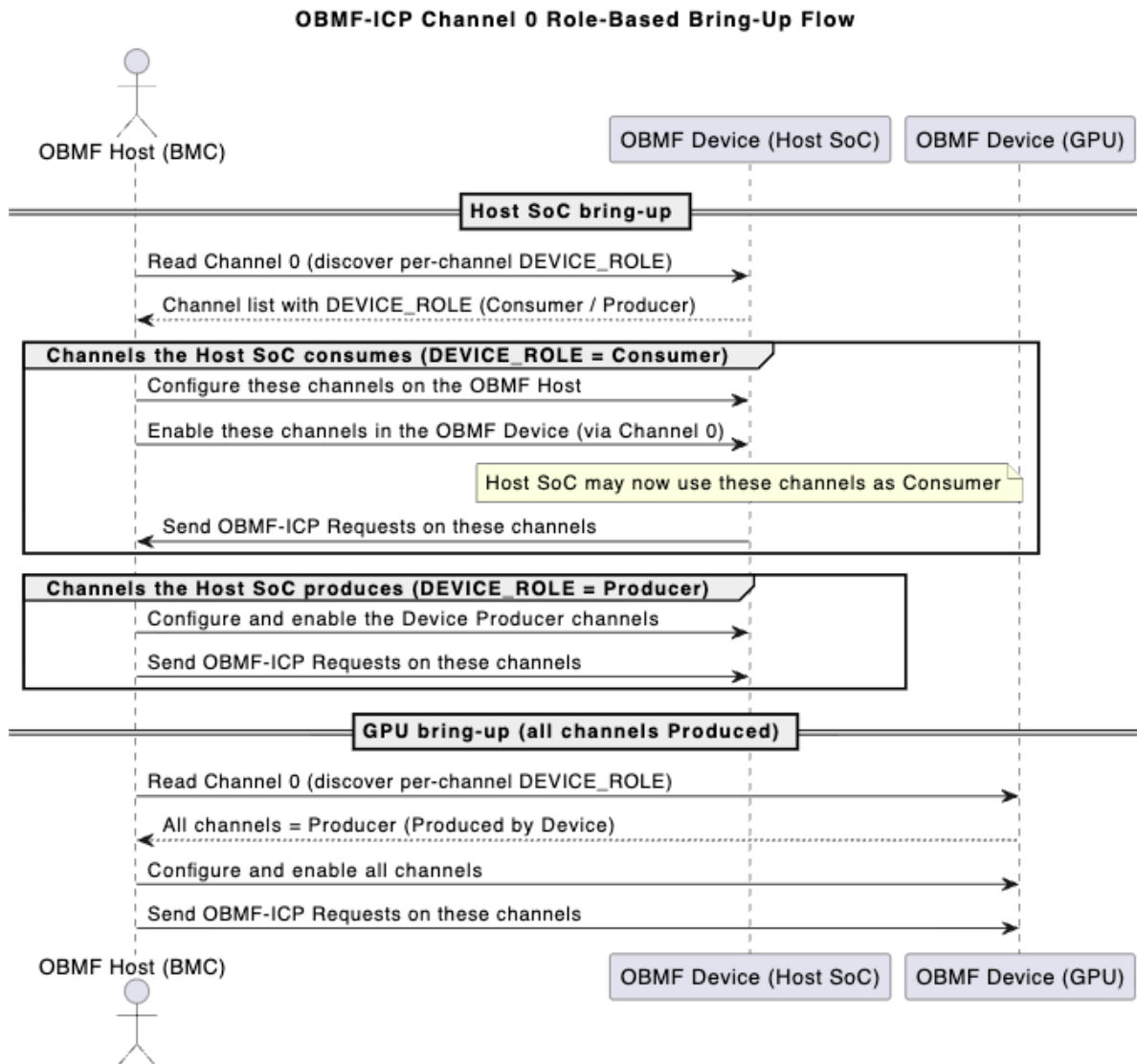


Figure 7 Example of OBFM Channels Discovery and Configuration

5. OBFM-ICP Protocol Definition

As stated in section 4.2, OBFM-ICP protocol defines the concept of channels to enable bi-directional communication between an OBFM-ICP Producer and an OBFM-ICP Consumer. Each OBFM-ICP Channel supports a service offered by OBFM-ICP Producer – each service is represented by a data structure (a register map) that OBFM-ICP Producer and OBFM-ICP Consumer share for MMIO based channels or the dedicated Interface-optimized channels.

Each channel is independent, that is, transactions initiated by OBFM-ICP Requester on one channel are expected to be serviced by the OBFM-ICP Responder independently of the state of

the other channels. For example, a processor acting as OBMF-ICP Consumer may request services over RTC channel, while waiting for a previous request to be fulfilled over a Flash access channel.

Each channel needs to have a Type. There are predefined channel types defined in OBMF-ICP specification and the possibility to define customized channel types with use of OEM channel types.

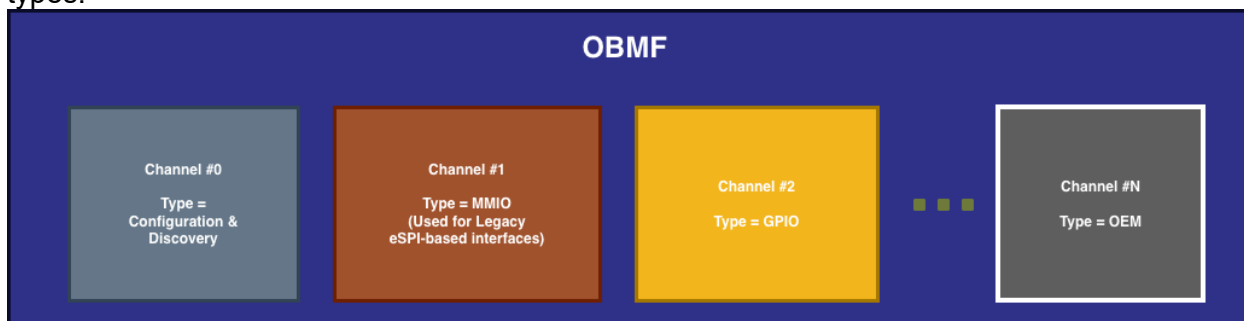


Figure 8 Multiple OBMF-ICP Channel Types

Channel 0 is used for early discovery and configuration. Channel 0 is always enabled and always available from early boot. Channel #0 allows for OBMF channels configuration. OBMF also supports optional Manufacturing Mode which allows for locking configuration in non-volatile storage during manufacturing. Runtime configuration shall always be available and can be used to overwrite the manufacturing configuration until the next reset. Refer to chapter 5.15 for more details.

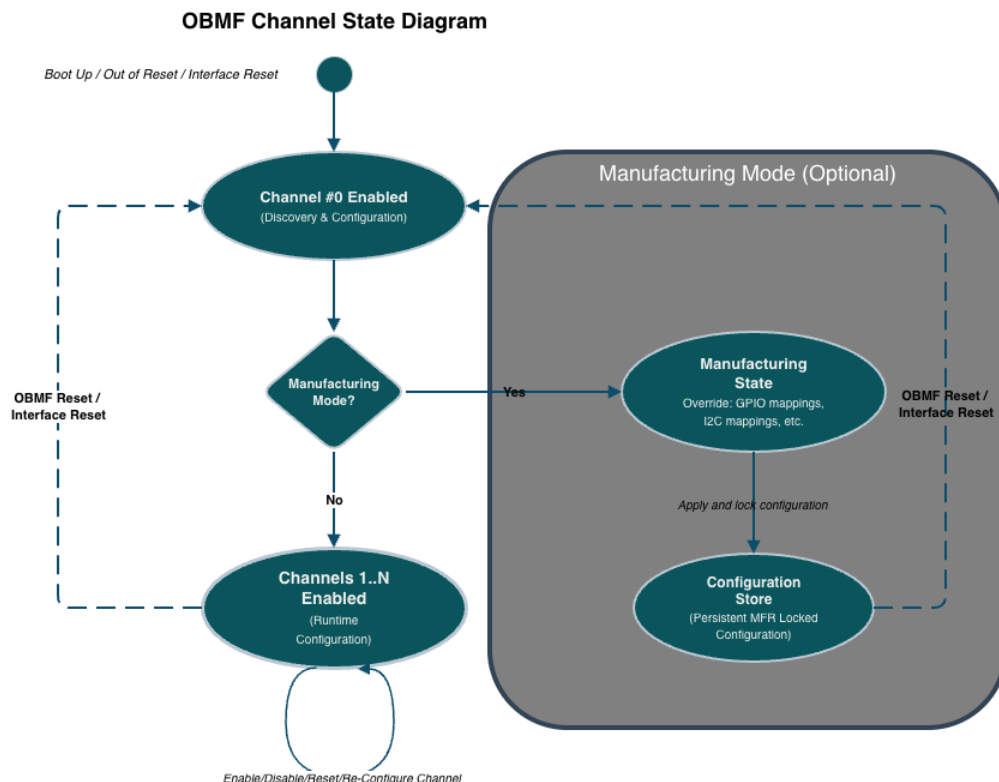


Figure 9 OBMF Runtime and Manufacturing Configuration support

5.1. OBMF-ICP Message Format

The following types of messages are defined by OBMF-ICP protocol:

- Request – allows the requester to send request to specific channel
- Response – allows the responder to provide data from channel or acknowledge reception of data sent to channel

As explained in more detail in section 4.2, one party initiates a Read request and the other party is expected to handle the request and provide a Read Response. Note that a specific channel may allow Read requests in one direction only. Similarly, if allowed by a specific OBMF-ICP Channel, the same interaction applies to Write transactions. Write transactions, if allowed by a specific channel, may be sent in the other direction as an asynchronous notification. Flow control, rate limiting and QoS depend on the OBMF-ICP interface binding and is out of scope of the OBMF-ICP protocol definition.

OBMF-ICP Message format is defined in the table below. OBMF-ICP message-body fields are little-endian unless explicitly stated otherwise. Transport-binding headers are encoded as defined by the respective binding specification.

Table 3: OBMF-ICP Message Format

Bytes[Bits]	Field	Description
0[7:0]	OBMF-ICP Channel	OBMF-ICP Channel number: Channel zero is reserved for discovery purposes – see section 5.15 All other channel assignments are as per descriptor definitions.
1[6:0]	Status	OBMF-ICP Response Status Code in Response (RqResp = 1) Reserved set to 0 in Request (RqResp = 0)
1[7]	RqResp	Indicates whether the OBMF-ICP message is a request or a response: 0: Request 1: Response
3:2	Size	Size of the Payload in bytes
N:4	Payload	Payload related to each type of OBMF-ICP transaction as defined in subsequent sections. Note: Payload size is determined by the size of the transport layer message.

The following subsections starting with 5.3 define the Payload for different OBMF-ICP Channel Types.

5.2. OBMF-ICP Status Codes

Each Channel defines a Status Code in the Response message definition. The Status Codes are divided into two groups:

- 0x00 – 0x3F: Generic Status Codes
- 0x40 – 0x7F: Channel-specific Status Codes

The Generic Status Codes are defined below. Channel-specific Status Codes are defined per channel definition when applicable.

Table 4: Generic Status Codes

Value	Description
0x00	Success
0x01	Invalid command or parameter for this channel
0x02	Request timeout
0x03	Channel not ready / busy
0x04	Channel permanent error
0x05	Unknown OBMF-ICP Channel / Channel not supported
0x06	Request size not supported
0x07 – 0x3F	Reserved

5.3. OBMF MMIO Channel

Table 5: OBMF MMIO Channel

Bytes[Bits]	Field	Description
0[2:0]	OBMF-ICP Transaction	The following transaction types are defined: 0: Short Read 1: Short Write 2: Long Read 3: Long Write 4-7: Reserved Producer is expected to support all four transaction types.
0[7:3]	Reserved	Reserved – Set as 0.
1	Reserved	Reserved – Set as 0.
N:2	Payload	Payload related to each type of OBMF-ICP transaction as defined in subsequent sections. Note: Payload size is determined by the size of the transport layer message.

For the MMIO channel, requests may be initiated by either endpoint. In the majority of cases the OBMF-ICP Consumer initiates the request and the OBMF-ICP Producer sends the response; however, the OBMF-ICP Producer may also initiate a request to the OBMF-ICP Consumer (for example, to notify the Consumer of an asynchronous change). The direction of each message is indicated by the RqResp bit in the OBMF-ICP message header. Both the Consumer and the Producer of an MMIO channel shall be able to act as OBMF-ICP Requester and OBMF-ICP Responder as required by the channel definition.

MMIO Channel Roles:

Table 6: MMIO Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	Yes	Yes
Consumer	Yes	Yes

MMIO Channel-specific Status Codes are defined below:

Table 7: MMIO Channel-specific Status Codes

Value	Description
0x40	Address out-of-range
0x41	Access denied for privilege level
0x42 – 0x7F	Reserved

5.3.1. Short Read Request

OBMF-ICP Read Request message field definitions are as follows:

Table 8: Short Read Request

Bytes[Bits]	Field	Description
3:0[31:0]	Address	32-bit address of data to read (no alignment restrictions).
5:4	Read Length	Read length in bytes. Maximum size is discovered using Channel 0 Discovery Structure

5.3.2. Short Read Response

OBMF-ICP Read Response message field definitions are as follows:

Table 9: Short Read Response

Bytes[Bits]	Field	Description
(Read Length-1):0	Data	If Status is equal to Success, this field carries <i>Read Length</i> number of bytes of the MMIO Data returned in response to the request Else if Status is different from Success, this field is omitted

5.3.3. Short Write Request

OBMF-ICP Write Request message field definitions are as follows:

Table 10: Short Write Request

Bytes[Bits]	Field	Description
3:0 [31:0]	Address	32-bit address of data to write (no alignment restrictions).
(Size-4):4	Data	Data to be written at the address location, where Size is from OBMF-ICP Message header

5.3.4. Short Write Response

OBMF-ICP Write Response message field definitions are as follows:

Table 11: Short Write Response

Bytes[Bits]	Field	Description
No Payload.		

5.3.5. Long Read Request

OBMF-ICP Read Request message field definitions are as follows:

Table 12: Long Read Request

Bytes[Bits]	Field	Description
7:0[63:0]	Address	64-bit address of data to read (no alignment restrictions).
9:8	Read Length	Read length in bytes. Maximum size is discovered using Channel 0 Discovery Structure

5.3.6. Long Read Response

OBMF-ICP Read Response message field definitions are as follows:

Table 13: Long Read Response

Bytes[Bits]	Field	Description
(Read Length - 1):0	Data	If Status is equal to Success, this field carries <i>Read Length</i> number of bytes of the MMIO Data returned in response to the request Else if Status is different from Success, this field is omitted

5.3.7. Long Write Request

OBMF-ICP Write Request message field definitions are as follows:

Table 14: Long Write Request

Bytes[Bits]	Field	Description
7:0[63:0]	Address	64-bit address of data to write (no alignment restrictions).
(Size-8):8	Data	Data to be written at the address location, where Size is from OBMF-ICP Message header

5.3.8. Long Write Response

OBMF-ICP Write Response message field definitions are as follows:

Table 15: Long Write Response

Bytes[Bits]	Field	Description
No Payload		

5.4. OBMF GPIO Channel

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. All use cases are supported by this specification.



