RAPIDS (Rapid AXI Programmable In-band Descriptor System) Specification

Generated by md\_to\_docx.py

today

Abstract

Table of Contents

# RAPIDS (Rapid AXI Programmable In-band Descriptor System) Specification

**Version:** 0.25 **Date:** 2025-10-18 **Status:** AXIS Migration Complete

## Overview

RAPIDS is a programmable DMA-style accelerator that provides: - High-performance data movement between network interfaces and system memory - In-band descriptor-based control - Multi-channel operation with per-channel buffering - AXI4-Stream network interfaces (AXIS migration complete) - AXI4 memory interfaces for system memory access - Comprehensive monitoring via MonBus

## Chapter 1: Interfaces

## RAPIDS Top-Level Interfaces v3.0 - AXIS4 Migration

### Clock and Reset

| Signal | IO | Description |
| --- | --- | --- |
| core\_clk | I | System clock |
| core\_rstn | I | Active-low reset |

### Source Data Path

#### AXI4 AR Master Interface (512-bit data)

| Signal | IO | Description |
| --- | --- | --- |
| axi\_src\_data\_ar\_valid | O | Read address valid |
| axi\_src\_data\_ar\_ready | I | Read address ready |
| axi\_src\_data\_ar\_addr[39:0] | O | Read address |
| axi\_src\_data\_ar\_len[7:0] | O | Burst length - 1 |
| axi\_src\_data\_ar\_size[2:0] | O | Transfer size |
| axi\_src\_data\_ar\_burst[1:0] | O | Burst type |
| axi\_src\_data\_ar\_id[7:0] | O | Transaction ID |
| axi\_src\_data\_ar\_lock | O | Lock type |
| axi\_src\_data\_ar\_cache[3:0] | O | Cache attributes |
| axi\_src\_data\_ar\_prot[2:0] | O | Protection attributes |
| axi\_src\_data\_ar\_qos[3:0] | O | Quality of Service |
| axi\_src\_data\_ar\_region[3:0] | O | Region identifier |
| axi\_src\_data\_ar\_user | O | User-defined |
| axi\_src\_data\_r\_valid | I | Read data valid |
| axi\_src\_data\_r\_ready | O | Read data ready |
| axi\_src\_data\_r\_data[511:0] | I | Read data |
| axi\_src\_data\_r\_id[7:0] | I | Transaction ID |
| axi\_src\_data\_r\_resp[1:0] | I | Read response |
| axi\_src\_data\_r\_last | I | Last transfer in burst |
| axi\_src\_data\_r\_user | I | User-defined |

#### AXI4 AW Master Interface (32-bit control data)

| Signal | IO | Description |
| --- | --- | --- |
| axi\_src\_ctrl\_aw\_valid | O | Write address valid |
| axi\_src\_ctrl\_aw\_ready | I | Write address ready |
| axi\_src\_ctrl\_aw\_addr[39:0] | O | Write address |
| axi\_src\_ctrl\_aw\_len[7:0] | O | Burst length - 1 |
| axi\_src\_ctrl\_aw\_size[2:0] | O | Transfer size |
| axi\_src\_ctrl\_aw\_burst[1:0] | O | Burst type |
| axi\_src\_ctrl\_aw\_id[7:0] | O | Transaction ID |
| axi\_src\_ctrl\_aw\_lock | O | Lock type |
| axi\_src\_ctrl\_aw\_cache[3:0] | O | Cache attributes |
| axi\_src\_ctrl\_aw\_prot[2:0] | O | Protection attributes |
| axi\_src\_ctrl\_aw\_qos[3:0] | O | Quality of Service |
| axi\_src\_ctrl\_aw\_region[3:0] | O | Region identifier |
| axi\_src\_ctrl\_aw\_user | O | User-defined |
| axi\_src\_ctrl\_w\_valid | O | Write data valid |
| axi\_src\_ctrl\_w\_ready | I | Write data ready |
| axi\_src\_ctrl\_w\_data[31:0] | O | Write data |
| axi\_src\_ctrl\_w\_strb[3:0] | O | Write strobes |
| axi\_src\_ctrl\_w\_last | O | Last transfer in burst |
| axi\_src\_ctrl\_w\_user | O | User-defined |
| axi\_src\_ctrl\_b\_valid | I | Write response valid |
| axi\_src\_ctrl\_b\_ready | O | Write response ready |
| axi\_src\_ctrl\_b\_id[7:0] | I | Transaction ID |
| axi\_src\_ctrl\_b\_resp[1:0] | I | Write response |
| axi\_src\_ctrl\_b\_user | I | User-defined |

#### AXI4 AR Master Interface (512-bit descriptor data)

| Signal | IO | Description |
| --- | --- | --- |
| axi\_src\_desc\_ar\_valid | O | Read address valid |
| axi\_src\_desc\_ar\_ready | I | Read address ready |
| axi\_src\_desc\_ar\_addr[39:0] | O | Read address |
| axi\_src\_desc\_ar\_len[7:0] | O | Burst length - 1 |
| axi\_src\_desc\_ar\_size[2:0] | O | Transfer size |
| axi\_src\_desc\_ar\_burst[1:0] | O | Burst type |
| axi\_src\_desc\_ar\_id[7:0] | O | Transaction ID |
| axi\_src\_desc\_ar\_lock | O | Lock type |
| axi\_src\_desc\_ar\_cache[3:0] | O | Cache attributes |
| axi\_src\_desc\_ar\_prot[2:0] | O | Protection attributes |
| axi\_src\_desc\_ar\_qos[3:0] | O | Quality of Service |
| axi\_src\_desc\_ar\_region[3:0] | O | Region identifier |
| axi\_src\_desc\_ar\_user | O | User-defined |
| axi\_src\_desc\_r\_valid | I | Read data valid |
| axi\_src\_desc\_r\_ready | O | Read data ready |
| axi\_src\_desc\_r\_data[511:0] | I | Read data |
| axi\_src\_desc\_r\_id[7:0] | I | Transaction ID |
| axi\_src\_desc\_r\_resp[1:0] | I | Read response |
| axi\_src\_desc\_r\_last | I | Last transfer in burst |
| axi\_src\_desc\_r\_user | I | User-defined |

#### AXI4-Lite Slave Interface (Source Configuration)

| Signal | IO | Description |
| --- | --- | --- |
| axil\_src\_cfg\_aw\_valid | I | Write address valid |
| axil\_src\_cfg\_aw\_ready | O | Write address ready |
| axil\_src\_cfg\_aw\_addr[39:0] | I | Write address |
| axil\_src\_cfg\_aw\_prot[2:0] | I | Protection attributes |
| axil\_src\_cfg\_w\_valid | I | Write data valid |
| axil\_src\_cfg\_w\_ready | O | Write data ready |
| axil\_src\_cfg\_w\_data[31:0] | I | Write data |
| axil\_src\_cfg\_w\_strb[3:0] | I | Write strobes |
| axil\_src\_cfg\_b\_valid | O | Write response valid |
| axil\_src\_cfg\_b\_ready | I | Write response ready |
| axil\_src\_cfg\_b\_resp[1:0] | O | Write response |
| axil\_src\_cfg\_ar\_valid | I | Read address valid |
| axil\_src\_cfg\_ar\_ready | O | Read address ready |
| axil\_src\_cfg\_ar\_addr[39:0] | I | Read address |
| axil\_src\_cfg\_ar\_prot[2:0] | I | Protection attributes |
| axil\_src\_cfg\_r\_valid | O | Read data valid |
| axil\_src\_cfg\_r\_ready | I | Read data ready |
| axil\_src\_cfg\_r\_data[31:0] | O | Read data |
| axil\_src\_cfg\_r\_resp[1:0] | O | Read response |

#### AXI4-Stream Master Interface (TX) - NEW v3.0

| Signal | IO | Description |
| --- | --- | --- |
| axis\_src\_tx\_tdata[511:0] | O | Stream data payload |
| axis\_src\_tx\_tstrb[63:0] | O | Byte strobes (write enables) |
| axis\_src\_tx\_tlast | O | Last transfer in packet |
| axis\_src\_tx\_tvalid | O | Stream data valid |
| axis\_src\_tx\_tready | I | Stream ready (backpressure) |
| axis\_src\_tx\_tuser[15:0] | O | User sideband (packet metadata) |

**TUSER Encoding (Source TX):**

[15:8] - Reserved for future use  
[7:0] - Packet type/flags

**Note:** AXIS uses standard tstrb for byte-level validity instead of custom chunk\_enables. All credits and ACK mechanisms removed from streaming interface.

### Sink Data Path

#### AXI4 AW Master Interface (512-bit data)

| Signal | IO | Description |
| --- | --- | --- |
| axi\_snk\_data\_aw\_valid | O | Write address valid |
| axi\_snk\_data\_aw\_ready | I | Write address ready |
| axi\_snk\_data\_aw\_addr[39:0] | O | Write address |
| axi\_snk\_data\_aw\_len[7:0] | O | Burst length - 1 |
| axi\_snk\_data\_aw\_size[2:0] | O | Transfer size |
| axi\_snk\_data\_aw\_burst[1:0] | O | Burst type |
| axi\_snk\_data\_aw\_id[7:0] | O | Transaction ID |
| axi\_snk\_data\_aw\_lock | O | Lock type |
| axi\_snk\_data\_aw\_cache[3:0] | O | Cache attributes |
| axi\_snk\_data\_aw\_prot[2:0] | O | Protection attributes |
| axi\_snk\_data\_aw\_qos[3:0] | O | Quality of Service |
| axi\_snk\_data\_aw\_region[3:0] | O | Region identifier |
| axi\_snk\_data\_aw\_user | O | User-defined |
| axi\_snk\_data\_w\_valid | O | Write data valid |
| axi\_snk\_data\_w\_ready | I | Write data ready |
| axi\_snk\_data\_w\_data[511:0] | O | Write data |
| axi\_snk\_data\_w\_strb[63:0] | O | Write strobes |
| axi\_snk\_data\_w\_last | O | Last transfer in burst |
| axi\_snk\_data\_w\_user | O | User-defined |
| axi\_snk\_data\_b\_valid | I | Write response valid |
| axi\_snk\_data\_b\_ready | O | Write response ready |
| axi\_snk\_data\_b\_id[7:0] | I | Transaction ID |
| axi\_snk\_data\_b\_resp[1:0] | I | Write response |
| axi\_snk\_data\_b\_user | I | User-defined |

#### AXI4 AW Master Interface (32-bit control data)

| Signal | IO | Description |
| --- | --- | --- |
| axi\_snk\_ctrl\_aw\_valid | O | Write address valid |
| axi\_snk\_ctrl\_aw\_ready | I | Write address ready |
| axi\_snk\_ctrl\_aw\_addr[39:0] | O | Write address |
| axi\_snk\_ctrl\_aw\_len[7:0] | O | Burst length - 1 |
| axi\_snk\_ctrl\_aw\_size[2:0] | O | Transfer size |
| axi\_snk\_ctrl\_aw\_burst[1:0] | O | Burst type |
| axi\_snk\_ctrl\_aw\_id[7:0] | O | Transaction ID |
| axi\_snk\_ctrl\_aw\_lock | O | Lock type |
| axi\_snk\_ctrl\_aw\_cache[3:0] | O | Cache attributes |
| axi\_snk\_ctrl\_aw\_prot[2:0] | O | Protection attributes |
| axi\_snk\_ctrl\_aw\_qos[3:0] | O | Quality of Service |
| axi\_snk\_ctrl\_aw\_region[3:0] | O | Region identifier |
| axi\_snk\_ctrl\_aw\_user | O | User-defined |
| axi\_snk\_ctrl\_w\_valid | O | Write data valid |
| axi\_snk\_ctrl\_w\_ready | I | Write data ready |
| axi\_snk\_ctrl\_w\_data[31:0] | O | Write data |
| axi\_snk\_ctrl\_w\_strb[3:0] | O | Write strobes |
| axi\_snk\_ctrl\_w\_last | O | Last transfer in burst |
| axi\_snk\_ctrl\_w\_user | O | User-defined |
| axi\_snk\_ctrl\_b\_valid | I | Write response valid |
| axi\_snk\_ctrl\_b\_ready | O | Write response ready |
| axi\_snk\_ctrl\_b\_id[7:0] | I | Transaction ID |
| axi\_snk\_ctrl\_b\_resp[1:0] | I | Write response |
| axi\_snk\_ctrl\_b\_user | I | User-defined |

#### AXI4 AR Master Interface (512-bit descriptor data)

| Signal | IO | Description |
| --- | --- | --- |
| axi\_snk\_desc\_ar\_valid | O | Read address valid |
| axi\_snk\_desc\_ar\_ready | I | Read address ready |
| axi\_snk\_desc\_ar\_addr[39:0] | O | Read address |
| axi\_snk\_desc\_ar\_len[7:0] | O | Burst length - 1 |
| axi\_snk\_desc\_ar\_size[2:0] | O | Transfer size |
| axi\_snk\_desc\_ar\_burst[1:0] | O | Burst type |
| axi\_snk\_desc\_ar\_id[7:0] | O | Transaction ID |
| axi\_snk\_desc\_ar\_lock | O | Lock type |
| axi\_snk\_desc\_ar\_cache[3:0] | O | Cache attributes |
| axi\_snk\_desc\_ar\_prot[2:0] | Protection attributes |  |
| axi\_snk\_desc\_ar\_qos[3:0] | O | Quality of Service |
| axi\_snk\_desc\_ar\_region[3:0] | O | Region identifier |
| axi\_snk\_desc\_ar\_user | O | User-defined |
| axi\_snk\_desc\_r\_valid | I | Read data valid |
| axi\_snk\_desc\_r\_ready | O | Read data ready |
| axi\_snk\_desc\_r\_data[511:0] | I | Read data |
| axi\_snk\_desc\_r\_id[7:0] | I | Transaction ID |
| axi\_snk\_desc\_r\_resp[1:0] | I | Read response |
| axi\_snk\_desc\_r\_last | I | Last transfer in burst |
| axi\_snk\_desc\_r\_user | I | User-defined |

#### AXI4-Lite Slave Interface (Sink Configuration)

| Signal | IO | Description |
| --- | --- | --- |
| axil\_snk\_cfg\_aw\_valid | I | Write address valid |
| axil\_snk\_cfg\_aw\_ready | O | Write address ready |
| axil\_snk\_cfg\_aw\_addr[39:0] | I | Write address |
| axil\_snk\_cfg\_aw\_prot[2:0] | I | Protection attributes |
| axil\_snk\_cfg\_w\_valid | I | Write data valid |
| axil\_snk\_cfg\_w\_ready | O | Write data ready |
| axil\_snk\_cfg\_w\_data[31:0] | I | Write data |
| axil\_snk\_cfg\_w\_strb[3:0] | I | Write strobes |
| axil\_snk\_cfg\_b\_valid | O | Write response valid |
| axil\_snk\_cfg\_b\_ready | I | Write response ready |
| axil\_snk\_cfg\_b\_resp[1:0] | O | Write response |
| axil\_snk\_cfg\_ar\_valid | I | Read address valid |
| axil\_snk\_cfg\_ar\_ready | O | Read address ready |
| axil\_snk\_cfg\_ar\_addr[39:0] | I | Read address |
| axil\_snk\_cfg\_ar\_prot[2:0] | I | Protection attributes |
| axil\_snk\_cfg\_r\_valid | O | Read data valid |
| axil\_snk\_cfg\_r\_ready | I | Read data ready |
| axil\_snk\_cfg\_r\_data[31:0] | O | Read data |
| axil\_snk\_cfg\_r\_resp[1:0] | O | Read response |

#### AXI4-Stream Slave Interface (RX) - NEW v3.0

| Signal | IO | Description |
| --- | --- | --- |
| axis\_snk\_rx\_tdata[511:0] | I | Stream data payload |
| axis\_snk\_rx\_tstrb[63:0] | I | Byte strobes (write enables) |
| axis\_snk\_rx\_tlast | I | Last transfer in packet |
| axis\_snk\_rx\_tvalid | I | Stream data valid |
| axis\_snk\_rx\_tready | O | Stream ready (backpressure) |
| axis\_snk\_rx\_tuser[15:0] | I | User sideband (packet metadata) |

**TUSER Encoding (Sink RX):**

[15:8] - Reserved for future use  
[7:0] - Packet type/flags

**Note:** AXIS uses standard tstrb for byte-level validity instead of custom chunk\_enables. All credits and ACK mechanisms removed from streaming interface.

