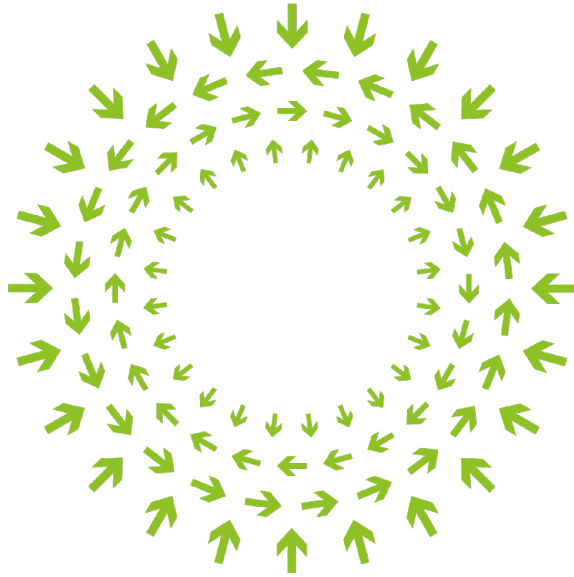




OPEN

Compute Project

BoW.Flexi Project: The Open Chiplet Economy BoW PHY
Workstream

BoW.Flexi

Version 1.0.0

Modular Base Specification

Effective 17th September 2026

Author: Morgan Whately, Infineon Technologies

Table of Contents

Version History	x
Terminology used in this document	xi
Conventions	xiv
1. License	1
1.1. Open Web Foundation (OWF) CLA	1
2. Acknowledgments	2
3. Compliance with OCP Tenets	3
3.1. Openness	3
3.2. Efficiency	3
3.3. Impact	3
3.4. Scale	3
3.5. Sustainability	3
4. BoW.Flexi Introduction	4
4.1. BoW.Flexi Layered Architecture	4
4.2. References	7
4.3. BoW.Flexi Overview	8
4.3.1. Motivation	8
5. BoW.Flexi Transaction Layer	9
5.1. BoW.Flexi Protocol	9
5.1.1. BoW.Flexi Commands	9
5.1.2. Configurable Address Width	13
5.1.3. Configurable Data Lane	14
5.1.4. Write Data Mask (Byte Strobe)	14
5.1.5. BoW.Flexi Initial Configuration	15
5.1.6. BoW.Flexi Configuration Register Usage	15
5.1.7. Commands Explained	16
5.1.8. Command Classes	18
5.1.9. Egress Arbitration	19
6. BoW.Flexi Link Layer	21
6.1. BoW.Flexi ECC Generation	21
6.1.1. Hsiao ECC Syndrome Table	22
6.2. Lane-Width Padding	23
7. BoW.Flexi Physical Layer	24
7.1. BoW.Flexi PHY Lane Width	24
7.2. Voltage Range Options	24
7.3. BoW.Flexi Power Modes	25
7.4. Wake from DEEPSLEEP	25
7.5. BoW.Flexi Signaling	26
7.5.1. Frame signal	26
7.5.2. Data Transmission	26
7.6. BoW.Flexi Reset	27
7.6.1. Hard Reset	27
7.6.2. Link Reset	27
7.7. BoW.Flexi Clocking	29
7.7.1. HCLK and TCLK Relationship	29
7.7.2. Clock Sharing	29
7.8. BoW.Flexi Physical Interface	30
7.8.1. Training	30

7.8.2. BoW.Flexi Die to Die Signals (Wires).....	30
7.9. BoW.Flexi Physical Configuration.....	31
7.9.1. Dead-Bug Views.....	31
7.9.2. Signal ordering.....	31
7.9.3. Bump/Pad Arrangements.....	32
7.9.4. Power and Ground	33
8. Compliance.....	34
8.1. BoW.Flexi Testability.....	35
8.1.1. Test Patterns.....	35
8.1.2. PRBS Generation	36
8.1.3. Test Approach	38
8.2. JTAG Transport over BoW.Flexi.....	42
8.2.1. Motivation and Scope.....	42
8.2.2. Signal Mapping	42
8.2.3. Register Control.....	43
8.2.4. Bring-up Sequence	44
8.2.5. Device Selection	45
8.2.6. Clocking and Datapath.....	46
9. BoW.Flexi Electrical Specifications.....	47
9.1. BoW.Flexi Slice Electrical Specifications.....	47
9.1.1. Parasitic Capacitance.....	47
9.2. BoW.Flexi Slice Receiver Bandwidth.....	47
9.3. BoW.Flexi Slice Timing Specifications	48
9.3.1. Multi-Lane Impact on Bit Error Rate	48
9.3.2. Multi-Lane Operation — Margin and Design Guidance.....	49
9.3.3. Transmitter Requirements	50
9.3.4. Receiver Requirements.....	50
9.4. BoW.Flexi Slice Channel Requirements	51
9.5. BoW.Flexi Compliance Testing.....	51
9.5.1. Margin-Based Compliance (Recommended)	51
9.5.2. Characterization (Recommended)	52
9.5.3. Bathtub Curve Extrapolation (Optional)	52
10. BoW.Flexi Registers	53
11. BoW.Flexi Command Examples.....	72
11.1. Non-Posted Write Command	72
11.1.1. Non-Posted Write.....	72
11.2. Write Response.....	74
11.3. Posted Write.....	74
11.4. Read Command.....	75
11.5. Read Response.....	75
11.6. Example Use of optional fields.....	76
11.7. BoW.Flexi Virtual-Wires.....	77
11.8. Configuration Command Examples.....	78
11.8.1. Error Report Example	79
11.8.2. Example sequence of interface configuration	81
11.8.3. Complete Configuration Example.....	81
11.9. Command Decode Example.....	84
12. How to map ARM AMBA Protocols to BoW.Flexi.....	85
12.1. Overview.....	85
12.2. Base Packet Fields (Common to All Protocols).....	85
12.3. Optional Field Usage Comparison	86
12.4. Optional Field Bit Ordering	86

12.5. Optional Field Configuration Registers.....	87
12.6. Packet Structure Comparison.....	89
12.6.1. Write Request Packets (Non-Posted, CMD=000).....	89
12.6.2. Read Request Packets (CMD=001).....	89
12.6.3. Read Response Packets (CMD=010).....	89
12.6.4. Write Response Packets (CMD=011).....	90
12.7. Feature Comparison.....	90
12.8. Packet Overhead Comparison (x8 Data Lane, 32-bit Bus, 32-bit Address).....	91
13. BoW.Flexi for Memories.....	92
13.1. Motivation.....	92
13.1.1. Basic Concept.....	92
13.2. DRAM Devices.....	93
13.2.1. Frame Signal Usage for Data Transactions.....	94
13.2.2. BoW.Flexi for DRAMs Concept.....	94
13.2.3. BoW.Flexi for DRAMs Byte Write Data Mask.....	96
13.3. Non Volatile Memory (NVM) Devices.....	97
13.3.1. Flash Devices.....	97
13.4. BoW.Flexi for Memory Registers.....	98
A. Example: AHB to Chiplet Bridge.....	100
A.1. Overview.....	100
A.1.1. Configuration Levels.....	100
A.1.2. Posted vs Non-Posted Write Selection.....	101
A.2. Physical Interface.....	101
A.3. AHB Interface Signals.....	101
A.3.1. AHB Subordinate Bridge Interface.....	101
A.3.2. AHB Manager Bridge Interface.....	102
A.3.3. Complete Bridge System Interface.....	104
A.3.4. Bridge Signal Mapping.....	105
A.4. Optional Field Usage.....	106
A.4.1. OPT_FIELD0: Transfer Attributes (Requests Only).....	106
A.4.2. OPT_FIELD1: Burst Control (Requests Only).....	107
A.4.3. OPT_FIELD2: Locked Transfers — HMASTLOCK (Requests Only).....	107
A.4.4. Write Byte Enables — Embedded in DATA.....	108
A.5. AHB Configuration Register.....	108
A.6. Packet Structure.....	109
A.6.1. Packet Format.....	109
A.6.2. Burst Handling.....	110
A.6.3. Locked Transfers.....	111
A.6.4. Packet Size Calculation.....	112
A.6.5. AHB Packet Timing Summary.....	112
A.7. Error Handling and Reporting.....	114
A.7.1. AHB Response Codes.....	114
A.7.2. Error Status Register (Address 0x10).....	114
A.7.3. Error Reporting Flow.....	115
A.7.4. Link-Level Errors.....	116
A.8. Transaction Examples.....	116
A.8.1. Posted Write Transaction (Minimum Mode).....	116
A.8.2. Non-Posted Write Transaction (Minimum Mode).....	117
A.8.3. Read Transaction (Minimum Mode).....	119
A.8.4. Non-Posted Write Transaction (Full AHB Mode).....	120
A.8.5. Burst Write Transaction (INCR4, Full AHB Mode).....	120
A.8.6. Configuration/Error Packet.....	121

A.9. Transaction Flows	123
A.9.1. Posted Write (Success).....	123
A.9.2. Non-Posted Write (Success).....	123
A.9.3. Non-Posted Write with AHB Error.....	123
A.9.4. Read Transaction (Success).....	124
A.9.5. Read Transaction with AHB Error	124
A.9.6. Error During Posted Write.....	125
A.10. Transaction Ordering.....	126
A.11. Configuration Sequence	126
A.12. Optional Field Configuration Register Settings.....	127
A.13. Reduced-Overhead Configurations.....	127
A.13.1. Default (Minimum Overhead)	127
A.13.2. Configured (Optional Fields Enabled)	127
B. Example: APB to Chiplet Bridge	129
B.1. Overview	129
B.2. Physical Interface	129
B.3. APB Interface Signals.....	129
B.3.1. APB Subordinate Bridge Interface	129
B.3.2. APB Manager Bridge Interface	130
B.3.3. Complete APB Bridge System.....	132
B.4. Optional Field Usage for APB	133
B.4.1. OPT_FIELD0: Protection + USER (Requests Only)	133
B.4.2. Error Reporting	133
B.4.3. Write Byte Strokes (PSTRB) — Embedded in DATA	134
B.4.4. Bridge Signal Mapping.....	134
B.5. APB Configuration Register	135
B.6. Packet Structure	135
B.6.1. Packet Format	135
B.6.2. Packet Size Calculation	136
B.6.3. APB Packet Timing Summary	137
B.7. Transaction Examples.....	138
B.7.1. Non-Posted Write Transaction	138
B.7.2. Read Transaction	139
B.7.3. Posted Write (Flow-Controlled).....	140
B.8. Error Handling.....	140
B.8.1. PSLVERR Response.....	140
B.8.2. Link-Level Errors	140
B.9. Configuration Sequence.....	140
B.10. Reduced-Overhead Configuration.....	141
B.10.1. Single Slave with Fixed Address Range	141
B.10.2. Minimal APB (No Optional Fields)	141
C. Example: Coexistence of Link, System and Advanced System Commands	142
C.1. Overview	142
C.2. Arbitration Between Sources	142
C.3. How the System Layer Knows the PHY Is Ready.....	145
C.4. Worked Sequence	146
D. Example: Layer Interaction at the Signal Level.....	147
D.1. Overview	147
D.2. Boundary Signals	149
D.2.1. AHB Bus to System Layer.....	149
D.2.2. System Layer to Transaction Layer.....	149
D.3. Worked Examples.....	150

List of Figures

4.1. BoW.Flexi Layered Architecture.....	5
5.1. Timing Relationships	13
7.1. BoW.Flexi Example 16b write to 16b address over 4 data lanes.....	26
7.2. BoW.Flexi Example 16b write to 16b address over 4 data lanes with ECC.....	27
7.3. BoW.Flexi Example 128b write to 16b address over 16 data lanes with ECC	27
7.4. BoW.Flexi Interface link reset sequence diagram	28
7.5. BoW.Flexi clocking	29
7.6. BoW.Flexi Pad/Bump Arrangement for Minimum Configuration	31
7.7. BoW.Flexi Pad/Bump Arrangement for x8 Configuration.....	32
7.8. BoW.Flexi Pad/Bump Arrangement with mismatch between Tx and Rx lane counts	32
7.9. BoW.Flexi Pad/Bump Arrangement for x8 Configuration with Power and Ground	33
8.1. Loopback Concept within a chiplet.....	35
8.2. Loopback Concept between chiplets	35
8.3. Stress Test Pattern: Isolated 1 and 0	36
8.4. Wafer sort test options: Single Slice testing (Left), Two Slice testing (Right)	39
8.5. Package testing from Chiplet_a to Chiplet_b	40
8.6. Package testing from Chiplet_a to Chiplet_b loopback to chiplet_a.....	41
8.7. BoW.Flexi JTAG transport bring-up sequence.....	44
11.1. BoW.Flexi Example 128b Read-Response over 16 data lanes with ECC.....	76
11.2. Configuration Read Request - Read Discovery Register (1-bit data lane).....	79
11.3. Configuration Read Response - Discovery Register (1-bit data lane).....	79
11.4. BoW.Flexi Command Decode Example.....	84
11.5. BoW.Flexi Example of how manager and subordinate connect.....	84
13.1. BoW.Flexi DRAM combined Bus x32	96
A.1. Complete AHB Chiplet Bridge System.....	104
A.2. Posted Write Timing.....	117
A.3. Non-Posted Write Timing	118
A.4. Read Transaction Timing.....	120
A.5. AHB Burst Write Sequence	121
A.6. Error Packet Timing	122
B.1. Complete APB Chiplet Bridge System.....	132
B.2. APB Write Sequence	138
B.3. APB Read Sequence	139
C.1. Egress arbitration across sources, with System-Layer arbitration and PHY-ready back-pressure	144
D.1. BoW.Flexi layer-to-layer signal connections (AHB system layer example)	148

List of Tables

1. Version History Table.....	x
4.1. Requirements for Flexibility on BoW interfaces	8
5.1. BoW.Flexi Commands Overview	9
5.2. BoW.Flexi packet Format	10
5.3. BoW.Flexi Payload order and description, default and max size.....	10
5.4. System Interface Command and Payload table, with the default, min, and max number of bits.....	11
5.5. Extension Command and Payload table, with the default, min, and max number of bits.....	12
5.6. Link Command and Payload table, with the default, min, and max number of bits.....	13
5.7. BoW.Flexi Data SIZE Configuration	14
5.8. BoW.Flexi Commands — description and behavior.....	16
5.9. BoW.Flexi link-interface arbitration order (example).....	20
5.10. BoW.Flexi egress arbitration sources	20
6.1. ECC Matrix [127:96]	22
6.2. ECC Matrix [95:64]	22
6.3. ECC Matrix [63:32]	22
6.4. ECC Matrix [31:0]	22
7.1. BoW.Flexi Voltage Ranges.....	25
7.2. BoW.Flexi Power modes.....	25
7.3. BoW.Flexi link reset timing parameters.....	28
7.4. BoW.Flexi Die-to-Die Signals (Wires).....	30
8.1. JTAG signal mapping to BoW.Flexi PHY pins.....	42
9.1. BoW.Flexi TX and RX Termination Resistance Requirements.....	47
9.2. BoW.Flexi Maximum Parasitic Capacitance	47
9.3. System BER vs. Per-Wire BER for Different Lane Counts.....	48
9.4. Margin Loss vs. Effective Per-Wire BER.....	49
9.5. BoW.Flexi Slice Receiver Voltage and Timing Requirements	51
9.6. Recommended Characterization Test Parameters	52
10.1. BoW.Flexi Register Address Map.....	53
10.2. Register Access Type Legend.....	55
10.3. BoW.Flexi Configuration and Status Register	56
10.4. BoW.Flexi Configuration Write Data Register.....	56
10.5. BoW.Flexi Configuration Read Data Register.....	56
10.6. BoW.Flexi Interface Configuration Register.....	57
10.7. BoW.Flexi Remote Error Register	58
10.8. BoW.Flexi Local Error Register.....	59
10.9. BoW.Flexi Discovery Register	59
10.10. BoW.Flexi Vendor Identity Register.....	61
10.11. BoW.Flexi Device Identity Register.....	61
10.12. BoW.Flexi Specification Version Register	61
10.13. BoW.Flexi Virtual Wire Out Register.....	62
10.14. BoW.Flexi Virtual Wire In Mask Register.....	62
10.15. BoW.Flexi Interface Configuration Extension Register 1	62
10.16. BoW.Flexi Interface Configuration Extension Register 2	63
10.17. BoW.Flexi Extension Data Lane Register.....	64
10.18. BoW.Flexi Optional Field Configuration Register	64
10.19. BoW.Flexi Optional Field Extended Configuration Register.....	65
10.20. BoW.Flexi Optional Field Trailer Configuration Register	66
10.21. BoW.Flexi Power State Register.....	66

10.22. BoW.Flexi Link Status Register	66
10.23. BoW.Flexi Advanced System Interface Register.....	67
10.24. BoW.Flexi Test Register.....	67
10.25. BoW.Flexi Test PRBS Seed Register.....	68
10.26. BoW.Flexi Test Error Mask Register.....	68
10.27. BoW.Flexi Test Clock Sampling Register	68
10.28. BoW.Flexi Test Error Log Register 0	68
10.29. BoW.Flexi Test Error Log Register 1	69
10.30. BoW.Flexi Extension Command Configuration Register (offset 0x6C).....	69
10.31. BoW.Flexi AMBA Configuration Register (offset 0x70).....	70
10.32. AMBA field usage	71
10.33. BoW.Flexi Message Registers (0x80–0x9C).....	71
11.1. Configuration and Status Register decode of 0x00005B1E.....	82
12.1. Base Packet Fields	85
12.2. Optional Field Usage Comparison	86
12.3. Optional Field Bit Ordering	86
12.4. Standardized Optional Field Register Settings	88
12.5. Write Request Packet Structure	89
12.6. Read Request Packet Structure	89
12.7. Read Response Packet Structure.....	89
12.8. Write Response Packet Structure.....	90
12.9. Feature Comparison.....	90
12.10. Packet Overhead Comparison	91
13.1. BoW.Flexi vs BoW.Memory Commands Descriptions	93
13.2. BoW.Flexi DRAM Commands	95
13.3. BoW.Flexi for Memory Extension Commands (e.g. 3'b000, 3'b101).....	95
13.4. BoW.Flexi NVM Commands.....	97
13.5. BoW.Flexi for Memory Register Address Map	98
13.6. BoW.Flexi for Memory Configuration Register.....	98
13.7. BoW.Flexi for Memory Address Register	99
13.8. BoW.Flexi for Memory Command Extension Register.....	99
A.1. AHB Bridge Feature Configuration	100
A.2. AHB Subordinate Bridge Signals.....	101
A.3. AHB Manager Bridge Signals	102
A.4. AHB to BoW.Flexi Packet Mapping.....	105
A.5. AHB Optional Field Allocation	106
A.6. AHB Packet Field Presence.....	109
A.7. AHB Transaction Field Descriptions.....	110
A.8. Supported Burst Types	110
A.9. AHB Default Packet Timing Summary (x8 data lane)	112
A.10. AHB Configured Packet Timing Summary (x8 data lane).....	113
A.11. BoW.Flexi AHB Error Register Extension (bits [31:12])	114
A.12. Receiver Decode Verification — Posted Write.....	116
A.13. Receiver Decode Verification — Non-Posted Write Request.....	118
A.14. Receiver Decode Verification — Write Response.....	118
A.15. Receiver Decode Verification — Read Request.....	119
A.16. Receiver Decode Verification — Read Response.....	119
A.17. Receiver Decode Verification — Configuration/Error Packet.....	122
A.18. Recommended Optional Field Register Configuration for AHB.....	127
B.1. APB Subordinate Bridge Signals	129
B.2. APB Manager Bridge Signals	130
B.3. APB to BoW.Flexi Packet Mapping	134

B.4. APB Packet Field Presence	135
B.5. APB Transaction Field Descriptions	136
B.6. APB Packet Timing Summary (32-bit data, 32-bit address, x8 data lane)	137
C.1. Egress arbitration stages	143
C.2. Signals between Layers	145
D.1. Transaction Layer logical fields and presence flags	149
D.2. Layer boundary normative references	150

Version History

Table 1. Version History Table

Date	Version	Author	Description
2026-09-08	1.0	Morgan Whately	Initial Release

Terminology used in this document

All registered trademarks mentioned in this document are the property of their respective owners.

This specification uses the following terminology:

- **AHB:** The Arm® AMBA® High-performance Bus protocol. The BoW.Flexi AHB bridge is a single configurable bridge that defaults to minimum overhead (no optional fields, single-beat transfers only) when unconfigured, and allows protection attributes, burst transfers, locked sequences, and more to be individually enabled/disabled via the AMBA Configuration Register.
- **APB:** The Arm® AMBA® Peripheral Bus protocol, designed for low-bandwidth peripheral access with minimal overhead. Optional PPROT support controlled by the AMBA Configuration Register.
- **BERT:** Bit Error Rate Testing - a method of testing the integrity of a data link by transmitting a known pattern (often PRBS) and counting the number of errors received
- **BIST:** Built-In Self Test - circuitry that allows a device to test itself, often used for manufacturing test and in-field diagnostics
- **BoW.Flexi:** A very low overhead configurable serial interface developed for chiplet communications where simplicity and low cost are optimized over absolute bandwidth
- **BoW.Flexi interface:** Two links configured to support communication between two chiplets
- **BoW.Flexi lane:** A single data wire on a BoW slice
- **BoW.Flexi link:** The connection of a Tx slice on a chiplet to a Rx slice on another chiplet
- **BoW.Flexi slice:** Consists of a source synchronous clock, a frame signal and 1 or more data lanes
- **Boundary Scan:** A JTAG-based (IEEE 1149.1) test technique that shifts test data through a chain of cells at the device I/O to test interconnect and pads without physical probes.
- **BoW for Memory, BoW.Memory:** The name for the high-bandwidth PHY defined previously by OCP, targeting Memory applications
- **Bunch of Wires, BoW:** The name for the high-bandwidth PHY defined previously by OCP, targeting Artificial Intelligence (AI) and High Performance Compute (HPC) applications
- **Configuration Command:** A BoW.Flexi command (CMD=111) that reads or writes a register in the partner chiplet's local register space; also used to carry error reports.
- **CSR:** Configuration and Status Register
- **Extension Command:** A command (CMD=110) whose class is configurable via the `ext_cmd_class` bit in the Interface Configuration Extension Register 1. By default (0) it is a System Interface command carried on the ordinary system source and carrying a 4-bit EXTCMD command-extension field (see Extension Command Configuration Register); when set to 1 — and the optional Advanced System Interface ports are implemented — it is an Advanced System Interface command routed to the dedicated advanced port pair (tx/rx_adv_sys_pkt_*). It carries implementation-specific transactions outside the standard command set.
- **Extension Command Field (EXTCMD):** A 4-bit field carried immediately after CMD (and the optional ECC) on the System Interface Extension Command (CMD=110), selecting one of 16 BoW.Flexi Standard extension commands (0x0-0xF, all reserved and not allocated in this revision). Configured via the Extension Command Configuration Register; may be omitted when `fixed_ext_cmd_en` is set. See [Section 5.1.1.3](#).

- **DEEPSLEEP:** A low-power state of the BoW.Flexi interface, in which the link is held in reset and only the wake-detection circuitry remains active. Configuration is retained across DEEPSLEEP. See [Section 7.4](#).
- **Die-to-Die, D2D:** Generic term used to refer to in-package interconnect between two dies (chiplets)
- **Discovery:** The ability to read a register on a chiplet to understand the capabilities of the interface of that chiplet
- **DLL:** Delay-Locked Loop — a feedback loop that aligns a derived clock or sampling phase to an input clock, used here to characterize sampling-position dither in the receiver timing budget.
- **FEXT:** Far-End Crosstalk — coupled noise injected from an aggressor lane into a victim lane that arrives at the far end of the channel. Bounded by the aggressor pattern; treated as a deterministic margin penalty.
- **Frame signal:** A control signal that indicates the start to end (duration) of a packet transmission. When high, indicates valid data transfer; when low, indicates end of packet and idle state
- **Hsiao codes:** An efficient form of Error Correcting Code (ECC) used for error detection and correction in data transmission
- **IEEE 1149.1:** The IEEE standard defining the JTAG Test Access Port (TAP) and boundary-scan architecture (the TCK, TMS, TDI and TDO signals).
- **IEEE 1838:** The IEEE standard for test access of multi-die (2.5D/3D) devices, defining how JTAG-based test is distributed across the dies in a package.
- **Initialization:** The process of preparing an interface for data transmission
- **IR drop:** Static or quasi-static voltage drop across the power-delivery network (current * resistance). Reduces the supply voltage seen by drivers/receivers and therefore consumes available signal margin.
- **JTAG:** Joint Test Action Group — the common name for the IEEE 1149.1 Test Access Port and boundary-scan interface used for structural test, board test and debug.
- **LFSR:** Linear-Feedback Shift Register — the structure used to generate PRBS test patterns (e.g. PRBS-31) for BIST and BER characterization.
- **Little Endian:** Byte ordering where the least significant byte is transmitted first
- **Mainband:** The BoW Protocol for use with the standard BoW PHY
- **MMIO:** Memory-Mapped I/O
- **NEXT:** Near-End Crosstalk — coupled noise injected from an aggressor lane into a victim lane that returns to the near (driver) end of the channel.
- **Non-Posted Write:** A write transaction that requires a write response (CMD=011) from the target device before the transmitter can proceed
- **NVM:** Non-Volatile Memory
- **OTP:** One-Time Programmable memory
- **OUI:** Organizationally Unique Identifier — a 24-bit IEEE-assigned vendor prefix used in the Vendor Identity Register to identify the chiplet vendor.
- **Packet:** The fundamental unit of data transfer in BoW.Flexi, consisting of a command field, optional ECC, and variable-length payload
- **Physical Layer, PHY:** The set of circuitries physically communicating bits from one die to another
- **Posted Write:** A write transaction whose Write Response acknowledges that the receiver has accepted the write and is ready to accept a further posted write (flow control), rather than confirming that the write has completed at its destination
- **PRBS:** Pseudo-Random Bit Sequence - a deterministic sequence of bits that appears random, used for testing data links

- **Read Command:** A command (CMD=001) that requests data from the partner chiplet's MMIO address space; answered by a Read Response.
- **Read Response:** The reply (CMD=010) returning the data requested by a Read Command. Its presence also implies a successful (OKAY) transfer.
- **Rx:** Receive
- **SDR:** Single Data Rate - data is transmitted on one edge of the clock
- **SECEDED:** Single Error Correction, Double Error Detection - a type of ECC that can correct single-bit errors and detect double-bit errors
- **Sideband:** This is the configuration protocol for the mainband interfaces
- **SSN:** Simultaneous Switching Noise — supply-rail bounce caused when many drivers switch at the same time through shared power/ground inductance. Scales approximately as $\sqrt{N}$ for random data and as N for pathological patterns; consumes voltage and timing margin as lane count increases.
- **TAP:** Test Access Port — the JTAG (IEEE 1149.1) state machine and signal interface on a device.
- **TCK, TMS, TDI, TDO:** The four JTAG signals — Test Clock, Test Mode Select, Test Data In and Test Data Out.
- **Trailing Optional Field (OPT_FIELD_T):** An optional field appended after the DATA field (rather than with the leading OPT_FIELD0-3). Sized and command-gated by the Optional Field Trailer Configuration Register; suited to data-phase information such as a per-transaction integrity code or status.
- **Tx:** Transmit
- **UI:** Unit Interval - the time duration of one bit on the data lane, used as a reference for timing measurements
- **Virtual Wire:** A command (CMD=101) that propagates the state of discrete sideband signals (for example interrupts or resets) across the link by updating the Virtual Wire registers.
- **Write Response:** The reply (CMD=011) acknowledging a write. For a Non-Posted Write it confirms the target has accepted the write; for a Posted Write it provides flow control (see **Posted Write**).

Conventions

This specification uses normative language to state requirements unambiguously. The following keywords are normative and are used consistently throughout this document:

- **shall** — a mandatory requirement. The behavior is required. **shall not** is a mandatory prohibition (the behavior is forbidden).
- **should** — a recommendation. The behavior is preferred but not required. **should not** is a strong recommendation against the behavior.
- **may** — an allowable option. The behavior is permitted, with no implied preference.

These keywords are invariant to capitalization: **shall**, **SHALL** and **Shall** carry the same normative meaning. Authors shall not use these keywords in their ordinary English sense; where a plain-English meaning is intended, an alternative word is used (for example, "can" or "is able to" for capability, "is expected to" or "will" for descriptive future behavior, and "might" or "could" for possibility).

This convention is consistent with IETF RFC 2119 and RFC 8174. Terms not listed above (for example **must**, **will**, **can**, **required to**, **need to**) are non-normative and do not, by themselves, express a requirement.

Chapter 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:

- Morgan Whately, Infineon Technologies
- Kash Johal, YorChip
- Matt McGuire, Microchip
- Anu Ramamurthy, Microchip
- Mark McClain, Everspin
- Brian Murray, VeriSilicon
- Kevin Donnelly, Eliyan

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.

Chapter 2. Acknowledgments

The Contributors of this Specification would like to acknowledge the following for their feedback:

Reviewers:

- Boon Chong Ang, Self

Chapter 3. Compliance with OCP Tenets

This specification adheres to the five core tenets of the Open Compute Project.

3.1. Openness

- All specifications are publicly available for download without restrictions or cost.
- Specification is unencumbered by technology license costs

3.2. Efficiency

- Leverages the Sideband PHY from the BoW 2.0 specification
- Optimized for flexibility and power efficiency for broad-based applications
- No initialization or link training required
- Energy proportional to activity with zero power at zero activity
- Support for deep sleep modes with zero wake-up training

3.3. Impact

This BoW.Flexi specification provides key advantages for chiplet-based systems:

- Can be implemented in legacy as well as state-of-the-art process nodes
- Can be implemented in any packaging technology including wire-bond
- Provides design flexibility while retaining interoperability.
- Supports power modes for applications such as IoT with low utilization

3.4. Scale

- Flexible to support data lanes from 1 to 128
- Supports data rates up to 4Gbps per I/O
- Flexible to support any packaging technology including wire-bond
- Portable across any pad pitch or bump pitch
- Portable across process nodes from legacy to advanced

3.5. Sustainability

- Provides very low peak power and energy proportional to activity to reduce power consumption of systems and minimize environmental harm

Chapter 4. BoW.Flexi Introduction

This specification describes the BoW.Flexi Protocol, the transmission format and the physical implementation of BoW.Flexi interface.

BoW.Flexi is a very low overhead configurable serial interface developed for chiplet communications where simplicity and low cost are prioritized over absolute bandwidth.

The BoW.Flexi interface consists of two uni-directional slices between chiplets. Each BoW.Flexi slice has a source synchronous clock, a frame signal, and 1 or more data lanes. The Tx slice on Chiplet_a connects to the Rx slice on Chiplet_b to create the first BoW.Flexi link. The Tx slice on Chiplet_b connects to the Rx slice on Chiplet_a to create the second BoW.Flexi link. The two links create a BoW.Flexi interface. The BoW.Flexi links are initialized in x1 data mode. Each new data bit is sent with the rising edge of the clock. The frame signal goes high at the start of a BoW.Flexi packet and goes low at the end. The data is sampled at the receiver using the falling edge of the clock.

BoW.Flexi transactions start with a 3-bit command followed by a payload. The payload varies by command. One command 3'b111 is utilized for the BoW.Flexi link configuration. One of the chiplets will be pre-assigned the role of the link configuration manager, while the other chiplet will have the role of subordinate. The manager can issue read and write requests to the subordinate chiplet BoW.Flexi register space (BoW.Flexi Link Configuration [Figure 4.1](#)).

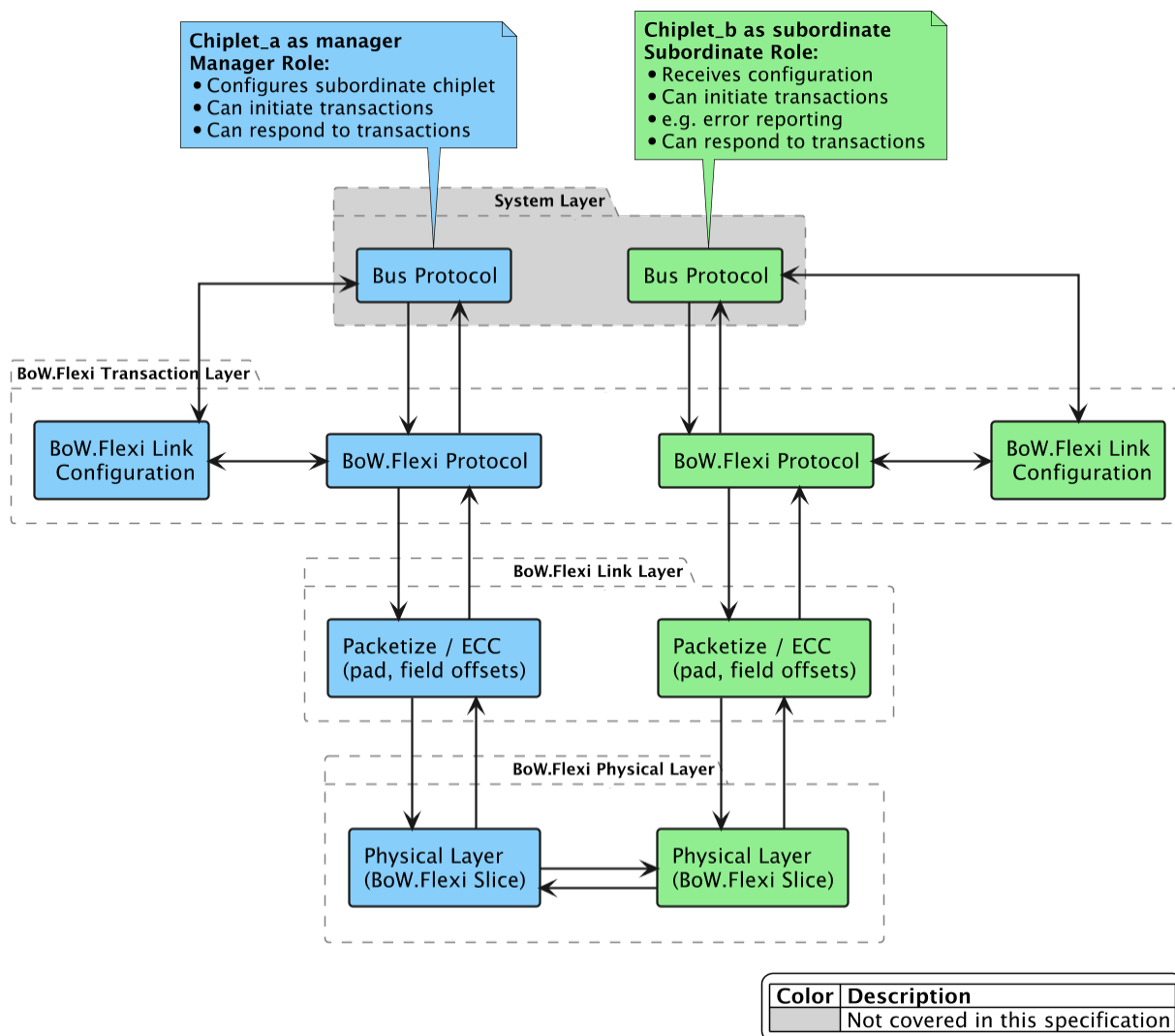
The BoW.Flexi link configuration is used to set the number of data lanes for the Tx and Rx, the address width, enable ECC, and other settings see [Interface Configuration Register](#).

Once the interface is configured the BoW.Flexi protocol can be used to transfer bus transactions between chiplets. Each side of the BoW.Flexi interface can initiate transactions. BoW.Flexi groups its commands into three classes — Link Interface, System Interface, and Advanced System Interface — and each class owns its own outstanding-transaction policy (see [Section 5.1.8](#)). The Transaction Layer enforces a single outstanding transaction for Link Interface commands. The System Layer owns the policy for both System Interface commands (single outstanding) and Advanced System Interface commands (its own policy, which may use a credit-based scheme). Ordinary System Interface traffic is presented to the Transaction Layer on a single combined system source. The Advanced System Interface is an optional, design-time class: when it is implemented and enabled, it presents its packets on a dedicated port pair directly to the Transaction Layer, which arbitrates it against the combined system/link traffic at a final, implementation-dependent stage.

BoW.Flexi does support up to 4 optional fields that can be used to communicate additional information about the transaction.

4.1. BoW.Flexi Layered Architecture

[Figure 4.1](#) shows the BoW.Flexi protocol stack as four layers. From the top (closest to the chiplet's internal bus) down to the physical die-to-die boundary, each layer has a distinct responsibility and a clean separation of concerns: upper layers know the **meaning** of a transaction, while lower layers know the on-wire **representation** (bit positions, ECC, padding). The link configuration protocol (Configuration Command) is part of the transaction layer.

Figure 4.1. BoW.Flexi Layered Architecture

The four layers are described below, from the bus-facing top down to the physical layer.

System Layer

Translates between the bus protocol (e.g. AHB, APB) and abstract BoW.Flexi transactions. It owns bus handshaking (ready/response), the posted/non-posted distinction, the register-vs-remote address decode, and the address-translation arithmetic. It also enforces the outstanding-transaction policy for System Interface commands — for example, an AHB System Layer holds `HREADY` low until the response returns, allowing only a single outstanding system transaction per direction; an Advanced System Interface layer may instead operate its own credit-based policy. Ordinary System Interface traffic is presented downward on a single combined system source. The optional Advanced System Interface, when implemented and enabled, is driven on a separate, dedicated port pair (`tx/rx_adv_sys_pkt_*`) rather than merged into the system source. This layer knows **nothing** about packet bit layout, padding, or ECC. See [Appendix A](#) and [Chapter 12](#).

Transaction Layer

Decides **what** transaction to send and **when**. It holds the egress arbiter (a round-robin link-interface arbiter, a top-level round-robin arbiter, and, when the optional Advanced System Interface is present, a final implementation-dependent stage), the Config/Error protocol engine, and Virtual Wire generation and capture. It presents each transaction as a set of logical fields together with a **valid/ready** contract (see [Table D.1](#) in [Appendix D](#)), and enforces the single-outstanding-transaction policy for Link Interface commands (Config/Error and Virtual Wire). Ordinary System Interface commands arrive on a single combined system source that the System Layer above has already arbitrated. When the optional Advanced System Interface is implemented and enabled, the Transaction Layer also has a dedicated advanced source (`tx/rx_adv_sys_pkt_*`) whose payload it forwards opaquely, arbitrating it against the combined system/link traffic at a final, implementation-dependent stage (see [Section 5.1.8](#)). It then assembles the granted transaction into a raw packet bit-stream, placing each field at its defined bit offset, and hands the Link Layer that bit-stream plus a valid-bit count — **not** individual fields. This layer knows the **meaning** and bit offset of each field, but nothing about the placement of padding or ECC.

Link Layer

Converts the Transaction Layer's raw packet bit-stream into a PHY-sized bit-stream. It inserts the ECC syndrome (see [Section 6.1](#)) and appends padding so the total length is a multiple of the active lane width. On the receive path it performs the inverse: ECC decode and correction and padding removal, returning the recovered bit-stream and its valid-bit count to the Transaction Layer.

Physical Layer

Moves the bit-stream across the die-to-die boundary. On transmit it serializes the bit-stream LSb-first onto `TD[N-1:0]` with TF framing and a forwarded TCLK. On receive it deserializes the incoming lanes and crosses from the RCLK domain into the system clock domain. See [Section 7.5](#).

4.2. References

- Bunch of Wires PHY Specification
 - <https://www.opencompute.org/chiplets/26/bunch-of-wires-bow-phy-specification-20>
- Transaction and Link Layer Specification for Bunch of Wires (BoW) Interfaces
 - <https://www.opencompute.org/chiplets/25/transaction-and-link-layer-specification-for-bunch-of-wires-bow-interfaces>
- BoW Memory Addendum to the BoW PHY Specification
 - <https://www.opencompute.org/documents/bow-memory-addendum-final-pdf>
- AMBA APB Protocol Specification
 - <https://developer.arm.com/documentation/ih0024/latest/>
- AMBA AHB Protocol Specification
 - <https://developer.arm.com/documentation/ih0033/latest/>
- AMBA AXI Protocol Specification
 - <https://developer.arm.com/documentation/ih0022/latest>

4.3. BoW.Flexi Overview

4.3.1. Motivation

BoW.Flexi targets chiplet links where simplicity and low cost matter more than absolute bandwidth — for example a low-data-rate sensor chiplet on a microcontroller, or chiplets in legacy nodes (e.g. 130nm).

The [Table 4.1](#) covers the main flexibility attributes for BoW.Flexi.

Table 4.1. Requirements for Flexibility on BoW interfaces

Flexibility Required	Justification	Link
Single Interface for Configuration and Data	A shared interface for both configuration and data transmission	Section 5.1.5
Configurable Number of Data Lanes	Support both Narrow and Wide interfaces, where the Tx and Rx data lane's can be asymmetrical The subordinate interface may be fixed in size, while the manager has flexibility to support many different subordinate devices	Section 7.1
Flexible packet Size based on CMD and frame Signal	Variable packet length interface → sized according to the information transmitted	Section 7.8
Configurable Address Width	Enable access to the full address space on the chiplet	Section 5.1.2
Configurable Data Size	SIZE field (3-bits) used to communicate the data size, for 1 to 128 Bytes	Section 5.1.3.1
Single Data Rate Operation	Interface can operate at system clock frequency; optional PLL	Section 7.8
Multiple Interface Voltage Options	Support for 0.7, 0.75, 0.8, and 0.9V enabling different frequencies at each operation point	Section 7.2
Optional ECC	Provide the ability to correct single bit transmission errors and detect double bit errors	Section 6.1
Bus Bridges	Connect simple chiplet internal buses between chiplets e.g. AHB bus	Appendix A
Virtual Wires Support	Communicate multiple signal level changes using a very short packet	Virtual-Wires
Wake Requests	Wake a chiplet from deep_sleep mode using the BoW.Flexi interface	Section 7.4
Support Memory Interfaces	Enable connections to memory devices	Chapter 13

Chapter 5. BoW.Flexi Transaction Layer

The Transaction Layer defines the BoW.Flexi packet protocol and the interfaces that generate and consume packets: the Config/Error and Virtual Wire link interfaces, the System interface, and the optional Advanced System interface. It assembles egress packets, arbitrates the shared link between the command classes, and disassembles ingress packets before handing the payload to the appropriate interface.

5.1. BoW.Flexi Protocol

The BoW.Flexi protocol defines a variable packet size format for initiating address-mapped transactions (e.g. register accesses, memory accesses). The BoW.Flexi configuration registers are used to set the relationship between the command and the payload format. All fields documented below are Little Endian. The packet size is variable, and the structure can be found in [Table 5.2](#). If ECC is enabled then an 8-bit syndrome is added between the command and the payload. The 8-bit ECC is encoded across the packet field via a SECDED encoder, see [Section 6.1](#).

Without ECC: [CMD:3][PAYLOAD][PAD]

With ECC: [CMD:3][ECC:8][PAYLOAD][PAD]

This approach follows the ECC generation scheme used by the [BoW Transaction and Link Layer](#)

5.1.1. BoW.Flexi Commands

The table below shows the BoW.Flexi commands. The function of these commands will be covered in the next sections. There are three types of commands:

- Link Interface Commands
- System Interface Commands
- Advanced System Interface Commands

Table 5.1. BoW.Flexi Commands Overview

CMD	Command	Description	Command Class
3'b000	Non-Posted Write	A Write transaction requiring a Write Response when completed	System Interface
3'b001	Read data from Address	Read Request	System Interface
3'b010	Read Response data	Response to Read Request	System Interface
3'b011	Write Response	Confirmation of Non-Posted Write completion or flow-control acknowledgment for a Posted Write	System Interface
3'b100	Posted Write	A Write transaction whose Write Response indicates the receiver is ready to accept another posted write (flow	System Interface

CMD	Command	Description	Command Class
		control)	
3'b101	Virtual Wire (e.g. 5, 8, 32b)	A command used to convey virtual wires between chiplets	Link Interface
3'b110	Extension Command	Command whose class is selected by ext_cmd_class in the Interface Configuration Extension Register 1: 0 (default) = System Interface command carrying a 4-bit EXTCMD extension field (configured via the Extension Command Configuration Register) on the combined system source; 1 = Advanced System Interface command on the dedicated advanced ports, permitted only when the peer reports adv_sys_if_present	System Interface (default) / Advanced System Interface
3'b111	Configuration / Error	A command for configuring the interface via register reads/writes. Errors are reported by writing to the Error Register	Link Interface

5.1.1.1. BoW.Flexi Transaction Structure

Table 5.2. BoW.Flexi packet Format

Bit	Name	Description
[2:0]	CMD	Command Description:see Table 5.1
[10:3]	ECC	SECEDED ECC is optional and default off
[P:11]	Payload	Payload size is a function of the command and the frame signal e.g. Read request command: the Payload will contain only the address and Size

Packet Format Structure

- **Bit-Order:** Least significant bit (LSb) first. Least significant bit on lowest numbered data signal.
- **Byte Order:** Little-endian, Least Significant Byte (LSB) first
- **Padding:** Zeros at end of packet to align to physical lane width of the interface

The BoW.Flexi Rx path uses the CMD field of the received packet to determine the appropriate behavior:

Table 5.3. BoW.Flexi Payload order and description, default and max size

Order	Name	Description
0	extcmd	4-bit command extension present only for the Extension Command (CMD=110) System Interface command. Selects one of 16 BoW.Flexi Standard extension commands (0x0-0xF, all reserved in this revision). Omitted when fixed_ext_cmd_en is set in the Extension Command Configuration Register. Part of the ordinary ECC-protected payload.
1	opt_field0	Enabled when opt_field0_cmd_support is non zero in the Optional Field Configuration Register. # of bits set using (opt_field0_size)
2	opt_field1	Enabled when opt_field1_cmd_support is non zero in the Optional Field

Order	Name	Description
		Configuration Register. # of bits set using (opt_field1_size)
3	opt_field2	Enabled when opt_field2_cmd_support is non zero in the Optional Field Extended Configuration Register. # of bits set using (opt_field2_size)
4	opt_field3	Enabled when opt_field3_cmd_support is non zero in the Optional Field Extended Configuration Register. # of bits set using (opt_field3_size)
5	size	Size of data field (request/response) 2^n bytes. Configuration register supports using a fixed size for all system transactions : fixed_size_enable & fixed_size
6	address	The requested address; Set by configuration register: address_width
7	data	Data to be written/read. Max size supported is set by configuration register: tx_data_size_max
8	opt_field_t	Trailing optional field (OPT_FIELD_T) appended after DATA. Enabled when opt_field_t_cmd_support is non zero in the Optional Field Trailer Configuration Register. # of bits set using (opt_field_t_size)

The system commands and payload structure can be found in [Table 5.4](#).