Figure 10 Example of GPIO channel use case with OBMF Bridge.

The OBMF-ICP GPIO Channel uses a unified message format for all GPIO operations. This format supports the simultaneous execution of set GPIO value or get GPIO value or get interrupt config or set interrupt config or notify interrupts for multiple GPIO lines within a single OBMF-ICP message payload.

GPIO Channel Roles:

Table 16: GPIO Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	Yes (GPIO Interrupts)	Yes
Consumer	Yes	Yes

GPIO Channel-specific Status Codes are defined below:

Table 17: GPIO Channel-specific Status Codes

Value	Description
0x40	Error GPIO Index not supported
0x41	Interrupt not supported for GPIO Index
0x42	Invalid operation for GPIO Configuration (Read on OUT or Write on IN)
0x43 – 0x7F	Reserved

GPIO channel supports option to maintain GPIO state across resets. It's configured via Output Persistent State Configuration property of GPIO channel configuration as per GPIO property. The diagram below outlines how the configuration is used across OBMF Device Resets.

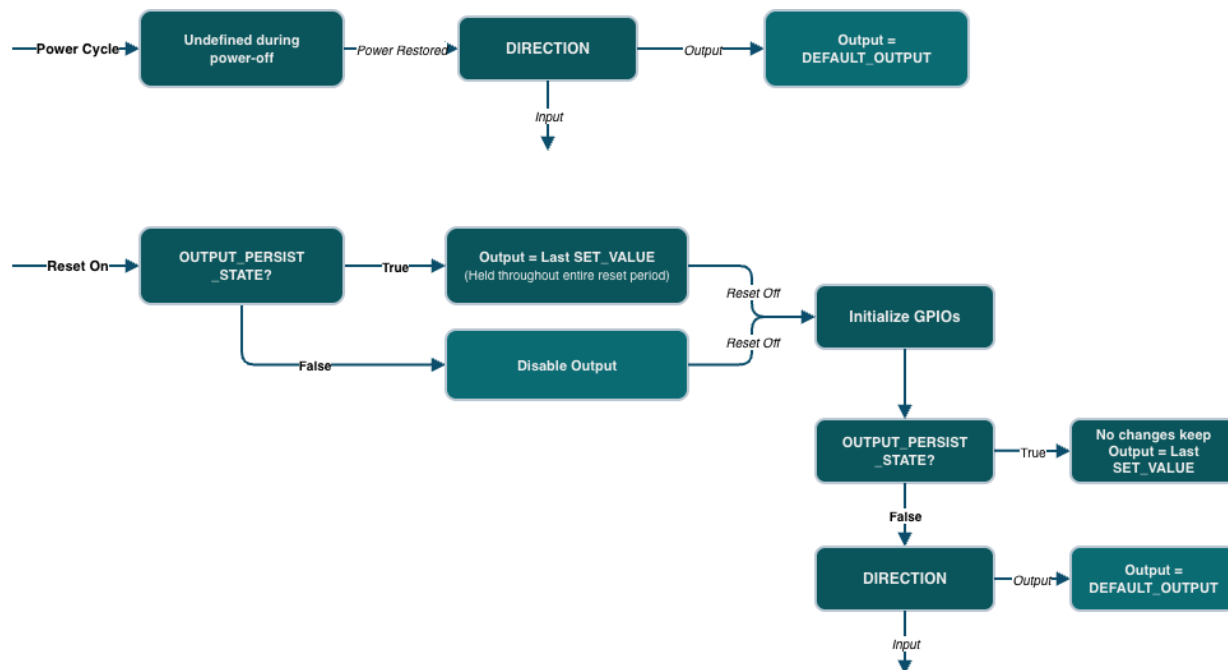


Figure 11 GPIO Persistent State across Reset

5.4.1. GPIO Channel Request

OBMF-ICP GPIO Channel Request message field definitions are as follows:

Table 18: OBMF-ICP GPIO Channel Request message

Bytes[Bits]	Field	Description
0	Command	The following transaction types are defined: 0: Get Values (value) - requests are initiated by the OBMF-ICP Consumer 1: Set Values (value) - requests are initiated by the OBMF-ICP Consumer 2: Get Interrupt Config - requests are initiated by the OBMF-ICP Consumer 3: Set Interrupt Config - requests are initiated by the OBMF-ICP Consumer 4: Interrupt Notify - requests are initiated by the OBMF-ICP Producer 5: Interrupt Notify Datagram (no ack) - requests are initiated by the OBMF-ICP Producer 6 - FF: Reserved
2:1 [11:0]	GPIO Index 0	GPIO Index 0 - 4095

Bytes[Bits]	Field	Description
2:1 [15:12]	GPIO Data 0	For Get Values 0-15: Reserved – Set as 0 For Set Values 0: Low 1: High 2-15: Reserved For Get Interrupt Config 0-15: Reserved – Set as 0 For Set Interrupt Config 0: Disable Interrupt 1: Level Low 2: Level High 3: Rising Edge 4: Falling Edge 5: Rising and Falling Edge 6-15: Reserved For Interrupt Notify and Interrupt Notify Datagram 0-15: Reserved – set as 0
4:3	GPIO Index /Data 1	See GPIO Index / Data 0
...	...	...
N:(N-1)	GPIO Index /Data N	See GPIO Index / Data 0

The GPIO Interrupt Notify Datagram is used to send interrupt notifications. This command does not require a response to be sent back over OBMF and is used for asynchronous interrupt notifications. It does not require OBMF Response to be sent back if binding allows for that – please follow the OBMF Binding Specs for details. When OBMF USB Binding implementation supports Interrupt Endpoint all GPIO Interrupt Notify shall be delivered via Interrupt Endpoints. For cases where all GPIOs cannot fit in a single Interrupt Transfer it shall be split into multiple Interrupt Endpoint Transfers. If channel supports the Datagram Notifications it is always used to send Interrupt Notifications for this channel.

5.4.2. GPIO Channel Response

OBMF-ICP Get GPIO Values Channel Response message field definitions are as follows:

Table 19: OBMF-ICP Get GPIO Values Channel Response message

Bytes[Bits]	Field	Description
0	Command	The following transaction types are defined: 0: Get Values (value) 1: Set Values (value) 2: Get Interrupt Config 3: Set Interrupt Config 4: Interrupt Notify 5: Interrupt Notify Datagram (no ack) 6 - FF: Reserved
Response Payload only applicable to Get Values (value) and Get Interrupt Config command response		
2:1 [11:0]	GPIO Index 0	GPIO Index 0 - 4095
2:1 [15:12]	GPIO Data 0	For GPIO Value 0: Low 1: High 2-15: Reserved For Get Interrupt Config: 0: Disable Interrupt 1: Level Low 2: Level High 3: Rising Edge 4: Falling Edge 5: Rising and Falling Edge 6-15: Reserved
4:3	GPIO Index /Data 1	See GPIO Index / Data 0
...	...	...
N: (N-1)	GPIO Index /Data N	See GPIO Index / Data 0

5.5. OBMF Serial Channel

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. All use cases are supported by this specification.



Figure 12 Example of Serial channel use case with OBMF Bridge.

OBMF-ICP Serial Channel features an optimized Serial Command that allows for the asynchronous transmission of multiple characters at once, including the handling of serial-specific events. This enhances efficient use of OBMF-ICP interface bandwidth and allows for optimizations on Producer and Consumer sides.

Note: This channel definition is inspired by the Serial over LAN (SOL) commands defined in the IPMI v2.0 Specification. OBMF-ICP Serial Channel definition intentionally builds on IPMI SOL because its high-quality, highly optimized design and broad data center industry adoption will ease OBMF-ICP Serial Channel integration in data centers. The definition is not copied - it is only inspired by IPMI SOL, and has been optimized down to the bare-minimum required functionality to comply with Linux TTY frameworks.

Serial Channel Roles:

Table 20: Serial Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	Yes	Yes
Consumer	Yes	Yes

Serial Channel-specific Status Codes are defined below:

Table 21: Serial Channel-specific Status Codes

Value	Description
0x40	Error line timeout
0x41 – 0x7F	Reserved

5.5.1. Serial Channel Request

OBMF-ICP Serial Channel Request message field definitions are as follows:

Table 22: OBMF-ICP Serial Channel Request message

Bytes[Bits]	Field	Description
0	Operation/Event	[7:4]: Reserved [3]: 1b = Carrier down (typically remote Serial/UART interface to BMC) [2]: 1b = Transmit overrun (typically remote Serial/UART interface to BMC) [1]: 1b = Break detected (typically remote Serial/UART interface to BMC) [0]: 1b = Generate Break (typically BMC to remote Serial/UART interface)
2:1	Character Count	Number of characters in the Character Data 0: Set to 0 for Operation/Status Request only
N:3	Character Data	Character data. May be zero or more bytes, up to the maximum allowed by the OBMF packet size.

5.5.2. Serial Channel Response

OBMF-ICP Serial Channel Response message field definitions are as follows:

Table 23: OBMF-ICP Serial Channel Response message

Bytes[Bits]	Field	Description
0	Packet ACK/NACK	[7:1]: Reserved [0]: Packet ACK/NACK 0b = ACK Packet – the whole packet is Acknowledged 1b = NACK Packet – the packet is Not Acknowledged or partially acknowledged if Accepted Character Count > 0
2:1	Accepted Character Count	Number of characters accepted (setting to 0 means the whole packet is NACK'ed). For partially accepted packets (NACK with Accepted Character Count > 0) the Sender should send the next packet as a retry starting with the first byte of data that the Receiver did not accept. Note: The sender can append more characters to the retried packet. Refer to example flow in Figure 13.

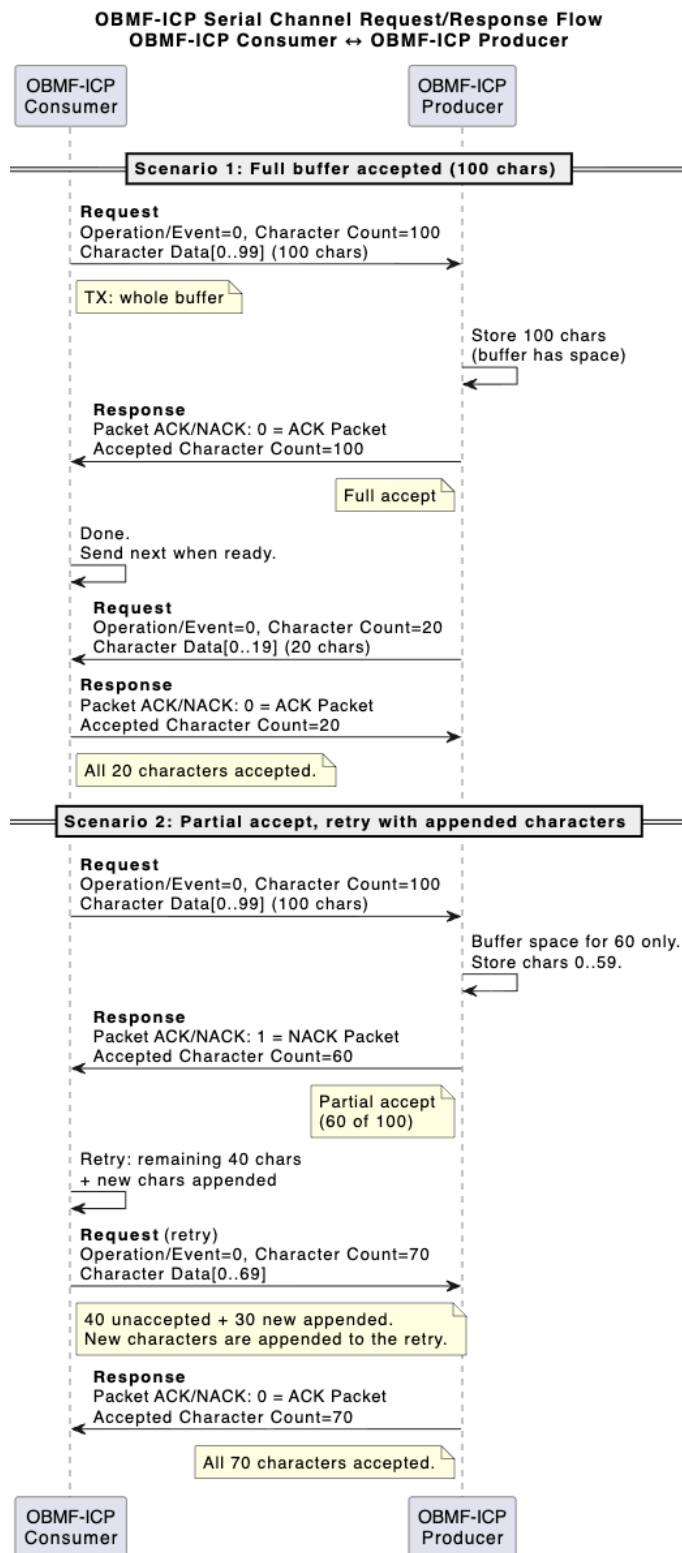


Figure 13 Example Serial Channel flow diagram

5.6. OBMF I2C Controller Channel

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. All use cases are supported by this specification. OBMF-ICP Bridge will implement Channel 0 and will support all the OBMF-ICP Device requirements including support for DEVICE_ROLE.



Figure 14 Example of I2C Controller channel use case with OBMF Bridge.

I2C Controller Channel Roles:

Table 24: I2C Controller Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	Yes (SMBus Host Notify)	Yes
Consumer	Yes	Yes

I2C Controller Channel-specific Status Codes are defined below:

Table 25: I2C Controller Channel-specific Status Codes

Value	Description
0x40	Error SMBus/I2C timeout
0x41	Error Arbitration Lost
0x42	Error Bus Busy
0x43	Error NACK
0x44 – 0x7F	Reserved

5.6.1. I2C Controller Channel Request

OBMF-ICP I2C Channel Request message field definitions are as follows:

Table 26: I2C Controller Channel Request

Bytes[Bits]	Field	Description
0[6:0]	Command	The following transaction types are defined: 0: I2C/SMBus Read - requests are initiated by the OBMF-ICP Consumer 1: I2C/SMBus Write - requests are initiated by the OBMF-ICP Consumer 2: SMBus Block Read - requests are initiated by the OBMF-ICP Consumer 3: SMBus Write Read - requests are initiated by the OBMF-ICP Consumer 4: SMBus Host Notify - requests are initiated by the OBMF-ICP Producer 5 – 127: Reserved Refer to table below for details of all SMBus operations encoding.
0[7]	Send STOP between transactions	0: Send STOP 1: Do not send STOP For SMBus Host Notify this field is set to 0
1	I2C/SMBus Address	Target device address For SMBus Host Notify this field contains Device Address from Host Notify Payload.
3:2	Read Length	Size of Read Data, for Write-only set to 0. For SMBus Block Read with PEC set to 1 to indicate that one more byte needs to be read. For SMBus Host Notify set as 0.
N:4	Write Data	Data to Write For SMBus Host Notify contains 16 bits device status following the Device Address.