### Monitor Bus AXI4-Lite Group Interfaces

#### AXI4-Lite Slave Interface (Error/Interrupt Read)

| Signal | IO | Description |
| --- | --- | --- |
| axil4\_mon\_err\_ar\_valid | I | Read address valid |
| axil4\_mon\_err\_ar\_ready | O | Read address ready |
| axil4\_mon\_err\_ar\_addr[31:0] | I | Read address |
| axil4\_mon\_err\_ar\_prot[2:0] | I | Protection attributes |
| axil4\_mon\_err\_r\_valid | O | Read data valid |
| axil4\_mon\_err\_r\_ready | I | Read data ready |
| axil4\_mon\_err\_r\_data[63:0] | O | Read data (64-bit monitor packets) |
| axil4\_mon\_err\_r\_resp[1:0] | O | Read response |

#### AXI4-Lite Master Interface (Monitor Write)

| Signal | IO | Description |
| --- | --- | --- |
| axil4\_mon\_wr\_aw\_valid | O | Write address valid |
| axil4\_mon\_wr\_aw\_ready | I | Write address ready |
| axil4\_mon\_wr\_aw\_addr[31:0] | O | Write address |
| axil4\_mon\_wr\_aw\_prot[2:0] | O | Protection attributes |
| axil4\_mon\_wr\_w\_valid | O | Write data valid |
| axil4\_mon\_wr\_w\_ready | I | Write data ready |
| axil4\_mon\_wr\_w\_data[31:0] | O | Write data |
| axil4\_mon\_wr\_w\_strb[3:0] | O | Write strobes |
| axil4\_mon\_wr\_b\_valid | I | Write response valid |
| axil4\_mon\_wr\_b\_ready | O | Write response ready |
| axil4\_mon\_wr\_b\_resp[1:0] | I | Write response |

#### AXI4-Lite Slave Interface (Monitor Configuration)

| Signal | IO | Description |
| --- | --- | --- |
| axil4\_mon\_cfg\_aw\_valid | I | Write address valid |
| axil4\_mon\_cfg\_aw\_ready | O | Write address ready |
| axil4\_mon\_cfg\_aw\_addr[31:0] | I | Write address |
| axil4\_mon\_cfg\_aw\_prot[2:0] | I | Protection attributes |
| axil4\_mon\_cfg\_w\_valid | I | Write data valid |
| axil4\_mon\_cfg\_w\_ready | O | Write data ready |
| axil4\_mon\_cfg\_w\_data[31:0] | I | Write data |
| axil4\_mon\_cfg\_w\_strb[3:0] | I | Write strobes |
| axil4\_mon\_cfg\_b\_valid | O | Write response valid |
| axil4\_mon\_cfg\_b\_ready | I | Write response ready |
| axil4\_mon\_cfg\_b\_resp[1:0] | O | Write response |
| axil4\_mon\_cfg\_ar\_valid | I | Read address valid |
| axil4\_mon\_cfg\_ar\_ready | O | Read address ready |
| axil4\_mon\_cfg\_ar\_addr[31:0] | I | Read address |
| axil4\_mon\_cfg\_ar\_prot[2:0] | I | Protection attributes |
| axil4\_mon\_cfg\_r\_valid | O | Read data valid |
| axil4\_mon\_cfg\_r\_ready | I | Read data ready |
| axil4\_mon\_cfg\_r\_data[31:0] | O | Read data |
| axil4\_mon\_cfg\_r\_resp[1:0] | O | Read response |

### Key Interface Changes from v2.1 to v3.0

#### **REMOVED - Custom Network Protocol:**

* ~~network\_\*\_pkt\_valid/ready~~ - Replaced by standard axis\_\*\_tvalid/tready
* ~~network\_\*\_pkt\_data[511:0]~~ - Replaced by axis\_\*\_tdata[511:0]
* ~~network\_\*\_pkt\_type[1:0]~~ - Moved to axis\_\*\_tuser[7:0]
* ~~network\_\*\_pkt\_addr[7:0]~~ - Removed (no addressing in streaming)
* ~~network\_\*\_pkt\_addr\_par~~ - Removed (parity optional via TUSER if needed)
* ~~network\_\*\_pkt\_eos~~ - Replaced by axis\_\*\_tlast
* ~~network\_\*\_pkt\_par~~ - Removed (parity optional via TUSER if needed)
* **~~ALL ACK signals~~** - Removed completely (no credit/ACK on streaming)
  + ~~network\_\*\_ack\_valid/ready~~
  + ~~network\_\*\_ack\_ack[1:0]~~
  + ~~network\_\*\_ack\_addr[7:0]~~
  + ~~network\_\*\_ack\_addr\_par~~
  + ~~network\_\*\_ack\_par~~
* ~~Embedded chunk\_enables format~~ - Replaced by standard axis\_\*\_tstrb[63:0]

#### **ADDED - Standard AXIS4 Protocol:**

* **axis\_src\_tx\_tdata[511:0]** - Source TX data stream
* **axis\_src\_tx\_tstrb[63:0]** - Byte-level write enables (64 bytes for 512-bit bus)
* **axis\_src\_tx\_tlast** - Packet boundary marker
* **axis\_src\_tx\_tvalid/tready** - Standard handshake protocol
* **axis\_src\_tx\_tuser[15:0]** - Optional metadata sideband
* **axis\_snk\_rx\_tdata[511:0]** - Sink RX data stream
* **axis\_snk\_rx\_tstrb[63:0]** - Byte-level write enables
* **axis\_snk\_rx\_tlast** - Packet boundary marker
* **axis\_snk\_rx\_tvalid/tready** - Standard handshake protocol
* **axis\_snk\_rx\_tuser[15:0]** - Optional metadata sideband

#### **Migration Benefits:**

1. **Industry Standard**: AXIS4 is widely supported, well-documented standard protocol
2. **Simplified Flow Control**: Standard tvalid/tready backpressure, no custom ACK channels
3. **Cleaner Byte Qualification**: Standard tstrb replaces embedded chunk\_enables
4. **Packet Framing**: Standard tlast replaces custom EOS markers
5. **Reduced Complexity**: Eliminated custom packet types, addresses, parity, ACK logic
6. **Tool Support**: Better IP integration, simulation, and verification tool support
7. **No Interface Credits**: Simplified interface - credits remain only in scheduler (internal)

#### **AXIS4 vs Custom Network Protocol Mapping:**

| Custom Network v2.1 | AXIS4 v3.0 | Notes |
| --- | --- | --- |
| network\_\*\_pkt\_data[511:0] | axis\_\*\_tdata[511:0] | Direct data payload |
| network\_\*\_pkt\_chunk\_enables[15:0] (embedded) | axis\_\*\_tstrb[63:0] | Byte-level granularity |
| network\_\*\_pkt\_eos | axis\_\*\_tlast | Standard packet boundary |
| network\_\*\_pkt\_valid/ready | axis\_\*\_tvalid/tready | Standard handshake |
| network\_\*\_pkt\_type[1:0] | axis\_\*\_tuser[7:0] | Metadata in sideband |
| network\_\*\_pkt\_addr[7:0] | **REMOVED** | No addressing in streaming |
| network\_\*\_pkt\_par | **REMOVED** | Optional via TUSER if needed |
| network\_\*\_ack\_\* (all) | **REMOVED** | No ACK/credit on interface |

### Interface Summary

#### **Total AXI Interfaces:**

* **Source:** 3 AXI4 Masters (data read, ctrl write, desc read) + 1 AXI4-Lite Slave (config)
* **Sink:** 3 AXI4 Masters (data write, ctrl write, desc read) + 1 AXI4-Lite Slave (config)
* **Monitor:** 1 AXI4-Lite Master (write) + 2 AXI4-Lite Slaves (error read, config)

#### **Total AXIS Interfaces (NEW v3.0):**

* **Source:** 1 AXIS4 Master (TX streaming)
* **Sink:** 1 AXIS4 Slave (RX streaming)

#### **Key Features:**

* **Standard AXIS4 Protocol** for high-bandwidth streaming
* **Comprehensive AXI4-Lite Configuration** for all subsystems
* **Monitor Bus Aggregation** with configurable filtering
* **Error/Interrupt Handling** via dedicated AXI4-Lite interface
* **Proper Clock/Reset** with core\_clk and core\_rstn
* **Simplified Flow Control** - No custom ACK or credit mechanisms on streaming interfaces
* **Industry-Standard Interfaces** - Better tool support and IP reuse

### AXIS Data Path Integration

#### Source Data Path (Memory -> AXIS TX):

AXI4 Read Master (512-bit)  
 ↓ Read data from system memory  
Source SRAM Control  
 ↓ Buffer management  
AXIS Master (rtl/amba/axis/axis\_master.sv)  
 ↓ axis\_src\_tx\_\* signals  
External AXIS Receiver

**Key Points:** - SRAM control writes to axis\_master FUB interface (fub\_axis\_tdata/tstrb/tlast/tvalid) - AXIS master outputs external m\_axis\_\* signals - Backpressure: axis\_src\_tx\_tready=0 -> SRAM control stalls - Packet framing: SRAM sets tlast on final beat

#### Sink Data Path (AXIS RX -> Memory):

External AXIS Transmitter  
 ↓ axis\_snk\_rx\_\* signals  
AXIS Slave (rtl/amba/axis/axis\_slave.sv)  
 ↓ Internal FUB interface  
Sink SRAM Control  
 ↓ Buffer management  
AXI4 Write Master (512-bit)  
 ↓ Write to system memory

**Key Points:** - AXIS slave receives external s\_axis\_\* signals - Outputs to SRAM via FUB interface (fub\_axis\_tdata/tstrb/tlast/tvalid) - Backpressure: SRAM full -> axis\_snk\_rx\_tready=0 -> upstream stalls - Packet framing: tlast=1 triggers SRAM to finalize packet

**See: See:** - ch03\_interfaces/04\_axis4\_interface\_spec.md - Complete AXIS4 specification - rtl/amba/axis/axis\_master.sv - AXIS master RTL - rtl/amba/axis/axis\_slave.sv - AXIS slave RTL ## AXI4-Lite Interface Specification and Assumptions

### Overview

This document defines the formal specification and assumptions for an AXI4-Lite interface implementation. AXI4-Lite is a subset of AXI4 optimized for simple, lightweight control register interfaces with inherent protocol simplifications.

### Interface Summary

#### Number of Interfaces

* **2 Master Read Interface**: Single read channel for Monitor Packets (one for each Source and Sink)
* **2 Master Write Interface**: Single write channel for Monitor Packets plus a timestamp (one for each Source and Sink)

#### Interface Parameters

| Parameter | Description | Valid Values | Default |
| --- | --- | --- | --- |
| DATA\_WIDTH | AXI data bus width in bits | 32, 64 | 32, 64 |
| ADDR\_WIDTH | AXI address bus width in bits | 32, 64 | 37 |
| STRB\_WIDTH | Write strobe width | DATA\_WIDTH/8 | 8 |

### Core Protocol Assumptions

#### Inherent AXI4-Lite Simplifications

AXI4-Lite protocol inherently provides the following constraints:

| Constraint | Description |
| --- | --- |
| **Single Transfers Only** | No burst transactions supported |
| **No Transaction IDs** | All transactions are in-order |
| **Fixed Transfer Size** | Always uses full data bus width |
| **No User Signals** | Simplified interface without user-defined extensions |

#### Implementation Assumptions

##### Assumption 1: Address Alignment to Data Bus Width

| Aspect | Requirement |
| --- | --- |
| **Alignment Rule** | All AXI4-Lite transactions aligned to data bus width |
| **32-bit bus alignment** | Address[1:0] must be 2’b00 (4-byte aligned) |
| **64-bit bus alignment** | Address[2:0] must be 3’b000 (8-byte aligned) |
| **Rationale** | Maximizes bus efficiency and eliminates unaligned access complexity |
| **Benefit** | Simplifies address decode and data steering logic |

##### Assumption 2: Fixed Transfer Size

| Aspect | Requirement |
| --- | --- |
| **Transfer Size Rule** | All transfers use maximum size equal to bus width |
| **32-bit bus** | AxSIZE = 3’b010 (4 bytes) |
| **64-bit bus** | AxSIZE = 3’b011 (8 bytes) |
| **Rationale** | Maximizes bus utilization and simplifies control logic |
| **Benefit** | No size decode logic required |

##### Assumption 3: No Address Wraparound

| Aspect | Requirement |
| --- | --- |
| **Wraparound Rule** | Transactions never wrap around top of address space |
| **Rationale** | Control register accesses never require wraparound behavior |
| **Benefit** | Simplified address boundary checking |

##### Assumption 4: Standard Protection Attributes

| Access Type | AxPROT Value | Description |
| --- | --- | --- |
| **Normal Access** | 3’b000 | Data, secure, unprivileged |
| **Privileged Access** | 3’b001 | Data, secure, privileged |
| **Rationale** |  | Covers the majority of control register access patterns |

### Master Read Interface Specification

#### Read Address Channel (AR)

| Signal | Width | Direction | Required Values | Description |
| --- | --- | --- | --- | --- |
| ar\_addr | ADDR\_WIDTH | Master->Slave | **8-byte aligned** | Read address |
| ar\_prot | 3 | Master->Slave | Implementation specific | Protection attributes |
| ar\_valid | 1 | Master->Slave | 0 or 1 | Address valid |
| ar\_ready | 1 | Slave->Master | 0 or 1 | Address ready |

#### Read Data Channel (R)

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| r\_data | 64 | Slave->Master | Read data |
| r\_resp | 2 | Slave->Master | Read response |
| r\_valid | 1 | Slave->Master | Read data valid |
| r\_ready | 1 | Master->Slave | Read data ready |

#### AXI4-Lite Simplifications (Read)

| Removed Signal | AXI4 Usage | AXI4-Lite Reason |
| --- | --- | --- |
| **ar\_id** | Transaction ID | Single transfers, no transaction IDs |
| **ar\_len** | Burst length | Single transfers only |
| **ar\_size** | Transfer size | Fixed to bus width |
| **ar\_burst** | Burst type | Single transfers only |
| **ar\_lock** | Lock type | Simplified access model |
| **ar\_cache** | Cache attributes | Simplified memory model |
| **ar\_qos** | Quality of Service | Simplified priority model |
| **ar\_region** | Region identifier | Simplified address space |
| **ar\_user** | User-defined | Simplified interface |
| **r\_id** | Transaction ID | No transaction IDs needed |
| **r\_last** | Last transfer | Single transfers only |
| **r\_user** | User-defined | Simplified interface |

### Master Write Interface Specification

#### Write Address Channel (AW)

| Signal | Width | Direction | Required Values | Description |
| --- | --- | --- | --- | --- |
| aw\_addr | ADDR\_WIDTH | Master->Slave | **8-byte aligned** | Write address |
| aw\_prot | 3 | Master->Slave | Implementation specific | Protection attributes |
| aw\_valid | 1 | Master->Slave | 0 or 1 | Address valid |
| aw\_ready | 1 | Slave->Master | 0 or 1 | Address ready |

#### Write Data Channel (W)

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| w\_data | 32 | Master->Slave | Write data |
| w\_strb | 4 | Master->Slave | Write strobes (byte enables) |
| w\_valid | 1 | Master->Slave | Write data valid |
| w\_ready | 1 | Slave->Master | Write data ready |

#### Write Response Channel (B)

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| b\_resp | 2 | Slave->Master | Write response |
| b\_valid | 1 | Slave->Master | Response valid |
| b\_ready | 1 | Master->Slave | Response ready |

#### AXI4-Lite Simplifications (Write)

| Removed Signal | AXI4 Usage | AXI4-Lite Reason |
| --- | --- | --- |
| **aw\_id** | Transaction ID | Single transfers, no transaction IDs |
| **aw\_len** | Burst length | Single transfers only |
| **aw\_size** | Transfer size | Fixed to bus width |
| **aw\_burst** | Burst type | Single transfers only |
| **aw\_lock** | Lock type | Simplified access model |
| **aw\_cache** | Cache attributes | Simplified memory model |
| **aw\_qos** | Quality of Service | Simplified priority model |
| **aw\_region** | Region identifier | Simplified address space |
| **aw\_user** | User-defined | Simplified interface |
| **w\_last** | Last transfer | Single transfers only |
| **w\_user** | User-defined | Simplified interface |
| **b\_id** | Transaction ID | No transaction IDs needed |
| **b\_user** | User-defined | Simplified interface |

### Address Requirements

#### Address Alignment Rules

| Alignment Type | Formula | Description |
| --- | --- | --- |
| **Valid Address** | (Address % 4) == 0 | Must be 8-byte aligned |
| **Mandatory Alignment** | Address[2:0] must be 3’b000 | Per Assumption 1 |

#### Address Validation Examples

| Address Category | Examples | Status |
| --- | --- | --- |
| **Valid (84byte aligned)** | 0x1000, 0x1004, 0x1008, 0x100C | Accepted |
| **Invalid (unaligned)** | 0x1001, 0x1002, 0x1003 | DECERR response |

### Response Codes

#### Response Code Specification

| Value | Name | Description | Usage in Control Registers |
| --- | --- | --- | --- |
| **2’b00** | OKAY | Normal access success | Successful register access |
| **2’b01** | EXOKAY | Exclusive access success | **Not used in AXI4-Lite** |
| **2’b10** | SLVERR | Slave error | Invalid register access |
| **2’b11** | DECERR | Decode error | **Address decode failure or misalignment** |

#### Response Usage Guidelines

| Response Type | Usage | Description |
| --- | --- | --- |
| **OKAY** | Normal completion | Successful register access |
| **EXOKAY** | Not applicable | AXI4-Lite doesn’t support exclusive accesses |
| **SLVERR** | Register error | Invalid register operation |
| **DECERR** | Address error | Misalignment or decode failure per Assumption 1 |

### Protection Signal Usage

#### Protection Signal Encoding

| Bit | Name | Description | Recommended Usage |
| --- | --- | --- | --- |
| **[0]** | Privileged | 0=Normal, 1=Privileged | Set based on processor mode |
| **[1]** | Non-secure | 0=Secure, 1=Non-secure | Set based on security domain |
| **[2]** | Instruction | 0=Data, 1=Instruction | Always 0 for control registers |

#### Common Protection Patterns

| Pattern | AxPROT Value | Description |
| --- | --- | --- |
| **Normal Data Access** | 3’b000 | Standard register access |
| **Privileged Data Access** | 3’b001 | Privileged register access |
| **Debug Access** | 3’b010 | Debug register access |
| **Privileged Debug** | 3’b011 | Privileged debug access |

### Implementation Benefits

#### Simplified Control Register Interface

| Benefit Area | Simplification | Impact |
| --- | --- | --- |
| **Address Decode** | Simple 8-byte aligned address comparison | Reduced decode logic |
| **Transaction Handling** | No burst or ID tracking required | Simplified state machines |
| **Flow Control** | Straightforward valid-ready handshakes | Reduced complexity |
| **Response Generation** | Simple OKAY/SLVERR/DECERR responses | Minimal response logic |
| **Size Handling** | Fixed 64-bit transfers only | No size decode needed |

#### Address Decode Implementation

| Implementation Aspect | Method | Benefit |
| --- | --- | --- |
| **4-byte Alignment Check** | addr[1:0] == 2’b00 | Simple bit masking |
| **Address Range Check** | addr >= base && addr <= limit | Simple comparisons |
| **Combined Check** | alignment\_ok && range\_ok | Single decode decision |

#### Error Generation Logic

| Error Condition | Check | Response |
| --- | --- | --- |
| **Address Misalignment** | addr[1:0] != 2’b00 | Generate DECERR |
| **Address Out of Range** | !addr\_in\_range(addr) | Generate DECERR |
| **Register Error** | register\_error\_condition | Generate SLVERR |
| **Normal Access** | All checks pass | Generate OKAY |

### Timing Requirements

#### Handshake Protocol

| Protocol Rule | Requirement | Description |
| --- | --- | --- |
| **Valid-Ready Transfer** | Transfer occurs when both VALID and READY are high | Standard AXI handshake |
| **Valid Independence** | VALID can be asserted independently of READY | Master controls valid |
| **Ready Dependency** | READY can depend on VALID state | Slave controls ready |
| **Signal Stability** | Once VALID asserted, all signals stable until READY | Data integrity |

#### Channel Dependencies

| Dependency | Requirement | Description |
| --- | --- | --- |
| **Write Channels** | AW and W channels are independent | Can be presented in any order |
| **Write Response** | B channel waits for both AW and W completion | Response dependency |
| **Read Channels** | R channel waits for AR channel completion | Response dependency |
| **Transaction Ordering** | Multiple outstanding transactions not supported | Inherent AXI4-Lite limitation |