The link commands can be found in the [Table 5.6](#).

Table 5.4. System Interface Command and Payload table, with the default, min, and max number of bits

Description	CMD	OPT_FIELD0..3		SIZE		ADDR			DATA			OPT_FIELD_T	
		Def/Min	Max	Def	Fixed	Def	Min	Max	Def	Min	Max	Def/Min	Max
Non-Posted Write	3'b000	0	255	3	0	32	1	64	32	8	1024	0	255
Read Command	3'b001	0	255	3	0	32	1	64	0	0	0	0	255
Read Response	3'b010	0	255	0	0	0	0	0	32	8	1024	0	255
Write Response	3'b011	0	255	0	0	0	0	0	0	0	0	0	255
Posted Write	3'b100	0	255	3	0	32	1	64	32	8	1024	0	255

It is possible to have additional optional fields in the payload: the four leading fields OPT_FIELD0/1/2/3 (placed immediately after CMD) and a single trailing field OPT_FIELD_T (appended after DATA). The size of the optional fields and which commands utilize them is configurable via the [Optional Field Configuration Register](#), [Optional Field Extended Configuration Register](#) and [Optional Field Trailer Configuration Register](#).

5.1.1.2. Optional Field Packing

The contents of each optional field are defined by the active system protocol mapping (for example the AHB or APB bridge). The following rules govern how the sub-fields within an OPT_FIELD are packed so that both ends of the link reconstruct identical bit positions:

- **Field position:** the leading optional fields OPT_FIELD0-3 are placed immediately after CMD; the single trailing field OPT_FIELD_T is placed after DATA and before the end-of-packet padding. A trailing field suits information associated with the data phase (for example a per-transaction integrity code or status), because the transmitter can stream DATA without first computing the field, and the receiver locates OPT_FIELD_T using the DATA length already known from SIZE and the data-size configuration.
- **Sub-field order:** sub-fields are packed least-significant-bit (LSb) first, with no gaps, in the order defined by the protocol mapping. Bit 0 of the OPT_FIELD holds the LSb of the first sub-field.
- **Ordering principle:** sub-fields are ordered with the most frequently used (most likely to be enabled) sub-field first, in the low bits, and the more optional sub-fields last. A variable-width USER defined field, when present, always occupies the most-significant bits of the OPT_FIELD.
- **Compaction:** when a sub-field is disabled it is omitted and any higher sub-fields shift down to close the gap. Because the most-frequently-used sub-fields are anchored in the low bits, their positions remain stable regardless of which optional sub-fields are enabled. The value of `opt_fieldN_size` is the sum of the widths of the enabled sub-fields.



If all fields are enabled without ECC the total number of bits transmitted is 3 (CMD) + 255*4=1020 (OPT_FIELD0-3) + 3 (SIZE) + 64 (ADDR) + 1024 (DATA) + 255 (OPT_FIELD_T) = 2369 bits. With ECC enabled, $\text{roundup}(2369/120)=20$ the total number of bits is $2369 + (20*8) = 2529$ bits.

5.1.1.3. Extension Command Field (EXTCMD)

When the Extension Command (CMD=110) is used as a System Interface command (`ext_cmd_class = 0`), it carries a 4-bit **EXTCMD** field that extends the command set. EXTCMD (see [Table 5.5](#)) is placed immediately after CMD (and the optional ECC syndrome) and is part of the ordinary ECC-protected payload; the Extension Command carries no address field. This yields the payload order [CMD][ECC?][EXTCMD][OPT_FIELD0..3?][SIZE?][DATA][OPT_FIELD_T?]. The SIZE and DATA fields follow the System Interface command format on a per-transaction basis.

Table 5.5. Extension Command and Payload table, with the default, min, and max number of bits

Description	CMD	EXTCMD		OPT_FIELD0..3		SIZE		DATA			OPT_FIELD_T	
		Def	Fixed	Def/Min	Max	Def	Fixed	Def	Min	Max	Def/Min	Max
Extension Command	3'b110	4	0	0	255	3	0	32	8	1024	0	255

EXTCMD selects one of 16 **BoW.Flexi Standard** extension commands (0x0–0xF). All sixteen are reserved and **not allocated** in this revision; there is no vendor-specific range. Each command is introduced in a specific specification revision, and the register field that advertises it defines the revision it covers, so feature additions are managed entirely within this specification. Per-command configuration is held in the sixteen reserved Extension Command registers (0xA0–0xDF), one register per EXTCMD value.

The Extension Command Configuration Register (offset 0x6C) configures this feature:

- `ext_cmd_supported[15:0]` (read-only) advertises which extension commands are implemented — one bit per EXTCMD value. A received extension command whose support bit is 0 is not executed; the receiver reports it via the `local_cmd_error` bit of the Local Error Register and returns an Error Packet (Configuration/Error, CMD=111).
- `fixed_ext_cmd_en` / `fixed_ext_cmd` allow the EXTCMD field to be **omitted** from the packet (analogous to `fixed_size`) when an implementation issues only a single extension command; the receiver then assumes `fixed_ext_cmd` for every extension command.
- `ext_cmd_fixed_size_en` / `ext_cmd_fixed_size` provide **independent** fixed-size control for extension commands. The global `fixed_size` in the Interface Configuration Register does not apply to the Extension Command.

Both sides of the link shall be configured identically; ensuring this consistency is a system responsibility.

There are two link commands that have a different payload structure: the Configuration Command and the Virtual Wire as shown below.

Table 5.6. Link Command and Payload table, with the default, min, and max number of bits

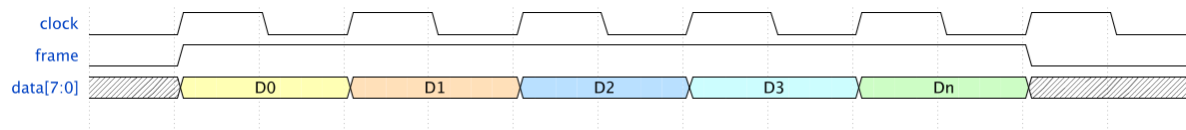
Description	CMD	LEN	WR	RD_RESP	ADDR			DATA		
		Fixed	Fixed	Fixed	Def.	Min	Max	Def.	Min	Max
Virtual Wire	3'b101	0	0	0	0	0	0	-	1	32
Configuration/Error	3'b111	2	1	1	8	8	19	32	8	32

The total packet size is the sum of the enabled and configured field sizes for each command.

5.1.1.4. Timing Relationships

The following diagram shows the timing relationship between clock, frame, and data signals:

Figure 5.1. Timing Relationships



Key Timing Points:

- **Clock Rising Edge:** Frame transitions, Data transitions
- **Clock Falling Edge:** Data captured by receiver
- **Frame High:** Data is valid. Packet in progress
- **Frame Low:** Idle state

5.1.2. Configurable Address Width

BoW.Flexi supports addressing up to 64 bits.

5.1.3. Configurable Data Lane

BoW.Flexi supports a configurable data transfer size.

5.1.3.1. 3 bit SIZE Field

BoW.Flexi supports a 3-bit **SIZE** field for system commands. This field encodes the data size as a power-of-2 number of bytes.

Table 5.7. BoW.Flexi Data SIZE Configuration

size[2]	size[1]	size[0]	# Bytes
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

5.1.4. Write Data Mask (Byte Strobe)

The BoW.Flexi protocol supports a write data mask that can be used to indicate which bytes of the data are valid for a write Command. The write data mask is sent as part of the DATA for write transactions and is used by the receiver to determine which bytes of the data should be written to the target address. When enabled the mask bit for each byte is prepended to its associated byte in the data field. For example, if the data lane width is 4 bytes and the write data mask is enabled, then the data field would be structured as follows:

```
[Mask Bit for Byte 0][Byte 0][Mask Bit for Byte 1][Byte 1][Mask Bit for Byte 2][Byte 2][Mask Bit for Byte 3][Byte 3]
```

The write data mask can be enabled or disabled via the configuration register [Interface Configuration Extension Register 1](#), field **write_data_mask_enabled**. When disabled, all bytes of the data are considered valid for write transactions. When enabled, the mask bits shall be included in the data field for write transactions and will be used by the receiver to determine which bytes to write.

When the write data mask is enabled, the SIZE field in the command packet indicates the size of the data excluding the mask bits. For example, if the SIZE field indicates a 4-byte write and the write data mask is enabled, then the data field would include 4 mask bits and 4 bytes of data.

5.1.5. BoW.Flexi Initial Configuration

The BoW.Flexi interface has a defined initial configuration (default column in [Interface Configuration Register](#)) where the interface operates in a x1 data lane mode for both Tx and Rx BoW.Flexi slices without ECC. BoW.Flexi supports a Configuration Command. Writes using the Configuration Command are configuration writes that return no response packet (i.e. no Write Response from the subordinate device to confirm the write was successful); they are distinct from a system Posted Write, which returns a flow-control Write Response. A read using the Configuration Command can be used to return the current configuration in the configuration registers. This command allows for the interface to configure the link between the chiplets. The command supports both reads and writes of the configuration registers, as well as read-responses. The registers exist in the configuration address space on both sides of the interface. These Configuration Commands are used to configure both sides of the link (e.g. add additional lanes, protocols used etc). The manager chiplet will configure both the Rx and Tx paths on the subordinate device.

For the configuration of the interface there shall to be a hierarchy between chiplets where the manager chiplet is responsible for the configuration. The [Discovery Register](#) has a field **disc_is_manager** that can be used to determine if the chiplet is capable of configuring the interface. The [Interface Configuration Register](#) has a field **is_manager** to determine which device is the manager. This bit is set by the manager device's boot process. If two devices are connected and both can be managers, one of the devices shall be pre-configured as the manager.

The manager device's firmware will have a priori knowledge of any subordinate expanded data signal locations or knowledge of how to discover the signal locations from the subordinate. The initial interface signal locations configuration are understood by the manager device. The manager can then configure the interface from its default values. See [Section 5.1.6](#), for the configuration parameters.

Setting the field **interface_configured** in the configuration register indicates that the configuration is complete and maps the behavior to the new configuration setting.

The configuration sequence is as follows:

1. The manager sends configuration writes to the subordinate device to set the desired parameters (e.g. number of lanes, address width, ECC).
2. The manager sends a write request to set the subordinate's **interface_configured** bit. The switchover on the subordinate is synchronous to the subordinate's local HCLK.
3. The manager then configures its own interface to match the subordinate's configuration (e.g. number of lanes, address width, etc.).
4. The manager sets its own **interface_configured** bit to complete the switchover.

The configuration shall be retained during DEEPSLEEP using retention flops.

Once configured the interface is now optimized for both System Interface and Link Interface commands.

5.1.6. BoW.Flexi Configuration Register Usage

The general flow a system should use to interact with their local chiplet configuration registers is as follows:

1. The Initiator (BoW.Flexi Firmware or Hardware State Machine) reads the local Config/Stat Register to ensure the 'Go' bit is 0x0 and the previous command is finished and checks that no error flags are set.

2. If the Initiator is executing a Configuration Write Command, the Initiator writes the Data to the local WriteData register.
3. The Initiator writes to the local Config/Stat Register, setting the Go, Write/Read, Addr, and **LEN** to the correct values for that command.
4. The Initiator monitors the local Config/Stat Register waiting for the Go bit to go to 0x0 (the value will stay 0x1 until a response command is received for Read).
 - a. The Initiator should make sure and check the Error bits when reading as well.
5. If the Initiator executed a Configuration Read Command, the returned value should be read from the local ReadData Register before executing the next command.

Implementation note

Because configuration writes return no response packet, software shall account for receiver processing latency when sending consecutive commands. Determine the maximum receiver latency during system design and store it in **config_posted_write_latency** (see [Interface Configuration Extension Register 2](#)). Where the maximum latency cannot be predetermined, insert wait states between commands (minimum 2 clock cycles recommended). For pointers to signal-level walk-throughs of the configuration sequence, see [Worked Examples](#).

5.1.7. Commands Explained

The table below summarizes each command, its CMD encoding, and the rules governing its packet structure and behavior. Detailed packet examples for each command appear in [Chapter 11](#).

Table 5.8. BoW.Flexi Commands — description and behavior

Command	CMD	Description & behavior
Posted Write	3'b100	Write request whose Write Response acknowledges that the receiver has accepted the write and is ready to accept a further posted write (flow control), rather than confirming that the write has completed at its destination. The receiver may return the response as soon as it has accepted the write, without waiting for downstream completion. As with a Non-Posted Write, only a single posted write may be in flight; the local bridge asserts HREADY on receipt of the response. See Section 11.3 .
Non-Posted Write	3'b000	Write request that requires a Write Response before the transmitter may issue further commands. Implementations may report out-of-bounds addresses as errors; interrupted non-posted writes return an Error Report. See Section 11.1.1 .
Read Command	3'b001	Requests data from the receiver's MMIO address space. Only one read may be in-flight; the requester cannot send further transactions until the Read Response arrives. Read Responses have higher priority than other commands. Out-of-bounds addresses result in an Error Report. See Section 11.4 .
Read Response	3'b010	Completes the outstanding read. Carries only CMD and DATA (no size or address fields); the frame signal going low terminates the transaction. A Read Response always corresponds to the last Read Command. See Section 11.5 .
Write Response	3'b011	Returned for a Non-Posted Write upon successful completion of the write, and for a Posted Write to acknowledge that the receiver is ready to accept a further posted write (flow control). CMD only (3 bits). On error, an Error Report (Configuration Command CMD=111) is returned instead — see Configuration/Error Command .

Command	CMD	Description & behavior
Virtual Wire	3'b101	Simple signal-level communication between chiplets. Once enabled via the configuration register, a full Virtual Wire packet is sent to establish all initial states. Lowest-numbered bit is sent first after the CMD; on a transition only bits up to and including the changed bit are transmitted (x1 mode); the last beat shall fill the full lane width; maximum 32 Virtual-Wires; bits beyond the last wire are zero-padded; ECC (when enabled) covers all transmitted bits. Wires can be masked via Virtual Wire In Mask Register ; removing the mask triggers a new packet if state has changed. See Section 11.7 .
Extension Command	3'b110	Extension command that lets a second protocol share the physical interface. Its command class is selected by <code>ext_cmd_class</code> in the Interface Configuration Extension Register 1. When <code>ext_cmd_class = 0</code> (default) it is an ordinary System Interface command carrying a 4-bit EXTCMD command-extension field immediately after CMD (and the optional ECC): <code>[CMD][EXTCMD][SIZE][DATA]</code> (or <code>[CMD][EXTCMD][OPT_FIELD0..3][SIZE][DATA]</code> when optional fields are enabled). EXTCMD selects one of 16 extension commands and is configured via the Extension Command Configuration Register; see Section 5.1.1.3 . When <code>ext_cmd_class = 1</code> — permitted only when the peer advertises <code>adv_sys_if_present</code> in the Advanced System Interface Register — it is an Advanced System Interface command that enters the Transaction Layer on the dedicated <code>tx/rx_adv_sys_pkt_*</code> port pair as an opaque bit-stream (with the Extension Command field already embedded) governed by its own System-Layer flow control (for example credits); in this class no EXTCMD field is parsed. In both cases the payload interpretation is system-defined. See Section 5.1.8 .
Configuration/Error	3'b111	Configures the interface and reports errors. Supports reads, writes, and read-responses to the configuration address space. Packet format: <code>[CMD:3][LEN:2][WR:1][RD_RESP:1][LOCAL_ADDR:8][DATA:variable][PAD:variable]</code> . Fields LEN , WR , RD_RESP , LOCAL_ADDR are always present. LEN encodes data size (00=8 bit, 01=16 bit, 10=32 bit, 11=Reserved); WR =1 writes, WR =0 reads; RD_RESP =1 marks a response to a read. The address range can be extended using Interface Configuration Extension Register 2 . See Section 11.8 .

5.1.7.1. Optional Discovery

The [Discovery Register](#), [Vendor Identity Register](#), and [Device Identity Register](#) can be used in the initial bring up of the system to identify the attached chiplet. Based on the data received back, the manager then writes the subordinate device configuration registers. After the configuration registers have been written, the manager will communicate to the subordinate device using these settings.

The manager device shall read the configuration registers to confirm the settings and check the interface is operating as expected. If the response is not correct, the manager device shall reset the subordinate device by sending a pre-defined (CLK,FRAME,Data0) sequence see [Section 7.6.2](#).

If the response is correct the interface can now communicate with the newly defined scheme (for details on physical signaling requirements, see [Section 7.5](#)).

5.1.7.2. Configuring initial state using NVM

With chiplets it is unlikely that the interface will need to change over the life of the product. If both sides of the interface have NVM/OTP, the interface can be configured locally as part of the chiplet first use process to eliminate the need for interface configuration following each power up or reset process. A bit in the

interface configuration register (interface_configured) can be used to indicate that the interface configuration has been completed.

5.1.7.3. Error Handling

An Error is reported via a write operation to the Configuration error register. This can be used to signal a BoW.Flexi Protocol error to the other chiplet. The Error register is partitioned into errors related to the interface (e.g. ECC errors, malformed packets) and errors related to higher level protocols (e.g. AHB decode error). The specific recovery action taken upon receiving an error is implementation dependent. However, the error classification bits [11:0] are defined by this specification and shall be set appropriately.

BoW.Flexi defines two error registers:

- **Remote Error Register** (Address 0x10): See [Remote Error Register](#). Stores error information received from the partner chiplet via a Configuration/Error Packet (CMD=111, WR=1). This register is optional for chiplets without a CPU.
- **Local Error Register** (Address 0x14): See [Local Error Register](#). Stores error information detected locally by this chiplet's controller.

Both registers share the same base layout: bits [11:0] are defined by the BoW.Flexi protocol for common error flags and error context. Bits [31:12] are reserved for implementation-dependent extensions (e.g. AHB specific error fields, see [Appendix A](#)).

The Error register write data length is dictated by the **LEN** field (2 bits, configuration commands only) and supports up to 32 bits.

5.1.8. Command Classes

BoW.Flexi commands are grouped into three classes. The class determines **which layer** owns the outstanding-transaction policy for that command. Link Interface commands are owned by the Transaction Layer, while both System Interface and Advanced System Interface commands are owned by the System Layer.

Link Interface

The BoW.Flexi-native link-management commands: the Configuration Command / Error (Configuration/Error, CMD=111) and Virtual Wire (CMD=101). These are generated and consumed entirely within the Transaction Layer (the Config/Error engine and the Virtual Wire logic). The **Transaction Layer** enforces the single-outstanding-transaction policy for this class — for example only one configuration read may be in flight at a time.

System Interface

The bus transactions carried on behalf of a System Layer bridge: Non-Posted Write (CMD=000), Read Command (CMD=001), Posted Write (CMD=100), and their Read Response (CMD=010) / Write Response (CMD=011) replies. Ordering and the single-outstanding-transaction policy for this class are owned by the **System Layer** (e.g. an AHB bridge holds **HREADY** low until the response returns).

Advanced System Interface

An optional, design-time class for a system protocol that supports more than one outstanding transaction — for example a credit-based or multi-outstanding bridge. It is carried on the Extension Command (CMD=110) encoding, which is **class-configurable** via

`ext_cmd_class` in the Interface Configuration Extension Register 1: by default (0) the Extension Command is an ordinary System Interface command; when set to 1 — only permitted when the peer advertises `adv_sys_if_present` in the Advanced System Interface Register — it is an Advanced System Interface command. This class carries **its own** flow-control policy (such as credits) in the **System Layer** and, unlike the other classes, enters the Transaction Layer on a **dedicated** port pair (`tx/rx_adv_sys_pkt_*`) that mirrors the Transaction-to-Link boundary: an opaque bit-stream (with the Extension Command field already embedded) plus a valid-bit count. The Transaction Layer performs **no** field manipulation on this source — only arbitration. Its maximum payload width `ADV_MAXBITS` is fixed at design time (`adv_sys_if_max_size` in the Advanced System Interface Register). The BoW.Flexi link itself still transmits packets sequentially.

How these classes share the link — the egress arbitration performed by the Transaction Layer — is described next, in [Section 5.1.9](#).

5.1.9. Egress Arbitration

The three command classes ([Section 5.1.8](#)) share a single link, so the Transaction Layer arbitrates egress access to it. The System Layer has already arbitrated the ready System Interface commands and forwards the winner on a **single** combined system source; the Transaction Layer therefore sees **one** ordinary system input. The Read Response/Write Response replies are system-class traffic and ride the combined System source, which the System Layer presents ahead of new requests, so the Transaction Layer never arbitrates system responses separately.

The Transaction Layer arbitrates the following egress sources:

- The three Link Interface sources — Error, Virtual Wire and Config.
- The combined System source from the System Layer.
- When the optional Advanced System Interface is implemented and enabled, the dedicated advanced source (`tx/rx_adv_sys_pkt_*`).

Normative requirement. Egress arbitration shall be **deadlock-free** and shall guarantee **forward progress**: every continuously-ready source shall be granted the link within a bounded number of packets, so that no source can be starved by traffic from another class. In particular, a steady stream of Link Interface traffic shall not indefinitely stall the System Interface, and vice versa. The specific arbitration policy (priority, rotation, weighting, and so on) is **implementation-dependent**. Beyond this bounded-service property there is **no** ordering guarantee between classes; in particular there is no ordering guarantee between Advanced System Interface traffic and the other classes.



Non-normative example. A staged round-robin arbiter satisfies the forward-progress requirement:

- A **link-interface arbiter** rotates round-robin between the three Link Interface sources in the order Error → Virtual Wire → Config ([Table 5.9](#)) and presents a single Link Interface source.

- A **top-level arbiter** alternates round-robin between that Link Interface source and the combined System source, producing a single combined system/link winner.
- When the advanced source is present, a **final stage** alternates round-robin between the combined system/link winner and the advanced source at each packet boundary.

Because every stage is round-robin, each ready source is served within a bounded number of packets and the egress path cannot deadlock.

Table 5.9. BoW.Flexi link-interface arbitration order (example)

Order	Source	Commands
1	Error	Error report (Configuration/Error, CMD=111 writing an error register)
2	Virtual Wire	Virtual Wire (CMD=101)
3	Config	Configuration Command read/write (Configuration/Error, CMD=111)

Table 5.10. BoW.Flexi egress arbitration sources

Source	Class	Notes
Link Interface	Link Interface	The three Link Interface sources: Error, Virtual Wire and Config.
System	System Interface	Combined system source from the System Layer: requests (Non-Posted Write, Read Command, Posted Write, CMD=000/001/100, and any Extension Command, CMD=110, configured as System Interface) and the Read Response (CMD=010) / Write Response (CMD=011) replies it returns. The System Layer presents pending responses ahead of new requests.
Advanced System (optional)	Advanced System Interface	Dedicated tx/rx_adv_sys_pkt_* source carrying Extension Command (CMD=110) packets when ext_cmd_class = 1. Present only when adv_sys_if_present = 1.

Because the classes are independent, the link can interleave them freely: a Link Interface configuration read can be immediately followed by a system transaction and then a burst of further system transactions (which the System Layer may source from either the System Interface or Advanced System Interface class). A worked, signal-level walk-through of this coexistence — including how the arbiter grants each source and how the System Layer learns the Physical Layer is ready to accept the next packet — is given in [Appendix C](#).

Chapter 6. BoW.Flexi Link Layer

The Link Layer converts the Transaction Layer's raw packet bit-stream into a PHY-sized bit-stream: it inserts the ECC syndrome and appends lane-width padding. On the receive path it performs the inverse — ECC decode/correction and padding removal — returning the recovered bit-stream and its valid-bit count to the Transaction Layer.

6.1. BoW.Flexi ECC Generation

The BoW.Flexi Protocol uses a link layer ECC generation matrix similar to the one used by [BoW Transaction and Link Layer](#).

The ECC bit positions are moved to bit positions $m[10:3]$ of the packet. Otherwise, the algorithm remains the same as described in the following text:

The codeword is based on Hsiao codes, using a check matrix, C , and a generator matrix, G , which is a submatrix of C . The transmitted codeword is the concatenation of a data word and additional check bits, calculated by multiplying the data word by G . To determine any errors in a received codeword, a syndrome is calculated by multiplying the codeword by C . If the syndrome equals zero, no errors have occurred, but if the syndrome does not equal zero, the type of error depends on the parity of the syndrome. A syndrome with odd parity implies a single-bit, correctable error, and the value of the syndrome determines the bit in error. On the other hand, a syndrome with even parity implies a multi-bit, uncorrectable error.

The check matrix, C , appears in the four tables below. The generator matrix, G , corresponds to bits (columns) $[127:11,2:0]$. After the bit position, the next 8 rows correspond to the ECC parity bits $[10:3]$ descending from the top. The final, bottom row in each table shows the decimal syndrome values that correspond to an error in a given bit.

ECC is computed as a 128-bit codeword carrying 120 payload bits plus an 8-bit syndrome. In the first codeword three of those bit positions hold the CMD field, leaving 117 payload bits; every subsequent codeword carries a full 120 payload bits. When the payload does not fill a complete codeword, the remaining payload positions are filled with zeros before the syndrome is generated or checked.

6.1.1. Hsiao ECC Syndrome Table

Table 6.1. ECC Matrix [127:96]

ECC	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
m[10]	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
m[9]	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	
m[8]	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
m[7]	1	1	1	1	0	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1
m[6]	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	1	1	1	0	1	1	0
m[5]	1	1	0	1	1	1	1	1	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1	0	1	1	0	0	1
m[4]	1	0	1	1	1	1	1	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	0	1	0	1
m[3]	0	1	1	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	1	1	1	0	0	1	0
Syndrome	254	253	251	247	239	223	191	127	248	244	242	241	236	234	233	230	229	227	220	218	217	214	213	211	206	205	203	199	188	186	185	182

Table 6.2. ECC Matrix [95:64]

ECC	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
m[10]	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m[9]	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
m[8]	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0
m[7]	1	1	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	1	1	1	1	0	1
m[6]	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	1	1
m[5]	1	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1	0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1
m[4]	0	1	1	0	1	1	1	0	1	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1
m[3]	1	1	0	1	1	1	0	1	1	1	1	0	0	1	0	1	1	0	1	1	0	1	1	1	1	0	1	1	1	1	1	1
Syndrome	181	179	174	173	171	167	158	157	155	151	143	124	122	121	118	117	115	110	109	107	103	94	93	91	87	79	62	61	59	55	47	31

Table 6.3. ECC Matrix [63:32]

ECC	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
m[10]	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
m[9]	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
m[8]	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
m[7]	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0
m[6]	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1
m[5]	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	0	0	1	0
m[4]	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	0	1
m[3]	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0
Syndrome	224	208	200	196	194	193	176	168	164	162	161	152	148	146	145	140	138	137	134	133	131	112	104	100	98	97	88	84	82	81	76	74

Table 6.4. ECC Matrix [31:0]

ECC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
m[10]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
m[9]	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
m[8]	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
m[7]	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0
m[6]	1	0	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	1	0	0	0	1	1	0
m[5]	0	1	1	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	0	0	0	0	0	1	0	0	1	0	1
m[4]	0	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	1	1
m[3]	1	0	1	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1
Syndrome	73	70	69	67	56	52	50	49	44	42	41	38	37	35	28	26	25	22	21	19	14	128	64	32	16	8	4	2	1	13	11	7

ECC Syndrome Location [10:3], CMD Location [2:0]

6.2. Lane-Width Padding

The transmitted bit-stream shall be a whole number of PHY beats. After ECC generation, the Link Layer appends **padding** bits at the end of the packet so that the total transmitted length is an integer multiple of the active parallel datapath width, which the Physical Layer serializer maps onto the PHY lanes (see [Section 7.1](#)).

- **Value:** padding bits are always zero.
- **Placement:** padding is the last field of the transmitted packet, after any DATA and after the ECC syndrome. It is not shown in the packet structure tables ([Table 5.2](#)) and carries no information.
- **Amount:** the number of padding bits is the smallest value that rounds the post-ECC packet length up to a multiple of the datapath width. A packet already aligned to the datapath width has zero padding.

Lane-width padding is distinct from the zero-fill used during ECC generation: ECC is always computed over 128-bit units and a partial unit is zero-filled **before** the syndrome is calculated (see [Section 6.1](#)), whereas lane-width padding is added **after** the syndrome and only to align the packet to the datapath.

On receive, the Link Layer removes the padding before ECC checking. The receiver does not rely on the padding to delimit the packet; the Physical Layer frame signal (RF) going low terminates the packet, and the recovered valid-bit count is passed up to the Transaction Layer.

Chapter 7. BoW.Flexi Physical Layer

The Physical Layer serializes the Link Layer bit-stream onto the die-to-die (D2D) lanes and recovers it on the far side. It defines the PHY lane width and parallel datapath, the Frame/Clock/Data signaling, the voltage and power modes, the wake and link-reset behavior, and the clocking relationships. The electrical characteristics of the underlying slice are given in the Electrical Specifications chapter.

7.1. BoW.Flexi PHY Lane Width

It is envisioned that there will be a wide range of chiplets. These chiplets will be optimized for their own functionality and performance. BoW.Flexi supports a variable lane width of the interface. The number of PHY data lanes can be any integer from 1 to 128 in 1-bit increments. With this the subordinate chiplet can size its interface based on its own bandwidth requirements and frequency of operation. The interface lane width is not required to be symmetrical from a link perspective. The Chiplet_a to Chiplet_b link can have a different width than the Chiplet_b to Chiplet_a link.

The parallel data interface between the Physical Layer and the Link Layer is configured independently of the PHY lane count. Its width can be any power of two from 1 to 128 bits (1, 2, 4, 8, 16, 32, 64 or 128). The Physical Layer serializer maps each parallel word onto the active PHY lanes, Lsb first.

7.2. Voltage Range Options

The default voltage for the BoW.Flexi interface is shown in the table below [Table 7.1](#). All BoW.Flexi chiplets shall be functional across the default range.

In systems with heterogeneous integration, the fundamental semiconductor technology differences may favor operating over a tighter voltage range. BoW.Flexi supports 3 optional voltage ranges as shown in the [Table 7.1](#).

If permitted from a technology and reliability perspective the chiplet vendor shall provide their interface performance (max frequency, power, and leakage) within these ranges.

This way the interface can be sized from both a lane width as well as the voltage range of operation to meet the system needs.

For the lowest physical layer leakage power, both the Tx and Rx should share the same power supply. This is a guideline but not required by the specification.

Table 7.1. BoW.Flexi Voltage Ranges

Range	Nominal	Min	Max	Notes
Default	0.75V	0.65V	0.90V	All BoW.Flexi shall be functional across this range
Low Range	0.7V	0.65V	0.75V	Optional; Low power mode
Mid Range	0.8V	0.75V	0.85V	Optional; Utilize a nominal power supply
High Range	0.9V	0.85V	0.95V	Optional; Useful for legacy nodes e.g. 130nm

7.3. BoW.Flexi Power Modes

Table 7.2. BoW.Flexi Power modes

Component	ACTIVE/SLEEP	DEEPSLEEP	RESET
BoW.Flexi PHY	on	on	off
Link Interface (Configuration)	on	retained	off
Virtual Wires	on	retained	off
System Interface	on	off	off
Wake on frame toggle	on	on	off
Link Reset Detector	on	on	off
Clock Sharing (if configured)	on	off	off

7.4. Wake from DEEPSLEEP

BoW.Flexi supports the ability to send a wake request to a chiplet by raising the frame signal (while clock is low, and Data0 is low). If Chiplet_a wants to wake Chiplet_b, it would raise its frame signal. Chiplet_b will receive the wake request, and transition to an appropriate power mode. Chiplet_b can send a valid packet to Chiplet_a that it is ready for communication. The first clock transition will be used to de-assert the Chiplet_a frame signal. Once Chiplet_a has received this packet, it can start communicating.

7.5. BoW.Flexi Signaling

The BoW.Flexi interface wiring and physical characteristics follow the BoW Sideband approach see [Section 9.1](#). Each BoW.Flexi interface is a pair of links, each with Frame, Clock, and lane signals. This section describes the use and sequence of those signals and is illustrated in [Figure 7.1](#), [Figure 7.2](#), [Figure 7.3](#).

7.5.1. Frame signal

The frame signal is a control signal — it is **not** part of the packet payload and is not included in the packet structure tables. The frame signal indicates the start and end of a packet transmission:

- **Frame high:** Data is valid. A packet is in progress. Data on the receiver's Rx lanes is sampled on the falling edge of the clock.
- **Frame low:** The interface is idle. No valid data is present on the data lanes.

The frame signal enables variable-length packets and efficient use of the data lanes. There is no requirement for the receiver to know the packet length in advance — the Frame deassertion terminates the packet.

7.5.2. Data Transmission

Before transmitting, the BoW.Flexi interface controller assembles a transmission packet as described in [Table 5.2](#). Once triggered, the transmitter will use the rising edge of the TCLK to assert the associated Frame (TF) signal and transmit the packet on Tx lanes (TD[N-1:0]). Data on the receiver's Rx lanes (RD[N-1:0]) is sampled on the falling edge of RCLK and is only considered valid as long as the Frame (RF) signal is high. The transmitter shall wait at least one TCLK period after de-asserting TF before starting transmission of another packet.

Packet transmission order is Little Endian.

Here is an example of a simple signal relationship diagram:

Figure 7.1. BoW.Flexi Example 16b write to 16b address over 4 data lanes

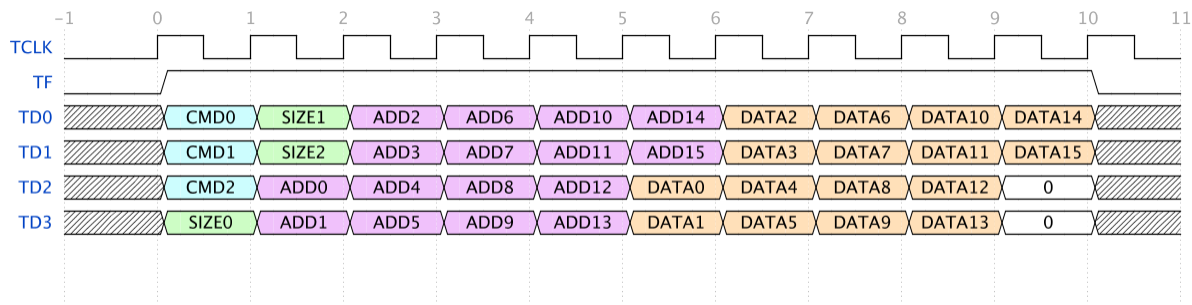
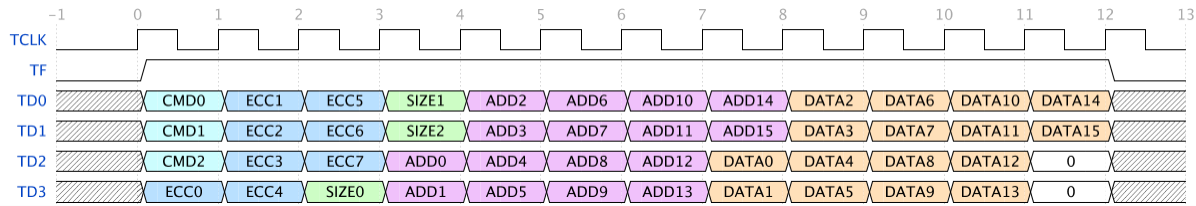
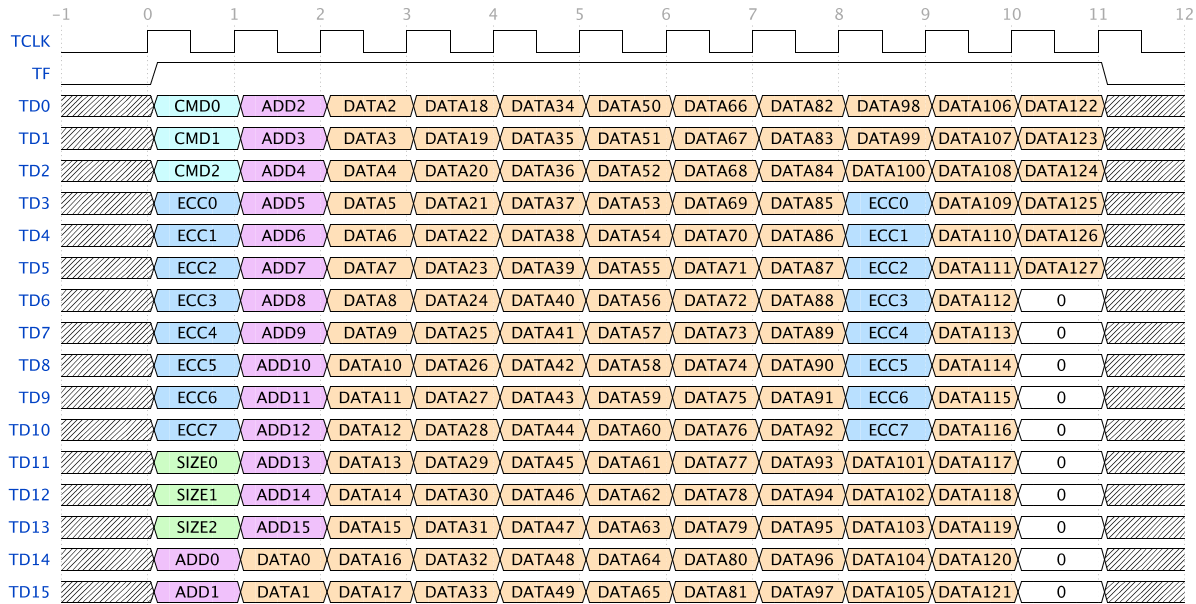


Figure 7.2. BoW.Flexi Example 16b write to 16b address over 4 data lanes with ECC**Figure 7.3. BoW.Flexi Example 128b write to 16b address over 16 data lanes with ECC**

7.6. BoW.Flexi Reset

The interface supports two reset states. A system-level **Hard Reset** resets the entire chiplet and is described below. A **Link Reset** returns only the BoW.Flexi interface to its default configuration and is part of the Physical Layer (see [Section 7.6.2](#)).

7.6.1. Hard Reset

Each chiplet is expected to have a method to reset the entire chiplet. A hard reset reverts the chiplet to its power-on state. This is outside the scope of BoW.Flexi.

7.6.2. Link Reset

A configurable interface like BoW.Flexi requires a method to ensure that it can revert back to default in case some link configuration is not operating correctly without requiring a hard reset. See [Figure 7.4](#).

While the clock is static low, the frame signal falling edge is used to sample the TD0 signal. If the sampled data over four consecutive cycles is 0101, a link reset is initiated.



When the optional JTAG-over-BoW.Flexi transport is enabled (see [Section 8.2](#)), the link-reset detector remains enabled on the JTAG subordinate but is disabled on the JTAG manager, and only the manager may issue a link-reset request. A manager-issued link reset is the mechanism by which both ends leave JTAG mode, reverting `jtag_over_flexi_en` (and the rest of the Extension Registers) to their default values.

A link reset returns the interface to its default (power-up) configuration:

- The Interface Configuration Register reverts to its reset values (x1 mode, no ECC, etc.)
- The Extension Registers (1 and 2) revert to their reset values
- The AMBA Configuration Register reverts to its reset values
- The Optional Field Configuration Registers revert to their reset values
- The Discovery, Vendor Identity, and Device Identity registers are **unaffected** (static read-only registers)
- The Config/Status Register is cleared
- Error registers are cleared

Figure 7.4. BoW.Flexi Interface link reset sequence diagram

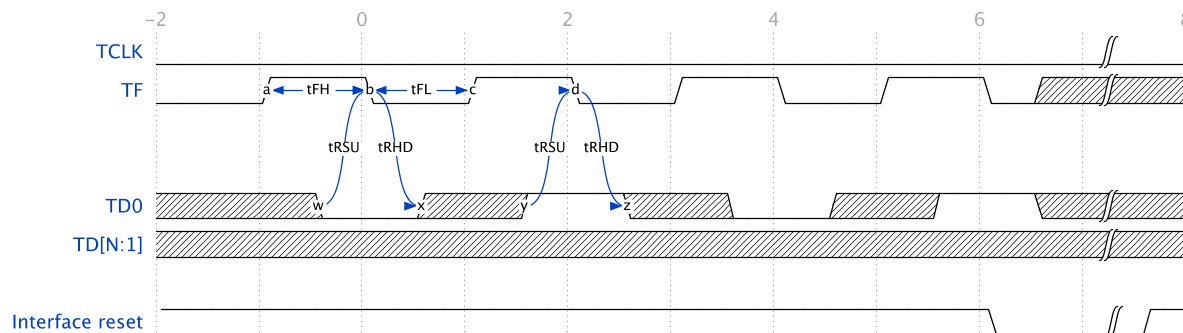
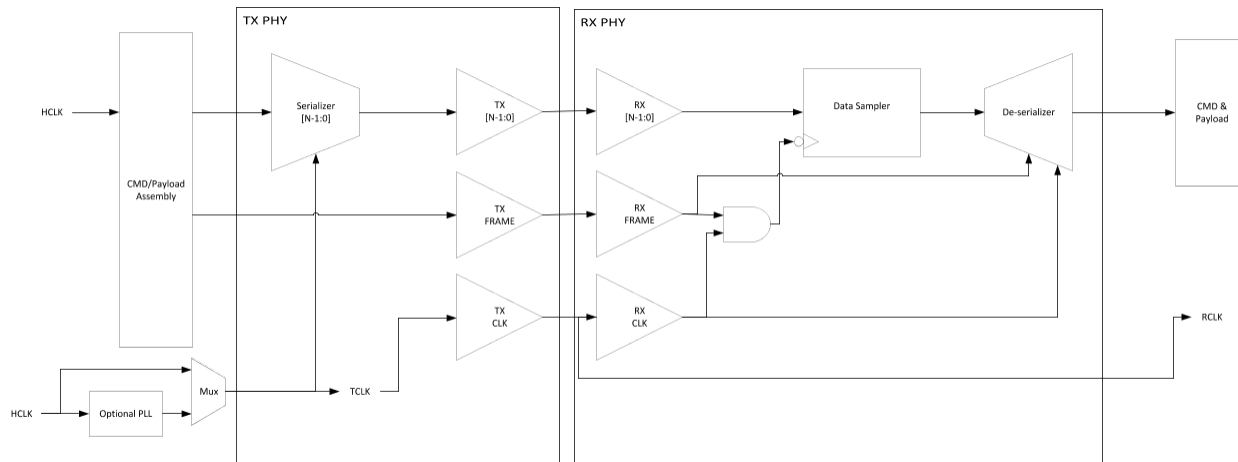


Table 7.3. BoW.Flexi link reset timing parameters

Parameter	Symbol	Min	Max	Units
Reset Frame Low Time	tFL	100	-	ns
Reset Frame High Time	tFH	100	-	ns
Reset Setup Time	tRSU	5	-	ns
Reset Hold Time	tRHD	5	-	ns

7.7. BoW.Flexi Clocking

Figure 7.5. BoW.Flexi clocking



7.7.1. HCLK and TCLK Relationship

HCLK is the system bus clock (AHB domain). TCLK is the BoW.Flexi interface clock used to serialize and transmit packets. The TCLK frequency can be equal to or higher than HCLK.

The BoW.Flexi bridge can operate in two modes:

- **Same-clock mode** (TCLK = HCLK): No clock domain crossing logic is required. This is the simplest and lowest-area configuration.
- **Frequency-scaled mode** (TCLK > HCLK): A PLL can be used to generate TCLK from HCLK. In this mode the bridge includes synchronization and FIFO logic to safely transfer data between the two clock domains. This mode increases link bandwidth without requiring the system bus to operate at a higher frequency.



TCLK and HCLK are always frequency-related (integer or rational ratio). Fully asynchronous operation (unrelated clock sources) is not supported.

7.7.2. Clock Sharing

The BoW.Flexi interface does not require that the transmit clock toggles if no data is transmitted (frame signal is low). BoW.Flexi does support a mode to enable providing a free running clock across the interface using the transmit clock. This eliminates the need to add a separate wire for the clock signal. This can be configured in the [Interface Configuration Extension Register 1](#) field: **clock_sharing_enabled**. The manager chiplet can utilize its TCLK signal to provide a clock to the subordinate chiplet. The subordinate chiplet can use this clock to send data back to the manager chiplet. The frame signal is high for packet transfers. It is also possible for a subordinate chiplet to share its clock to the manager chiplet. The clock sharing is only available in ACTIVE/SLEEP mode see [Table 7.2](#). Clock sharing provides a free running clock across the interface.



Additional details on clock sharing implementation (e.g., frequency negotiation, jitter requirements) are planned for a future revision.

7.8. BoW.Flexi Physical Interface

BoW.Flexi is a single data rate (SDR) low voltage interface optimized for sending variable length packets across a xN lane interface based on the BoW.Sideband physical layer approach.

The signals are as follows:

- Clock
- Frame
- xN data lanes

The interface utilizes a frame signal to indicate the packet start, length and end.

This specification is frequency independent; it is up to the system to ensure that the chiplets are compatible.

- BoW.Flexi shall be capable of operating correctly (i.e., transmitting/receiving configuration data) after exiting hard or link reset.
- BoW.Flexi always includes both transmit and receive functionality.
- BoW.Flexi operates in Single-Data Rate (SDR). Data transitions at the Tx are aligned with the rising edge of the forwarded clock, and data is captured at the receiver with the falling edge of the clock that follows the rising (launching) edge.
- BoW.Flexi slices are always unterminated or source-terminated.

7.8.1. Training

No training is defined for BoW.Flexi. The interface operates in single data rate mode where the received data is sampled using the falling edge of the received clock.

7.8.2. BoW.Flexi Die to Die Signals (Wires)

Table 7.4 summarizes these signals.

Table 7.4. BoW.Flexi Die-to-Die Signals (Wires)

Function	# Wires	Signal Name	Notes
Tx Clock	1	TCLK	
Tx Data	N	TD[N-1:0]	N is set using the one-hot fixed data lanes in Table 10.6 or by enabling the <code>adv_tx_data_lane</code> and setting <code>ext_reg_tx_num_data_lanes + 1</code> in Table 10.17
Tx Frame	1	TF	
Rx Clock	1	RCLK	

Function	# Wires	Signal Name	Notes
Rx Data	M	RD[M-1:0]	M is set using the one-hot fixed data lanes in Table 10.6 or by enabling the <code>adv_rx_data_lane</code> and setting <code>ext_reg_rx_num_data_lanes + 1</code> in Table 10.17
Rx Frame	1	RF	

7.9. BoW.Flexi Physical Configuration

7.9.1. Dead-Bug Views

The physical diagrams and descriptions in this document are to be interpreted as looking down at the top layer of the unpackaged chiplets. BoW.Flexi supports both wire-bond and chiplet packaging. For flip-chip packages, these views are equivalent to looking through the bottom of the package with the balls up (dead bug view). For the view as seen looking down on a flip-chip package as mounted on a PCB (live bug view), these views shall be mirrored.