For SMBus compliant devices the PEC support is done by extending Read and Write Lengths with PEC size. For the SMBus Block Read the size is unknown thus the Read Length shall be set to 1 if PEC is required.

For devices that require SMBus Host Notify protocol the I2C Controller capabilities reported over Channel 0 shall indicate whether it is supported by the device or not. If SMBALERT# is used GPIO Channel shall be used to support SMBALERT# notification.

SMBus Operations encoding using OBMF I2C Controller Commands

Table 27: SMBus Operations Encoding Using OBMF I2C Controller Commands

SMBus Operation	OBMF I2C Controller Command	Read Length	Write Length
Quick Command (Write, R/W=0)	1 — I2C/SMBus Write	0	0

Quick Command (Read, R/W=1)	0 — I2C/SMBus Read	0	0
Send Byte	1 — I2C/SMBus Write	0	Data (1)
Receive Byte	0 — I2C/SMBus Read	1	0
Write Byte	1 — I2C/SMBus Write	0	CMD + Data (2)
Write Word	1 — I2C/SMBus Write	0	CMD + Data (3)
Read Byte	3 — SMBus Write Read	1	CMD (1)
Read Word	3 — SMBus Write Read	2	CMD (1)
Process Call	3 — SMBus Write Read	2	CMD + Data (3)
SMBus Block Write	1 — I2C/SMBus Write	0	CMD + Data + PEC
SMBus Block Read	2 — SMBus Block Write Read	0 (no PEC) / 1 (PEC)	CMD (1)
Block Write–Block Read Process Call	2 — SMBus Block Write Read	0 (no PEC) / 1 (PEC)	CMD + Data + PEC
SMBus Host Notify	4 — SMBus Host Notify	0	Device Status (2)

5.6.2. I2C Controller Channel Response

OBFM-ICP I2C Channel Response message field definitions are as follows:

Table 28: I2C Controller Channel Response

Bytes[Bits]	Field	Description
0[6:0]	Command	The following transaction types are defined: 0: I2C/SMBus Read 1: I2C/SMBus Write 2: SMBus Block Write Read 3: SMBus Write Read 4: SMBus Host Notify 5 – 127: Reserved
0[7]	Reserved	Reserved set to 0
2:1	Read Length	Size of Read Data Reserved for Write-only Response set to 0
3	Reserved	Reserved set to 0
N:4	Read Data	Data Read from Target

5.7. OBMF I2C Target Channel

OBMF-ICP specification recommends use of MCTP over USB Binding and standardized MCTP Bridge implementation in OBMF Bridge device whenever possible. This channel is used exclusively for cases where MCTP Bridge and MCTP over USB cannot be supported by OBMF device and direct OBMF to MCTP over SMBus interface is the only option. In this configuration, the I2C Target channel is limited to supporting only I2C Writes to the OBMF I2C Target. To enable complete MCTP support, including I2C Write responses from the BMC and OBMF Bridge device, this channel must be paired with an OBMF I2C Controller channel configured on the same SMBus interface. The figure below illustrates an example configuration using a discrete OBMF Bridge. Alternatively, the OBMF endpoint may be embedded directly within an ASIC or Host SoC, in which case no discrete Bridge is required. Both configurations are within scope of this specification.



Figure 15 Example of I2C Target channel use case with OBMF Bridge.

Since the OBMF I2C Target channel is optimized to acknowledge all written bytes up to the maximum buffer supported or configured by the BMC, the OBMF driver on the BMC needs to emulate byte-level acknowledge behavior to provide compliance with the MCTP over SMBus binding.

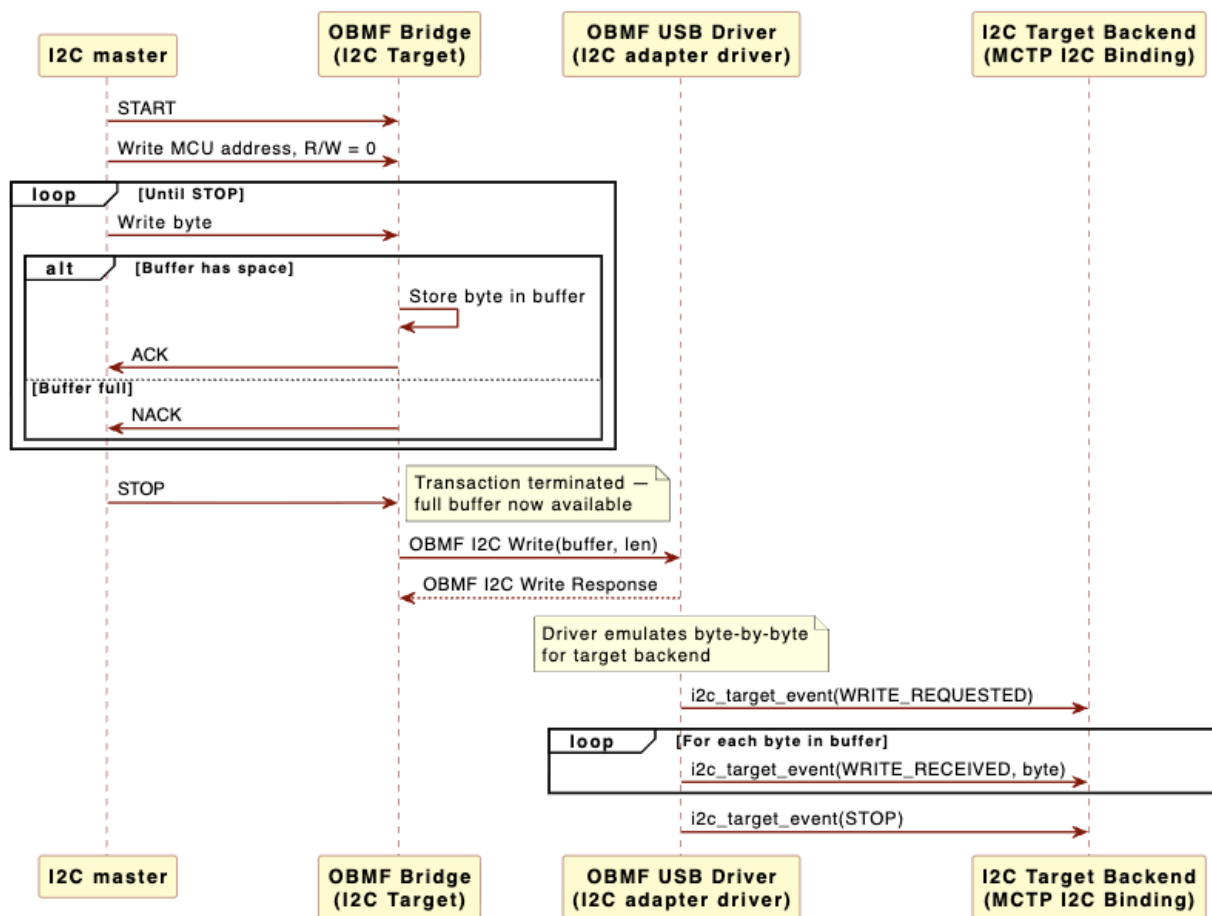


Figure 16 Conceptual view of how I2C Writes are handled by the OBMF Target Channel.

I2C Target Channel Roles:

Table 29: I2C Target Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	No	Yes
Consumer	Yes	No

I2C Target Channel-specific Status Codes are defined below:

Table 30: I2C Target Channel-specific Status Codes

Value	Description
0x40	Error Transaction
0x41 – 0x7F	Reserved

5.7.1. I2C Target Channel Request

OBMF-ICP I2C Channel Request message field definitions are as follows:

Table 31: I2C Target Channel Request

Bytes[Bits]	Field	Description
0[6:0]	Command	The following transaction types are defined: 0: Reserved 1: I2C Write 2 – 127: Reserved
0[7]	Reserved	Reserved write as 0
1	I2C/SMBus Address	Target device address
3:2	Reserved	Reserved
N:4	Write Data	Data to Write

5.7.2. I2C Target Channel Response

OBMF-ICP I2C Channel Response message field definitions are as follows:

Table 32: I2C Target Channel Response

Bytes[Bits]	Field	Description
0[6:0]	Command	The following transaction types are defined: 0: Reserved 1: I2C Write 2 – 127: Reserved
0[7]	Reserved	Reserved write as 0

5.8. OBMF I3C Controller Channel

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. All use cases are supported by this specification.



Figure 17 Example of I3C Controller channel use case with OBMF Bridge.

I3C Controller Channel Roles:

Table 33: I3C Controller Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	Yes (IBI, Hot Join, Error)	Yes
Consumer	Yes	Yes

I3C Channel-specific Status Codes are defined below:

Table 34: I3C Channel-specific Status Codes

Value	Description
0x40	Error I3C Bus Timeout (timed out waiting for SCL or target response)
0x41	Error Arbitration Lost
0x42	Error Bus Busy
0x43	Error NACK (target device NACKed its broadcast/unicast address or data byte)
0x44	Error Bus / Framing Fault (physical bus fault or electrical exception)
0x45	Error Parity Failure (parity error during write, CCC, or dynamic address assignment)
0x46	Error Illegal CCC Reply Format
0x47 – 0x7F	Reserved

5.8.1. I3C Controller Channel Request

OBMF-ICP I3C Channel Request message field definitions are as follows:

Table 35: OBMF-ICP I3C Channel Request message

Bytes	Field	Description
0	Command	Consumer-initiated (Host to Bridge over Bulk OUT/IN): 0: Execute Sequence (I3C_CMD_EXECUTE_SEQUENCE) – chained SDR/CCC operations initiated by Consumer. 1: Dynamic Address Assignment (I3C_CMD_DO_DAA) – ENTDA sequence initiated by Consumer. 2: Bus Recovery (I3C_CMD_BUS_RECOVERY) – reset/recovery pattern initiated by Consumer. 3: Set Associated I2C (I3C_CMD_SET_ASSOCIATED_I2C) – associate I2C channel initiated by Consumer. 4: Get Stats (I3C_CMD_GET_STATS) – retrieve hardware error/event counters initiated by Consumer. Producer-initiated (Bridge to Host over Interrupt IN): 5: IBI Notification (I3C_EVT_IBI_NOTIFICATION) 6: Hot-Join Notification (I3C_EVT_HOTJOIN_NOTIFICATION) 7: Bus Error Notification (I3C_EVT_BUS_ERROR) 8 – 255: Reserved
1	parameter_1	For Execute Sequence (0): num_ops (total number of chained operations in this sequence). For DAA (1): num_excluded_addresses (number of 7-bit addresses that cannot be used for DAA assignment). For Bus Recovery (2): action_type recovery enum value (0: Assert SCL Low, 1: Standalone Target Reset, 2: Chained RSTACT + Reset, 3: Standalone HDR Exit). For IBI Notification (5): dynamic_addr (7-bit dynamic address of triggering device). For Bus Error (7): error_code hardware fault enum (1: Line Lockup, 2: Arbitration Loss). For Set Associated I2C (3), Get Stats (4), and Hot-Join (6): Reserved, set to 0.
3:2	parameter_2	For Execute Sequence (0): Start of operations descriptor array (see below). For DAA (1): target_scl_od_khz (requested Open-Drain SCL frequency in KHz; set to 0 for default). For Bus Recovery (2): scl_low_duration_us (16-bit duration in μs to hold SCL low; used only when action_type is 0). For Set Associated I2C (3): i2c_channel (16-bit associated OBMF I2C channel number). For IBI Notification (5): payload_len (16-bit incoming IBI data byte count). For Get Stats (4), Hot-Join (6), and Bus Error (7): Reserved, set to 0.
5:4	parameter_3	For DAA (1): target_scl_pp_khz (requested Push-Pull SCL frequency in KHz; set to 0 for default). For Bus Recovery (2): Byte 4 = rstact_value (reset action payload e.g., 0x02; used when action_type is 2). Byte 5 = rstact_target_addr (target 7-bit dynamic address or 0x7E for broadcast; used when action_type is 2). For IBI Notification (5): pending_read (16-bit private read byte count: 0 = standard IBI; M > 0 = number of bytes for valid private read data). For all other commands (3, 4, 6, 7): Reserved, set to 0.
N:2 / N:6	variable_payload	For DAA (1): excluded_addresses_list (starts at offset Byte 6). Array of 1-byte 7-bit addresses equal to num_excluded_addresses. For Execute Sequence (0): operations array (starts at offset Byte 2). Sequential array of num_ops operation descriptors (see below). For IBI Notification (5): payload_data (starts at offset Byte 6). Raw IBI payload

Bytes	Field	Description
		bytes and/or private read bytes (omitted if payload_len is 0 and pending_read is 0). For Bus Recovery (2), Set Associated I2C (3), and Get Stats (4): Omitted (no payload bytes after Byte 5).

I3C_CMD_EXECUTE_SEQUENCE Operations Array

Table 36: I3C_CMD_EXECUTE_SEQUENCE Operations Array

Byte Size	Field	Description
1	op_type	Operation type enum (0: I3C_TRANSFER_CCC, 1: I3C_TRANSFER_CCC_W_DEFINING_BYTE, 2: I3C_TRANSFER_PRIVATE_SDR).
1	send_stop	1 = Issue STOP after transfer; 0 = Issue Repeated-Start (Sr) after transfer. The last operation in the sequence must end with a STOP.
2	target_scl_od_khz	Requested Open-Drain SCL frequency in KHz (e.g., 400 for 400kHz). Set to 0 to use the default bus speed.
2	target_scl_pp_khz	Requested Push-Pull SCL frequency in KHz (e.g., 12500 for 12.5MHz). Set to 0 to use the default bus speed.
2	descriptor_len	Size in bytes of the remaining payload for this descriptor.

I3C_TRANSFER_CCC

Table 37: I3C_TRANSFER_CCC

Byte Size	Field	Description
1	ccc_id	Common Command Code ID (e.g., 0x06 RSTACT, 0x87 SETDASA)
1	dynamic_addr	Target dynamic address, or 0x7E for Broadcast CCC.
1	is_write	1 = Write, 0 = Read.
2	data_len	Number of bytes to transfer.
Variable	write_data	Data to transmit (only present if is_write is 1). Length equals data_len.

