

# Versal ACAP Programmable Network on Chip and Integrated Memory Controller v1.0

## *LogiCORE IP Product Guide*

Vivado Design Suite

PG313 (v1.0) November 8, 2021

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this [link](#) for more information.



# Table of Contents

<b>Chapter 1: IP Facts.....</b>	<b>4</b>
<b>Chapter 2: Overview.....</b>	<b>5</b>
Navigating Content by Design Process.....	5
Versal Programmable Network on Chip Overview.....	6
DDR Memory Controller.....	11
<b>Chapter 3: NoC Architecture.....</b>	<b>14</b>
NoC Components.....	15
Quality of Service.....	33
Data Integrity.....	34
Standards.....	38
AXI Conversion.....	39
Clocking.....	48
Resets.....	49
<b>Chapter 4: Integrated Memory Controller (DDRMC) Architecture...</b>	<b>50</b>
Memory Controller.....	50
<b>Chapter 5: Designing with the Core.....</b>	<b>68</b>
Introduction to the Inter-NoC Interface.....	68
Configuring the AXI NoC.....	70
Configuring the Memory Controller.....	80
Configuring the AXIS NoC.....	90
<b>Chapter 6: NoC and Memory Controller Simulation.....</b>	<b>96</b>
RTL versus SystemC Models.....	97
The noc_sim_wrapper.....	97
Simulation Settings.....	98
Simulating the Design.....	100
<b>Chapter 7: NoC Performance Tuning.....</b>	<b>101</b>

Performance Metrics.....	101
System Design Considerations.....	106
Overview of Performance Tuning.....	121
<b>Chapter 8: Memory Controller Pinout Rules and Future Expansion Options.....</b>	<b>123</b>
Pinout Rules.....	123
Pinout Options for Future Expansion.....	178
<b>Chapter 9: System Address Map.....</b>	<b>202</b>
Address Decoding and the System Address Map.....	202
<b>Appendix A: Memory Interface Debug.....</b>	<b>205</b>
General Memory Debug Checklist.....	205
Vivado Hardware Manager – Memory Debug .....	206
UART Debug.....	214
DDRMC Calibration Debug.....	218
<b>Appendix B: Additional Resources and Legal Notices.....</b>	<b>233</b>
Xilinx Resources.....	233
References.....	233
Revision History.....	233
Please Read: Important Legal Notices.....	234

## IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family <sup>1</sup>	Versal® ACAP
Supported User Interfaces	AXI3, AXI4, and AXI4-Stream
Provided with Core	
Design Files	RTL
Example Design	N/A
Test Bench	Verilog
Constraints File	XDC
Simulation Model	SystemVerilog, SystemC
Supported S/W Driver	N/A
Tested Design Flows <sup>2</sup>	
Design Entry	Vivado® IP integrator
Simulation	For supported simulators, see the <a href="#">Xilinx Design Tools: Release Notes Guide</a> .
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: <a href="#">75764</a>
All Vivado IP Change Logs	Master Vivado IP Change Logs: <a href="#">72775</a>
<a href="#">Xilinx Support web page</a>	

**Notes:**

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the [Xilinx Design Tools: Release Notes Guide](#).

# Overview

---

## Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal® ACAP design process [Design Hubs](#) and the [Design Flow Assistant](#) materials can be found on the [Xilinx.com](#) website. This document covers the following design processes:

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include:
  - [Features](#)
  - [Chapter 3: NoC Architecture](#)
  - [Quality of Service](#)
  - [Performance Metrics](#)
- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
  - [Address Decoding and the System Address Map](#)
  - [Chapter 7: NoC Performance Tuning](#)
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - [Chapter 5: Designing with the Core](#)
  - [Clocking](#)
  - [Resets](#)
  - [Address Decoding and the System Address Map](#)

- **Board System Design:** Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations. Topics in this document that apply to this design process include:
  - [Pinout Rules](#)
  - [Pinout Options for Future Expansion](#)
- **Tutorials:** The following tutorials provide guidance on Network-on-Chip (NoC) and DDR Memory Design and Optimization:
  - [Introduction to NoC/DDRMC Design Flow](#)
  - [Introduction to NoC/DDRMC Performance Tuning](#)

---

## Versal Programmable Network on Chip Overview

The Xilinx® Versal® programmable network on chip (NoC) is an AXI-interconnecting network used for sharing data between IP endpoints in the programmable logic (PL), the processing system (PS), and other integrated blocks. This device-wide infrastructure is a high-speed, integrated data path with dedicated switching. The NoC can be logically configured to represent complex topologies using a series of horizontal and vertical paths and a set of customizable architectural components.

The NoC was designed for scalability. It is composed of a series of interconnected horizontal (HNoC) and vertical (VNoC) paths, supported by a set of customizable, hardware implemented components that can be configured in different ways to meet design timing, speed and logic utilization requirements.

The HNoC and VNoC are dedicated, high bandwidth paths connecting integrated blocks between the processor system and the programmable logic (PL) without consuming large amounts of programmable logic.

The NoC supports end-to-end quality of service (QoS) to effectively manage transactions and balance competing latency and bandwidth requirements of each traffic stream.

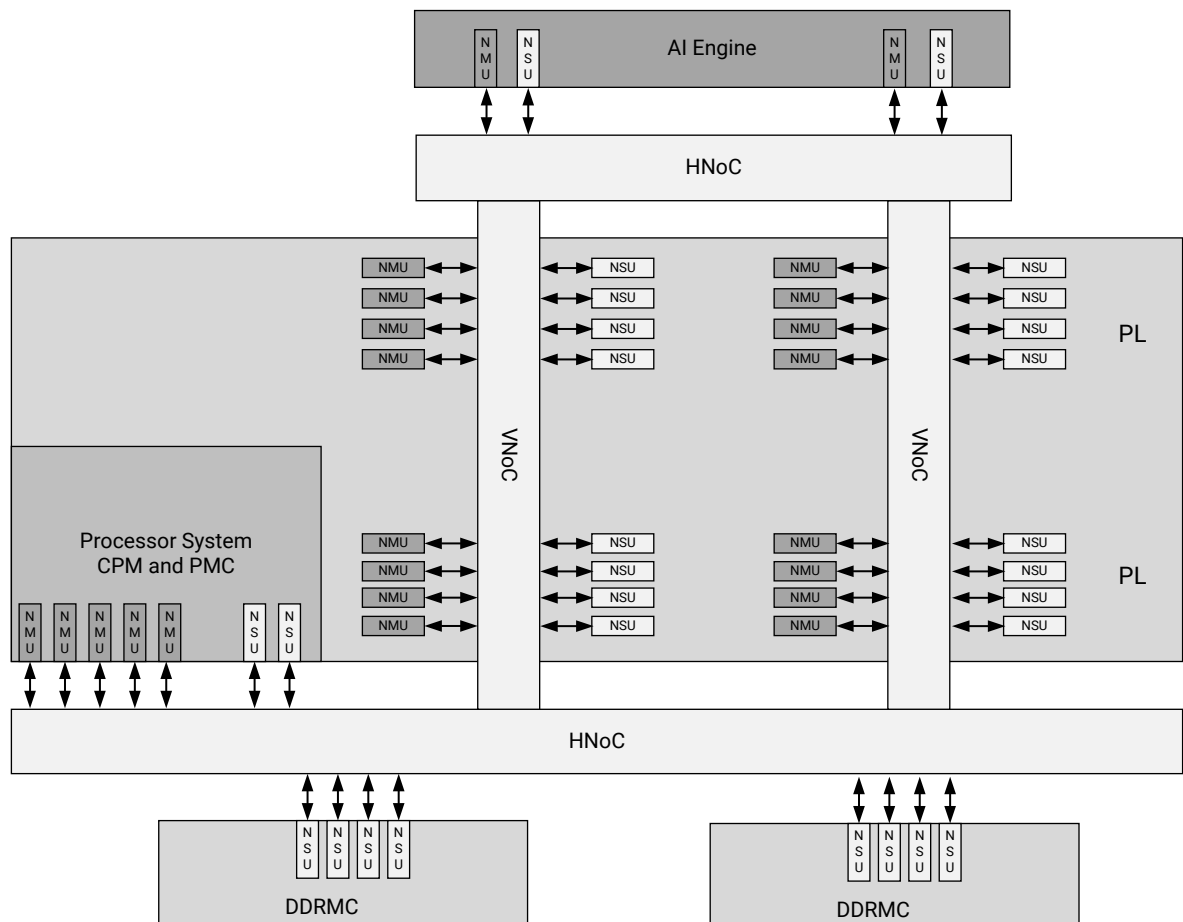
The NoC components comprise NoC master units (NMU), NoC slave units (NSU), NoC packet switches (NPS), and NoC Inter-Die-Bridge (NIDB). The NMU is the traffic ingress point; the NSU is the traffic egress point. All IPs have some number of these master and slave connections. The NIDB connects two super logic regions (SLRs) together, providing high bandwidth between dies. The NPS is the crossbar switch, used to fully form the network.

## Horizontal versus Vertical NoC

As shown in the following figure, NoC paths partition into Horizontal NoCs (HNoCs) and Vertical NoCs (VNoCs). The HNoCs are placed on both the bottom and top of the die. There are four bi-directional physical NoC channels in HNoC. While the bottom HNoC has four bi-directional physical channels in all devices, in the Prime and AI series devices the top HNoC has only two bi-directional physical channels. The bottom HNoC typically connects to a selection of blocks such as PS, PMC, CPM, and DDRMC (Integrated DDR Memory Controller) to list a few. The PS, PMC, and CPM are collectively referred to as Control, Interfaces and Processing System (CIPS). The HNoC consists of NoC components (NMU, NSU, NPS, and more). Similarly, the top HNoC typically connects to a selection of blocks such as DDRMC. For SSIT devices, the top HNoC connects to NoC Inter-Die Bridge (NIDB) providing high bandwidth between two die.

The VNoC refers to the vertical NoC column. There are two bi-directional physical NoC channels in the VNoC. Each Versal device could have more than one VNoC. The VNoC connects to PL. HNoC and VNoC are connected to provide a full network on chip.

Figure 1: NoC Block Diagram



X22049-033021

Design capture is achieved using Vivado® IP integrator, from where you can specify the interconnectivity of all of the endpoints. Virtual Channels (VCs) (not shown) can be used to provide differential quality of service (QoS).

## Features

The NoC is composed of a series of horizontal (HNoC) and vertical (VNoC) paths, supported by a set of customizable, hardware implemented components that can be configured in different ways to meet design timing, speed, and logic utilization requirements. The following features are supported:

- PL to PL communication.
- PL to CIPS communication.
- CIPS to PL communication.
- CIPS to DDR communication.
- CIPS to AI Engine communication.
- High bandwidth data transport.
- Supports standard AXI3 and AXI4 interfaces to the NoC. A soft bridge is required for AXI4-lite support.
- Supports clock domain crossing.
- Internal register programming interconnect for programming NoC registers.
- Multiple routing options:
  - Based on physical address.
  - Based on destination interface.
  - Virtual address support.
- Inter-die connectivity with hardened SSIT bridging.
- Transports bit-stream from source die PMC to PMC in destination die in SSIT configurations.
- Programmable routing tables for load balancing and deadlock avoidance.
- Debug and performance analysis features.
- End-to-end data protection for Reliability, Availability, Serviceability (RAS).
- Virtual channels and quality of service (QoS) are supported throughout the NoC to effectively manage transactions and balance competing latency and bandwidth requirements of each traffic stream:
  - Using ingress rate control, the NoC master unit (NMU) can control the injection rate of packets into the NoC.