7.9.2. Signal ordering

A BoW.Flexi interface shall conform to these wire order rules at the edge of the chip:

- Minimum Configuration: The signals are in the following order at the chip edge, going clockwise around the chiplet in a dead-bug view: TD[0], TF, TCLK, RCLK, RF, RD[0]
- Additional Tx and Rx data lanes: The Tx data lanes count down to the minimum configuration, while the Rx data lanes count up.

Figure 7.6. BoW.Flexi Pad/Bump Arrangement for Minimum Configuration

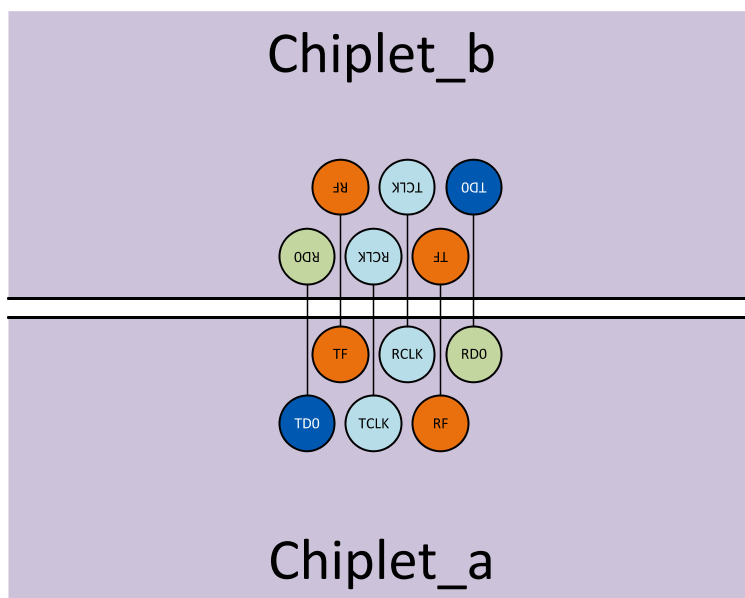
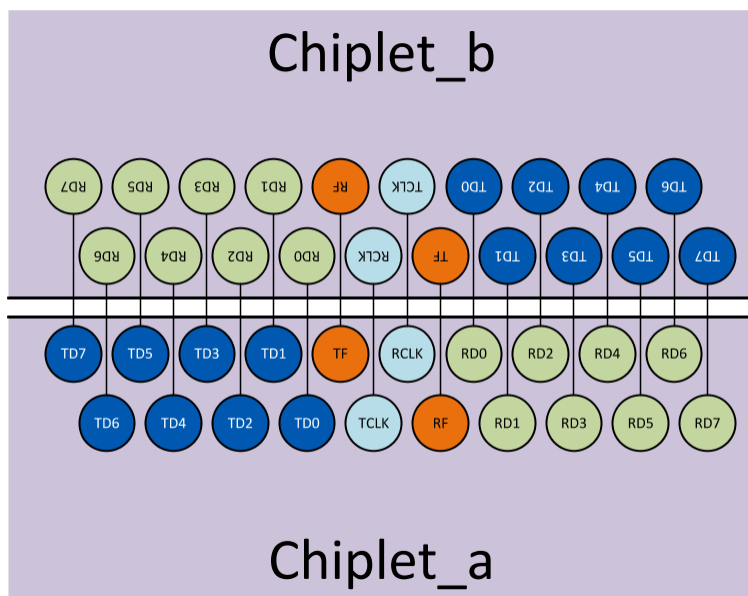
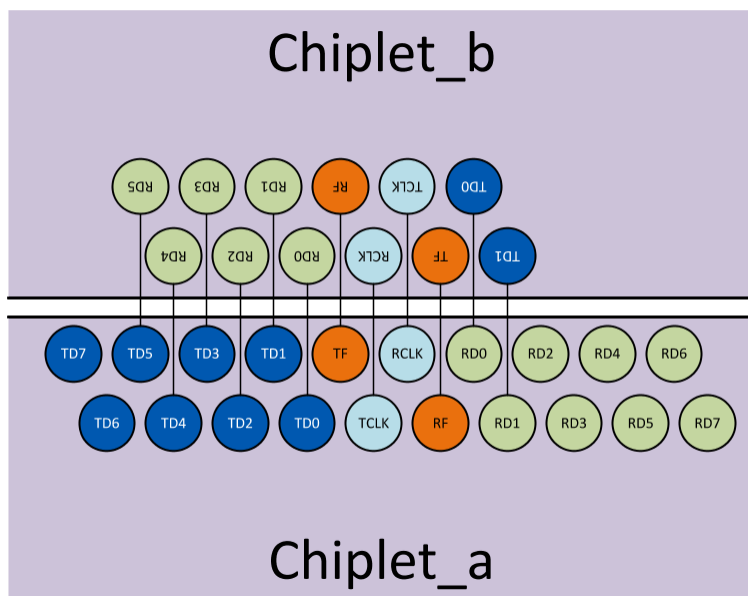


Figure 7.7. BoW.Flexi Pad/Bump Arrangement for x8 Configuration**Figure 7.8. BoW.Flexi Pad/Bump Arrangement with mismatch between Tx and Rx lane counts**

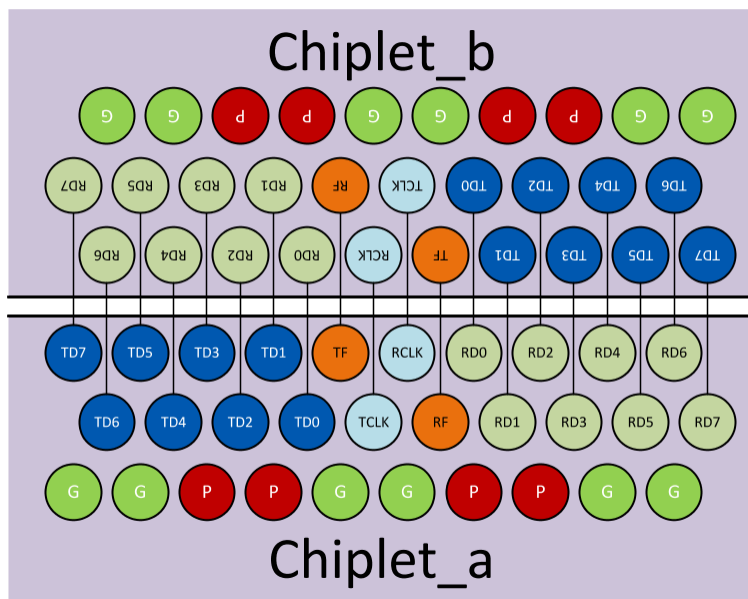
7.9.3. Bump/Pad Arrangements

Note that bump patterns are not specified by BoW; only the signal ordering at the chip edge is specified for interoperability.

7.9.4. Power and Ground

The Power and Ground arrangements are not specified by BoW.Flexi due to the wide variety of possible implementations. However, the following arrangements are recommended if a third row of pads/bumps are added.

Figure 7.9. BoW.Flexi Pad/Bump Arrangement for x8 Configuration with Power and Ground



Chapter 8. Compliance

Compliance to be fully defined in next revision.

8.1. BoW.Flexi Testability

Testing of the BoW.Flexi interface can be as simple as writing and reading configuration registers and comparing the results to the expected values. However, to support more comprehensive testing of the physical interface, the BoW.Flexi Link Interface includes a built-in self-test (BIST) approach for generating pseudo-random bit sequences (PRBS) on the data lanes and checking with bit error rate testing (BERT). This allows for loopback testing at wafer sort [Figure 8.1](#), package / in-field test [Figure 8.2](#).

Figure 8.1. Loopback Concept within a chiplet

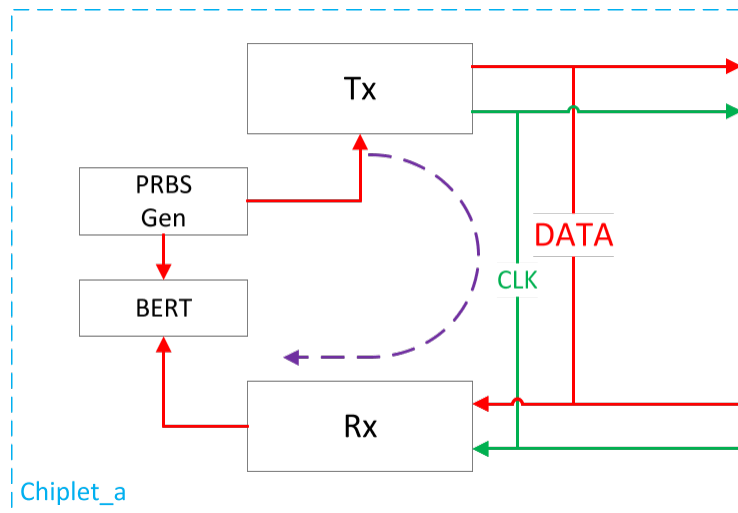
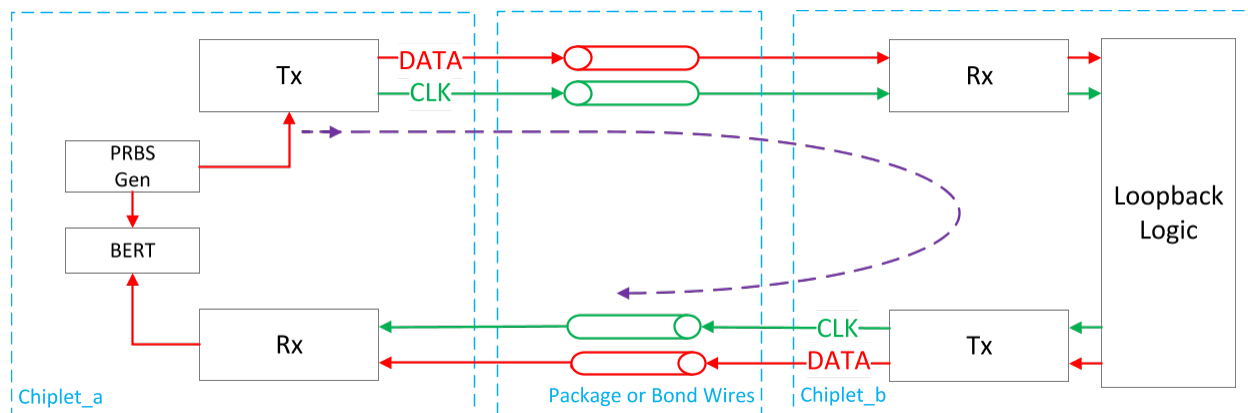


Figure 8.2. Loopback Concept between chiplets



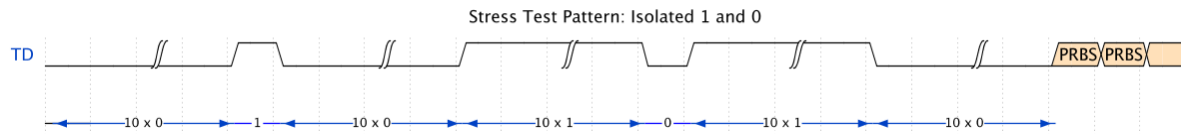
8.1.1. Test Patterns

BoW.Flexi Supports PRBS-31 patterns. Each side of the interface shall support the same set of test patterns. The BoW.Flexi BIST patterns are initialized with the isolated 1 and 0 pattern on all the frame/data lanes, followed by a PRBS pattern for stress testing. This initial pattern allows for synchronization of the start of test on the checker side.

- **Isolated 1 and 0 pattern** to test DC wander and single bit response:

- $['0'] * 10 + ['1'] * 10 + ['0'] * 10 + ['1'] * 10 + ['0'] * 10 + ['1'] * 10$
- This is prepended to a PRBS pattern as seen in [Figure 8.3](#)

Figure 8.3. Stress Test Pattern: Isolated 1 and 0



The pattern generators/checkers shall support the following pattern:

- **PRBS-31 Pattern**, defined by polynomial: $X^{31} + X^{28} + 1$

The PRBS generators shall be configured for the number of data lanes supported. The 31-bit LFSR state is advanced by W steps each cycle to produce W output bits. The frame signal is assigned to the lowest order bit of the PRBS pattern, with the data bits occupying the remaining bits in their respective order.

8.1.2. PRBS Generation

The PRBS pattern generator and checker are clocked by the same clock source as the BoW.Flexi interface. On each interface clock cycle, the generator produces W bits in parallel, where W is the configured interface lane width (1 to 128 lanes). The PRBS sequence is identical to what would be produced by a serial LFSR shifting at W times the interface clock rate.

8.1.2.1. Parallel PRBS Architecture

The PRBS generator uses a single LFSR whose state advances by W positions per interface clock cycle. Rather than shifting the LFSR W times sequentially, the W -step state transition is computed combinatorially using the companion matrix of the selected polynomial raised to the power W :

The LFSR state is 31 bits; the next state after W shifts is a linear function (XOR network) of the current 31-bit state.

The W output bits presented to the interface each cycle are the first W bits of the PRBS sequence starting from the current LFSR state. On the RX side, the checker uses the same parallel computation to predict the expected pattern and compares it against received data.

Since the interface width is configured at link setup and remains fixed during a test, the W -step transition logic may be implemented as a pre-computed XOR matrix selected at configuration time.

8.1.2.2. PRBS Test Duration

The test runs for a programmable number of interface clock cycles controlled by the `test_mode_count` field in [Test Register](#). The total number of PRBS bits exercised per test is:

$$\text{Total bits} = W * 2^{\text{test_mode_count}}$$

The PRBS-31 polynomial ensures that parallel output words do not repeat within practical test durations for all supported interface widths.

8.1.2.3. PRBS Test Register Fields

The register fields controlling PRBS test operation are defined in the following registers:

- **Test Register** — Test enable, loopback mode selection, PRBS polynomial selection, test duration, error count, and test status.
- **Test PRBS Seed Register** — Initial LFSR seed for the PRBS generator and checker.

8.1.2.4. Error Logging and IO Identification

To support IO repair and failure analysis, the PRBS checker records which IO lanes have experienced errors and the cycle count at which the first error on each lane was detected. This information is stored in two error log registers (**Test Error Log Register 0** and **Test Error Log Register 1**), allowing up to two unique failing IOs to be identified per test run.

The frame signal cannot be repaired. If an error is detected on the frame, the IO index is recorded as 0 in the error log. Data lane IO indices start from 1 up to the maximum number of lanes supported (e.g. 128 for a 128-lane interface).

Each error log entry contains:

- **error_log_valid**: Indicates this entry contains a valid error record.
- **error_log_io_index**: The index of the IO lane that failed. Index 0 is the frame; indices 1–128 are the data lanes.
- **error_log_cycle_count**: The interface clock cycle at which the first error was detected on this IO.

Log entries are populated in order: when an error is detected on an IO that has not previously been logged, the next available entry is written with the IO index and current cycle count. If both entries are occupied and a new IO fails, it is not recorded (the **test_error_count** in **Test Register** still increments). The error log registers are cleared when a new test is started (writing **test_enable** = 1).

8.1.2.4.1. IO Masking for Debug

To facilitate debugging when one or more IOs are known to be faulty, the error mask register (**Test Error Mask Register**) allows up to two IO lanes to be masked. When an IO is masked:

- Errors on that IO do not increment **test_error_count**.
- Errors on that IO are not recorded in the error log registers.
- The **test_pass** flag ignores errors on masked IOs.

This enables a second test pass to identify additional failing IOs beyond the initial faults, supporting iterative repair workflows:

1. Run initial PRBS test, read error log to identify first set of failing IOs.
2. Program failing IO indices into the error mask register.
3. Re-run PRBS test to detect additional failures not obscured by the known-bad IOs.

The error log and mask register fields are defined in the following registers:

- [Test Error Log Register 0](#) and [Test Error Log Register 1](#) — Per-IO error records capturing the failing IO index and the cycle count of the first error.
- [Test Error Mask Register](#) — IO mask configuration for suppressing known-bad lanes during subsequent test passes.

8.1.3. Test Approach

The BoW.Flexi IOs support bidirectional data transmission and reception, enabling the loopback paths introduced above (wafer sort, package, and in-field test). The frame/data PRBS pattern is launched aligned with the rising edge of the clock. Two modes for data sampling are available:

1. Flow-Through (Default)
2. Rx Sampled

In Flow-Through mode the frame/data is launched with the rising edge of the clock and flows through to the loopback pathway where it is sampled at the BERT. The sampling clock at the BERT can be adjusted using `test_clk_sample_adj` to test the skew between the signal as well as the clock delay through the loop. This data can be used to determine if there are any defects along the signal path. The configuration of the `test_clk_sample_adj` is implementation dependent.

In the Rx sampled mode, the receivers on BoW.Flexi sample the frame/data on the falling edge of the received clock. Latency may be added to re-synchronize the frame/data with the rising edge of the clock. The frame/data may traverse two receivers before getting sampled at the BERT. The latency of each of the stages shall be accounted for. When testing asymmetrical slices, the additional data bits will bypass one of the receivers and therefore can have a different latency.

This allows for testing of the timing margin of the interface by effectively shifting the sampling point by half a UI relative to the default rising edge sampling. In Flow-Through mode, the received data is not resampled and is instead passed through directly to the output registers. This allows for testing of the data path without any timing shift.

BoW.Flexi supports testing of the interface at:

1. Wafer sort
2. Package test
3. In-field test

8.1.3.1. Wafer Sort Test

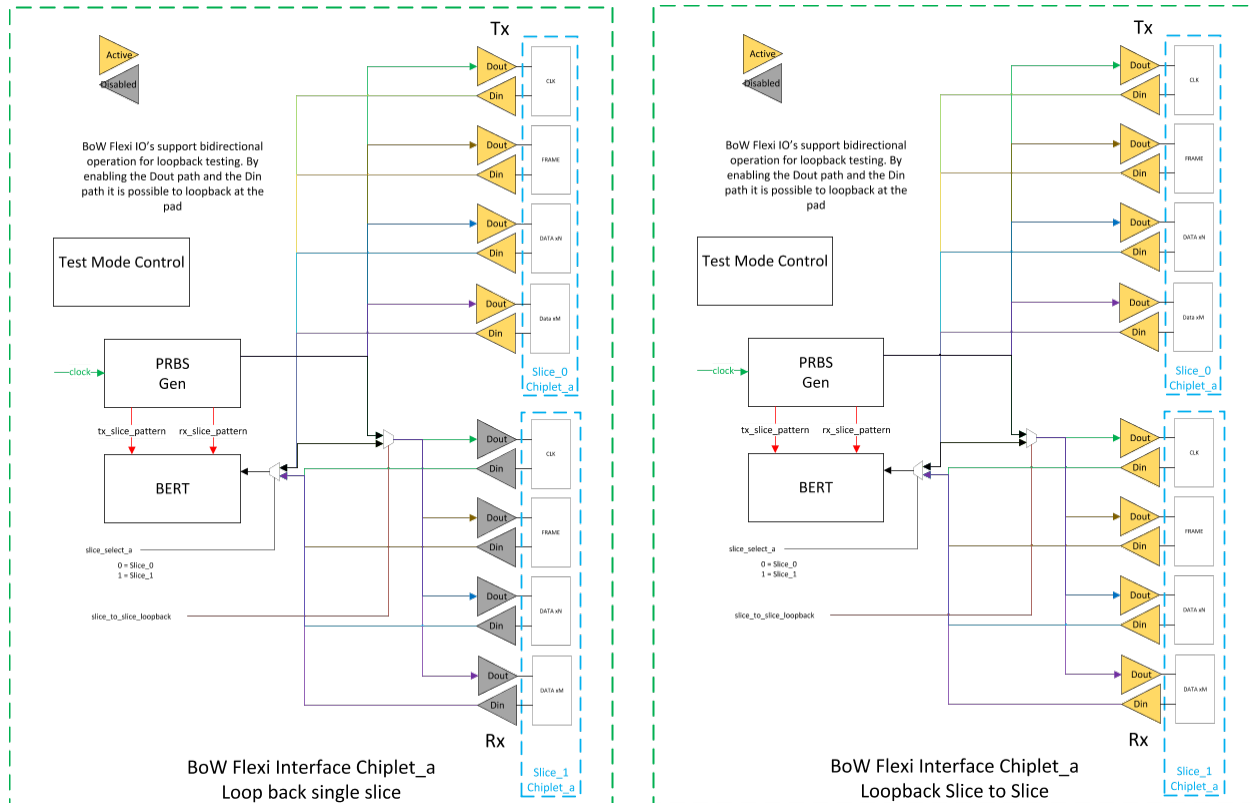
Wafer sort exercises the BoW.Flexi I/O before the chiplet is packaged.

1. **Single-slice loopback:** enable the Dout and Din paths in the I/Os of one BoW.Flexi slice and loop a PRBS pattern from the slice's TX pads back to the same slice's RX pads via the probe card.
2. Configure the BERT checker to compare the looped pattern against the expected PRBS sequence.
3. **Two-slice loopback (same chiplet):** loop a pattern from the first BoW.Flexi slice's TX through the second BoW.Flexi slice and back to the BERT checker. Verifies inter-slice routing in addition to the slice itself.
 - a. **Requires additional logic on die to forward data from one slice to the next**



For asymmetrical slices, the PRBS bits on the extra (unmatched) IOs traverse only one IO pad, so their effective data path is shorter than the matched bits.

Figure 8.4. Wafer sort test options: Single Slice testing (Left), Two Slice testing (Right)

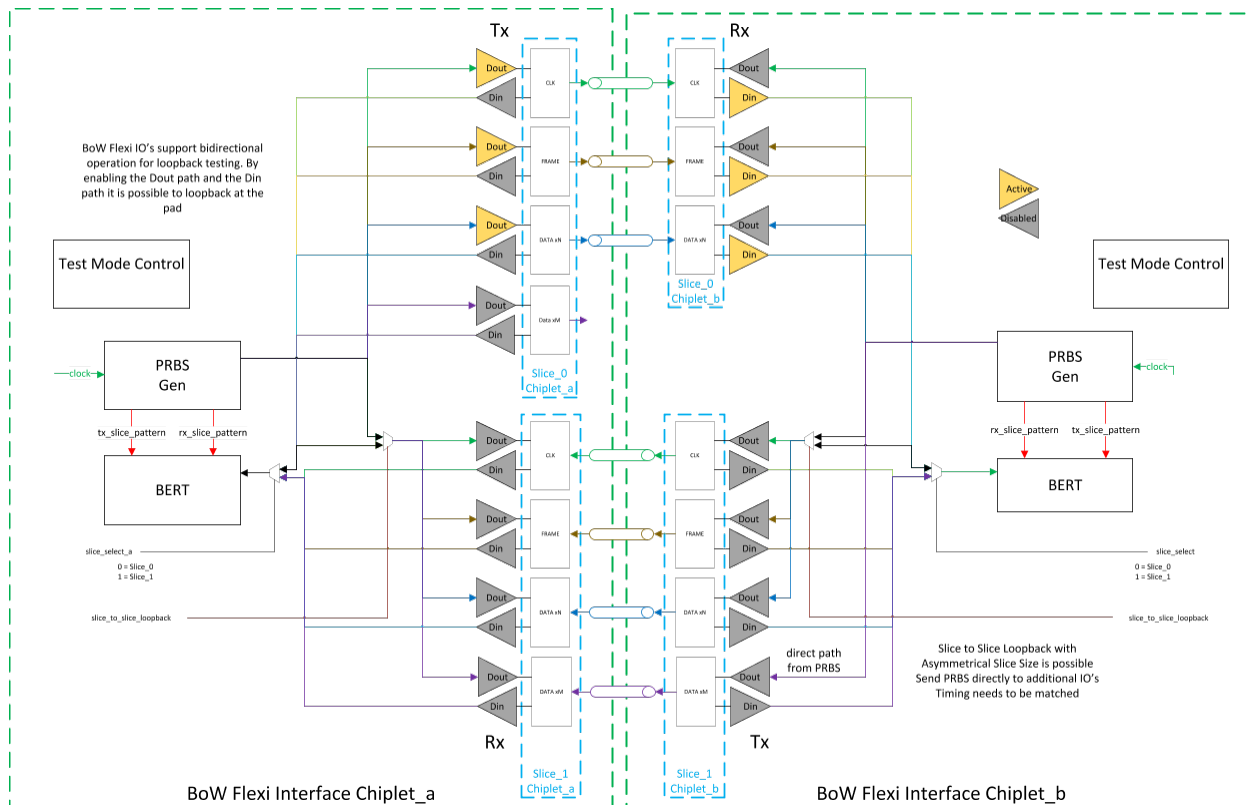


8.1.3.2. Package Test

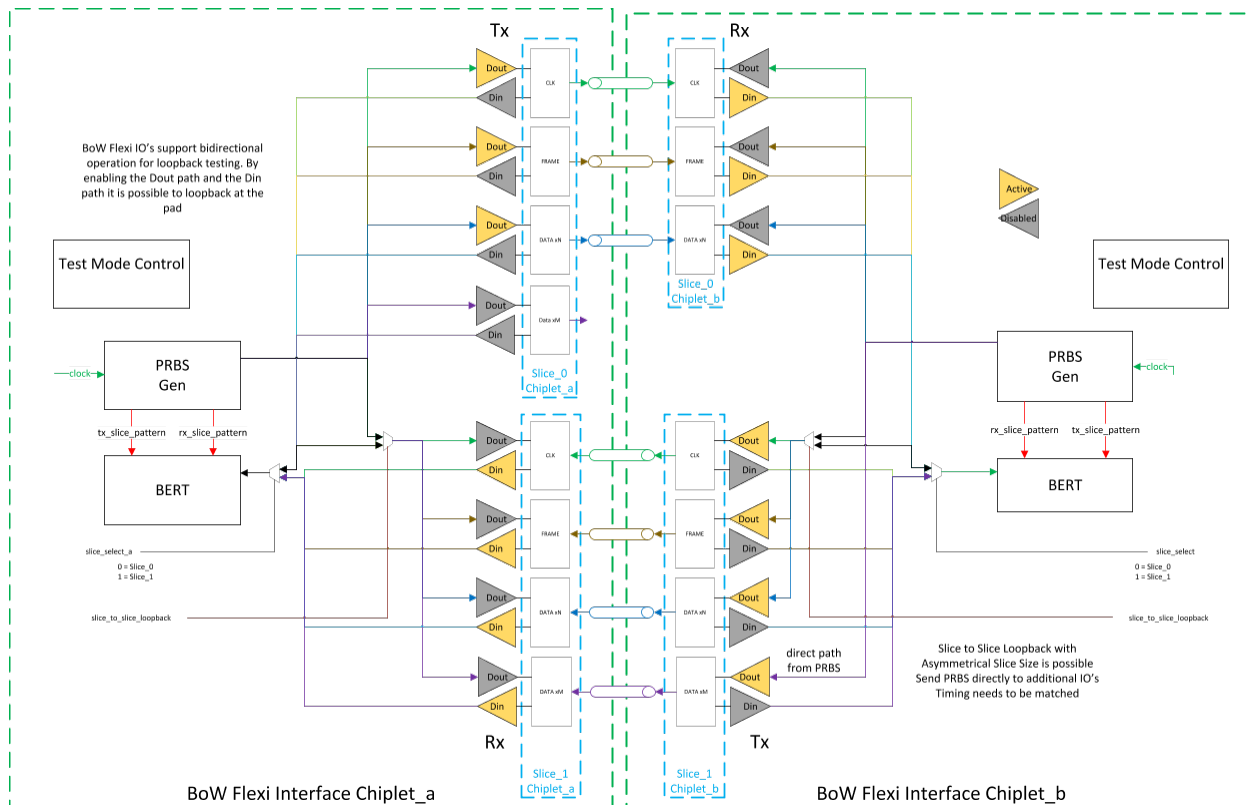
Two procedures are defined for in-package test.

Option 1 — one-way slice-to-slice (chiplet_a Tx → chiplet_b Rx):

1. Loop data from Slice_0 TX on chiplet_a to Slice_0 RX on chiplet_b.
2. Synchronize the PRBS generators on chiplet_a and chiplet_b, and synchronize **test_mode_count** between chiplets.
3. Sample frame/data at the chiplet_b BERT for the duration of **test_mode_count**.
4. On completion, chiplet_a reads **Test Register** on chiplet_b to retrieve the result.
5. Repeat with TX/RX roles swapped to cover the reverse direction.

Figure 8.5. Package testing from Chiplet_a to Chiplet_b**Option 2 — round-trip loopback (covers the full data path in one run):**

1. Enable loopback Slice_0 (chiplet_a) → Slice_0 (chiplet_b) → Slice_1 (chiplet_b) → Slice_1 (chiplet_a) into the chiplet_a BERT.
2. Synchronize PRBS and **test_mode_count** between the two chiplets.
3. Capture the pattern at the chiplet_a BERT.
4. On completion, chiplet_a reads its local **Test Register** to retrieve the result.

Figure 8.6. Package testing from Chiplet_a to Chiplet_b loopback to chiplet_a

8.1.3.3. In-Field Loopback Testing

The same approach as the package test can be used for in-field testing. This can be run periodically to verify the link is operating correctly.

8.2. JTAG Transport over BoW.Flexi

JTAG-over-BoW.Flexi is an **optional** capability of the Physical Layer. A chiplet is not required to implement it; a chiplet that does implement it shall comply with this section. The feature carries a standard IEEE 1149.1 JTAG interface across a BoW.Flexi link so that a die-to-die partner can be reached for boundary scan, structural test and debug, following the multi-die test model of IEEE 1838.

The optional IEEE 1149.1 TRST signal is **not supported**: TAP reset is performed with TMS (see [JTAG Reset](#)).

8.2.1. Motivation and Scope

Multi-die systems still need a JTAG path to every chiplet for boundary scan, structural test and debug. Rather than routing dedicated JTAG wires to each die, the feature **reuses the existing BoW.Flexi die-to-die pins** to transport the four standard JTAG signals (TCK, TMS, TDI and TDO). The chiplet that owns the external JTAG header is the **JTAG manager**; each die reached across a link is a **JTAG subordinate**. When the feature is enabled the manager and subordinate TAPs are joined into a single IEEE 1149.1 scan chain, extended across further links to reach every chiplet in the system.

The feature is deliberately narrow in scope:

- It lives entirely in the **Physical Layer**. When JTAG mode is active the PHY acts as a transparent buffer for the JTAG signals; the serializer, Link Layer and Transaction Layer are bypassed and **no data is delivered to the higher layers**.
- BoW.Flexi transports only the primary IEEE 1149.1 signal set. Any IEEE 1838 hierarchical scan (primary/secondary TAP selection, the Scan Interface Port state machine) is layered **on top** of this transport and shall be provided by the on-die TAP logic; it is outside the scope of BoW.Flexi.

8.2.2. Signal Mapping

JTAG transport carries four signals: TCK, TMS and TDI flow **into** the device under test, and TDO flows **out**. A BoW.Flexi PHY exposes, per direction, a clock, a frame and one or more data lanes (Tx: TCLK, TF, TD0; Rx: RCLK, RF, RD0). JTAG transport only needs a single lane (x1), so it fits within the six wires of a minimum-width link.

Because the two PHYs are cross-coupled (each Tx pin drives the partner's matching Rx pin), the pin roles are **mirrored** between the manager and the subordinate. The table below is therefore given per side rather than as a single mapping.

Table 8.1. JTAG signal mapping to BoW.Flexi PHY pins

JTAG signal	Direction (at DUT)	Manager PHY pin	Subordinate PHY pin	Notes
TCK	into DUT	TCLK (Tx out)	RCLK (Rx in)	Manager forwards TCK; arrives on the subordinate RCLK pin.
TMS	into DUT	TF (Tx out)	RF (Rx in)	
TDI	into DUT	TD0 (Tx out)	RD0 (Rx in)	

JTAG signal	Direction (at DUT)	Manager PHY pin	Subordinate PHY pin	Notes
TDO	from DUT	RD0 (Rx in)	TD0 (Tx out)	Manager reads TDO; already the natural Rx/Tx direction.
—	unused	RCLK, RF (Rx)	TCLK, TF (Tx)	Unused by the JTAG transport on both ends.



Every transported signal keeps its natural BoW.Flexi direction — there is no bidirectional pin. RD0 / TD0 carry TDO exactly as the manager reads receive data. The manager's RF receive pin and the subordinate's TF transmit pin are unused by the JTAG transport.

8.2.3. Register Control

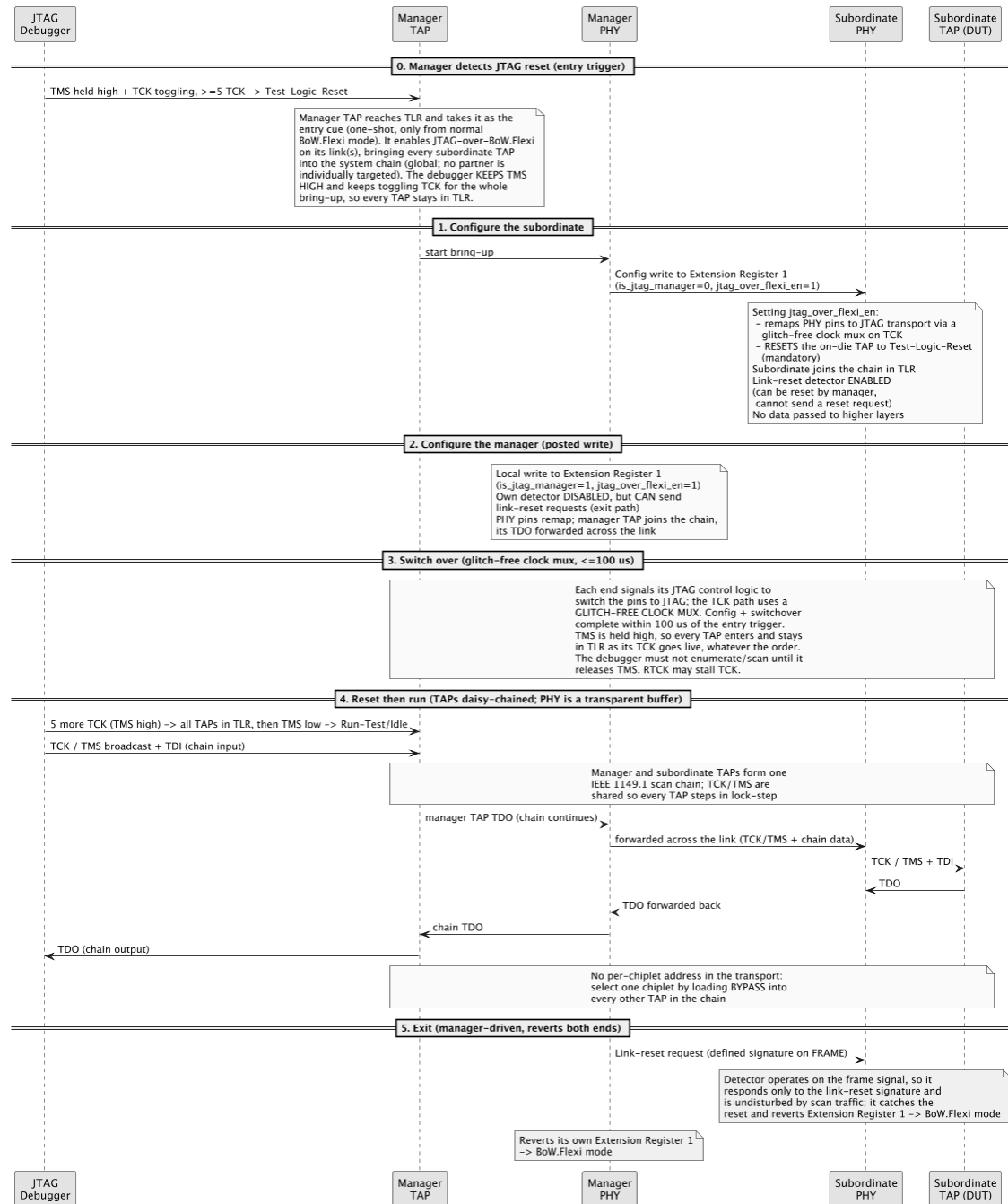
The mode is selected by two bits of the [Interface Configuration Extension Register 1](#):

- **Bit 31 — jtag_over_flexi_en**: when set to 1 the Physical Layer shall switch the pins to the JTAG transport defined above and shall bypass the serializer and higher layers, so that no data is delivered to the Link or Transaction Layers. The bit reverts to 0 (normal BoW.Flexi operation) on link reset.
- **Bit 30 — is_jtag_manager**: selects this end's role — 0 = JTAG subordinate (default), 1 = JTAG manager. It selects the mirrored pin map for that side and governs the link-reset behavior: the subordinate **shall keep its link-reset detector enabled** (so the manager can reset it out of JTAG mode) but **shall not issue link-reset requests**, whereas the manager **shall disable its own link-reset detector** but **retains the ability to send link-reset requests**, which provides the exit path.

JTAG Reset No initial-state or per-die addressing field is needed. There is a **single** TAP reset mechanism: TMS held high for at least five TCK cycles drives every TAP to Test-Logic-Reset, as defined by IEEE 1149.1. In addition, setting `jtag_over_flexi_en` on a subordinate shall reset its on-die TAP to Test-Logic-Reset, so the subordinate joins the chain in reset — the same state as the manager TAP — before its TCK goes live. The Bring-up Sequence below covers the entry trigger, the glitch-free pin switchover and the required bring-up time.

8.2.4. Bring-up Sequence

Figure 8.7. BoW.Flexi JTAG transport bring-up sequence



JTAG entry is a **system-wide** event, not a per-chiplet one. The trigger is the debugger's standard TAP reset: when the manager's own TAP is driven to Test-Logic-Reset by TMS held high for at least five TCK cycles, the manager takes this as the cue to enter JTAG mode and enable the transport on its BoW.Flexi link(s), bringing every reachable subordinate TAP into one scan chain. This trigger fires only on the transition **from** normal BoW.Flexi operation; once JTAG mode is active, later resets are simply forwarded across the link and do not re-run bring-up. Requiring a clocked TMS sequence, rather than an asynchronous power-up reset, ensures the

manager enters JTAG mode only when a debugger is actually driving TCK. The debugger shall **keep TMS high and continue toggling TCK** for the whole bring-up: because TMS=1 holds every TAP in Test-Logic-Reset, subordinates may switch over at different times and still remain in reset until the debugger releases TMS. No individual die is, or can be, targeted at this stage; a specific chiplet is selected later with BYPASS (see [Section 8.2.5](#)).

On this trigger the manager brings the link up in the order below. All JTAG-mode configuration writes are **posted**: no response is returned and the initiator shall not wait for one.

1. **Configure the subordinate.** The manager shall issue a posted Configuration write to the subordinate's Interface Configuration Extension Register 1 with `is_jtag_manager=0` and `jtag_over_flexi_en=1`. Setting `jtag_over_flexi_en` shall remap the subordinate's pins to the JTAG transport through a **glitch-free clock mux** on the TCK path (see below) **and reset its on-die TAP to Test-Logic-Reset**. The subordinate therefore joins the chain in Test-Logic-Reset, the same state as the manager TAP. The subordinate shall keep its link-reset detector enabled — so the manager can later reset it out of JTAG mode — but shall not issue a link-reset request of its own.
2. **Configure the manager.** The manager shall post a write to its own Interface Configuration Extension Register 1 with `is_jtag_manager=1` and `jtag_over_flexi_en=1`, remapping its own pins to the JTAG transport and joining its TAP to the chain. The manager shall disable its own link-reset detector so returning TDO traffic is not misread as a reset pattern, but shall retain the ability to send link-reset requests.
3. **Switch over and assemble the chain.** Each end completes its internal configuration and then signals its JTAG control logic to switch the pins to the JTAG transport. The TCK path shall be switched with a **glitch-free clock mux** so that no runt or partial TCK edge reaches the TAP during the transition; the TAP TCK may be held gated until the path is stable and released while TMS is high. Configuration and switchover shall complete within a **bring-up time** of at most **100 us** from the entry trigger. Because TMS is held high throughout, every subordinate TAP enters and remains in Test-Logic-Reset as its TCK goes live, regardless of the order in which the ends switch over. The debugger shall not enumerate or scan the chain until it releases TMS. Where available, adaptive clocking (RTCK) may be used to stall TCK until switchover completes.
4. **Reset and run.** With the chain assembled and TMS still high, the debugger shall apply at least **five further TCK cycles** to guarantee every TAP — including the last to join — is in Test-Logic-Reset. Driving TMS low then advances the whole chain in lock-step to Run-Test/Idle and normal IEEE 1149.1 operation begins: TCK and TMS are broadcast to every TAP and the scan data is daisy-chained — TDI enters the manager's TAP, its TDO is forwarded across the link to the subordinate's TDI, and the subordinate TDO returns across the link to the external TDO.
5. **Exit.** The manager shall send a link-reset request. The subordinate link-reset detector operates on the FRAME signal, so it responds only to the defined link-reset signature and is undisturbed by ordinary scan traffic; the manager drives that signature on the shared pins to trigger the exit. Each subordinate's enabled detector catches it and reverts its Interface Configuration Extension Register 1; the manager reverts its own. All ends return to BoW.Flexi operation together.

8.2.5. Device Selection

The transport carries **no chiplet address**: the manager TAP and every subordinate TAP form a single IEEE 1149.1 daisy chain sharing TCK and TMS, with scan data flowing through each TAP in turn. A particular chiplet is selected the standard 1149.1 way — by loading the BYPASS instruction into every **other** TAP in the chain. BYPASS reduces a TAP to its single-bit bypass register, so the non-selected chiplets add only one shift cycle each and leave their own registers undisturbed, while the targeted chiplet holds its real instruction (EXTEST, SAMPLE/PRELOAD, IDCODE, ...).

Instructions are loaded the standard IEEE 1149.1 way, by the debugger walking the TAPs through **Shift-IR/Update-IR**. Because the PHY is a transparent buffer once `jtag_over_flexi_en` is set, those scans cross the link exactly as on a directly wired chain, and BoW.Flexi neither interprets nor stores them. The debugger knows the chain composition — device order and instruction-register lengths — which it discovers by the usual IDCODE/BSDL scan.

8.2.6. Clocking and Datapath

In JTAG mode the Physical Layer operates as a set of direct IO buffers: the serializer, ECC, lane padding and clock-domain-crossing logic are bypassed and the four JTAG signals pass through. Entry to and exit from JTAG mode shall switch the TCK path with a **glitch-free clock mux** so that no runt or partial TCK edge reaches the TAP. TCK may therefore be slow and asynchronous to HCLK and to the normal BoW.Flexi datapath; the frequency-related TCLK/HCLK requirement of [Section 7.7](#) does not apply while `jtag_over_flexi_en` is set. No BoW.Flexi packets, framing or ECC are generated or interpreted on the link in this mode.

Chapter 9. BoW.Flexi Electrical Specifications

9.1. BoW.Flexi Slice Electrical Specifications

BoW.Flexi slices shall meet the TX and RX termination resistance requirements for unterminated or source-terminated operation as specified below.

Table 9.1. BoW.Flexi TX and RX Termination Resistance Requirements

Parameter	Unterminated	Source-Terminated
TX DC Termination	As required to meet TX rise-time	$36\ \Omega - 50\ \Omega$ ($0.72 - 1.0\ Z_{chan}$)
RX DC Termination	–	–
Within-Slice DC Termination Matching	–	$\sigma = 1.333\%$ (8% over 6σ)

9.1.1. Parasitic Capacitance

Since BoW.Flexi slices are targeted for relatively dense and simple realizations, it is expected that the primary frequency-dependent parasitics seen at a slice's I/Os will be capacitive in nature. The maximum "equivalent" capacitance allowed on each side of each BoW.Flexi I/O is specified below. It is recommended that implementations retain as low a capacitance as practical to reduce power consumption and improve signal integrity.

Table 9.2. BoW.Flexi Maximum Parasitic Capacitance

Parameter	Requirement
Maximum Equivalent Capacitance (TX or RX)	800 fF



This capacitance limit is identical to BoW-32/BoW-64 mainband slices.

9.2. BoW.Flexi Slice Receiver Bandwidth

A BoW.Flexi slice receiver shall maintain an effective 3dB bandwidth of at least $(1.0/T_{bit})$ Hz. For example, for a slice that supports up to the maximum rate of 1 Gb/s, the receiver bandwidth should be at least 1 GHz.

9.3. BoW.Flexi Slice Timing Specifications

The following timing definitions and requirements apply to BoW.Flexi transmitters and receivers. Both TX and RX voltage and timing error component requirements account for deterministic (bounded) terms separately from random (unbounded) terms. A bound is placed on both the maximum deterministic error and on the maximum total error budget at the target per-wire error rate of 1e-25.



The 1e-25 error rate is a **per-wire design margin target** defined for an isolated data lane (single clock, single data). It is validated through voltage and timing margin analysis, not through direct bit-error-rate testing. Compliance is demonstrated through margin-based methods (see [Section 9.5](#)).

For Gaussian random noise, the relationship between the target BER and the required peak-to-peak noise margin is:

$$\sigma_{\text{multiplier}} = 2 * Q^{-1}(\text{BER}/2)$$

where Q^{-1} is the inverse of the Gaussian Q-function. For BER = 1e-25, this yields a peak-to-peak noise budget of 20.85 σ .

9.3.1. Multi-Lane Impact on Bit Error Rate

BoW.Flexi supports data lane widths from x1 to x128. The per-wire BER target of 1e-25 is defined for an isolated lane pair (one clock, one data) free from inter-lane interference. As the lane count increases, inter-lane physical effects consume voltage and timing margin, causing the **effective per-wire BER to degrade** — an expected and accepted consequence of wider interfaces. The specific mechanisms and per-lane-count design guidance are given in [Section 9.3.2](#).

The system-level BER (probability of error on any data lane per UI) is:

$$\text{BER}_{\text{system}} = N_{\text{data}} * \text{BER}_{\text{per_wire}}$$

where N_{data} is the number of data lanes. The frame signal is excluded from this calculation as it is sampled at a different rate and does not carry payload data.