I3C_TRANSFER_CCC_W_DEFINING_BYTE

Table 38: I3C_TRANSFER_CCC_W_DEFINING_BYTE

Byte Size	Field	Description
1	ccc_id	Common Command Code ID
1	defining_byte	CCC sub-command defining byte.
1	dynamic_addr	Target dynamic address, or 0x7E for Broadcast CCC.
1	is_write	1 = Write, 0 = Read.
2	data_len	Number of bytes to transfer.
Variable	write_data	Data to transmit (only present if is_write is 1). Length equals data_len.

I3C_TRANSFER_PRIVATE_SDR

Table 39: I3C_TRANSFER_PRIVATE_SDR

Byte Size	Field	Description
1	dynamic_addr	Target dynamic address
1	is_write	1 = Write, 0 = Read.
2	data_len	Number of bytes to transfer.
Variable	write_data	Data to transmit (only present if is_write is 1). Length equals data_len.

5.8.2. I3C Controller Channel Response

OBMF-ICP I3C Channel Response message field definitions are as follows:

Table 40: OBMF-ICP I3C Channel Response message

Bytes	Field	Description
0	Command	Echoes the command transaction type executed: 0: Execute Sequence (I3C_CMD_EXECUTE_SEQUENCE) – chained SDR/CCC operations initiated by Consumer. 1: Dynamic Address Assignment (I3C_CMD_DO_DAA) – ENTDA sequence initiated by Consumer. 2: Bus Recovery (I3C_CMD_BUS_RECOVERY) – reset/recovery pattern initiated by Consumer. 3: Set Associated I2C (I3C_CMD_SET_ASSOCIATED_I2C) – associate I2C channel initiated by Consumer. 4: Get Stats (I3C_CMD_GET_STATS) – retrieve hardware error/event counters initiated by Consumer. Producer-initiated (Bridge to Host over Interrupt IN): 5: IBI Notification (I3C_EVT_IBI_NOTIFICATION) 6: Hot-Join Notification (I3C_EVT_HOTJOIN_NOTIFICATION) 7: Bus Error Notification (I3C_EVT_BUS_ERROR) 8 – 255: Reserved
1	result_count	For Execute Sequence (0): num_ops (number of operation results returned). On success, equals num_ops from request. On failure, returns the number of successful ops plus the first failed operation. For DAA (1): num_discovered_devices (total number of I3C devices assigned an address during this DAA cycle). For Get Stats (4): N (number of counter entries returned). For Bus Recovery (2) and Set Associated I2C (3): Reserved, set to 0.
N:2	Response Data Array	Command-specific data array (starts at offset Byte 2): For Execute Sequence (0): Sequential array of num_ops result descriptors (see structure below). For DAA (1): Array of num_discovered_devices 9-byte device structures (see below). For Get Stats (4): Array of N 3-byte counter entries (see below). For Bus Recovery (2) and Set Associated I2C (3): Omitted (no payload bytes after Byte 1).

Operation Result Descriptor Structure (for Execute Sequence)

Table 41: Operation Result Descriptor Structure (for Execute Sequence)

Byte Size	Field	Description
1	op_status	Execution status of this specific sub-operation enum 0: I3C_OP_SUCCESS, 1: I3C_OP_NACK, 2: I3C_OP_ARB_LOST, 3: I3C_OP_TIMEOUT, 4: I3C_OP_BUS_ERR
2	actual_len	Actual number of bytes successfully transferred (written or read) on the physical bus.
Variable	read_data	Raw data bytes retrieved from the target device. Present only if the corresponding request was a Read operation AND actual_len > 0. Omitted entirely for Write operations.

Discovered Device Structure for DAA**Table 42: Discovered Device Structure for DAA**

Byte Size	Field	Description
1	dynamic_addr	7-bit Dynamic Address successfully assigned to the device.
6	pid	48-bit Provisioned ID (MSB first).
1	bcr	Bus Characteristics Register.
1	dcr	Device Characteristics Register.

Counter Entry Structure (for Get Stats)**Table 43: Counter Entry Structure (for Get Stats)**

Byte Size	Field	Description
1	counter_type	Counter type enum 0: Arbitration Lost, 1: Invalid Broadcast Addr, 2: Parity Failure, 3: Write Parity Failure, 4: Assign Addr Parity Failure, 5: Illegal CCC Reply, 6: NACK, 7: Hot-Join Event, 8+: Vendor Unique counter
2	counter_value	16-bit unsigned counter value.

5.9. OBMF IPMI Channel

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. All use cases are supported by this specification.



Figure 18 Example of IPMI channel use case with OBMF Bridge.

IPMI Channel Roles:

Table 44: IPMI Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	No	Yes
Consumer	Yes	No

IPMI Channel-specific Status Codes are defined below:

Table 45: IPMI Channel-specific Status Codes

Value	Description
0x40 – 0x7F	Reserved

For the OBMF IPMI Channel, requests are initiated by the OBMF-ICP Consumer.

5.9.1. IPMI Channel Request

OBMF-ICP IPMI Channel Request message field definitions are as follows:

Table 46: OBMF-ICP IPMI Channel Request message

Bytes[Bits]	Field	Description
0	Command	The following transaction types are defined: 0: Send IPMI Message 1-255: Reserved
1	NetFn/LUN	As defined in IPMI Spec (differentiates Req from Resp)
N:2	IPMI Command + Payload	IPMI Command + Payload

5.9.2. IPMI Channel Response

OBMF-ICP IPMI Channel Response message field definitions are as follows:

Table 47: OBMF-ICP IPMI Channel Response message

Bytes[Bits]	Field	Description
0	Command	The following transaction types are defined: 0: Send IPMI Message 1-255: Reserved
1	NetFn/LUN	As defined in IPMI Spec (differentiates Req from Resp)
N:2	IPMI Command + Payload/Completion Code	IPMI Command + Payload/Completion Code

5.10. OBMF SPI Controller Channel

SPI Channel Roles:

Table 48: SPI Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	No	Yes
Consumer	Yes	No

SPI Channel-specific Status Codes are defined below:

Table 49: SPI Channel-specific Status Codes

Value	Description
0x40 – 0x7F	Reserved

For the OBMF SPI Channel, requests are initiated by the OBMF-ICP Consumer. When accessing multiple SPI devices that require different SPI controller configuration (e.g. different SPI Clock Speed) OBMF-ICP Consumer shall use Channel 0 SPI Controller Channel Configuration to reconfigure the SPI Controller Channel before accessing the device.

5.10.1. SPI Controller Channel Request

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. Although this specification supports generic SPI controller use cases, the OBMF SPI channel is not intended for booting Host from SPI Flash Boot Storage. For these use cases, OBMF-ICP defines the Memory Device Channel (5.12 OBMF Memory Device Channel).



Figure 19 Example of SPI channel use case with OBMF Bridge.

OBMF-ICP SPI Channel Request message field definitions are as follows:

Table 50: OBMF-ICP SPI Channel Request message

Bytes[Bits]	Field	Description
0	Command	[7:6]: CS Number 0: CS0 1: CS1 2: CS2 3: CS3 [5:4] CS Control [5]: 1 - Assert CS before SPI transfer [4]: 1 - De-assert CS after SPI transfer [3:0] SPI Command 1: SPI Read 2: SPI Write 3: SPI Write/Read 4: SPI Posted Write. Response is not expected. No flow control beyond OBMF-ICP Transport layer. 5 – 15: Reserved
2:1	Write Data Size	Size of Written Data
4:3	Read Data Size	Size of Read Data
N:5	Data	SPI Data to Write

5.10.2. SPI Controller Channel Response

OBMF-ICP SPI Channel Response message field definitions are as follows:

Table 51: OBMF-ICP SPI Channel Response message

Bytes[Bits]	Field	Description
0	Command	[7:4]: Reserved [3:0] SPI Command 1: SPI Read 2: SPI Write 3: SPI Write/Read 4: SPI Posted Write 5 – 15: Reserved
2:1	Read Data Size	Size of Read Data
N:3	Data	SPI Read Data

5.11. OBMF I/O Channel

For the OBMF I/O Channel, requests are initiated by the OBMF-ICP Consumer.

Table 52: OBMF I/O Channel

Bytes[Bits]	Field	Description
0[3:0]	OBMF-ICP Transaction	The following transaction types are defined: 0: Sequential Port I/O Read 8-bit 1: Sequential Port I/O Write 8-bit 2: Sequential Port I/O Read 16-bit 3: Sequential Port I/O Write 16-bit 4: Sequential Port I/O Read 32-bit 5: Sequential Port I/O Write 32-bit 6: Fixed Port I/O Read 8-bit 7: Fixed Port I/O Write 8-bit 8: Fixed Port I/O Read 16-bit 9: Fixed Port I/O Write 16-bit 10: Fixed Port I/O Read 32-bit 11: Fixed Port I/O Write 32-bit
0[7:4]	Reserved	Reserved
N:1	Payload	Payload related to each type of OBMF-ICP transaction as defined in subsequent sections. Note: Payload size is determined by the size of the transport layer message.

Note: The Fixed Port I/O Read/Write transaction is used to trigger multiple read/write on the same I/O address in a single OBMF packet. One typical usage is to support legacy I/O device with internal FIFO design accessed on a fixed address, like FIFO mode in 16550 UART controller.

For OBMF I/O Channel Response format is defined separately for each OBMF-ICP Transaction separately.

IO Channel Roles:

Table 53: IO Channel Roles

Device Role	OBMF Requester	OBMF Responder
Producer	No	Yes
Consumer	Yes	No

IO Channel-specific Status Codes are defined below:

Table 54: IO Channel-specific Status Codes

Value	Description
0x40	Address out-of-range
0x41	Access denied for privilege level
0x42 – 0x7F	Reserved

5.11.1. Sequential Port I/O Read Transaction 8-bit

This transaction requests the OBMF responder to perform a sequence of I/O port reads of size 8-bits (1 byte) starting at the port number indicated in the request and incremented for each byte. If size is 1 byte in total, then it is a single port access but with bigger size requests, two or more ports can be accessed. Compare to the fixed address transactions defined in sections 5.11.10 to 5.11.15 if multiple accesses to the same port number are necessary.

5.11.2. Sequential Port I/O Read 8-bit Request

OBMF-ICP I/O Read 8-bit Request message field definitions are as follows:

Table 55: OBMF-ICP I/O Read 8-bit Request message

Bytes[Bits]	Field	Description
1:0[15:0]	Port address	Starting Port Number of OBMF-ICP I/O Channel
2	Size	Read size in bytes. Maximum size is discovered using Channel 0 Discovery Structure

5.11.3. Sequential Port I/O Read 8-bit Response

OBMF-ICP I/O Read 8-bit Response message field definitions are as follows:

Table 56: OBMF-ICP I/O Read 8-bit Response message

Bytes[Bits]	Field	Description
(N-1):0	Data	If Status is equal to Success, this field carries Size bytes of the IO Data returned in response to the request Else if Status is different from Success, this field is omitted

5.11.4. Sequential Port I/O Port Write 8-bit Request

OBMF-ICP I/O Port Write 8-bit Request message field definitions are as follows:

Table 57: OBMF-ICP I/O Port Write 8-bit Request message

Bytes[Bits]	Field	Description
1:0 [15:0]	Port address	Starting Port Number OBMF-ICP Channel
2	Size	Write size in bytes. Maximum size is discovered using Channel 0 Discovery Structure
(3+Size-1):3	Data	Data payload related to Port Write transaction

5.11.5. Sequential Port I/O Port Write 8-bit Response

OBMF-ICP I/O Port Write 8-bit Response message field definitions are as follows:

Table 58: OBMF-ICP I/O Port Write 8-bit Response message

Bytes[Bits]	Field	Description
No Payload.		

5.11.6. Sequential Port I/O Read Transaction 16-bit

The format of the messages is the same as Sequential Port I/O Read 8-bit, except that the size of the data must be a multiple of 2 bytes and accesses are 16-bit (2-byte) wide.

5.11.7. Sequential Port I/O Write Transaction 16-bit

The format of the messages is the same as Sequential Port I/O Write 8-bit, except that the size of the data must be a multiple of 2 bytes and accesses are 16-bit (2-byte) wide.

5.11.8. Sequential Port I/O Read Transaction 32-bit

The format of the messages is the same as Sequential Port I/O Read 8-bit, except that the size of the data must be a multiple of 4 bytes and accesses are 32-bit (4-byte) wide.

5.11.9. Sequential Port I/O Write Transaction 32-bit

The format of the messages is the same as Sequential Port I/O Write 8-bit, except that the size of the data must be a multiple of 4 bytes and accesses are 32-bit (4-byte) wide.

5.11.10. Fixed Port I/O Read Transaction 8-bit

The format of the messages is the same as Sequential Port I/O Read 8-bit, except that all the accesses target the same address.

5.11.11. Fixed Port I/O Write Transaction 8-bit

The format of the messages is the same as Sequential Port I/O Write 8-bit, except that all the accesses target the same address.

5.11.12. Fixed Port I/O Read Transaction 16-bit

The format of the messages is the same as Fixed Port I/O Read 8-bit, except that the size of the data must be a multiple of 2 bytes and accesses are 16-bit (2-byte) wide.

5.11.13. Fixed Port I/O Write Transaction 16-bit

The format of the messages is the same as Fixed Port I/O Write 8-bit, except that the size of the data must be a multiple of 2 bytes and accesses are 16-bit (2-byte) wide.

5.11.14. Fixed Port I/O Read Transaction 32-bit

The format of the messages is the same as Fixed Port I/O Read 8-bit, except that the size of the data must be a multiple of 4 bytes and accesses are 32-bit (4-byte) wide.

5.11.15. Fixed Port I/O Write Transaction 32-bit

The format of the messages is the same as Fixed Port I/O Write 8-bit, except that the size of the data must be a multiple of 4 bytes and accesses are 32-bit (4-byte) wide.

5.12. OBMF Memory Device Channel

The example usage of this Channel with a discrete OBMF Bridge is shown in the figure below. Other usage may include an embedded OBMF endpoint integrated into an ASIC or Host SoC, which would not require a discrete Bridge. All use cases are supported by this specification.

The Memory Device Channel provides byte-addressable read, write, and erase access to a storage device exposed by the OBMF-ICP Producer. The channel supports two primary use cases:

1. **Host SoC Boot and Configuration Storage access** — The Host SoC accesses boot or configuration storage (typically Flash Storage) that is physically behind the BMC or emulated by the BMC. Examples include BIOS Flash, Firmware update, and BIOS/OS persistent configuration regions.