- There are eight virtual channels on each physical link. Each AXI request and response occupies a separate Virtual Channel:
  - Each ingress AXI interface (at NMU) can be statically programmed to select the virtual channel it maps to.
  - Virtual channel mapping can be re-programmed (the NMU must be quiesced first).
  - All AXI QoS values are optionally carried through the NoC.
- The NoC connection hardware (or access points) use a master-slave, memory mapped configuration. The most basic connection over the NoC consists of a single master connected to a single slave using a single packet switch. Using this approach, the master takes the AXI information and packetizes it for transport over the NoC to the slave, via packet switches. The slave decomposes the packets back to AXI information delivered to the connected back-end design. To achieve this, a NoC access point manages all clock domain crossing, switching, and data buffering between the AXI and NoC side and vice versa.
- Error-Correcting Code (ECC) is supported for memory mapped transactions (ECC of AXI4-Stream is not supported).

The NoC functional blocks are as follows:

- **NoC Master Unit (NMU):** Used to connect a master to the NoC.
- **NoC Slave Unit (NSU):** Used to connect a slave to the NoC.
- **NoC Packet Switch (NPS):** Used to perform transport and packet switching along the NoC and to set up and use virtual channels.

NMU and NSU components are accessed from the programmable logic side through standard AXI3 or AXI4 interfaces using the following basic AXI features:

- AXI3, AXI4, and AXI4-Stream support.
- Configurable AXI interface widths: 32, 64, 128, 256, or 512-bit interfaces.
- 64-bit addressing.
- Handling of AXI exclusive accesses.
  - The exclusive access mechanism can provide semaphore-type operations without requiring the bus to remain dedicated to a particular master for the duration of the operation. This means the semaphore-type operations do not impact either the bus access latency or the maximum achievable bandwidth.

For all AXI features see *Vivado Design Suite: AXI Reference Guide* ([UG1037](#)).

Each Versal® device provides dedicated, hardware-constructed DDR memory controllers integrated into the NoC structure. The DDR memory controller interface contains four dedicated NSU controllers. The DDR controllers are configured using the NoC IP Wizard. To make optimal use of the available NoC features, the NoC structure provides support for interleaving across multiple physical DDR controllers (two or four).

- The NMU handles chopping and ordering needed to support DDR controller interleaving.
- A transaction targeting interleaved DDR controllers is handled as follows:
  - The transaction is chopped into smaller packets to align with the interleave granule and memory space of each physical controller.
  - Each sub-packet is addressed separately to align to the correct physical DDR interface.
  - Responses are re-assembled at the NMU and returned to the attached master.

## NoC Functions

The NoC components comprise:

- **NoC Master Unit (NMU):** The ingress block that connects a master to the NoC.
- **NoC Slave Unit (NSU):** The egress block that connects the NoC to a slave.
- **NoC Packet Switch (NPS):** The switch block that connects NoC blocks to form the full NoC network.
- **NoC Inter-Die Bridge (NIDB):** The block that bridges Vertical NoC (VNoC) between multiple Stacked Silicon Interconnect Technology (SSIT) dies.
- **NoC Clock Re-convergent Buffer (NCRB):** The block that handles clock skew at HNoC to VNoC re-convergent points.

Summary of some additional NoC terminology:

- NoC Peripheral Interconnect (NPI), the internal register programming interconnect of the NoC.
- NoC Packet Protocol (NPP).
- NoC Peripheral Interconnect Root (NIR), the NPI ingress block that connects to PMC.
- NoC Peripheral Interconnect Switch (NIS), the NPI switch block that switches NPI packets.
- NoC Peripheral Interconnect Protocol Unit (NIP), the NPI egress block that connects the NPI to NoC component or Hard IP registers. This block can be standalone or integrated into Hard IP.

---

# DDR Memory Controller

The integrated DDR Memory Controllers (DDRMCs) support both DDR4 and LPDDR4/4X memory interfaces. It has four programmable NoC interface ports and is designed to handle multiple streams of traffic. Five Quality of Service (QoS) classes are available to ensure appropriate prioritization of commands. The controller accepts burst transactions and implements command reordering to maximize efficiency of the memory interface. Optional external interface reliability features include ECC error detection/correction and command address parity.

## Memory Controller Feature Summary

- DDR4 and LPDDR4/4X protocols.
- Component, SODIMM, UDIMM, RDIMM, LRDIMM topology support.
- Up to x64 data width or x72 for ECC interfaces.
- Up to x32 data width for dual channel configurations.
- Supports up to 4H DDR4 3DS Logical Ranks.
- Supports Multi-Rank and Dual Slot topologies.
- Quality of Service (QoS) classes:
  - Read: Isochronous, Low Latency, Best Effort.
  - Write: Isochronous, Best Effort.
- Out-of-order execution of commands for enhanced SDRAM efficiency.
- ECC support: Single Error Correct, Dual Error Detect:
  - On-the-fly scrubbing.
  - Background scrubbing.
  - Correctable and uncorrectable error logging.
  - Error injection for writes.
- Address Parity.
- Data Mask and Dynamic Bus Inversion (DBI).
- Fixed Burst Length 8 for DDR4.
- Fixed Burst Length 16 for LPDDR4.
- 1T or 2T timing for Address/Command bus.

- Refresh support for 1x, 2x, and 4x rates.
  - No support for on-the-fly refresh rate control as a function of temperature.
- 2x and 4x fine granularity refresh modes for DDR4.
- Per-Bank refresh for LPDDR4/4X.
- Read-After-Write and Write-After-Write hazard checking.
- AXI ID ordering.

## Memory Configuration Support

**Table 1: DDR4 Component Interface**

DDR Channels	Data Bus Width	Data Bus Width with ECC	DRAM Component	Max. Number of Ranks
1	64	72	DDR4: x8, x16 DDR4 3DS: x8	1 SDP, 2 DDP
1	32	40	DDR4: x4, x8, x16 DDR4 3DS: x4, x8	1 SDP, 2 DDP
1	16	24	DDR4: x4, x8, x16 DDR4 3DS: x4, x8	1 SDP, 2 DDP
2 <sup>2</sup>	32	N/A	DDR4: x8, x16 DDR4 3DS: x8	1 SDP, 2 DDP
2 <sup>2</sup>	16	24	DDR4: x8, x16 DDR4 3DS: x8	1 SDP, 2 DDP

**Notes:**

1. For any component interface, all memory components must be identical.
2. The read reordering queue for dual channel configuration is shared and read efficiency will be affected.
3. Multi-rank for component interfaces is only supported for dual-die packages. Sharing the DQ bus with multiple components as separate physical ranks on the PCB is not supported.

**Table 2: DDR4 DIMM Interface**

Data Bus Width	Data Bus Width with ECC	DIMM Type	Max. Number of Physical Ranks per DIMM	Max Number of Slots
64	72	RDIMM with DDR4 or DDR4 3DS	2	2
64	72	UDIMM/SODIMM with DDR4	2	1
64	72	LRDIMM with DDR4 or DDR4 3DS	4	1
64	72	LRDIMM with DDR4 or DDR4 3DS	2	2

**Notes:**

1. For any DIMM interface with multiple slots, all DIMMs must be identical.

**Table 3: LPDDR4/4X Component Interface**

LPDDR4/4X Channels	Data Bus Width	Data Bus Width with ECC	DRAM Component <sup>1</sup>	Max. Number of Ranks <sup>3</sup>
1	32	48	LPDDR4/4X : x32, x16	2
1	16	32	LPDDR4/4X : x32, x16	2

Table 3: LPDDR4/4X Component Interface (cont'd)

LPDDR4/4X Channels	Data Bus Width	Data Bus Width with ECC	DRAM Component <sup>1</sup>	Max. Number of Ranks <sup>3</sup>
2 <sup>2</sup>	32	N/A	LPDDR4/4X : x32, x16	2
2 <sup>2</sup>	16	32	LPDDR4/4X : x32, x16	2

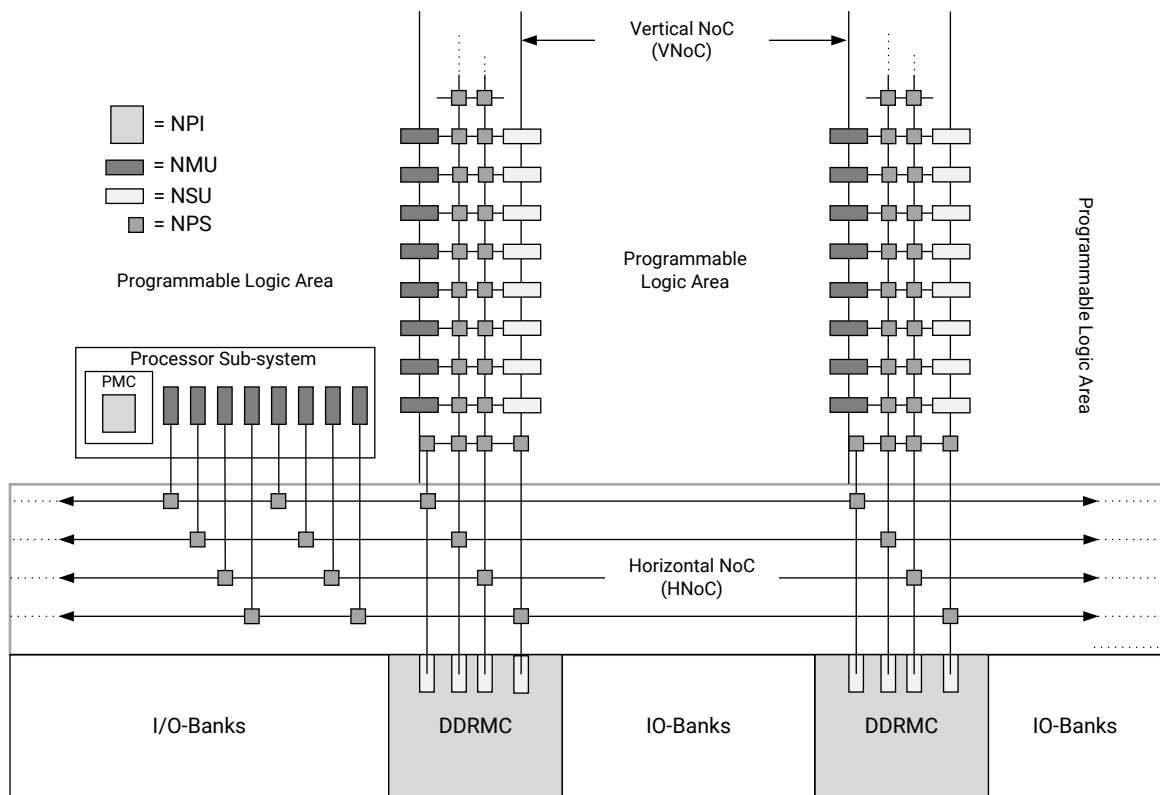
**Notes:**

1. For any component interface, all memory components must be identical on a x16 channel basis. A mix of dual-channel (x32) devices and single-channel (x16) devices can be used, but the memory density per x16 channel must match between all devices. Byte-mode devices are not supported.
2. The read reordering queue for dual channel configuration is shared and read efficiency will be affected.
3. Dual-rank for component interfaces is only supported for dual-die packages. Sharing the DQ bus with multiple components as separate physical ranks on the PCB is not supported.