#### Reset Behavior

| Reset Phase | Requirement | Description |
| --- | --- | --- |
| **Active Reset** | aresetn is active-low reset signal | Standard AXI reset |
| **Reset Requirements** | All VALID signals deasserted during reset | Clean reset state |
| **Reset Recovery** | All VALID signals low after reset deassertion | Proper startup |

### Validation Requirements

#### Functional Validation

| Validation Area | Requirements |
| --- | --- |
| **Address Alignment** | Verify all accesses are 8-byte aligned per Assumption 1 |
| **Fixed Size** | Verify all transfers are full 64-bit width per Assumption 2 |
| **Response Correctness** | Verify appropriate response codes (DECERR for misaligned access) |
| **Handshake Compliance** | Verify all valid-ready handshakes |
| **Register Behavior** | Verify read/write register functionality |
| **No Wraparound** | Verify no address wraparound scenarios per Assumption 3 |

#### Timing Validation

| Validation Area | Requirements |
| --- | --- |
| **Setup/Hold** | Verify signal timing requirements |
| **Reset Behavior** | Verify proper reset sequence |
| **Back-pressure** | Verify ready signal behavior under load |

#### Error Injection Testing

| Test Type | Injection Method | Expected Response |
| --- | --- | --- |
| **Misaligned Address** | Inject addresses with addr[2:0] != 0 | DECERR response |
| **Out of Range** | Inject addresses outside valid range | DECERR response |
| **Register Errors** | Inject register-specific errors | SLVERR response |

### Example Transactions

#### 64-bit Register Write

| Parameter | Value | Description |
| --- | --- | --- |
| **Bus Width** | 64 bits (8 bytes) | Data bus configuration |
| **Target Address** | 0x1000 (8-byte aligned) | Valid aligned address |
| **Write Data** | 0xDEADBEEFCAFEBABE | 64-bit data value |
| **Required Settings** | aw\_addr=0x1000, aw\_prot=3’b000, w\_data=0xDEADBEEFCAFEBABE, w\_strb=8’b11111111 | Transaction configuration |

#### AW Transaction Flow

| Step | Action | Signal States |
| --- | --- | --- |
| **1** | Assert aw\_valid with address | aw\_valid=1, aw\_addr=0x1000 |
| **2** | Assert w\_valid with data | w\_valid=1, w\_data=0xDEADBEEFCAFEBABE |
| **3** | Wait for handshakes | aw\_ready=1, w\_ready=1 |
| **4** | Wait for response | b\_valid=1, b\_resp=OKAY |
| **5** | Complete transaction | b\_ready=1 |

#### 64-bit Register Read

| Parameter | Value | Description |
| --- | --- | --- |
| **Bus Width** | 64 bits (8 bytes) | Data bus configuration |
| **Target Address** | 0x1008 (8-byte aligned) | Valid aligned address |
| **Required Settings** | ar\_addr=0x1008, ar\_prot=3’b000 | Transaction configuration |

#### AR Transaction Flow

| Step | Action | Signal States |
| --- | --- | --- |
| **1** | Assert ar\_valid with address | ar\_valid=1, ar\_addr=0x1008 |
| **2** | Wait for address handshake | ar\_ready=1 |
| **3** | Wait for data response | r\_valid=1, r\_resp=OKAY |
| **4** | Complete transaction | r\_ready=1 |
| **5** | Capture data | r\_data (64 bits) |

#### Misaligned Address Example

| Parameter | Value | Description |
| --- | --- | --- |
| **Bus Width** | 64 bits (8 bytes) | Data bus configuration |
| **Target Address** | 0x1004 (misaligned) | Invalid address |
| **Expected Behavior** | Address decode detects misalignment -> DECERR response -> No register access | Error handling |

### Common Use Cases

#### Typical Applications

| Application | Description |
| --- | --- |
| **Control/Status Registers** | 64-bit device configuration and monitoring |
| **Memory-Mapped Peripherals** | Simple register-based devices |
| **Debug Interfaces** | Debug and trace control registers |
| **Configuration Space** | PCIe configuration space access |
| **Performance Counters** | 64-bit performance monitoring registers |

#### Performance Considerations

| Consideration | Impact | Description |
| --- | --- | --- |
| **Latency** | Single-cycle responses preferred | Simple registers |
| **Throughput** | Limited by single outstanding transaction | AXI4-Lite constraint |
| **Efficiency** | 64-bit transfers maximize data efficiency | Modern system optimization |

## AXI4 Interface Specification and Assumptions

### Overview

This document defines the formal specification and assumptions for an AXI4 interface implementation that supports two distinct transfer modes to optimize for different interface types while ensuring robust, predictable operation.

### Interface Summary

#### Number of Interfaces

* **5 Master Read Interfaces**: Descriptor sink, descriptor source, data source, flag sink, and flag source
* **3 Master Write Interfaces**: Data sink, control sink, and control source

#### Interface Parameters

| Parameter | Description | Valid Values | Default |
| --- | --- | --- | --- |
| DATA\_WIDTH | AXI data bus width in bits | 32, 64, 128, 256, 512, 1024 | 32 |
| ADDR\_WIDTH | AXI address bus width in bits | 32, 64 | 37 |
| ID\_WIDTH | AXI ID tag width in bits | 1-16 | 8 |
| USER\_WIDTH | AXI user signal width in bits (optional) | 0-16 | 1 |

#### Interface Types and Transfer Modes

| Interface Group | Channels | Transfer Mode | Address Alignment | Monitor | DCG | Notes |
| --- | --- | --- | --- | --- | --- | --- |
| **AXI4 Master Read-Split** | AR, R | **Flexible** | 4-byte | Yes | Yes | Data interfaces with chunk enables |
| **AXI4 Master Write-Split** | AW, W, B | **Flexible** | 4-byte | Yes | Yes | Data interfaces with chunk enables |
| **AXI4 Master Read** | AR, R | **Simplified** | Bus-width | No | Yes | Control interfaces, fully aligned |
| **AXI4 Master Write** | AW, W, B | **Simplified** | Bus-width | Yes | Yes | Control interfaces, fully aligned |

#### Interface Group Parameter Settings

| Interface Group | Data Width | Address Width | ID Width | User Width | Transfer Mode |
| --- | --- | --- | --- | --- | --- |
| **AXI4 Master Read-Split** | 512 bits | 37 bits | 8 bits | 1 bit | **Flexible** |
| **AXI4 Master Write-Split** | 512 bits | 37 bits | 8 bits | 1 bit | **Flexible** |
| **AXI4 Master Read** | 32 bits | 37 bits | 8 bits | 1 bit | **Simplified** |
| **AXI4 Master Write** | 32 bits | 37 bits | 8 bits | 1 bit | **Simplified** |

#### Interface Configuration Summary

| Interface Type | Interface Group | Transfer Mode | Alignment | Notes |
| --- | --- | --- | --- | --- |
| **Descriptor Sink** | **AXI4 Master Read** | **Simplified** | 32-bit aligned | Control interface |
| **Descriptor Source** | **AXI4 Master Read** | **Simplified** | 32-bit aligned | Control interface |
| **Data Source** | **AXI4 Master Read-Split** | **Flexible** | 4-byte aligned | High-bandwidth data |
| **Data Sink** | **AXI4 Master Write-Split** | **Flexible** | 4-byte aligned | High-bandwidth data |
| **Program Sink** | **AXI4 Master Write** | **Simplified** | 32-bit aligned | Control interface |
| **Program Source** | **AXI4 Master Write** | **Simplified** | 32-bit aligned | Control interface |
| **Flag Sink** | **AXI4 Master Read** | **Simplified** | 32-bit aligned | Control interface |
| **Flag Source** | **AXI4 Master Read** | **Simplified** | 32-bit aligned | Control interface |

### Transfer Mode Specifications

This specification defines two distinct transfer modes to optimize different interface types:

#### Mode 1: Simplified Transfer Mode (Control Interfaces)

Used for control interfaces (descriptors, programs, flags) that prioritize simplicity and predictable timing.

##### **Simplified Mode Assumptions**

| Aspect | Requirement |
| --- | --- |
| **Address Alignment** | All addresses aligned to full data bus width |
| **Transfer Size** | All transfers use maximum size equal to bus width |
| **Burst Type** | Incrementing bursts only (AxBURST = 2’b01) |
| **Transfer Complexity** | Maximum simplicity for predictable operation |

#### Mode 2: Flexible Transfer Mode (Data Interfaces)

Used for high-bandwidth data interfaces that need to handle arbitrary address alignment while maintaining efficiency.

##### **Flexible Mode Assumptions**

| Aspect | Requirement |
| --- | --- |
| **Address Alignment** | 4-byte aligned addresses (minimum alignment) |
| **Transfer Sizes** | Multiple sizes supported: 4, 8, 16, 32, 64 bytes |
| **Burst Type** | Incrementing bursts only (AxBURST = 2’b01) |
| **Alignment Strategy** | Progressive alignment to optimize bus utilization |

### Mode 1: Simplified Transfer Mode Specification

#### Assumption 1: Address Alignment to Data Bus Width

| Aspect | Requirement |
| --- | --- |
| **Alignment Rule** | All AXI transactions aligned to data bus width |
| **32-bit bus (4 bytes)** | Address[1:0] must be 2’b00 |
| **64-bit bus (8 bytes)** | Address[2:0] must be 3’b000 |
| **128-bit bus (16 bytes)** | Address[3:0] must be 4’b0000 |
| **256-bit bus (32 bytes)** | Address[4:0] must be 5’b00000 |
| **512-bit bus (64 bytes)** | Address[5:0] must be 6’b000000 |
| **1024-bit bus (128 bytes)** | Address[6:0] must be 7’b0000000 |
| **Rationale** | Maximizes bus efficiency and eliminates unaligned access complexity |

#### Assumption 2: Fixed Transfer Size

| Aspect | Requirement |
| --- | --- |
| **Transfer Size Rule** | All transfers use maximum size equal to bus width |
| **32-bit bus** | AxSIZE = 3’b010 (4 bytes) |
| **64-bit bus** | AxSIZE = 3’b011 (8 bytes) |
| **128-bit bus** | AxSIZE = 3’b100 (16 bytes) |
| **256-bit bus** | AxSIZE = 3’b101 (32 bytes) |
| **512-bit bus** | AxSIZE = 3’b110 (64 bytes) |
| **1024-bit bus** | AxSIZE = 3’b111 (128 bytes) |
| **Rationale** | Maximizes bus utilization and simplifies address alignment |

### Mode 2: Flexible Transfer Mode Specification

#### Assumption 1: 4-Byte Address Alignment

| Aspect | Requirement |
| --- | --- |
| **Alignment Rule** | All AXI transactions aligned to 4-byte boundaries |
| **Address Constraint** | Address[1:0] must be 2’b00 |
| **Rationale** | Balances flexibility with AXI protocol requirements |
| **Benefit** | Supports arbitrary data placement while maintaining AXI compliance |

#### Assumption 2: Multiple Transfer Sizes

| Transfer Size | AxSIZE Value | Use Case |
| --- | --- | --- |
| **4 bytes** | 3’b010 | Initial alignment, small transfers |
| **8 bytes** | 3’b011 | Progressive alignment |
| **16 bytes** | 3’b100 | Progressive alignment |
| **32 bytes** | 3’b101 | Progressive alignment |
| **64 bytes** | 3’b110 | Optimal full-width transfers |
| **128 bytes** | 3’b111 | Maximum efficiency (1024-bit bus) |

#### Assumption 3: Progressive Alignment Strategy

| Aspect | Requirement |
| --- | --- |
| **Alignment Goal** | Align to 64-byte boundaries for optimal bus utilization |
| **Alignment Sequence** | Use progressive sizes: 4 -> 8 -> 16 -> 32 -> 64 bytes |
| **Optimization** | Choose largest possible transfer size at each step |
| **Example** | Address 0x1004: 4-byte transfer -> aligned to 0x1008, then larger transfers |

#### Assumption 4: Chunk Enable Support

| Aspect | Requirement |
| --- | --- |
| **Chunk Granularity** | 16 chunks of 32-bits each (512-bit bus) |
| **Write Strobes** | Generated from chunk enables for precise byte control |
| **Alignment Transfers** | Chunk patterns optimized for alignment sequences |
| **Benefits** | Precise data validity, optimal memory utilization |

### Common Protocol Assumptions (Both Modes)

#### Assumption 1: Incrementing Bursts Only

| Aspect | Requirement |
| --- | --- |
| **Burst Type** | All AXI bursts use incrementing address mode (AxBURST = 2’b01) |
| **Excluded Types** | No FIXED (2’b00) or WRAP (2’b10) bursts supported |
| **Rationale** | Simplifies address generation logic and covers most use cases |
| **Benefit** | Eliminates wrap boundary calculations and fixed address handling |

#### Assumption 2: No Address Wraparound

| Aspect | Requirement |
| --- | --- |
| **Wraparound Rule** | Transactions never wrap around top of address space |
| **Example** | No 0xFFFFFFFF -> 0x00000000 transitions |
| **Rationale** | Real systems never allow this due to memory layout |
| **Benefit** | Dramatically simplified boundary crossing detection logic |

### Flexible Mode: Address Calculation Examples

#### Progressive Alignment Examples

**Example 1: Address 0x1004 -> 0x1040 (64-byte boundary)**

| Step | Address | Size | AxSIZE | Length | Bytes Transferred | Notes |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0x1004 | 4 bytes | 3’b010 | 1 beat | 4 | Initial alignment |
| 2 | 0x1008 | 8 bytes | 3’b011 | 1 beat | 8 | Progressive alignment |
| 3 | 0x1010 | 16 bytes | 3’b100 | 1 beat | 16 | Progressive alignment |
| 4 | 0x1020 | 32 bytes | 3’b101 | 1 beat | 32 | Progressive alignment |
| 5 | 0x1040 | **64 bytes** | 3’b110 | N beats | 64xN | **Optimal transfers** |

**Example 2: Address 0x1010 -> 0x1040 (64-byte boundary)**

| Step | Address | Size | AxSIZE | Length | Bytes Transferred | Notes |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0x1010 | 16 bytes | 3’b100 | 1 beat | 16 | Optimal initial size |
| 2 | 0x1020 | 32 bytes | 3’b101 | 1 beat | 32 | Progressive alignment |
| 3 | 0x1040 | **64 bytes** | 3’b110 | N beats | 64xN | **Optimal transfers** |

#### Chunk Enable Pattern Examples

**512-bit Bus with 16x32-bit chunks**

| Transfer Size | Address Offset | Chunk Pattern | Description |
| --- | --- | --- | --- |
| **4 bytes** | 0x04 | 16’h0002 | Chunk 1 only |
| **8 bytes** | 0x08 | 16’h000C | Chunks 2-3 |
| **16 bytes** | 0x10 | 16’h00F0 | Chunks 4-7 |
| **32 bytes** | 0x20 | 16’hFF00 | Chunks 8-15 |
| **64 bytes** | 0x00 | 16’hFFFF | All chunks |

### Master Read Interface Specification

#### Read Address Channel (AR)

| Signal | Width | Direction | Simplified Mode | Flexible Mode | Description |
| --- | --- | --- | --- | --- | --- |
| ar\_addr | ADDR\_WIDTH | Master->Slave | **Bus-width aligned** | **4-byte aligned** | Read address |
| ar\_len | 8 | Master->Slave | 0-255 | 0-255 | Burst length - 1 |
| ar\_size | 3 | Master->Slave | **Fixed per bus** | **Variable: 4-64 bytes** | Transfer size |
| ar\_burst | 2 | Master->Slave | **2’b01 (INCR only)** | **2’b01 (INCR only)** | Burst type |
| ar\_id | ID\_WIDTH | Master->Slave | Any | Any | Transaction ID |
| ar\_lock | 1 | Master->Slave | 1’b0 | 1’b0 | Lock type (normal) |
| ar\_cache | 4 | Master->Slave | Implementation specific | 4’b0011 | Cache attributes |
| ar\_prot | 3 | Master->Slave | Implementation specific | 3’b000 | Protection attributes |
| ar\_qos | 4 | Master->Slave | 4’b0000 | 4’b0000 | Quality of Service |
| ar\_region | 4 | Master->Slave | 4’b0000 | 4’b0000 | Region identifier |
| ar\_user | USER\_WIDTH | Master->Slave | Optional | Optional | User-defined |
| ar\_valid | 1 | Master->Slave | 0 or 1 | 0 or 1 | Address valid |
| ar\_ready | 1 | Slave->Master | 0 or 1 | 0 or 1 | Address ready |

#### Read Data Channel (R)

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| r\_data | DATA\_WIDTH | Slave->Master | Read data |
| r\_id | ID\_WIDTH | Slave->Master | Transaction ID |
| r\_resp | 2 | Slave->Master | Read response |
| r\_last | 1 | Slave->Master | Last transfer in burst |
| r\_user | USER\_WIDTH | Slave->Master | User-defined (optional) |
| r\_valid | 1 | Slave->Master | Read data valid |
| r\_ready | 1 | Master->Slave | Read data ready |

### Master Write Interface Specification

#### Write Address Channel (AW)

| Signal | Width | Direction | Simplified Mode | Flexible Mode | Description |
| --- | --- | --- | --- | --- | --- |
| aw\_addr | ADDR\_WIDTH | Master->Slave | **Bus-width aligned** | **4-byte aligned** | Write address |
| aw\_len | 8 | Master->Slave | 0-255 | 0-255 | Burst length - 1 |
| aw\_size | 3 | Master->Slave | **Fixed per bus** | **Variable: 4-64 bytes** | Transfer size |
| aw\_burst | 2 | Master->Slave | **2’b01 (INCR only)** | **2’b01 (INCR only)** | Burst type |
| aw\_id | ID\_WIDTH | Master->Slave | Any | Any | Transaction ID |
| aw\_lock | 1 | Master->Slave | 1’b0 | 1’b0 | Lock type (normal) |
| aw\_cache | 4 | Master->Slave | Implementation specific | 4’b0011 | Cache attributes |
| aw\_prot | 3 | Master->Slave | Implementation specific | 3’b000 | Protection attributes |
| aw\_qos | 4 | Master->Slave | 4’b0000 | 4’b0000 | Quality of Service |
| aw\_region | 4 | Master->Slave | 4’b0000 | 4’b0000 | Region identifier |
| aw\_user | USER\_WIDTH | Master->Slave | Optional | Optional | User-defined |
| aw\_valid | 1 | Master->Slave | 0 or 1 | 0 or 1 | Address valid |
| aw\_ready | 1 | Slave->Master | 0 or 1 | 0 or 1 | Address ready |