Table 9.3. System BER vs. Per-Wire BER for Different Lane Counts

Data Lanes	Per-Wire = 1e-25	Per-Wire = 1e-22	Per-Wire = 1e-20	Per-Wire = 1e-18
x1	1e-25	1e-22	1e-20	1e-18
x2	2e-25	2e-22	2e-20	2e-18
x4	4e-25	4e-22	4e-20	4e-18
x8	8e-25	8e-22	8e-20	8e-18
x16	1.6e-24	1.6e-21	1.6e-19	1.6e-17

Data Lanes	Per-Wire = 1e-25	Per-Wire = 1e-22	Per-Wire = 1e-20	Per-Wire = 1e-18
x32	3.2e-24	3.2e-21	3.2e-19	3.2e-17
x64	6.4e-24	6.4e-21	6.4e-19	6.4e-17
x128	1.3e-23	1.3e-20	1.3e-18	1.3e-16

9.3.2. Multi-Lane Operation — Margin and Design Guidance

As lane count increases, three physical effects reduce the voltage and timing margins available to each data lane:

- **Crosstalk:** Each active data lane acts as an aggressor to its neighbors. Both near-end crosstalk (NEXT) and far-end crosstalk (FEXT) inject bounded noise into the victim lane. Crosstalk amplitude depends on trace spacing, coupling length, and termination but is **deterministic** (worst case bounded by the all-ones/all-zeros aggressor pattern).
- **Simultaneous Switching Noise (SSN):** When N drivers switch simultaneously, the transient current through shared power/ground inductance creates supply bounce proportional to $N * di/dt$. For random data SSN scales approximately as $\sqrt{N_{data}}$; for pathological patterns it scales as N_{data} .
- **Power delivery:** Wider interfaces draw more supply current, increasing IR drop and reducing available voltage swing. This is bounded and treated as a deterministic penalty.

These effects reduce the effective margin and therefore degrade the per-wire BER from the isolated-lane ideal of 1e-25. At the steep tail of the Gaussian distribution, small margin losses cause large BER changes (at the 1e-25 operating point one-sided margin is 10.43 σ):

$$BER \approx \text{erfc}((V_{\text{margin_remaining}}) / (\sigma_{\text{noise}} * \sqrt{2}))$$

Table 9.4. Margin Loss vs. Effective Per-Wire BER

Margin Loss	Remaining Margin	Effective Per-Wire BER
0% (isolated)	10.43 σ	1e-25
2%	10.22 σ	~1e-24
5%	9.91 σ	~1e-23
10%	9.39 σ	~1e-21
15%	8.87 σ	~1e-19
20%	8.34 σ	~1e-17

Design guidance by lane count:

- **x1 to x4:** Crosstalk and SSN penalties are negligible for typical die-to-die routing. Implementations should achieve per-wire BER close to the 1e-25 isolated-lane target. Deterministic budgets as specified.

- **x8 to x16:** Crosstalk from immediate neighbors becomes relevant. SSN from 8–16 simultaneous transitions is manageable with adequate decoupling. Typical margin loss 2–5%; expected per-wire BER 1e-24 to 1e-23. Implementations should ensure adequate ground/power plane routing between signal traces.
- **x32 to x64:** Crosstalk mitigation (increased trace spacing, shielding, or staggered routing) might be necessary. SSN requires careful decoupling design. Typical margin loss 5–10%; expected per-wire BER 1e-23 to 1e-21. PDN impedance should be characterized at the switching frequency.
- **x128:** Significant design attention required for crosstalk, SSN, and power delivery. Typical margin loss 10–15%; expected per-wire BER 1e-21 to 1e-19, yielding a system BER of approximately 1e-19 to 1e-17.



The per-wire BER values above are guidance based on typical die-to-die implementations. Actual values depend on routing geometry, substrate technology, and power delivery design. Implementations should verify per-wire margins through simulation with all lanes active under worst-case data patterns.

9.3.3. Transmitter Requirements

- **Maximum Rise-Time:** The maximum 20%–80% rise-time at the output of a BoW.Flexi TX shall not exceed 11.5% of a UI. This rise-time shall be simulated with the TX (including all of its parasitics) driving an ideal load of 50 Ω (Z_{chan}).
- **Deterministic Timing Error:** The total deterministic (bounded) timing errors introduced by the TX shall not exceed 18% of a UI peak-to-peak. The evaluation of these timing errors shall include all possible deterministic contributors, such as reference clock, clock distribution networks, duty cycle error (deviation from 50% duty cycle), skew between CLK and any D line, and power supply variation induced jitter or skew. This is a peak-to-peak requirement — if a given design has $\pm 5\%$ UI of duty cycle error, this implies a TX deterministic timing error of no better than 10% UI.
- **Total Timing Error:** The total timing error introduced between the CLK and any data (D) line at the output of the TX shall not exceed 29% of a UI peak-to-peak at an error rate of 1e-25. The evaluation of errors shall encompass all possible deterministic as well as random timing error contributors. Assuming a Gaussian distribution for the random jitter, the total timing error $t_{\text{err,tot}}$ can be computed as:

$$t_{\text{err,tot}} = t_{\text{err,deterministic}} + 20.85 * \sigma_{\text{tj,random}}$$

9.3.4. Receiver Requirements

- **Clock versus Data Path Delay:** Data shall be captured on the same cycle as launch (the CLK edge that launched the data is the CLK edge used to capture it).
- **Maximum RX Deterministic Voltage Error ($V_{\text{err,det,RX}}$):** Shall include all deterministic voltage errors that would shift the receiver's voltage threshold relative to its ideal position in the middle of the signal swing (e.g. residual offset, reference level error, supply noise). This specification accounts for double-sided loss in margin.
- **Maximum Total Required RX Voltage Margin ($V_{\text{err,tot,RX}}$):** Shall include all possible voltage errors at the RX at a probability of 1e-25 or higher. For Gaussian random voltage noise, the total required voltage margin can be computed as:

$$V_{\text{err,tot,RX}} = V_{\text{err,det,RX}} + 20.85 * \sigma_{\text{Verr,random,RX}}$$

- **Maximum RX Deterministic Timing Error ($t_{err,det,RX}$):** Shall include all deterministic timing errors that would shift the receiver's sampling timing relative to its ideal position for any data line (e.g. residual sample timing position error, DLL dither, power supply induced jitter). This specification accounts for double-sided loss in margin.
- **Maximum Total RX Timing Error ($t_{err,tot,RX}$):** Shall include all possible timing errors at the RX at a probability of 1e-25 or higher. For Gaussian random jitter, the maximum total required timing error can be computed as:

$$t_{err,tot,RX} = t_{err,det,RX} + 20.85 * \sigma_{terr,random,RX}$$

Table 9.5. BoW.Flexi Slice Receiver Voltage and Timing Requirements

Parameter	Value	Unit
$V_{err,det,RX}$ (Max RX Deterministic Voltage Error)	250	mV
$V_{err,tot,RX}$ (Max Total RX Voltage Margin)	355	mV
$t_{err,det,RX}$ (Max RX Deterministic Timing Error)	28%	T_{bit}
$t_{err,tot,RX}$ (Max Total RX Timing Error)	36.5%	T_{bit}

9.4. BoW.Flexi Slice Channel Requirements

This specification does not place any direct requirements on characteristics such as channel loss or crosstalk for BoW.Flexi slice channels. Instead, BoW.Flexi channels are considered conforming if they can achieve the required per-wire error rate of 1e-25 for an isolated lane in conjunction with reference transmitters and receivers that meet all of the requirements provided in [Section 9.1](#) and [Section 9.3](#).

For multi-lane configurations, the channel routing shall ensure that crosstalk and SSN penalties remain within the margins described in [Section 9.3.1](#). Implementations targeting wider lane configurations (x32 and above) should characterize crosstalk and power delivery impedance through electromagnetic simulation with all lanes active.

9.5. BoW.Flexi Compliance Testing

Since the 1e-25 per-wire BER target cannot be verified through direct bit-error-rate measurement, compliance shall be demonstrated through the following methods:

9.5.1. Margin-Based Compliance (Recommended)

The primary compliance method is voltage and timing margin analysis. An implementation is compliant if:

1. The measured or simulated voltage eye opening at the receiver exceeds:

$$V_{eye_min} = 2 * V_{err,tot,RX} = 710 \text{ mV (peak-to-peak)}$$

2. The measured or simulated timing eye opening at the receiver exceeds:

$$t_{\text{eye_min}} = 1.0 - 2 * t_{\text{err,tot,TX}} - 2 * t_{\text{err,tot,RX}} = 27\% \text{ UI (peak-to-peak)}$$

where both the TX and RX contributions (deterministic plus $20.85 * \text{random}$) are subtracted from the unit interval.

3. For x1 configuration: the above margins are maintained under worst-case supply voltage and temperature.
4. For multi-lane configurations: the above margins are evaluated with **all lanes simultaneously active** under worst-case data patterns to capture crosstalk and SSN effects. Margin degradation relative to the isolated-lane case shall be documented.

9.5.2. Characterization (Recommended)

For interface characterization, a directly-testable BER of **1e-12** is recommended as a pass/fail criterion per lane. This rate requires approximately 10,000 seconds (~2.8 hours) per lane at 1 Gb/s (10^{13} bit sample).

Table 9.6. Recommended Characterization Test Parameters

Parameter	Value
Characterization BER threshold	1e-12 per wire
Minimum sample size per lane	10^{13} bits
Test pattern	PRBS-31
Test condition	Nominal voltage, maximum frequency, all lanes active



The characterization test shall be performed with all configured lanes active simultaneously to capture crosstalk and SSN effects. Testing a single lane in isolation does not verify multi-lane operation.

9.5.3. Bathtub Curve Extrapolation (Optional)

Implementations may additionally demonstrate compliance by measuring the BER bathtub curve (BER vs. sampling point offset) at achievable error rates (1e-6 to 1e-12) and extrapolating to determine the margin at the target BER. The worst-case lane (typically the innermost lane with maximum aggressor exposure) shall be used for characterization.

Chapter 10. BoW.Flexi Registers

This section documents a set of local registers for interfacing with a BoW.Flexi link configuration. These registers will be visible to the local CPU/microcontroller and can be accessed by both manager and subordinate from across a BoW.Flexi interface.



The BoW.Flexi registers are defined here as a standalone block. The register set is logically owned by the Transaction Layer's Config/Error engine, but a System Layer implementation may **host** these registers within its own address map — exposing them alongside or combined with the system register space — or they may stand alone. The register definitions and offsets below are independent of that hosting choice.

Every chiplet requires a Tx and Rx slice. The BoW.Flexi configuration protocol is managed through a series of register interactions. On a BoW.Flexi interface, remote registers of the companion chiplet are accessed through the Configuration Command. The BoW.Flexi interface supports three kinds of transactions for accessing these registers:

1. Configuration Read requests
2. Configuration Write requests (including error reporting)
3. Configuration responses to read request

Correspondingly, a BoW.Flexi interface requires a minimum of 5 Read/Write registers for operation, along with 3 Read Only registers for discovery:

The Read/Write registers are as follows

1. A register to enable local BoW.Flexi interactions (config/stat)
2. A register to configure the interface
3. A Read Data register where data returned from a read request is staged
4. A Write Data register where write data to be transmitted is staged
5. A Local Error Code register for reporting errors detected in a received message e.g. uncorrectable ECC.
 - a. It is up to the system to define what actions are taken when an error is detected.
 - i. The content of this register could be used to send error response packets back to the remote side.

The [Local Register Address Map](#) details the registers in each chiplet BoW.Flexi Link Configuration accessible by local firmware controlling the interface.

Table 10.1. BoW.Flexi Register Address Map

Register	Required	Address	Link
BoW.Flexi Configuration and Status Register	1b'1	0x00	Table 10.3
BoW.Flexi Write Data Register	1b'1	0x04	Table 10.4

Register	Required	Address	Link
BoW.Flexi Read Data Register	1b'1	0x08	Table 10.5
BoW.Flexi Interface Configuration Register	1b'1	0x0C	Table 10.6
BoW.Flexi Remote Error Register	1b'0	0x10	Table 10.7
BoW.Flexi Local Error Register	1b'1	0x14	Table 10.8
BoW.Flexi Discovery Register	1b'1	0x18	Table 10.9
BoW.Flexi Vendor Identity Register	1b'1	0x1C	Table 10.10
BoW.Flexi Device Identity Register	1b'1	0x20	Table 10.11
BoW.Flexi Version Register	1b'0	0x24	Table 10.12
BoW.Flexi Virtual Wire Out Register	1b'0	0x28	Table 10.13
BoW.Flexi Virtual Wire In Mask Register	1b'0	0x2C	Table 10.14
BoW.Flexi Interface Configuration Extension Register 1	1b'0	0x30	Table 10.15
BoW.Flexi Interface Configuration Extension Register 2	1b'0	0x34	Table 10.16
BoW.Flexi Extended Data Lane Register	1b'0	0x38	Table 10.17
BoW.Flexi Optional Field Configuration Register	1b'0	0x3C	Table 10.18
BoW.Flexi Optional Field Extended Configuration Register	1b'0	0x40	Table 10.19
BoW.Flexi Optional Field Trailer Configuration Register	1b'0	0x44	Table 10.20
BoW.Flexi Power State Register	1b'0	0x48	Table 10.21
BoW.Flexi Link Status Register	1b'0	0x4C	Table 10.22
BoW.Flexi Advanced System Interface Register	1b'0	0x50	Table 10.23
BoW.Flexi Test Register	1b'0	0x54	Table 10.24
BoW.Flexi Test PRBS Seed Register	1b'0	0x58	Table 10.25
BoW.Flexi Test Error Mask Register	1b'0	0x5C	Table 10.26
BoW.Flexi Test Clock Sampling Register	1b'0	0x60	Table 10.27
BoW.Flexi Test Error Log Register 0	1b'0	0x64	Table 10.28
BoW.Flexi Test Error Log Register 1	1b'0	0x68	Table 10.29
BoW.Flexi Extension Command Configuration Register	1b'0	0x6C	Table 10.30
BoW.Flexi AMBA Configuration Register	1b'0	0x70	Table 10.31

Register	Required	Address	Link
BoW.Flexi Memory Configuration Register	1b'0	0x74	Table 13.6
BoW.Flexi Memory Address Register	1b'0	0x78	Table 13.7
BoW.Flexi Memory Command Extension Register	1b'0	0x7C	Table 13.8
BoW.Flexi Message Register 0	1b'0	0x80	Table 10.33
BoW.Flexi Message Register 1	1b'0	0x84	Table 10.33
BoW.Flexi Message Register 2	1b'0	0x88	Table 10.33
BoW.Flexi Message Register 3	1b'0	0x8C	Table 10.33
BoW.Flexi Message Register 4	1b'0	0x90	Table 10.33
BoW.Flexi Message Register 5	1b'0	0x94	Table 10.33
BoW.Flexi Message Register 6	1b'0	0x98	Table 10.33
BoW.Flexi Message Register 7	1b'0	0x9C	Table 10.33
Reserved for BoW.Flexi Standard Extension Commands	1b'0	0xA0-0xDF	Reserved for the sixteen BoW.Flexi Standard extension commands (one register per EXTCMD value); filled as commands are defined
Reserved	1b'0	0xE0-0xFF	Reserved for future BoW.Flexi standard registers

Configuration register space contains required registers for basic operation of the interface, as well as optional registers for additional features. The required registers shall be implemented by all BoW.Flexi implementations, while the optional registers can be implemented at the discretion of the designer to enable additional features.

Register space is reserved for future BoW.Flexi standard registers, so new features can be added within this specification. Designers should ensure that any use of reserved space does not conflict with the required and optional registers defined in this specification.

The **Access** column in the register tables that follow uses these types:

Table 10.2. Register Access Type Legend

Access	Meaning
RW	Read/Write — the field can be read and written by firmware.
RO	Read Only — the field can be read; writes are ignored.

Access	Meaning
W1C	Write 1 to Clear — writing 1 to a bit clears it to 0; writing 0 leaves it unchanged. Used for sticky status and error bits.
(blank)	Reserved — the field is reserved for future use. Reserved fields shall be written as 0 and their read values ignored.

Table 10.3. BoW.Flexi Configuration and Status Register

Bits	Identifier	Access	Reset	Description
31	config_local_error	W1C	0x0	1 = Local Error: error detected in a received message (e.g. double bit ECC error)
30	config_remote_error	W1C	0x0	1= Error Cmd message packet received from Partner chiplet
29	config_virtual_wire	RW	0x0	Virtual Wire Transaction
28	config_read_response	RW	0x0	1 = Read Response
27:26	config_len	RW	0x0	Transaction Length (only 2 bits for config)
25	config_go	RW	0x0	Control bit for interaction between BoW.Flexi Config/Error.
24	config_write	RW	0x0	1 = Write 0 = Request
23:19	config_virtual_wire_length	RW	0x5	Virtual Wire length in bits
18:0	Addr	RW	0x0	Address in the Partner configuration address space used for the command. By default the configuration address space is 256 bytes (8 least significant address bits). Additional address bits can be enabled using the config_register_address_bits field in Table 10.16

This register is used to manage the interface configuration, error reporting, and virtual wires.

Table 10.4. BoW.Flexi Configuration Write Data Register

Bits	Identifier	Access	Reset	Description
31:0	config_write_data	RW	0x0	CSR Write Data

Table 10.5. BoW.Flexi Configuration Read Data Register

Bits	Identifier	Access	Reset	Description
31:0	config_read_data	R	0x0	Only valid while the Go bit is Low and the previous command was a Read

Table 10.6. BoW.Flexi Interface Configuration Register

Bits	Identifier	Access	Reset	Description
31:29	fixed_size	RW	0x2	Fixed size value for all system transactions (not used for configuration commands)
28	fixed_size_enable	RW	0	Fixed size enable; when enabled the SIZE field is completely omitted from system command packets. The receiver infers the data length from the fixed_size register value. Read Response (CMD=010) never includes a SIZE field regardless of this setting. Configuration commands (CMD=111) always use their own LEN field. The Extension Command (CMD=110) command is not governed by this field; it has independent size control in the Extension Command Configuration Register.
27	adv_tx_data_lane	RW	0	Use Extended Data Lane Register Table 10.17
26	x32_tx_data_lane	RW	0	x32 data lane for Tx (one-hot for # of data lanes)
25	x16_tx_data_lane	RW	0	x16 data lane for Tx (one-hot for # of data lanes)
24	x8_tx_data_lane	RW	0	x8 data lane for Tx (one-hot for # of data lanes)
23	x4_tx_data_lane	RW	0	x4 data lane for Tx (one-hot for # of data lanes)
22	x2_tx_data_lane	RW	0	x2 data lane for Tx (one-hot for # of data lanes)
21	adv_rx_data_lane	RW	0	Use Extended Data Lane Register Table 10.17
20	x32_rx_data_lane	RW	0	x32 data lane for Rx (one-hot for # of data lanes)
19	x16_rx_data_lane	RW	0	x16 data lane for Rx (one-hot for # of data lanes)
18	x8_rx_data_lane	RW	0	x8 data lane for Rx (one-hot for # of data lanes)
17	x4_rx_data_lane	RW	0	x4 data lane for Rx (one-hot for # of data lanes)
16	x2_rx_data_lane	RW	0	x2 data lane for Rx (one-hot for # of data lanes)
15	is_manager	RW	0	1 = This chiplet configures the interface + 0 = This chiplet does not configure the interface
14:12	tx_data_size_max	RW	2	Maximum SIZE value supported for tx 0 = 1 Byte 1 = 2 Bytes 2 = 4 Bytes 3 = 8 Bytes 4 = 16 Bytes 5 = 32 Bytes 6 = 64 Bytes 7 = 128 Bytes
11:9	rx_data_size_max	RW	2	Maximum SIZE value supported for rx 0 = 1 Byte

Bits	Identifier	Access	Reset	Description
				1 = 2 Bytes 2 = 4 Bytes 3 = 8 Bytes 4 = 16 Bytes 5 = 32 Bytes 6 = 64 Bytes 7 = 128 Bytes
8	<code>ecc_enabled</code>	RW	0	0 = No ECC 1 = ECC Enabled
7	<code>virtual_wires_enabled</code>	RW	0	Number of virtual Wires supported: 0: Disabled 1: 32 bits
6:1	<code>address_width</code>	RW	0x1F	# of address bits = 1 + register_value, max # of address bits = 64. Default = 0x1F (32 bits)
0	<code>interface_configured</code>	RW	0	This bit should be the last write as part of the configuration setup. Once set the new configuration is used.

The default configuration of the interface is x1 mode. To increase the number of data lanes there are two mechanisms.

1. Enable one of the following `x2_tx_data_lane`, `x4_tx_data_lane`, `x8_tx_data_lane`, `x16_tx_data_lane` or `x32_tx_data_lane`
2. Set the `adv_tx_data_lane` bit to 1, and then set the number of lanes in the `ext_reg_tx_num_data_lanes` field in the [Extended Data Lane Register](#)

Note: If the `adv_tx_data_lane` bit is set to 1, the number of lanes will be determined solely by the value in the `ext_reg_tx_num_data_lanes` field, and the individual lane enable bits will be ignored

Table 10.7. BoW.Flexi Remote Error Register

Bits	Identifier	Access	Reset	Description
31:12	Implementation Dependent			Upper bits reserved for protocol-specific extensions (e.g. AHB Table A.11)
11:9	<code>remote_error_size</code>	R	0x0	SIZE field [2:0] of the most recent error-causing system command (or LEN[1:0] zero-extended for configuration commands). Latched on first error after clear
8:6	<code>remote_error_cmd</code>	R	0x0	Command code [2:0] of the most recent transaction that caused an error. Latched on first error after clear
5	<code>remote_overflow_error</code>	W1C	0	1 = Receive buffer overflow detected. Write 1 to clear
4	<code>remote_ecc_error</code>	W1C	0	1 = Uncorrectable (double-bit) ECC error detected (if ECC is enabled). Write 1 to clear
3	<code>remote_timeout_error</code>	W1C	0	1 = Response timeout occurred (timeout period set by <code>response_time_out</code> in Table 10.15). Write 1 to clear

Bits	Identifier	Access	Reset	Description
2	remote_size_error	W1C	0	1 = SIZE mismatch or invalid data size detected. Write 1 to clear
1	remote_cmd_error	W1C	0	1 = Invalid or unsupported command code received. Write 1 to clear
0	remote_frame_error	W1C	0	1 = frame signal error (unexpected assertion or de-assertion). Write 1 to clear

The remote error register has bits reserved for implementation dependent error reporting. This can be used for protocol specific errors. The output of these registers is provided to the protocol specific extension.

Table 10.8. BoW.Flexi Local Error Register

Bits	Identifier	Access	Reset	Description
31:12	Implementation Dependent			Upper bits reserved for protocol-specific extensions (e.g. AHB Table A.11)
11:9	local_error_size	R	0x0	SIZE field [2:0] of the most recent error-causing system command (or LEN[1:0] zero-extended for configuration commands). Latched on first error after clear
8:6	local_error_cmd	R	0x0	Command code [2:0] of the most recent transaction that caused an error. Latched on first error after clear
5	local_overflow_error	W1C	0	1 = Receive buffer overflow detected. Write 1 to clear
4	local_ecc_error	W1C	0	1 = Uncorrectable (double-bit) ECC error detected (if ECC is enabled). Write 1 to clear
3	local_timeout_error	W1C	0	1 = Response timeout occurred (timeout period set by response_time_out in Table 10.15). Write 1 to clear
2	local_size_error	W1C	0	1 = SIZE mismatch or invalid data size detected. Write 1 to clear
1	local_cmd_error	W1C	0	1 = Invalid or unsupported command code received. Write 1 to clear
0	frame_error	W1C	0	1 = frame signal error (unexpected assertion or de-assertion). Write 1 to clear

The local error register has bits reserved for implementation dependent error reporting. This can be used for protocol specific errors. The output of these registers is provided to the protocol specific extension.

Table 10.9. BoW.Flexi Discovery Register

Bits	Identifier	Access	Reset	Description
31:29	disc_fixed_size	R	CFG	Fixed size value for all system transactions (not used for configuration commands)
28	disc_use_fixed_size	R	CFG	Use fixed size ; when enabled no size info is transmitted for

Bits	Identifier	Access	Reset	Description
				system commands (configuration commands always use LEN)
27	adv_tx_data_lane_config	R	CFG	Use Extended Data Lane Register Table 10.17
26	x32_tx_data_lane_support	R	CFG	Support for x32 data lane for Tx
25	x16_tx_data_lane_support	R	CFG	Support for x16 data lane for Tx
24	x8_tx_data_lane_support	R	CFG	Support for x8 data lane for Tx
23	x4_tx_data_lane_support	R	CFG	Support for x4 data lane for Tx
22	x2_tx_data_lane_support	R	CFG	Support for x2 data lane for Tx
21	adv_rx_data_lane_config	R	CFG	Use Extended Data Lane Register Table 10.17
20	x32_rx_data_lane_support	R	CFG	Support for x32 data lane for Rx
19	x16_rx_data_lane_support	R	CFG	Support for x16 data lane for Rx
18	x8_rx_data_lane_support	R	CFG	Support for x8 data lane for Rx
17	x4_rx_data_lane_support	R	CFG	Support for x4 data lane for Rx
16	x2_rx_data_lane_support	R	CFG	Support for x2 data lane for Rx
15	disc_is_manager	R	CFG	Chiplet is capable of configuring the interface (both sides) 0 = This chiplet cannot configure the interface 1 = This chiplet can configure the interface
14:12	disc_tx_data_size_max	R	CFG	Maximum size value supported for tx 0 = 1 Byte 1 = 2 Bytes 2 = 4 Bytes 3 = 8 Bytes 4 = 16 Bytes 5 = 32 Bytes 6 = 64 Bytes 7 = 128 Bytes
11:9	disc_rx_data_size_max	R	CFG	Maximum size value supported for rx 0 = 1 Byte 1 = 2 Bytes 2 = 4 Bytes 3 = 8 Bytes 4 = 16 Bytes 5 = 32 Bytes 6 = 64 Bytes 7 = 128 Bytes
8	disc_ecc_supported	R	CFG	ECC Supported

Bits	Identifier	Access	Reset	Description
7	disc_virtual_wires_supported	R	CFG	Virtual Wires Supported
6:1	disc_address_width	R	CFG	# of addresses used, default 0x1F (32 bit address) Address value = 1 + register_value
0	single_configuration	R	CFG	0 = Supports Multiple Configuration Settings 1 = The device can only be configured with the discovery settings

The reset value in the discovery register is design time configurable. This is denoted by the CFG field.

Table 10.10. BoW.Flexi Vendor Identity Register

Bits	Identifier	Access	Reset	Description
31:24	reserved			Reserved
23:0	vendor_id	R	CFG	Vendor ID

The vendor format follows the IEEE Organizationally unique identifier (OUI) registry, which is a 24-bit number that uniquely identifies a vendor, manufacturer, or organization. The OUI is assigned by the IEEE Registration Authority (<https://standards.ieee.org/products-programs/regauth/>) and is used in various networking and communication protocols to identify the source of a device or component. There is a fee for this. Company IDs can be found on the IEEE website: <https://standards-oui.ieee.org/>

Bit 23 represents the most significant bit of the OUI, while bit 0 represents the least significant bit. For example, if a company has been assigned the OUI 00-1A-2B, then the vendor_id field would be set to 0x001A2B in the vendor identity register. This allows for unique identification of the vendor associated with a particular BoW.Flexi implementation.

Table 10.11. BoW.Flexi Device Identity Register

Bits	Identifier	Access	Reset	Description
31:16	reserved			Reserved
15:0	device_id	R	CFG	Device ID

Table 10.12. BoW.Flexi Specification Version Register

Bits	Identifier	Access	Reset	Description
31:13	Reserved			Reserved
12:0	spec_version	R	CFG	BoW.Flexi specification version supported: [12:8] Major Version; [7:3] Minor Version; [2:0] Patch Version

Table 10.13. BoW.Flexi Virtual Wire Out Register

Bits	Identifier	Access	Reset	Description
31:0	virtual_wire_out	R	0x0	Virtual Wire output register

This register stores the values received from Virtual Wire transactions.

Table 10.14. BoW.Flexi Virtual Wire In Mask Register

Bits	Identifier	Access	Reset	Description
31:0	virtual_wire_in_mask	RW	0x0	Virtual Wire mask register

This register is used to mask bits on the Virtual Wire input. This can be used to block Virtual Wire inputs from initiating a BoW.Flexi transfer.

Table 10.15. BoW.Flexi Interface Configuration Extension Register 1

Bits	Identifier	Access	Reset	Description
31	jtag_over_flexi_en	RW	0	Optional JTAG-over-BoW.Flexi transport enable. When set to 1 the Physical Layer switches the die-to-die pins to the 4-signal JTAG pathway; the serializer and higher layers are bypassed and no data is delivered to the Link or Transaction Layers. On a subordinate (is_jtag_manager=0) setting this bit shall also reset the on-die TAP to Test-Logic-Reset, so the subordinate joins the scan chain in a known state. Default 0 = normal BoW.Flexi operation. Reverts to 0 on link reset. See the JTAG (Boundary Scan) Transport section of the Physical Layer chapter.
30	is_jtag_manager	RW	0	Selects this end's role when jtag_over_flexi_en=1: 0 = JTAG subordinate (default), 1 = JTAG manager. The subordinate keeps its link-reset detector enabled (so the manager can reset it out of JTAG mode) but shall not issue link-reset requests; the manager disables its own link-reset detector but retains the ability to send link-reset requests, providing the exit path. Reverts to 0 on link reset.
29	ext_cmd_class	RW	0	Class of the Extension Command (CMD=110): 0 = System Interface command (default), handled identically to other system commands and carried on the tx_sys_pkt_* source; 1 = Advanced System Interface command, routed to the dedicated advanced ports (tx/rx_adv_sys_pkt_*). Shall be programmed identically on both ends. Before setting this bit to 1 the manager shall read the peer adv_sys_if_present in the Advanced System Interface Register and shall leave it 0 if the peer reports 0. Reverts to 0 on link reset.
28	write_data_mask_enabled	RW	0	When enabled the data mask is transmitted as part of the write command and can be used to indicate which bytes of the write data are valid. When disabled all bytes are assumed to be valid. For AHB mode both this bit and wstrb_in_data in the

Bits	Identifier	Access	Reset	Description
				AMBA Configuration Register must be set to enable write byte enables.
27:22	Reserved	RO	0	Reserved.
21:8	response_time_out	RW	0	Response timeout counter (HCLK cycles). When set to 0 this mechanism is disabled. When non-zero the counter starts after the full request packet has been transmitted. If no response packet is received before the counter expires: (1) HRESP=ERROR is driven on the local AHB bus; (2) an Error Packet (CMD=111) is sent to the remote chiplet to inform it that the request has timed out.
7	clock_sharing_enabled	RW	0	BoW.Flexi free running Tx clock
6:3	protocol_mode	RW	0	0 = BoW.Flexi (Serial Mode) 1 = AHB (optional fields disabled by default; feature set controlled by AMBA Configuration Register) 2 = Reserved 3 = BoW Flexi for Memory-DRAM 4 = BoW Flexi for Memory-NVM 5 = AXI (optional fields disabled by default; feature set controlled by AMBA Configuration Register) 6 = APB (optional fields disabled by default; feature set controlled by AMBA Configuration Register) 7+ = Reserved
2:0	data_mode	RW	0	0 = BoW.Flexi (Serial Mode) 1 = BoW Flexi for Memory Uni-Directional 2 = BoW Flexi for Memory Bi-Directional 3+ = Reserved

Bits 31:30 select the optional JTAG-over-BoW.Flexi transport: `jtag_over_flexi_en` (bit 31) switches the Physical Layer to the 4-signal JTAG pathway and `is_jtag_manager` (bit 30) selects the manager or subordinate role. See [Section 8.2](#) for the full behavior.

Table 10.16. BoW.Flexi Interface Configuration Extension Register 2

Bits	Identifier	Access	Reset	Description
31:8	Reserved			Reserved
7:4	config_posted_write_latency	RW	1	Number of system clock cycles between end of a configuration write command, and issuing next command
3:0	config_register_address_bits	RW	0	#bits = 8 + register value; max value 0xB (total 19 bits)

The configuration address space can be increased by adding additional address bits using the field **config_register_address_bits**. This value is added to the default configuration address i.e. 8-bits, allowing a larger address space for configuration commands. For example, if **config_register_address_bits** is set to 4, the configuration address space would be 12-bits, allowing for 4096 possible configuration addresses instead of 256.

Table 10.17. BoW.Flexi Extension Data Lane Register

Bits	Identifier	Access	Reset	Description
31:26	Reserved			Reserved
25:23	Reserved			Reserved for data lane expansion from 128 to 1024 lanes
22:16	ext_reg_tx_num_data_lanes	RW	0x00	Register value + 1 = Number of tx data lanes
15:10	Reserved			Reserved
9:7	Reserved			Reserved for data lane expansion from 128 to 1024 lanes
6:0	ext_reg_rx_num_data_lanes	RW	0x00	Register value + 1 = Number of rx data lanes

Table 10.18. BoW.Flexi Optional Field Configuration Register

Bits	Identifier	Access	Reset	Description
31	Reserved			Reserved
30:23	opt_field1_size	RW	0	number of bits for optional field 1
22:16	opt_field1_cmd_support	RW	0	opt_field1 command support Each bit enables the optional field for the corresponding command: bit[0]: Non-Posted Write (000); bit[1]: Read (001); bit[2]: Read Response (010); bit[3]: Write Response (011); bit[4]: Posted Write (100); bit[5]: Virtual Wire (101); bit[6]: Extension (110); Configuration Command (111) does not support optional fields
15	Reserved			Reserved
14:7	opt_field0_size	RW	0	number of bits for optional field 0
6:0	opt_field0_cmd_support	RW	0	opt_field0 command support Each bit enables the optional field for the corresponding command: bit[0]: Non-Posted Write (000); bit[1]: Read (001); bit[2]: Read Response (010); bit[3]: Write Response (011); bit[4]: Posted Write (100); bit[5]: Virtual Wire (101); bit[6]: Extension (110); Configuration Command (111) does not support optional fields

This register can configure which commands will utilize the optional fields. The optional fields are not supported for the Configuration Command [CMD:3]=111.

Table 10.19. BoW.Flexi Optional Field Extended Configuration Register

Bits	Identifier	Access	Reset	Description
31	Reserved			Reserved
30:23	opt_field3_size	RW	0	number of bits for optional field 3
22:16	opt_field3_cmd_support	RW	0	opt_field3 command support Each bit enables the optional field for the corresponding command: bit[0]: Non-Posted Write (000); bit[1]: Read (001); bit[2]: Read Response (010); bit[3]: Write Response (011); bit[4]: Posted Write (100); bit[5]: Virtual Wire (101); bit[6]: Extension (110); Configuration Command (111) does not support optional fields
15	Reserved			Reserved
14:7	opt_field2_size	RW	0	number of bits for optional field 2
6:0	opt_field2_cmd_support	RW	0	opt_field2 command support Each bit enables the optional field for the corresponding command: bit[0]: Non-Posted Write (000); bit[1]: Read (001); bit[2]: Read Response (010); bit[3]: Write Response (011); bit[4]: Posted Write (100); bit[5]: Virtual Wire (101); bit[6]: Extension (110); Configuration Command (111) does not support optional fields

This register serves as an extension to the [Optional Field Configuration Register](#) to allow for configuration of additional optional fields. As with that register, the optional fields configured here are not supported for the Configuration Command [CMD:3]=111.

The optional-field configuration is split across two register groups:

- The Optional Field Configuration Register, Optional Field Extended Configuration Register and Optional Field Trailer Configuration Register are protocol-agnostic. They define the **transport**: the fixed size of each field (`opt_fieldN_size`) and the per-command presence mask (`opt_fieldN_cmd_support`).
- The protocol-specific content is defined by the single AMBA Configuration Register: which protocol attributes ride in each field and their individual sub-widths.

The two groups shall be kept consistent: `opt_fieldN_size` shall equal the sum of the widths of the sub-fields enabled by the AMBA Configuration Register. For example, in AHB mode the bridge sets `opt_field0_size` = 4 + `user_width`. The placement of sub-fields within each OPT_FIELD follows the BoW.Flexi optional-field packing convention (see [Section 5.1.1.2](#)).

The sub-field enables follow two conventions:

- **Fixed-width sub-fields** use a single-bit enable in the AMBA Configuration Register (for example `prot_en`, `burst_en`, `lock_en`).
- **Variable-width sub-fields** (such as the USER defined field) are controlled by a single width field that serves as both the enable and the size: a value of 0 disables the sub-field, and a value of N enables it as an N-bit field (for example `user_width`). There is no separate enable bit.



For the link to interoperate, both sides of the link shall be programmed with identical `opt_fieldN_size` and `opt_fieldN_cmd_support` values, together with matching protocol configuration registers. This guarantees that every transmitted bit is reconstructed at the same bit position by the receiver.

Table 10.20. BoW.Flexi Optional Field Trailer Configuration Register

Bits	Identifier	Access	Reset	Description
31:15	Reserved			Reserved
14:7	<code>opt_field_t_size</code>	RW	0	number of bits for the trailing optional field (OPT_FIELD_T)
6:0	<code>opt_field_t_cmd_support</code>	RW	0	opt_field_t command support Each bit enables the trailing optional field for the corresponding command: bit[0]: Non-Posted Write (000); bit[1]: Read (001); bit[2]: Read Response (010); bit[3]: Write Response (011); bit[4]: Posted Write (100); bit[5]: Virtual Wire (101); bit[6]: Extension (110); Configuration Command (111) does not support optional fields

This register configures the single **trailing** optional field, OPT_FIELD_T, which is appended after the DATA field rather than with the leading optional fields. Like the other optional-field registers it is protocol-agnostic (transport size and per-command mask only), the trailer is disabled by default (`opt_field_t_size` = 0, `opt_field_t_cmd_support` = 0), and it reverts to its reset value on link reset. Both sides of the link shall be programmed with identical `opt_field_t_size` and `opt_field_t_cmd_support`. See [Section 5.1.1.2](#) for its position and packing.

Table 10.21. BoW.Flexi Power State Register

Bits	Identifier	Access	Reset	Description
31:0	Reserved			Reserved

Table 10.22. BoW.Flexi Link Status Register

Bits	Identifier	Access	Reset	Description
31:0	Reserved			Reserved

The Advanced System Interface Register (offset 0x50) advertises whether the optional Advanced System Interface is implemented and, if so, its maximum payload size (ADV_MAXBITS). When `adv_sys_if_present` is 0

the Extension Command (CMD=110) is always an ordinary System Interface command and `ext_cmd_class` in the Interface Configuration Extension Register 1 shall remain 0. A manager shall read the peer's `adv_sys_if_present` before setting `ext_cmd_class` to 1.

Table 10.23. BoW.Flexi Advanced System Interface Register

Bits	Identifier	Access	Reset	Description
31:13	Reserved	RO	0	Reserved
12:1	<code>adv_sys_if_max_size</code>	RO	128	Maximum Advanced System Interface payload size in bits (ADV_MAXBITS). A free integer fixed at design time; default 128. Defines the width of the <code>tx_adv_sys_pkt_data</code> / <code>rx_adv_sys_pkt_data</code> buses and shall fit within the link MAXBITS budget. Field width matches the link-layer <code>arb_pkt_bit_count</code> (12 bits).
0	<code>adv_sys_if_present</code>	RO	0	1 = the optional Advanced System Interface ports (<code>tx/rx_adv_sys_pkt_*</code>) are implemented at design time. When 0 the class is not available and <code>ext_cmd_class</code> in the Interface Configuration Extension Register 1 shall remain 0.

Table 10.24. BoW.Flexi Test Register

Bits	Identifier	Access	Reset	Description
31:24	<code>test_mode_count</code>	RW	0	Test duration in interface clock cycles = $2^{\text{test_mode_count}}$
23	<code>test_done</code>	R	0	Set to 1 when the test has completed the programmed number of cycles
22	<code>test_pass</code>	R	0	Set to 1 when the test has completed with zero bit errors
21:6	<code>test_error_count</code>	R	0	Number of bit errors detected by the RX checker during the test. Saturates at maximum value
5	<code>sample_data_at_rx</code>	RW	0	0 = flow through test; Data is sampled at the BERT (default) 1 = Data is sampled using the falling edge of clock at each receiver before getting forwarded to the next stage
4	<code>test_prbs_sel</code>	RW	0	Selects the PRBS polynomial: 0 = PRBS-31 ($X^{31} + X^{28} + 1$) 1+ = Reserved
3:1	<code>test_mode_sel</code>	RW	0	Selects the test mode: 0 = No test mode selected 1 = single slice loopback test 2 = slice to slice loopback test 3 = Chiplet to Chiplet long loopback test 4 = Chiplet to Chiplet Tx open loop test 5 = Chiplet to Chiplet Rx open loop test 6-7 = Reserved
0	<code>test_enable</code>	RW	0	Enables test mode. When set the PRBS generator/checker is active and clocked by the interface clock

Table 10.25. BoW.Flexi Test PRBS Seed Register

Bits	Identifier	Access	Reset	Description
31	Reserved			Reserved
30:0	test_prbs_seed	RW	0x1	Initial LFSR state (seed) for the PRBS generator. Must be non-zero. The checker uses the same seed for expected pattern computation. For PRBS-9 only bits [8:0] are used

Table 10.26. BoW.Flexi Test Error Mask Register

Bits	Identifier	Access	Reset	Description
31:25	Reserved			Reserved
24	error_mask_1_enable	RW	0	1 = Mask IO specified by error_mask_1_io_index. Errors on this IO are ignored by the checker and error log
23:16	error_mask_1_io_index	RW	0	Index of the IO lane (0 to 128) to mask, Index 0 is the frame plus 128 data lanes
15:9	Reserved			Reserved
8	error_mask_0_enable	RW	0	1 = Mask IO specified by error_mask_0_io_index. Errors on this IO are ignored by the checker and error log
7:0	error_mask_0_io_index	RW	0	Index of the IO lane (0 to 128) to mask, Index 0 is the frame plus 128 data lanes

Table 10.27. BoW.Flexi Test Clock Sampling Register

Bits	Identifier	Access	Reset	Description
31:8	Reserved			Reserved
7:0	test_clk_sample_adj	RW	0	Clock sampling adjustment for BERT. Implementation dependent. Used to adjust the sampling point of the received clock relative to the data at the BERT checker

Table 10.28. BoW.Flexi Test Error Log Register 0

Bits	Identifier	Access	Reset	Description
31	error_log_valid	R	0	1 = This log entry contains a valid error record
30:23	error_log_io_index	R	0	Index of the IO lane (0 to 128) that reported the first error for this entry, Index 0 is the frame plus 128 data lanes
22:0	error_log_cycle_count	R	0	Interface clock cycle count at which the first error was detected on this IO. Saturates at maximum value

Table 10.29. BoW.Flexi Test Error Log Register 1

Bits	Identifier	Access	Reset	Description
31	error_log_valid	R	0	1 = This log entry contains a valid error record
30:23	error_log_io_index	R	0	Index of the IO lane (0 to 128) that reported the first error for this entry, Index 0 is the frame plus 128 data lanes
22:0	error_log_cycle_count	R	0	Interface clock cycle count at which the first error was detected on this IO. Saturates at maximum value

Table 10.30. BoW.Flexi Extension Command Configuration Register (offset 0x6C)

Bits	Identifier	Access	Reset	Description
31:25	Reserved			Reserved
24	ext_cmd_fixed_size_en	RW	0	When 1 the SIZE field is omitted from Extension Command (CMD=110) packets and the receiver infers the data length from ext_cmd_fixed_size. Independent of the global fixed_size in the Interface Configuration Register, which does not apply to extension commands.
23:21	ext_cmd_fixed_size	RW	0x2	Fixed size value assumed for Extension Command packets when ext_cmd_fixed_size_en is set
20	fixed_ext_cmd_en	RW	0	When 1 the 4-bit EXTCMD field is omitted from Extension Command (CMD=110) packets and the receiver assumes fixed_ext_cmd for every extension command. Both sides of the link shall be configured identically; ensuring this consistency is a system responsibility.
19:16	fixed_ext_cmd	RW	0	EXTCMD value assumed for every Extension Command packet when fixed_ext_cmd_en is set
15:0	ext_cmd_supported	RO	CFG	One bit per BoW.Flexi Standard extension command: bit N = 1 indicates EXTCMD = N is supported. All sixteen values are BoW.Flexi Standard (reserved, none allocated in this revision). A received extension command whose support bit is 0 is not executed and is reported via the local_cmd_error bit of the Local Error Register. Read-only; the supported set (and hence the specification revision covered) is fixed at design time.

This register configures the Extension Command (CMD=110) command extension for the System Interface class. See [Section 5.1.1.3](#) for the field usage, the 16-command namespace, the fixed-field controls, and the unsupported-command error behavior.

Table 10.31. BoW.Flexi AMBA Configuration Register (offset 0x70)

Bits	Identifier	Access	Reset	Description
31:28	user_width	RW	4'b0000	USER defined field width in bits, appended to the protocol's OPT_FIELD (0 = USER not transmitted; valid range 0-15). AHB/APB append to OPT_FIELD0; AXI appends to OPT_FIELD2.
27:24	id_width	RW	4'b0000	Transaction ID width in bits. AXI only (default 0, valid range 1-8); determines OPT_FIELD0 sizing for AXI. Reserved (reads 0) in AHB/APB modes.
23:20	Reserved			Reserved
19:16	data_width	RW	4'b0000	Data bus width. The field position is common to all modes but the value encoding is selected by protocol_mode: AHB: 0=32, 1=64, 2=128, 3=256 bit APB: 0=8, 1=16, 2=32 bit AXI: 0=32, 1=64, 2=128, 3=256, 4=512 bit
15:8	Reserved			Reserved
7	qos_en	RW	0	Transmit QOS in OPT_FIELD2. AXI only ; Reserved in AHB/APB.
6	cache_en	RW	0	Transmit CACHE in OPT_FIELD2. AXI only ; Reserved in AHB/APB.
5	lock_en	RW	0	Transmit lock attribute (AHB HMASTLOCK in OPT_FIELD2 / AXI AxLOCK in OPT_FIELD2). Reserved in APB.
4	len_en	RW	0	Transmit burst length (AXI AXLEN in OPT_FIELD1). AXI only ; AHB derives length from burst_en, Reserved in APB.
3	burst_en	RW	0	Transmit burst type (AHB HBURST / AXI BURST in OPT_FIELD). Reserved in APB (single transfers only).
2	wstrb_in_data	RW	0	Enable per-byte write strobes embedded in the DATA field (AHB HWSTRB / APB PSTRB / AXI WSTRB), 1 bit per data byte. For AHB this bit and write_data_mask_enabled in the Interface Configuration Extension Register 1 must both be set.
1	prot_en	RW	0	Transmit protection attributes (AHB HPROT / APB PPROT / AXI AxPROT) in the protocol's OPT_FIELD.
0	bridge_en	RW	0	Enable the protocol bridge selected by protocol_mode. 0 = bridge disabled, 1 = bridge enabled.