## NoC Architecture

The following figure is a representation of the NoC. It shows how various elements are connected to construct a NoC system and how the NoC is incorporated within the device.

**Figure 2: NoC Block Diagram**



X22054-033021

As shown, the NoC system is a large-scale interconnection of instances of NoC master units (NMUs), NoC slave units (NSUs), and NoC packet switches (NPSs), each controlled and programmed from a NoC programming interface (NPI). To complete the system, NoC inter-die bridges (NIDBs) (not shown) are added to enable Stacked Silicon Interconnect Technology (SSIT).

The Versal® Programmable NoC is statically routed by Vivado® software at design time. That is, the assignment of NoC ingress and egress points to specific NMUs and NSUs and the routing paths to implement connections between ingress and egress are computed at design time by the NoC compiler, part of the Vivado® Design Suite. The NoC compiler considers the connectivity of the design and quality of service constraints supplied by the designer to solve for a globally optimal solution. This solution is then expressed as configuration data in the final application PDI files.

The NoC and DDRMC system must be configured/programmed from the NPI at early boot and be ready before NoC data paths or DDRMC are used. The NPI programs NoC and DDRMC registers that define the routing table, rate modulation, QoS configuration, and timing parameters. Programming of the NoC and DDRMC from the NPI requires no user intervention, it is fully automated and executed by the platform management controller (PMC). For more information about boot and configuration refer to the *Versal ACAP Technical Reference Manual* (AM011).

## NoC Components

### NoC Master Unit

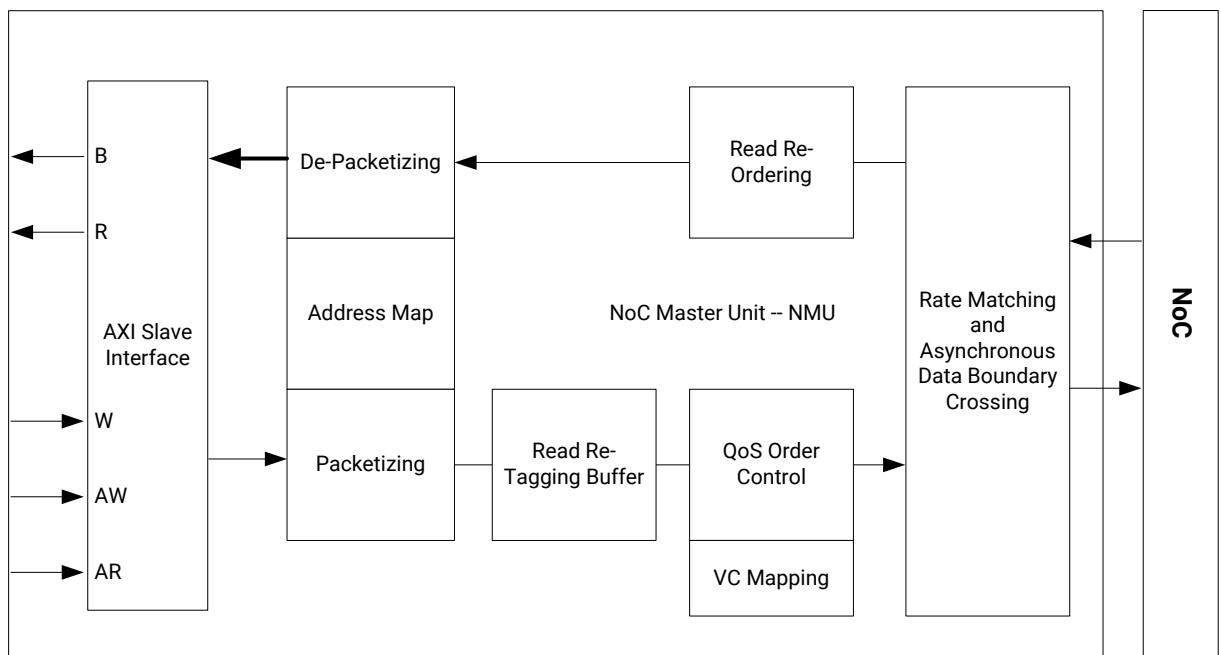
The NoC master unit (NMU) is the ingress point to the NoC. The NMU provides:

- Asynchronous clock domain crossing and rate matching between the AXI master and the NoC.
- Conversion from/to AXI protocol to NoC Packet Protocol (NPP).
- Address matching and route control.
- WRAP burst support for 32, 64, and 128-bit interfaces.
- INCR and FIXED burst support.
- Read re-tagging to allow out of order service and prevent interconnect blocking.
- Write order enforcement.
- Ingress QoS control.
- Handling of the AXI exclusive access feature.
- Support for configurable data width from 32 to 512-bit on AXI interfaces and 128 to 512-bit on AXI4-Stream interfaces. AXI data width is configured via parameter propagation from the connected IP.
- AXI3, AXI4, and AXI4-Stream support.
- Acceptance of up to 32 AXI reads and 32 AXI writes.

- Support for up to 64 outstanding NPP writes. The maximum size of a NPP write is 256 bytes. An AXI write that is more than 256 bytes may span multiple NPP writes.
- Support for up to 64 outstanding NPP reads of 32 bytes each. The read re-order buffer (RROB) holds 64 32-byte entries. An AXI read that is more than 32 bytes consumes multiple entries.
- DDR controller interleaving support at 128B – 4 KB interleave granularity.
- Programmable virtual channel mapping.
- The NMU is available in two variants:
  - **Full functionality:** All the above specifications apply, the NMU is used on the programmable logic.
  - **Latency-optimized:** Fixed 128-bit wide AXI interface and all transactions are address route based.

**Note:** Integrated blocks (CIPS and AI Engine) use latency-optimized NMU/NSU blocks while the PL uses full functionality blocks.

Figure 3: NoC Master Unit



X21775-102419

The NMU is located at the transaction initiator side of the system. It is equipped with a standard AXI4 interface which includes some optional sideband signals providing additional addressing and routing controls. As shown in the previous figure, an asynchronous data crossing and rate matching block form the interface between the NoC and application (AXI) side of the master unit. The Rate Matching buffers write data from the slow application domain until there is enough payload to prevent write bubbles.



Data packetizing is performed when AXI requests enter the NMU clock domain. As part of the packetizing process, read and write transactions are broken into smaller transfers (this process is called chopping). Chopping is always performed on chop-size aligned boundaries. Two parameters affect chopping: chop size (fixed at 256 bytes) and memory interleave size (when two DDR controllers are interleaved). If memory interleave granularity is smaller than 256 bytes, reads and writes are chopped into transfers equal to the interleave granularity. Non-interleaved transactions, or transactions with interleave granularity greater than or equal to 256 bytes are chopped into 256-byte transfers. Each chopped transaction is divided into NoC packets, or "flits". Each flit can carry up to 16 bytes of data in addition to various header information.

In parallel with the packetizing process, address lookup is performed to determine the destination ID.

In read processes, read re-tagging is performed on each read packet and an ordering ID is assigned based on available slots in the Read Reorder Buffer (RROB). The RROB maintains a linked list of per-AXI-ID assigned tags, allowing responses to be returned in the correct order.

The final stage before a packet is injected into the NoC switch fabric is to perform access QoS control.

Read responses are placed in the RROB. In accordance with AXI ordering rules, logic at the output of the buffer selects read responses to return to the requesting AXI master. This logic relies on the linked list structure from the request path to determine the correct response ordering.

## AXI Memory Mapped Support

### Addressing

The NoC supports a rich set of addressing mechanisms. Every configured NMU and NSU in a design is assigned a unique 12-bit ID referred to as the destination ID (destID). When a packet arrives at an input port of a NoC packet switch (NPS) the destID and the virtual channel number of the packet are used to look up into the routing tables to find the output port to send the packet. The output port determines the next segment of the route, delivering the packet to the next NPS or NMU/NSU.

An NMU in AXI3 or AXI4 memory mapped mode may be configured to use one of four strategies to determine the destID. These strategies in order of precedence are:

- **Fixed DestID:** A configuration register in the NMU may be programmed to contain a single destination ID. When enabled, the programmed value specifies the destination for all memory mapped transactions originating from that NMU, regardless of address.
- **User-defined DestID:** User logic in the PL region may drive the destination ID onto the destination interface pins of the NMU. This ID is then used to route the memory mapped transaction to the destination. Separate destination interface pins are provided for read and write transactions. This feature is not supported in the current release of Vivado.

- **Address remap:** The NMU supports an address remapping mechanism to override the default address map, provide a programmed destination ID, and support simple address virtualization.
- **Address decode:** The NMU may decode the physical address of the AXI transaction to determine the destination ID of the addressed resource. The NMU contains a set of address decoders and associated programmable registers to allow transactions destined for different regions of the physical address space to be mapped to preprogrammed destIDs.

### Fixed Destination ID

The NMU can be configured to drive a single destination ID for any memory mapped AXI transaction, routing every transaction to the same destination regardless of the AXI address. This mode may be used to route virtually addressed transactions from an upstream master to a memory management unit such as the CIPS SMMU for address translation. Once the virtual address is translated to a physical address the transaction may be routed to destinations within the CIPS or reinjected into the NoC to be routed to other memories or NSUs.

### NoC Master Specified Destination ID

Logic in the PL region may optionally drive the destination ID onto a set of pins on the NMU boundary. This ID is then used to route the transaction through the NoC to the intended NSU. The destination interface pins are shown in the following table.

*Table 4: Destination Interface Pins*

Interface Pin	Description
NMU_RD_DEST_MODE	Enable NMU_RD_USR_DST
NMU_RD_USR_DST[11:0]	12-bit destination ID to use for read transactions
NMU_WR_DEST_MODE	Enable NMU_WR_USR_DST
NMU_WR_USR_DST[11:0]	12-bit destination ID to use for write transactions

When a destination interface is enabled by driving the RD\_DEST\_MODE or WR\_DEST\_MODE pin high, the value on the corresponding USR\_DST pins will override any address decode.

**Note:** NoC Master Specified Destination ID routing is not supported in the current release.

### Address Re-mapping

The NMU supports address remapping as a way to override the default address map as well as to provide simple address virtualization. There are sixteen sets of address remap registers in the configuration space of the NMU. Each address remap register supports:

- Matching address ranges from 4 KB to the full address map;
- Address matching to determine the destination ID to route the transaction;
- Optional replacement address for the matched portion of the address.

Each remap register contains of the following fields:

- ENABLE
- REMAP
- MASK[63:12]
- MATCH[63:12]
- REPLACE[63:12]
- DEST[11:0]

When an address matcher is enabled every incoming AXI read and write address is compared. The bits where MASK is set to 1 are compared between the AXI address and the corresponding bits in the MATCH field. If the REMAP field is set and the two address values match, the MASK bits in the AXI address are replaced with the corresponding values (in the same bit positions) from the REPLACE field and the value of the DEST field is used as the destination ID. If the REMAP field is not set, the original address is used unmodified and the DEST value is used as the destination ID.

### Address Decoding and the System Address Map

The Versal® ACAP programmable NoC system address map defines the default address locations of slaves in the Versal device. The address map is built into the integrated interconnect of the NoC. The NoC provides some capabilities to perform address re-mapping which allow the address map to be customized to the target application.

Refer to [Address Decoding and the System Address Map](#) for further information.

### Memory Controller Interleaving

The NMU supports interleaving memory transactions across two or four hardened memory controllers. The memory regions can be programmed to be interleaved at a granularity of 128 bytes to 4 KBytes, or may be programmed to disable interleaving. The choice of whether to interleave, the interleave granularity, and the address of the interleaved region are determined at the time the NoC is configured. The interleave granularity must match the stripe size of the CCI-500 if the PS full power domain is enabled. The default stripe size for the CCI-500 is 4kB.