#### Write Data Channel (W)

| Signal | Width | Direction | Simplified Mode | Flexible Mode | Description |
| --- | --- | --- | --- | --- | --- |
| w\_data | DATA\_WIDTH | Master->Slave | Write data | Write data | Write data |
| w\_strb | DATA\_WIDTH/8 | Master->Slave | **All 1’s** | **From chunk enables** | Write strobes |
| w\_last | 1 | Master->Slave | Last transfer | Last transfer | Last transfer in burst |
| w\_user | USER\_WIDTH | Master->Slave | Optional | Optional | User-defined |
| w\_valid | 1 | Master->Slave | 0 or 1 | 0 or 1 | Write data valid |
| w\_ready | 1 | Slave->Master | 0 or 1 | 0 or 1 | Write data ready |

#### Write Response Channel (B)

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| b\_id | ID\_WIDTH | Slave->Master | Transaction ID |
| b\_resp | 2 | Slave->Master | Write response |
| b\_user | USER\_WIDTH | Slave->Master | User-defined (optional) |
| b\_valid | 1 | Slave->Master | Response valid |
| b\_ready | 1 | Master->Slave | Response ready |

### Address Calculation Rules

#### Simplified Mode Address Generation

| Parameter | Formula | Description |
| --- | --- | --- |
| **First Address** | Must be bus-width aligned | Starting address |
| **Address N** | First\_Address + (N x Bus\_Width\_Bytes) | Address for beat N |
| **Alignment Check** | (Address % Bus\_Width\_Bytes) == 0 | Must always be true |

#### Flexible Mode Address Generation

| Parameter | Formula | Description |
| --- | --- | --- |
| **First Address** | Must be 4-byte aligned | Starting address |
| **Address N** | First\_Address + (N x Transfer\_Size) | Address for beat N |
| **Alignment Check** | (Address % 4) == 0 | Must always be true |
| **Progressive Alignment** | Choose largest size <= bytes\_to\_boundary | Optimization strategy |

#### 4KB Boundary Considerations (Both Modes)

| Validation Rule | Formula | Description |
| --- | --- | --- |
| **4KB Boundary** | Bursts cannot cross 4KB (0x1000) boundaries | AXI specification |
| **Max Burst Calculation** | Max\_Beats = (4KB - (Start\_Address % 4KB)) / Transfer\_Size | Burst limit |
| **Boundary Check** | Verify no 4KB crossings in burst | Mandatory validation |

### Write Strobe Generation

#### Simplified Mode Strobe Generation

| Bus Width | Strobe Pattern | Description |
| --- | --- | --- |
| **32-bit** | 4’b1111 | All bytes valid |
| **512-bit** | 64’hFFFFFFFFFFFFFFFF | All bytes valid |

#### Flexible Mode Strobe Generation

**From Chunk Enables (512-bit bus example):**

// Convert 16x32-bit chunk enables to 64x8-bit write strobes  
for (int chunk = 0; chunk < 16; chunk++) begin  
 if (chunk\_enable[chunk]) begin  
 w\_strb[chunk\*4 +: 4] = 4'hF; // 4 bytes per chunk  
 end  
end

**Alignment Transfer Examples:**

| Transfer Size | Chunk Pattern | Strobe Pattern | Description |
| --- | --- | --- | --- |
| **4 bytes** | 16’h0001 | 64’h000000000000000F | First 4 bytes |
| **16 bytes** | 16’h000F | 64’h00000000000000FF | First 16 bytes |
| **32 bytes** | 16’h00FF | 64’h0000000000FFFFFF | First 32 bytes |
| **64 bytes** | 16’hFFFF | 64’hFFFFFFFFFFFFFFFF | All 64 bytes |

### Response Codes

#### Response Code Specification

| Value | Name | Description | Simplified Mode Usage | Flexible Mode Usage |
| --- | --- | --- | --- | --- |
| **2’b00** | OKAY | Normal access success | Bus-width aligned access | 4-byte aligned access |
| **2’b01** | EXOKAY | Exclusive access success | Bus-width aligned exclusive | 4-byte aligned exclusive |
| **2’b10** | SLVERR | Slave error | Slave-specific error | Slave-specific error |
| **2’b11** | DECERR | Decode error | **Bus-width misalignment** | **4-byte misalignment** |

### Implementation Benefits

#### Simplified Mode Benefits

| Benefit Area | Simplification | Impact |
| --- | --- | --- |
| **Address Generation** | Simple increment by bus width | Minimal logic complexity |
| **Size Checking** | No dynamic size validation | No validation logic needed |
| **Strobe Generation** | All strobes always high | Trivial implementation |
| **Timing** | Predictable single-size transfers | Optimal timing closure |

#### Flexible Mode Benefits

| Benefit Area | Capability | Impact |
| --- | --- | --- |
| **Data Placement** | Arbitrary 4-byte aligned placement | Maximum flexibility |
| **Bus Utilization** | Progressive alignment optimization | High efficiency achieved |
| **Chunk Control** | Precise byte-level validity | Optimal memory utilization |
| **Alignment Strategy** | Automatic alignment to boundaries | Performance optimization |

#### Mode Selection Guidelines

| Interface Type | Recommended Mode | Rationale |
| --- | --- | --- |
| **High-bandwidth data** | **Flexible** | Maximize throughput, handle arbitrary alignment |
| **Control/status** | **Simplified** | Predictable timing, minimal complexity |
| **Descriptors** | **Simplified** | Fixed-size structures, simple implementation |
| **Programs** | **Simplified** | Single-word writes, minimal overhead |
| **Flags** | **Simplified** | Fixed-size status, predictable behavior |

### Validation Requirements

#### Simplified Mode Validation

| Validation Area | Requirements |
| --- | --- |
| **Address Alignment** | Verify all addresses aligned to full bus width |
| **Fixed Size** | Verify AxSIZE always matches DATA\_WIDTH |
| **Full Strobes** | Verify w\_strb is always all 1’s |
| **Burst Type** | Verify AxBURST is always 2’b01 |

#### Flexible Mode Validation

| Validation Area | Requirements |
| --- | --- |
| **Address Alignment** | Verify all addresses are 4-byte aligned |
| **Size Validation** | Verify AxSIZE matches actual transfer size |
| **Chunk Consistency** | Verify chunk enables match transfer size |
| **Strobe Generation** | Verify strobes generated correctly from chunks |
| **Progressive Alignment** | Verify alignment strategy optimization |
| **Boundary Checking** | Verify no 4KB boundary crossings |

#### Common Validation

| Validation Area | Requirements |
| --- | --- |
| **No Wraparound** | Verify addresses never wrap around |
| **Incrementing Only** | Verify AxBURST is always 2’b01 |
| **Response Handling** | Verify proper response generation |
| **Error Conditions** | Verify alignment violation responses |

### Performance Characteristics

#### Simplified Mode Performance

| Metric | Typical Value | Description |
| --- | --- | --- |
| **Latency** | 3 cycles | Address + Data + Response |
| **Throughput** | 1 transfer per clock | Sustained rate |
| **Efficiency** | 100% | Perfect bus utilization |
| **Complexity** | Minimal | Simple implementation |

#### Flexible Mode Performance

| Metric | Alignment Phase | Optimized Phase | Description |
| --- | --- | --- | --- |
| **Latency** | 3-15 cycles | 3 cycles | Variable based on alignment |
| **Throughput** | Variable | 1 transfer per clock | Depends on alignment pattern |
| **Efficiency** | 25-100% | 100% | Improves with alignment |
| **Complexity** | Moderate | Minimal | Progressive optimization |

#### Performance Optimization Strategy

**Flexible Mode Alignment Strategy:** 1. **Initial Phase**: Use largest possible transfer size for current alignment 2. **Progressive Phase**: Incrementally align to larger boundaries  
3. **Optimized Phase**: Use full bus-width transfers once aligned 4. **Result**: Achieve maximum efficiency while handling arbitrary starting addresses

This dual-mode approach provides the best of both worlds: simplified, predictable operation for control interfaces and flexible, high-performance operation for data interfaces. ## AXI4-Stream (AXIS4) Interface Specification and Assumptions

### Overview

This document defines the formal specification and assumptions for AXI4-Stream (AXIS4) interface implementations used in the RAPIDS system. AXIS4 provides high-bandwidth, unidirectional streaming data transfer with built-in flow control and packet framing capabilities.

### Interface Summary

#### Number of Interfaces

* **2 Master (Transmit) Interfaces**: Source data path and network master output
* **2 Slave (Receive) Interfaces**: Sink data path and network slave input

#### Interface Parameters

| Parameter | Description | Valid Values | Default |
| --- | --- | --- | --- |
| TDATA\_WIDTH | Stream data bus width in bits | 32, 64, 128, 256, 512, 1024 | 512 |
| TID\_WIDTH | Stream ID width in bits (optional) | 0-8 | 0 |
| TDEST\_WIDTH | Stream destination width in bits (optional) | 0-8 | 0 |
| TUSER\_WIDTH | User-defined sideband width in bits (optional) | 0-128 | 0 |
| TKEEP\_ENABLE | Enable TKEEP byte qualifier signals | 0, 1 | 1 |
| TSTRB\_ENABLE | Enable TSTRB byte strobe signals | 0, 1 | 0 |

#### Interface Configuration Summary

| Interface Type | Direction | TDATA Width | TID | TDEST | TUSER | TKEEP | Purpose |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Network Master (TX)** | Output | 512 bits | No | Optional | Optional | Yes | Transmit packets to network |
| **Network Slave (RX)** | Input | 512 bits | No | Optional | Optional | Yes | Receive packets from network |
| **Source Data Stream** | Output | 512 bits | No | No | Optional | Yes | Memory-to-network streaming |
| **Sink Data Stream** | Input | 512 bits | No | No | Optional | Yes | Network-to-memory streaming |

### Core Protocol Assumptions

#### AXI4-Stream Fundamentals

| Aspect | Requirement |
| --- | --- |
| **Transfer Protocol** | Valid-ready handshake on every transfer |
| **Data Flow** | Unidirectional streaming (master -> slave) |
| **Packet Framing** | TLAST signal indicates packet boundaries |
| **Flow Control** | Backpressure via TREADY signal |
| **Byte Granularity** | TKEEP indicates valid bytes within transfer |

#### Implementation Assumptions

##### Assumption 1: TKEEP-Based Byte Qualification

| Aspect | Requirement |
| --- | --- |
| **Byte Validity Rule** | TKEEP indicates which bytes of TDATA contain valid data |
| **512-bit bus (64 bytes)** | TKEEP[63:0] - one bit per byte |
| **Contiguous Bytes** | Valid bytes are always contiguous (no gaps) |
| **Alignment** | Valid bytes always start from TDATA[7:0] (byte 0) |
| **Rationale** | Simplifies data alignment and reduces complexity |
| **Benefit** | Eliminates data steering logic for non-contiguous bytes |

**TKEEP Encoding Examples (64-byte bus):**

| Transfer Size | TKEEP Value | Description |
| --- | --- | --- |
| **64 bytes** | 64’hFFFFFFFFFFFFFFFF | All bytes valid (full transfer) |
| **32 bytes** | 64’h00000000FFFFFFFF | First 32 bytes valid |
| **16 bytes** | 64’h000000000000FFFF | First 16 bytes valid |
| **8 bytes** | 64’h00000000000000FF | First 8 bytes valid |
| **4 bytes** | 64’h000000000000000F | First 4 bytes valid |
| **1 byte** | 64’h0000000000000001 | First byte valid |

##### Assumption 2: TLAST for Packet Boundaries

| Aspect | Requirement |
| --- | --- |
| **Packet Delimiter** | TLAST=1 on final transfer of packet |
| **Single-Beat Packets** | TLAST=1 for single-transfer packets |
| **Multi-Beat Packets** | TLAST=0 for all transfers except last |
| **Mandatory Signal** | TLAST required for all packet-based protocols |
| **Rationale** | Enables downstream packet processing and buffering |

##### Assumption 3: No TSTRB Usage

| Aspect | Requirement |
| --- | --- |
| **TSTRB Disabled** | TSTRB signals not used (TSTRB\_ENABLE=0) |
| **Byte Qualification** | TKEEP provides sufficient byte-level control |
| **Rationale** | RAPIDS data is always read-oriented (no write strobes needed) |
| **Benefit** | Reduces interface width and complexity |

##### Assumption 4: Optional TID and TDEST

| Aspect | Requirement |
| --- | --- |
| **TID Usage** | Transaction ID not required for RAPIDS use cases |
| **TDEST Usage** | Destination routing optional (set by interface type) |
| **Default Configuration** | TID\_WIDTH=0, TDEST\_WIDTH=0 for simple streaming |
| **Rationale** | RAPIDS uses point-to-point connections, no routing needed |

### Master (Transmit) Interface Specification

#### AXIS Master Signals

| Signal | Width | Direction | Required | Description |
| --- | --- | --- | --- | --- |
| m\_axis\_tdata | TDATA\_WIDTH | Master->Slave | Yes | Stream data payload |
| m\_axis\_tvalid | 1 | Master->Slave | Yes | Data valid indicator |
| m\_axis\_tready | 1 | Slave->Master | Yes | Ready for data (backpressure) |
| m\_axis\_tlast | 1 | Master->Slave | Yes | Last transfer in packet |
| m\_axis\_tkeep | TDATA\_WIDTH/8 | Master->Slave | Yes | Byte qualifier (valid bytes) |
| m\_axis\_tid | TID\_WIDTH | Master->Slave | Optional | Transaction ID |
| m\_axis\_tdest | TDEST\_WIDTH | Master->Slave | Optional | Routing destination |
| m\_axis\_tuser | TUSER\_WIDTH | Master->Slave | Optional | User sideband data |

#### Master Transfer Rules

| Rule | Requirement | Description |
| --- | --- | --- |
| **Transfer Occurrence** | Transfer occurs when TVALID=1 AND TREADY=1 | Standard AXIS handshake |
| **TVALID Assertion** | Master can assert TVALID independently of TREADY | Master controls data availability |
| **TVALID Stability** | Once TVALID=1, all T\* signals must remain stable until TREADY=1 | Data integrity requirement |
| **TREADY Dependency** | Slave can assert TREADY based on TVALID state | Backpressure control |
| **TKEEP Alignment** | Valid bytes start from byte 0, must be contiguous | Per Assumption 1 |
| **TLAST Requirement** | TLAST=1 on final beat of every packet | Per Assumption 2 |

### Slave (Receive) Interface Specification

#### AXIS Slave Signals

| Signal | Width | Direction | Required | Description |
| --- | --- | --- | --- | --- |
| s\_axis\_tdata | TDATA\_WIDTH | Master->Slave | Yes | Stream data payload |
| s\_axis\_tvalid | 1 | Master->Slave | Yes | Data valid indicator |
| s\_axis\_tready | 1 | Slave->Master | Yes | Ready for data (backpressure) |
| s\_axis\_tlast | 1 | Master->Slave | Yes | Last transfer in packet |
| s\_axis\_tkeep | TDATA\_WIDTH/8 | Master->Slave | Yes | Byte qualifier (valid bytes) |
| s\_axis\_tid | TID\_WIDTH | Master->Slave | Optional | Transaction ID |
| s\_axis\_tdest | TDEST\_WIDTH | Master->Slave | Optional | Routing destination |
| s\_axis\_tuser | TUSER\_WIDTH | Master->Slave | Optional | User sideband data |

#### Slave Flow Control

| Aspect | Behavior | Description |
| --- | --- | --- |
| **TREADY=1** | Slave ready to accept data | Normal operation |
| **TREADY=0** | Slave cannot accept data (backpressure) | Flow control active |
| **TREADY Timing** | Can be asserted/deasserted on any cycle | Dynamic flow control |
| **Buffer Management** | TREADY reflects downstream buffer availability | Prevents overflow |

### Packet Structure and Framing

#### Single-Beat Packet

**Packet with data <= 64 bytes (fits in one transfer):**

Clock: ─┐ ┌─┐ ┌─  
 └─┘ └─┘  
  
TVALID: ──┐ ┌───  
 └─┘  
  
TREADY: ────────  
 (always ready)  
  
TDATA: [Packet Data (64 bytes max)]  
  
TKEEP: [Valid byte mask (e.g., 64'h00000000FFFFFFFF for 32 bytes)]  
  
TLAST: ──┐ ┌───  
 └─┘  
  
Transfer: Single beat completes entire packet

#### Multi-Beat Packet

**Packet with data > 64 bytes (requires multiple transfers):**

Clock: ─┐ ┌─┐ ┌─┐ ┌─┐ ┌─  
 └─┘ └─┘ └─┘ └─┘  
  
TVALID: ──┐ ┌───  
 └─────────┘  
  
TREADY: ────────────────  
 (always ready)  
  
TDATA: [Beat 0][Beat 1][Beat 2][Beat 3]  
 64B 64B 64B 32B (partial)  
  
TKEEP: [64'hFFFF...][64'hFFFF...][64'hFFFF...][64'h00000000FFFFFFFF]  
 All valid All valid All valid 32 bytes valid  
  
TLAST: ──────────────┐ ┌───  
 └─┘  
  
Transfer: 4 beats (256 bytes total)  
- Beats 0-2: Full 64-byte transfers, TLAST=0  
- Beat 3: Partial 32-byte transfer, TLAST=1

#### Packet with Backpressure

**Backpressure applied mid-packet:**

Clock: ─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─  
 └─┘ └─┘ └─┘ └─┘ └─┘ └─┘  
  
TVALID: ──┐ ┌───  
 └───────────────────┘  
  
TREADY: ──┐ ┌─┐ ┌───┐ ┌─────  
 └─┘ └─┘ └─┘  
  
TDATA: [Beat 0] (stall) [Beat 1] (stall) [Beat 2]  
  
TKEEP: [Valid] (held) [Valid] (held) [Valid]  
  
TLAST: ──────────────────────────┐ ┌───  
 └─┘  
  
Transfer: 3 beats with backpressure  
- Beat 0: Transfers immediately  
- Stall: TREADY=0, TVALID held high, data held stable  
- Beat 1: Transfers after 2 stall cycles  
- Stall: Another 1-cycle stall  
- Beat 2: Final beat transfers, TLAST=1

**Key Observations:** 1. Master must hold TVALID=1 and all T\* signals stable during backpressure 2. Slave controls flow via TREADY signal 3. Transfer only occurs when both TVALID=1 AND TREADY=1