This single register configures the active Arm® AMBA® protocol bridge (AHB, APB, or AXI). A device operates in only one bridge mode at a time, selected by protocol_mode in [Interface Configuration Extension](#)

Register 1, so the three protocols share a common field overlay. Fields that do not apply to the selected mode are Reserved (read as 0) in that mode. The applicability of each field per mode is summarized below.

Table 10.32. AMBA field usage

Field	AHB	APB	AXI	Notes
user_width	Y	Y	Y	USER field width appended to the protocol OPT_FIELD
id_width	-	-	Y	AXI transaction ID width (default 0)
data_width	Y	Y	Y	Bus-width encoding differs per mode (see field description)
qos_en	-	-	Y	AXI QOS attribute
cache_en	-	-	Y	AXI CACHE attribute
lock_en	Y	-	Y	AHB HMASTLOCK / AXI AxLOCK
len_en	-	-	Y	AXI AXLEN burst length
burst_en	Y	-	Y	AHB HBURST / AXI BURST
wstrb_in_data	Y	Y	Y	Per-byte write strobes embedded in DATA
prot_en	Y	Y	Y	AHB HPROT / APB PPROT / AXI AxPROT
bridge_en	Y	Y	Y	Enable the selected bridge

The eight Message Registers (Message Register 0 through Message Register 7, offsets 0x80–0x9C) provide a small block of general-purpose storage for passing implementation-defined messages between chiplets. All eight share the layout below; the interpretation of the message content is implementation-defined. When a write occurs to a message register it raises a local flag to indicate that it has been updated. The management of these flags is implementation-defined.

Table 10.33. BoW.Flexi Message Registers (0x80–0x9C)

Bits	Identifier	Access	Reset	Description
31:0	msg_data	RW	0	General-purpose message payload word exchanged between chiplets. The interpretation of the message content is implementation-defined.

Chapter 11. BoW.Flexi Command Examples

11.1. Non-Posted Write Command

There are two Write Commands: Posted and Non-Posted.

Padding: Fill with zeros at the end of the packet to align to the data path lane boundary (for a x1 there is no need for padding)

11.1.1. Non-Posted Write

Here we provide examples of Non-Posted Write commands.

BoW.Flexi Example 8b Non-Posted Write to 16b address

```
[CMD:3][SIZE:3][ADDR:16][DATA:8][PAD:variable]
```

Non-Posted Write Request:

Bits [2:0]:	CMD (000)
Bits [5:3]:	SIZE (000)
Bits [21:6]:	ADDR (16 bits)
Bits [29:22]:	DATA (8 bits)
Bits [end]:	PAD (padding to align to data lane boundary)

BoW.Flexi Example 64b Non-Posted Write to 32b address with ECC

```
[CMD:3][ECC:8][SIZE:3][ADDR:32][DATA:64][PAD:variable]
```

Non-Posted Write Request:

Bits [2:0]:	CMD (000)
Bits [10:3]:	ECC
Bits [13:11]:	SIZE (011) 8 Byte
Bits [45:14]:	ADDR (32 bits)
Bits [109:46]:	DATA (64 bits)
Bits [end]:	PAD (padding to align to data lane boundary)

BoW.Flexi Example 128b write to 16b address

```
[CMD:3][SIZE:3][ADDR:16][DATA:128][PAD:variable]
```

Non-Posted Write Request:

Bits [2:0]:	CMD (000)
Bits [5:3]:	SIZE (100) 16 Byte
Bits [21:6]:	ADDR (16 bits)
Bits [149:22]:	DATA (128 bits)
Bits [end]:	PAD (padding to align to data lane boundary)

11.1.1.1. ECC Coverage for Large Packets

All ECC calculations are done over a 128-bit codeword: 120 payload bits plus the 8-bit ECC syndrome. The first codeword also carries the 3-bit CMD, leaving 117 payload bits.

When packet payload exceeds 117 bits (excluding the 3-bit CMD and 8-bit ECC fields):

1. First ECC (bits 10:3): Protects CMD [2:0], ECC_a[10:3] and up to 117 bits of payload [127:11]
2. Second ECC (bits 138:131): Protects bits [130:128], ECC_b[138:131] and the remaining payload bits up to bit position [255].
3. This can continue for larger payloads with the ECC added in the same way as #2 above.

See [Table 6.1](#) through [Table 6.4](#) for complete encoding matrices.

BoW.Flexi Example 128b write to 16b address with ECC

```
[CMD:3][ECC:8][SIZE:3][ADDR:16][DATA:128][PAD:variable]

Non-Posted Write Request:
Bits [2:0]:      CMD (000) (Covered by ECC_a)
Bits [10:3]:     ECC_a - 1st 128b (Covered by ECC_a)
Bits [13:11]:    SIZE (100) 16 Byte (Covered by ECC_a)
Bits [29:14]:    ADDR (16 bits) (Covered by ECC_a)
Bits [127:30]:   DATA (98 bits) (Covered by ECC_a)
Bits [130:128]:  DATA (3 bits), (Covered by ECC_b)
Bits [138:131]:  ECC_b - 2nd 128b (Covered by ECC_b)
Bits [165:139]:  DATA (27 bits) (Covered by ECC_b)
Bits [end]:      PAD (padding to align to data lane boundary)
```

See [Figure 7.3](#) for the data sequence in a x16 data lanes configuration.

In the configuration registers of the BoW.Flexi interface, the address width, data lane as well as ECC support can be set. [BoW.Flexi Example 128b write to 16b address](#) shows what the data stream would look like for a 128-bit write to a 16-bit address. [BoW.Flexi Example 128b write to 16b address with ECC](#) covers the case where ECC is present. The ECC (8 bits) can cover the Payload up to 117 bits and the 3-bit Command. The ECC is placed after the Command to keep location of the ECC consistent for all the different commands. In BoW.Flexi the length of the transmission varies as a function of the command and the size (**size**). Zeros will be padded for all additional bits transmitted across the interface when xN IOs are used while the Frame is active (last beat only).

When the receiver detects a frame signal assertion, it resets its data registers to zero. The transmitted data is then loaded into the Rx registers sequentially with each cycle transferred across the interface. When the frame is de-asserted the content of the Rx registers can be evaluated and error correction occurs if necessary. Once the ECC evaluation has been completed, the Command can be interpreted and the request can be initiated.

11.1.1.2. Non-Posted Write with Write Data Mask

When the write data mask is enabled (field **write_data_mask_enabled** in [Interface Configuration Extension Register 1](#)), a mask bit is prepended to each byte of the DATA field, as described in [Section 5.1.4](#). A mask bit of 1 indicates the byte is valid and shall be written; a mask bit of 0 indicates the byte shall not be modified at the target. The **SIZE** field still reports the data size excluding the mask bits.

In the example below, a 4-byte (32-bit) Non-Posted Write targets a 16-bit address with the write data mask enabled. Bytes 0 and 2 are written (mask=1) while bytes 1 and 3 are left unchanged (mask=0). Each of the 4 bytes is preceded by its mask bit, so the DATA field is $4 * (1 + 8) = 36$ bits.

BoW.Flexi Example 32b Non-Posted Write to 16b address with write data mask

```
[CMD:3][SIZE:3][ADDR:16][MASK0:1][DATA0:8][MASK1:1][DATA1:8][MASK2:1][DATA2:8][MASK3:1][DATA3:8][PAD:variable]
```

Non-Posted Write Request (write data mask enabled):

```
Bits [2:0]:      CMD (000)
Bits [5:3]:      SIZE (010) 4 Byte
Bits [21:6]:     ADDR (16 bits)
Bits [22]:       MASK0 (1) Byte 0 written
Bits [30:23]:    DATA0 (Byte 0)
Bits [31]:       MASK1 (0) Byte 1 not written
Bits [39:32]:    DATA1 (Byte 1, ignored at target)
Bits [40]:       MASK2 (1) Byte 2 written
Bits [48:41]:    DATA2 (Byte 2)
Bits [49]:       MASK3 (0) Byte 3 not written
Bits [57:50]:    DATA3 (Byte 3, ignored at target)
Bits [end]:      PAD (padding to align to data lane boundary)
```

The receiver uses each mask bit to determine whether the associated byte is committed to the target address. When ECC is enabled, the mask bits are protected together with the rest of the packet in the same way as all other transfers. A Write Response is returned on successful completion, as for any other Non-Posted Write.

11.2. Write Response

- A Write Response command is sent in response to a non-posted Write Command (on completion of the write) or a Posted Write (as a flow-control acknowledgment that the receiver is ready for a further posted write)
 - The **size** field is not sent, nor is the address.
 - The frame signal will truncate the transmission after the command.

BoW.Flexi Example write response to 16b address

```
[CMD:3][PAD:variable]
Bits [2:0]:      CMD (011)
Bits [end]:      PAD (padding to align to data lane boundary)
```

11.3. Posted Write

A posted write request is a write transaction whose Write Response is a flow-control acknowledgment: the receiver returns it once it has accepted the write and is ready for a further posted write, without waiting for the write to complete at its destination.

Here we provide examples of posted write commands.

BoW.Flexi Example 8b posted write to 16b address

```
[CMD:3][SIZE:3][ADDR:16][DATA:8][PAD:variable]

Posted Write Request:
Bits [2:0]:      CMD (100)
Bits [5:3]:      SIZE (000)
Bits [21:6]:     ADDR (16 bits)
Bits [29:22]:    DATA (8 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

BoW.Flexi Example 64b Posted Write to 32b address with ECC

```
[CMD:3][ECC:8][SIZE:3][ADDR:32][DATA:64][PAD:variable]
```

Posted Write Request:

```
Bits [2:0]:      CMD (100)
Bits [10:3]:     ECC
Bits [13:11]:    SIZE (011) 8 Byte
Bits [45:14]:    ADDR (32 bits)
Bits [109:46]:   DATA (64 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

11.4. Read Command

- A Read Command initiates a Read operation to the MMIO address space.

BoW.Flexi Example 8b read to 16b address

```
[CMD:3][SIZE:3][ADDR:16][PAD:variable]
```

```
Bits [2:0]:      CMD (001) Read
Bits [5:3]:      SIZE (000)
Bits [21:6]:     ADDR (16 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

11.5. Read Response

BoW.Flexi Example 16b Read-Response

```
[CMD:3][DATA:16][PAD:variable]
```

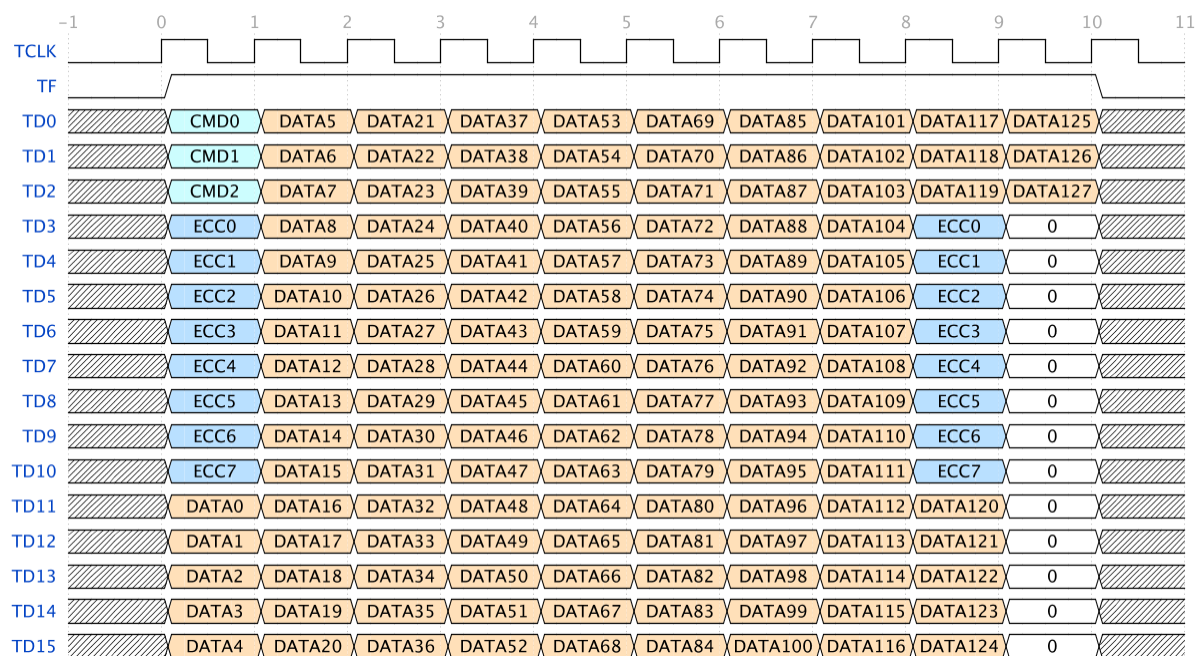
```
Bits [2:0]:      CMD (010) Read-Response
Bits [18:3]:     DATA (16 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

BoW.Flexi Example 128b Read-Response with ECC

```
[CMD:3][ECC:8][DATA:128][PAD:variable]
```

```
Bits [2:0]:      CMD (010) Read-Response (Covered by ECC_a)
Bits [10:3]:     ECC_a - 1st 128b (Covered by ECC_a)
Bits [127:11]:   DATA (covered by ECC_a)
Bits [130:128]:  DATA (covered by ECC_b)
Bits [138:131]:  ECC_b - 2nd 128b (Covered by ECC_b)
Bits [146:139]:  DATA (covered by ECC_b)
Bits [end]:      PAD (padding to align to data lane boundary)
```

See [Figure 11.1](#) for the data sequence in a x16 data lanes configuration.

Figure 11.1. BoW.Flexi Example 128b Read-Response over 16 data lanes with ECC

11.6. Example Use of optional fields

BoW.Flexi supports the addition of up to four optional fields. These fields can be used to communicate additional information. The size of these fields can be adjusted in the configuration register. The commands that utilize these fields can be set in the configuration register.

For the example below:

- Set `opt_field0_size` in [Optional Field Configuration Register](#) to `8b'00000011`
- Set `opt_field1_size` in [Optional Field Configuration Register](#) to `8b'00000100`
- Set `opt_field0_cmd_support` in [Optional Field Configuration Register](#) to `7b'0010110`
- Set `opt_field1_cmd_support` in [Optional Field Configuration Register](#) to `7b'0010110`
 - Enable `OPT_FIELD0/1` for Read, Read Response and Posted Write

BoW.Flexi Example 32b read to 32b address with `opt_field0` (3bits) and `opt_field1` (4bits)

Request packet:

```
[CMD:3][OPT_FIELD0:3][OPT_FIELD1:4][SIZE:3][ADDR:32][PAD:variable]
```

```
Bits [2:0]:      CMD (001) Read
Bits [5:3]:      OPT_FIELD0
Bits [9:6]:      OPT_FIELD1
Bits [12:10]:    SIZE (010)
Bits [44:13]:    ADDR (32 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

Response packet:

```
[CMD:3][OPT_FIELD0:3][OPT_FIELD1:4][DATA:32][PAD:variable]

Bits [2:0]:      CMD (010) Read Response
Bits [5:3]:      OPT_FIELD0
Bits [9:6]:      OPT_FIELD1
Bits [41:10]:    DATA (32 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

11.7. BoW.Flexi Virtual-Wires

Virtual-Wires provide a very low latency path for signaling between two chiplets. Each chiplet can support 32b Virtual Wire inputs and 32b Virtual Wire outputs. Chiplet_a system interface virtual-Wire inputs, shall be reflected on Chiplet_b system interface Virtual Wire outputs (and vice versa). The Virtual Wire packet contains a 3 bit command, and up to 32 bits of data. If a Virtual Wire input changes on Chiplet_a, e.g. bit 5, a Virtual Wire packet is assembled. The field **config_virtual_wire_length** and **config_virtual_wire** in the [Configuration and Status Register](#) is set.

The Virtual Wire packet only needs to send data up to bit 5 as the higher order bits have not changed see [BoW.Flexi Example 5b Virtual Wire](#). Writing the field **go** in the [Configuration and Status Register](#) initiates the Virtual Wire transfer.

Once the packet is received the Virtual Wire outputs on Chiplet_b will be updated.

Virtual-Wires are to be used for semi-static signals between chiplets. These can be used to convey system status.

A Virtual Wire mask register is available to block Virtual-Wires from getting transmitted.

When multiple data lanes are used between chiplets, the Virtual-Wires transmitted will need to fill all the data lanes, as it is not possible to pad the Virtual Wire data (for less than 32b). Virtual Wire bits contain meaningful data and therefore **cannot be zero-padded** within the 32-bit Virtual Wire field. For example, if the number of data lanes is 8 and Virtual Wire bit 2 changes, the 3-bit command and the first 5 Virtual Wire bits shall be sent to fill the first data beat (8 bits total). The receiver interprets all bits within the lane-aligned boundary as valid Virtual Wire state. If the full 32b Virtual Wire has been transmitted, then padding can be added as necessary to align to the data lane boundary.

Guideline: the most frequently changing virtual wires should be assigned to the least significant Virtual Wire bits so the more frequent changes have shorter packets.

BoW.Flexi Example 32b Virtual Wire

```
[CMD:3][Virtual_Wire:32][PAD:variable]

Bits [2:0]:      CMD (101) Virtual Wire Command
Bits [34:3]:     Virtual Wire
Bits [end]:      PAD (padding to align to data lane boundary)
```

BoW.Flexi Example 5b Virtual Wire

```
[CMD:3][Virtual_Wire:5]

Bits [2:0]:      CMD (101) Virtual Wire Command
Bits [7:3]:     Virtual Wire
```

Bits [end]: For Virtual wires less than 32, the number of data lanes needs to be taken into context to ensure all data bits transmitted are associated with the Virtual wires

BoW.Flexi Example 5b Virtual Wire with ECC

[CMD:3][ECC:8][Virtual_Wire:5]

Bits [2:0]: CMD (101) Virtual Wire
 Bits [10:3]: ECC
 Bits [15:11]: Virtual Wire
 Bits [end]: For Virtual wires less than 32, the number of data lanes needs to be taken into context to ensure all data bits transmitted are associated with the Virtual wires

11.8. Configuration Command Examples

Example: Configuration Read Request - Read Discovery Register (1-bit data lane)

Read from address 0x18 (Discovery Register)

Request packet:

Total: 3 (CMD) + 2 (LEN) +1 (WR) +1 (RD_RESP) + 8 (LOCAL_ADDR) = 15 bits

Request packet:

Byte 0: 0x07 = 0b00000111
 Bit 0: CMD[0] = 1
 Bit 1: CMD[1] = 1
 Bit 2: CMD[2] = 1
 Bit 3: LEN[0] = 0
 Bit 4: LEN[1] = 0
 Bit 5: WR = 0 (read)
 Bit 6: RD_RESP = 0 (request)
 Bit 7: LOCAL_ADDR[0] = 0

Byte 1: 0x0C = 0b00001100
 Bits 0-6: LOCAL_ADDR[7:1] = 0b0001100 (address 0x18)

Complete hex packet: 07 0C

Response packet (assuming register content = 0x03):

Total: 3 (CMD) + 2 (LEN) +1 (WR) +1 (RD_RESP) + 8 (LOCAL_ADDR) + 8 (DATA) = 23 bits

Byte 0: 0x47 = 0b01000111
 Bit 0: CMD[0] = 1
 Bit 1: CMD[1] = 1
 Bit 2: CMD[2] = 1
 Bit 3: LEN[0] = 0
 Bit 4: LEN[1] = 0
 Bit 5: WR = 0 (Read Response)
 Bit 6: RD_RESP = 1 (response)
 Bit 7: LOCAL_ADDR[0] = 0

Byte 1: 0x8C = 0b10001100
 Bits 0-6: LOCAL_ADDR[7:1] = 0b0001100 (address 0x18)
 Bit 7: DATA[0] = 1

Byte 2: 0x01 = 0b00000001

```

Bits 0-6: DATA[7:1] = 0b0000001
Complete hex packet: 47 8C 01

```

Figure 11.2. Configuration Read Request - Read Discovery Register (1-bit data lane)

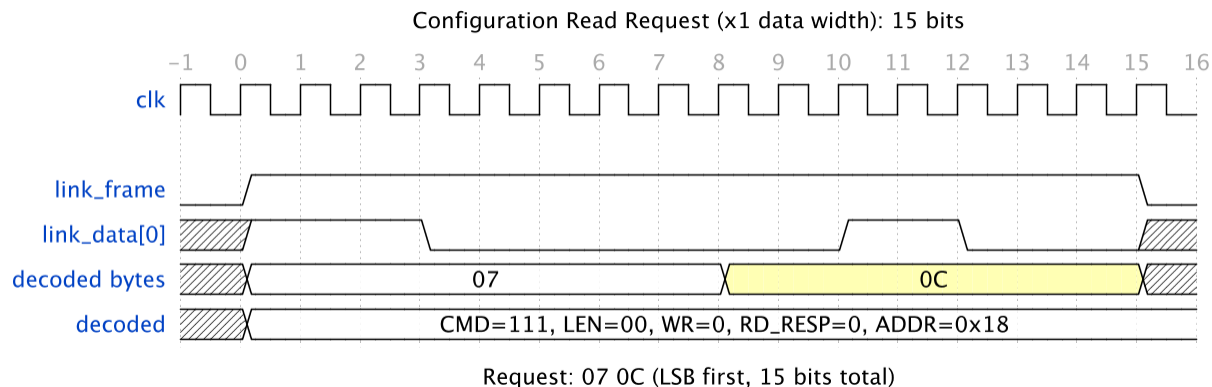
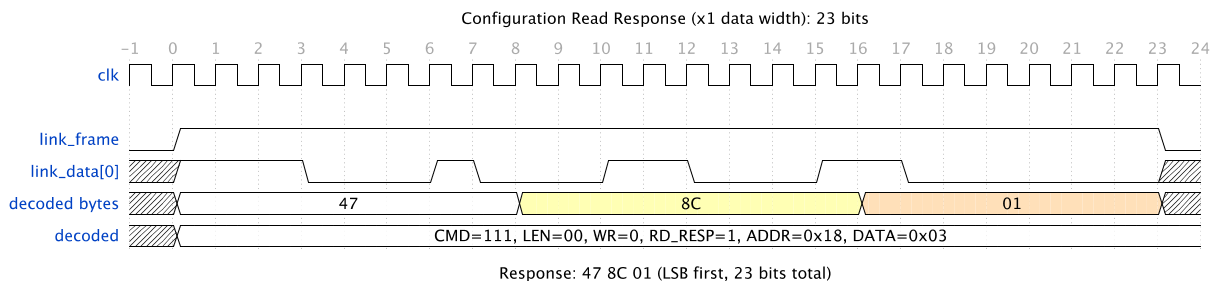


Figure 11.3. Configuration Read Response - Discovery Register (1-bit data lane)



11.8.1. Error Report Example

Example: Error Report - ECC Error Detected During Read (1-bit data lane)

Chiplet B receives a Read Request from Chiplet A but detects an uncorrectable (double-bit) ECC error in the received packet. Chiplet B reports this error to Chiplet A by sending a Configuration/Error Packet (CMD=111) with WR=1 to the Remote Error Register (address 0x10).

Error Context:

- Error type: Uncorrectable ECC error (bit 4 = 1)
- Command that caused error: Read (error_cmd = 3'b001)
- Data size of failed command: 4 bytes (error_size = 3'b010)
- Error status value: 0x00000450

Error Status Register Value Breakdown

```

Bits [31:12]: 0x00000 (no implementation-specific error bits set)
Bits [11:9]: 010 (error_size = 4 bytes)
Bits [8:6]: 001 (error_cmd = Read)

```

```

Bit [5]: 0      (no overflow)
Bit [4]: 1      (ecc_error)
Bit [3]: 0      (no timeout)
Bit [2]: 0      (no size error)
Bit [1]: 0      (no cmd error)
Bit [0]: 0      (no frame error)

```

Error packet:

```

Total: 3 (CMD) + 2 (LEN) + 1 (WR) + 1 (RD_RESP) + 8 (LOCAL_ADDR) + 32 (DATA) = 47 bits
Padding: (8 - (47 mod 8)) mod 8 = 1 bit
Total with padding: 48 bits = 6 bytes

```

Error packet byte-by-byte:

```

Byte 0: 0x37 = 0b00110111
  Bit 0: CMD[0] = 1
  Bit 1: CMD[1] = 1
  Bit 2: CMD[2] = 1
  Bit 3: LEN[0] = 0
  Bit 4: LEN[1] = 1 (LEN=10: 32-bit data)
  Bit 5: WR = 1 (write to error register)
  Bit 6: RD_RESP = 0 (not a read response)
  Bit 7: LOCAL_ADDR[0] = 0

Byte 1: 0x08 = 0b00001000
  Bits 0-6: LOCAL_ADDR[7:1] = 0b00001000 (address 0x10)
  Bit 7: DATA[0] = 0 (frame_error = 0)

Byte 2: 0x28 = 0b00101000
  Bit 0: DATA[1] = 0 (cmd_error = 0)
  Bit 1: DATA[2] = 0 (size_error = 0)
  Bit 2: DATA[3] = 0 (timeout_error = 0)
  Bit 3: DATA[4] = 1 (ecc_error = 1)
  Bit 4: DATA[5] = 0 (overflow_error = 0)
  Bit 5: DATA[6] = 1 (error_cmd[0] = 1)
  Bit 6: DATA[7] = 0 (error_cmd[1] = 0)
  Bit 7: DATA[8] = 0 (error_cmd[2] = 0)

Byte 3: 0x02 = 0b00000010
  Bit 0: DATA[9] = 0 (error_size[0] = 0)
  Bit 1: DATA[10] = 1 (error_size[1] = 1)
  Bit 2: DATA[11] = 0 (error_size[2] = 0)
  Bits 3-7: DATA[16:12] = 0b000000 (implementation dependent = 0)

Byte 4: 0x00 = 0b00000000
  Bits 0-7: DATA[24:17] = 0b00000000

Byte 5: 0x00 = 0b00000000
  Bits 0-6: DATA[31:25] = 0b00000000
  Bit 7: PAD[0] = 0

Complete hex packet: 37 08 28 02 00 00

```

Interpretation:

Upon receiving this packet, Chiplet A:

1. Identifies CMD=111 with WR=1 and RD_RESP=0 as an error report

2. Writes the 32-bit error status (0x00000450) to its local Remote Error Register
3. The firmware reads the Remote Error Register and determines:
 - a. An uncorrectable ECC error occurred (bit 4)
 - b. The error happened during a Read command (bits [8:6] = 001)
 - c. The read was requesting 4 bytes of data (bits [11:9] = 010)
4. The firmware can then take appropriate recovery action (e.g. retry the read, reset the link)



The error report is a configuration write as part of the Configuration/Error Command — no response packet is sent from Chiplet A.

11.8.2. Example sequence of interface configuration

Chiplet A - Manager

Chiplet B - Subordinate

1. Using x1 data lane, Chiplet A issues (Configuration CMD) read request to discovery register chiplet B 0x18 ([Discovery Register](#))
2. Using x1 data lane, Chiplet B sends a Configuration Read-Response with the register content
3. Chiplet A decides on the configuration
4. Using x1 data lane, Chiplet A issues (Configuration CMD) writes to Chiplet B Interface Configuration Register 0x0C ([Interface Configuration Register](#)) to configure Chiplet B's Rx and Tx (Set Rx and Tx to xN data lanes)
5. Chiplet A configures its interface to match the configuration of Chiplet B
6. Using xN data lanes, Chiplet A issues (Configuration CMD) reads to Chiplet B Interface Configuration Register, 0x0C to confirm the interface is configured as expected.

11.8.3. Complete Configuration Example

Scenario: Configure Chiplet B to set x4 data lanes and activate the interface

Step 1: Read current configuration (Address 0x0C)

Request packet:

```
[CMD=111][LEN=10][WR=0][RD_RESP=0][ADDR=0x0C]
```

Step 2: Receive current value (x1 default, ECC enabled, address_width=16)



The contents of the Discovery and Interface Configuration registers are set at design time (via NVM/OTP or hard-wiring) and therefore do not necessarily match the protocol defaults. The value 0x00005B1E returned in this example decodes as:

+

Table 11.1. Configuration and Status Register decode of 0x00005B1E

Field	Value	Meaning
fixed_size (31:29)	000	0 bytes (unused; fixed_size_enable=0)
fixed_size_enable (28)	0	SIZE transmitted with each system command
adv_*_data_lane (27,21)	0,0	Use one-hot lane-enable bits below
x32/x16/x8/x4/x2 tx (26:22)	00000	x1 Tx (default)
x32/x16/x8/x4/x2 rx (20:16)	00000	x1 Rx (default)
is_manager (15)	0	Reading side does not configure the link
tx_data_size_max (14:12)	101=5	32-byte max Tx transfer
rx_data_size_max (11:9)	101=5	32-byte max Rx transfer
ecc_enabled (8)	1	ECC enabled
virtual_wires_enabled (7)	0	Virtual-Wires disabled
address_width (6:1)	001111=0x0F	16-bit address (1 + register_value)
interface_configured (0)	0	Configuration not yet committed

Response packet:

```
[CMD=111][LEN=10][WR=0][RD_RESP=1][ADDR=0x0C][DATA=0x00005B1E]
```

Step 3: Write new configuration (set x4 Tx bit 23, x4 Rx bit 17, set bit 0 interface configured)**Request:**

```
[CMD=111][LEN=10][WR=1][RD_RESP=0][ADDR=0x0C][DATA=0x00825B1F]
```

11.8.3.1. Using ECC with configuration registers

Example: Configuration Read Request with Hsiao ECC - Read Discovery Register (8-bit data lane)

Read from address 0x18 (Discovery Register)

```
[CMD:3][ECC:8][LEN:2][WR:1][RD_RESP:1][LOCAL_ADDR:8][DATA:variable][PAD:variable]
```

Request packet: 3 bytes (24 bits)

Protected data: CMD=111, LEN=00, WR=0, RD_RESP=0, ADDR=0x18

ECC calculation of Request Packet using Hsiao matrix: See [Table 6.1](#) through [Table 6.4](#) for complete encoding matrices. Bit positions listed are non-ECC data positions where the Hsiao matrix row has value=1 (within the 23-bit request packet).

```
m[3]: XOR(bits 0,1,2,12,13,15,18,19,21) = 1^1^1^0^0^0^1^1^0 = 1
m[4]: XOR(bits 0,1,11,12,14,16,18,20,22) = 1^1^0^0^0^0^0^1^0^0 = 1
m[5]: XOR(bits 0,2,11,13,14,17,19,20) = 1^1^0^0^0^0^0^1^0 = 1
m[6]: XOR(bits 1,2,11,15,16,17,21,22) = 1^1^0^0^0^0^0^0^0 = 0
m[7]: XOR(bits 12,13,14,15,16,17) = 0^0^0^0^0^0^0 = 0
m[8]: XOR(bits 18,19,20,21,22) = 1^1^0^0^0 = 0
m[9]: 0 (positions 28-31 outside packet)
m[10]: 0 (no non-ECC positions)
ECC[7:0] = {m[10],m[9],m[8],m[7],m[6],m[5],m[4],m[3]} = 0b00000111 = 0x07

Byte 0: 0x3F = 0b00111111
  CMD[2:0] = 111, ECC[4:0] = 0b00111
Byte 1: 0x00 = 0b00000000
  ECC[7:5] = 0b000, LEN[1:0] = 00, WR = 0, RD_RESP = 0, LOCAL_ADDR[0] = 0
Byte 2: 0x0C = 0b00001100
  LOCAL_ADDR[7:1] = 0b0001100 (address 0x18), PAD = 0
Complete hex packet: 3F 00 0C
```

ECC calculation of Response Packet using Hsiao matrix, (register content = 0x03):

Computed by the same procedure as the Request packet above — XOR the data bits selected by each Hsiao matrix row, this time over the 31-bit response packet (positions 0–30) which adds the 8-bit DATA payload. The full XOR expansions are omitted for brevity; the resulting syndrome is:

```
ECC[7:0] = {m[10],m[9],m[8],m[7],m[6],m[5],m[4],m[3]} = 0b00001100 = 0x0C

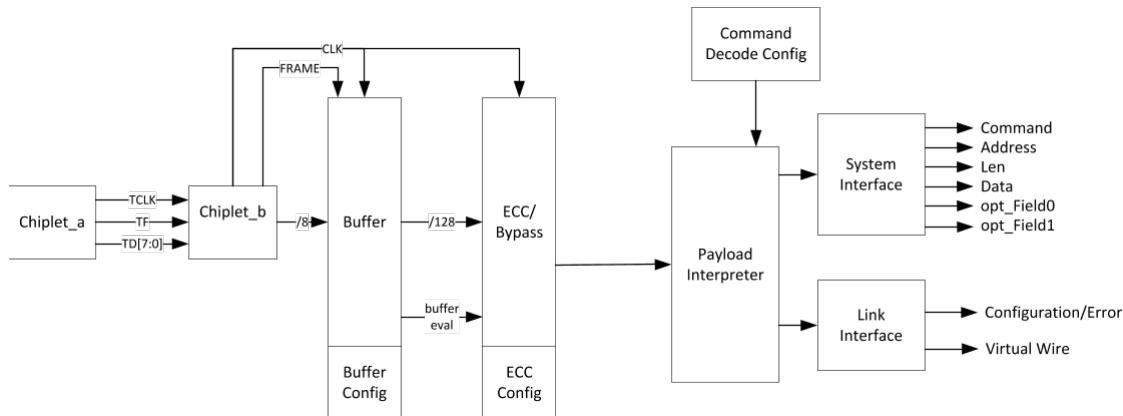
Byte 0: 0x67 = 0b01100111
  CMD[2:0] = 111, ECC[4:0] = 0b01100
Byte 1: 0x40 = 0b01000000
  ECC[7:5] = 0b000, LEN[1:0] = 00, WR = 0, RD_RESP = 1, LOCAL_ADDR[0] = 0
Byte 2: 0x8C = 0b10001100
  LOCAL_ADDR[7:1] = 0b0001100 (address 0x18), DATA[0] = 1
Byte 3: 0x01 = 0b00000001
  DATA[7:1] = 0b0000001, PAD = 0
Complete hex packet: 67 40 8C 01
```

If ECC is enabled the configuration register access will use the ECC in the same way as all other accesses.

If the interface does not respond as expected, the link can be reset using the method described in [Section 7.6.2](#).

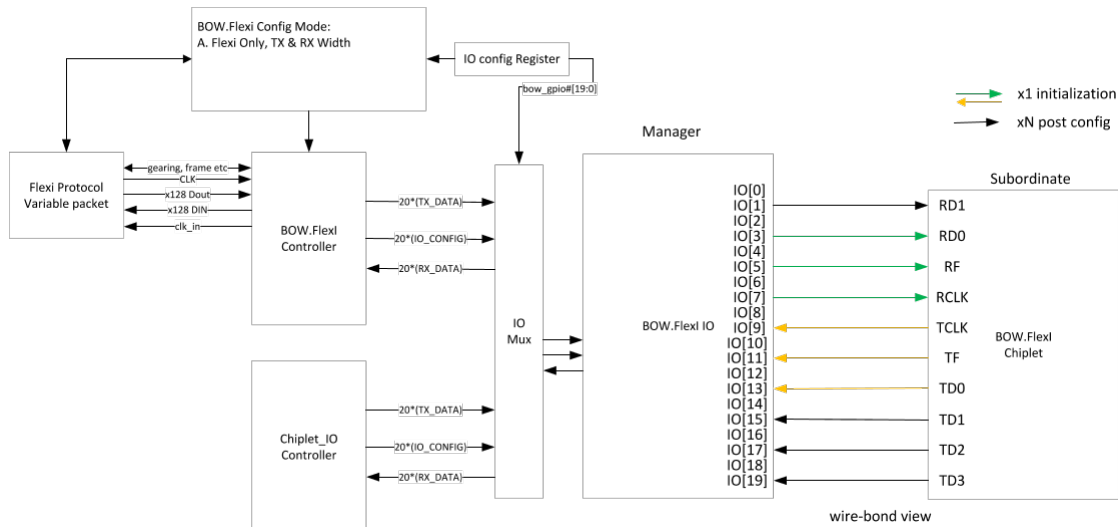
11.9. Command Decode Example

Figure 11.4. BoW.Flexi Command Decode Example



The figure above shows an example of a simple view of how a buffer could be used on the receive side to capture the transmitted data. The frame signal provides the indication of the start and end of the transmission. The payload interpreter takes information from the packet and can send the appropriate data out e.g. Virtual-Wires.

Figure 11.5. BoW.Flexi Example of how manager and subordinate connect



The manager device will likely be implemented on an advanced logic node. While the subordinate devices may exist on an older node where logic is not as abundant. The manager should be able to configure itself so that it can work with many different subordinate devices. For example the placement of the IOs may not physically be in the same locations. The manager can configure which IOs to send and receive signals from the subordinate device.

The BoW.Flexi manager will be capable of taking different data sources and mapping them to the BoW.Flexi PHY. The IOs driven out for the BoW.Flexi are not necessarily in the linear order.

Chapter 12. How to map ARM AMBA Protocols to BoW.Flexi

12.1. Overview

This chapter provides a comparative summary of how three ARM AMBA bus protocols — APB, AHB, and AXI — map to the BoW.Flexi packet format. The objective here is to identify how fields are mapped in the different command packets for consistency. The System Interface is responsible for managing how the transactions occur and therefore outside of the BoW.Flexi specification. There may be additional mechanisms e.g. credits, buffering etc. that would be required to enable these protocols, again, this is outside of the BoW.Flexi specification.

Each protocol has a single mapping with a default behavior that applies when no optional fields are configured. By default the mapping uses only the base BoW.Flexi packet structure (CMD + SIZE + ADDR + DATA), carrying the essential transaction information with minimal overhead. Optional fields (OPT_FIELD0–3) can then be individually enabled via the configuration registers to carry additional protocol-specific information such as protection attributes, burst transfers, locked sequences, and user-defined fields. AXI is an exception in that its transaction ID (OPT_FIELD0) is enabled on every command, since IDs are fundamental to AXI transaction ordering.

The three protocols are:

1. **APB** — Simple peripheral access; no bursts, no IDs. Defaults to no optional fields, with optional PPROT enabled via configuration.
2. **AHB** — Single-manager memory-mapped access. Defaults to single-beat transfers with no optional fields; protection (HPROT), burst (HBURST), locked transfers (HMASTLOCK), and user-defined fields can be individually enabled via configuration.
3. **AXI** — High-performance memory-mapped access supporting multiple outstanding and out-of-order transactions. Carries a transaction ID (OPT_FIELD0) on every command, burst length (OPT_FIELD1) and burst attributes plus an optional user-defined field (OPT_FIELD2) on requests, and response status (OPT_FIELD3) on responses.

12.2. Base Packet Fields (Common to All Protocols)

All three ARM protocol bridges share the following base BoW.Flexi fields:

Table 12.1. Base Packet Fields

Field	Bits	Usage
CMD	3	Command type: 000=Non-Posted Write; 001=Read; 010=Read Response; 011=Write Response; 100=Posted Write
SIZE	3	Transfer size per beat: 2^{SIZE} bytes. All protocol bridges use the standard BoW.Flexi 3-bit SIZE field (max 128 bytes)

Field	Bits	Usage
ADDR	1–64	Target address (configurable width via address_width in Configuration Command; default 32 bits)
DATA	Variable	Write data (requests) or read data (responses). Width determined by SIZE

12.3. Optional Field Usage Comparison

The table below summarizes how each protocol uses the four optional fields:

Table 12.2. Optional Field Usage Comparison

Optional Field	APB	AHB	AXI
OPT_FIELD0	PPROT[2:0] + USER (3+N bits); Optional; Requests only	HPROT[3:0] + USER (4+N bits); Optional; Requests only	Transaction ID (4–8 bits); All commands
OPT_FIELD1	Not used	HBURST[2:0] (3 bits); Optional; Requests only	AXLEN[7:0] (8 bits); Requests only
OPT_FIELD2	Not used	HMASTLOCK (1 bit); Optional; Requests only	Burst attributes + USER (14+N bits); Requests only
OPT_FIELD3	Not used	Not used	RESP[1:0] (2 bits); Responses only



Both APB and AHB report errors using the BoW.Flexi Configuration/Error Packet (CMD=111) mechanism rather than embedding a response status in an optional field. The presence of a response packet (CMD=010 or CMD=011) implies success (PSLVERR=0 or HRESP=OKAY). This approach avoids adding overhead to every response for an infrequent error condition. AXI, by contrast, carries the AXI response status (RESP[1:0]) in OPT_FIELD3 of response packets, because AXI defines richer response encodings (OKAY, EXOKAY, SLVERR, DECERR).

12.4. Optional Field Bit Ordering

Each optional field is fixed in size at configuration time and enabled per-command. Sub-fields are packed least-significant-bit first following the BoW.Flexi optional-field packing convention (see [Section 5.1.1.2](#)): the most frequently used sub-field occupies the low bits and the variable-width USER defined field, when present, occupies the most-significant bits. The following table defines the standardized bit layout of each optional field for the three protocols.

Table 12.3. Optional Field Bit Ordering

Protocol	Optional Field	Size (bits)	Bit Layout (LSb-first)	Enabled For
APB	OPT_FIELD0	3+N	[2:0]=PPROT[2:0]; [3+N-1:3]=PUSER[N-1:0]	Requests

Protocol	Optional Field	Size (bits)	Bit Layout (LSb-first)	Enabled For
APB	OPT_FIELD1	—	Not used	—
APB	OPT_FIELD2	—	Not used	—
APB	OPT_FIELD3	—	Not used (PSLVERR via CMD=111)	—
AHB	OPT_FIELD0	4+N	[3:0]=HPROT[3:0]; [4+N-1:4]=HUSER[N-1:0]	Requests
AHB	OPT_FIELD1	3	[2:0]=HBURST[2:0]	Requests (burst_en=1)
AHB	OPT_FIELD2	1	[0]=HMASTLOCK	Requests (lock_en=1)
AHB	OPT_FIELD3	—	Not used (HRESP via CMD=111)	—
AXI	OPT_FIELD0	4–8	[w-1:0]=ID (AWID/ARID/BID/RID)	All commands
AXI	OPT_FIELD1	8	[7:0]=AXLEN[7:0]	Requests
AXI	OPT_FIELD2	14+N	[1:0]=AXBURST; [2]=AXLOCK; [6:3]=AXCACHE; [9:7]=AXPROT; [13:10]=AXQOS; [13+N:14]=AxUSER[N-1:0]	Requests
AXI	OPT_FIELD3	2	[1:0]=xRESP[1:0]	Responses



N denotes the configurable USER defined field width (`user_width` in the AMBA Configuration Register); when `user_width` = 0 the USER sub-field is omitted. w denotes the configurable AXI ID width (`id_width`, 1–8 bits).

12.5. Optional Field Configuration Registers

The optional-field configuration is split into a protocol-agnostic transport definition and the protocol-specific content widths:

- The Optional Field Configuration Register and Optional Field Extended Configuration Register define the **transport**: the fixed size of each field (`opt_fieldN_size`) and the per-command presence mask (`opt_fieldN_cmd_support`).
- The AMBA Configuration Register defines the **content**: which protocol attributes ride in each field and their individual sub-widths (`user_width`, `id_width`, `data_width`, and the per-attribute enables).

The table below gives the standardized register settings that realize the bit ordering above for each protocol. Both sides of the link shall be programmed identically so that every transmitted bit is reconstructed at the same bit position by the receiver.

Table 12.4. Standardized Optional Field Register Settings

Protocol	Field	opt_fieldN_size	opt_fieldN_cmd_support	Presence
APB	OPT_FIELD0	3 + user_width	7'b0010011	Requests only
APB	OPT_FIELD1	0	7'b0000000	Disabled
APB	OPT_FIELD2	0	7'b0000000	Disabled
APB	OPT_FIELD3	0	7'b0000000	Disabled
AHB	OPT_FIELD0	4 + user_width	7'b0010011	Requests only
AHB	OPT_FIELD1	3	7'b0010011	Requests only (burst_en=1)
AHB	OPT_FIELD2	1	7'b0010011	Requests only (lock_en=1)
AHB	OPT_FIELD3	0	7'b0000000	Disabled
AXI	OPT_FIELD0	id_width (1–8)	7'b0011111	All commands
AXI	OPT_FIELD1	8	7'b0010011	Requests only
AXI	OPT_FIELD2	14 + user_width	7'b0010011	Requests only
AXI	OPT_FIELD3	2	7'b0001100	Responses only



The `opt_fieldN_cmd_support` mask bit positions map to commands as: bit[0]=Non-Posted Write (CMD=000), bit[1]=Read (CMD=001), bit[2]=Read Response (CMD=010), bit[3]=Write Response (CMD=011), bit[4]=Posted Write (CMD=100). A request-only field therefore uses 7'b0010011, a response-only field uses 7'b0001100, and a field present on all system commands uses 7'b0011111.

12.6. Packet Structure Comparison

The tables below show the packet structure for each protocol. The **Overhead** column shows the total bits excluding ADDR and DATA fields — this represents the fixed protocol overhead per packet.