When configured to support interleaved memory controllers, the NMU is configured to stripe transactions bound for the DDR across the two or four memory controllers as follows:

1. The transaction is chopped into smaller packets to align with the memory space of each physical channel. Packet chopping occurs on the interleave boundary between each memory channel.
2. Each sub-packet is addressed separately to the correct physical DDR controller.
3. Responses are re-assembled at the NMU and returned to the attached master.

## Read Reorder Buffer

The NMU contains a read reorder buffer (RROB) that can accept returned data from all issued read transactions. An ordering tag is assigned based on available slots in the RROB. The RROB maintains a set of per-AXI ID linked lists of assigned tags enabling the responses to be returned in the correct order. Upon receiving an AXI read request an ordering ID is assigned based on the available slots in the RROB. Each linked list stores the original AXI ID so that it can be correctly returned with the response transaction. The RROB can hold up to 64 entries of 32 bytes per entry. Read responses are placed in the RROB as they arrive. Logic at the output of the buffer selects responses that can be returned in the correct order according to AXI ordering rules. This logic relies on the linked list structure from the request path to determine the correct response ordering.

The maximum number of outstanding entries is programmable from 2 to 64, with a default value of 64.

## Outstanding Transaction Support

The NMU provides a register indicating the number of pending transactions. This is the number of original AXI transactions received on the AXI interface that have not yet completed. A transaction completes when the final valid/ready handshake of the last response occurs on the AXI interface.

There are separate pending transaction registers for read and write transactions. There is a separate register to indicate whether any read or write transactions are pending. This logic is used for cases where the NoC may need to be powered down or the NMU quiesced before reprogramming takes place.

The NMU also provides a control register to prevent any new transactions from entering the NMU. If there are any writes where the AW phase has been received then the W phases are allowed to complete before the interface blocks.

## Write Response Tracker (Single Slave per AXI ID)

The NMU contains a 64 entry write tracker buffer. This buffer tracks the set of responses received for each chopped write transaction and returns a single write response to the master when all of the chopped responses have been received.

The write tracker is also responsible for ensuring that only transactions to a single slave per AXI-ID is allowed to be outstanding at any given time (SSID check). Back pressure will be applied on a write request if there is an outstanding write transaction with the same AXI-ID but to a different destination.

The maximum number of outstanding write requests is programmable from 2 to 64, with a default value of 64.

## AXI4-Stream Support

An NMU may be configured to support the AXI4-Stream protocol. Please refer to the *AMBA AXI4-Stream Protocol Specification* ([ARM IHI 0051A](#)) for rules and description of the protocol. In AXI4-Stream mode the NMU can act as a slave to carry one uni-directional stream connection with the following constraints:

- Source data width can be 128-512 bits.
- Source and destination data widths must match each other.
- The TKEEP signal is supported and can be used to send partial transfers.
- The TSTROBE signal is not supported.
- The TID signal is supported and can be used to disambiguate source streams. The TID field is 6 bits wide.
- The TDEST field is 10 bits wide. TDEST may be used for local routing after egress from the NoC. The TDEST field is 10 bits wide.
- Stream packets are not ECC protected in the NoC. If data protection is required it must be embedded in the data stream.
- The stream source does not have complete knowledge of the data widths of all the slaves it communicates with. The packing rules assume the worst case (512 bits) to ensure correct operation under all conditions. If the NoC destination data width is >128 bits, the stream beats must be pre-arranged to be able to pack together. Stream packets should be sent such that the source data beats can be packed together and form an integer number of destination stream packet beats. If the number of beats with the same TID/TDEST is not 64-byte aligned, the last beat can have TLAST=1, but this may result in inefficient data packing.

The NoC will pack stream transactions together but has no functionality to reorder based on TID around width conversion points. This leads to a requirement that the sender must group transactions with the same TID to take into account of width conversions. The transactions can either be grouped together as  $n$  beats or a group can be terminated with TLAST, leading to less efficient data packing.

The minimum size  $n$  that the sender must adhere to depends on the configured data width at the NMU and NSU and takes into account the 128-bit NoC transport in between.

### Destination ID

An NMU in AXI4-Stream mode may be configured to use one of two strategies to set the destination ID:

- **Fixed DestID:** A configuration register in the NMU may be programmed to contain a single destination ID. When enabled, the programmed value specifies the destination for all stream packets originating from that NMU. This enables 1 to 1 and N to 1 connection topologies.

- **Master defined DestID:** Logic in the PL region may optionally drive the destination ID onto the destination interface pins of the NMU. In AXI4-Stream mode, only the NMU\_WR\_DEST\_MODE and NMU\_WR\_USR\_DST pins of the destination interface are used. This ID is then used to route the stream through the NoC. This enables N to M connection topologies.

**Note:** This feature is not supported in the current release of Vivado

## Error Conditions

The NMU is able to detect and report a number of error conditions in both the protocol and packet domains. Each error type can be enabled or disabled individually. When an error occurs certain data is logged and an interrupt may be raised.

The NMU error conditions fall into the following categories:

- AXI address parity errors. Address parity is checked separately for read and write transactions.
- AXI write data parity errors.
- AXI protocol rule violations.
- Timeouts.
- NoC packet ECC errors. Single-bit correctable and double-bit non-correctable errors are caught and recorded.
- Virtual channel credit overflow and underflow.
- Address decode and remap errors.

**Note:** While the NMU supports detection and reporting of these error conditions, there is currently no software support for these features.

## NMU Versions

### NMU512 (PL)

The NMU512 is a full-featured NoC Master Unit, used to provide ingress to the vertical NoC (VNoC) channels from AXI master units in the Programmable Logic (PL) array. The NMU512 has a maximum AXI data width of 512 bits (64 bytes) and is configurable from 32 bits to 512 bits in AXI memory mapped mode, and 128 bits to 512 bits in AXI4-Stream mode. The NMU512 supports AXI3, AXI4, Memory Mapped, and AXI4-Stream protocols.

### NMU128 (Low Latency)

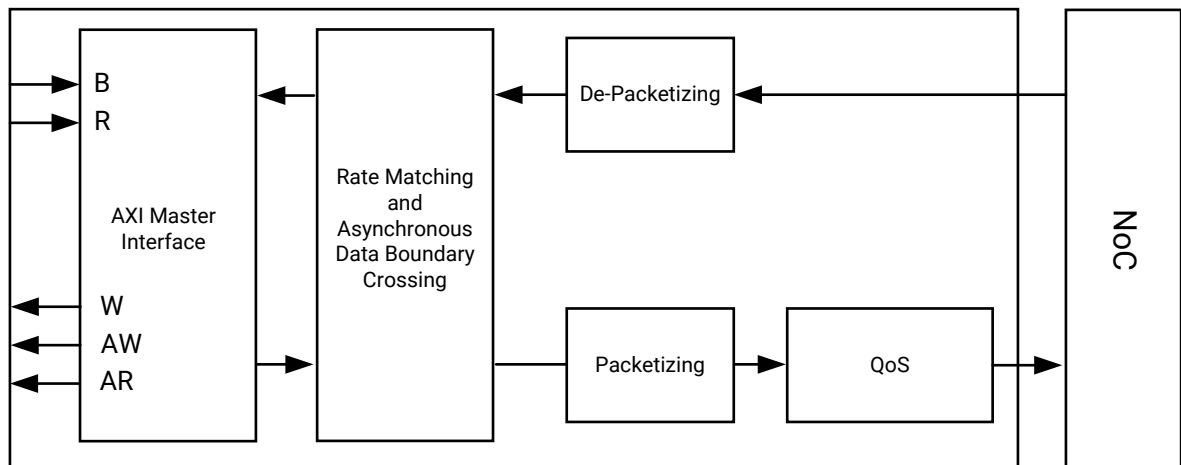
The NMU128 is optimized for the low latency requirements of hardened blocks such as the CIPS. The NMU128 has a fixed 128-bit AXI data width. It does not support the AXI4-Stream protocol and does not support master-defined destination IDs. Otherwise, it supports all of the features of the NMU512.

## NoC Slave Unit

The NoC slave unit (NSU) is the egress point to the NoC. The NSU provides:

- Conversion of NoC packetized data (NPD) to and from AXI protocol data.
- Asynchronous clock domain crossing and rate-matching between the AXI slave and the NoC.
- AXI exclusive access handling.
- Configurable AXI interface widths of 32, 64, 128, 256, or 512 bits. AXI4-Stream interfaces support 128, 256, and 512-bit widths. AXI data width is configured via parameter propagation from the connected IP.
- Support for AXI3, AXI4, and AXI4-Stream.
- AXI-ID compression.
- Programmable support for 32 outstanding read and 32 outstanding write transactions.
- 32 entry interleaved read tracker and 32 entry chop-merge write tracker.
- Support for two virtual read and two virtual write channels.
- Response to QoS control.
- Two versions of the NoC slave unit block are available:
  - A *fully featured* version used for all interface constructs except for memory.
  - A *memory specialized* version omitting the AXI interface. This unit is integrated in the hard memory controllers at the NoC endpoints of the device.

Figure 4: NoC Slave Unit



X21776-102419

The main function of the NSU is to receive and respond to packets from the NoC that are addressed to the NSU packet interface and intended to go to the attached AXI slave.

As shown in the previous figure, the NSU logic de-packetizes the received NoC data packets and converts them into AXI transactions. The re-created AXI transaction passes through the buffered asynchronous data crossing and rate-matching (fast to slow) logic to the AXI master interface where the request is processed and a response accepted. The packetizing block receives AXI responses through the clock domain crossing and rate-matching logic (between the slow and fast domains). The read responses are buffered before forwarding to minimize bubbles (stalls) in the read responses.

### ***Exclusive Monitor (AXI Exclusive Access Support)***

The NoC supports AXI exclusive accesses. The packetized transactions sent into the NoC carry the original 16-bit AXI ID that was presented to the NMU in the packet header.

For exclusive accesses to DDR the NMU will preserve the original size attributes of the transaction if the transaction length is 1 and the size is 128 bits or less. For all other transactions to DDR, the NMU will treat the transaction as modifiable, pack the data together, and modify the size to 128 bits.

The NSU contains a single AXI monitor. The monitor uses the 10-bit SMID in the header as the ID for comparisons. An exclusive access is sent out as exclusive on the NSU AXI interface. An exclusive write access that fails the exclusive check in the monitor is suppressed by the NSU and will not be seen on the NSU AXI interface.

The end-to-end processing of an exclusive access proceeds as follows.

At the NMU:

- The NMU receives the 16-bit AXI ID from the master.
- For reads, the AXI ID is stored in the RROB indexed by the read tag.
- The SMID, AXI ID, and the tag value are transmitted in the NoC packet header.

At the DDR memory controller NSU (MC):

- The MC uses the SMID to check the exclusive monitor. The SMID may also be used for security and access permission checks.
- For reads, the combination of read tag and NoC source ID will be unique at the MC for each packet. The MC will reorder based on this combination.
- For writes, the tag is ignored and only the NoC source ID is used for reordering.

At a non-MC NSU:

- The NSU uses the SMID for exclusive monitor check.
- The NSU creates a master interface AXI ID by selecting two bits from the NoC source ID. This allows the NSU to reorder transactions from different masters, but does not reorder transactions from the same master.



- The NSU stores the NPP tag and returns it with the response.