### TUSER Sideband Signaling

#### Purpose and Usage

| Aspect | Description |
| --- | --- |
| **Purpose** | Carry packet metadata alongside data stream |
| **Scope** | Valid on first beat of packet only (when new packet starts) |
| **Width** | Application-specific (0-128 bits) |
| **Examples** | Packet type, priority, timestamp, sequence number |

#### RAPIDS-Specific TUSER Encoding (Example)

**64-bit TUSER encoding for network packets:**

| Bits | Field | Description |
| --- | --- | --- |
| [7:0] | **Packet Type** | 0x01=Data, 0x02=Control, 0x03=Status |
| [15:8] | **Priority** | 0-255 priority level |
| [31:16] | **Sequence Number** | Per-channel sequence tracking |
| [47:32] | **Channel ID** | Source/destination channel identifier |
| [63:48] | **Reserved** | Future use |

**Usage Pattern:** - TUSER valid only on first beat of packet (can be ignored on subsequent beats) - Downstream logic captures TUSER on first beat, uses for entire packet - Simplifies metadata handling (no per-beat metadata processing)

### Flow Control and Backpressure

#### Backpressure Mechanisms

| Mechanism | Implementation | Description |
| --- | --- | --- |
| **Immediate Backpressure** | TREADY=0 for N cycles | Slave cannot accept data |
| **Conditional Backpressure** | TREADY toggles based on buffer state | Dynamic flow control |
| **Sustained Backpressure** | TREADY=0 for extended period | Upstream buffer fills |

#### Backpressure Propagation

Source SRAM Control -> AXIS Master -> AXIS Slave -> Sink SRAM Control  
 ↑ ↓  
 TREADY Backpressure  
 (flow control)

**Propagation Rules:** 1. Slave asserts TREADY=0 when downstream buffer nearly full 2. Master stops sending data (TVALID may remain high, data held) 3. Backpressure propagates to source (SRAM read stalls) 4. System self-regulates to prevent buffer overflow

#### Buffer Depth Considerations

| Buffer Type | Recommended Depth | Rationale |
| --- | --- | --- |
| **AXIS Input FIFO** | 16-32 beats | Absorb backpressure latency |
| **AXIS Output FIFO** | 16-32 beats | Smooth bursty traffic |
| **Packet Buffer** | 4-8 packets | Handle multi-packet scenarios |

### Reset Behavior

#### Reset Requirements

| Reset Phase | Requirement | Description |
| --- | --- | --- |
| **Active Reset** | aresetn is active-low reset signal | Standard AXI reset |
| **TVALID During Reset** | TVALID=0 when aresetn=0 | No spurious transfers |
| **TREADY During Reset** | TREADY can be 0 or 1 (don’t care) | Slave state undefined |
| **State Clearing** | All FIFOs/buffers flushed during reset | Clean startup |
| **Post-Reset** | TVALID=0 for at least 1 cycle after reset release | Stable initialization |

#### Reset Timing

Clock: ─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─  
 └─┘ └─┘ └─┘ └─┘ └─┘ └─┘  
  
aresetn: ──────┐ ┌─────  
 └───────────┘  
  
TVALID: ──────────────────┐ ┌───  
 └─┘  
 (stable after reset)  
  
TREADY: ──────────────────? ? ?──  
 (don't care initially)  
  
Note: TVALID must be 0 during and immediately after reset

### Implementation Benefits

#### Streaming Efficiency

| Benefit Area | Advantage | Impact |
| --- | --- | --- |
| **Continuous Streaming** | No address overhead (unlike AXI4) | Maximum throughput |
| **Simple Handshake** | Valid-ready protocol only | Minimal control logic |
| **Burst Transfers** | Multi-beat packets for efficiency | High bandwidth utilization |
| **Flow Control** | Built-in backpressure mechanism | Prevents data loss |

#### Packet-Based Processing

| Benefit Area | Advantage | Impact |
| --- | --- | --- |
| **Packet Framing** | TLAST delimits packets | Enables packet-level processing |
| **Byte Granularity** | TKEEP handles partial transfers | Supports variable-length packets |
| **Metadata Support** | TUSER carries packet info | Rich packet classification |

#### Resource Efficiency

| Benefit Area | Simplification | Impact |
| --- | --- | --- |
| **No Address Logic** | Streaming-only interface | Reduced logic complexity |
| **No Transaction IDs** | Point-to-point connections | Simplified state machines |
| **Optional Signals** | TID/TDEST/TUSER as needed | Minimal interface width |
| **Byte Alignment** | Contiguous bytes only | No complex data steering |

### Timing Requirements

#### Setup and Hold Times

| Timing Parameter | Requirement | Description |
| --- | --- | --- |
| **TVALID to TREADY** | Setup time: 0 ns | Combinational ready allowed |
| **TREADY to TVALID** | Setup time: 1 clock cycle | TVALID registered before TREADY check |
| **T\* Signal Stability** | Hold until TREADY=1 | Data integrity during backpressure |

#### Clock Domain Considerations

| Scenario | Requirement | Solution |
| --- | --- | --- |
| **Synchronous Operation** | Single clock domain | Direct connection |
| **Asynchronous Operation** | Different clock domains | Insert AXIS async FIFO |
| **Clock Frequency Ratio** | Producer faster than consumer | Backpressure handles rate mismatch |

### Validation Requirements

#### Functional Validation

| Validation Area | Requirements |
| --- | --- |
| **Valid-Ready Handshake** | Verify all transfers occur only when TVALID=1 AND TREADY=1 |
| **TKEEP Encoding** | Verify byte validity matches TKEEP pattern |
| **TLAST Assertion** | Verify TLAST=1 on final beat of every packet |
| **Backpressure Handling** | Verify master holds TVALID and T\* signals during TREADY=0 |
| **Packet Integrity** | Verify multi-beat packets reconstruct correctly |
| **Byte Alignment** | Verify valid bytes are contiguous starting from byte 0 |

#### Timing Validation

| Validation Area | Requirements |
| --- | --- |
| **Signal Stability** | Verify T\* signals stable when TVALID=1 until TREADY=1 |
| **Reset Behavior** | Verify TVALID=0 during and after reset |
| **Clock Crossing** | Verify async FIFO metastability protection (if used) |

#### Stress Testing

| Test Type | Description | Expected Behavior |
| --- | --- | --- |
| **Sustained Backpressure** | TREADY=0 for 100+ cycles | Master waits without data corruption |
| **Rapid Backpressure Toggle** | TREADY toggles every cycle | Transfer rate adapts correctly |
| **Maximum Throughput** | TREADY=1 always | Full bandwidth utilization |
| **Single-Beat Packets** | All packets fit in one beat | TLAST=1 on every transfer |
| **Large Multi-Beat Packets** | Packets spanning 100+ beats | Correct packet reconstruction |

### Example Transactions

#### Example 1: Single-Beat Packet (32 bytes)

**Configuration:** - TDATA\_WIDTH = 512 bits (64 bytes) - Packet size = 32 bytes - Single transfer

**Signals:**

Clock cycle: ─┐ ┌─  
 └─┘  
  
TVALID: ──┐ ┌───  
 └─┘  
  
TREADY: ──────────  
 (ready)  
  
TDATA: [32 bytes of packet data | 32 bytes unused]  
  
TKEEP: 64'h00000000FFFFFFFF (first 32 bytes valid)  
  
TLAST: ──┐ ┌───  
 └─┘  
  
TUSER: [Packet metadata - type, priority, etc.]

**Result:** Entire packet transfers in single beat.

#### Example 2: Multi-Beat Packet with Backpressure (200 bytes)

**Configuration:** - TDATA\_WIDTH = 512 bits (64 bytes) - Packet size = 200 bytes - Requires 4 beats: 64 + 64 + 64 + 8 bytes

**Signals:**

Clock cycle: ─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─  
 └─┘ └─┘ └─┘ └─┘ └─┘ └─┘  
  
TVALID: ──┐ ┌───  
 └───────────────────────┘  
  
TREADY: ──┐ ┌─────┐ ┌─┐ ┌─┐ ┌───  
 └─┘ └─┘ └─┘ └─┘  
 (backpressure cycles 2-3, 5)  
  
Beat 0: [64 bytes] TKEEP=64'hFFFF..., TLAST=0  
 Transfers immediately  
  
Stall: (cycles 2-3, TREADY=0, master holds data)  
  
Beat 1: [64 bytes] TKEEP=64'hFFFF..., TLAST=0  
 Transfers after stall  
  
Stall: (cycle 5, TREADY=0 again)  
  
Beat 2: [64 bytes] TKEEP=64'hFFFF..., TLAST=0  
 Transfers after stall  
  
Beat 3: [8 bytes] TKEEP=64'h00000000000000FF, TLAST=1  
 Final beat transfers  
  
TUSER: [Valid on beat 0 only]

**Result:** 200-byte packet transfers across 4 beats with intermittent backpressure. Total transfer takes 7 clock cycles (4 beats + 3 stall cycles).

#### Example 3: Back-to-Back Packets

**Configuration:** - Two packets: Packet A (64 bytes), Packet B (128 bytes) - No gaps between packets

**Signals:**

Clock cycle: ─┐ ┌─┐ ┌─┐ ┌─  
 └─┘ └─┘ └─┘  
  
TVALID: ──┐ ┌───  
 └─────────┘  
  
TREADY: ────────────────  
  
Beat 0: [Packet A - 64 bytes] TKEEP=64'hFFFF..., TLAST=1  
 TUSER=[Packet A metadata]  
  
Beat 1: [Packet B beat 0 - 64 bytes] TKEEP=64'hFFFF..., TLAST=0  
 TUSER=[Packet B metadata]  
  
Beat 2: [Packet B beat 1 - 64 bytes] TKEEP=64'hFFFF..., TLAST=1  
 TUSER=(ignored, mid-packet)

**Result:** Back-to-back packets transfer efficiently without idle cycles. TUSER updates on first beat of each new packet.

### Common Use Cases

#### Network Interface Applications

| Use Case | Configuration | Description |
| --- | --- | --- |
| **Packet Transmission** | 512-bit TDATA, TKEEP, TLAST, TUSER | High-bandwidth network TX |
| **Packet Reception** | 512-bit TDATA, TKEEP, TLAST, TUSER | High-bandwidth network RX |
| **Streaming DMA** | 512-bit TDATA, TKEEP, TLAST | Memory-to-network data transfer |
| **Flow-Controlled Streaming** | Dynamic TREADY | Backpressure-aware streaming |

#### Data Path Integration

| Integration Pattern | Description |
| --- | --- |
| **Source Path** | AXI4 Read -> SRAM -> AXIS Master -> Network |
| **Sink Path** | Network -> AXIS Slave -> SRAM -> AXI4 Write |
| **Loopback** | AXIS Master -> AXIS Slave (testing) |
| **Multi-Stage Pipeline** | AXIS -> Processing -> AXIS (chained) |

#### Performance Characteristics

| Metric | Typical Value | Description |
| --- | --- | --- |
| **Latency** | 1-2 cycles | TVALID assertion to TREADY response |
| **Throughput** | 1 beat per clock | Sustained rate (no backpressure) |
| **Efficiency** | 95-100% | With occasional backpressure |
| **Packet Rate** | Dependent on size | 64-byte packets: ~10Gbps @ 200MHz |

### RAPIDS-Specific Considerations

#### Interface Assignments

| RAPIDS Block | AXIS Role | Direction | Width | Notes |
| --- | --- | --- | --- | --- |
| **Network Slave** | Slave (RX) | Input | 512-bit | Receives packets from network |
| **Network Master** | Master (TX) | Output | 512-bit | Transmits packets to network |
| **Source SRAM Control** | Master (TX) | Output | 512-bit | Streams data from SRAM |
| **Sink SRAM Control** | Slave (RX) | Input | 512-bit | Receives data to SRAM |

#### Packet Processing Flow

**Sink Path (Network -> Memory):**

Network AXIS Input (512-bit packets)  
 ↓ TVALID/TREADY/TLAST/TKEEP  
Network Slave (packet validation)  
 ↓ Internal handshake  
Sink SRAM Control (buffering)  
 ↓ AXI4 Write  
System Memory

**Source Path (Memory -> Network):**

System Memory  
 ↓ AXI4 Read  
Source SRAM Control (buffering)  
 ↓ Internal handshake  
Network Master (packet formation)  
 ↓ TVALID/TREADY/TLAST/TKEEP  
Network AXIS Output (512-bit packets)

#### Buffer Sizing Guidelines

| Buffer Location | Recommended Size | Rationale |
| --- | --- | --- |
| **Network Input FIFO** | 32 beats (2KB) | Absorb network burst traffic |
| **Network Output FIFO** | 32 beats (2KB) | Smooth AXI4 read latency |
| **SRAM Depth** | 1024-4096 entries | Match typical packet sizes |

### Comparison with Other AXIS Variants

#### AXIS vs Full AXI4

| Feature | AXIS | Full AXI4 |
| --- | --- | --- |
| **Addressing** | No addressing (streaming) | Full address bus |
| **Channels** | Single data channel | 5 independent channels (AR, R, AW, W, B) |
| **Transaction IDs** | Optional (rarely used) | Mandatory for out-of-order |
| **Burst Support** | Continuous streaming | Fixed-length bursts |
| **Complexity** | Low | High |
| **Use Case** | Streaming data | Random-access memory |

#### AXIS Configuration Trade-offs

| Configuration | Advantages | Disadvantages |
| --- | --- | --- |
| **Wide Data Path (512-bit)** | High throughput, fewer transfers | More routing resources |
| **Narrow Data Path (64-bit)** | Simpler routing, lower resource | Lower throughput, more transfers |
| **With TUSER** | Rich metadata support | Increased interface width |
| **Without TUSER** | Minimal interface | Limited metadata capability |

### Appendix: Signal Quick Reference

#### Mandatory Signals

| Signal | Width | Source | Description |
| --- | --- | --- | --- |
| TDATA | TDATA\_WIDTH | Master | Streaming data payload |
| TVALID | 1 | Master | Data valid indicator |
| TREADY | 1 | Slave | Ready for data (backpressure) |
| TLAST | 1 | Master | Last transfer in packet |

#### Optional Signals

| Signal | Width | Source | When to Use |
| --- | --- | --- | --- |
| TKEEP | TDATA\_WIDTH/8 | Master | Byte-level data validity (recommended) |
| TSTRB | TDATA\_WIDTH/8 | Master | Write strobes (rarely used) |
| TID | TID\_WIDTH | Master | Transaction routing/identification |
| TDEST | TDEST\_WIDTH | Master | Destination routing |
| TUSER | TUSER\_WIDTH | Master | User-defined sideband metadata |

#### Signal Relationships

TVALID=1 + TREADY=1 -> Transfer occurs  
TVALID=1 + TREADY=0 -> Master waits (holds all T\* signals)  
TVALID=0 + TREADY=? -> No transfer (TREADY don't care)  
TLAST=1 -> Final beat of packet  
TKEEP[n]=1 -> Byte n of TDATA is valid  
TKEEP[n]=0 -> Byte n of TDATA is invalid (ignored)

**Next:** [Chapter 3 - Interface 5: MonBus](05_monbus_interface_spec.md) ## Monitor Bus Architecture and Event Code Organization

### Overview

The Monitor Bus architecture provides a unified, scalable framework for monitoring and error reporting across multiple bus protocols in complex SoC designs. This system supports AXI, APB, Network (Mesh Network on Chip), ARB (Arbiter), CORE, and custom protocols through a standardized 64-bit packet format with protocol-aware event categorization.

### Interface Summary

#### Number of Interfaces

* **1 Monitor Bus Output Interface**: Unified 64-bit packet stream
* **Multiple Protocol Input Interfaces**: AXI, APB, Network, ARB, CORE, Custom protocol monitors
* **Local Memory Interface**: Error/interrupt packet storage
* **External Memory Interface**: Bulk packet storage

#### Interface Parameters

| Parameter | Description | Valid Values | Default |
| --- | --- | --- | --- |
| PACKET\_WIDTH | Monitor bus packet width | 64 | 64 |
| PROTOCOL\_WIDTH | Protocol identifier width | 3 | 3 |
| EVENT\_CODE\_WIDTH | Event code width | 4 | 4 |
| PACKET\_TYPE\_WIDTH | Packet type width | 4 | 4 |
| CHANNEL\_ID\_WIDTH | Channel identifier width | 6 | 6 |
| UNIT\_ID\_WIDTH | Unit identifier width | 4 | 4 |
| AGENT\_ID\_WIDTH | Agent identifier width | 8 | 8 |
| EVENT\_DATA\_WIDTH | Event data width | 35 | 35 |

### Core Design Assumptions

#### Assumption 1: Hierarchical Event Organization

| Aspect | Requirement |
| --- | --- |
| **Organization Rule** | Protocol -> Packet Type -> Event Code hierarchy |
| **Event Space** | Each protocol x packet type combination has exactly 16 event codes |
| **Mapping** | 1:1 mapping between packet types and event codes |
| **Rationale** | Provides clear, scalable event organization |

#### Assumption 2: Protocol Isolation

| Aspect | Requirement |
| --- | --- |
| **Isolation Rule** | Each protocol owns its event space |
| **Conflict Prevention** | No cross-protocol event conflicts |
| **Independent Evolution** | Protocols can evolve independently |
| **Rationale** | Prevents interference and enables protocol-specific optimization |

#### Assumption 3: Two-Tier Memory Architecture

| Aspect | Requirement |
| --- | --- |
| **Local Storage** | Critical events (errors/interrupts) stored locally |
| **External Storage** | Non-critical events routed to external memory |
| **Routing Decision** | Based on packet type configuration |
| **Rationale** | Balances immediate access with bulk storage needs |

#### Assumption 4: Configurable Packet Routing

| Aspect | Requirement |
| --- | --- |
| **Routing Rule** | Different packet types can route to different destinations |
| **Configuration** | Base/limit registers define routing per packet type |
| **Priority Support** | Configurable priority levels per packet type |
| **Rationale** | Enables flexible memory allocation and access patterns |

### Interface Signal Specification

#### Monitor Bus Output Interface

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| mon\_packet | 64 | Monitor->System | Monitor packet data |
| mon\_valid | 1 | Monitor->System | Packet valid signal |
| mon\_ready | 1 | System->Monitor | Ready to accept packet |
| mon\_error | 1 | Monitor->System | Monitor error condition |