12.6.1. Write Request Packets (Non-Posted, CMD=000)

Table 12.5. Write Request Packet Structure

Protocol	Packet Structure (per payload order)	Overhead (bits)
APB	[CMD:3][SIZE:3][ADDR][DATA][PAD] (default); [OPT_FIELD0:PPROT(3)+USER(N)] added when enabled	6 (9+N configured)
AHB	[CMD:3][SIZE:3][ADDR][DATA][PAD] (default); [OPT_FIELD0:HPROT(4)+USER(N)][OPT_FIELD1:HBURST(3)][OPT_FIELD2:HMASTLOCK(1)] added when enabled	6 (14+N configured)
AXI	[CMD:3][OPT_FIELD0:ID(4-8)][OPT_FIELD1:AXLEN(8)][OPT_FIELD2:ATTR(14)+USER(N)][SIZE:3][ADDR][DATA(+WSTRB)][PAD]	32+N (ID=4)

12.6.2. Read Request Packets (CMD=001)

Table 12.6. Read Request Packet Structure

Protocol	Packet Structure (per payload order)	Overhead (bits)
APB	[CMD:3][SIZE:3][ADDR][PAD] (default); [OPT_FIELD0:PPROT(3)+USER(N)] added when enabled	6 (9+N configured)
AHB	[CMD:3][SIZE:3][ADDR][PAD] (default); [OPT_FIELD0:HPROT(4)+USER(N)][OPT_FIELD1:HBURST(3)][OPT_FIELD2:HMASTLOCK(1)] added when enabled	6 (14+N configured)
AXI	[CMD:3][OPT_FIELD0:ID(4-8)][OPT_FIELD1:AXLEN(8)][OPT_FIELD2:ATTR(14)+USER(N)][SIZE:3][ADDR][PAD]	32+N (ID=4)

12.6.3. Read Response Packets (CMD=010)

Table 12.7. Read Response Packet Structure

Protocol	Packet Structure (per payload order)	Overhead (bits)
APB	[CMD:3][DATA][PAD]	3
AHB	[CMD:3][DATA][PAD]	3

Protocol	Packet Structure (per payload order)	Overhead (bits)
AXI	[CMD:3][OPT_FIELD0:ID(4-8)][OPT_FIELD3:RESP(2)][DATA][PAD]	9 (ID=4)

12.6.4. Write Response Packets (CMD=011)

Table 12.8. Write Response Packet Structure

Protocol	Packet Structure (per payload order)	Overhead (bits)
APB	[CMD:3][PAD]	3
AHB	[CMD:3][PAD]	3
AXI	[CMD:3][OPT_FIELD0:ID(4-8)][OPT_FIELD3:RESP(2)][PAD]	9 (ID=4)

12.7. Feature Comparison

Table 12.9. Feature Comparison

Feature	APB	AHB	AXI
Burst support	No	Optional (4/8/16 beats)	Yes (up to 256 beats)
Transaction IDs	No	No	Yes (4–8 bits)
Multiple outstanding	No	No	Yes
Out-of-order completion	No	No	Yes
Protection attributes	Optional (PPROT 3 bits)	Optional (HPROT 4 bits)	Yes (AxCACHE 4 bits)
Locked transfers	No	Optional (HMASTLOCK)	Yes (AxLOCK)
Cache attributes	No	No	Yes (AxQOS 4 bits)
QoS	No	No	Yes (AxQOS 4 bits)
USER defined field	Optional (N bits)	Optional (N bits)	Optional (N bits)
Write byte strobes	Optional (PSTRB)	Optional (HWSTRB)	Yes (WSTRB)
Error reporting	Error mechanism (CMD=111)	Error mechanism (CMD=111)	Response status (RESP: OKAY; EXOKAY; SLVERR; DECERR)
Posted writes	Optional	Optional	Yes (CMD=100)

12.8. Packet Overhead Comparison (x8 Data Lane, 32-bit Bus, 32-bit Address)

The following table shows the clock cycle count for each transaction type at a x8 data lane width:

Table 12.10. Packet Overhead Comparison

Transaction	APB	AHB	AXI
Write Request (single; 32-bit)	9 cycles	9 cycles	12 cycles
Read Request (32-bit address)	5 cycles	5 cycles	8 cycles
Read Response (32-bit)	5 cycles	5 cycles	6 cycles
Write Response	1 cycle	1 cycle	2 cycles
4-beat Burst Write (optional fields enabled)	N/A	22 cycles	24 cycles



Cycle counts assume: 32-bit address, 32-bit data, PSTRB/WSTRB disabled, ECC disabled, AHB USER width = 0, no byte strobes appended to DATA, x8 data lane. Both protocol bridges use the standard BoW.Flexi 3-bit SIZE field. By default, APB and AHB use no optional fields, achieving identical base overhead. Enabling the AHB optional fields (HPROT + HBURST/LOCK) adds 8 bits of overhead and enables burst capability. AXI cycle counts assume an ID width of 4 bits (USER width = 0) with OPT_FIELD0/1/2 enabled on requests and OPT_FIELD0/3 enabled on responses; AXI always carries the transaction ID, so it has higher base overhead than APB/AHB.

Chapter 13. BoW.Flexi for Memories

13.1. Motivation

In a chiplet system, there is a desire to be able to expand memory (volatile/non-volatile) by adding a memory chiplet. Memory devices have traditionally been constrained by package signal count. In a chiplet system, this constraint is not as rigid and it may be possible to increase the number of chiplet wires to achieve the performance requirements.

Memories typically operate with a CMD/ADDR and DATA type interface. This could be a fully parallel interface or mapped to a serial interface (e.g. OCTAL-xSPI JEDEC JESD251).

The read and write performance of a memory device is usually governed by an atomic read or write operation. The interface should be sized in such a way that it can request an atomic operation within the duration of that operation. A read request should not be longer than the time required to get the data out from the memory. If the data comes out in a single cycle, the read request should not be longer than a cycle.

DRAMs support bank operations (Activate, Refresh, Precharge) concurrent with read/writes data transfers.

Bank operations have their own time requirements that may be longer than a read/write operation. If this is the case, these requests could be spread over multiple clock cycles, with no impact to the read/write bandwidth.

If a read/write operation is spread over multiple cycles it is possible to interleave the read/write operations and the bank operations. The CMD/ADDRESS interface for a memory should be sized appropriately to ensure that it does not impact the overall read/write bandwidth.

13.1.1. Basic Concept

The BoW.Flexi protocol maps well to a memory interface. Its command structure can be used as is for the standard Read/Write Commands. BoW.Memory command structure [Table 13.1](#) can be configured to support the different types of memory. BoW.Flexi for Memory may benefit from some changes:

- Bi-Directional Data path
 - Support Bus turn-around to reduce data path signal count
 - Data (read/write) on a bi-directional data path are sent without a command
 - The frame signal indicates a transfer that includes a command
 - Read Data does not need a CMD header (i.e. no read-response)
- The Configuration/Error command works the same as the standard BoW.Flexi
- BoW.Flexi for memory supports pipelined read/write operation (multiple requests can be in-flight)
 - Configuration for the number of latency cycles
- Support for Wrapped (cache line) Read
- Support for Byte mask in the data path
- Use optional field to support multiple burst modes per transaction

Table 13.1. BoW.Flexi vs BoW.Memory Commands Descriptions

Cmd	BoW.Flexi	BoW.Memory Description
3'b000	Non-Posted Write	CMD0: e.g. Write Array + concatenate command
3'b001	Read data from Address	Read data from Address
3'b010	Read Response data	CMD2: e.g. Activate
3'b011	Write response	CMD3: e.g. Refresh
3'b100	Posted Write	Write data to Addr
3'b101	Virtual Wire (e.g. 5, 8, 32b)	CMD5: e.g. Read Array + concatenate command
3'b110	Extension Command	Extension Command
3'b111	Configuration/Error	Configuration/Error

When **protocol_mode** is set to a BoW.Flexi for Memory mode (DRAM or NVM) in Extension Register 1, the command semantics change to support Memory specific operations, as described in [Table 13.1](#).

13.2. DRAM Devices

Most volatile memories have similar read/write performance. The ratio of reads to writes is application dependent, however, in most cases, there will be more reads than writes.

The interface for standalone volatile memories normally supports equal read and write bandwidth by using a shared Data In/Out interface. The shared Data In/Out approach does enable a lower signal-count, but the complexity of the interface is required to manage the transition between reads and writes. The bus requires turn-around and cycles are added to ensure a smooth transition. On the memory controller side, writes are usually batched together to reduce the turn-around time.

The simplest chiplet interfaces are uni-directional with a separate Tx and Rx slice. Memory devices don't normally initiate transactions, so for BoW.Flexi for Memory, it is possible to have a uni-directional interface for the command, address and config information. Memory data traffic is often highly asymmetric (for example with far more reads than writes), so for BoW.Flexi for Memory it is possible to use a bi-directional bus for the read/write data. As described in the BoW Memory Addendum, BoW memory PHYs are unidirectional at any given time when in operation but can be operated as either Rx or Tx in a time-multiplexed (half-duplex) bi-directional manner to support read/write data. While the BoW Memory Addendum was focused primarily on high-speed mainband signals, those concepts (including bus turnaround between write and read data) can be applied to the BoW.Flexi specification.

BoW.Flexi for Memory will utilize the same configuration approach as BoW.Flexi. On power up, the manager device will initialize the memory device using the same approach as BoW.Flexi. The interface width and protocols used can be configured at this time. The configuration/error command will operate in the same way as in BoW.Flexi.

13.2.1. Frame Signal Usage for Data Transactions

The frame signal from the memory will be active for BoW.Flexi Configuration Commands e.g. 3'b111 configuration read response or during an error report. For memory data operations (e.g. read, write), the frame signal from the memory will remain low.

If the data interface from the memory is configured as bi-directional, the frame signal will be uni-directional (output from the memory) and will remain low for memory data operations (i.e. read and write). The frame signal will only be active for configuration/error command operations.

13.2.2. BoW.Flexi for DRAMs Concept

DRAM devices have two control interfaces:

- Data Transaction: Read/Writes to open pages
- Bank Management: Activate (open page), Precharge (close page), and Refresh

The number of banks, columns, and rows varies by device. The values below are provided as an example.

To maximize the data throughput, the bank management happens concurrently with the read/write transactions. The Read/Write Command requires the bank (3-bits) and column addresses (~7-bits). The Activate command (opening page) requires the bank address (3-bits) as well as the row address (~13-bits).

The simplest way of doing this is to effectively concatenate both interfaces into a single packet. A 2-bit command extension is added to support the following operations [Table 13.3](#)

The command structure below is used to communicate a read and activate operation.

```
[CMD:3][BANKa:3][COL:8][EXT:2][BANKb:3][ROW:13]

Read and Activate:
Bits [2:0]:      CMD (3 bits) (Read and Activate)
Bits [5:3]:      BANKa (3 bits)
Bits [13:6]:     COL (8 bits)
Bits [15:14]:    EXT (2 bits)
Bits [18:16]:    Bankb (3 bits)
Bits [31:19]:    ROW (13 bits)
Bits [end]:      PAD (padding to align to data lane boundary)
```

The above example could then be mapped to a 32-bit interface and sent in a single clock cycle.

This may not be the most economical approach from a signal perspective as the bank management transactions may be a lot less frequent compared to the data transactions. The memory requires the ability to accept a Read/Write transaction within an atomic memory operation. It also requires the ability to accept a bank transaction within its Row to Row Delay time (tRRD) time.

For DRAM here are a list of BoW.Flexi commands:

Table 13.2. BoW.Flexi DRAM Commands

Cmd	BoW.DRAM Description
3'b000	Write Array + Activate/Precharge/Refresh
3'b001	Read data from Address (Bank/Column)
3'b010	Activate (Bank/Row)
3'b011	Refresh/Pre-Charge (Bank)
3'b100	Write data to Addr (Bank/Column)
3'b101	Read Array + Activate/Precharge/Refresh
3'b110	Extension Command
3'b111	Configuration/Error

Example DRAM Commands

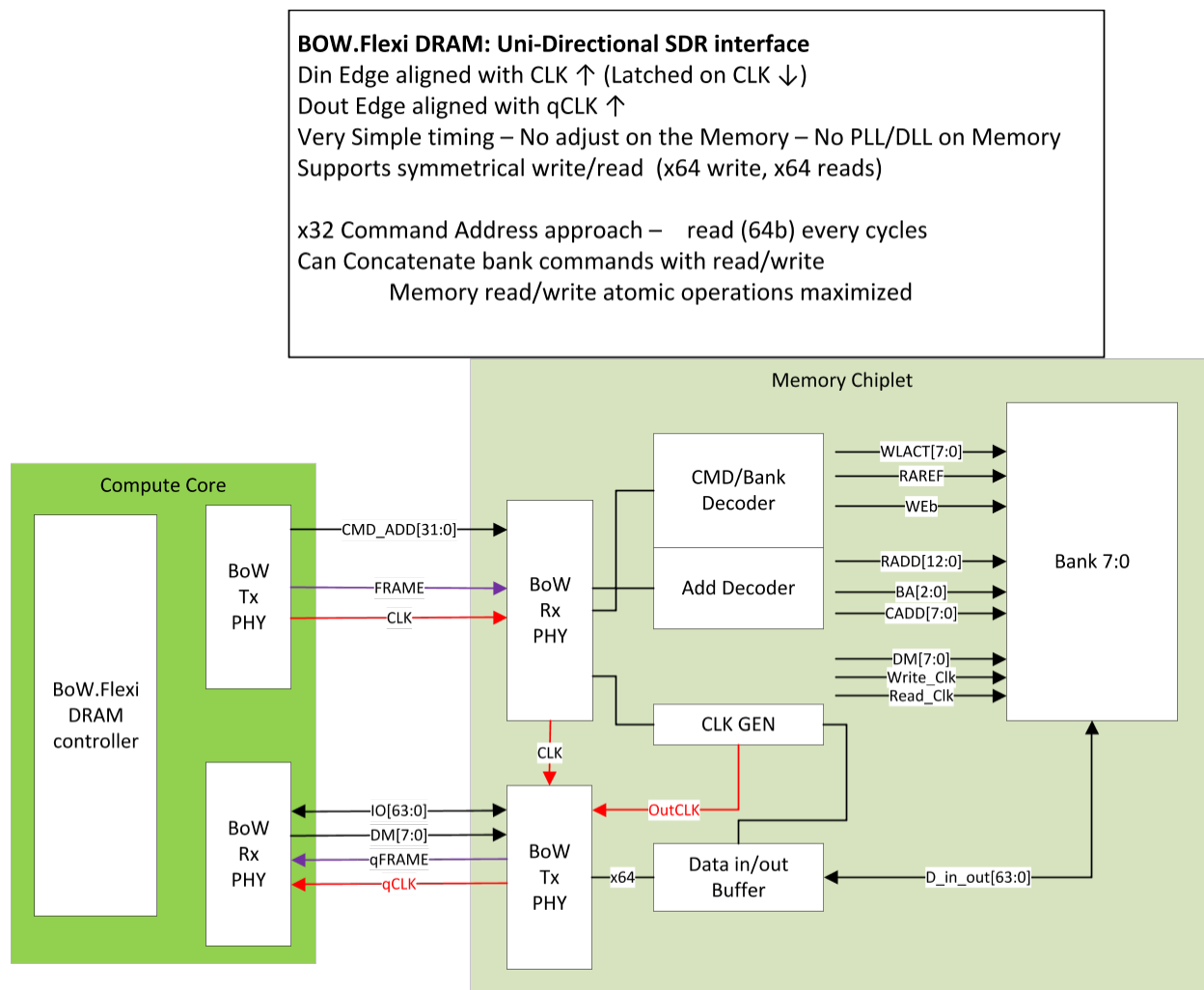
Read/Write	[CMD:3][BANKa:3][COL:8]
Activate	[CMD:3][BANKb:3][ROW:13]
Refresh/Precharge	[CMD:3][BANKb:3][REF:1]
Read/Write + Activate	[CMD:3][BANKa:3][COL:8][EXT:2][BANKb:3][ROW:13]
Read/Write + Refresh/Precharge	[CMD:3][BANKa:3][COL:8][EXT:2][BANKb:3][REF:1]

The commands 3'b000, 3'b101, provide a mechanism to add a bank command, see [Table 13.3](#).

Table 13.3. BoW.Flexi for Memory Extension Commands (e.g. 3'b000, 3'b101)

Cmd	Extension CMD Description
2'b11	Reserved
2'b10	Activate
2'b01	Pre-Charge/Refresh
2'b00	None

The simplest approach is to be able to concatenate two transactions per command, and send all the data in a single beat, as illustrated in [Figure 13.1](#). A 2-bit field (see [Table 13.3](#)) is used to add bank commands after a read/write command in the same transfer.

Figure 13.1. BoW.Flexi DRAM combined Bus x32

```
[CMD:3][BANKa:3][COL:8][EXT:2][BANKb:3][ROW:13][PAD:variable]
```

Read and Activate:

Bits [2:0]: CMD (3 bits) (Read and Extension) (000)

Bits [5:3]: BANKa (3 bits)

Bits [13:6]: COL (8 bits)

Bits [15:14]: EXT (2 bits)

Bits [18:16]: Bankb (3 bits)

Bits [31:19]: ROW (13 bits)

Bits [end]: PAD (padding to align to data lane boundary)

13.2.3. BoW.Flexi for DRAMs Byte Write Data Mask

DRAM devices typically support a byte write Data Mask (DM). An additional input bit is required for each byte. The DMs are sent using dedicated data inputs. The timing is the same as the data inputs.

13.3. Non Volatile Memory (NVM) Devices

13.3.1. Flash Devices

NVM devices typically support fast reads, with a write performance that is orders of magnitude slower. The device may have multiple banks, and this can be used to be able to support read-while-write. In the case of Flash devices, a page is written at a time. The data is written into a page buffer. When the buffer is complete, the write address is provided and the program sequence can be initiated. If we take an example of a 4Kbit page, the page buffer would be filled by completing 32 writes (128b).

Write Page Buffer:

```
[CMD:3][PageBufferAddress:5][DATA:128b][PAD:variable]
```

The program operation is launched by writing a register on the device.

Read:

In general for memory read, the full address is provided and the data will appear on the output bus after the read latency.

```
[CMD:3][ADDRESS:24][PAD:variable]
```

With this approach The CMD/ADDRESS/DATAin and the DATAout buses are uni-directional.

Register read/write operations use the Configuration Command 3'b111.

The configuration address space can be increased by writing the **config_register_address_bits** field in the [Interface Configuration Extension Register 2](#).

Table 13.4. BoW.Flexi NVM Commands

Cmd	BoW.Memory Description
3'b000	CMD0: e.g. Page Buffer Write
3'b001	Read data from Address
3'b010	CMD2: Reserved
3'b011	CMD3: Reserved
3'b100	Write data to Address (if supported)
3'b101	CMD4: Reserved
3'b110	Extension Command
3'b111	Configuration/Error

13.4. BoW.Flexi for Memory Registers

Table 13.5. BoW.Flexi for Memory Register Address Map

Register	Required	Address	Link
BoW.Flexi Memory Configuration Register	1b'1	0x74	Table 13.6
BoW.Flexi Memory Address Register	1b'1	0x78	Table 13.7
BoW.Flexi Memory Command Extension Register	1b'1	0x7C	Table 13.8

Table 13.6. BoW.Flexi for Memory Configuration Register

Bits	Identifier	Access	Reset	Description
31:26	Reserved			Reserved
25:24	burst_mode	RW	0	0 = Linear, 1: Wrapped 2: Dynamic use OPT_FIELD0 to Specify 3: Reserved
23:21	command_type	RW	0	0 = DRAM 1 = NVM 2+ = Reserved
20:19	power_down_mode	RW	0	0 = active 1 = self_refresh 2 = no retention
18:16	bus_turn_around_cycles	RW	1	Cycles
15:14	Reserved			Reserved
13:8	read_latency	RW	0	Number of Cycles from CMD to data appearing on bus
7:2	write_latency	RW	0	Number of Cycles from CMD to data appearing on bus
1	byte_mask_enabled	RW	0	Byte mask enabled
0	memory_interface_configured	RW	0	Memory interface has been configured

Table 13.7. BoW.Flexi for Memory Address Register

Bits	Identifier	Access	Reset	Description
31:25	Reserved			Reserved
24:19	register_space	RW	0	How many register addresses - will likely use Configuration Command for this
18	Reserved			Reserved
17	address_scheme_for_bank_operations	RW	0	0 = bank_column 1 = Reserved
16	addressing_scheme_for_read_write	RW	0	0 = bank_row 1 = bank_row_column
15:11	num_row_address	RW	0	Number of row addresses
10:6	num_column_address	RW	0	Number of column addresses
5:0	num_bank_address	RW	0	Number of bank addresses

Table 13.8. BoW.Flexi for Memory Command Extension Register

Bits	Identifier	Access	Reset	Description
31:9	Reserved			Reserved
8:6	cmd_10_pointer	RW	0x6	Pointer for Concatenate Command 10 e.g. Read
5:3	cmd_01_pointer	RW	0x2	Pointer for Concatenate Command 01 e.g. Activate
2:0	Reserved			Reserved

Appendix A. Example: AHB to Chiplet Bridge

A.1. Overview

This appendix describes an example of BoW.Flexi bridge protocol for ARM AMBA AHB bus systems. The AHB bridge is a configurable implementation that covers the full spectrum from the simplest minimum-overhead single-beat transfers (best for compatibility) through to full AHB with burst and locked transfer support.

The bridge converts AHB transactions into the BoW.Flexi serialized packet format for transmission across chiplet boundaries. Features are individually enabled and disabled via the AMBA Configuration Register, allowing a single implementation to be configured from minimum overhead to maximum capability.

A.1.1. Configuration Levels

The following table shows the features available at each configuration level:

Table A.1. AHB Bridge Feature Configuration

Feature	Default	Configured	Configuration Control
Single-beat transfers	Yes	Yes	Always available
Burst transfers (WRAP/INCR 4/8/16)	No	Yes	ahb_burst_enable
Protection attributes (HPROT)	No	Optional	ahb_prot_enable
Locked transfers (HMASTLOCK)	No	Optional	ahb_lock_enable
USER defined field	No	Optional	ahb_user_width
Write byte enables (WSTRB)	No	Optional	ahb_wstrb_in_data
Variable data bus width (32-256 bit)	No	Yes	ahb_data_width
Address width (1-64 bit)	Yes (default 32)	Yes (default 32)	address_width in Configuration Command
Error reporting	CMD=111	CMD=111	Always via error mechanism

At minimum configuration (all optional features disabled), the bridge uses no optional fields and packets contain only CMD + SIZE + ADDR + DATA. As features are enabled, additional optional fields carry the corresponding AHB protocol information.



The `protocol_mode` field in [Interface Configuration Extension Register 1](#) selects the AHB bridge (mode 1). The bridge defaults to no optional fields (the minimum-overhead configuration); individual features are enabled via the AMBA Configuration Register.

A.1.2. Posted vs Non-Posted Write Selection

The AHB bridge uses the `HPROT[2]` (bufferable) bit to determine whether a write transaction is posted or non-posted:

- **HPROT[2] = 1** (bufferable): The bridge issues a **Posted Write** (CMD=3'b100). The remote chiplet returns a Write Response (CMD=3'b011) as a flow-control acknowledgment once it has accepted the write and is ready for a further posted write; it need not wait for the write to complete downstream. The bridge stalls the local AHB bus (HREADY=0) until this Write Response or an Error Packet (CMD=3'b111) is received.
- **HPROT[2] = 0** (non-bufferable): The bridge issues a **Non-Posted Write** (CMD=3'b000). The bridge stalls the local AHB bus (HREADY=0) until a Write Response (CMD=3'b011) — sent only after the write completes — or an Error Packet (CMD=3'b111) is received.

Since the posted/non-posted distinction is encoded in the BoW.Flexi command field, the `HPROT[2]` bit does not need to be transmitted across the interface when `prot_en = 0` (the default). When `prot_en = 1`, `HPROT[3:0]` is transmitted in `OPT_FIELD0`, but the `HPROT[2]` bit is still used locally to select the CMD encoding.

A.2. Physical Interface

The AHB bridge operates over the standard BoW.Flexi physical layer (see [Section 7.5](#)). Each chiplet provides two uni-directional links regardless of which AHB features are enabled.

A.3. AHB Interface Signals

A.3.1. AHB Subordinate Bridge Interface

The Subordinate Bridge acts as an AHB subordinate on the local bus, accepting transactions from the local AHB manager and converting them to BoW.Flexi packets. The **Minimum** column indicates whether the signal is relevant in minimum configuration.

Table A.2. AHB Subordinate Bridge Signals

Signal	Width	Direction	Minimum	Description
HCLK	1	Input	Yes	AHB clock
HRESETn	1	Input	Yes	AHB reset (active low)
HSEL	1	Input	Yes	Subordinate select signal
HADDR	Variable	Input	Yes	Address bus (configurable width; default 32-bit)

Signal	Width	Direction	Minimum	Description
HTRANS[1:0]	2	Input	Yes	Transfer type (IDLE; BUSY; NONSEQ; SEQ). Minimum mode only uses IDLE/NONSEQ
HWRITE	1	Input	Yes	Write enable (1=write; 0=read)
HSIZE[2:0]	3	Input	Yes	Transfer size (bytes per beat)
HBURST[2:0]	3	Input	No	Burst type (SINGLE; WRAP4; INCR4; WRAP8; INCR8; WRAP16; INCR16). Only when ahb_burst_enable=1
HPROT[3:0]	4	Input	Optional	Protection control. Present when ahb_prot_enable=1
HMASTLOCK	1	Input	No	Locked sequence indicator. Only when ahb_lock_enable=1
HWDATA	Variable	Input	Yes	Write data bus (configurable 32/64/128/256 bit)
HREADY	1	Output	Yes	Transfer ready (subordinate can insert wait states)
HRESP	1	Output	Yes	Transfer response (0=OKAY; 1=ERROR)
HRDATA	Variable	Output	Yes	Read data bus (same width as HWDATA)

A.3.2. AHB Manager Bridge Interface

The Manager Bridge acts as an AHB manager on the remote bus, receiving BoW.Flexi packets and initiating AHB transactions.

Table A.3. AHB Manager Bridge Signals

Signal	Width	Direction	Minimum	Description
HCLK	1	Input	Yes	AHB clock
HRESETn	1	Input	Yes	AHB reset (active low)
HADDR	Variable	Output	Yes	Address bus (configurable width; default 32-bit)
HTRANS[1:0]	2	Output	Yes	Transfer type (IDLE; BUSY; NONSEQ; SEQ). Minimum mode only uses IDLE/NONSEQ
HWRITE	1	Output	Yes	Write enable (1=write; 0=read)
HSIZE[2:0]	3	Output	Yes	Transfer size (bytes per beat)
HBURST[2:0]	3	Output	No	Burst type. Only when ahb_burst_enable=1
HPROT[3:0]	4	Output	Optional	Protection control. Present when ahb_prot_enable=1

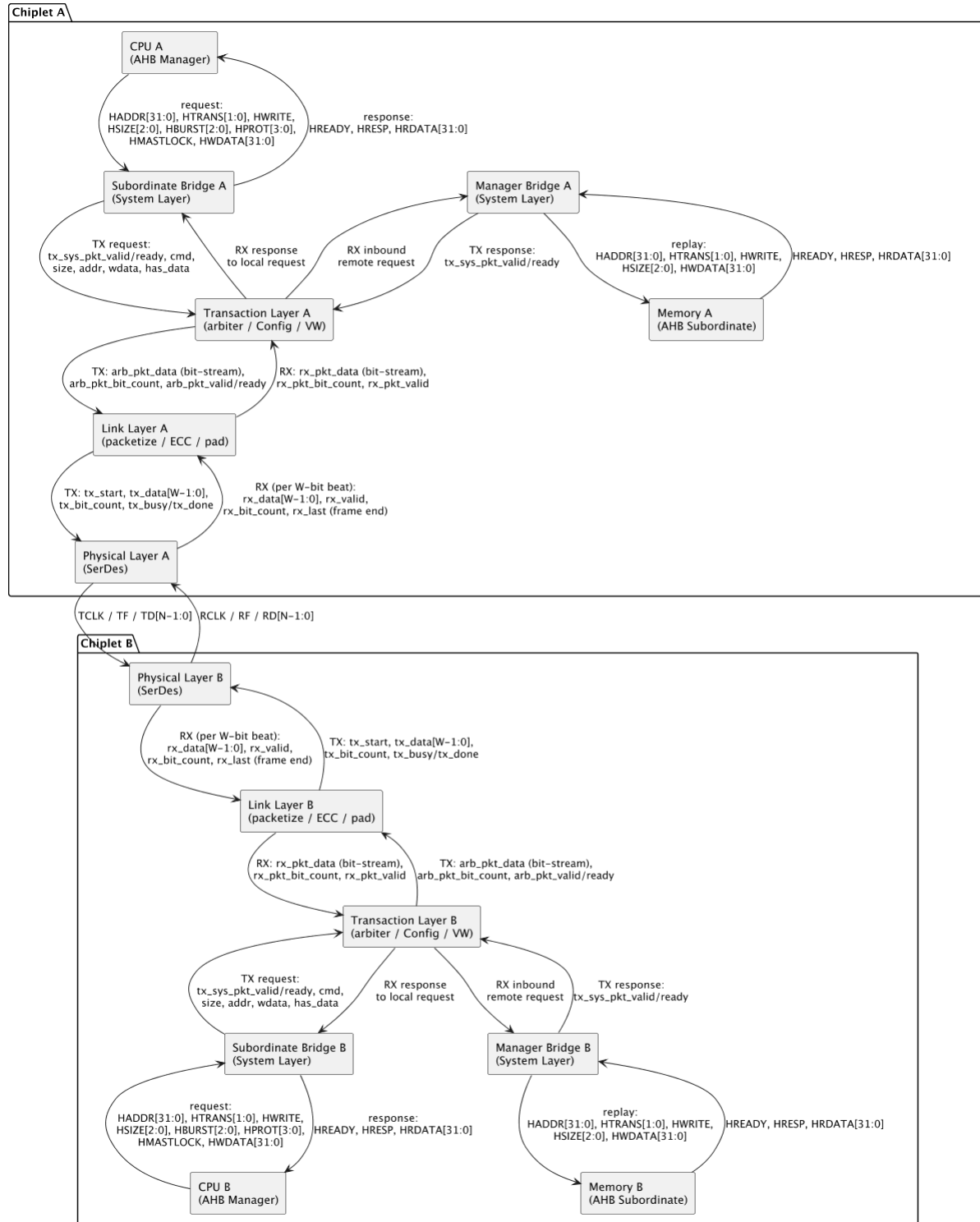
Signal	Width	Direction	Minimum	Description
HMASTLOCK	1	Output	No	Locked sequence indicator. Only when ahb_lock_enable=1
HWDATA	Variable	Output	Yes	Write data bus (configurable 32/64/128/256 bit)
HREADY	1	Input	Yes	Transfer ready from subordinate
HRESP	1	Input	Yes	Transfer response (0=OKAY; 1=ERROR)
HRDATA	Variable	Input	Yes	Read data bus (same width as HWDATA)



The data bus width (HWDATA/HRDATA) is configurable via `data_width` in the AMBA Configuration Register. Default is 32-bit. The address bus width (HADDR) is configurable from 1 to 64 bits via `address_width` in the Configuration Command. Default is 32-bit.

A.3.3. Complete Bridge System Interface

Figure A.1. Complete AHB Chiplet Bridge System



A.3.4. Bridge Signal Mapping

The following table shows how AHB signals map to BoW.Flexi packet fields. The **Minimum** column indicates whether the mapping applies in the default (minimum-overhead) configuration.

Table A.4. AHB to BoW.Flexi Packet Mapping

AHB Signal	Packet Field	Minimum	Notes
HADDR	ADDR	Yes	Address transmitted in little-endian byte order (width set by address_width in Configuration Command; 1-64 bits; default 32)
HSIZE[2:0]	SIZE[2:0]	Yes	Maps directly to SIZE field (request packets only)
HWRITE	CMD[2:0]	Yes	1=CMD[100] Posted Write or CMD[000] Non-Posted Write; 0=CMD[001] Read Request
HPROT[3:0]	OPT_FIELD0[3:0] (request only)	No (optional)	Protection control. Present when ahb_prot_enable=1. Not transmitted in minimum config
HUSER[N-1:0]	OPT_FIELD0[3+N:4] (request only)	No	USER defined field appended to HPROT. Present when ahb_user_width > 0
HBURST[2:0]	OPT_FIELD1[2:0] (request only)	No	Burst type encoding. Present when ahb_burst_enable=1
HMASTLOCK	OPT_FIELD2[0] (request only)	No	Locked sequence indicator. Present when ahb_lock_enable=1
HWDATA	DATA	Yes	Write data transmitted in little-endian byte order
HRDATA	DATA	Yes	Read data received in little-endian byte order
HRESP	-	Yes	Reported via Configuration/Error Packet (CMD=111). Presence of response packet implies OKAY
HTRANS[1:0]	-	Yes	Used internally by bridge to detect valid transfers (NONSEQ/SEQ)



In minimum mode (all optional fields disabled), packets contain only CMD + SIZE + ADDR + DATA — the minimum overhead configuration. HRESP errors are always reported using the Configuration/Error Packet (CMD=111) mechanism regardless of configuration level.

A.4. Optional Field Usage

The BoW.Flexi protocol supports up to 4 optional fields (OPT_FIELD0/1/2/3) that can be configured per-command via the Optional Field Configuration Register and Optional Field Extended Configuration Register. The AHB bridge utilizes these fields to carry protocol information beyond the base CMD/SIZE/ADDR/DATA.

Table A.5. AHB Optional Field Allocation

Optional Field	Size (bits)	Purpose in Requests	Purpose in Responses	Commands Enabled (cmd_support)
OPT_FIELD0	4+N (configurable)	Transfer attributes: HPROT[3:0] + optional USER[N-1:0] defined field	Not present in response packets	7'b0010011 (Non-Posted Write; Read; Posted Write)
OPT_FIELD1	3 (fixed)	Burst control: HBURST[2:0]	Not present in response packets	7'b0010011 (Non-Posted Write; Read; Posted Write)
OPT_FIELD2	1 (fixed)	Locked transfer: HMASTLOCK	Not present in response packets	7'b0010011 (Non-Posted Write; Read; Posted Write) when ahb_lock_enable=1; else disabled
OPT_FIELD3	Not used	Not present in request packets	Not used (HRESP reported via Configuration/Error Packet CMD=111)	7'b0000000 (disabled)



Each optional field has a **fixed size** set globally in the configuration registers (`opt_fieldN_size`). The size does not change per command — only the **presence** of a field can be enabled or disabled per command via `opt_fieldN_cmd_support`.



Sub-fields within each OPT_FIELD follow the BoW.Flexi optional-field packing convention — packed LSb-first with the most frequently used sub-field in the low bits and the USER defined field, when present, in the most-significant bits (see [Section 5.1.1.2](#)).

A.4.1. OPT_FIELD0: Transfer Attributes (Requests Only)

OPT_FIELD0 carries HPROT and optional USER defined field for request packets only. It is enabled for request commands (`opt_field0_cmd_support = 7'b0010011`: Non-Posted Write, Read, Posted Write).

The base 4-bit encoding:

Bits [3:0]: HPROT[3:0] (Protection control)

When `user_width > 0` in the AMBA Configuration Register, the USER defined field is appended:

Bits [4+N-1:4]: HUSER[N-1:0] (USER defined field)

The total `opt_field0_size` is set to 4 + `user_width`. When `user_width = 0` (default), `OPT_FIELD0` is 4 bits carrying HPROT only.

When `prot_en = 0`, `OPT_FIELD0` is disabled entirely (`opt_field0_cmd_support = 0`). This is the minimum configuration.

A.4.2. OPT_FIELD1: Burst Control (Requests Only)

`OPT_FIELD1` is fixed at 3 bits and carries the burst type for request packets only. It is enabled when `burst_en = 1`.

The 3-bit encoding:

Bits [2:0]: HBURST[2:0] (Burst type)

The HBURST encoding determines both the burst type and beat count:

- **SINGLE (3'b000)**: 1 beat
- **INCR (3'b001)**: NOT SUPPORTED — bridge shall respond with error
- **WRAP4 (3'b010)**: 4 beats, wrapping
- **INCR4 (3'b011)**: 4 beats, incrementing
- **WRAP8 (3'b100)**: 8 beats, wrapping
- **INCR8 (3'b101)**: 8 beats, incrementing
- **WRAP16 (3'b110)**: 16 beats, wrapping
- **INCR16 (3'b111)**: 16 beats, incrementing

Configuration controls:

- When `burst_en = 0`: `OPT_FIELD1` is disabled (single transfers only, `opt_field1_cmd_support = 0`)



AHB does not require a separate burst length field. The fixed burst lengths (4, 8, 16 beats) are fully encoded in `HBURST[2:0]`.



The AHB undefined-length INCR burst (`HBURST=001`) is not supported by the BoW.Flexi AHB bridge. Only fixed-length bursts are supported, as BoW.Flexi packets require a known length at transmission time.

A.4.3. OPT_FIELD2: Locked Transfers — HMASTLOCK (Requests Only)

OPT_FIELD2 is fixed at 1 bit and carries the locked-sequence indicator for request packets only. It is enabled independently of bursts when `lock_en = 1`.

The 1-bit encoding:

Bit [0]: HMASTLOCK (Locked sequence indicator)

Configuration controls:

- When `lock_en = 0`: OPT_FIELD2 is disabled (`opt_field2_cmd_support = 0`) and no lock bit is transmitted.



HMASTLOCK is kept in its own field, separate from the burst type, so a locked **single** transfer can be expressed without enabling bursts (and vice-versa). Enforcing the locked sequence — holding the remote arbiter and not interleaving other traffic across the locked window — is a System-Layer responsibility; OPT_FIELD2 only transports the hint the far bridge replays as HMASTLOCK on its remote AHB bus.

A.4.4. Write Byte Enables — Embedded in DATA

When per-byte write selection is required, write byte enables are embedded in the DATA field (1 enable bit per data byte), controlled by `wstrb_in_data` in the AMBA Configuration Register:



Both `wstrb_in_data` (AMBA Configuration Register) and `write_data_mask_enabled` ([Interface Configuration Extension Register 1](#)) shall be set to 1 to enable write byte enable functionality. These two fields control the same underlying mechanism — both shall be enabled for write strobes to be transmitted.

When both `wstrb_in_data` and `write_data_mask_enabled` are enabled:

Per beat: [HWDATA: DATA_BUS_WIDTH bits][WSTRB: DATA_BUS_WIDTH/8 bits]

When `wstrb_in_data = 0` or `write_data_mask_enabled = 0` (default, including minimum mode), write enables are not transmitted and all bytes are assumed valid.

A.5. AHB Configuration Register

The AHB bridge is configured by the unified AMBA Configuration Register (offset 0x70, see [Table 10.31](#)). In AHB mode the applicable fields are `user_width`, `data_width`, `burst_en`, `lock_en`, `prot_en`, `wstrb_in_data`, and `bridge_en`. This register controls which AHB attributes are transmitted across the link. For minimum-overhead operation, set `burst_en = 0`, `prot_en = 0`, and `lock_en = 0`.

A.6. Packet Structure

A.6.1. Packet Format

All AHB transaction packets follow the BoW.Flexi payload order (Table 5.3). Only enabled fields are present in the packet. The general template is:

```
[CMD:3] [OPT_FIELD0] [OPT_FIELD1] [OPT_FIELD2] [SIZE:3] [ADDR:variable]
[DATA(+WSTRB):variable] [PAD:variable]
```

The following table shows which fields are present for each command and configuration level:

Table A.6. AHB Packet Field Presence

Transaction	CMD	OPT_FIELD0 (HPROT+USER)	OPT_FIELD1 (HBURST)	OPT_FIELD2 (HMAST LOCK)	SIZE + ADDR	DATA	WSTRB append ed
Write (minimum mode)	000/100	—	—	—	Yes	Yes	No
Write (full, wstrb enabled)	000/100	HPROT+USER(4 +N)	HBURST(3)	HMAST LOCK(1)	Yes	Yes	Yes
Write (full, wstrb disabled)	000/100	HPROT+USER(4 +N)	HBURST(3)	HMAST LOCK(1)	Yes	Yes	No
Read Request (minimum mode)	001	—	—	—	Yes	—	—
Read Request (full)	001	HPROT+USER(4 +N)	HBURST(3)	HMAST LOCK(1)	Yes	—	—
Read Response	010	—	—	—	—	Yes	—
Write Response	011	—	—	—	—	—	—

All packets are padded to align to the data-lane boundary. Minimum mode corresponds to `prot_en = 0`, `burst_en = 0`, `lock_en = 0`.



Response packets (CMD=010 and CMD=011) do NOT include an HRESP field. The presence of a response packet implies HRESP=OKAY. Errors are always reported via Configuration/Error Packet (CMD=111). Response packets can include optional fields, however, it is outside the scope of this example.

Table A.7. AHB Transaction Field Descriptions

Field	Bits	Description
CMD	3	Command type (000=Non-Posted Write; 001=Read; 010=Read Response; 011=Write Response; 100=Posted Write)
OPT_FIELD0 (HPROT+USER)	4+N	Request packets only. Base 4 bits: HPROT[3:0]. Optional USER[N-1:0] defined field appended when ahb_user_width > 0. Not present in response packets
OPT_FIELD1 (HBURST)	3 (fixed)	Request packets only. HBURST[2:0]. Burst length is implicit from HBURST encoding (4/8/16 beats). Present when ahb_burst_enable=1. Not present in response packets
OPT_FIELD2 (HMASTLOCK)	1 (fixed)	Request packets only. HMASTLOCK locked-sequence indicator, independent of bursts. Present when ahb_lock_enable=1. Not present in response packets
SIZE	3	Transfer size per beat (2 ^{SIZE} bytes). Request packets only
ADDR	32 or 64	Target address (width set by configuration register). Request packets only
DATA	Variable	Write data for request packets. Read data for Read Response packets. Multi-beat bursts pack all beats into the DATA field in packed mode
PAD	Variable	Zero padding to align to data lane boundary. Only required for data lanes > x1

A.6.2. Burst Handling

When `burst_en = 1`, the bridge supports AHB burst transactions via `OPT_FIELD1`. The beat count is implicit from the HBURST encoding — no explicit length field is required.

Table A.8. Supported Burst Types

HBURST	Type	Beats
3'b000	SINGLE	1
3'b010	WRAP4	4
3'b011	INCR4	4
3'b100	WRAP8	8
3'b101	INCR8	8
3'b110	WRAP16	16

HBURST	Type	Beats
3'b111	INCR16	16



HBURST=3'b001 (INCR, undefined length) is not supported. If the local AHB manager issues an INCR burst, the Subordinate Bridge shall respond with HRESP=ERROR.

A.6.2.1. Packed Burst Mode (Default)

For burst writes, the bridge concatenates all beat data into a single BoW.Flexi packet:

When `wstrb_in_data = 1`:

```
Total_Data_Field = BEATS * (DATA_BUS_WIDTH + DATA_BUS_WIDTH/8) bits
```

When `wstrb_in_data = 0`:

```
Total_Data_Field = BEATS * DATA_BUS_WIDTH bits
```

Where BEATS is determined by HBURST (4, 8, or 16 for burst transfers).

This mode minimizes the number of BoW.Flexi packets but requires buffering at the bridge.

A.6.2.2. Per-Beat Mode (Alternative)

For systems with limited buffering, each AHB beat can be sent as a separate BoW.Flexi packet:

- Each beat is a separate BoW.Flexi packet with its own CMD and DATA
- The first beat carries HTRANS=NONSEQ semantics with the full address
- Subsequent beats carry the address incremented by the bridge per HBURST type
- The receiving bridge reconstructs the burst using HBURST and the starting address

The mode selection is implementation-specific and may be configured via implementation-defined registers.

A.6.3. Locked Transfers

When `lock_en = 1`, locked transfers are supported via HMASTLOCK in OPT_FIELD2[0]:

```
OPT_FIELD2: HMASTLOCK=1 → bit[0]=1
```

The receiving bridge:

1. Asserts HMASTLOCK on the remote AHB bus
2. Drives the transaction(s)
3. De-asserts HMASTLOCK when the next non-locked transfer arrives



Locking is independent of bursts — OPT_FIELD2 can be enabled with or without OPT_FIELD1 (bursts), so a locked single transfer is expressible in minimum-burst configurations. Enforcing the atomic locked sequence (no interleaved traffic between locked

beats) is owned by the System Layer; the BoW.Flexi link only transports the HMASTLOCK hint.

A.6.4. Packet Size Calculation

For a given data lane (DL) in bits, the packet size in transfer cycles is:

A.6.4.1. Default (No Optional Fields)

Request Packets (CMD=000, 001, 100):

```
Packet_Bits = CMD(3) + SIZE(3) + ADDR(ADDR_WIDTH) + DATA(variable, 0 for reads)
Padding_Bits = (DL - (Packet_Bits mod DL)) mod DL
Transfer_Cycles = (Packet_Bits + Padding_Bits) / DL
```

Response Packets (CMD=010, 011):

```
Read Response (CMD=010):
  Packet_Bits = CMD(3) + DATA(variable)

Write Response (CMD=011):
  Packet_Bits = CMD(3)
```

A.6.4.2. Configured (HPROT + Burst + Lock Enabled)

Write Request Packets (CMD=000 or CMD=100, Single Beat, wstrb_in_data = 0):

```
Packet_Bits = CMD(3) + OPT_FIELD0(4+N) + OPT_FIELD1(3) + OPT_FIELD2(1) + SIZE(3) +
ADDR(ADDR_WIDTH) + DATA(DATA_BUS_WIDTH)
```

Read Request Packets (CMD=001):

```
Packet_Bits = CMD(3) + OPT_FIELD0(4+N) + OPT_FIELD1(3) + OPT_FIELD2(1) + SIZE(3) +
ADDR(ADDR_WIDTH)
```

A.6.4.3. Configuration/Error Packets

```
Packet_Bits = CMD(3) + LEN(2) + WR(1) + RD_RESP(1) + LOCAL_ADDR(8) + DATA(8/16/32)
Padding_Bits = (DL - (Packet_Bits mod DL)) mod DL
Transfer_Cycles = (Packet_Bits + Padding_Bits) / DL
```

A.6.5. AHB Packet Timing Summary

A.6.5.1. Default (No Optional Fields, 32-bit bus, 32-bit address)

Table A.9. AHB Default Packet Timing Summary (x8 data lane)

Transaction Type	Packet Bits	Padding Bits (x8)	Total Bits (x8)	Clock Cycles (x8)
Posted Write (8-bit)	46	2	48	6
Posted Write (16-bit)	54	2	56	7

Transaction Type	Packet Bits	Padding Bits (x8)	Total Bits (x8)	Clock Cycles (x8)
Posted Write (32-bit)	70	2	72	9
Posted Write (64-bit)	102	2	104	13
Non-Posted Write (32-bit) Request	70	2	72	9
Write Response	3	5	8	1
Read Request (any size)	38	2	40	5
Read Response (8-bit)	11	5	16	2
Read Response (16-bit)	19	5	24	3
Read Response (32-bit)	35	5	40	5
Read Response (64-bit)	67	5	72	9
Read Response (128-bit)	131	5	136	17
Config/Error Write (8-bit)	23	1	24	3
Config/Error Write (16-bit)	31	1	32	4
Config/Error Write (32-bit)	47	1	48	6
Config/Error Read Request	15	1	16	2
Config/Error Read Response (8-bit)	23	1	24	3
Config/Error Read Response (32-bit)	47	1	48	6

A.6.5.2. Configured (HPROT, burst enabled, 32-bit address)

Table A.10. AHB Configured Packet Timing Summary (x8 data lane)

Transaction Type	Packet Bits	Padding Bits (x8)	Total Bits (x8)	Clock Cycles (x8)
Non-Posted Write (single; 32-bit bus)	78	2	80	10
Non-Posted Write (single; 64-bit bus)	110	2	112	14
Posted Write (single; 32-bit bus)	78	2	80	10
Posted Write (single; 64-bit bus)	110	2	112	14
Read Request (32-bit address)	46	2	48	6

Transaction Type	Packet Bits	Padding Bits (x8)	Total Bits (x8)	Clock Cycles (x8)
Read Request (64-bit address)	78	2	80	10
Write Response	3	5	8	1
Read Response (32-bit bus)	35	5	40	5
Read Response (64-bit bus)	67	5	72	9
Burst Write INCR4 (4x32-bit; no WSTRB)	174	2	176	22
Burst Write INCR4 (4x32-bit; with WSTRB)	190	2	192	24
Config/Error Write (32-bit)	47	1	48	6
Config/Error Read Request	15	1	16	2

A.7. Error Handling and Reporting

A.7.1. AHB Response Codes

The AHB protocol defines two response values:

- **OKAY (HRESP=0):** Transfer completed successfully
- **ERROR (HRESP=1):** Transfer error (subordinate signals HRESP=1)

The AHB bridge uses the BoW.Flexi Configuration/Error Packet (CMD=111) mechanism for error reporting at all configuration levels:

- The **presence** of a response packet (CMD=010 Read Response or CMD=011 Write Response) implies HRESP=OKAY
- When the remote subordinate signals HRESP=ERROR, the remote bridge sends a Configuration/Error Packet (CMD=111) with the appropriate error code instead of a normal response
- The local bridge decodes the error packet and drives HRESP=ERROR on the local AHB bus

This approach avoids adding 1 bit of overhead to every response packet for an infrequent error condition.