## AXI ID Compression

The NoC request packet from the NMU carries the triple {Source ID, AXI-ID, Tag} for read requests and {Source ID, AXI-ID} for write requests. Non-DDR memory controller NSUs send a compressed AXI-ID to the downstream AXI slave. There are two programmable compression modes:

- A 2-bit compressed AXI-ID is created by choosing any two bits of the {Source ID, AXI-ID}. The bit selection is programmed as part of the NSU configuration..
- A 2-bit fixed value determined by a configuration register. This mode effectively forces read and write requests to remain in order.

By default the NSU will select the least significant two bits of the request AXI-ID as the compressed ID. DDR memory controller NSUs do not compress the AXI-ID.

## Error Conditions

The NSU is able to detect and report a number of error conditions in both the protocol and packet domains. Each error type can be enabled or disabled individually. When an error occurs certain data is logged and an interrupt may be raised.

The NSU error conditions fall into the following categories:

- AXI read data parity errors.
- AXI protocol rule violations.
- Timeouts.
- NoC packet ECC errors. Single-bit correctable and double-bit non-correctable errors are caught and recorded.
- Virtual channel credit overflow and underflow.

**Note:** While the NSU supports detection and reporting of these error conditions, there is currently no software support for these features.

## NSU Versions

### NSU512 (PL)

The NSU512 is a full-featured NoC Slave Unit, used to provide egress from the vertical NoC (VNoC) channels to AXI slave units in the Programmable Logic (PL) array. The NSU512 has a maximum data width of 512 bits (64 bytes). It is configurable from 32 bits to 512 bits in AXI memory mapped mode, and 128 bits to 512 bits in AXI4-Stream mode. The NSU512 supports AXI3, AXI4, Memory Mapped and AXI4-Stream protocols.

### **NSU128 (Low Latency)**

The NSU128 is optimized for the low latency requirements of hardened blocks such as the CIPS. The NSU128 has a fixed 128-bit AXI data width and does not support the AXI4-Stream protocol. Otherwise, it supports all of the features of the NSU512.

### **DDRMC-NSU**

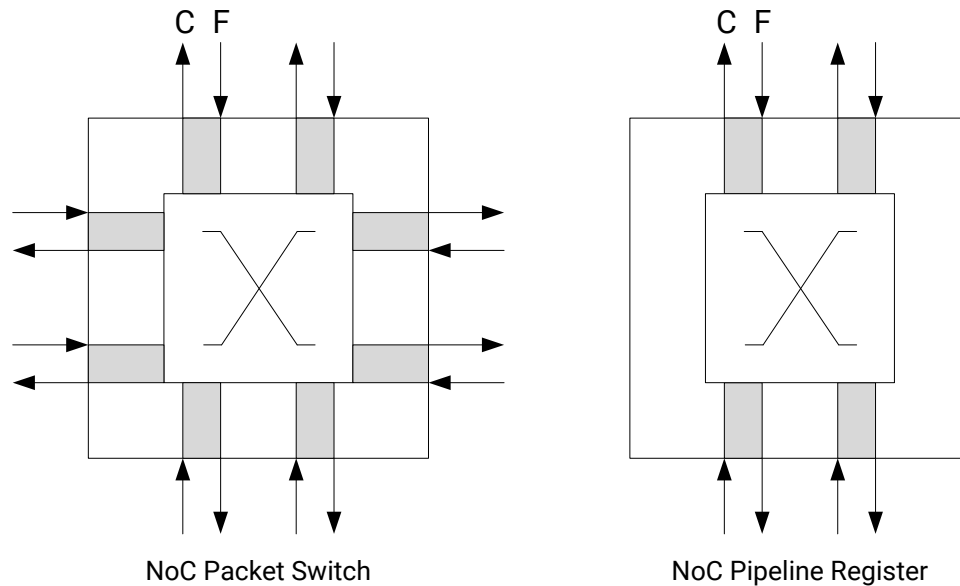
Each DDR memory controller has a partial NSU (DDRMC-NSU) for each port. The DDRMC-NSU serves to convert from the NoC packet domain to the memory controller domain without first converting to AXI protocol.

## **NoC Packet Switch**

NMU and NSU interfaces are connected in the NoC by NoC Packet Switches (NPSs) as shown in the following figure. The NPS features are as follows:

- Each NPS is a full-duplex 4x4 switch.
- Each port supports eight virtual channels in each direction.
- Each port is fully buffered and contains eight FIFOs (one per virtual channel).
- The switching system uses a credit-based flow control.
- A minimum of two latency cycles through the switch.
- Configurable QoS.
- Programmable routing table per input-port / per virtual channel:
  - The routing table is programmed through the NPI at boot-time.
  - The routing table can be re-programmed if the NoC is on hold (quiesced).

Figure 5: NoC Packet Switch



F = Incoming data flit or valid state  
C = Control Credit state

X22050-092319

Incoming packets are received by the switch. The receiving switch sends flow control credits back to the transmitter switch. The flow control credits are the guarantee that the switch will accept the data transaction, and are therefore important for switch-to-switch communication.

When data arrives at a switch, it checks the destination ID in the packet header to determine the output port of the switch to route the packet to. The destination ID is checked to flag any fatal error conditions if a packet is mis-routed. The buffered destination address is used in a routing table to select the required switch output port.

An incoming data transaction is used in output arbitration when the corresponding virtual channel buffer is empty; otherwise the data is stored in the virtual channel buffer for subsequent virtual channel arbitration. Each cycle arbitration candidate from each input port is presented at each output port of a switch.

Each virtual channel has an associated FIFO, therefore there are eight FIFOs available per switch port.

- The HNoC ports have a seven deep FIFO.
- The VNoC ports have a five deep FIFO.

## ***Credit Based Flow Control***

Each NMU, NPS, and NSU source needs to have credit before it can send data to the receiver. After a reset, every NoC component has its source-credit reset to zero. The source unit connects to the destination unit using a bi-directional ready signal that indicates credit exchange is ready. Components wait until both directions are ready before starting the credit exchange.

The destination unit can send up to one credit per cycle, per virtual channel, to the source unit. The source unit can send up to one data transaction per cycle to the destination unit.

## ***NoC Pipeline Register***

In some Versal devices where the physical distance between NoC switches (NPS blocks) is long, a NoC Repeater (RPTR) block is inserted. Each NoC Repeater in a path adds one NoC clock cycle of latency, but is otherwise transparent to the user.

## ***Error Conditions***

The NPS can detect and report a number of error conditions in the packet domain. Each error type can be enabled or disabled individually. When an error occurs, certain data is logged and an interrupt may be raised.

The NPS error conditions fall into the following categories:

- Virtual channel credit overflow and underflow
- NoC packet routing errors
- Output port parity errors

**Note:** While the NPS supports detection and reporting of these error conditions, there is currently no software support for these features.

# **Differentiated Quality of Service**

## ***Virtual Channel Arbitration***

For every cycle, each output port performs Least Recently Used (LRU) arbitration on all virtual channels of the three input ports. High priority requests always take precedence over low priority requests, however a virtual channel must meet the following conditions to be eligible for arbitration:

- Each request must be valid.
- The output port must have credit to send a packet on to the next NPS.

- There must be no blocking from the same virtual channel of another input port:
  - Each multi-data transaction in the output-port-arbitrator is hard-locked such that when one of the three input ports has granted access to a virtual channel, the whole transaction must be completed before switching to the same virtual channel on another input port. This ensures that write transactions are not interleaved.
- A virtual channel token must be available.
  - Each output port contains one token for each input port, therefore each output port contains 24 tokens (three ports × eight virtual channels). Each time the arbiter issues a grant to a virtual channel, its corresponding token is decremented. The tokens are loaded from NPI programmed registers when both of the following conditions are true:
    - At least one virtual channel is requesting with zero tokens.
    - There are no other pending request with one or more tokens.
  - NPI token registers should never be programmed to zero as this would cause blocking of the corresponding virtual channel. NPI token registers can be re-programmed at any time; the new value takes effect when the next token reload occurs.
- There are no higher priority virtual channel requests.

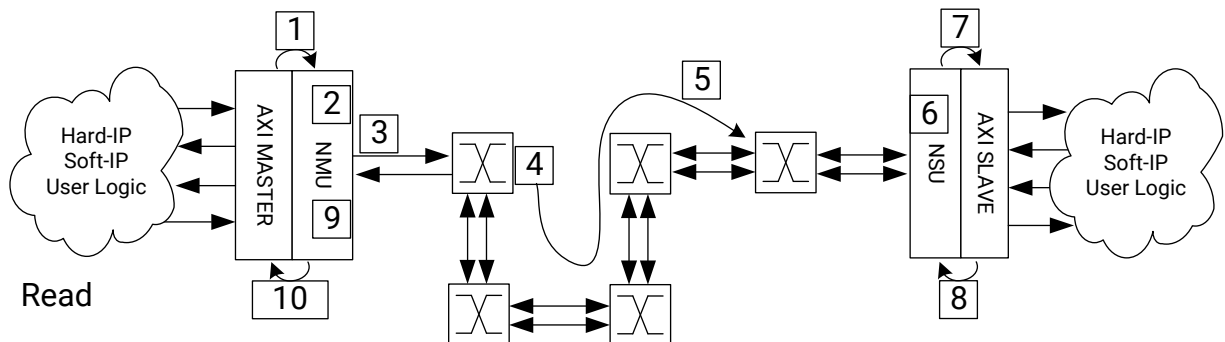
## NoC Communication

Basic NoC read and write transactions are shown in the following figures. A typical transaction can be summarized as follows:

1. An AXI master sends read/write requests to a connected NoC access point (NMU).
2. A PL AXI master should enable traffic only when all of the PL AXI slaves it communicates with are out of reset.
3. The NMU relays the requests through a set of NoC packet switches (NPSs) before the requests reach a destination (NoC slave unit NSU or output port).
4. The NSU passes the received requests to the attached AXI slave for processing.
5. While processing the requests, the AXI slave sends read/write responses back to the attached NSU.
6. The NSU relays the responses through the NPS network to the NMU from which the requests originated. From here, the responses are communicated to the AXI master.

## Read Transactions

Figure 6: Read Transactions



X22051-112918

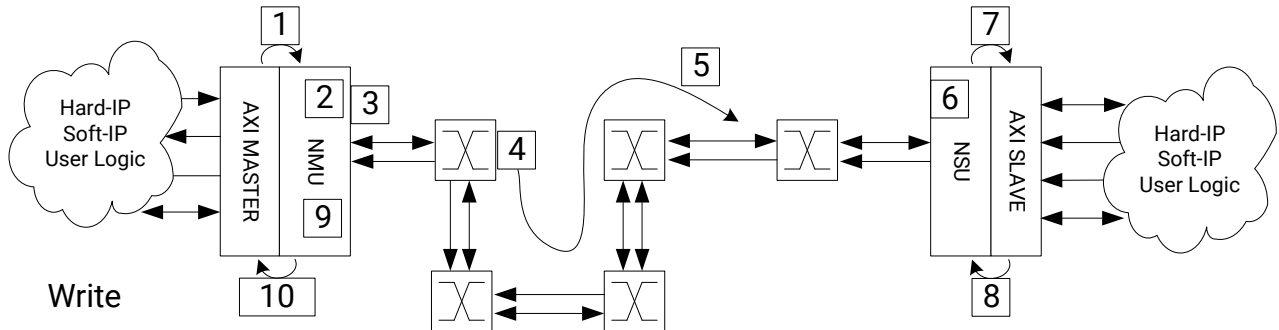
A summary of NoC behavior during a typical read transaction is as follows:

1. The AXI master sends a read request to the NMU.
2. The NMU performs the following functions:
  - Asynchronous crossing and rate-matching from the AXI master clock domain to the NoC clock domain.
  - Destination lookup of the target NSU.
  - Address remapping (in cases of virtualization).
  - AXI conversion of the read request (AxAddr, AxSize, AxLen, AxBurst, AxCache) from the AxSizeMaster to the AxSizeNoC.
  - Read chopping.
  - Read tagging and read-reorder buffer entry insertion to keep track of out-of-order read data returns.
  - Packetizes the read request into the NoC Packet Protocol (NPP) format, performs rate limiting, and ECC generation.
  - VC-mapping, VC-arbitration, and Data Bus Inversion (DBI) generation before sending the packet to the NoC channel.
3. The NMU forwards the NPP to an NPS in the system.
4. The NPS performs the following functions:
  - Destination table lookup for the target out port.
  - Least Recently Used (LRU) arbitration at the output port.
5. The NPP packets are passed through a single NPS or multiple NPSs before they reach the destination NSU.