#### Protocol Input Interfaces

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| axi\_event | 64 | AXI Monitor->Bus | AXI event packet |
| axi\_event\_valid | 1 | AXI Monitor->Bus | AXI event valid |
| axi\_event\_ready | 1 | Bus->AXI Monitor | Ready for AXI event |
| apb\_event | 64 | APB Monitor->Bus | APB event packet |
| apb\_event\_valid | 1 | APB Monitor->Bus | APB event valid |
| apb\_event\_ready | 1 | Bus->APB Monitor | Ready for APB event |
| network\_event | 64 | Network Monitor->Bus | Network event packet |
| network\_event\_valid | 1 | Network Monitor->Bus | Network event valid |
| network\_event\_ready | 1 | Bus->Network Monitor | Ready for Network event |
| arb\_event | 64 | ARB Monitor->Bus | ARB event packet |
| arb\_event\_valid | 1 | ARB Monitor->Bus | ARB event valid |
| arb\_event\_ready | 1 | Bus->ARB Monitor | Ready for ARB event |
| core\_event | 64 | CORE Monitor->Bus | CORE event packet |
| core\_event\_valid | 1 | CORE Monitor->Bus | CORE event valid |
| core\_event\_ready | 1 | Bus->CORE Monitor | Ready for CORE event |

#### Control and Status Signals

| Signal | Width | Direction | Description |
| --- | --- | --- | --- |
| clk | 1 | Input | System clock |
| resetn | 1 | Input | Active-low reset |
| monitor\_enable | 1 | Input | Global monitor enable |
| packet\_type\_enables | 16 | Input | Per-type enable bits |
| local\_memory\_full | 1 | Output | Local memory full flag |
| external\_memory\_error | 1 | Output | External memory error |

### Packet Format and Field Allocation

#### 64-bit Monitor Bus Packet Structure

| Field | Bits | Width | Description |
| --- | --- | --- | --- |
| **Packet Type** | [63:60] | 4 | Event category (Error, Completion, etc.) |
| **Protocol** | [59:57] | 3 | Bus protocol (AXI=0, Network=1, APB=2, ARB=3, CORE=4) |
| **Event Code** | [56:53] | 4 | Specific events within category |
| **Channel ID** | [52:47] | 6 | Transaction/channel identifier |
| **Unit ID** | [46:43] | 4 | Subsystem identifier |
| **Agent ID** | [42:35] | 8 | Module identifier |
| **Event Data** | [34:0] | 35 | Event-specific payload |

#### Packet Type Definitions

| Value | Name | Purpose | Applicable Protocols |
| --- | --- | --- | --- |
| **0x0** | Error | Protocol violations, response errors | All |
| **0x1** | Completion | Successful transaction completion | All |
| **0x2** | Threshold | Threshold crossed events | All |
| **0x3** | Timeout | Timeout conditions | All |
| **0x4** | Performance | Performance metrics | All |
| **0x5** | Credit | Credit management | Network only |
| **0x6** | Channel | Channel status | Network only |
| **0x7** | Stream | Stream events | Network only |
| **0x8** | Address Match | Address matching | AXI only |
| **0x9** | APB Specific | APB protocol events | APB only |
| **0xA-0xE** | Reserved | Future expansion | - |
| **0xF** | Debug | Debug and trace events | All |

### Protocol-Specific Event Codes

#### AXI Protocol Events

##### Error Events (PktTypeError + PROTOCOL\_AXI)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | AXI\_ERR\_RESP\_SLVERR | Slave error response |
| **0x1** | AXI\_ERR\_RESP\_DECERR | Decode error response |
| **0x2** | AXI\_ERR\_DATA\_ORPHAN | Data without command |
| **0x3** | AXI\_ERR\_RESP\_ORPHAN | Response without transaction |
| **0x4** | AXI\_ERR\_PROTOCOL | Protocol violation |
| **0x5** | AXI\_ERR\_BURST\_LENGTH | Invalid burst length |
| **0x6** | AXI\_ERR\_BURST\_SIZE | Invalid burst size |
| **0x7** | AXI\_ERR\_BURST\_TYPE | Invalid burst type |
| **0x8** | AXI\_ERR\_ID\_COLLISION | ID collision detected |
| **0x9** | AXI\_ERR\_WRITE\_BEFORE\_ADDR | Write data before address |
| **0xA** | AXI\_ERR\_RESP\_BEFORE\_DATA | Response before data complete |
| **0xB** | AXI\_ERR\_LAST\_MISSING | Missing LAST signal |
| **0xC** | AXI\_ERR\_STROBE\_ERROR | Write strobe error |
| **0xD** | AXI\_ERR\_RESERVED\_D | Reserved |
| **0xE** | AXI\_ERR\_RESERVED\_E | Reserved |
| **0xF** | AXI\_ERR\_USER\_DEFINED | User-defined error |

##### Timeout Events (PktTypeTimeout + PROTOCOL\_AXI)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | AXI\_TIMEOUT\_CMD | Command/Address timeout |
| **0x1** | AXI\_TIMEOUT\_DATA | Data timeout |
| **0x2** | AXI\_TIMEOUT\_RESP | Response timeout |
| **0x3** | AXI\_TIMEOUT\_HANDSHAKE | Handshake timeout |
| **0x4** | AXI\_TIMEOUT\_BURST | Burst completion timeout |
| **0x5** | AXI\_TIMEOUT\_EXCLUSIVE | Exclusive access timeout |
| **0x6-0xE** | Reserved | Future expansion |
| **0xF** | AXI\_TIMEOUT\_USER\_DEFINED | User-defined timeout |

##### Performance Events (PktTypePerf + PROTOCOL\_AXI)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | AXI\_PERF\_ADDR\_LATENCY | Address phase latency |
| **0x1** | AXI\_PERF\_DATA\_LATENCY | Data phase latency |
| **0x2** | AXI\_PERF\_RESP\_LATENCY | Response phase latency |
| **0x3** | AXI\_PERF\_TOTAL\_LATENCY | Total transaction latency |
| **0x4** | AXI\_PERF\_THROUGHPUT | Transaction throughput |
| **0x5** | AXI\_PERF\_ERROR\_RATE | Error rate |
| **0x6** | AXI\_PERF\_ACTIVE\_COUNT | Active transaction count |
| **0x7** | AXI\_PERF\_BANDWIDTH\_UTIL | Bandwidth utilization |
| **0x8** | AXI\_PERF\_QUEUE\_DEPTH | Average queue depth |
| **0x9** | AXI\_PERF\_BURST\_EFFICIENCY | Burst efficiency metric |
| **0xA-0xE** | Reserved | Future expansion |
| **0xF** | AXI\_PERF\_USER\_DEFINED | User-defined performance |

#### APB Protocol Events

##### Error Events (PktTypeError + PROTOCOL\_APB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | APB\_ERR\_PSLVERR | Peripheral slave error |
| **0x1** | APB\_ERR\_SETUP\_VIOLATION | Setup phase protocol violation |
| **0x2** | APB\_ERR\_ACCESS\_VIOLATION | Access phase protocol violation |
| **0x3** | APB\_ERR\_STROBE\_ERROR | Write strobe error |
| **0x4** | APB\_ERR\_ADDR\_DECODE | Address decode error |
| **0x5** | APB\_ERR\_PROT\_VIOLATION | Protection violation (PPROT) |
| **0x6** | APB\_ERR\_ENABLE\_ERROR | Enable phase error |
| **0x7** | APB\_ERR\_READY\_ERROR | PREADY protocol error |
| **0x8-0xE** | Reserved | Future expansion |
| **0xF** | APB\_ERR\_USER\_DEFINED | User-defined error |

##### Timeout Events (PktTypeTimeout + PROTOCOL\_APB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | APB\_TIMEOUT\_SETUP | Setup phase timeout |
| **0x1** | APB\_TIMEOUT\_ACCESS | Access phase timeout |
| **0x2** | APB\_TIMEOUT\_ENABLE | Enable phase timeout (PREADY stuck) |
| **0x3** | APB\_TIMEOUT\_PREADY\_STUCK | PREADY stuck low |
| **0x4** | APB\_TIMEOUT\_TRANSFER | Overall transfer timeout |
| **0x5-0xE** | Reserved | Future expansion |
| **0xF** | APB\_TIMEOUT\_USER\_DEFINED | User-defined timeout |

##### Completion Events (PktTypeCompletion + PROTOCOL\_APB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | APB\_COMPL\_TRANS\_COMPLETE | Transaction completed |
| **0x1** | APB\_COMPL\_READ\_COMPLETE | Read transaction complete |
| **0x2** | APB\_COMPL\_WRITE\_COMPLETE | Write transaction complete |
| **0x3-0xE** | Reserved | Future expansion |
| **0xF** | APB\_COMPL\_USER\_DEFINED | User-defined completion |

##### Threshold Events (PktTypeThreshold + PROTOCOL\_APB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | APB\_THRESH\_LATENCY | APB latency threshold |
| **0x1** | APB\_THRESH\_ERROR\_RATE | APB error rate threshold |
| **0x2** | APB\_THRESH\_ACCESS\_COUNT | Access count threshold |
| **0x3** | APB\_THRESH\_BANDWIDTH | Bandwidth threshold |
| **0x4-0xE** | Reserved | Future expansion |
| **0xF** | APB\_THRESH\_USER\_DEFINED | User-defined threshold |

##### Performance Events (PktTypePerf + PROTOCOL\_APB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | APB\_PERF\_READ\_LATENCY | Read transaction latency |
| **0x1** | APB\_PERF\_WRITE\_LATENCY | Write transaction latency |
| **0x2** | APB\_PERF\_THROUGHPUT | Transaction throughput |
| **0x3** | APB\_PERF\_ERROR\_RATE | Error rate |
| **0x4** | APB\_PERF\_ACTIVE\_COUNT | Active transaction count |
| **0x5** | APB\_PERF\_COMPLETED\_COUNT | Completed transaction count |
| **0x6-0xE** | Reserved | Future expansion |
| **0xF** | APB\_PERF\_USER\_DEFINED | User-defined performance |

##### Debug Events (PktTypeDebug + PROTOCOL\_APB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | APB\_DEBUG\_STATE\_CHANGE | APB state changed |
| **0x1** | APB\_DEBUG\_SETUP\_PHASE | Setup phase event |
| **0x2** | APB\_DEBUG\_ACCESS\_PHASE | Access phase event |
| **0x3** | APB\_DEBUG\_ENABLE\_PHASE | Enable phase event |
| **0x4** | APB\_DEBUG\_PSEL\_TRACE | PSEL trace |
| **0x5** | APB\_DEBUG\_PENABLE\_TRACE | PENABLE trace |
| **0x6** | APB\_DEBUG\_PREADY\_TRACE | PREADY trace |
| **0x7** | APB\_DEBUG\_PPROT\_TRACE | PPROT trace |
| **0x8** | APB\_DEBUG\_PSTRB\_TRACE | PSTRB trace |
| **0x9-0xE** | Reserved | Future expansion |
| **0xF** | APB\_DEBUG\_USER\_DEFINED | User-defined debug |

#### Network Protocol Events

##### Error Events (PktTypeError + PROTOCOL\_MNOC)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | NETWORK\_ERR\_PARITY | Parity error |
| **0x1** | NETWORK\_ERR\_PROTOCOL | Protocol violation |
| **0x2** | NETWORK\_ERR\_OVERFLOW | Buffer/Credit overflow |
| **0x3** | NETWORK\_ERR\_UNDERFLOW | Buffer/Credit underflow |
| **0x4** | NETWORK\_ERR\_ORPHAN | Orphaned packet/ACK |
| **0x5** | NETWORK\_ERR\_INVALID | Invalid type/channel/payload |
| **0x6** | NETWORK\_ERR\_HEADER\_CRC | Header CRC error |
| **0x7** | NETWORK\_ERR\_PAYLOAD\_CRC | Payload CRC error |
| **0x8** | NETWORK\_ERR\_SEQUENCE | Sequence number error |
| **0x9** | NETWORK\_ERR\_ROUTE | Routing error |
| **0xA** | NETWORK\_ERR\_DEADLOCK | Deadlock detected |
| **0xB-0xE** | Reserved | Future expansion |
| **0xF** | NETWORK\_ERR\_USER\_DEFINED | User-defined error |

##### Credit Events (PktTypeCredit + PROTOCOL\_MNOC)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | NETWORK\_CREDIT\_ALLOCATED | Credits allocated |
| **0x1** | NETWORK\_CREDIT\_CONSUMED | Credits consumed |
| **0x2** | NETWORK\_CREDIT\_RETURNED | Credits returned |
| **0x3** | NETWORK\_CREDIT\_OVERFLOW | Credit overflow detected |
| **0x4** | NETWORK\_CREDIT\_UNDERFLOW | Credit underflow detected |
| **0x5** | NETWORK\_CREDIT\_EXHAUSTED | All credits exhausted |
| **0x6** | NETWORK\_CREDIT\_RESTORED | Credits restored |
| **0x7** | NETWORK\_CREDIT\_EFFICIENCY | Credit efficiency metric |
| **0x8** | NETWORK\_CREDIT\_LEAK | Credit leak detected |
| **0x9-0xE** | Reserved | Future expansion |
| **0xF** | NETWORK\_CREDIT\_USER\_DEFINED | User-defined credit event |

##### Channel Events (PktTypeChannel + PROTOCOL\_MNOC)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | NETWORK\_CHANNEL\_OPEN | Channel opened |
| **0x1** | NETWORK\_CHANNEL\_CLOSE | Channel closed |
| **0x2** | NETWORK\_CHANNEL\_STALL | Channel stalled |
| **0x3** | NETWORK\_CHANNEL\_RESUME | Channel resumed |
| **0x4** | NETWORK\_CHANNEL\_CONGESTION | Channel congestion detected |
| **0x5** | NETWORK\_CHANNEL\_PRIORITY | Channel priority change |
| **0x6-0xE** | Reserved | Future expansion |
| **0xF** | NETWORK\_CHANNEL\_USER\_DEFINED | User-defined channel event |

##### Stream Events (PktTypeStream + PROTOCOL\_MNOC)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | NETWORK\_STREAM\_START | Stream started |
| **0x1** | NETWORK\_STREAM\_END | Stream ended (EOS) |
| **0x2** | NETWORK\_STREAM\_PAUSE | Stream paused |
| **0x3** | NETWORK\_STREAM\_RESUME | Stream resumed |
| **0x4** | NETWORK\_STREAM\_OVERFLOW | Stream buffer overflow |
| **0x5** | NETWORK\_STREAM\_UNDERFLOW | Stream buffer underflow |
| **0x6-0xE** | Reserved | Future expansion |
| **0xF** | NETWORK\_STREAM\_USER\_DEFINED | User-defined stream event |

#### ARB Protocol Events

##### Error Events (PktTypeError + PROTOCOL\_ARB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | ARB\_ERR\_STARVATION | Client request starvation |
| **0x1** | ARB\_ERR\_ACK\_TIMEOUT | Grant ACK timeout |
| **0x2** | ARB\_ERR\_PROTOCOL\_VIOLATION | ACK protocol violation |
| **0x3** | ARB\_ERR\_CREDIT\_VIOLATION | Credit system violation |
| **0x4** | ARB\_ERR\_FAIRNESS\_VIOLATION | Weighted fairness violation |
| **0x5** | ARB\_ERR\_WEIGHT\_UNDERFLOW | Weight credit underflow |
| **0x6** | ARB\_ERR\_CONCURRENT\_GRANTS | Multiple simultaneous grants |
| **0x7** | ARB\_ERR\_INVALID\_GRANT\_ID | Invalid grant ID detected |
| **0x8** | ARB\_ERR\_ORPHAN\_ACK | ACK without pending grant |
| **0x9** | ARB\_ERR\_GRANT\_OVERLAP | Overlapping grant periods |
| **0xA** | ARB\_ERR\_MASK\_ERROR | Round-robin mask error |
| **0xB** | ARB\_ERR\_STATE\_MACHINE | FSM state error |
| **0xC** | ARB\_ERR\_CONFIGURATION | Invalid configuration |
| **0xD-0xE** | Reserved | Future expansion |
| **0xF** | ARB\_ERR\_USER\_DEFINED | User-defined error |

##### Timeout Events (PktTypeTimeout + PROTOCOL\_ARB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | ARB\_TIMEOUT\_GRANT\_ACK | Grant ACK timeout |
| **0x1** | ARB\_TIMEOUT\_REQUEST\_HOLD | Request held too long |
| **0x2** | ARB\_TIMEOUT\_WEIGHT\_UPDATE | Weight update timeout |
| **0x3** | ARB\_TIMEOUT\_BLOCK\_RELEASE | Block release timeout |
| **0x4** | ARB\_TIMEOUT\_CREDIT\_UPDATE | Credit update timeout |
| **0x5** | ARB\_TIMEOUT\_STATE\_CHANGE | State machine timeout |
| **0x6-0xE** | Reserved | Future expansion |
| **0xF** | ARB\_TIMEOUT\_USER\_DEFINED | User-defined timeout |

##### Completion Events (PktTypeCompletion + PROTOCOL\_ARB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | ARB\_COMPL\_GRANT\_ISSUED | Grant successfully issued |
| **0x1** | ARB\_COMPL\_ACK\_RECEIVED | ACK successfully received |
| **0x2** | ARB\_COMPL\_TRANSACTION | Complete transaction (grant+ack) |
| **0x3** | ARB\_COMPL\_WEIGHT\_UPDATE | Weight update completed |
| **0x4** | ARB\_COMPL\_CREDIT\_CYCLE | Credit cycle completed |
| **0x5** | ARB\_COMPL\_FAIRNESS\_PERIOD | Fairness analysis period |
| **0x6** | ARB\_COMPL\_BLOCK\_PERIOD | Block period completed |
| **0x7-0xE** | Reserved | Future expansion |
| **0xF** | ARB\_COMPL\_USER\_DEFINED | User-defined completion |

##### Threshold Events (PktTypeThreshold + PROTOCOL\_ARB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | ARB\_THRESH\_REQUEST\_LATENCY | Request-to-grant latency threshold |
| **0x1** | ARB\_THRESH\_ACK\_LATENCY | Grant-to-ACK latency threshold |
| **0x2** | ARB\_THRESH\_FAIRNESS\_DEV | Fairness deviation threshold |
| **0x3** | ARB\_THRESH\_ACTIVE\_REQUESTS | Active request count threshold |
| **0x4** | ARB\_THRESH\_GRANT\_RATE | Grant rate threshold |
| **0x5** | ARB\_THRESH\_EFFICIENCY | Grant efficiency threshold |
| **0x6** | ARB\_THRESH\_CREDIT\_LOW | Low credit threshold |
| **0x7** | ARB\_THRESH\_WEIGHT\_IMBALANCE | Weight imbalance threshold |
| **0x8** | ARB\_THRESH\_STARVATION\_TIME | Starvation time threshold |
| **0x9-0xE** | Reserved | Future expansion |
| **0xF** | ARB\_THRESH\_USER\_DEFINED | User-defined threshold |