A.7.2. Error Status Register (Address 0x10)

The AHB bridge uses the standard BoW.Flexi error register layout (see [Remote Error Register](#) and [Local Error Register](#)). The base protocol error fields occupy bits [11:0]. The AHB bridge defines the implementation-dependent bits [31:12] as follows:

Table A.11. BoW.Flexi AHB Error Register Extension (bits [31:12])

Bits	Identifier	Access	Reset	Description
31:24	ahb_error_count	W1C	0x00	Rolling error counter (saturates at 0xFF).

Bits	Identifier	Access	Reset	Description
				Increments on each new error event. Write 1 to any bit in this field to clear the counter to zero.
23	ahb_error_posted	R	0	1 = Most recent AHB error occurred during a posted write transaction. Updated on every error event.
22:20	ahb_error_hsize	R	0x0	HSIZE[2:0] of the AHB transaction that caused the most recent error. Updated on every error event.
19:14	Reserved			Reserved for future AHB-specific use
13	ahb_decode_error	W1C	0	1 = Address decode error; no AHB subordinate responded at the target address. Write 1 to clear
12	ahb_subordinate_error	W1C	0	1 = AHB subordinate returned HRESP=ERROR. Write 1 to clear



When an error is detected (including AHB HRESP=ERROR), the bridge sends a Configuration/Error Packet with **WR**=1, **LOCAL_ADDR**=0x10, and the 32-bit error status value.

A.7.3. Error Reporting Flow

When an error is detected, the bridge follows this procedure:

1. **Capture Error Context:**
 - Set appropriate error bit (base bits [5:0] or AHB-specific bits [13:12])
 - Capture command in **error_cmd** [8:6] (updated on every error report)
 - Capture SIZE in **error_size** [11:9] (updated on every error report)
 - Capture HSIZE in **ahb_error_hsize** [22:20] (updated on every error report)
 - Increment **error_count** [31:24] (saturates at 255, W1C to clear)
2. **Send Error Packet:**
 - Create Configuration/Error Packet with CMD=111
 - Set **WR**=1, **RD_RESP**=0
 - Set **LOCAL_ADDR**=0x10
 - Set **LEN**=10 (32-bit data)
 - Set **DATA**=ERROR_STATUS register value
3. **Handle Transaction:**
 - Do NOT send response packet (CMD=010 or CMD=011)
 - Send Error Packet (CMD=111) instead

- Discard invalid packet
- Update local STATUS register



When an error occurs during an AHB transaction, the bridge does NOT send a normal response packet. Instead, it sends an Error Packet (CMD=111) to report the error condition. The requesting bridge will timeout waiting for the response and can correlate this with the received Error Packet.

A.7.4. Link-Level Errors

For errors detected at the BoW.Flexi link level (e.g., ECC errors, framing errors), the standard BoW.Flexi error reporting mechanism is used via the Configuration/Error Packet (CMD=111) and the error registers ([Remote Error Register](#)).

A.8. Transaction Examples

A.8.1. Posted Write Transaction (Minimum Mode)

A Posted Write receives a Write Response from the target as a flow-control acknowledgment that the receiver has accepted the write and is ready for a further posted write; unlike a Non-Posted Write, this response need not wait for the write to complete downstream. This example shows minimum configuration (no optional fields).

Example: Posted Write of 32-bit data (0xDEADBEEF) to address 0x10000000 (x8 data lane)

Total: 3 (CMD) + 3 (SIZE) + 32 (ADDR) + 32 (DATA) = 70 bits
 Padding: $(8 - (70 \bmod 8)) \bmod 8 = 2$ bits
 Total with padding: 72 bits = 9 bytes

Bit-level packet construction:

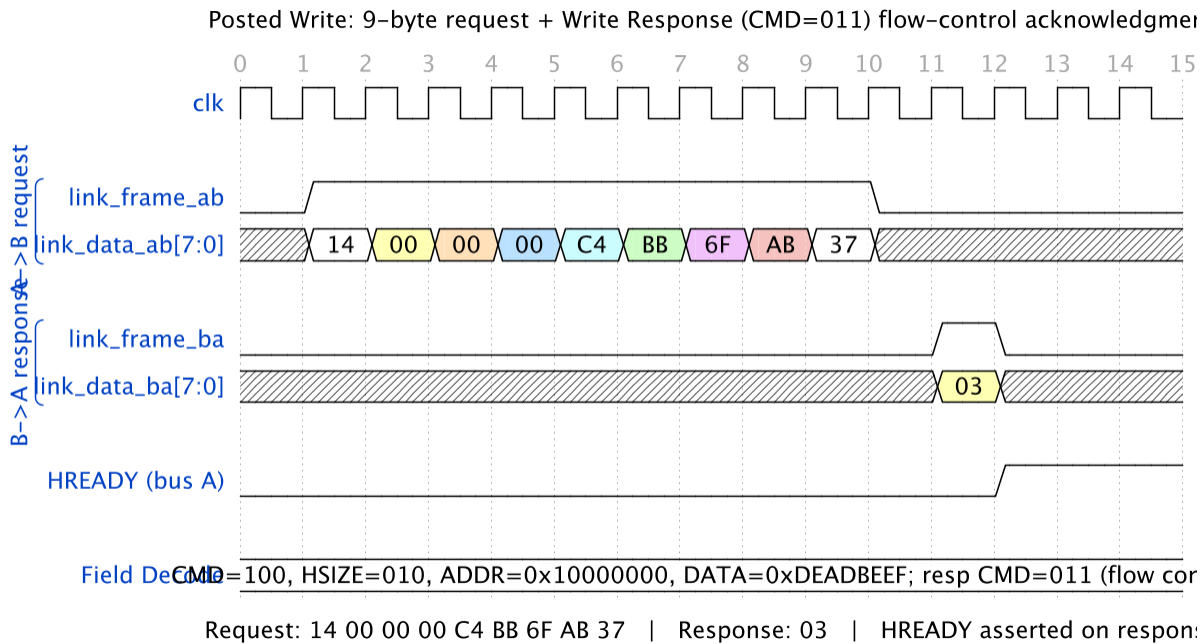
Bits [2:0] = 100 (CMD: Posted Write)
 Bits [5:3] = 010 (SIZE: 32-bit)
 Bits [37:6] = 0x10000000 (address)
 Bits [69:38] = 0xDEADBEEF (data)
 Bits [71:70] = 00 (padding)

Complete hex packet: 14 00 00 00 C4 BB 6F AB 37

Table A.12. Receiver Decode Verification — Posted Write

Field	Packet Bits	Decoded Value	Sent Value	Match
CMD	[2:0] = 3'b100	Posted Write	Posted Write	Yes
SIZE	[5:3] = 3'b010	32-bit	32-bit	Yes
ADDR	[37:6]	0x10000000	0x10000000	Yes
DATA	[69:38]	0xDEADBEEF	0xDEADBEEF	Yes

Field	Packet Bits	Decoded Value	Sent Value	Match
PAD	[71:70] = 2'b00	(alignment)	(alignment)	Yes

Figure A.2. Posted Write Timing

A.8.2. Non-Posted Write Transaction (Minimum Mode)

Non-Posted Writes require a write response from the target.

Example: Non-Posted Write of 32-bit data (0xABCD1234) to address 0x20000004 (x8 data lane)

Request: 3 (CMD) + 3 (SIZE) + 32 (ADDR) + 32 (DATA) = 70 bits
 Padding: 2 bits → Total: 72 bits = 9 bytes

Response: 3 (CMD) = 3 bits
 Padding: 5 bits → Total: 8 bits = 1 byte

Request packet:

Bits [2:0] = 000 (CMD: Non-Posted Write)
 Bits [5:3] = 010 (SIZE: 32-bit)
 Bits [37:6] = 0x20000004 (address)
 Bits [69:38] = 0xABCD1234 (data)
 Bits [71:70] = 00 (padding)

Complete hex packet: 10 01 00 00 08 8D 44 F3 2A

Response packet (OKAY — receipt of response implies success):

Byte 0: 0x03 = CMD=011 (Write Response) + 5 bits padding

Complete hex packet: 03

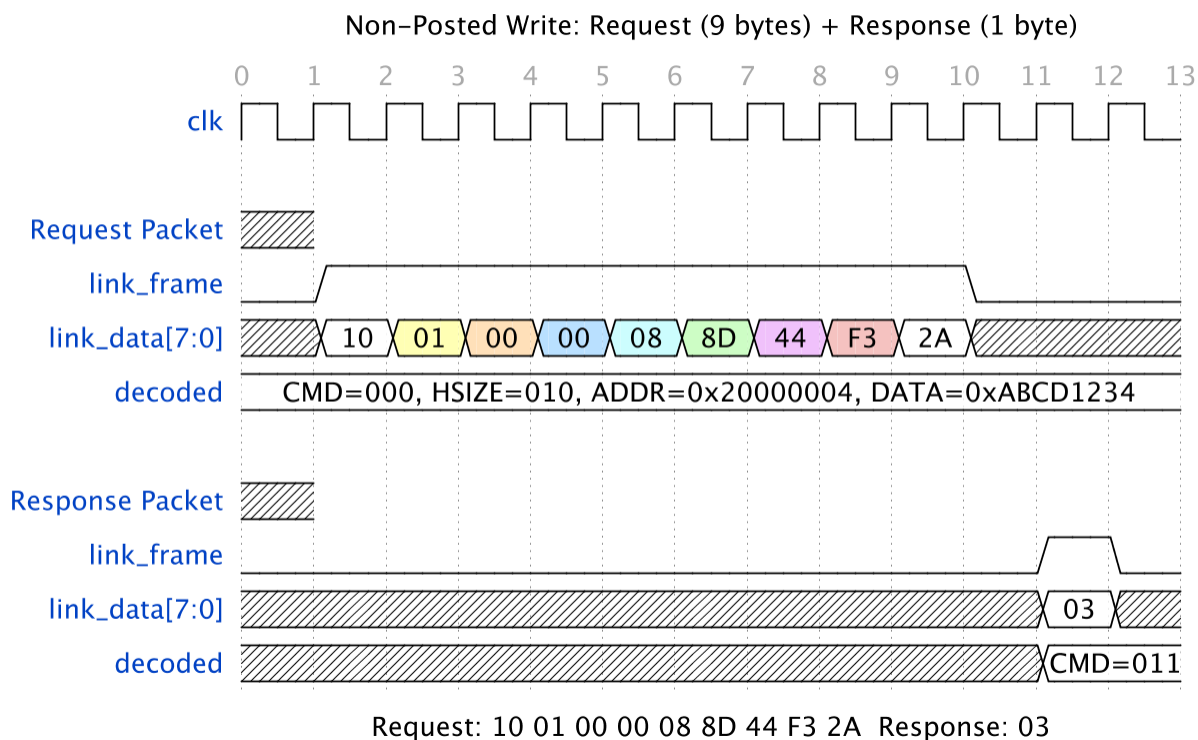
Table A.13. Receiver Decode Verification — Non-Posted Write Request

Field	Packet Bits	Decoded Value	Sent Value	Match
CMD	[2:0] = 3'b000	Non-Posted Write	Non-Posted Write	Yes
SIZE	[5:3] = 3'b010	32-bit	32-bit	Yes
ADDR	[37:6]	0x20000004	0x20000004	Yes
DATA	[69:38]	0xABCD1234	0xABCD1234	Yes
PAD	[71:70] = 2'b00	(alignment)	(alignment)	Yes

Table A.14. Receiver Decode Verification — Write Response

Field	Packet Bits	Decoded Value	Sent Value	Match
CMD	[2:0] = 3'b011	Write Response (OKAY)	Write Response (OKAY)	Yes
PAD	[7:3] = 5'b00000	(alignment)	(alignment)	Yes

Figure A.3. Non-Posted Write Timing



A.8.3. Read Transaction (Minimum Mode)

Example: Read 32-bit data from address 0x30000008 (x8 data lane)

Request: 3 (CMD) + 3 (SIZE) + 32 (ADDR) = 38 bits
 Padding: 2 bits → Total: 40 bits = 5 bytes

Response: 3 (CMD) + 32 (DATA) = 35 bits
 Padding: 5 bits → Total: 40 bits = 5 bytes

Request packet:

Bits [2:0] = 001 (CMD: Read)
 Bits [5:3] = 010 (SIZE: 32-bit)
 Bits [37:6] = 0x30000008 (address)
 Bits [39:38] = 00 (padding)

Complete hex packet: 11 02 00 00 0C

Response packet (returning 0x12345678):

Bits [2:0] = 010 (CMD: Read Response)
 Bits [34:3] = 0x12345678 (data)
 Bits [39:35] = 00000 (padding)

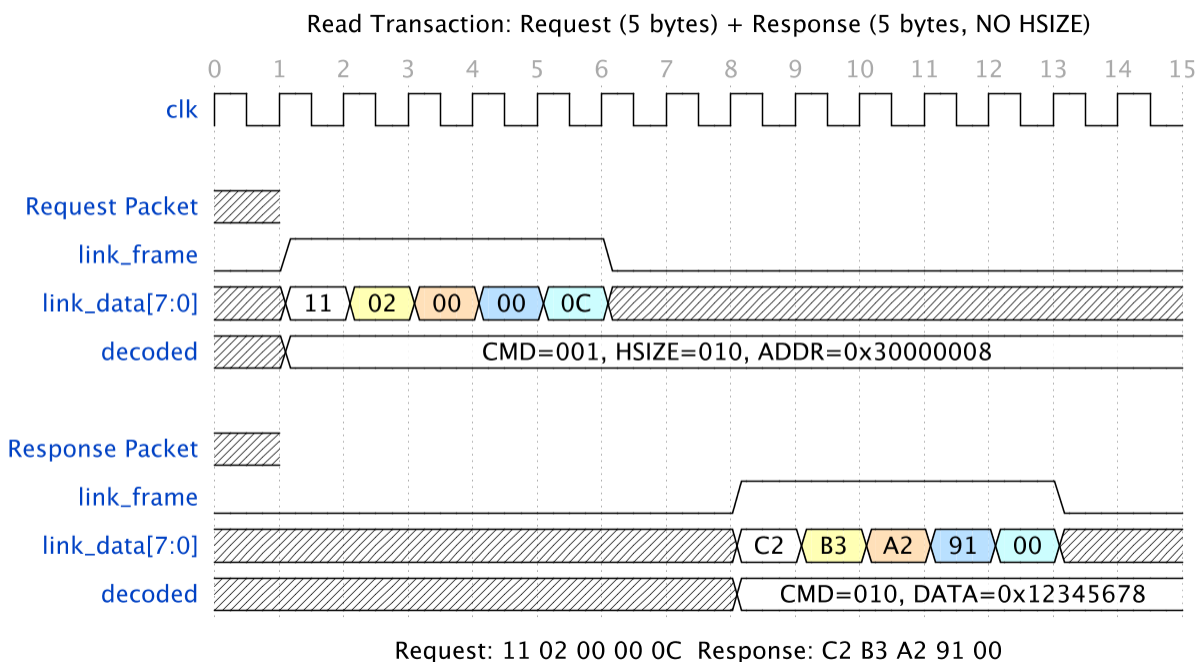
Complete hex packet: C2 B3 A2 91 00

Table A.15. Receiver Decode Verification — Read Request

Field	Packet Bits	Decoded Value	Sent Value	Match
CMD	[2:0] = 3'b001	Read Request	Read Request	Yes
SIZE	[5:3] = 3'b010	32-bit	32-bit	Yes
ADDR	[37:6]	0x30000008	0x30000008	Yes
PAD	[39:38] = 2'b00	(alignment)	(alignment)	Yes

Table A.16. Receiver Decode Verification — Read Response

Field	Packet Bits	Decoded Value	Sent Value	Match
CMD	[2:0] = 3'b010	Read Response	Read Response	Yes
DATA	[34:3]	0x12345678	0x12345678	Yes
PAD	[39:35] = 5'b00000	(alignment)	(alignment)	Yes

Figure A.4. Read Transaction Timing

A.8.4. Non-Posted Write Transaction (Full AHB Mode)

This example shows a single-beat write with HPROT enabled.

Example: Non-Posted Write of 32-bit data (0xDEADBEEF) to address 0x40000000 (x8 data lane, HPROT=0011)

Configuration:

```
OPT_FIELD0 = 4 bits: HPROT = 0011 (non-cacheable, non-bufferable, privileged, data)
OPT_FIELD1 = 3 bits: HBURST(000) = 000
OPT_FIELD2 = 1 bit: HMASTLOCK(0) = 0
SIZE = 3 bits: 010 (4 bytes)
ADDR = 32 bits
DATA = 32 bits
```

Request (per payload order: OPT_FIELD0, OPT_FIELD1, OPT_FIELD2, SIZE, ADDR, DATA):

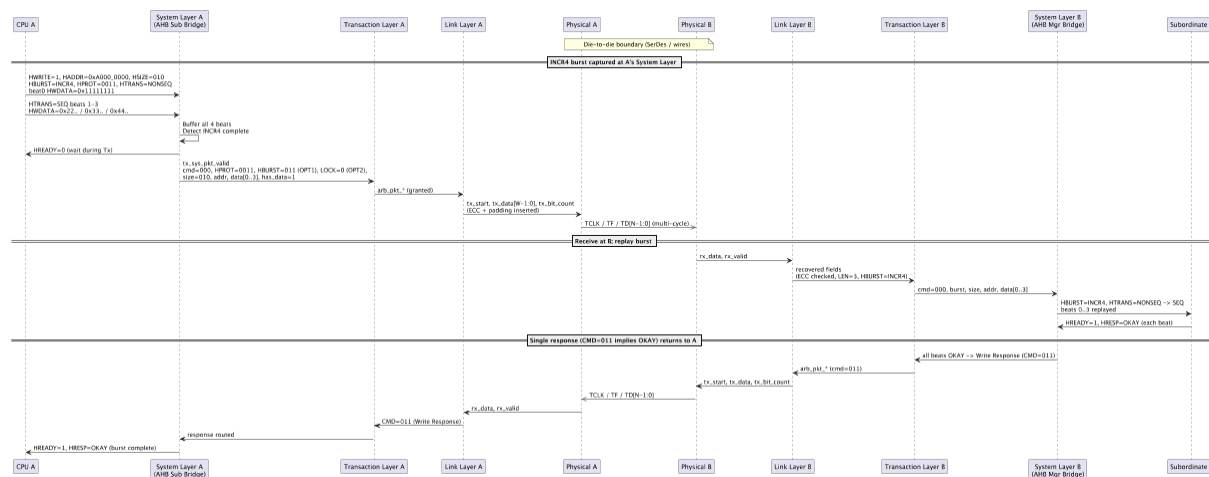
Total: 3 + 4 + 3 + 1 + 3 + 32 + 32 = 78 bits

Padding (x8): 2 bits

Clock Cycles (x8): 80 / 8 = 10

A.8.5. Burst Write Transaction (INCR4, Full AHB Mode)

This example shows an INCR4 burst write of 4 * 32-bit words using packed burst mode.

Figure A.5. AHB Burst Write Sequence

Request (packed burst — single packet with concatenated data):

```

CMD:      3 bits   = 000 (Non-Posted Write)
OPT_FIELD0: 4 bits = 0011 (HPROT=0011)
OPT_FIELD1: 3 bits = 011 (HBURST=INCR4)
OPT_FIELD2: 1 bit  = 0 (HMASTLOCK=0)
SIZE:     3 bits   = 010 (4 bytes per beat)
ADDR:     32 bits  = 0xA0000000
DATA:     128 bits = [beat0][beat1][beat2][beat3] (4 * 32-bit words)

```

Total: 3 + 4 + 3 + 1 + 3 + 32 + 128 = 174 bits

Padding (x8): 2 bits

Clock Cycles (x8): 176 / 8 = 22

Response:

CMD: 3 bits = 011 (Write Response) → 8 bits with padding = 1 byte



The response implies HRESP=OKAY for all beats.

A.8.6. Configuration/Error Packet

Configuration and Error Packets use command 3'b111 with the following format:

```

Bits [2:0]:  CMD (111)
Bits [4:3]:  LEN (2 bits: 00=8bit, 01=16bit, 10=32bit)
Bits [5]:    WR (1=write, 0=read)
Bits [6]:    RD_RESP (1=Read Response)
Bits [14:7]: LOCAL_ADDR (8 bits)
Bits [14+D:15]: DATA (8/16/32 bits based on LEN, 0 for read requests)
Bits [end]:  PAD (padding to align to data lane boundary)

```

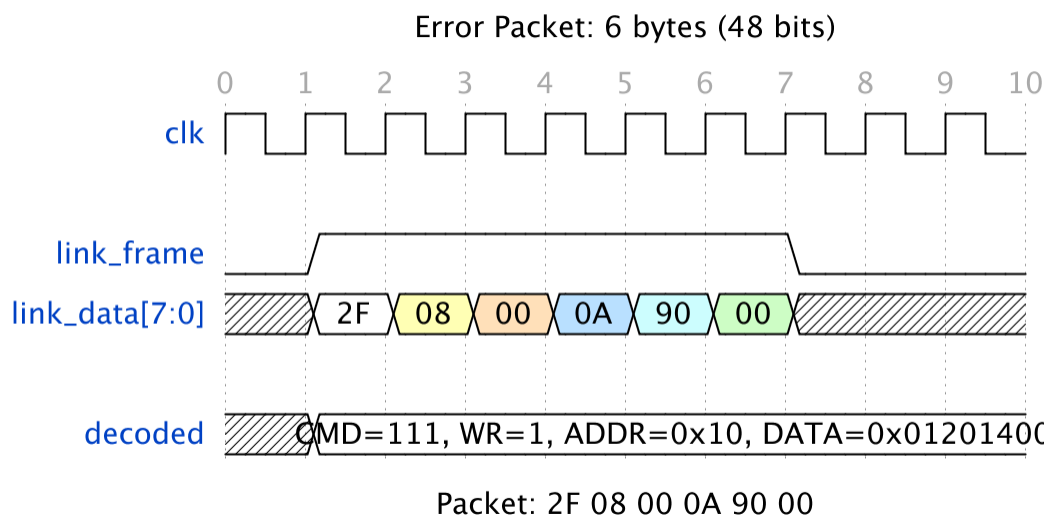
Example: Error Report — AHB Subordinate Error with 32-bit status (x8 data lane)

Total: 3 (CMD) + 2 (LEN) + 1 (WR) + 1 (RD_RESP) + 8 (LOCAL_ADDR) + 32 (DATA) = 47 bits

Padding: 1 bit → Total: 48 bits = 6 bytes

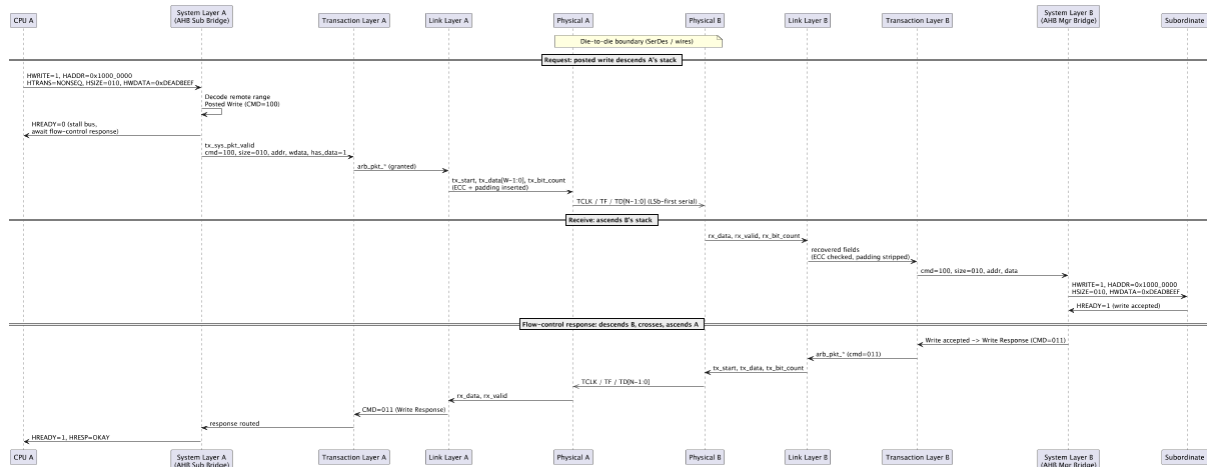
Table A.17. Receiver Decode Verification — Configuration/Error Packet

Field	Packet Bits	Decoded Value	Sent Value	Match
CMD	[2:0] = 3'b111	Config/Error	Config/Error	Yes
LEN	[4:3] = 2'b10	32-bit data	32-bit data	Yes
WR	[5] = 1'b1	Write	Write	Yes
RD_RESP	[6] = 1'b0	Not Read Response	Not Read Response	Yes
LOCAL_ADDR	[14:7]	0x10	0x10	Yes
DATA	[46:15]	0x01201440	0x01201440	Yes
PAD	[47] = 1'b0	(alignment)	(alignment)	Yes

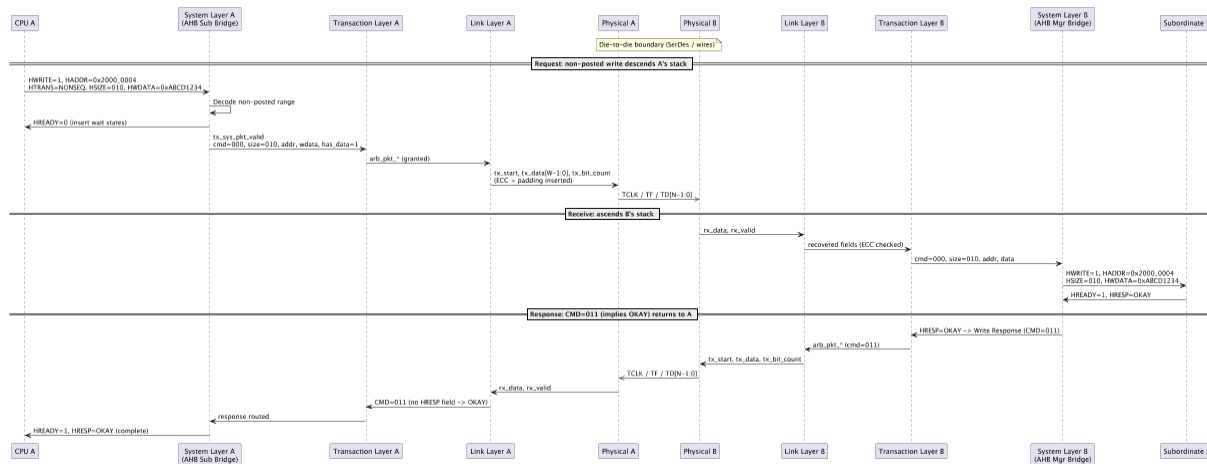
Figure A.6. Error Packet Timing

A.9. Transaction Flows

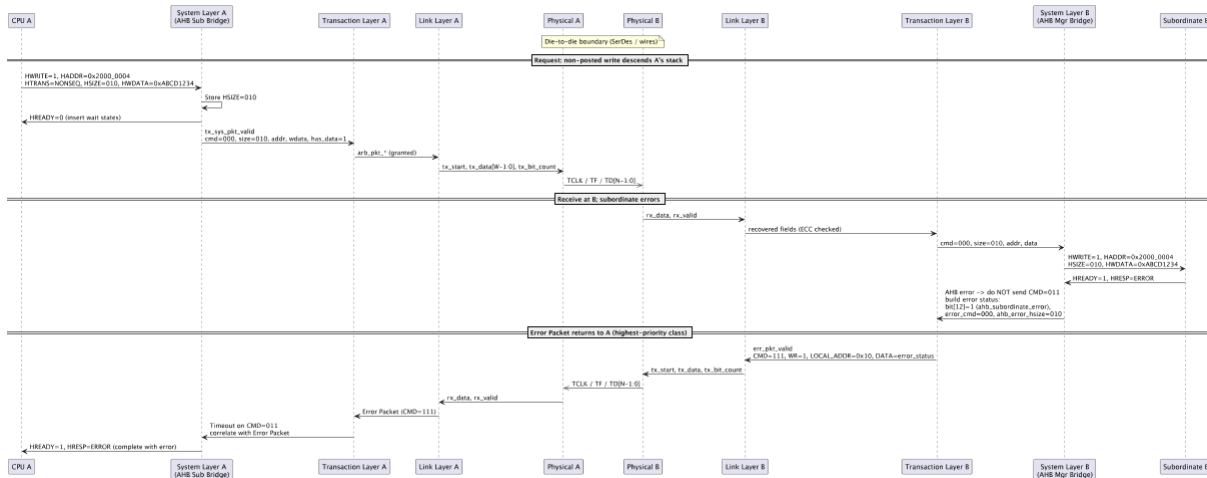
A.9.1. Posted Write (Success)



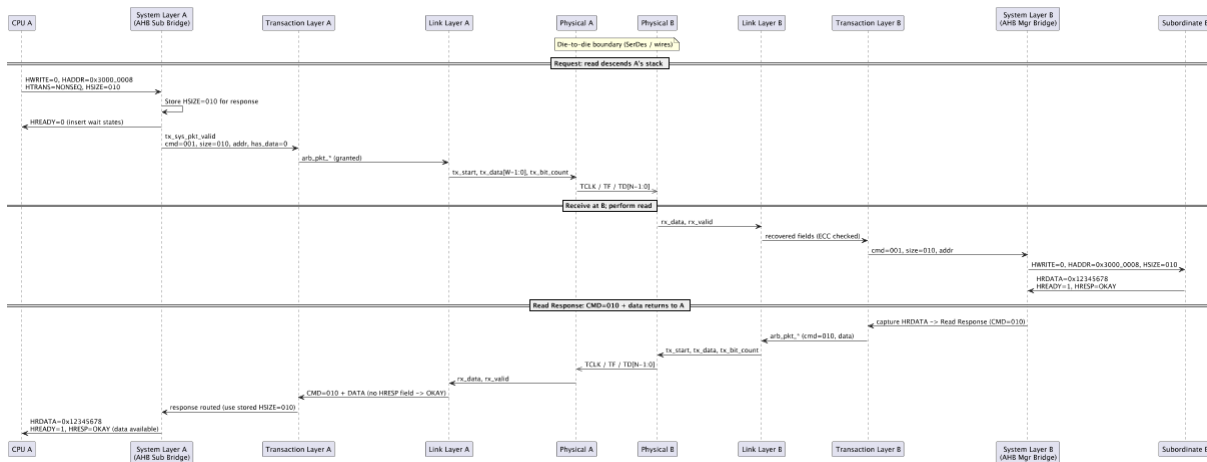
A.9.2. Non-Posted Write (Success)



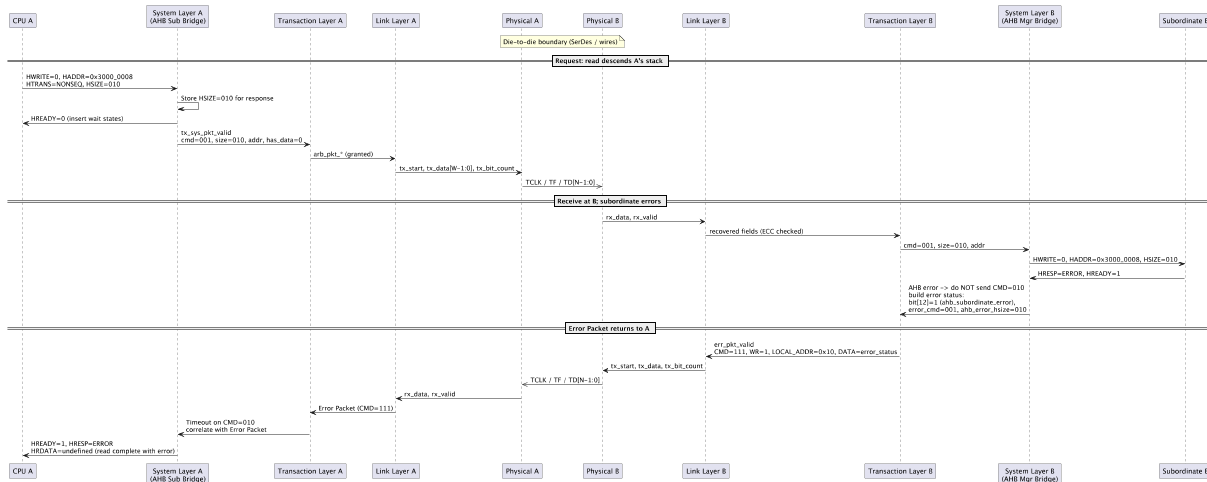
A.9.3. Non-Posted Write with AHB Error



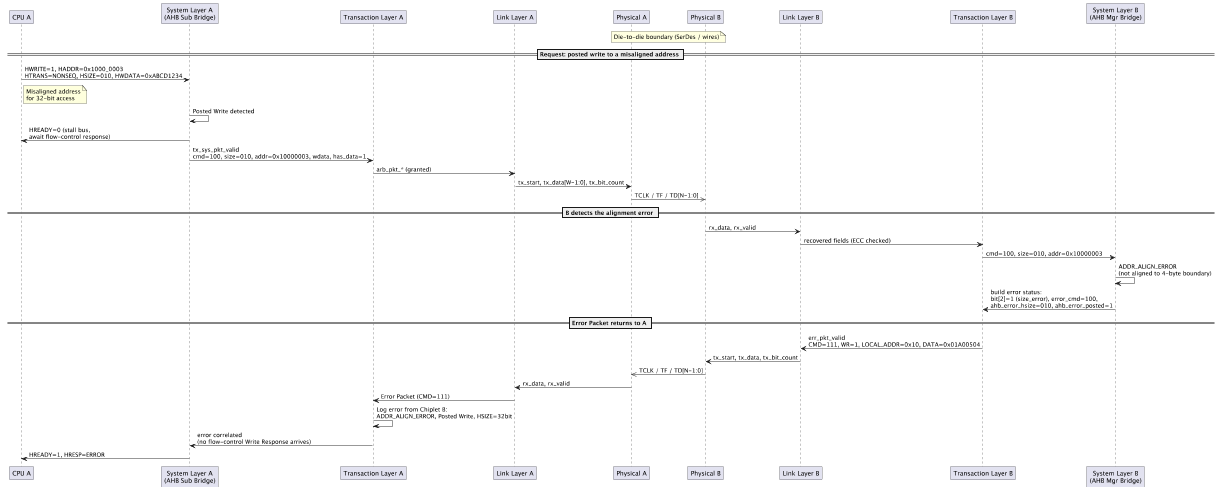
A.9.4. Read Transaction (Success)



A.9.5. Read Transaction with AHB Error

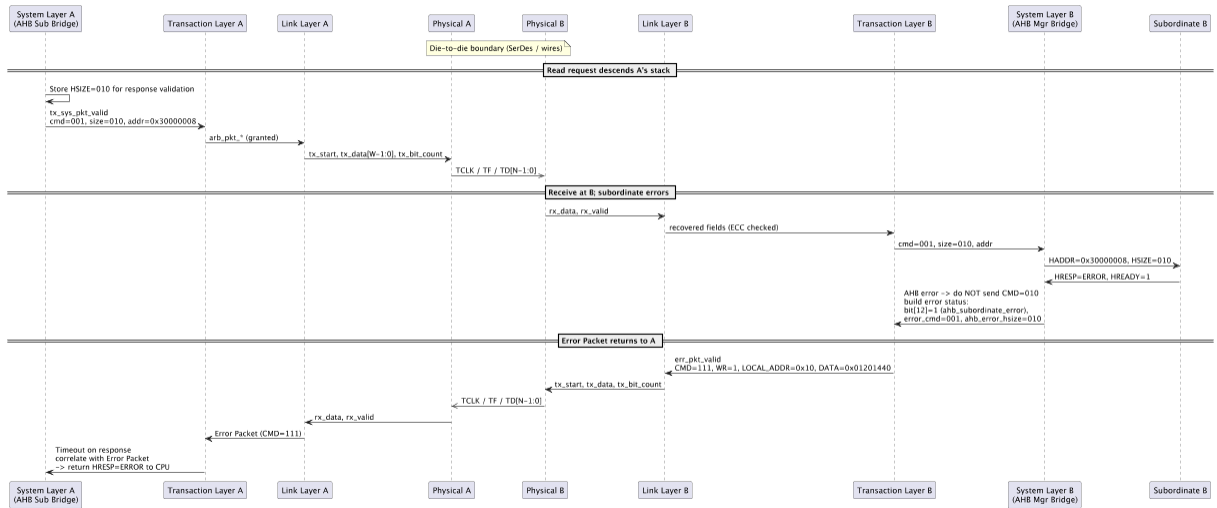


A.9.6. Error During Posted Write



Example Error Scenario: AHB Subordinate Returns Error

Chiplet A sends a read request. Chiplet B's AHB subordinate returns HRESP=ERROR.



Error Status Value (0x01201440):

```

Bits [31:24]: 00000001 - error_count = 1
Bit [23]: 0 - ahb_error_posted = 0 (not posted)
Bits [22:20]: 010 - ahb_error_hsize = 010 (32-bit)
Bits [19:13]: 00000000 - reserved
Bit [12]: 1 - ahb_subordinate_error
Bits [11:9]: 010 - error_size = 010 (4 bytes)
Bits [8:6]: 001 - error_cmd = 001 (read request)
Bits [5:0]: 000000 - no frame/timeout/ecc/overflow errors
  
```

A.10. Transaction Ordering

AHB requires strict in-order completion — responses shall be returned in the same order as requests. The bridge shall enforce this by:

1. Not issuing a new AHB transaction on the remote bus until the previous one completes (HREADY=1)
2. Forwarding response packets back to the requesting bridge in the same order they were received
3. Ensuring locked sequences are not broken by interleaved transactions



AHB does not support multiple outstanding transactions or out-of-order completion. Each AHB manager can have at most one pending transaction at a time.

A.11. Configuration Sequence

The following sequence configures the BoW.Flexi interface for AHB bridge operation:

1. Initialize the BoW.Flexi link in default x1 mode
2. Configure data lanes as required via Configuration Command
3. Set `protocol_mode = 1` (AHB) in [Interface Configuration Extension Register 1](#). With no optional fields enabled the bridge operates at minimum overhead.
4. To enable optional fields (protection, burst, lock, user), set the Optional Field Configuration Register (offset 0x3C):
 - `opt_field0_size = 4 + user_width` (e.g., 4 with no USER)
 - `opt_field0_cmd_support = 7'b0010011` (Non-Posted Write, Read, Posted Write)
 - `opt_field1_size = 3`
 - `opt_field1_cmd_support = 7'b0010011` (Non-Posted Write, Read, Posted Write)
5. And set the Optional Field Extended Configuration Register (offset 0x40):
 - `opt_field2_size = 1`
 - `opt_field2_cmd_support = 7'b0010011` (Non-Posted Write, Read, Posted Write — when `lock_en = 1`; `7'b0000000` otherwise)
 - `opt_field3_cmd_support = 7'b0000000` (not used — errors via CMD=111)
6. Set the AMBA Configuration Register (offset 0x70):
 - `bridge_en = 1`
 - Configure `burst_en`, `prot_en`, `lock_en` as needed
 - Configure `data_width` as needed
7. Set `address_width` in the Configuration Command to match the AHB address bus width (default 0x1F = 32 bits)

A.12. Optional Field Configuration Register Settings

The table below summarizes the recommended Optional Field Configuration Register and Optional Field Extended Configuration Register settings for common AHB configurations:

Table A.18. Recommended Optional Field Register Configuration for AHB

Configuration	opt_field0_size	opt_field0_cmd_sup_port	opt_field1_size	opt_field1_cmd_sup_port	opt_field2_size	opt_field2_cmd_sup_port	opt_field3_size	opt_field3_cmd_sup_port
AHB Write Request (Non-Posted)	4+N bits (HPROT+USER)	bit[0]=1	3 bits (HBURST)	bit[0]=1	1 bit (HMASTLOCK)	bit[0]=1	not used	bit[0]=0
AHB Write Request (Posted)	4+N bits (HPROT+USER)	bit[4]=1	3 bits (HBURST)	bit[4]=1	1 bit (HMASTLOCK)	bit[4]=1	not used	bit[4]=0
AHB Read Request	4+N bits (HPROT+USER)	bit[1]=1	3 bits (HBURST)	bit[1]=1	1 bit (HMASTLOCK)	bit[1]=1	not used	bit[1]=0
AHB Read Response	not present	bit[2]=0	not present	bit[2]=0	not used	bit[2]=0	not used	bit[2]=0
AHB Write Response	not present	bit[3]=0	not present	bit[3]=0	not used	bit[3]=0	not used	bit[3]=0

A.13. Reduced-Overhead Configurations

The AHB bridge provides a single implementation that can be configured from minimum to maximum functionality:

A.13.1. Default (Minimum Overhead)

For the simplest configuration (the default when no optional fields are enabled):

- Disable OPT_FIELD0 (prot_en = 0)
- Disable OPT_FIELD1 (burst_en = 0)
- Disable WSTRB (wstrb_in_data = 0)

Packets contain only CMD + SIZE + ADDR + DATA — the lowest overhead configuration.

A.13.2. Configured (Optional Fields Enabled)

All features are individually controlled via the AMBA Configuration Register. Examples:

A.13.2.1. Protection Only

- Enable OPT_FIELD0 with HPROT only (`prot_en = 1, user_width = 0`)
- Disable OPT_FIELD1 (`burst_en = 0`)

Adds 4 bits of overhead per request for HPROT.

A.13.2.2. Burst Without HPROT

For DMA-style systems where protection attributes are not needed:

- Disable OPT_FIELD0 (`prot_en = 0`)
- Enable OPT_FIELD1 (`burst_en = 1`)

A.13.2.3. All Features Enabled

All features enabled:

- OPT_FIELD0: HPROT + optional USER
- OPT_FIELD1: HBURST
- OPT_FIELD2: HMASTLOCK
- Optional: WSTRB in DATA field

Appendix B. Example: APB to Chiplet Bridge

B.1. Overview

This appendix describes an example of the bridge protocol between two chiplet devices where one chiplet accesses APB (AMBA APB4/APB5) peripherals on the partner chiplet. The bridge converts APB transactions into the BoW.Flexi serialized packet format for transmission across chiplet boundaries.

APB is the simplest of the AMBA protocols, designed for low-bandwidth peripheral access with a two-cycle (setup + access) handshake. The BoW.Flexi APB bridge provides:

- Single-beat read and write transfers only (no bursts)
- Optional PPROT (protection) via OPT_FIELD0 when enabled
- Error reporting via Configuration/Error Packet (CMD=111) mechanism
- Optional PSTRB (byte strobes) embedded in DATA field
- Minimal packet overhead for low-latency peripheral access

The default APB configuration transmits no optional fields, providing the lowest overhead. PPROT and PSTRB can be individually enabled via the AMBA Configuration Register when required.

B.2. Physical Interface

The APB bridge operates over the standard BoW.Flexi physical layer (see [Section 7.5](#)). Each chiplet provides two uni-directional links, identical to the other bridge examples.

B.3. APB Interface Signals

B.3.1. APB Subordinate Bridge Interface

The Subordinate Bridge acts as an APB completer on the local bus, accepting transactions from the local APB requester and converting them to BoW.Flexi packets.

Table B.1. APB Subordinate Bridge Signals

Signal	Width	Direction	Description
PCLK	1	Input	APB clock
PRESETn	1	Input	APB reset (active low)
PSEL	1	Input	Peripheral select
PADDR	Variable	Input	Address bus (configurable; 1-64 bits via address_width)

Signal	Width	Direction	Description
PENABLE	1	Input	Access enable (second cycle of transfer)
PWRITE	1	Input	Write enable (1=write; 0=read)
PWDATA	Variable	Input	Write data bus (8/16/32 bit)
PSTRB	Variable	Input	Write byte strobes (1 bit per byte of PWDATA)
PPROT[2:0]	3	Input	Protection type
PREADY	1	Output	Transfer ready (can extend transfer)
PRDATA	Variable	Output	Read data bus (same width as PWDATA)
PSLVERR	1	Output	Transfer error (0=OKAY; 1=ERROR)

B.3.2. APB Manager Bridge Interface

The Manager Bridge acts as an APB requester on the remote bus, receiving BoW.Flexi packets and initiating APB transactions to the target peripheral.