6. At the destination, the NSU performs the following functions:
  - De-packetizes the read request and performs ECC checking and correction.
  - AXI-ID compression and AXI exclusive access monitoring.
  - Read chopping for downsizing or AXI4 to AXI3 conversion.
    - Read tracker entry insertion to keep track of read data interleaving from the NoC slave.
  - AXI conversion of the request from the `AxSizeNoC` to the `AxSizeSlave`.
  - Asynchronous crossing and rate-matching from the NoC clock domain to the AXI slave clock domain.
  - Sending the AXI format read request to the NoC slave AXI.
7. The read request is processed by the slave AXI, which returns the response to the NSU.
8. The NSU performs the following functions:
  - Asynchronous crossing and rate-matching from the AXI slave clock domain to the NoC clock domain.
    - AXI conversion of the read response from the `AxSizeSlave` to the `AxSizeNoC`.
    - Re-assembly of the read data in the read tracker to match the `AxSizeNoC`.
  - Packetizing of the read response into the NPP packet format and ECC generation.
  - VC-mapping and VC-arbitration before sending the packet to the NoC channel.
9. The NPP formatted read response packets (probably) pass through multiple NPSs before reaching the NMU.
10. When the read response packets reach the NMU, it performs the following functions:
  - Data Bus Inversion (DBI) and ECC checking. ECC correction and de-packetizing of the read response.
    - Re-assembly and reordering of the read data into the request order and `AxSizeMaster` boundary.
  - AXI conversion of the read response data from the `AxSizeNoC` to the `AxSizeMaster`.
  - Asynchronous crossing and rate-matching from the NoC clock domain to the PL master clock domain.
  - Returns the AXI read response to the NoC master AXI.

## Write Transactions

Figure 7: Write Transactions



X22052-112918

A summary of component behavior during a typical write transaction is as follows:

1. The AXI master sends a write request to the NMU.
2. The NMU performs the following functions:
  - Asynchronous crossing and rate-matching from the PL master clock domain to the NoC clock domain.
  - Destination lookup of the target NSU.
  - Address remapping (in cases of virtualization).
  - AXI conversion of the write request (`AxAddr`, `AxSize`, `AxLen`, `AxBurst`, `AxCache`, `writestrobe`, and `writedata`) from the `AxSizeMaster` to the `AxSizeNoC`.
  - Write chopping.
    - Single-slave-per-id (SSID) check for outstanding write transactions with the same AXI-ID but with a different DST (NoC destination ID).
    - Write tracker entry insertion.
  - Packetizing of the write request into the NPP packet format, rate limiting, and ECC generation.
  - VC-mapping, VC-arbitration, and DBI generation before sending the packet to the NoC channel.
3. The NMU forwards the NPP to an NPS in the system.
4. The NPS performs the same steps as for a read operation.
  - Destination table lookup for the target out port.
  - Least Recently Used (LRU) arbitration at the output port (24:1).
5. The NPP write packets are (possibly) passed through multiple NPSs before they reach the destination NSU.



6. At the destination, the NSU performs the following functions:
  - De-packetizing of the write request and ECC checking and correction.
  - Write chopping for downsizing or AXI4 to AXI3 conversion.
    - Write tracker entry insertion.
  - AXI conversion of the request from the `AxSizeNoC` to the `AxSizeSlave`.
  - Asynchronous crossing and rate-matching from the NoC clock domain to the PL slave clock domain.
  - Sending the AXI format write request to the NoC AXI slave.
7. The write request is processed by the slave AXI, which returns the response to the NSU.
8. The NSU performs the following functions:
  - Asynchronous crossing and rate-matching from the PL slave clock domain to the NoC clock domain.
    - Merges the write responses in write tracker (in cases of write chopping).
  - Packetizes the write response into NPP packets and generates ECC.
  - VC-mapping, VC-arbitration before sending the packet to the NoC channel.
9. The NPP formatted write response packets pass through a single NPS or multiple NPSs before reaching the NMU.
10. When the read response packets reach the NMU, it performs the following functions:
  - DBI and ECC checking. ECC correction and de-packetizing of the write response.
    - Merges the write responses (in cases where write chopping is performed during write requests).
  - Asynchronous crossing and rate-matching from the NoC clock domain to the PL master clock domain.
  - Returns the write response back to the NoC AXI master.

## Quality of Service

Every connection through the NoC has an associated quality of service (QoS) requirement. You can set the QoS requirement for each connection through the NoC. The set of desired NoC connections together with their corresponding QoS requirements constitute a traffic specification. The traffic specification is used internally by the NoC compiler to compute a configuration for the NoC.

A QoS requirement has two components:

- Traffic class defines how traffic on the connection is prioritized in the NoC compiler and in the hardware. The traffic class is set at the NMU and is for all paths starting from that NMU.
- Read and write bandwidth requirements describe how much data bandwidth the connection is expected to consume in each direction. Bandwidth requirements are associated with the egress port (NoC slave); each connection might have different read and write bandwidth requirements. Bandwidth can be expressed in units of MB/s or Gb/s.

**Note:** Bandwidth scales in multiples of 1,000, so Gb/s = 8 \* MB/s / 1,000.

The supported traffic classes in priority order are:

- **Low Latency:** The NoC compiler minimizes the structural latency while satisfying the bandwidth requirement. Low latency traffic receives a high priority at all switch and memory controller arbitration points. The low latency traffic class only applies to read traffic.
- **Isochronous:** Includes a mechanism to guarantee the maximum latency of DDR memory traffic. Isochronous traffic is treated as high priority (low latency) traffic through the NoC. Once the request is delivered to the DDRMC queue a timeout counter is started. If the timeout is reached the request is moved to the front of the queue.
- **Best Effort:** The NoC compiler works to satisfy the bandwidth and latency requirements after low latency and isochronous path requirements have been satisfied. Latency values are associated with the egress port (NoC slave). Best effort transactions receive the lowest arbitration priority throughout the NoC.

**Note:** In the current release you cannot specify the isochronous timeout latency.

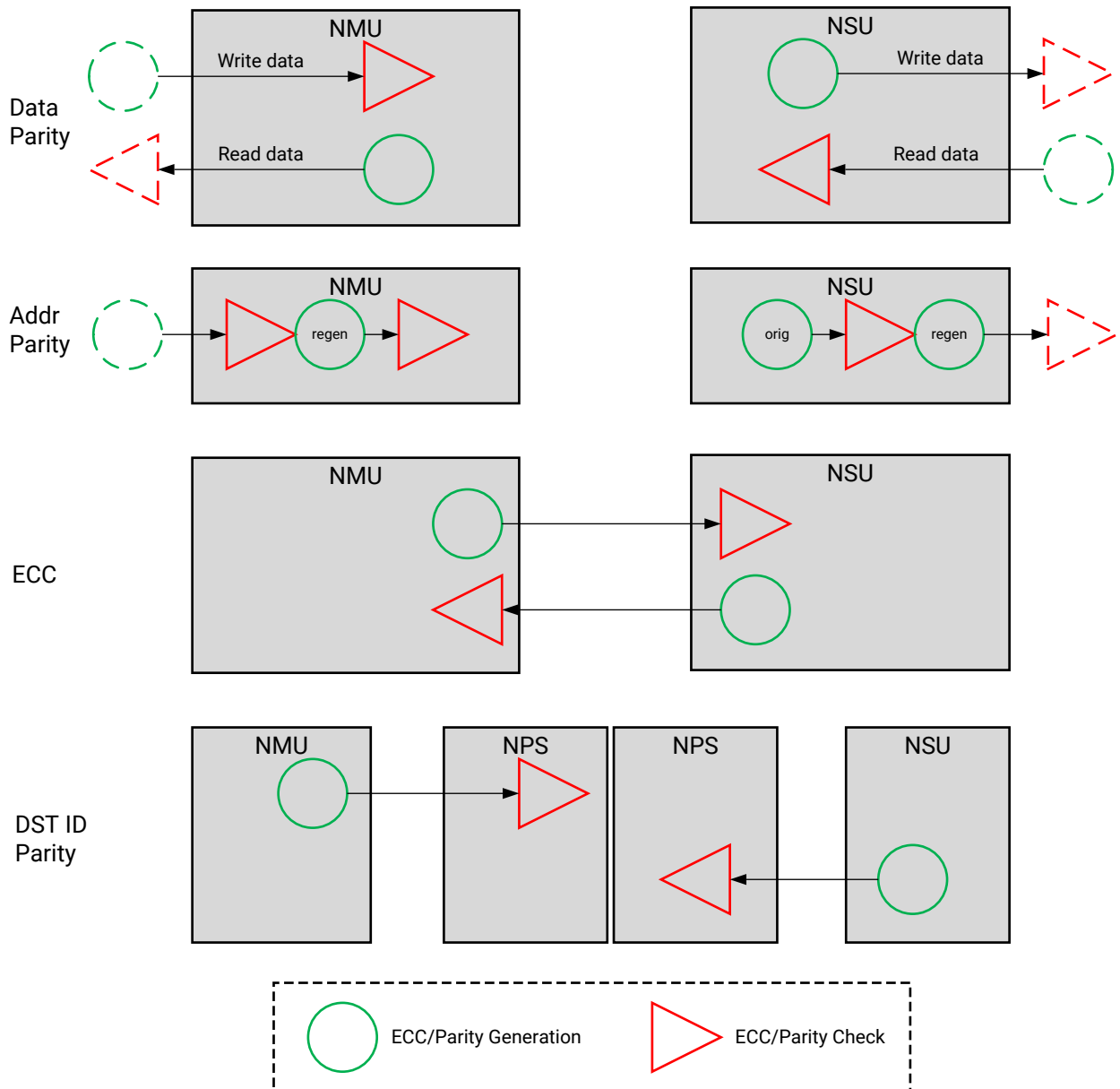
## Data Integrity

The Versal ACAP Programmable NoC supports end-to-end data protection for AXI memory mapped transactions. In the AXI protocol domain of the NMU and NSU the NoC supports 1 bit per byte of even parity on data lines and 1 bit of even parity on the address. Parity checking in the AXI protocol domain is enabled at configuration time. See [Configuring the AXI NoC](#) for more details about how to configure parity checking in the AXI protocol domain. No AXI parity checking is available between the NoC and the PS or AI Engine. Detection of parity errors results in a fatal interrupt.