##### Performance Events (PktTypePerf + PROTOCOL\_ARB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | ARB\_PERF\_GRANT\_ISSUED | Grant issued event |
| **0x1** | ARB\_PERF\_ACK\_RECEIVED | ACK received event |
| **0x2** | ARB\_PERF\_GRANT\_EFFICIENCY | Grant completion efficiency |
| **0x3** | ARB\_PERF\_FAIRNESS\_METRIC | Fairness compliance metric |
| **0x4** | ARB\_PERF\_THROUGHPUT | Arbitration throughput |
| **0x5** | ARB\_PERF\_LATENCY\_AVG | Average latency measurement |
| **0x6** | ARB\_PERF\_WEIGHT\_COMPLIANCE | Weight compliance metric |
| **0x7** | ARB\_PERF\_CREDIT\_UTILIZATION | Credit utilization efficiency |
| **0x8** | ARB\_PERF\_CLIENT\_ACTIVITY | Per-client activity metric |
| **0x9** | ARB\_PERF\_STARVATION\_COUNT | Starvation event count |
| **0xA** | ARB\_PERF\_BLOCK\_EFFICIENCY | Block/unblock efficiency |
| **0xB-0xE** | Reserved | Future expansion |
| **0xF** | ARB\_PERF\_USER\_DEFINED | User-defined performance |

##### Debug Events (PktTypeDebug + PROTOCOL\_ARB)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | ARB\_DEBUG\_STATE\_CHANGE | Arbiter state machine change |
| **0x1** | ARB\_DEBUG\_MASK\_UPDATE | Round-robin mask update |
| **0x2** | ARB\_DEBUG\_WEIGHT\_CHANGE | Weight configuration change |
| **0x3** | ARB\_DEBUG\_CREDIT\_UPDATE | Credit level update |
| **0x4** | ARB\_DEBUG\_CLIENT\_MASK | Client enable/disable mask |
| **0x5** | ARB\_DEBUG\_PRIORITY\_CHANGE | Priority level change |
| **0x6** | ARB\_DEBUG\_BLOCK\_EVENT | Block/unblock event |
| **0x7** | ARB\_DEBUG\_QUEUE\_STATUS | Request queue status |
| **0x8** | ARB\_DEBUG\_COUNTER\_SNAPSHOT | Counter values snapshot |
| **0x9** | ARB\_DEBUG\_FIFO\_STATUS | FIFO status change |
| **0xA** | ARB\_DEBUG\_FAIRNESS\_STATE | Fairness tracking state |
| **0xB** | ARB\_DEBUG\_ACK\_STATE | ACK protocol state |
| **0xC-0xE** | Reserved | Future expansion |
| **0xF** | ARB\_DEBUG\_USER\_DEFINED | User-defined debug |

#### CORE Protocol Events

##### Error Events (PktTypeError + PROTOCOL\_CORE)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | CORE\_ERR\_DESCRIPTOR\_MALFORMED | Missing magic number (0x900dc0de) |
| **0x1** | CORE\_ERR\_DESCRIPTOR\_BAD\_ADDR | Invalid descriptor address |
| **0x2** | CORE\_ERR\_DATA\_BAD\_ADDR | Invalid data address (fetch or runtime) |
| **0x3** | CORE\_ERR\_FLAG\_COMPARISON | Flag mask/compare mismatch |
| **0x4** | CORE\_ERR\_CREDIT\_UNDERFLOW | Credit system violation |
| **0x5** | CORE\_ERR\_STATE\_MACHINE | Invalid FSM state transition |
| **0x6** | CORE\_ERR\_DESCRIPTOR\_ENGINE | Descriptor engine FSM error |
| **0x7** | CORE\_ERR\_FLAG\_ENGINE | Flag engine FSM error |
| **0x8** | CORE\_ERR\_PROGRAM\_ENGINE | Program engine FSM error |
| **0x9** | CORE\_ERR\_DATA\_ENGINE | Data engine error |
| **0xA** | CORE\_ERR\_CHANNEL\_INVALID | Invalid channel ID |
| **0xB** | CORE\_ERR\_CONTROL\_VIOLATION | Control register violation |
| **0xC-0xE** | Reserved | Future expansion |
| **0xF** | CORE\_ERR\_USER\_DEFINED | User-defined error |

##### Timeout Events (PktTypeTimeout + PROTOCOL\_CORE)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | CORE\_TIMEOUT\_DESCRIPTOR\_FETCH | Descriptor fetch timeout |
| **0x1** | CORE\_TIMEOUT\_FLAG\_RETRY | Flag comparison retry timeout |
| **0x2** | CORE\_TIMEOUT\_PROGRAM\_WRITE | Program write timeout |
| **0x3** | CORE\_TIMEOUT\_DATA\_TRANSFER | Data transfer timeout |
| **0x4** | CORE\_TIMEOUT\_CREDIT\_WAIT | Credit wait timeout |
| **0x5** | CORE\_TIMEOUT\_CONTROL\_WAIT | Control enable wait timeout |
| **0x6** | CORE\_TIMEOUT\_ENGINE\_RESPONSE | Sub-engine response timeout |
| **0x7** | CORE\_TIMEOUT\_STATE\_TRANSITION | FSM state transition timeout |
| **0x8-0xE** | Reserved | Future expansion |
| **0xF** | CORE\_TIMEOUT\_USER\_DEFINED | User-defined timeout |

##### Completion Events (PktTypeCompletion + PROTOCOL\_CORE)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | CORE\_COMPL\_DESCRIPTOR\_LOADED | Descriptor successfully loaded |
| **0x1** | CORE\_COMPL\_DESCRIPTOR\_CHAIN | Descriptor chain completed |
| **0x2** | CORE\_COMPL\_FLAG\_MATCHED | Flag comparison successful |
| **0x3** | CORE\_COMPL\_PROGRAM\_COMPLETED | Post-programming completed |
| **0x4** | CORE\_COMPL\_DATA\_TRANSFER | Data transfer completed |
| **0x5** | CORE\_COMPL\_CREDIT\_CYCLE | Credit cycle completed |
| **0x6** | CORE\_COMPL\_CHANNEL\_COMPLETE | Channel processing complete |
| **0x7** | CORE\_COMPL\_ENGINE\_READY | Sub-engine ready |
| **0x8-0xE** | Reserved | Future expansion |
| **0xF** | CORE\_COMPL\_USER\_DEFINED | User-defined completion |

##### Threshold Events (PktTypeThreshold + PROTOCOL\_CORE)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | CORE\_THRESH\_DESCRIPTOR\_QUEUE | Descriptor queue depth threshold |
| **0x1** | CORE\_THRESH\_CREDIT\_LOW | Credit low threshold |
| **0x2** | CORE\_THRESH\_FLAG\_RETRY\_COUNT | Flag retry count threshold |
| **0x3** | CORE\_THRESH\_LATENCY | Processing latency threshold |
| **0x4** | CORE\_THRESH\_ERROR\_RATE | Error rate threshold |
| **0x5** | CORE\_THRESH\_THROUGHPUT | Throughput threshold |
| **0x6** | CORE\_THRESH\_ACTIVE\_CHANNELS | Active channel count threshold |
| **0x7** | CORE\_THRESH\_PROGRAM\_LATENCY | Program write latency threshold |
| **0x8** | CORE\_THRESH\_DATA\_RATE | Data transfer rate threshold |
| **0x9-0xE** | Reserved | Future expansion |
| **0xF** | CORE\_THRESH\_USER\_DEFINED | User-defined threshold |

##### Performance Events (PktTypePerf + PROTOCOL\_CORE)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | CORE\_PERF\_END\_OF\_DATA | Stream continuation signal |
| **0x1** | CORE\_PERF\_END\_OF\_STREAM | Stream termination signal |
| **0x2** | CORE\_PERF\_ENTERING\_IDLE | FSM returning to idle |
| **0x3** | CORE\_PERF\_CREDIT\_INCREMENTED | Credit added by software |
| **0x4** | CORE\_PERF\_CREDIT\_EXHAUSTED | Credit blocking execution |
| **0x5** | CORE\_PERF\_STATE\_TRANSITION | FSM state change |
| **0x6** | CORE\_PERF\_DESCRIPTOR\_ACTIVE | Data processing started |
| **0x7** | CORE\_PERF\_FLAG\_RETRY | Flag comparison retry |
| **0x8** | CORE\_PERF\_CHANNEL\_ENABLE | Channel enabled by software |
| **0x9** | CORE\_PERF\_CHANNEL\_DISABLE | Channel disabled by software |
| **0xA** | CORE\_PERF\_CREDIT\_UTILIZATION | Credit utilization metric |
| **0xB** | CORE\_PERF\_PROCESSING\_LATENCY | Total processing latency |
| **0xC** | CORE\_PERF\_QUEUE\_DEPTH | Current queue depth |
| **0xD-0xE** | Reserved | Future expansion |
| **0xF** | CORE\_PERF\_USER\_DEFINED | User-defined performance |

##### Debug Events (PktTypeDebug + PROTOCOL\_CORE)

| Code | Event Name | Description |
| --- | --- | --- |
| **0x0** | CORE\_DEBUG\_FSM\_STATE\_CHANGE | Descriptor FSM state change |
| **0x1** | CORE\_DEBUG\_DESCRIPTOR\_CONTENT | Descriptor content trace |
| **0x2** | CORE\_DEBUG\_FLAG\_ENGINE\_STATE | Flag engine state trace |
| **0x3** | CORE\_DEBUG\_PROGRAM\_ENGINE\_STATE | Program engine state trace |
| **0x4** | CORE\_DEBUG\_CREDIT\_OPERATION | Credit system operation |
| **0x5** | CORE\_DEBUG\_CONTROL\_REGISTER | Control register access |
| **0x6** | CORE\_DEBUG\_ENGINE\_HANDSHAKE | Engine handshake trace |
| **0x7** | CORE\_DEBUG\_QUEUE\_STATUS | Queue status change |
| **0x8** | CORE\_DEBUG\_COUNTER\_SNAPSHOT | Counter values snapshot |
| **0x9** | CORE\_DEBUG\_ADDRESS\_TRACE | Address progression trace |
| **0xA** | CORE\_DEBUG\_PAYLOAD\_TRACE | Payload content trace |
| **0xB-0xE** | Reserved | Future expansion |
| **0xF** | CORE\_DEBUG\_USER\_DEFINED | User-defined debug |

### Memory Architecture and Packet Routing

#### Two-Tier Memory Architecture

##### Local Error/Interrupt Memory

| Characteristic | Description |
| --- | --- |
| **Storage Types** | Error Packets (Type 0x0) and Timeout Packets (Type 0x3) |
| **Access Method** | Immediate CPU access without memory subsystem delays |
| **Capacity** | Large enough to prevent overflow during error bursts |
| **Priority** | Critical events requiring immediate attention |
| **Indexing** | Fast search and retrieval mechanisms |

##### Configurable External Memory

| Characteristic | Description |
| --- | --- |
| **Storage Types** | Performance, Completion, Threshold, Debug packets |
| **Access Method** | Base and limit registers define memory regions |
| **Capacity** | Bulk storage for non-critical events |
| **DMA Support** | Can be accessed via DMA for efficient transfer |
| **Time Stamping** | 32-bit timestamp appended when routing externally |

#### Routing Configuration

##### Base and Limit Registers

| Register Set | Purpose | Configuration |
| --- | --- | --- |
| **Completion Config** | Type 0x1 routing | base\_addr, limit\_addr, enable, priority |
| **Threshold Config** | Type 0x2 routing | base\_addr, limit\_addr, enable, priority |
| **Performance Config** | Type 0x4 routing | base\_addr, limit\_addr, enable, priority |
| **Debug Config** | Type 0xF routing | base\_addr, limit\_addr, enable, priority |

##### Routing Decision Logic

| Packet Type | Destination | Address Calculation |
| --- | --- | --- |
| **Error (0x0)** | Local Memory | local\_error\_write\_pointer |
| **Timeout (0x3)** | Local Memory | local\_error\_write\_pointer |
| **Completion (0x1)** | External Memory | completion\_config.base\_addr + offset |
| **Performance (0x4)** | External Memory | performance\_config.base\_addr + offset |
| **Debug (0xF)** | External Memory | debug\_config.base\_addr + offset |

#### Address Space Management

##### Memory Layout Example

| Address Range | Usage | Description |
| --- | --- | --- |
| **0x1000\_0000 - 0x1000\_FFFF** | Local Error Memory | Immediate access storage |
| **0x2000\_0000 - 0x2001\_FFFF** | Performance Packets | External bulk storage |
| **0x2010\_0000 - 0x2011\_FFFF** | Completion Packets | External bulk storage |
| **0x2020\_0000 - 0x202F\_FFFF** | Debug Packets | External bulk storage |

#### Transaction State and Bus Transaction Structure

##### Transaction State Enumeration

| State | Value | Description | Usage |
| --- | --- | --- | --- |
| **TRANS\_EMPTY** | 3’b000 | Unused entry | Available slot |
| **TRANS\_ADDR\_PHASE** | 3’b001 | Address phase active (AXI) / Packet sent (Network) / Setup phase (APB) | Initial phase |
| **TRANS\_DATA\_PHASE** | 3’b010 | Data phase active (AXI) / Waiting for ACK (Network) / Access phase (APB) | Data transfer |
| **TRANS\_RESP\_PHASE** | 3’b011 | Response phase active (AXI) / ACK received (Network) / Enable phase (APB) | Response handling |
| **TRANS\_COMPLETE** | 3’b100 | Transaction complete | Successful completion |
| **TRANS\_ERROR** | 3’b101 | Transaction has error | Error condition |
| **TRANS\_ORPHANED** | 3’b110 | Orphaned transaction | Missing components |
| **TRANS\_CREDIT\_STALL** | 3’b111 | Credit stall (Network only) | Network-specific stall |

##### Enhanced Transaction Structure

| Field | Width | Description | Protocol Usage |
| --- | --- | --- | --- |
| **valid** | 1 | Entry is valid | All protocol’s |
| **protocol** | 3 | Protocol type (AXI/Network/APB/ARB/CORE) | All protocols |
| **state** | 3 | Transaction state | All protocols |
| **id** | 32 | Transaction ID (AXI) / Sequence (Network) / PSEL encoding (APB) | All protocols |
| **addr** | 64 | Transaction address / Channel addr / PADDR | All protocols |
| **len** | 8 | Burst length (AXI) / Packet count (Network) / Always 0 (APB) | AXI, Network |
| **size** | 3 | Access size (AXI) / Reserved (Network) / Transfer size (APB) | AXI, APB |
| **burst** | 2 | Burst type (AXI) / Payload type (Network) / PPROT[1:0] (APB) | All protocols |

##### Phase Completion Flags

| Flag | Description | Protocol Usage |
| --- | --- | --- |
| **cmd\_received** | Address phase received / Packet sent / Setup phase | All protocols |
| **data\_started** | Data phase started / ACK expected / Access phase | All protocols |
| **data\_completed** | Data phase completed / ACK received / Enable phase | All protocols |
| **resp\_received** | Response received / Final ACK / PREADY asserted | All protocols |

##### Protocol-Specific Tracking Fields

| Field | Width | Description | Protocol |
| --- | --- | --- | --- |
| **channel** | 6 | Channel ID (AXI ID / Network channel / PSEL bit position) | All protocols |
| **eos\_seen** | 1 | EOS marker seen | Network only |
| **parity\_error** | 1 | Parity error detected | Network only |
| **credit\_at\_start** | 8 | Credits available at start | Network only |
| **retry\_count** | 3 | Number of retries | Network only |
| **desc\_addr\_match** | 1 | Descriptor address match detected | AXI only |
| **data\_addr\_match** | 1 | Data address match detected | AXI only |
| **apb\_phase** | 2 | Current APB phase | APB only |
| **pslverr\_seen** | 1 | PSLVERR detected | APB only |
| **pprot\_value** | 3 | PPROT value | APB only |
| **pstrb\_value** | 4 | PSTRB value for writes | APB only |
| **arb\_grant\_id** | 8 | Current grant ID | ARB only |
| **arb\_weight** | 8 | Current weight value | ARB only |
| **core\_fsm\_state** | 3 | Current CORE FSM state | CORE only |
| **core\_channel\_id** | 6 | CORE channel identifier | CORE only |

#### APB Transaction Phases

| Phase | Value | Description |
| --- | --- | --- |
| **APB\_PHASE\_IDLE** | 2’b00 | Bus idle |
| **APB\_PHASE\_SETUP** | 2’b01 | Setup phase (PSEL asserted) |
| **APB\_PHASE\_ACCESS** | 2’b10 | Access phase (PENABLE asserted) |
| **APB\_PHASE\_ENABLE** | 2’b11 | Enable phase (waiting for PREADY) |

#### APB Protection Types

| Protection | Value | Description |
| --- | --- | --- |
| **APB\_PROT\_NORMAL** | 3’b000 | Normal access |
| **APB\_PROT\_PRIVILEGED** | 3’b001 | Privileged access |
| **APB\_PROT\_SECURE** | 3’b010 | Secure access |
| **APB\_PROT\_INSTRUCTION** | 3’b100 | Instruction access |

#### Network Payload Types

| Payload | Value | Description |
| --- | --- | --- |
| **NETWORK\_PAYLOAD\_CONFIG** | 2’b00 | CONFIG\_PKT |
| **NETWORK\_PAYLOAD\_TS** | 2’b01 | TS\_PKT |
| **NETWORK\_PAYLOAD\_RDA** | 2’b10 | RDA\_PKT |
| **NETWORK\_PAYLOAD\_RAW** | 2’b11 | RAW\_PKT |

#### Network ACK Types

| ACK Type | Value | Description |
| --- | --- | --- |
| **NETWORK\_ACK\_STOP** | 2’b00 | MSAP\_STOP |
| **NETWORK\_ACK\_START** | 2’b01 | MSAP\_START |
| **NETWORK\_ACK\_CREDIT\_ON** | 2’b10 | MSAP\_CREDIT\_ON |
| **NETWORK\_ACK\_STOP\_AT\_EOS** | 2’b11 | MSAP\_STOP\_AT\_EOS |