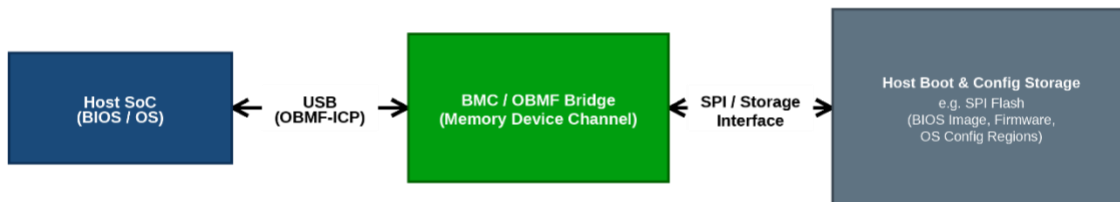


Figure 20 Example of Host SoC Boot and Configuration Storage access

2. **Generic Memory Device access** — An OBMF Device or Bridge exposes an arbitrary storage device through this channel. The channel definition is aligned with the Linux MTD (Memory Technology Device) driver model, enabling the BMC to treat the exposed storage behind OBMF Bridge as a standard memory device regardless of the underlying storage technology that could also include any virtual storage like RAM-based storage.

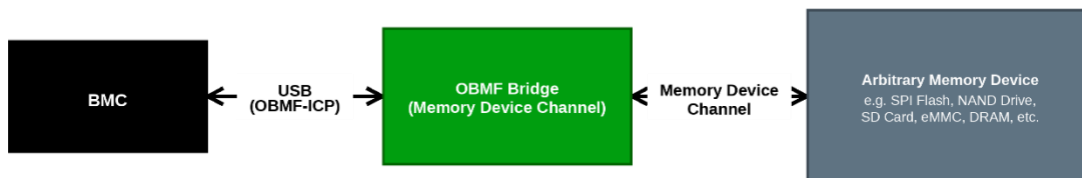


Figure 21 Example of Generic Memory Device access

Memory Device Channel Roles:

Table 59: Memory Device Channel Roles

Device Role	OBF Requester	OBF Responder
Producer	No	Yes
Consumer	Yes	No

Memory Device Channel -specific Status Codes are defined below:

Table 60: Memory Device Channel-specific Status Codes

Value	Description
0x40	Access denied for this address.
0x41	Unsupported transaction size.
0x42	Erase address or length is not aligned to the device erase block size.
0x43	Write or erase attempted on a write-protected device or region.
0x44 – 0x7F	Reserved.

5.12.1. Memory Device Channel Request

OBF-ICP Memory Device Channel Request message field definitions are as follows:

Table 61: OBF-ICP Memory Device Channel Request message

Bytes[Bits]	Field	Description
0[1:0]	Command	Command: 0: Memory Read 1: Memory Write 2: Memory Erase 3: Reserved
0[7:2]	Reserved	Reserved
4:1[31:0]	Address	32-bit byte offset into the device address space. No alignment requirement for Read or Write. For Erase, must be a multiple of ERASE_SIZE (expressed in KB).
6:5[15:0]	Data Size	Transfer size, unit depends on Command: - Read: size in bytes. Must not exceed READ_SIZE. - Write: size in bytes. Must not exceed WRITE_SIZE. Followed by Data Size bytes of write data. - Erase: size in KB. Must be a non-zero multiple of ERASE_SIZE. Examples: 0x0004 = 4 KB, 0x0040 = 64 KB, 0x0080 = 128 KB.
N:7	Data	Present only for Memory Write. Contains Data Size bytes of write payload.

5.12.2. Memory Device Channel Response

OBMF-ICP Memory Device Channel Response message field definitions are as follows:

Table 62: OBMF-ICP Memory Device Channel Response message

Bytes[Bits]	Field	Description
(N-1):0	Data	If Status is equal to Success and Command is 0 (read), this field carries <i>Data Size</i> bytes of data returned in response to the request Else if Status is different from Success or for other commands, this field is omitted

5.13. OBMF TPM Channel

OBMF TPM Channel is a dedicated channel for accessing Trusted Platform Module Register Space as defined by TCG PC Client Platform TPM Profile (PTP) Specification for TPM 2.0.

OBMF TPM channel is using MMIO channel access methods as defined in 5.3 OBMF MMIO Channel.

5.14. OBMF RTC Channel

OBMF RTC Channel is a dedicated channel for accessing Real Time Clock Register space. RTC register space is not standardized by any standardization body and might differ between RTC vendors, but the reference definition has been published by OBMF-ICP WG on OBMF-ICP GitHub (<https://github.com/opencomputeproject/ocp-obmf/wiki>).

OBMF RTC channel is using MMIO channel access methods as defined in 5.3 OBMF MMIO Channel.

5.15. OBMF-ICP Control Channel – Channel 0

OBMF-ICP control channel – Channel 0 is using subset of MMIO channel access methods. Channel 0 is a special Channel Type dedicated for OBMF-ICP Device discovery and configuration. It leverages the MMIO Short Read/Write transactions for accessing this channel in order to avoid duplication in OBMF-ICP Request/Response definitions. OBMF-ICP Device shall support MMIO Short Read and Short Write with a minimum Request and Response payload size of 64 bytes in order to comply with OBMF-ICP and to allow Channel 0 discovery. This 64-byte minimum is a special requirement for Channel 0 and MMIO Short transactions. It is not required for all MMIO channels which are exposed separately with separate MMIO channel capabilities. OBMF-ICP Device does not have to support all MMIO Channel commands if it does not support MMIO Channel Types but Short Read/Write with min 64 byte payload is mandatory in order to support Channel 0 operations.

This structure must be provided by the OBMF-ICP Device on Channel 0. The OBMF-ICP Host (OBMF Consumer on Channel 0) can read this content to obtain the list of all the other channels that the OBMF-ICP Device makes available or requests support from OBMF-ICP Host to the OBMF-ICP Host. The OBMF-ICP Host should enable all the necessary channels by writing to the “Enable” fields in this data structure. In addition to the channel list, this discovery structure allows OBMF-ICP endpoints to establish OBMF-ICP protocol version compatibility and other capabilities.

The OBMF-ICP Devices can support optional Manufacturing Mode which allows for configuration of OBMF-ICP channels in Manufacturing and storing/locking the configuration in non-volatile memory. This configuration will be applied after reset. The runtime configuration can still be applied afterwards. The manufacturing and runtime configuration flows are presented on Figure 22 below.

The following describes the Channel #0 configuration and lock flows are defined as follows

A. Reset / Boot entry

1. Channel #0 is always enabled for discovery and configuration.
2. Check Manufacturing Lock support first (MANUF_CONFIG_LOCK_SUPPORTED).
 - a. If the device does not support Manufacturing Mode: it always boots with MANUF_CONFIG_VALID = 0 and RUNTIME_CONFIG_APPLIED = 0, has no persistent copy, and goes directly to the runtime path. Lock requests will be ignored by this device.
 - b. If the device supports Manufacturing Mode, it loads any persistent (manufacturing) configuration and sets CONFIG_STATUS[MANUF_CONFIG_VALID] = 1 if a valid copy exists or enters Manufacturing Configuration if MANUF_CONFIG_LOCKED is not set.

Manufacturing configuration (device unlocked)

1. Configure all channels. Write each channel's CONFIGURATION_DATA and CHANNEL_CONTROL (enable, channel-specific settings, etc.).
2. Apply per channel APPLY_CONFIGURATION to request device to use current runtime config and apply. The bit self-clears when the apply completes.
3. Validate if configuration was applied correctly (CONFIGURATION_VALID and HEALTH_STATUS). If the configuration was rejected (CONFIGURATION_VALID = 0, HEALTH_STATUS = 3 or 15, CHANNEL_ERROR_INDICATOR = 1), correct it and re-apply before continuing.
4. Select which channels to lock. On each channel that should be locked, set MANUF_CONFIG_LOCK_REQUEST. Leave it = 0 on any channel that should stay changeable (e.g. for manufacturing test or later rework).
5. Commit lock globally. Set MANUF_CONFIG_LOCK_REQUEST. The device writes the selected channels to persistent storage and the request bit self-clears. Selected channels get CONFIGURATION_LOCKED set. Globally MANUF_CONFIG_LOCKED is not set until all channels are locked while MANUF_CONFIG_VALID shall be set if at least one channel is locked.
6. Manufacturing complete when all channels are locked then MANUF_CONFIG_LOCKED is set and manufacturing mode is inactive. On the next reset the device re-enters Channel #0 as a committed (field) device and takes the runtime path.

Note: OBMF-ICP does not define how the device could reenter the Manufacturing Mode once it is locked. It is up to device vendor to determine whether it shall be supported and what are the methods to reenter the manufacturing mode while meeting vendor's security threat model and other requirements. OBMF-ICP defines the Vendor specific placeholders in the Channel #0 definition to allow for such solutions. If OBMF-ICP Manufacturing Configuration is stored on Flash storage by OBMF-ICP device, Vendor shall support a method to Recover the content of that storage. The definition of Recovery flow is out of scope of the OBMF-ICP specification.

Runtime configuration (committed device, non-persistent)

1. A committed device runs the persistent configuration (MANUF_CONFIG_VALID) or some vendor-specific default configuration is applied for devices that do not support configuration locking
2. A new CONFIGURATION_DATA can be written to device at this stage however these writes are buffered. The device keeps running the previously applied configuration and the validity bits do not change yet.
3. Once configuration is set the APPLY_CONFIGURATION bit is used to apply runtime configuration.
4. A combination of CONFIGURATION_VALID and HEALTH_STATUS shall be used to determine if configuration was applied successfully or rejected and needs to be changed and reapplied. Once all applied channels report as valid, RUNTIME_CONFIG_APPLIED is set.
5. Runtime operations. Enable/disable, reset, re-configure, and clear errors are available anytime via CHANNEL_CONTROL while the channel is enabled.
6. On every OBMF/interface reset, runtime overrides are discarded and the device reverts to the persistent configuration (RUNTIME_CONFIG_APPLIED is cleared).

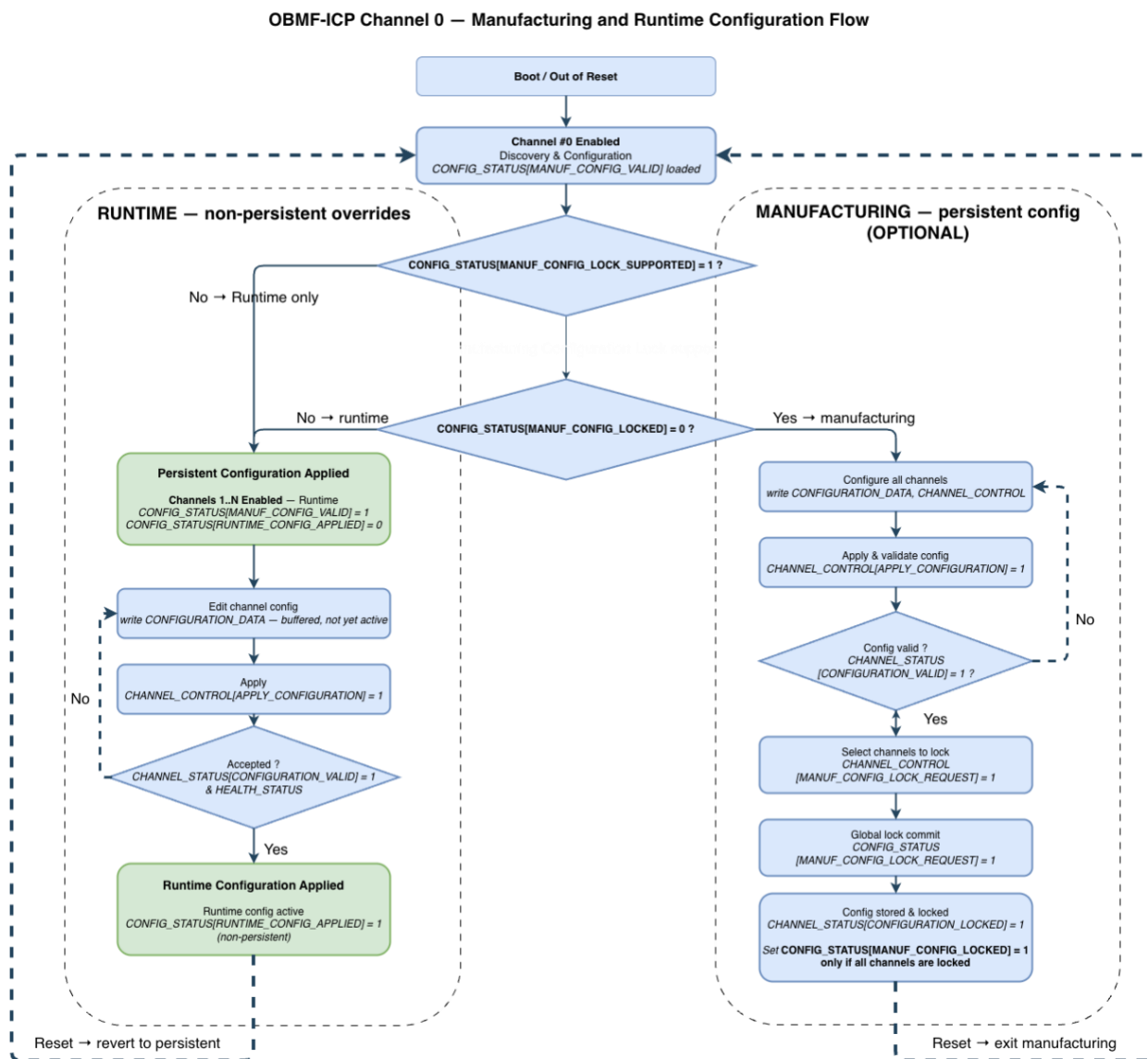


Figure 22 OBMF Manufacturing and Runtime Configuration Flow

Channel 0 Tables convention (“Acc” stands for Access Type):

Table 63: Channel 0 Tables Convention

Bits	Field	Acc	Description
Offset	REGISTER_NAME	Register size in bytes e.g. 2B	
Byte [bits range]	BITFIELD_NAME	r – read-only rw – read-write	Description of bitfield.