In the packet domain, after chopping and conversion to the NoC packet format the NoC supports SECDED ECC across the entire flit. The ECC syndrome is completed and checked in the NMU and NSU. No ECC checking is performed in the switch fabric. To detect routing errors an additional parity bit is provided to protect the destination ID field. If a parity error occurs in a switch, the switch signals a fatal error via an interrupt. Packet domain parity and ECC generation and checking is always enabled. For all errors, information is logged about the offending transaction, including the transaction or packet type, the address and the source and destination IDs. Correctable ECC errors are corrected on the fly, and the count of correctable errors is incremented. Uncorrectable ECC errors result in a fatal interrupt.

**Note:** By default, all interrupts are masked.

Figure 8: End-to-End Protection Overview



X25510-070521

## Parity

AXI Parity is generated by the user in the AXI master or slave, and sent to the NMU or NSU at the AXI interface input. The NMU and NSU also generate AXI Parity at the AXI interface output.

Data and AxAddress parity covers the protocol domain (between the AXI interface and the conversion to NoC Packet Protocol (NPP)) for both the request and response directions, in both the NMU and NSU pipelines.

- 1 bit per byte for Data
- 1 bit per byte for AxAddress

Parity is checked in the NMU/NSU pipeline when an AXI field is consumed (used by logic). When an AXI field is modified by NMU/NSU logic, parity is regenerated.

For AXI requests with unaligned addresses, read/write data parity is checked from 16B-aligned addresses. For example, if the starting address of an AXI request is `0x4` and the AXI master/slave provides the wrong parity on byte `0x2`, it is detected as a parity error. Write data parity is checked regardless of the AXI write strobe value.

For the NMU:

- Address parity for read/write requests and data parity for write requests is generated by the user in the AXI master, and sent to the NMU at the AXI interface input.
- Data parity for write requests is delivered along with the data and checked at the conversion from AXI protocol to NPP.
- The AxAddress can be consumed and modified for address mapping/remapping and AXI chopping, so address parity is checked and regenerated in the NMU. Address map/remap regenerates seven parity bits [7:1], with parity bit 1 covering address nibble [15:12]. AXI chopping regenerates 1-bit parity [0] for the lower 12 address bits. Regenerated address parity is checked at the conversion from AXI protocol to NPP.
- Data parity for read responses is generated as 1 bit per byte after the ECC check stage, when the data is converted from NPP to AXI protocol.

For the NSU:

- Address parity for read/write requests and data parity for write requests is generated by the NSU after the ECC check.
- Data parity for write requests is sent to the AXI slave as is, because byte-level data is not modified.
- Address parity for read/write requests is also sent to the AXI slave as is for AXI4 protocol.
- For AXI3 protocol, the lower 8 bits of address can be modified for AXI chopping, so the parity for lower 8-bit address is checked and regenerated in the AXI conversion stage.
- Data parity for read response is generated by the user in the Axi slave, sent to the NSU AXI interface input, and delivered along with data until it is checked at ECC generation stage.

Static fields in the NMU/NSU tracker entries are parity protected and checked when those fields are consumed. Tracker data buffers (NMU Read Reorder Buffer and NSU Read Tracker) are data parity protected.

The NPP packet (DST ID + LAST) field is also protected by 1-bit even parity. DST-ID parity is generated by the NMU/NSU and checked by the NPS. DST-ID parity is always generated regardless of AXI protocol (AXI3/AXI4 request/response, AXI4-Stream).

- The NMU generates DST-ID parity for read/write request NPP packets.
- The NSU generates DST-ID parity for read/write response NPP packets.

## Data Poisoning

The NoC carries data poisoning information in AXI-MM mode.

The WPOISON bit can be set by the AXI master and sent to the AXI slave. An AXI slave receives the WPOISON bit, and it can act accordingly. For example, the DDRMC, upon receiving the WPOISON bit, corrupts the ECC syndrome being written to memory. The WPOISON bit can be set by NoC components at:

- NMU AXI master input (write request).
- NMU, if a write data parity error is detected.

The NSU and NPP carry the poison bit from the NMU to the NSU. The WPOISON bit accumulates among multiple flits for upsizing, and splits into multiple flits for downsizing.

The RPOISON bit can be set by the AXI slave and sent to the AXI master. The AXI master receives the RPOISON bit and it can act accordingly. The RPOISON bit can be set by NoC components at:

- NSU AXI slave input (read response).
- NSU, if a read data parity error is detected.

The NMU and NPP carry the poison bit from the NSU to the NMU. The RPOISON bit accumulates among multiple flits for upsizing, and splits into multiple flits for downsizing.

---

## Standards

This core adheres to the following standard(s):

- *AMBA AXI and ACE Protocol Specification* ([ARM IHI0022E](#))
- *AMBA AXI4-Stream Protocol Specification* ([ARM IHI 0051A](#))

## AXI Conversion

The NMU and NSU support AXI3/AXI4/AXI4-Stream protocols interfacing with the NoC Master and the NoC Slave. Between a NoC master request and a response return, multiple AXI conversions including transformations of the AxAAddr, AxSize, AxLen, and choppings are performed. This section covers the AXI conversion overview, the NoC AXI support/restriction and the NoC Master/Slave configuration assumptions for AXI conversion.

## AXI Support and Restrictions

AXI support and restrictions in the NoC are summarized in the following table.

**Table 5: AXI Support**

	Supported	Not Supported/Restrictions
Interface Width	32-512	1024
AxSize	8-512	1024
AxBurst	INCR: Support modifiable and non-modifiable transactions with restrictions	
	WRAP: Supported with the following restrictions: <ul style="list-style-type: none"> <li>Transaction size can be 32B or 64B for read, 64B for write</li> <li>AxSizeMaster limited to 2 (32-bit), 3 (64-bit), or 4 (128-bit)</li> </ul>	AxSizeMaster not equal to 2, 3, or 4 AxSizeMaster is 4 and InterfaceWidthSlave is less than 4
	FIXED: Supported with the following restrictions: <ul style="list-style-type: none"> <li>AxSizeMaster <math>\leq</math> AxSizeNoC(4)</li> <li>AxSizeMaster <math>\leq</math> InterfaceWidthSlave</li> <li>Transaction is not modified (no upsize or downsize) from the NoC Master to the NoC Slave</li> </ul>	
AxCache[1]	Supports INCR, WRAP, and FIXED with the restrictions stated in AxBurst	Non-modifiable transfer with AxSizeSlave > 128 is not supported
AxLen	AXI4 - 256 AXI3 - 16	

Table 5: AXI Support (cont'd)

	Supported	Not Supported/Restrictions
Exclusive Access	<p>Exclusive Access is limited to AxBurst == INCR</p> <p>For DDR, AxLenMaster = 1-flit and AxSizeMaster &lt;= 4 (128-bit), NMU would send the AxSizeMaster instead of the AxSize of 4 (128-bit) for modifiable and non-modifiable.</p> <p>For DDR, AxLenMaster &gt; 1-flit, NMU would send upsized (packed) AxSize of 4 (128-bit) for modifiable and non-modifiable. Read over-fetch is expected.</p> <p>For non-MC slave, AXI conversion always sends traffic in non-modifiable INCR for both modifiable and non-modifiable. The AxCache[1] bit is not modified by the NoC.</p> <p>For AXI3/4, with Exclusive Access size of 32B/64B/128B, the corresponding AxSizeSlave cannot be less than 2B/4B/8B respectively (i.e., downsize chopping in NSU).</p>	
AXI4-Stream	<p>32-bit and 64-bit interface widths are unsupported in the NMU and NSU.</p> <p>Support and behave as modifiable Write with Interface WidthMaster == AxSizeMaster; Interface WidthSlave == AxSizeSlave;</p> <p>Support 6-bit TID</p> <p>Support 10-bit TDEST</p>	<p>TKEEP is supported.</p> <p>TSTRB is not supported.</p> <p>NMU can switch TID (interleave on TID) or TDEST (interleave on TDEST) only if:</p> <ol style="list-style-type: none"> <li>1. The current TID/TDEST has finished sending a multiple of 64B data</li> <li>2. The transaction is ending with TLAST = 1</li> </ol>
Other features		<p>No support for AXI3 write interleaving</p> <p>No support for AXI3 Locked Access</p> <p>No support for RUSER for Read Response</p> <p>No support for TSTRB for AXI Stream</p>

## AXI Conversion Overview

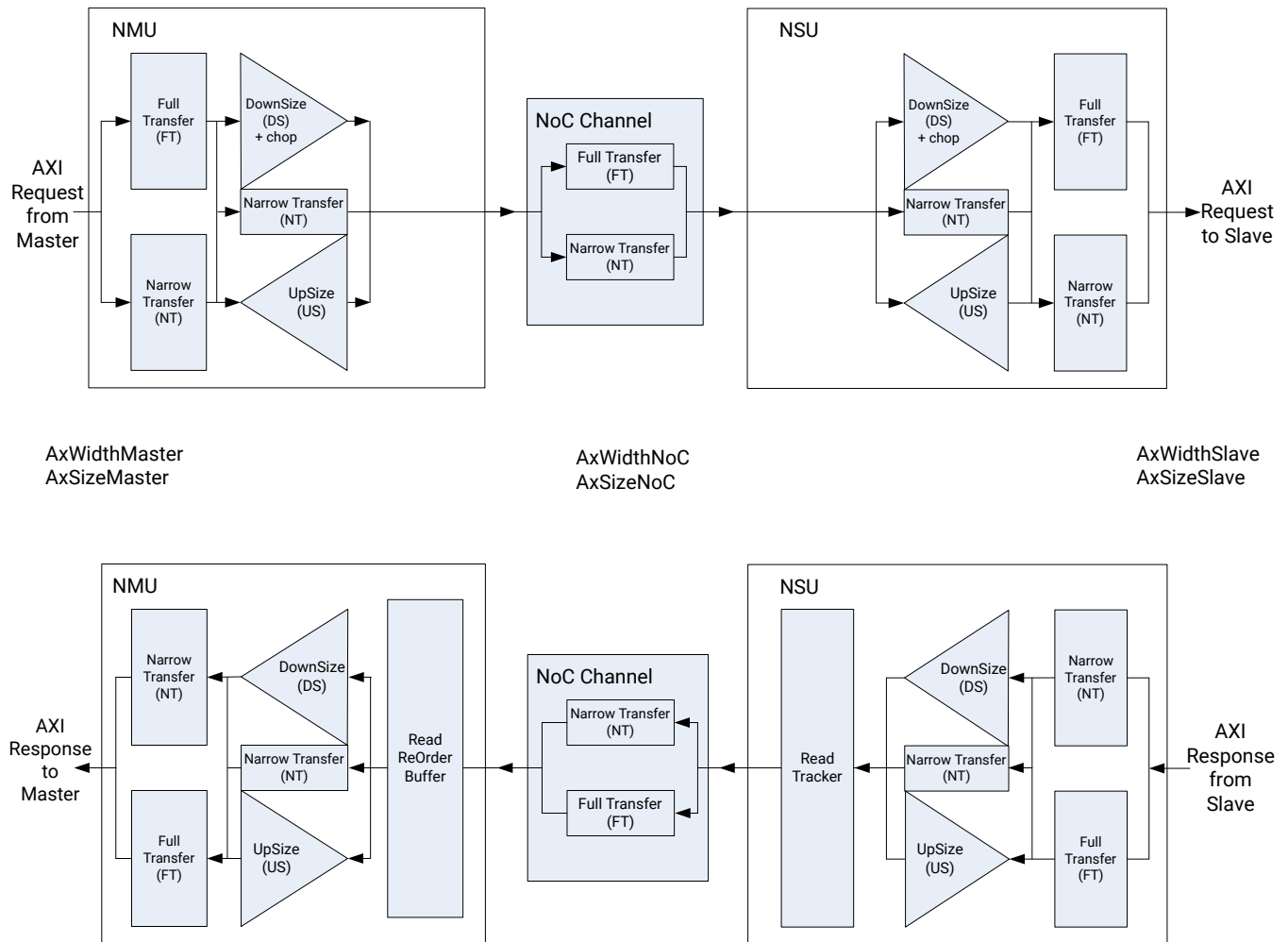
For a request or response transaction, AXI information including AxSize, AxLen, AxBurst, AxCache[1] (modifiable/non-modifiable transaction), AxLock (Normal/Exclusive access), and AxAddr, are carried along the transaction to/from the NoC Master, the NMU, the NoC channel, the NSU and the NoC Slave. The NoC header always carries consistent AXI information that describes the Read/Write request it is carrying. Because the NoC channel has a 128-bit interface width, an AXI request is always converted (downsized or upsized) from the AxSize of Master to match the AxSize of the NoC (128-bit) for a modifiable transaction. For a non-modifiable transaction, the AXI request could go through narrow-transfer or forced-downsize.