#### ARB State Types

| State | Value | Description |
| --- | --- | --- |
| **ARB\_STATE\_IDLE** | 3’b000 | Idle state |
| **ARB\_STATE\_ARBITRATE** | 3’b001 | Performing arbitration |
| **ARB\_STATE\_GRANT** | 3’b010 | Grant issued, waiting for ACK |
| **ARB\_STATE\_BLOCKED** | 3’b011 | Arbitration blocked |
| **ARB\_STATE\_WEIGHT\_UPD** | 3’b100 | Weight update in progress |
| **ARB\_STATE\_ERROR** | 3’b101 | Error state |

#### CORE State Types

| State | Value | Description |
| --- | --- | --- |
| **CORE\_STATE\_IDLE** | 3’b000 | Idle state |
| **CORE\_STATE\_DESC\_FETCH** | 3’b001 | Fetching descriptor |
| **CORE\_STATE\_FLAG\_CHECK** | 3’b010 | Checking flag condition |
| **CORE\_STATE\_PROGRAM\_WRITE** | 3’b011 | Writing program |
| **CORE\_STATE\_DATA\_TRANSFER** | 3’b100 | Transferring data |
| **CORE\_STATE\_CREDIT\_WAIT** | 3’b101 | Waiting for credits |
| **CORE\_STATE\_ERROR** | 3’b110 | Error state |

### Configuration and Control

#### Monitor Configuration Registers

##### Global Configuration

| Field | Width | Description |
| --- | --- | --- |
| **monitor\_enable** | 1 | Global monitor enable |
| **error\_local\_enable** | 1 | Enable local error storage |
| **external\_route\_enable** | 1 | Enable external routing |
| **unit\_id** | 4 | Unit identifier |
| **agent\_id** | 8 | Agent identifier |
| **packet\_type\_enables** | 16 | Per-type enable bits |

##### Packet Type Enable Mapping

| Bit | Enable | Description |
| --- | --- | --- |
| **0** | PKT\_ENABLE\_ERROR | Enable error packets |
| **1** | PKT\_ENABLE\_COMPLETION | Enable completion packets |
| **2** | PKT\_ENABLE\_THRESHOLD | Enable threshold packets |
| **3** | PKT\_ENABLE\_TIMEOUT | Enable timeout packets |
| **4** | PKT\_ENABLE\_PERF | Enable performance packets |
| **5** | PKT\_ENABLE\_CREDIT | Enable credit packets (Network) |
| **6** | PKT\_ENABLE\_CHANNEL | Enable channel packets (Network) |
| **7** | PKT\_ENABLE\_STREAM | Enable stream packets (Network) |
| **8** | PKT\_ENABLE\_ADDR\_MATCH | Enable address match (AXI) |
| **9** | PKT\_ENABLE\_APB | Enable APB packets |
| **15** | PKT\_ENABLE\_DEBUG | Enable debug packets |

#### Protocol-Specific Configuration

##### AXI Monitor Configuration

| Field | Width | Description |
| --- | --- | --- |
| **active\_trans\_threshold** | 16 | Active transaction threshold |
| **latency\_threshold** | 32 | Latency threshold (cycles) |
| **addr\_timeout\_cnt** | 4 | Address timeout count |
| **data\_timeout\_cnt** | 4 | Data timeout count |
| **resp\_timeout\_cnt** | 4 | Response timeout count |
| **burst\_boundary\_check** | 1 | Enable burst boundary checking |
| **address\_match\_enable** | 1 | Enable address matching |
| **desc\_addr\_match\_base** | 64 | Descriptor address match base |
| **desc\_addr\_match\_mask** | 64 | Descriptor address match mask |
| **data\_addr\_match\_base** | 64 | Data address match base |
| **data\_addr\_match\_mask** | 64 | Data address match mask |

##### Network Monitor Configuration

| Field | Width | Description |
| --- | --- | --- |
| **credit\_low\_threshold** | 8 | Credit low threshold |
| **packet\_rate\_threshold** | 16 | Packet rate threshold |
| **max\_route\_hops** | 8 | Maximum routing hops |
| **enable\_credit\_tracking** | 1 | Enable credit tracking |
| **enable\_deadlock\_detect** | 1 | Enable deadlock detection |
| **deadlock\_timeout** | 4 | Deadlock detection timeout |

##### ARB Monitor Configuration

| Field | Width | Description |
| --- | --- | --- |
| **grant\_timeout\_cnt** | 16 | Grant ACK timeout count |
| **fairness\_window** | 32 | Fairness analysis window |
| **weight\_update\_enable** | 1 | Enable weight tracking |
| **starvation\_threshold** | 16 | Starvation detection threshold |
| **efficiency\_threshold** | 8 | Grant efficiency threshold |

##### CORE Monitor Configuration

| Field | Width | Description |
| --- | --- | --- |
| **descriptor\_timeout\_cnt** | 16 | Descriptor fetch timeout count |
| **flag\_retry\_limit** | 8 | Maximum flag retry count |
| **credit\_low\_threshold** | 8 | Credit low threshold |
| **processing\_timeout\_cnt** | 32 | Processing timeout count |
| **enable\_descriptor\_trace** | 1 | Enable descriptor content tracing |
| **enable\_fsm\_trace** | 1 | Enable FSM state tracing |

### Validation Requirements

#### Functional Validation

| Validation Area | Requirements |
| --- | --- |
| **Packet Format** | Verify 64-bit packet structure and field encoding |
| **Event Organization** | Verify hierarchical event code organization |
| **Protocol Isolation** | Verify independent protocol event spaces |
| **Routing Logic** | Verify packet routing based on type and configuration |
| **Memory Management** | Verify local and external memory operations |
| **Configuration** | Verify register configuration and enable controls |

#### Performance Validation

| Validation Area | Requirements |
| --- | --- |
| **Throughput** | Verify monitor bus can handle peak event rates |
| **Latency** | Verify low-latency path for critical events |
| **Memory Efficiency** | Verify efficient memory usage patterns |
| **Power Consumption** | Verify power-efficient operation |

#### Error Handling Validation

| Validation Area | Requirements |
| --- | --- |
| **Error Injection** | Verify error detection and reporting |
| **Overflow Handling** | Verify behavior when memories fill |
| **Configuration Errors** | Verify invalid configuration detection |
| **Recovery Mechanisms** | Verify error recovery procedures |

### Usage Examples

#### Creating Monitor Packets

| Packet Type | Example Usage |
| --- | --- |
| **AXI Error** | Protocol=AXI, Type=Error, Code=AXI\_ERR\_RESP\_SLVERR |
| **Network Credit** | Protocol=Network, Type=Credit, Code=NETWORK\_CREDIT\_EXHAUSTED |
| **APB Performance** | Protocol=APB, Type=Performance, Code=APB\_PERF\_TOTAL\_LATENCY |
| **ARB Threshold** | Protocol=ARB, Type=Threshold, Code=ARB\_THRESH\_FAIRNESS\_DEV |
| **CORE Completion** | Protocol=CORE, Type=Completion, Code=CORE\_COMPL\_DESCRIPTOR\_LOADED |

#### Packet Decoding

| Decoding Step | Method |
| --- | --- |
| **Extract Type** | packet[63:60] |
| **Extract Protocol** | packet[59:57] |
| **Extract Event Code** | packet[56:53] |
| **Extract Channel ID** | packet[52:47] |
| **Extract Event Data** | packet[34:0] |

#### Monitor Bus Packet Helper Functions

##### Packet Field Extraction

| Function | Return Type | Description |
| --- | --- | --- |
| **get\_packet\_type(pkt)** | logic [3:0] | Extract packet type [63:60] |
| **get\_protocol\_type(pkt)** | protocol\_type\_t | Extract protocol [59:57] |
| **get\_event\_code(pkt)** | logic [3:0] | Extract event code [56:53] |
| **get\_channel\_id(pkt)** | logic [5:0] | Extract channel ID [52:47] |
| **get\_unit\_id(pkt)** | logic [3:0] | Extract unit ID [46:43] |
| **get\_agent\_id(pkt)** | logic [7:0] | Extract agent ID [42:35] |
| **get\_event\_data(pkt)** | logic [34:0] | Extract event data [34:0] |

##### Packet Creation Function

| Function | Parameters | Description |
| --- | --- | --- |
| **create\_monitor\_packet()** | packet\_type, protocol, event\_code, channel\_id, unit\_id, agent\_id, event\_data | Create complete 64-bit packet |

##### Event Code Creation Functions

| Function | Parameter | Description |
| --- | --- | --- |
| **create\_axi\_error\_event()** | axi\_error\_code\_t | Create AXI error event code |
| **create\_axi\_timeout\_event()** | axi\_timeout\_code\_t | Create AXI timeout event code |
| **create\_axi\_completion\_event()** | axi\_completion\_code\_t | Create AXI completion event code |
| **create\_axi\_threshold\_event()** | axi\_threshold\_code\_t | Create AXI threshold event code |
| **create\_axi\_performance\_event()** | axi\_performance\_code\_t | Create AXI performance event code |
| **create\_axi\_addr\_match\_event()** | axi\_addr\_match\_code\_t | Create AXI address match event code |
| **create\_axi\_debug\_event()** | axi\_debug\_code\_t | Create AXI debug event code |
| **create\_apb\_error\_event()** | apb\_error\_code\_t | Create APB error event code |
| **create\_apb\_timeout\_event()** | apb\_timeout\_code\_t | Create APB timeout event code |
| **create\_apb\_completion\_event()** | apb\_completion\_code\_t | Create APB completion event code |
| **create\_network\_error\_event()** | network\_error\_code\_t | Create Network error event code |
| **create\_network\_timeout\_event()** | network\_timeout\_code\_t | Create Network timeout event code |
| **create\_network\_completion\_event()** | network\_completion\_code\_t | Create Network completion event code |
| **create\_network\_credit\_event()** | network\_credit\_code\_t | Create Network credit event code |
| **create\_network\_channel\_event()** | network\_channel\_code\_t | Create Network channel event code |
| **create\_network\_stream\_event()** | network\_stream\_code\_t | Create Network stream event code |
| **create\_arb\_error\_event()** | arb\_error\_code\_t | Create ARB error event code |
| **create\_arb\_timeout\_event()** | arb\_timeout\_code\_t | Create ARB timeout event code |
| **create\_arb\_completion\_event()** | arb\_completion\_code\_t | Create ARB completion event code |
| **create\_arb\_threshold\_event()** | arb\_threshold\_code\_t | Create ARB threshold event code |
| **create\_arb\_performance\_event()** | arb\_performance\_code\_t | Create ARB performance event code |
| **create\_arb\_debug\_event()** | arb\_debug\_code\_t | Create ARB debug event code |
| **create\_core\_error\_event()** | core\_error\_code\_t | Create CORE error event code |
| **create\_core\_timeout\_event()** | core\_timeout\_code\_t | Create CORE timeout event code |
| **create\_core\_completion\_event()** | core\_completion\_code\_t | Create CORE completion event code |
| **create\_core\_threshold\_event()** | core\_threshold\_code\_t | Create CORE threshold event code |
| **create\_core\_performance\_event()** | core\_performance\_code\_t | Create CORE performance event code |
| **create\_core\_debug\_event()** | core\_debug\_code\_t | Create CORE debug event code |

##### Validation Functions

| Function | Parameters | Description |
| --- | --- | --- |
| **is\_valid\_event\_for\_packet\_type()** | packet\_type, protocol, event\_code | Validate event code for packet type and protocol |

##### String Functions for Debugging

| Function | Parameter | Description |
| --- | --- | --- |
| **get\_axi\_error\_name()** | axi\_error\_code\_t | Get human-readable AXI error name |
| **get\_arb\_error\_name()** | arb\_error\_code\_t | Get human-readable ARB error name |
| **get\_core\_error\_name()** | core\_error\_code\_t | Get human-readable CORE error name |
| **get\_packet\_type\_name()** | logic [3:0] | Get packet type name string |
| **get\_protocol\_name()** | protocol\_type\_t | Get protocol name string |
| **get\_event\_name()** | packet\_type, protocol, event\_code | Get comprehensive event name |

### Debug and Monitoring Signals

#### Essential Debug Signals

| Signal | Width | Purpose |
| --- | --- | --- |
| **debug\_packet\_counts** | 32 x 16 | Packet count per type |
| **debug\_protocol\_counts** | 32 x 5 | Packet count per protocol |
| **debug\_error\_counts** | 32 | Total error packet count |
| **debug\_local\_memory\_level** | 16 | Local memory usage level |
| **debug\_external\_memory\_level** | 16 | External memory usage level |

#### Performance Counters

| Counter | Width | Purpose |
| --- | --- | --- |
| **total\_packets\_processed** | 32 | Total packets processed |
| **packets\_dropped** | 32 | Packets dropped due to overflow |
| **routing\_errors** | 32 | Routing configuration errors |
| **memory\_full\_events** | 32 | Memory full occurrences |

### Protocol Coverage Summary

#### Complete Protocol Event Matrix

| Protocol | Error | Timeout | Completion | Threshold | Performance | Debug | Protocol-Specific |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **AXI** | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | AddrMatch [PASS] 16 |
| **Network** | [PASS] 16 | [PASS] 16 | [PASS] 16 | [FAIL] 0 | [FAIL] 0 | [FAIL] 0 | Credit/Channel/Stream [PASS] 48 |
| **APB** | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | None |
| **ARB** | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | None |
| **CORE** | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | [PASS] 16 | None |

**Total Event Codes**: 544 defined across all protocols and packet types.

## Chapter 2: Programming Model

## Programming

TBD## Registers

## TBD

## Chapter 3: Architecture Summary

### Design Principles

**RAPIDS Architecture** consists of three main subsystems:

1. **Scheduler Group** - Controls operation scheduling and descriptor management
2. **Sink Data Path** - Network -> Memory transfers (receive path)
3. **Source Data Path** - Memory -> Network transfers (transmit path)

### Key Features

**Multi-Channel Operation:** - Up to 32 independent channels - Per-channel SRAM buffering (configurable depth) - Independent flow control per channel

**AXIS Network Interfaces:** - AXI4-Stream for network data (512-bit default) - Native TREADY/TVALID backpressure - TLAST for end-of-stream marking - TSTRB for byte-level valid indication

**Memory Interfaces:** - AXI4 for high-performance memory access - Burst-capable read/write engines - Address alignment handling - Write response management

**Monitoring:** - Comprehensive event monitoring via MonBus - Performance counters - Error detection and reporting - Debug visibility

## Chapter 4: Block Descriptions

### Scheduler Group

**Scheduler** - Central control FSM coordinating all operations **Descriptor Engine** - Manages descriptor FIFO and parsing **Control Write Engine** - Handles control packet writes **Control Read Engine** - Handles control packet reads

### Sink Data Path (Network -> Memory)

**AXIS Slave** - Receives data from network via AXI4-Stream **Sink SRAM Control** - Multi-channel buffering and flow control **Sink AXI Write Engine** - Writes buffered data to system memory

### Source Data Path (Memory -> Network)

**Source AXI Read Engine** - Reads data from system memory **Source SRAM Control** - Multi-channel buffering and flow control **AXIS Master** - Transmits data to network via AXI4-Stream

### MonBus AXIL Group

**AXIL Slave** - Control/status register access **MonBus Reporter** - Event monitoring and packet generation

## Chapter 5: SRAM Storage Format (v0.25)

### Sink SRAM Format

**Width:** 530 bits **Format:** {TYPE[1:0], CHUNK\_VALID[15:0], DATA[511:0]}

* TYPE: Packet type classification (CDA vs DATA)
* CHUNK\_VALID: 16 x 32-bit chunk enables (derived from TSTRB)
* DATA: 512-bit payload data
* EOS: NOT stored (tracked separately for completion signaling)

**Rationale:** Downstream AXI write engine doesn’t need EOS (controlled by scheduler)

### Source SRAM Format

**Width:** 531 bits **Format:** {EOS[1], TYPE[2], CHUNK\_VALID[16], DATA[512]}

* EOS: Stream end marker (generates TLAST on AXIS output)
* TYPE: Packet type classification
* CHUNK\_VALID: 16 x 32-bit chunk enables (converted to TSTRB)
* DATA: 512-bit payload data

**Rationale:** AXIS master needs EOS to assert TLAST on final beat

### Storage Efficiency

* Overhead: 3.5-3.7% (18-19 metadata bits per 512-bit data line)
* Total SRAM: ~535 KB (32 channels x 256 lines x 530 bits)
* Intentional asymmetry (sink 530, source 531) is optimal for AXIS

## Chapter 6: Migration Notes (v0.25)

### AXIS Migration Complete

**Removed:** - [FAIL] Custom network protocol credit return logic - [FAIL] Data consumption FIFO (replaced with direct AXIS backpressure) - [FAIL] EOL/EOD stream markers (replaced with EOS only) - [FAIL] Network-specific event codes

**Added:** - [PASS] AXI4-Stream master/slave interfaces - [PASS] TSTRB to CHUNK\_VALID conversion (64 bytes -> 16 x 4-byte chunks) - [PASS] TLAST generation from stored EOS (source path) - [PASS] AXIS-optimized monitor event codes - [PASS] Native AXIS backpressure (TREADY/TVALID)

**Optimized:** - [PASS] SRAM storage format (removed 3 bits: EOL, EOD, EOS in sink) - [PASS] Direct passthrough for AXIS handshaking - [PASS] Simplified completion tracking (EOS-only)

### Performance Impact

* **Throughput:** Improved due to simpler handshaking
* **Latency:** Reduced by eliminating credit return logic
* **Storage:** More efficient (3.5% overhead vs 4.3% in v0.24)
* **Compatibility:** Standard AXIS interfaces enable ecosystem integration

## Chapter 7: Future Enhancements

### Potential TYPE Field Extensions

Current 2-bit TYPE field (4 possible values): - 2'b00 - CDA (Control Descriptor Addressing, user\_id = 255) - 2'b01 - DATA (Normal user traffic, user\_id < 255) - 2'b10 - Reserved (management/diagnostic packets?) - 2'b11 - Reserved (high-priority express packets?)

### Scalability Considerations

* Parameterized channel count (tested up to 32)
* Configurable SRAM depth per channel
* Flexible data width (512-bit typical, supports others)
* Adaptable chunk granularity (32-bit standard)

## Document Metadata

**Version:** 0.25 **Date:** 2025-10-18 **Status:** AXIS Migration Complete **Generator:** md\_to\_docx.py **Repository:** github.com/sean-galloway/RTLDesignSherpa **Path:** projects/components/rapids/