Table B.2. APB Manager Bridge Signals

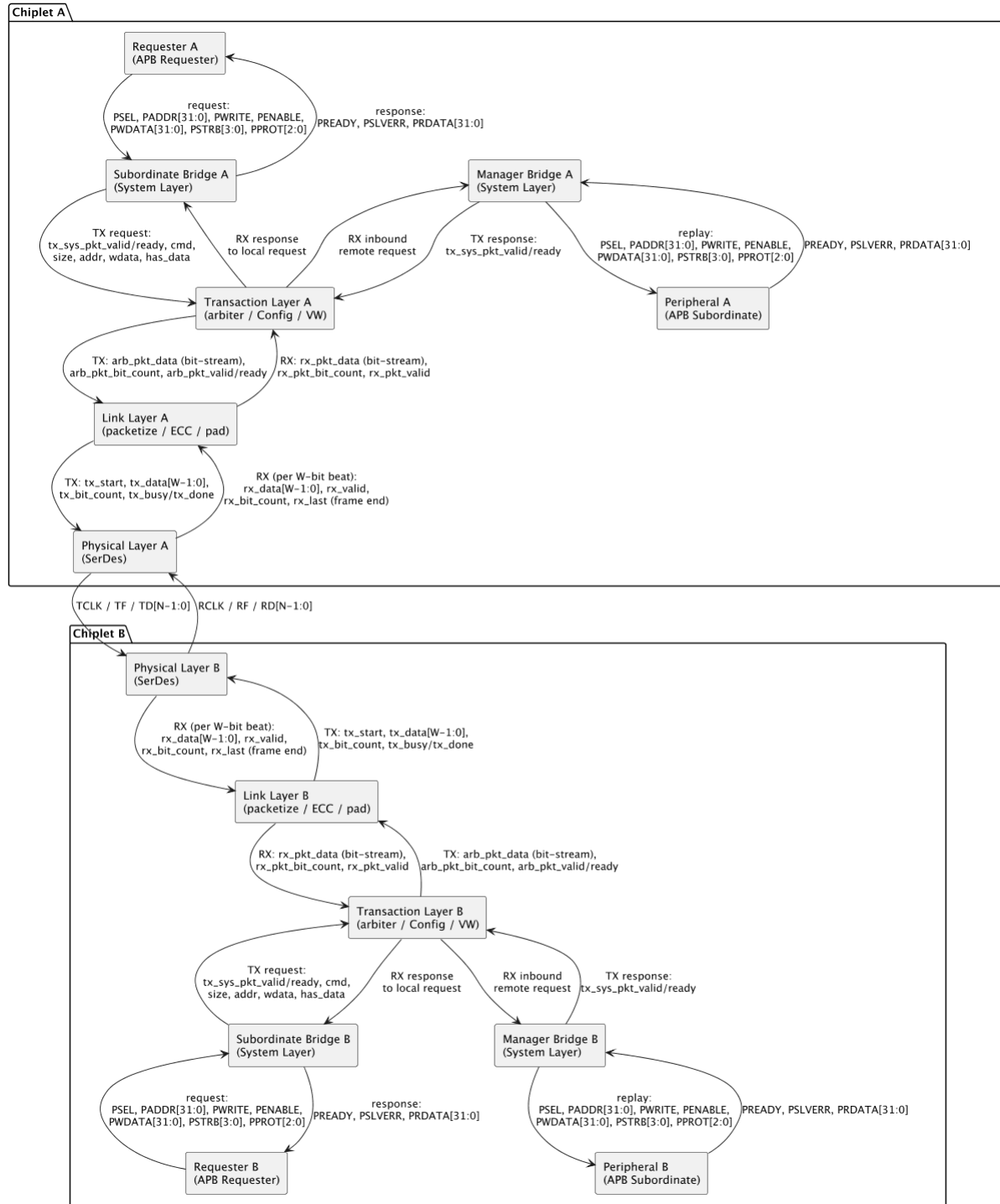
Signal	Width	Direction	Description
PCLK	1	Input	APB clock
PRESETn	1	Input	APB reset (active low)
PSEL	1	Output	Peripheral select
PADDR	Variable	Output	Address bus (configurable; 1-64 bits via address_width)
PENABLE	1	Output	Access enable (second cycle of transfer)
PWRITE	1	Output	Write enable (1=write; 0=read)
PWDATA	Variable	Output	Write data bus (8/16/32 bit)
PSTRB	Variable	Output	Write byte strobes (1 bit per byte of PWDATA)
PPROT[2:0]	3	Output	Protection type
PREADY	1	Input	Transfer ready from peripheral
PRDATA	Variable	Input	Read data bus (same width as PWDATA)
PSLVERR	1	Input	Transfer error (0=OKAY; 1=ERROR)



The address bus width (PADDR) is configurable from 1 to 64 bits via `address_width` in the Configuration Command. Default is 32-bit. The data bus width (PWDATA/PRDATA) is configurable via `data_width` in the AMBA Configuration Register. Default is 32-bit.

B.3.3. Complete APB Bridge System

Figure B.1. Complete APB Chiplet Bridge System



B.4. Optional Field Usage for APB

The APB bridge uses a minimal optional field configuration, reflecting the simple nature of the APB protocol:

- **OPT_FIELD0** (fixed 3+N bits, requests only): Carries PPROT[2:0] and optional USER[N-1:0] defined field when enabled via `prot_en` in the AMBA Configuration Register. Disabled by default — no optional fields are transmitted in the default configuration.
- **OPT_FIELD3**: Not used. PSLVERR errors are reported using the BoW.Flexi Configuration/Error Packet (CMD=111) mechanism, consistent with the AHB bridge.

OPT_FIELD1 and OPT_FIELD2 are not used by the APB bridge (APB has no burst or complex attributes).



Because APB only supports a single outstanding transaction per peripheral select, no transaction ID is needed. The `protocol_mode` field in [Interface Configuration Extension Register 1](#) distinguishes the `opt_field` interpretation for different bridge types.

B.4.1. OPT_FIELD0: Protection + USER (Requests Only)

When `prot_en` = 1 in the AMBA Configuration Register, OPT_FIELD0 carries the 3-bit PPROT signal:

```
OPT_FIELD0[0] = PPROT[0] (0=Normal, 1=Privileged)
OPT_FIELD0[1] = PPROT[1] (0=Secure, 1=Non-secure)
OPT_FIELD0[2] = PPROT[2] (0=Data, 1=Instruction)
```

When `user_width` > 0 in the AMBA Configuration Register, the USER defined field is appended:

```
Bits [3+N-1:3]: PUSER[N-1:0] (USER defined field)
```

The total `opt_field0_size` is set to 3 + `user_width`. When `user_width` = 0 (default), OPT_FIELD0 is 3 bits carrying PPROT only. The sub-fields within OPT_FIELD0 follow the BoW.Flexi optional-field packing convention (see [Section 5.1.1.2](#)).

OPT_FIELD0 is enabled for request commands only (`opt_field0_cmd_support` = 7'b0010011: Non-Posted Write, Read, Posted Write).

When `prot_en` = 0, OPT_FIELD0 is disabled (`opt_field0_cmd_support` = 0) and PPROT is not transmitted. This is appropriate when the bridge is configured for a single slave with a fixed address range, as the protection attributes provide no additional routing or access control information — the remote bridge always targets the same peripheral.

B.4.2. Error Reporting

The APB bridge reports errors using the BoW.Flexi Configuration/Error Packet (CMD=111) mechanism, the same as the other BoW.Flexi bridges: the **presence** of a response packet (CMD=010 Read Response or CMD=011 Write Response) implies PSLVERR=0 (OKAY), while a PSLVERR=1 condition is signaled by the remote bridge sending a Configuration/Error Packet instead of a normal response, which the local bridge

decodes to drive PSLVERR=1 on the local APB bus. This avoids adding an error bit to every response packet for an infrequent condition.

B.4.3. Write Byte Strobes (PSTRB) — Embedded in DATA

APB4/APB5 supports byte-level write strobes via PSTRB. When `wstrb_in_data = 1` in the AMBA Configuration Register, the DATA field of write request packets includes PSTRB after the write data:

Write DATA field: [PWDATA: DATA_WIDTH bits][PSTRB: DATA_WIDTH/8 bits]

For example with a 32-bit APB bus: write DATA = 32 + 4 = 36 bits.

When `wstrb_in_data = 0`, PSTRB is not transmitted and all bytes are assumed valid.

B.4.4. Bridge Signal Mapping

Table B.3. APB to BoW.Flexi Packet Mapping

APB Signal	Packet Field	Notes
PADDR	ADDR	Address transmitted in little-endian byte order (width set by <code>address_width</code> in Configuration Command; 1-64 bits; default 32)
PWRITE	CMD[2:0]	1=CMD[100] Posted Write or CMD[000] Non-Posted Write; 0=CMD[001] Read Request
PWDATA	DATA	Write data transmitted in little-endian byte order
PSTRB	DATA (appended)	Write byte strobes embedded after write data (1 bit per data byte). Enabled via <code>apb_strb_enable</code> . Only present in write request packets
PPROT[2:0]	OPT_FIELD0[2:0] (request only)	Protection type: [0]=Normal/Privileged; [1]=Secure/Non-secure; [2]=Data/Instruction. Present when <code>apb_prot_enable=1</code>
PRDATA	DATA	Read data received in little-endian byte order
PSLVERR	-	Reported via Configuration/Error Packet (CMD=111). Presence of response packet implies OKAY
PENABLE	-	Managed internally by bridge (APB protocol handshake)
PSEL	-	Managed internally by bridge (APB protocol handshake)
PREADY	-	Not transmitted; bridge waits for PREADY locally before sending response

B.5. APB Configuration Register

The APB bridge is configured by the unified AMBA Configuration Register (offset 0x70, see [Table 10.31](#)). In APB mode the applicable fields are `user_width`, `data_width`, `wstrb_in_data` (PSTRB), `prot_en` (PPROT), and `bridge_en`.

B.6. Packet Structure

B.6.1. Packet Format

B.6.1.1. APB Transaction Packets

APB transactions are always single-beat. The general packet template is:

```
[CMD:3][OPT_FIELD0][SIZE:3][ADDR:variable][DATA(+PSTRB):variable][PAD:variable]
```



For simplicity the APB bridge keeps the standard BoW.Flexi 3-bit SIZE field, but only the lower two bits are valid. Because APB is single-beat, SIZE never exceeds 4 bytes (encodings 000/001/010 = 1/2/4 bytes); the most-significant SIZE bit is always 0.

The following table shows which fields are present for each command and configuration:

Table B.4. APB Packet Field Presence

Transaction	CMD	OPT_FIELD 0 (PPROT+US ER)	SIZE + ADDR	DATA	PSTRB appended
Posted/Non-Posted Write (default)	000/100	—	Yes	Yes	No
Posted/Non-Posted Write (strb)	000/100	—	Yes	Yes	Yes
Posted/Non-Posted Write (PPROT + strb)	000/100	PPROT+US ER(3+N)	Yes	Yes	Yes
Posted/Non-Posted Write (PPROT, no strb)	000/100	PPROT+US ER(3+N)	Yes	Yes	No
Read Request (default)	001	—	Yes	—	—
Read Request (PPROT)	001	PPROT+US ER(3+N)	Yes	—	—
Read Response	010	—	—	Yes	—
Write Response	011	—	—	—	—

All packets are padded to align to the data-lane boundary.



Response packets (CMD=010 and CMD=011) do NOT include a PSLVERR field. The presence of a response packet implies PSLVERR=0 (OKAY). If the remote peripheral signals PSLVERR=1, the bridge reports this using the Configuration/Error Packet (CMD=111) mechanism.



APB is inherently a non-posted protocol (every write requires completion acknowledgment via PREADY). A Posted Write (CMD=100) still returns a Write Response, but that response is a flow-control acknowledgment that the bridge has accepted the write and is ready for another — it is returned without waiting for the peripheral's PREADY completion, reducing write latency.

Table B.5. APB Transaction Field Descriptions

Field	Bits	Description
CMD	3	Command type (000=Non-Posted Write; 001=Read; 010=Read Response; 011=Write Response; 100=Posted Write)
OPT_FIELD0 (PPROT)	3 (fixed)	Request packets only. Carries PPROT[2:0] (Protection: bit[0]=Privileged; bit[1]=Non-secure; bit[2]=Instruction). Not present in response packets. Can be disabled when apb_prot_enable=0
SIZE	3	Transfer size (2^{SIZE} bytes: 000=1 byte; 001=2 bytes; 010=4 bytes). Request packets only. APB supports a max of 4 bytes (SIZE=2); values above this are reserved. Not present in response packets
ADDR	Variable	Target peripheral address (width set by address_width in Configuration Command). Request packets only
DATA (+PSTRB)	Variable	Write data for request packets; Read data for Read Response packets. When apb_strb_enable=1: write packets contain [PWDATA][PSTRB] where PSTRB is DATA_WIDTH/8 bits
PAD	Variable	Zero padding to align to data lane boundary. Only required for data lanes > x1

B.6.2. Packet Size Calculation

APB packets are compact due to the single-beat nature of the protocol.

B.6.2.1. Write Request Packets (CMD=000 or CMD=100)

Default configuration (no PPROT, no PSTRB, 32-bit data, 32-bit address):

```
Packet_Bits = CMD(3) + SIZE(3) + ADDR(32) + DATA(32) = 70 bits
```

With PPROT and PSTRB enabled (32-bit data, 32-bit address, no USER):

```
Packet_Bits = CMD(3) + OPT_FIELD0(3+N) + SIZE(3) + ADDR(32) + DATA(32) + PSTRB(4) = 77+N bits
Padding_Bits = (DL - ((77+N) mod DL)) mod DL
```

B.6.2.2. Read Request Packets (CMD=001)

Default (no PPROT, 32-bit address):

$\text{Packet_Bits} = \text{CMD}(3) + \text{SIZE}(3) + \text{ADDR}(32) = 38 \text{ bits}$

With PPROT (32-bit address, no USER):

$\text{Packet_Bits} = \text{CMD}(3) + \text{OPT_FIELD0}(3+N) + \text{SIZE}(3) + \text{ADDR}(32) = 41+N \text{ bits}$

B.6.2.3. Read Response Packets (CMD=010)

With 32-bit data:

$\text{Packet_Bits} = \text{CMD}(3) + \text{DATA}(32) = 35 \text{ bits}$

B.6.2.4. Write Response Packets (CMD=011)

$\text{Packet_Bits} = \text{CMD}(3) = 3 \text{ bits}$

B.6.3. APB Packet Timing Summary

Table B.6. APB Packet Timing Summary (32-bit data, 32-bit address, x8 data lane)

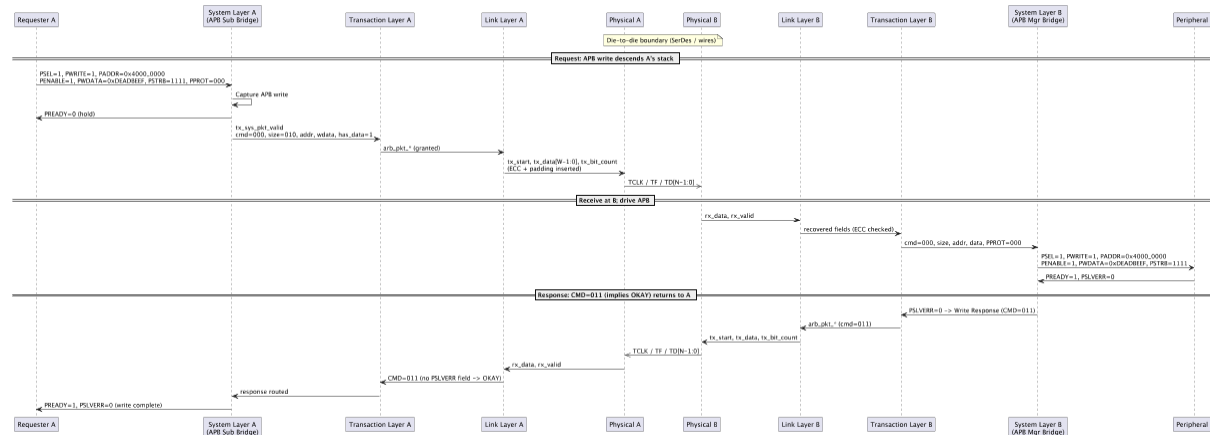
Transaction Type	Packet Bits	Padding Bits (x8)	Total Bits (x8)	Clock Cycles (x8)
Non-Posted Write (32-bit data; 32-bit addr; with PPROT)	73	7	80	10
Non-Posted Write (32-bit data; 32-bit addr; no PPROT)	70	2	72	9
Non-Posted Write (32-bit data; 32-bit addr; PPROT+PSTRB)	77	3	80	10
Posted Write (32-bit data; 32-bit addr; with PPROT)	73	7	80	10
Read Request (32-bit addr; with PPROT)	41	7	48	6
Read Request (32-bit addr; no PPROT)	38	2	40	5
Write Response	3	5	8	1
Read Response (32-bit data)	35	5	40	5
Config/Error Write (32-bit)	47	1	48	6
Config/Error Read Request	15	1	16	2

B.7. Transaction Examples

B.7.1. Non-Posted Write Transaction

A typical APB register write.

Figure B.2. APB Write Sequence



Example: Write 0x0000001F to address 0x00000100, with PPROT=3'b010 (Non-secure, Data, Normal) (8-bit data lane, 32-bit bus, 32-bit address, PSTRB enabled)

Request Packet:

```

CMD:      3 bits = 000 (Non-Posted Write)
OPT_FIELD0: 3 bits = 010 (PPROT: Non-secure, Data, Normal)
SIZE:     3 bits = 010 (4 bytes)
ADDR:     32 bits = 0x00000100
DATA:     32 bits = 0x0000001F
PSTRB:    4 bits = 1111 (all bytes valid)
  
```

```

Total: 3 + 3 + 3 + 32 + 32 + 4 = 77 bits
Padding (x8): (8 - (77 mod 8)) mod 8 = 3 bits
Total with padding: 80 bits = 10 bytes
Clock Cycles (x8): 80 / 8 = 10
  
```

Bit-level packet construction:

```

Bits [2:0]   = 000 (CMD: Non-Posted Write)
Bits [5:3]   = 010 (OPT_FIELD0: PPROT)
Bits [8:6]   = 010 (SIZE: 4 bytes)
Bits [40:9]  = 0x00000100 (ADDR)
Bits [72:41] = 0x0000001F (DATA)
Bits [76:73] = 1111 (PSTRB)
Bits [79:77] = 000 (PAD)
  
```

Response Packet:

```

CMD:      3 bits = 011 (Write Response)

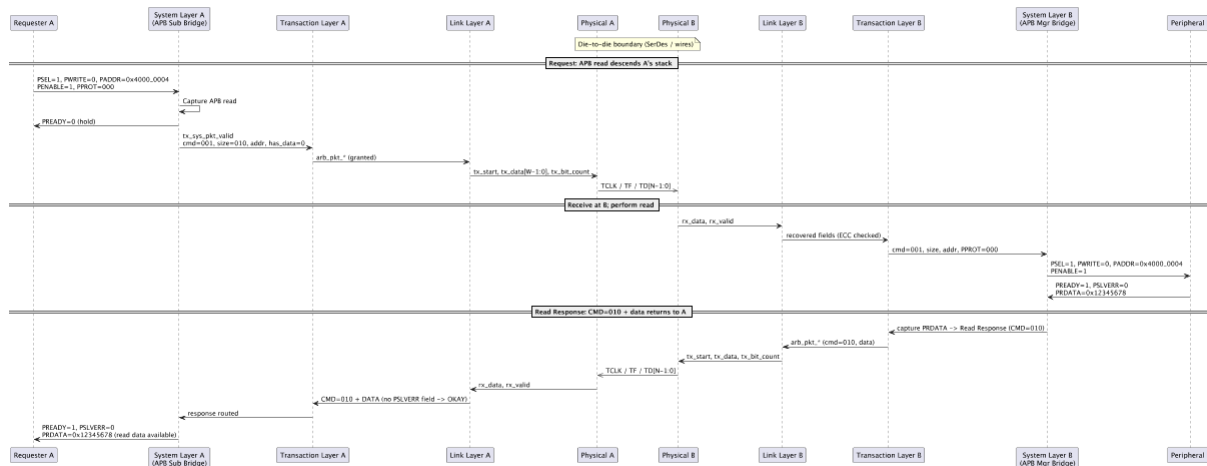
Total: 3 bits
Padding: (8 - (3 mod 8)) mod 8 = 5 bits
Total: 8 bits = 1 byte
  
```



The presence of the Write Response packet (CMD=011) implies PSLVERR=0. If the peripheral signals PSLVERR=1, a Configuration/Error Packet (CMD=111) is sent instead.

B.7.2. Read Transaction

Figure B.3. APB Read Sequence



Example: Read from address 0x0000200, with PPROT=3'b000 (Secure, Data, Normal) (8-bit data lane, 32-bit bus, 32-bit address)

Request Packet:

```
CMD:      3 bits = 001 (Read)
OPT_FIELD0: 3 bits = 000 (PPROT: Secure, Data, Normal)
SIZE:     3 bits = 010 (4 bytes)
ADDR:    32 bits = 0x0000200
```

```
Total: 3 + 3 + 3 + 32 = 41 bits
Padding: (8 - (41 mod 8)) mod 8 = 7 bits
Total with padding: 48 bits = 6 bytes
```

Response Packet (returning 0xABCD1234, no error):

```
CMD:      3 bits = 010 (Read Response)
DATA:    32 bits = 0xABCD1234
```

```
Total: 3 + 32 = 35 bits
Padding: (8 - (35 mod 8)) mod 8 = 5 bits
Total with padding: 40 bits = 5 bytes
```



The presence of the Read Response packet (CMD=010) implies PSLVERR=0.

B.7.3. Posted Write (Flow-Controlled)

For peripheral registers where completion acknowledgment is not required (e.g., interrupt clear registers, FIFO writes), a Posted Write lets the remote bridge return its Write Response as soon as it has accepted the write — without waiting for the peripheral's PREADY completion — reducing write latency:

Request Packet:

```

CMD:      3 bits = 100 (Posted Write)
OPT_FIELD0: 3 bits = 010 (PPROT)
SIZE:     3 bits = 010 (4 bytes)
ADDR:     32 bits = 0x00000300
DATA:     32 bits = 0x00000001

```

Total: 3 + 3 + 3 + 32 + 32 = 73 bits

Padding (x8): (8 - (73 mod 8)) mod 8 = 7 bits

Total with padding: 80 bits = 10 bytes

A Write Response (CMD=011) is returned as a flow-control acknowledgment.



The remote bridge returns the flow-control Write Response once it has accepted the write, and still completes the full APB handshake (PSEL→PENABLE→PREADY) locally. If the peripheral signals PSLVERR, the error is reported via the standard BoW.Flexi Configuration/Error mechanism (CMD=111).

B.8. Error Handling

B.8.1. PSLVERR Response

The APB bridge uses the BoW.Flexi Configuration/Error Packet (CMD=111) mechanism for PSLVERR reporting:

- **Non-Posted Write:** When the remote peripheral asserts PSLVERR, the remote bridge sends a Configuration/Error Packet (CMD=111) instead of a Write Response. The local bridge decodes the error packet and drives PSLVERR=1 on the local APB bus.
- **Read:** When the remote peripheral asserts PSLVERR, the remote bridge sends a Configuration/Error Packet (CMD=111) instead of a Read Response. The local bridge decodes the error packet and drives PSLVERR=1, returning invalid PRDATA.
- **Posted Write:** The flow-control Write Response is not sent; instead the error is reported via the CMD=111 Configuration/Error Packet to the remote error register.

B.8.2. Link-Level Errors

For errors at the BoW.Flexi link level, the standard error mechanism applies (see [Remote Error Register](#)).

B.9. Configuration Sequence

The following sequence configures the BoW.Flexi interface for APB bridge operation:

1. Initialize the BoW.Flexi link in default x1 mode
2. Configure data lanes as required via Configuration Command
3. Set `protocol_mode = 6` in [Interface Configuration Extension Register 1](#) for APB
4. Optionally set the Optional Field Configuration Register (offset 0x3C):
 - If PPROT desired: `opt_field0_size = 3 + user_width`, `opt_field0_cmd_support = 7'b0010011`
 - Otherwise: `opt_field0_cmd_support = 0`
5. Set the Optional Field Extended Configuration Register (offset 0x40):
 - `opt_field3_size = 0`
 - `opt_field3_cmd_support = 7'b00000000` (not used — errors via CMD=111)
6. Set the AMBA Configuration Register (offset 0x70):
 - `bridge_en = 1`
 - Configure data width as needed
 - Enable/disable PPROT, PSTRB, response as needed
7. Set `address_width` in the Configuration Command to match the APB address bus width (default 0x1F = 32 bits)

B.10. Reduced-Overhead Configuration

B.10.1. Single Slave with Fixed Address Range

When the bridge is configured to access a single peripheral with a fixed address range, PPROT can be disabled (`prot_en = 0`) since the protection attributes serve no useful purpose — the remote bridge always routes to the same slave regardless of privilege or security level. This eliminates 3 bits of overhead from every request packet.

B.10.2. Minimal APB (No Optional Fields)

For the simplest possible configuration:

- Disable `OPT_FIELD0` (no PPROT)
- Set `wstrb_in_data = 0` (all bytes valid)

This produces the most compact packets: * Write request: `CMD(3) + SIZE(3) + ADDR + DATA = 70` bits (32-bit bus, 32-bit addr) * Read request: `CMD(3) + SIZE(3) + ADDR = 38` bits * Read response: `CMD(3) + DATA = 35` bits * Write response: `CMD(3) = 3` bits

This is the optimal configuration for accessing simple control/status registers across the chiplet boundary.



For a comprehensive comparison of all ARM protocol bridges (APB, AHB, AXI) and their BoW.Flexi field usage, see [Chapter 12](#).

Appendix C. Example: Coexistence of Link, System and Advanced System Commands



Informative example. This appendix shows, at the signal level, how the three BoW.Flexi command classes share a single link, how the Transaction Layer's egress arbiter chooses between the link-interface, combined system, and optional dedicated advanced-system sources, and how a System Layer learns that the Physical Layer is ready to accept the next packet. For the normative class definitions and arbitration order see [Section 5.1.8](#).

C.1. Overview

A single BoW.Flexi link carries three classes of command ([Section 5.1.8](#)):

- **Link Interface** — Configuration Command / Error (CMD=111) and Virtual Wire (CMD=101), generated and consumed inside the Transaction Layer. Single outstanding is enforced **here**, in the Transaction Layer.
- **System Interface** — Non-Posted Write, Read Command, Posted Write and their responses, carried for a System Layer bridge (e.g. AHB). Single outstanding is enforced **above**, in the System Layer.
- **Advanced System Interface** — an optional, design-time system protocol that allows more than one outstanding transaction, using its own flow control (for example a credit scheme) in the System Layer. It is carried on the Extension Command (CMD=110) when `ext_cmd_class = 1` and enters the Transaction Layer on a **dedicated** source (`tx/rx_adv_sys_pkt_*`).

The System Interface class is owned by the System Layer, which arbitrates its new requests and any Read Response/Write Response it shall return and forwards the winner on a single combined system source (`tx_sys_pkt_*`), pending responses ahead of new requests. The optional Advanced System Interface, when implemented and enabled, is **not** merged into that source: it enters the Transaction Layer on a dedicated `tx/rx_adv_sys_pkt_*` source whose flow control (for example credits) remains in the System Layer. The only shared resource below the System Layer is the serializer, and access to it is granted by the Transaction Layer's egress arbiter: a round-robin arbiter among the Link Interface sources (Error, Virtual Wire, Config), followed by a round-robin arbiter between the resulting Link Interface source and the combined System source (the combined system/link winner), followed — when the advanced class is present — by a final, **implementation-dependent** stage that selects between that winner and the dedicated advanced source. The link-interface and top-level stages are round-robin, so no source can be starved and the egress path is deadlock-free.

C.2. Arbitration Between Sources

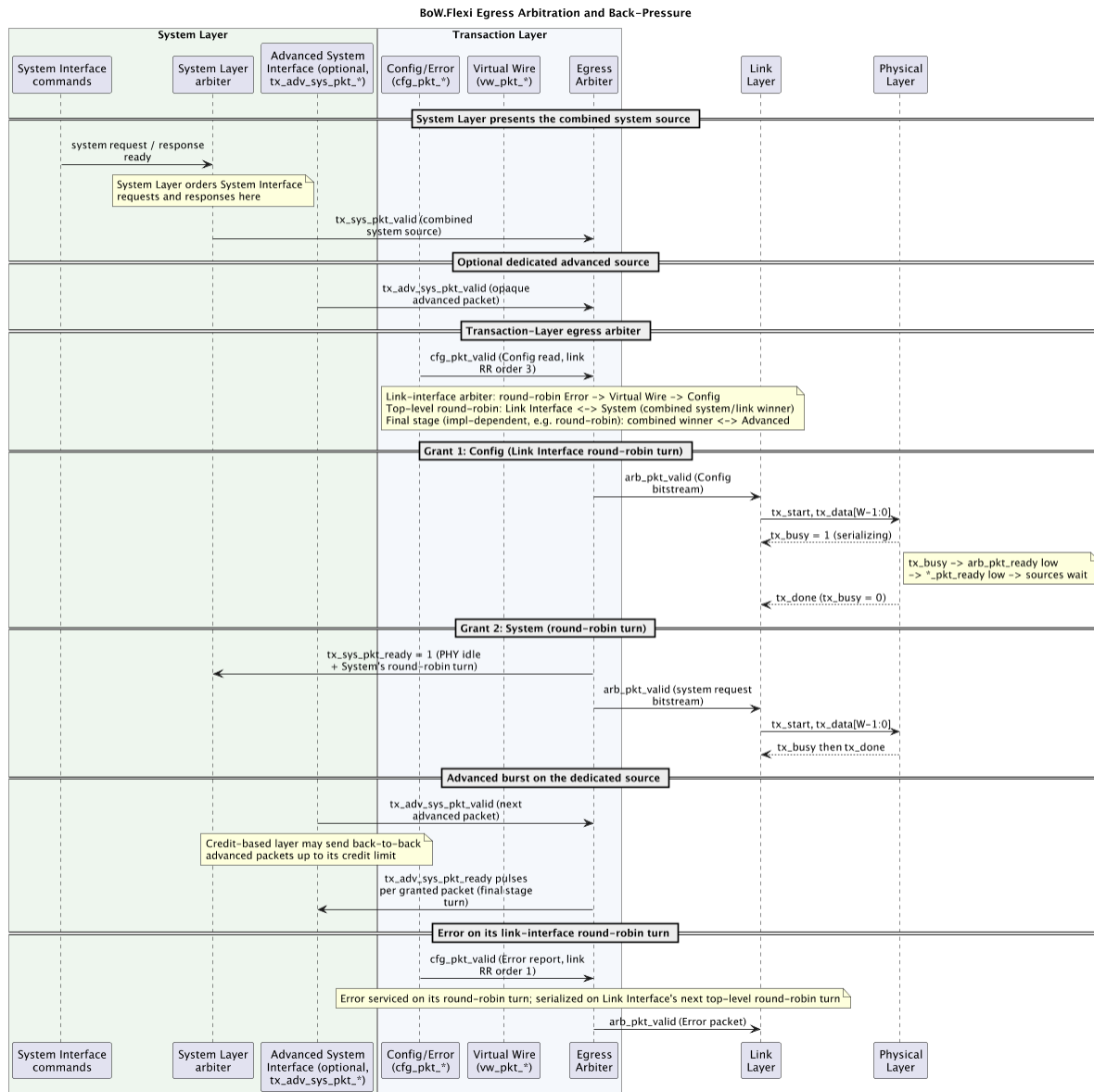
Every source that can drive the link raises its own **valid** and waits for the arbiter to assert the matching **ready**. Arbitration is re-evaluated at each packet boundary (packets are never interrupted mid-transmission) and proceeds in stages: first the three Link Interface sources are rotated round-robin in the order of [Table 5.9](#); then the resulting Link Interface source competes with the combined System source on a **round-**

robin basis (the combined system/link winner); finally, when the optional Advanced System Interface is present, an **implementation-dependent** stage selects between that winner and the dedicated advanced source:

Table C.1. Egress arbitration stages

Stage	Source signal	Class	Notes
Link-interface round-robin (order 1)	cfg_pkt_valid (Error)	Link Interface	An Error report; rotated round-robin with the other link-interface sources so a fault is always serviced within a bounded number of packets.
Link-interface round-robin (order 2)	vw_pkt_valid	Link Interface	Virtual Wire update.
Link-interface round-robin (order 3)	cfg_pkt_valid (Config)	Link Interface	Configuration read/write of the partner register space.
Top-level round-robin	Link Interface source vs tx_sys_pkt_valid	Link Interface vs System Interface	The link-interface winner alternates with the combined System source so neither class can starve the other. The System Layer presents pending responses ahead of new requests on tx_sys_pkt_*. The result is the combined system/link winner.
Final stage (implementation-dependent)	combined system/link winner vs tx_adv_sys_pkt_val id	combined vs Advanced System Interface	Present only when the optional advanced class is designed in and enabled (ext_cmd_class = 1). The policy is implementation-dependent; round-robin is the recommended example. There is no ordering guarantee between advanced and other traffic.

Figure C.1 traces a window in which the Config source and the System source both request the link, the System Layer then drives a burst of system transactions, and finally an Error report takes its link-interface round-robin turn and is serialized on the Link Interface source's next top-level round-robin turn.

Figure C.1. Egress arbitration across sources, with System-Layer arbitration and PHY-ready back-pressure

Reading the diagram:

1. **System Layer presents a request** — above the Transaction Layer, the System Layer selects among its ready System Interface commands (and any pending responses) and presents the winner on the tx_sys_pkt_* source.
2. **Config takes its round-robin turn** — with no Error or Virtual Wire pending, Config takes the link-interface round-robin turn and becomes the Link Interface source. On this top-level round-robin turn the Link Interface source is granted, so the configuration packet is serialized; the System source is guaranteed the next turn.
3. **System next** — on the following round-robin turn the arbiter grants the combined System source and the system request is serialized, independently of any pending Link Interface traffic.

4. **System burst** — the System source is then granted on its round-robin turns. Whether the System Layer may present back-to-back System Interface requests is governed by its policy: a single-outstanding bus (AHB) waits for its response before issuing the next. A credit-based Advanced System Interface instead bursts on its own dedicated source at the final arbitration stage. The Transaction arbiter only gates each packet on link availability and on the source winning its arbitration turn.
5. **Error on its link-interface round-robin turn** — when the Error source asserts, it is serviced on its link-interface round-robin turn (ahead of at most one other link-interface packet), then serialized on the Link Interface source's next top-level round-robin turn. Round-robin bounds the wait, so a fault is always reported without deadlock.

C.3. How the System Layer Knows the PHY Is Ready

The System Layer never observes the Physical Layer directly. Instead, link availability is folded into the single ready signal it already uses for the **valid/ready** handshake. The back-pressure propagates **up** the stack:

Table C.2. Signals between Layers

Signal	Meaning to the layer above
tx_busy (Physical → Link)	The serializer is shifting a packet onto the wires; no new tx_start may be issued.
arb_pkt_ready (Link → Transaction)	The Link packet-builder can accept the next bit-stream — deasserted while tx_busy is high or ECC/padding is in progress.
tx_sys_pkt_ready (Transaction → System)	This source has both won its round-robin turn and the link is free. Only then is the System Layer's request accepted.

So a System Layer source sees `tx_sys_pkt_ready = 1` only when two conditions hold simultaneously:

1. the Physical Layer is idle (`tx_busy = 0`, propagated as `arb_pkt_ready = 1`), and
2. the round-robin arbiter has selected the System source for this turn — i.e. it is the System source's turn, or no Link Interface source is currently requesting.

This single signal is all the System Layer needs: it asserts `tx_sys_pkt_valid` with its fields and waits. When `tx_sys_pkt_valid` && `tx_sys_pkt_ready` are both high on a clock edge, the transaction has been accepted into the link and the System Layer may advance its own state machine (for a single-outstanding bus, that means deasserting `HREADY` and waiting for the response). The dedicated `tx_adv_sys_pkt_*` source uses the identical **valid/ready** contract, so a credit-based Advanced System Interface decrements a credit and may present its next packet immediately. No knowledge of `TCLK`, lane count or serializer depth is required at the System Layer.



Because the arbiter re-evaluates only at packet boundaries, a source that has already been granted (for example a Virtual Wire packet) always completes its current packet before the arbiter selects the next source. Outstanding-policy ownership is unaffected: the Transaction Layer still gates Link Interface single-outstanding, while the System Layer gates **both** the System Interface policy

(on `tx_sys_pkt_*`) and the Advanced System Interface policy (on the dedicated `tx_adv_sys_pkt_*` source).

C.4. Worked Sequence

The following timeline lists the per-cycle events for a representative window in which a configuration read and a burst of Advanced System Interface transactions coexist. The System Layer in this example implements the optional Advanced System Interface class (credit-based), so it can present several advanced packets back-to-back on the dedicated `tx_adv_sys_pkt_*` source:

1. **Cycle 0** — firmware launches a Config read (`cfg_pkt_valid=1`); the System Layer presents its first advanced packet on `tx_adv_sys_pkt_valid=1`.
2. **Cycle 1** — the Transaction arbiter grants Config (its round-robin turn). `tx_adv_sys_pkt_ready` stays low. `tx_busy` rises as the Config packet serializes.
3. **Cycle k** — `tx_done` pulses; `arb_pkt_ready` returns high. With the advanced source due its turn at the final stage, the arbiter grants it: `tx_adv_sys_pkt_ready=1` for one cycle and the first advanced packet is accepted. The System Layer consumes one credit.
4. **Cycle k+1** — because the System Layer still has credits, it immediately presents the next advanced packet on the same `tx_adv_sys_pkt_*` source (`tx_adv_sys_pkt_valid=1`). The arbiter grants it as soon as the link is free and the final stage selects the advanced source; the Transaction Layer forwards the opaque payload without inspecting it.
5. **Cycle m** — the System Layer continues presenting advanced packets until it runs out of credits, at which point it deasserts `tx_adv_sys_pkt_valid` until a response replenishes a credit.
6. **Cycle n** — the partner returns an advanced response. On the receive path the Transaction Layer presents it on `rx_adv_sys_pkt_*`; the System Layer matches it to its outstanding request, retires that transaction and replenishes a credit. (All response pairing and ordering for this class is owned by the System Layer; the Transaction Layer only arbitrates the source.)

This ordering demonstrates the required behavior: the link interleaves Link Interface, System Interface and advanced-system traffic; the System Layer owns **all** system-class ordering (single-outstanding on `tx_sys_pkt_*` and credit-based on the dedicated `tx_adv_sys_pkt_*` source); and the Transaction Layer's egress arbiter — link-interface round-robin, top-level round-robin, and the implementation-dependent final stage — coordinates access to the Physical Layer with one **ready** signal per source.

Appendix D. Example: Layer Interaction at the Signal Level



Example appendix. This appendix is informative and previews a worked, signal-level description of how the four BoW.Flexi layers interact for a single transaction. It is intended to aid implementers and may change in a future revision. For the normative layer definitions see [BoW.Flexi Layered Architecture](#).

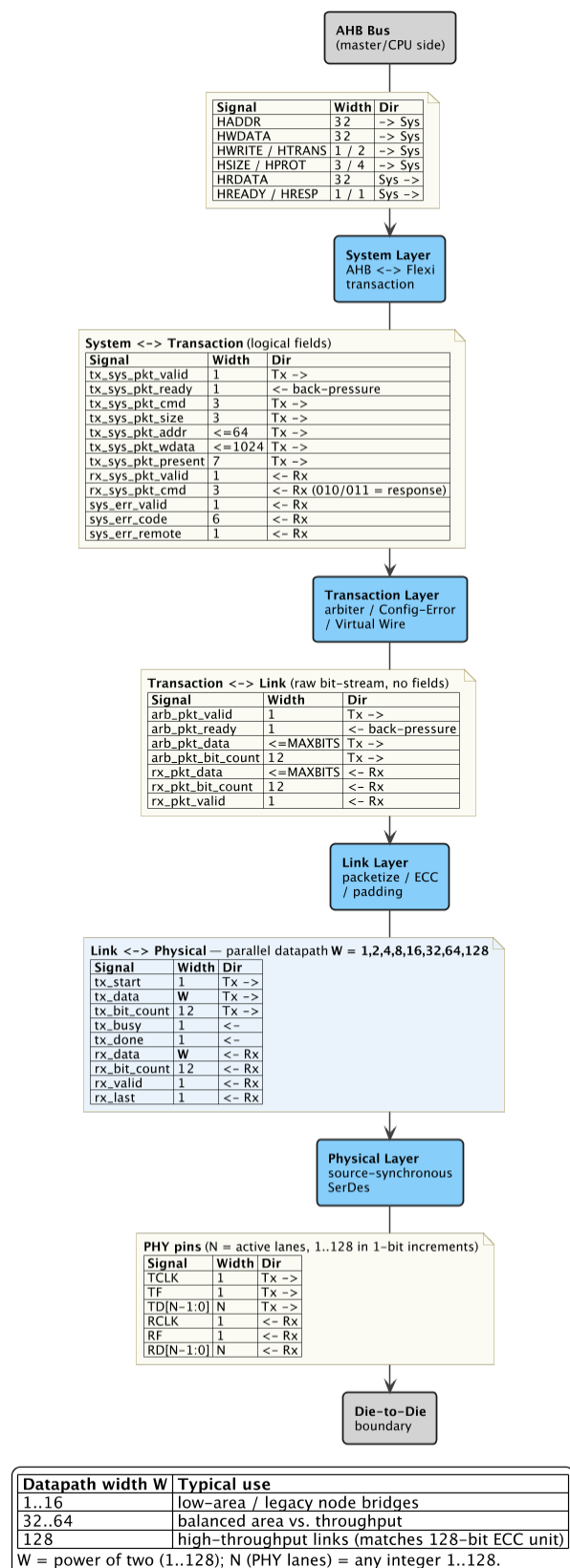
D.1. Overview

The BoW.Flexi stack is partitioned into four layers, each with a single responsibility and a narrow **valid/ready** field interface to its neighbors ([Figure 4.1](#)). This chapter follows one AHB write transaction down the transmit (Tx) stack and the matching response back up the receive (Rx) stack, naming the signals exchanged at every boundary.

Reading the stack from the system bus on the left to the die-to-die boundary on the right:

- **System Layer** — speaks AHB and abstract BoW.Flexi transactions.
- **Transaction Layer** — decides **what** to send and **when** (arbitration, Config/Error, Virtual Wire).
- **Link Layer** — turns logical fields into a PHY-sized, ECC-protected, padded bit-stream.
- **Physical Layer** — serializes the bit-stream across the wires.

[Figure D.1](#) shows the per-boundary signal bundles in a single drawing.

Figure D.1. BoW.Flexi layer-to-layer signal connections (AHB system layer example)

D.2. Boundary Signals

Each layer boundary uses a decoupled **valid/ready** handshake: a transfer happens on any cycle where both **valid** and **ready** are asserted. This lets a lower layer apply back-pressure (for example while the serializer is busy) without the upper layer needing to know why.

D.2.1. AHB Bus to System Layer

The System Layer terminates the local AHB bus. On the request side it captures **HADDR**, **HWDATA**, **HWRITE**, **HTRANS**, **HSIZE** and (optionally) **HPROT**; on the response side it drives **HRDATA**, **HREADY** and **HRESP**. The **HPROT[2]** (bufferable) bit selects a Posted versus Non-Posted Write, and **HREADY** is held low to enforce the single-outstanding-transaction rule until the response arrives.

D.2.2. System Layer to Transaction Layer

This boundary is the top edge of the BoW.Flexi specification: above it sits the System Layer (the bus bridge), below it the Transaction Layer. It is **bidirectional** and carries the same logical field set (**cmd**, **opt_field0..3**, **size**, **addr**, **data**, **opt_field_t**) in both directions, plus a **ready** back-pressure signal and a dedicated error-reporting path:

- **Egress (In to the link)** — the System Layer drives a new outbound transaction on the **tx_sys_pkt_*** bundle; **tx_sys_pkt_ready** tells it when the Transaction Layer (arbiter) can accept it.
- **Ingress (Out of the link)** — when a System Interface command **or** a Read/Write Response is received from the partner, the Transaction Layer presents the decoded fields to the System Layer on the **rx_sys_pkt_*** bundle. The System Layer drives inbound commands onto the remote bus and matches inbound responses to its own outstanding request.
- **Error** — when a Config/Error packet (CMD=111) writes an error register, the Transaction Layer pulses a dedicated error path (**sys_err_***) so the System Layer can terminate the outstanding transfer with an error.



The Transaction Layer does **not** track which system requests are outstanding or pair responses to requests. Read/Write Responses (CMD=010/011) simply arrive on **rx_sys_pkt_*** like any other inbound system command; the System Layer — which owns the single-outstanding (or credit) policy — matches them to its pending request.

The System Layer presents the transaction as a set of **logical fields** — it knows their meaning but nothing about bit positions, padding or ECC. Each field carries a presence flag indicating whether it is included for the current command.

Table D.1. Transaction Layer logical fields and presence flags

Field	Meaning	Presence flag
cmd	Command encoding (3 bits)	always present
opt_field0	Optional field 0	has_opt_field0
opt_field1	Optional field 1	has_opt_field1

Field	Meaning	Presence flag
opt_field2	Optional field 2	has_opt_field2
opt_field3	Optional field 3	has_opt_field3
size	SIZE field (power-of-2 bytes)	has_size
addr	Target address	has_addr
data	Read/write payload	has_data
opt_field_t	Trailing optional field (after DATA)	has_opt_field_t

The concrete signal bundles that carry these fields across each boundary — `tx/rx_sys_pkt_*`, the Config/Error and Virtual Wire sources, the optional `tx/rx_adv_sys_pkt_*` advanced ports, and the Link/Physical datapath — are defined normatively in the layer chapters. This preview does not restate them; the table below points to where each boundary is specified.

Table D.2. Layer boundary normative references

Layer boundary	Normative definition
AHB bus to System Layer	Appendix A
System Layer to Transaction Layer (<code>tx/rx_sys_pkt_*</code> , Config/Error, Virtual Wire, error path)	Section 5.1.8 , Section 5.1.9
Optional advanced-system ports (<code>tx/rx_adv_sys_pkt_*</code>)	Section 5.1.8 , Appendix C
Transaction Layer to Link Layer (raw packet bit-stream, then ECC and padding)	Section 6.1
Link Layer to Physical Layer and die-to-die pins	Section 7.5

D.3. Worked Examples

Signal-level walk-throughs — a Posted Write descending the stack, receiving a multi-beat Read Response, and configuring the partner chiplet through the local Config/Status handshake — follow directly from the normative flows. See the command classes and egress arbitration ([Section 5.1.8](#), [Section 5.1.9](#)), the coexistence example ([Appendix C](#)), and the register model ([Table 10.1](#)).