Channel 0 MMIO Register Table

Table 64: Channel 0 MMIO Register Table

Bits	Field	Acc	Description
0x00 OBMF_VER 2B			
0[7:0]	VERSION_MINOR_BCD	r	Minor protocol version, BCD encoded. 0 - Version 1.0 minor value for this specification.
1[7:0]	VERSION_MAJOR_BCD	r	Major protocol version, BCD encoded. 1 - Version 1 major value for this specification.
0x02 VENDOR_ID 2B			
0-1[15:0]	VENDOR_ID	r	PCI-SIG Vendor ID of the device or subsystem vendor. Use 0xFFFF only when no PCI-SIG Vendor ID is available.
0x04 DEVICE_ID 2B			
0-1[15:0]	DEVICE_ID	r	Vendor-assigned device ID, preferably aligned with the PCIe Device ID when the device has one.
0x06 OBMF_DEVICE_NAME 32B			
0-31[255:0]	OBMF_DEVICE_NAME	r	UTF-8 encoded, NUL-padded device name. If actual name is longer than 31 bytes, truncate and NUL-terminate.
0x26 NUM_OF_CHANNELS 1B			
[7:0]	NUM_OF_CHANNELS	r	Number of channel offset entries populated below. Channel 0 is not included in this count.
0x27 CONFIG_STATUS 1B			
[0]	MANUF_CONFIG_LOCK_SUPPORTED	r	Manufacturing Mode / configuration locking capability (read-only). 0 - Not supported. Per-channel and global MANUF_CONFIG_LOCK_REQUEST have no effect, no persistent configuration copy exists, and the device always boots MANUF_CONFIG_VALID = 0 and RUNTIME_CONFIG_APPLIED = 0. 1 - Supported. The device can store and lock configuration in persistent (manufacturing) storage.
[1]	MANUF_CONFIG_LOCKED	r	Global configuration lock status. Blocks persistent (manufacturing) writes. Volatile runtime overrides remain permitted until reset. When set, manufacturing mode is no longer active. 0 - Global configuration is not locked. 1 - Global configuration is locked. No further persistent writes are allowed.
[2]	MANUF_CONFIG_LOCK_REQUEST	rw	Global commit trigger. Write-1, self-clearing. Read as 0. 0 - No effect. 1 - Write to persistent storage only the channels whose per-channel MANUF_CONFIG_LOCK_REQUEST = 1.
[3]	MANUF_CONFIG_VALID	r	Persistent configuration status. 0 - Persistent Manufacturing configuration is not valid or was not loaded. 1 - Persistent Manufacturing configuration was loaded successfully.
[4]	RUNTIME_CONFIG_APPLIED	r	Runtime configuration status. Set to 1 only after a runtime configuration has been applied and validated;

Bits	Field	Acc	Description
			0 otherwise, including a rejected apply. Cleared on reset. 0 - No runtime configuration applied. 1 - A runtime configuration has been applied and validated (each enabled channel that received an apply reports CONFIGURATION_VALID = 1 with no CHANNEL_ERROR_INDICATOR).
[7:5]	RESERVED	rw	Reserved. Read as 0, write as 0.
0x28 VENDOR_CONFIGURATION_OFFSET 4B			
0-3[31:0]	VENDOR_CONFIGURATION_OFFSET	r	Offset to Vendor-defined configuration region. Set to 0 if Vendor-defined Configuration region is not supported.
0x2C CHANNEL_OFFSET 4B			
0-3[31:0]	CHANNEL_OFFSET	r	32-bit byte offset from the base of this Channel 0 MMIO configuration memory space to the first Channel configuration space common header. 0x00000000 means absent or unimplemented.
0x30 CHANNEL_OFFSET 4B			
0-3[31:0]	CHANNEL_OFFSET	r	Same definition as above.
0x34 CHANNEL_OFFSET 4B			
0-3[31:0]	CHANNEL_OFFSET	r	Same definition as above.
0x38 CHANNEL_OFFSET 4B			
0-3[31:0]	CHANNEL_OFFSET	r	Same definition as above.
0x3C CHANNEL_OFFSET 4B			
0-3[31:0]	CHANNEL_OFFSET	r	Same definition as above.
0x2C + 4*(N-1) Nth CHANNEL_OFFSET 4B			
0-3[31:0]	Nth CHANNEL_OFFSET	r	Continuation for N = 1..NUM_OF_CHANNELS. Each offset points to a channel configuration space in this same MMIO memory space. Entries must not overlap.

Each channel configuration space pointed to by CHANNEL_N_OFFSET starts with this common header. Offsets are relative to the channel configuration space base address.

Channel Configuration Space Common Header

Table 65: Channel Configuration Space Common Header

Bits	Field	Acc	Description
0x00 CHANNEL_TYPE 1B			
[7:0]	CHANNEL_TYPE	r	OBFM-ICP Channel Type: 00h - Configuration (Reserved for Channel #0) 01h - MMIO 02h - GPIO 03h - Serial 04h - I2C Controller 05h - I2C Target 06h - I3C Controller 07h - IPMI 08h - SPI Controller 09h - IO 0Ah - Memory Device 0Bh - TPM 0Ch - RTC

Bits	Field	Acc	Description
			0Dh - F7h – Reserved F8h - FFh – OEM
0x01 CHANNEL_NUMBER 1B			
[7:0]	CHANNEL_NUMBER	r	Channel number matching the CHANNEL_N_OFFSET table entry. Channel 0 is reserved for management and configuration and should not appear here.
0x02 CHANNEL_NAME 16B			
0-15[127:0]	CHANNEL_NAME	rw	UTF-8, NULL-padded channel name. OBMF-ICP defines 16 bytes for the base channel name.
0x12 DEVICE_ROLE 1B			
[7:0]	DEVICE_ROLE	r	Indicates the data-flow role of this channel: 0 - Producer. This channel is produced by the OBMF Device and consumed by the OBMF Host. 1 - Consumer. This channel is produced by the OBMF Host and consumed by the OBMF Device. 2 - Symmetric. This channel is fully symmetric and the device plays both the roles of the OBMF-ICP Producer and Consumer. 2-255 - Reserved.
0x13 CHANNEL_STATUS 1B			
[3:0]	HEALTH_STATUS	r	Generic channel health status. 0 - Unknown or uninitialized. 1 - Status OK Channel is Ready. 2 - Channel Disabled. 3 - Channel not ready, configuration not present or configuration error. 4 - 14 - Reserved 15 - Channel error.
[4]	CHANNEL_RESET_DONE	r	Generic reset completion status. 0 - Reset not complete or no reset requested. 1 - Reset completed.
[5]	CONFIGURATION_VALID	r	Configuration Data validity. 0 - Configuration Data is not valid. 1 - Configuration Data is valid.
[6]	CONFIGURATION_LOCKED	r	Persistent per-channel configuration lock status. 0 - Persistent channel configuration is not locked. 1 - Persistent channel configuration is locked. Runtime overrides may still be permitted.
[7]	CHANNEL_ERROR_INDICATOR	r	Generic channel error indicator. 0 - No channel error. 1 - Channel error present; see CHANNEL_SPECIFIC_STATUS where applicable.
0x14 CHANNEL_CONTROL 1B			
[0]	CHANNEL_ENABLE	rw	Generic channel enable control. 0 - Channel disabled. 1 - Channel enabled.
[1]	CHANNEL_RESET	rw	Generic channel reset request. 0 - No reset requested. 1 - Request channel reset. The bit is write-1 and self-clearing; completion is reported by CHANNEL_STATUS[CHANNEL_RESET_DONE].
[2]	APPLY_CONFIGURATION	rw	Apply channel Configuration Data. 0 - No apply request.

Bits	Field	Acc	Description
			1 - Apply the current Configuration Data for this channel. The bit is write-1 and self-clearing; completion is reported by CHANNEL_STATUS[CONFIGURATION_VALID].
[3]	MANUF_CONFIG_LOCK_REQUEST	rw	Per-channel lock selection. Write-1, self-clearing after global commit request (CONFIG_STATUS[MANUF_CONFIG_LOCK_REQUEST]). Writing 1 marks this channel to be written to persistent storage at the next global commit (CONFIG_STATUS[MANUF_CONFIG_LOCK_REQUEST] = 1). The resulting per-channel lock status is reported by CHANNEL_STATUS[CONFIGURATION_LOCKED]. 0 - Channel not selected for locking. 1 - Select this channel to be written to persistent storage at global commit.
[4]	CLEAR_CHANNEL_ERROR	rw	Clear generic channel error status. 0 - No effect. 1 - Request clearing CHANNEL_ERROR and CHANNEL_ERROR_INDICATOR when the underlying error condition is no longer present.
[7:5]	RESERVED	rw	Reserved. Read as 0, write as 0.
0x15 CHANNEL_SPECIFIC_STATUS 1B			
[7:0]	CHANNEL_SPECIFIC_STATUS	r	Channel-specific status byte. Bit assignments are defined by the channel type tables below.
0x16 CHANNEL_SPECIFIC_CONTROL 1B			
[7:0]	CHANNEL_SPECIFIC_CONTROL	rw	Channel-specific control byte. Bit assignments are defined by the channel type tables below. Use read-modify-write for individual control bit updates.
0x17 MAX_REQUEST_PAYLOAD_SIZE 2B			
0-1[15:0]	MAX_REQUEST_PAYLOAD_SIZE	r	Defines the maximum size of a Request for this channel. Can also be used as an indication of the buffer sizes supported by the device for this channel.
0x19 MAX_RESPONSE_PAYLOAD_SIZE 2B			
0-1[15:0]	MAX_RESPONSE_PAYLOAD_SIZE	r	Defines the maximum size of a Response for this channel. Can also be used as an indication of the buffer sizes supported by the device for this channel.
0x1B RESERVED 1B			
[7:0]	RESERVED	r	Reserved for future common-header expansion.
0x1C CONFIG_SIZE 4B			
0-3[31:0]	CONFIG_SIZE	r	Size in bytes of the channel-specific Configuration Data at CONFIGURATION_DATA. Empty configuration spaces set this to 0.
0x20 CONFIGURATION_DATA N			
0...N	CONFIGURATION_DATA	rw	Channel-specific Configuration Data. Layout is defined per channel below and starts at this offset in the channel MMIO configuration space.

5.15.1. Channel Specific Status and Control Registers

This chapter defines the channel-specific status and control bytes (CHANNEL_SPECIFIC_STATUS and CHANNEL_SPECIFIC_CONTROL) in each channel configuration space.

MMIO Channel Control and Status Registers

Table 66: MMIO Channel Control and Status Registers

Bits	Field	Acc	Description
0x00	MMIO_STATUS 1B		
[0]	MMIO_WINDOW_READY	r	MMIO window is configured and ready.
[1]	MMIO_ACCESS_ERROR	r	Sticky status. Last MMIO channel operation reported an access error.
[2]	MMIO_WINDOW_SIZE_ERROR	rw	MMIO window size is too small - cannot be used by the OBMF-ICP Consumer
[7:3]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01	MMIO_CONTROL 1B		
[0]	CLEAR_STATUS	rw	Write 1 to request clearing sticky MMIO status bits.
[7:1]	RESERVED	rw	Reserved. Read as 0, write as 0.

GPIO Channel Control and Status Registers

Table 67: GPIO Channel Control and Status Registers

Bits	Field	Acc	Description
0x00	GPIO_STATUS 1B		
[0]	INDEX_ERROR	r	One or more GPIO entries contain an unsupported, duplicate, or out-of-range GPIO_INDEX.
[1]	INTERRUPT_PENDING	r	One or more GPIO interrupt notifications are pending.
[2]	OUTPUT_PERSIST_ACTIVE	r	One or more GPIO persistent output states are active.
[7:3]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01	GPIO_CONTROL 1B		
[0]	CLEAR_STATUS	rw	Write 1 to request clearing sticky GPIO status bits.
[7:1]	RESERVED	rw	Reserved. Read as 0, write as 0.

Serial Channel Control and Status Registers

Table 68: Serial Channel Control and Status Registers

Bits	Field	Acc	Description
0x00	SERIAL_STATUS 1B		
[7:0]	RESERVED	r	Reserved. UART frame-level status and error events are reported by the UART frame definition. Serial line override state is read through SERIAL_CONTROL.
0x01	SERIAL_CONTROL 1B		
[0]	RX_STATE	rw	RW. Read returns the current RX control state. Write sets the RX manual override value; effective only when RX_OVERRIDE_ENABLE is set.

Bits	Field	Acc	Description
			0 - Drive RX low. 1 - Release RX/high.
[1]	TX_STATE	rw	RW. Read returns the current TX control state. Write sets the TX manual override value; effective only when TX_OVERRIDE_ENABLE is set. 0 - Drive TX low. 1 - Release TX/high.
[2]	RX_OVERRIDE_ENABLE	rw	RX line override enable. 0 - RX override disabled. 1 - RX override enabled.
[3]	TX_OVERRIDE_ENABLE	rw	TX line override enable. 0 - TX override disabled. 1 - TX override enabled.
[7:4]	RESERVED	rw	Reserved. Read as 0, write as 0.

I2C Controller Channel Control and Status Registers

Table 69: I2C Controller Channel Control and Status Registers

Bits	Field	Acc	Description
0x00 I2C_CONTROLLER_STATUS 1B			
[0]	SDA_LOW_TIMEOUT	r	Sticky status. SDA remained low beyond the implementation-defined timeout.
[1]	SCL_LOW_TIMEOUT	r	Sticky status. SCL remained low beyond the implementation-defined timeout.
[2]	BUS_RECOVERY_FAILED	r	Sticky status. Last bus recovery attempt failed.
[3]	BUS_RECOVERY_SUCCESS	r	Sticky status. Last bus recovery attempt completed successfully.
[7:4]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01 I2C_CONTROLLER_CONTROL 1B			
[0]	SDA_STATE	rw	RW. Read returns the current SDA control state. Write sets the SDA manual override value; effective only when SDA_OVERRIDE_ENABLE is set. 0 - Drive SDA low. 1 - Release SDA/high.
[1]	SCL_STATE	rw	RW. Read returns the current SCL control state. Write sets the SCL manual override value; effective only when SCL_OVERRIDE_ENABLE is set. 0 - Drive SCL low. 1 - Release SCL/high.
[2]	TRIGGER_BUS_RECOVERY	rw	Write 1 to trigger I2C bus recovery.
[3]	CLEAR_STATUS	rw	Write 1 to request clearing sticky I2C Controller Channel status bits.
[4]	AUTO_BUS_RECOVERY_ENABLE	rw	Enable automatic bus recovery when SDA_LOW_TIMEOUT is detected. 0 - Automatic bus recovery disabled. 1 - Automatically trigger bus recovery on SDA low timeout.
[5]	SDA_OVERRIDE_ENABLE	rw	SDA manual override enable. 0 - SDA override disabled. 1 - SDA override enabled; SDA_STATE controls the SDA line.

Bits	Field	Acc	Description
[6]	SCL_OVERRIDE_ENABLE	rw	SCL manual override enable. 0 - SCL override disabled. 1 - SCL override enabled; SCL_STATE controls the SCL line.
[7]	RESERVED	rw	Reserved. Read as 0, write as 0.

I2C Target Channel Control and Status Registers

Table 70: I2C Target Channel Control and Status Registers

Bits	Field	Acc	Description
0x00 I2C_TARGET_STATUS 1B			
[0]	ADDRESS_MATCH_EVENT	r	Sticky status. One of the configured target addresses matched.
[1]	ADDRESS_LIST_ERROR	r	Sticky status. Target address list contains an invalid or unsupported address.
[2]	TARGET_BUFFER_OVERRUN	r	Sticky status. Target receive buffer overrun occurred.
[7:3]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01 I2C_TARGET_CONTROL 1B			
[0]	TARGET_ENABLE	rw	Enable target address matching. 0 - Target matching disabled. 1 - Target matching enabled.
[1]	CLEAR_STATUS	rw	Write 1 to request clearing sticky I2C Target Channel status bits.
[7:2]	RESERVED	rw	Reserved. Read as 0, write as 0.

I3C Controller Channel Control and Status Registers

Table 71: I3C Controller Channel Control and Status Registers

Bits	Field	Acc	Description
0x00 I3C_CONTROLLER_STATUS 1B			
[0]	BUS_ERROR_DETECTED	r	Sticky status. Physical bus fault or framing error detected.
[1]	PARITY_ERROR_DETECTED	r	Sticky status. Parity error detected during command or DAA.
[2]	BUS_RECOVERY_FAILED	r	Sticky status. The last bus recovery attempt failed.
[3]	BUS_RECOVERY_SUCCESS	r	Sticky status. The last bus recovery attempt was completed successfully.
[7:4]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01 I3C_CONTROLLER_CONTROL 1B			
[0]	CLEAR_STATUS	rw	Write 1 to request clearing sticky bits in I3C_CONTROLLER_STATUS.
[7:1]	RESERVED	rw	Reserved. Read as 0, write as 0.

IPMI Channel Control and Status Registers

Table 72: IPMI Channel Control and Status Registers

Bits	Field	Acc	Description
0x00 IPMI_STATUS 1B			
[0]	IPMI_READY	r	IPMI channel-specific service is ready.

Bits	Field	Acc	Description
[1]	IPMI_INTERFACE_ERROR	r	Sticky status. IPMI channel-specific service reported an interface error.
[7:2]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01 IPMI_CONTROL 1B			
[0]	CLEAR_STATUS	rw	Write 1 to request clearing sticky IPMI status bits.
[7:1]	RESERVED	rw	Reserved. Read as 0, write as 0.

SPI Controller Channel Control and Status Registers

Table 73: SPI Controller Channel Control and Status Registers

Bits	Field	Acc	Description
0x00 SPI_CONTROLLER_STATUS 1B			
[0]	MODE_UNSUPPORTED	r	Requested SPI configuration setting or flag combination is not supported.
[1]	TRANSFER_ERROR	r	Last SPI controller transaction reported a channel-level transfer error.
[7:2]	RESERVED	r	Reserved. Read as 0, write as 0.
0x01 SPI_CONTROLLER_CONTROL 1B			
[0]	CLEAR_STATUS	rw	Write 1 to request clearing sticky SPI Controller Channel status bits.
[7:1]	RESERVED	rw	Reserved. Read as 0, write as 0.

Memory Device Channel Control and Status Registers

Table 74: Memory Device Channel Control and Status Registers

Bits	Field	Acc	Description
0x00 MEM_DEV_STATUS 1B			
[0]	WRITE_PROTECTED	r	Device is currently write-protected (hardware or software lock active). Reflects live protection state; may differ from the WRITEABLE capability flag in MEM_DEV_CFG if protection changes at runtime.
[1]	MEDIA_ERROR	r	Sticky. Last operation reported a device-level media error (e.g. uncorrectable ECC, internal hardware failure). Cleared by writing CLEAR_STATUS.
[7:2]	RESERVED	r	Read as 0.
0x01 MEM_DEV_CONTROL 1B			
[0]	CLEAR_STATUS	rw	Write 1 to clear the sticky MEDIA_ERROR status bit. Self-clearing.
[7:1]	RESERVED	rw	Write as 0.

5.15.2. Channel Configuration Data

This chapter defines the channel-specific Configuration Data (CONFIGURATION_DATA) in each channel configuration space.

MMIO Channel Configuration Data

Table 75: MMIO Channel Configuration Data

Bits	Field	Acc	Description
0x00 MAX_MMIO_READ_SIZE 2B			
0-1[15:0]	MAX_MMIO_READ_SIZE	rw	Maximum MMIO read transfer size in bytes supported by this MMIO Channel, capped to a 16-bit value.
0x02 MAX_MMIO_WRITE_SIZE 2B			
0-1[15:0]	MAX_MMIO_WRITE_SIZE	rw	Maximum MMIO write transfer size in bytes supported by this MMIO Channel, capped to a 16-bit value.
0x04 MMIO_WINDOW_BASE_ADDRESS 4B			
0-3[31:0]	MMIO_WINDOW_BASE_ADDRESS	r	Base Address of the MMIO window
0x08 MMIO_WINDOW_SIZE 4B			
0-3[31:0]	MMIO_WINDOW_SIZE	r	Desired size (in bytes) of the MMIO window

GPIO Channel Configuration Data

Table 76: GPIO Channel Configuration Data

Bits	Field	Acc	Description
0x00 GPIO_CONFIGURATION_HEADER 2B			
0-1[15:0]	GPIO_COUNT	r	Total number of GPIO entries in GPIO_CONFIGURATION_ARRAY. This field is 2 bytes to support up to 4096 GPIOs. 0 - No GPIOs are exposed. 1-4096 - Number of GPIO entries. 4097-65535 - Reserved.
0x02 GPIO_CONFIGURATION_ARRAY 50B * GPIO_COUNT			
0...	GPIO_CONFIGURATION_ARRAY	rw	Array of 50-byte GPIO configuration entries. GPIO entry i starts at 0x02 + 0x32*i, where i is zero-based. Each entry includes a programmable GPIO_INDEX.

GPIO Configuration Entry

Table 77: GPIO Configuration Entry

Bits	Field	Acc	Description
0x00 GPIO_INDEX 2B			
0-1[11:0]	GPIO_INDEX	rw	Programmable GPIO index used by GPIO channel commands. The value identifies the GPIO represented by this entry.
0-1[15:12]	RESERVED	r	Read as 0, write as 0.
0x02 GPIO_NAME 32B			
0-31[255:0]	GPIO_NAME	rw	UTF-8, NUL-padded GPIO name.
0x22 GPIO_DIRECTION 1B			
[7:0]	GPIO_DIRECTION	rw	0x00 = Output 0x01 = Input
0x23 GPIO_DEFAULT_OUTPUT 1B			
[7:0]	GPIO_DEFAULT_OUTPUT	rw	0x00 = Low 0x01 = High
0x24 GPIO_OUTPUT_DRIVE_CONFIG 1B			
[7:0]	GPIO_OUTPUT_DRIVE_CONFIG	rw	0x00 = Push/Pull (Hi ↔ Lo) 0x01 = Open Drain (Z ↔ Lo) 0x02 = Open Source (Hi ↔ Z)
0x25 GPIO_OUTPUT_PERSIST_STATE 1B			
[7:0]	GPIO_OUTPUT_PERSIST_STATE	rw	0x00 = False 0x01 = True

Bits	Field	Acc	Description
			OUTPUT_PERSIST_STATE governs whether the GPIO output setting from the "SET_VALUE" command will persist through a device reset. (No glitches or transient state changes are allowed)
0x26	GPIO_BIAS_PULL_CONFIG 1B		
[7:0]	GPIO_BIAS_PULL_CONFIG	rw	0x00 = No Pull 0x01 = Pull Up 0x02 = Pull Down
0x27	GPIO_BIAS_PULL_STRENGTH 2B		
0-1[15:0]	GPIO_BIAS_PULL_STRENGTH	rw	Value in 16Ω 0x0000 = Not specified 0x0001 = 16Ω (Strongest) 0xFFFF ≈ 1MΩ (Weakest)
0x29	GPIO_OUTPUT_DRIVE_STRENGTH 2B		
0-1[15:0]	GPIO_OUTPUT_DRIVE_STRENGTH	rw	Drive strength in mA 0x0000 = Not specified 0x0001 = 1mA (Weakest) 0xFFFF = 65535mA (Strongest)
0x2B	GPIO_SLEW_RATE 2B		
0-1[15:0]	GPIO_SLEW_RATE	rw	Value in 0.1 ns 0x0000 = Not specified 0x0001 = 0.1ns (Fastest) 0xFFFF ≈ 6554ns (Slowest)
0x2D	GPIO_INPUT_DEBOUNCE_TIME 2B		
0-1[15:0]	GPIO_INPUT_DEBOUNCE_TIME	rw	Debounce time in μs 0x0000 = Not specified 0x0001 = 1μs (Shortest) 0xFFFF = 65535μs (Longest)
0x2F	RESERVED 3B		
0-2[23:0]	RESERVED	r	Reserved for alignment. Read as 0, write as 0.

Serial Channel Configuration Data

Table 78: Serial Channel Configuration Data

Bits	Field	Acc	Description
0x00	UART_SPEED_HZ 4B		
0-3[31:0]	UART_SPEED_HZ	rw	Baud rate in Hz, unsigned 32-bit little endian.
0x04	UART_DATA_BITS 1B		
[7:0]	UART_DATA_BITS	rw	Number of data bits.
0x05	UART_STOP_BITS 1B		
[7:0]	UART_STOP_BITS	rw	Number of stop bits.
0x06	UART_LINE_CONTROL 1B		
[3:0]	UART_PARITY	rw	UART parity. 0 - None. 1 - Odd. 2 - Even. 3 - Mark. 4 - Space.

Bits	Field	Acc	Description
[7:4]	UART_FLOW_CONTROL	rw	UART flow control. 0 - None. 1 - XON/XOFF. 2 - RTS/CTS. 3 - DSR/DTR.

I2C Controller Channel Configuration Data

Table 79: I2C Controller Channel Configuration Data

Bits	Field	Acc	Description
0x00 I2C_CONTROLLER_SPEED 1B			
[7:0]	I2C_CONTROLLER_SPEED	rw	I2C bus speed. 0 - 100 kHz. 1 - 400 kHz. 2 - 1 MHz. 3 - 3.4 MHz. 4-127 - Reserved.
0x01 I2C_CONTROLLER_OPTIONS 1B			
[0]	BUS_RECOVERY_SUPPORTED	r	Indicates whether device supports Bus Recovery via TRIGGER_BUS_RECOVERY. 0 - Setting not supported. 1 - Setting supported.
[1]	AUTO_BUS_RECOVERY_SUPPORTED	r	Indicates whether device supports Automatic Bus Recovery via TRIGGER_BUS_RECOVERY. 0 - Setting not supported. 1 - Setting supported.
[2]	BUS_OVERRIDE_SUPPORTED	r	Indicates whether device supports Bus Override via SDA_OVERRIDE_ENABLE and SCL_OVERRIDE_ENABLE. 0 - Setting not supported. 1 - Setting supported.
[7:3]	RESERVED	r	Reserved for alignment. Read as 0, write as 0.

I2C Target Channel Configuration Data

Table 80: I2C Target Channel Configuration Data

Bits	Field	Acc	Description
0x00 I2C_TARGET_SPEED 1B			
[7:0]	I2C_TARGET_SPEED	rw	I2C target bus speed capability. 0 - 100 kHz. 1 - 400 kHz. 2 - 1 MHz. 3 - 3.4 MHz. 4-127 - Reserved.
0x01 I2C_TARGET_BUFFER_SIZE 1B			
[7:0]	I2C_TARGET_BUFFER_SIZE	rw	I2C Target Buffer Size in Bytes, this is the maximum supported length of I2C Write Transaction. It is applicable to all supported addresses.
0x02 I2C_TARGET_ADDRESS_COUNT 1B			

Bits	Field	Acc	Description
[7:0]	I2C_TARGET_ADDRESS_COUNT	r	Number of configurable I2C target address entries supported by this channel.
0x03	I2C_TARGET_ADDRESS_LIST	1B * I2C_TARGET_ADDRESS_COUNT	
0...	I2C_TARGET_ADDRESS_LIST	rw	Array of 1-byte configurable I2C target address entries. Size of array in bytes is I2C_TARGET_ADDRESS_COUNT.

I3C Controller Channel Configuration Data

Table 81: I3C Controller Channel Configuration Data

Bits	Field	Acc	Description
0x00	DEV_NACK_RETRY_COUNT	2B	
0-1[15:0]	DEV_NACK_RETRY_COUNT	rw	Configured hardware retry count when a target NACKs ('0' = default).
0x02	I3C_FEATURE_FLAGS	1B	
[7:0]	I3C_FEATURE_FLAGS	r	Bitmask of hardware supported features: • Bit 0: IBI private read support. • Bits 1–7: Reserved,
0x03	MAX_CHAINED_OPS	1B	
[7:0]	MAX_CHAINED_OPS	r	Maximum number of chained operations supported in 'I3C_CMD_EXECUTE_SEQUENCE'.
0x04	MAX_SCL_PP_KHZ	2B	
0-1[15:0]	MAX_SCL_PP_KHZ	r	Maximum push-pull clock frequency in KHz supported by hardware.
0x06	MAX_SCL_OD_KHZ	2B	
0-1[15:0]	MAX_SCL_OD_KHZ	r	Maximum open-drain clock frequency in KHz supported by hardware.
0x08	MAX_READ_TURNAROUND	2B	
0-1[15:0]	MAX_READ_TURNAROUND	r	Maximum read turnaround time in microseconds supported by hardware.
0x0A	MAX_DEV_NACK_RETRY	2B	
0-1[15:0]	MAX_DEV_NACK_RETRY	r	Maximum device NACK retry attempts supported by hardware.
0x0C	MAX_PAYLOAD_SIZE	2B	
0-1[15:0]	MAX_PAYLOAD_SIZE	r	Maximum payload transfer byte limit per command supported by hardware.
0x0E	IBI_TARGET_COUNT	1B	
[7:0]	IBI_TARGET_COUNT	r	Number of targets currently registered in the table.
0x0F	IBI_TARGET_CONFIG_ARRAY	6B	
*	IBI_TARGET_CONFIG_ARRAY		
0...	IBI_TARGET_CONFIG_ARRAY	rw	Array of 4-byte structures per active dynamic address (see layout below).

IBI_TARGET_CONFIG_ARRAY

Table 82: IBI_TARGET_CONFIG_ARRAY

Bits	Field	Acc	Description
0x00 DYNAMIC_ADDR 1B			
[7:0]	DYNAMIC_ADDR	rw	Target Dynamic Address: The 7-bit dynamic address assigned to the I3C target device (Bits [7] is Reserved, set to 0).
0x01 CONTROL_FLAGS 1B			
[7:0]	CONTROL_FLAGS	rw	Bit [0] - AUTO_PENDING_READ_ENABLE: - 0: Disabled (Default). The MCU forwards only the standard IBI payload to the Host. The MCU issues zero physical read transactions on the I3C bus (no zero-length reads). The Host is responsible for issuing follow-up private reads via I3C_CMD_EXECUTE_SEQUENCE. - 1: Enabled. The MCU automatically executes a private SDR read transaction on the physical I3C bus immediately following the IBI event. • Bits [7:1] - RESERVED: - Reserved. Read as 0, write as 0.
0x02 CONFIGURED_MAX_IBI_PAYLOAD_LEN 2B			
0-1[15:0]	CONFIGURED_MAX_IBI_PAYLOAD_LEN	rw	Maximum byte count configured to read during the immediate IBI phase. This value includes the 1-byte Mandatory Data Byte (MDB) Initial value is set by the producer If the written value exceeds the producer's capability, the producer must clamp the stored value to its hardware maximum.
0x04 CONFIGURED_MAX_PENDING_READ_LEN 2B			
0-1[15:0]	CONFIGURED_MAX_PENDING_READ_LEN	rw	Maximum byte count the MCU is configured to read during the automatic follow-up private SDR read (used when AUTO_PENDING_READ_ENABLE = 1). Initial value is set by the producer If the written value exceeds the producer's capability, the producer must clamp the stored value to its hardware maximum.

IPMI Channel Configuration Data

Table 83: IPMI Channel Configuration Data

Bits	Field	Acc	Description
0x00 IPMI_CONFIGURATION_DATA 0B			
N/A	N/A	n/a	No channel-specific Configuration Data. Use the common channel header only.

SPI Controller Channel Configuration Data

Table 84: SPI Controller Channel Configuration Data

Bits	Field	Acc	Description
0x00 SPI_SUPPORTED_CONFIG_FLAGS 4B			

Bits	Field	Acc	Description
0-3[0]	SPI_CPHA	r	Clock phase setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[1]	SPI_CPOL	r	Clock polarity setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[2]	SPI_CS_HIGH	r	Chip select active high setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[3]	SPI_LSB_FIRST	r	LSB-first transmit setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[4]	SPI_3WIRE	r	3-wire SI/SO shared-line setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[5]	SPI_LOOP	r	Loopback setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[6]	SPI_NO_CS	r	No chip-select setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[7]	SPI_READY	r	Ready/flow-control setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[8]	SPI_TX_DUAL	r	Dual-line TX setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[9]	SPI_TX_QUAD	r	Quad-line TX setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[10]	SPI_RX_DUAL	r	Dual-line RX setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[11]	SPI_RX_QUAD	r	Quad-line RX setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[12]	SPI_CS_WORD	r	CS-per-word setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[13]	SPI_TX_OCTAL	r	Octal-line TX setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[14]	SPI_RX_OCTAL	r	Octal-line RX setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[15]	SPI_3WIRE_HIZ	r	3-wire high-impedance turnaround setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[16]	SPI_RX_CPHA_FLIP	r	RX CPHA flip setting support. 0 - Setting not supported. 1 - Setting supported.

Bits	Field	Acc	Description
0-3[17]	SPI_MOSI_IDLE_LOW	r	MOSI idle-low setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[18]	SPI_READ_BITS_PER_WORD	r	SPI Read operation bits-per-word setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[19]	SPI_WRITE_BITS_PER_WORD	r	SPI Write operation bits-per-word setting support. 0 - Setting not supported. 1 - Setting supported.
0-3[31:20]	RESERVED	r	Reserved for future use. Read as 0, write as 0.
0x04 SPI_CONFIG_FLAGS 4B			
0-3[0]	SPI_CPHA	rw	Clock phase: 0 = sample on leading edge, 1 = sample on trailing edge.
0-3[1]	SPI_CPOL	rw	Clock polarity: 0 = idle low, 1 = idle high.
0-3[1:0]	SPI_MODE_X	rw	Combined mode field. MODE_0 = 0, MODE_1 = 1, MODE_2 = 2, MODE_3 = 3.
0-3[2]	SPI_CS_HIGH	rw	Chip select active high, inverted from default active-low.
0-3[3]	SPI_LSB_FIRST	rw	Transmit LSB first; default is MSB first.
0-3[4]	SPI_3WIRE	rw	SI/SO lines shared, half-duplex on single data wire.
0-3[5]	SPI_LOOP	rw	Loopback mode, TX internally connected to RX.
0-3[6]	SPI_NO_CS	rw	No chip-select line; CS management disabled.
0-3[7]	SPI_READY	rw	Slave drives low to pause transfer for flow control.
0-3[8]	SPI_TX_DUAL	rw	TX on 2 lines, dual SPI transmit.
0-3[9]	SPI_TX_QUAD	rw	TX on 4 lines, quad SPI transmit.
0-3[10]	SPI_RX_DUAL	rw	RX on 2 lines, dual SPI receive.
0-3[11]	SPI_RX_QUAD	rw	RX on 4 lines, quad SPI receive.
0-3[12]	SPI_CS_WORD	rw	Toggle CS between each word transfer.
0-3[13]	SPI_TX_OCTAL	rw	TX on 8 lines, octal SPI transmit.
0-3[14]	SPI_RX_OCTAL	rw	RX on 8 lines, octal SPI receive.
0-3[15]	SPI_3WIRE_HIZ	rw	High impedance turnaround in 3-wire mode.
0-3[16]	SPI_RX_CPHA_FLIP	rw	Flip CPHA on RX, separating RX sampling edge from TX.
0-3[17]	SPI_MOSI_IDLE_LOW	rw	MOSI idles low; default may be high-Z or high.
0-3[23:18]	SPI_READ_BITS_PER_WORD	rw	Requested word size for SPI Read operations. Value 0 means 8 bits per word. Values 1-63 encode that number of bits per word.
0-3[29:24]	SPI_WRITE_BITS_PER_WORD	rw	Requested word size for SPI Write operations. Value 0 means 8 bits per word. Values 1-63 encode that number of bits per word.
0-3[31:30]	RESERVED	rw	Reserved for future use. Must be written as 0.
0x08 SPI_NUM_DEVICES 1B			
[7:0]	SPI_NUM_DEVICES	rw	Number of SPI devices. Maps to number of supported Chip Selects. CS Max Index = (SPI_NUM_DEVICES - 1). Maximum number of devices supported by OBMF-ICP is 4.
0x09 RESERVED 3B			
0-2[23:0]	RESERVED	r	Reserved for alignment. Read as 0, write as 0.
0x0C SPI_RD_MAX_SPEED_HZ 4B			

Bits	Field	Acc	Description
0-3[31:0]	SPI_RD_MAX_SPEED_HZ	rw	Requested maximum SPI clock frequency in Hz for SPI Read operations.
0x10 SPI_WR_MAX_SPEED_HZ 4B			
0-3[31:0]	SPI_WR_MAX_SPEED_HZ	rw	Requested maximum SPI clock frequency in Hz for SPI Write operations.

IO Channel Configuration Data

Table 85: IO Channel Configuration Data

Bits	Field	Acc	Description
0x00 TRANSACTION_TYPES_SUPPORTED 2B			
0[0]	Sequential Port I/O Read 8-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[1]	Sequential Port I/O Write 8-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[2]	Sequential Port I/O Read 16-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[3]	Sequential Port I/O Write 16-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[4]	Sequential Port I/O Read 32-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[5]	Sequential Port I/O Write 32-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[6]	Fixed Port I/O Read 8-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[7]	Fixed Port I/O Write 8-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[8]	Fixed Port I/O Read 16-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[9]	Fixed Port I/O Write 16-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[10]	Fixed Port I/O Read 32-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[11]	Fixed Port I/O Write 32-bit	r	Indicates support for this transaction type: 0 - not supported. 1 - supported
0[15:12]	RESERVED	r	Reserved. Read as 0, write as 0.
0x02 IO_RANGE_COUNT 2B			
0-1[15:0]	IO_RANGE_COUNT	r	Total number of I/O_RANGE_CFG entries in IO_RANGE_CFG_ARRAY.
0x04 IO_RANGE_CFG_ARRAY 10B * IO_RANGE_COUNT			

Bits	Field	Acc	Description
0...	IO_RANGE_CFG_ARRAY	rw	Array of IO_RANGE_CFG structures.

I/O RANGE_CFG**Table 86: I/O RANGE_CFG**

Bits	Field	Acc	Description
0x00	IO_RANGE_FLAGS 2B		
0[0]	RANGE_ENABLE	rw	0 - range is disabled 1 - range is enabled
1:0[15:1]	RESERVED	r	reserved
0x02	IO_RANGE_START 2B		
1:0[15:0]	IO_RANGE_START	rw	The first port number in the range
0x04	IO_RANGE_END 2B		
1:0[15:0]	IO_RANGE_END	rw	The last port number in the range
0x06	IO_RANGE_MASK 2B		
6:5[15:0]	IO_RANGE_MASK	rw	0 in any bit position indicates that any value in the corresponding address bit in a received cycle will be treated as valid. 1 in any bit position indicates that any value in the corresponding address bit in a received cycle will be ignored.
0x08	IO_RANGE_SERVICE_TYPE 2B		
8:7[15:0]	IO_RANGE_SERVICE_TYPE	rw	0 - reserved 1 - KCS 2 - BT 3 - RTC 4 - POST Codes 5 - SUPER_IO 6 - COMA 7 - COMB 8 - I8042 controller 9 - Microcontroller1 10 - Microcontroller2 0x000b-0xffff - reserved

MEM_DEV_CFG**Table 87: MEM_DEV_CFG**

Bits	Field	Acc	Description
0x00	CAPABILITIES 4B		
0[0]	WRITEABLE	r	Device accepts write and erase operations. When 0, the Consumer shall not issue Write or Erase commands; responses will return WRITE_PROTECTED.
0[1]	NO_ERASE	r	Erase is not required before write (e.g. RAM-backed or internally managed storage). When set, the Consumer shall not issue Erase commands.
0[31:2]	RESERVED	r	Reserved. Read as 0.
0x04	DEVICE_SIZE 4B		
0-3[31:0]	DEVICE_SIZE	r	Total addressable size of the device in bytes, unsigned 32-bit little-endian. Must be a non-zero multiple of ERASE_SIZE (converted to bytes).

Bits	Field	Acc	Description
0x08 ERASE_SIZE 2B			
0-1[15:0]	ERASE_SIZE	r	Erase block size in KB, unsigned 16-bit little-endian. Must be a non-zero power of two. The Erase command Data Size field uses the same unit. Examples: 0x0001 = 1 KB, 0x0002 = 2 KB, 0x0004 = 4 KB, 0x0008 = 8 KB, 0x0010 = 16 KB, 0x0020 = 32 KB, 0x0040 = 64 KB, 0x0080 = 128 KB.
0x0A MAX_READ_SIZE 2B			
0-1[15:0]	MAX_READ_SIZE	r	Maximum Read request size in bytes, unsigned 16-bit little-endian. The Consumer shall not issue a single Read command larger than this value. Examples: 0x0040 = 64 B, 0x0080 = 128 B, 0x0100 = 256 B, 0x0200 = 512 B, 0x0400 = 1 KB, 0x0800 = 2 KB, 0x1000 = 4 KB.
0x0C MAX_WRITE_SIZE 2B			
0-1[15:0]	MAX_WRITE_SIZE	r	Maximum Write request size in bytes, unsigned 16-bit little-endian. The Consumer shall not issue a single Write command larger than this value. Examples: 0x0040 = 64 B, 0x0080 = 128 B, 0x0100 = 256 B.
0x0E RESERVED 2B			
0-1[15:0]	RESERVED	r	Reserved. Read as 0.

6. Security Considerations

6.1. Aggregation of Legacy Interfaces and Physical Protection

Legacy interfaces such as GPIO, I2C, SPI, and UART were originally designed for trusted, on-board communications and do not include native cryptographic security. As a result, they are susceptible to modern physical probing, man-in-the-middle (MITM) attacks, and supply chain threats. OBMF-ICP does not inherently mitigate the security weaknesses of the interfaces it consolidates. However, OBMF-ICP mitigates these vulnerabilities by consolidating multiple legacy interfaces over a single, unified transport. Although OBMF-ICP itself does not provide a security layer, this aggregation enables system designers to apply protection to a single transport point, making it feasible to secure previously exposed legacy traffic from physical attacks.

6.2. Leveraging Existing Security Layers

OBMF-ICP explicitly does not define or mandate native cryptographic protection or content security; instead, cryptographic protections such as encryption are outside the specification's scope. Implementations shall leverage established, industry-standard security protocols to secure traffic as needed. If a protocol tunneled over OBMF-ICP requires defense against targeted threats, it must be protected by its own security mechanism. If the entire OBMF-ICP payload needs to be encapsulated in a suitable transport-level protocol such as a Security Protocol and Data Model (SPDM) session implementation shall follow appropriate binding and support for SPDM session.

6.3. Channel-Specific Security Protections

Should a particular OBMF-ICP channel require dedicated security features, the channel shall follow the specific security requirements and features supported by interface tunneled over this specific channel. For example, if secure transport is necessary for a Trusted Platform Module (TPM) channel, OBMF-ICP implementations shall use the TCG TPM Communication over SPDM Secure Session specification to ensure the required security for that channel.

6.4. Device Authentication and Origin Validation (Informative)

OBMF-ICP can be implemented as a ROM-only or firmware-based solution. In either case, the integrator is responsible for choosing components where protocol-level security is enforced on the device itself. To validate device identity and origin, it is recommended to use SPDM Attestation. Additionally, for future-proofing against advanced cryptographic attacks, implementations should consider supporting post-quantum cryptography (PQC) in their SPDM attestation workflows.

6.5. Channel Isolation

Because OBMF-ICP aggregates multiple channels with distinct features and dedicated for specific usage, implementations shall provide robust and independent channel isolation. This ensures that unauthorized or malicious actions affecting one channel cannot compromise the security or performance of others.

Implementation Note: The methods for achieving channel isolation depend on hardware and system architecture. On the device side, it is recommended to use hardware offloading to enforce isolation. On the host, if software isolation is insufficient for high-security use cases, implementers may employ a dedicated service processor to intercept, segregate, and securely filter OBMF-ICP traffic before exposing it to the wider system.

6.6. Transport Guarantees

OBFM-ICP does not implement integrity verification mechanisms nor mechanisms to detect and/or recover an unresponsive or misbehaving communication endpoint. OBFM-ICP depends on the underlying transport layers to provide guaranteed service.

OBFM-ICP does not implement timeouts. In order to deal with scenarios such as no response from the communicating endpoint, timeouts may need to be implemented by the upper layers that use OBFM-ICP (that is typically a practice today, e.g. over eSPI).

If the underlying transport reports a fatal error (e.g. USB endpoint disconnected), OBFM-ICP implementation should report the error to the higher layers in an implementation specific way.