A request could be chopped by either a programmable chop (REG\_CHOPSIZE) by the NMU or in the case of AXI downsize in the NSU, a chop may be required to keep the AxLen less than 16 for AXI3 and 256 for AXI4.



The following figure shows the logic view of the NMU and the NSU for AXI conversion. There are upsize and downsize blocks in both request and response directions. In addition, the NMU/NSU supports full transfer and narrow-transfer requests/responses. The NSU performs narrow transfer or full transfer based on `AxCACHE[1]` (modifiable/non-modifiable), and the `AxSize` of the master, the NoC, and the slave.

Figure 9: AXI Conversion Overview



X25462-062121

## AxSize/AxLen Conversion and Chopping

The NMU and NSU perform different `AxSize/AxLen` conversions based on `AxCACHE[1]` (modifiable/non-modifiable), `AxBurst` (INCR, WRAP, FIXED) and `AxLock` (exclusive access).

This section first covers NMU and NSU AXI conversion for `AxBurst` of INCR to provide some background of conversion. Then it covers the specifics of `AxBurst` of WRAP, `AxBurst` of FIXED, Exclusive Access and AXI4-STREAM.

For modifiable transactions with AxBurst of INCR, the NMU performs the following conversions on a read request or write request:

- Convert the AxBurst of the master to the AxBurst of the NoC (128-bit).
- For DDR transactions, chopping is done at minimum size between the DDR-interleave-granularity (if DDR-interleave is enabled) and the default chop size of 256 bytes.
- For non-DDR transactions, chopping is done at the default chop size, 256 bytes.
- For exclusive access, the NMU does not perform chopping. The NMU sends modifiable exclusive access in non-modifiable format in NPP. AxCache[1] is not modified by the NMU.

For modifiable transactions with AxBurst of INCR, the NSU performs the following conversions on a read request or write request:

- Convert the AxBurst of the NoC 4 (128-bit) to the AxBurst of the Slave. The AxBurst of the slave is assumed to be the same as the Interface Width of the slave.
- Chopping is done at the address boundary of (AxBurstSlave\*256) for AXI4 and address boundary of (AxBurstSlave\*16) for AXI3, regardless of AxBurstLen.
- For exclusive access, the NSU does not perform chopping. The NSU sends modifiable exclusive access in non-modifiable format in NPP. AxCache[1] is not modified by the NSU.

For non-modifiable transactions with AxBurst of INCR, the NMU performs the following conversions on a read request or a write request:

- If the AxBurst of the master is larger than the AxBurst of the NoC (128-bit), downsize and signal an interrupt.
- If the AxBurst of the master is less than AxBurst of the NoC (128-bit), do a narrow transfer.
- For DDR transactions, if the AxBurst of the master is less than the AxBurst of the NoC (128-bit), the NMU would force-upsize to 4 (128-bit) and send an interrupt to the CIPS, because the DDR NSU has a fixed AxBurst of 4 (128-bit) and does not perform AXI conversion.
- For exclusive access, the NMU does not perform chopping.

For non-modifiable transactions with AxBurst of INCR, the NSU performs the following conversions on a read request or a write request:

- If the AxBurst of the NoC (narrow-transferred from master) is larger than the Interface Width of the slave, downsize and signal an interrupt.
- If the AxBurst of the NoC (narrow-transferred from master) is smaller than the Interface Width of the slave, narrow-transfer the AxBurst of the NoC(narrow-transferred from master). It is assumed that the NoC slave supports any AxBurst less than or equal to its Interface Width.
- Chopping is done at the address boundary of (AxBurstSlave\*256) for AXI4 and address boundary of (AxBurstSlave\*16) for AXI3, regardless of AxBurstLen.

For modifiable or non-modifiable write response, both the NMU and the NSU do not perform AXI conversion. Write responses from chopped NMU or NSU transactions are merged by the write response trackers in the NMU and the NSU.

For modifiable transactions with AxBurst of INCR, the NSU performs the following conversions on a read response:

- If the AxBurst of the slave is less than 4 (128-bit), the read tracker has a 128-bit read data buffer and it collects 128 bits of read data before returning read data to the NoC.
- If the AxBurst of the slave is larger than 4 (128-bit), downsize to the AxBurst of the NoC (128-bit).
- The slave is back-pressured if the 128-bit read data buffer is full.

For modifiable transactions with AxBurst of INCR, the NMU performs the following conversions on a read response:

- The NMU read re-order buffer (RROB) saves 128-bit read data into the read data buffer.
- The RROB presents read return data to the AXI conversion block in the format of the AxBurst of the master.
- If the width of the master is 256 or 512, the RROB is responsible for providing contiguous read data cycles to fill the width of 256 or 512. (i.e., no read interleaving should occur).

For non-modifiable transactions with AxBurst of INCR, the NSU performs the following conversions on a read response:

- If the AxBurst of the slave is less than 4 (128-bit), provide narrow-transfer read data to the read tracker. The read tracker will not upsize the read data to 128-bit format. Rather, it will return the narrow-transfer read data to the NoC.

For non-modifiable transactions with AxBurst of INCR, the NMU performs the following conversions on a read response:

- The NMU RROB saves narrow-transferred read data and performs 128-bit upsizing in the read data buffer (RDB).
- The NMU AXI conversion block behaves the same as for a modifiable transaction.

### For WRAP Transactions

- WRAP read of 32B and 64B are supported. WRAP write of 64B is supported.
- The AxBurst of the master must be equal to 2, 3, or 4 (32-, 64-, or 128-bit).
- For AxBurstMaster of 4 (128-bit) read/write requests, AxBurst of WRAP is sent as is. For WRAP read, the NMU RROB data return path treats return data as INCR.

- For any other case (AxBurstMaster of 2 or 3), the NMU translates WRAP operation into a single AxBurst of INCR transaction at the wrapped size boundary.
  - For reads, the NMU RROB waits until complete data has been returned to the RROB data buffer before returning data to the master in WRAP order.
  - For writes, the NMU waits until complete write data is available from the master, and then it sends the data to the NSU in INCR order.
- The AXI conversion blocks in the NSU perform AxBurst of WRAP for AxBurstMaster of 4.
- The interface width of the slave must be wider than or equal to 4 (128-bit) to support AxBurst of WRAP in the NSU.
- The AXI conversion blocks in the NMU convert to AxBurst of INCR for AxBurstMaster of 2 or 3.
- DDR only receives AxBurst of WRAP for AxBurstMaster of 4.

### For FIXED Transactions

- The NMU and NSU follow the non-modifiable flow to narrow-transfer all requests and responses.
- The AxBurst of the master must be less than or equal to 4 (128-bit).
- The AxBurst of the master must be less than or equal to the Interface Width of the slave.
- The AXI conversion block uses the same AxBurst for chopping and narrow-transfer.
- AxBurst is limited to 15 by the AXI Spec. Maximum transaction size is 16B\*16 = 256B. FIXED transactions are chopped at a default chopsize of 256 bytes. Chop is based on AxBurst alignment to the chopsize.

### For Exclusive Access

- Only INCR AxBurst is supported.
- No chopping is performed by the AXI conversion block.
- For DDR, AxBurstMaster = 0(1-flit) and AxBurstMaster <= 4(128-bit), the NMU sends the AxBurstMaster instead of the AxBurst of 4 (128-bit) in the AxBurstNoC. (This follows the non-modifiable non-DDR INCR packet conversion rule).
- For DDR, AxBurstMaster > 0(1-flit), the NMU converts the packet to AxBurst of 4 (128-bit) for modifiable and non-modifiable. Read over-fetch is expected. (This follows the modifiable non-DDR INCR packet conversion rule).
- For non-DDR slaves, the AXI conversion is same as for normal INCR non-modifiable.
- For AXI3/4, with Exclusive Access size of 32B/64B/128B, the corresponding AxBurstSlave cannot be less than 2B/4B/8B respectively (to avoid downsize chopping in the NSU).

### For AXI4-Stream

- The NMU and NSU follow the modifiable write flow to fully pack the 128-bit NoC channel.
- The NMU handles AXI4-Stream as AXI-MM-modifiable-write with:

**Note:** This bullet is for illustration only. AxBSize is not an AXI4-Stream parameter.

- AxBSize of the master is set to be the same as the Interface Width of the master.
- AxBSize of slave is set to be the same as the InterfaceWidth of the slave.
- For partially filled data flits (the first or last flit could be partially filled after conversion), TKEEP is used to indicate NULL data.

## AXI Conversion Examples

This section describes examples of AXI conversion. In the examples, the green boxes represent unused data bytes and yellow boxes represent data bytes of interest.

In AXI conversion, because there could be upsizing from the AxBSize of the master to the AxBSize of the NoC, or upsizing from the AxBSize of the NoC to the AxBSize of the slave, read data over-fetch might occur in the NoC slave. It is assumed that read data over-fetch would not cause any side effect in the NoC slave. For write data, the NMU and NSU are responsible for creating correct write strobes.

In example 1, the NMU receives a request with AxBurst = WRAP and AxBSize = 4 (128-bit). The request is transferred as is from the NMU through the NoC Channel to the NSU.

### Example 1

WRAP (Behaves as non-modifiable INCR without downsize capability)

Applies to NoC Master / NoC Channel / NoC Slave

**Table 6: Interface Width = 128-bit, AxBSize = 4 (128-bit), AxBLen = 4 (4), AxBBurst = WRAP, AxBAddr = 0x10**

Start Addr \ Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x10																
0x20																
0x30																
0x00																

## Example 2

In Example 2, the NMU receives a request with  $AxBurst = WRAP$  and  $AxSize = 3$  (64-bit). The request is converted to a modifiable/non-modifiable INCR burst in the NMU and is sent through the NoC channel to the NSU.

WRAP (converted to Modifiable/Non-modifiable INCR)

NoC Master

**Table 7: Interface Width = 128-bit, AxSize = 3 (64-bit), AxLen = 7 (8), AxBurst = WRAP, AxAddr = 0x18**

Start Addr \ Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x18																
0x20																
0x28																
0x30																
0x38																
0x00																
0x08																
0x10																

NoC Channel, no chop needed

**Table 8: Interface Width = 128-bit, AxSize = 4 (128-bit), AxLen = 3 (4), AxBurst = INCR, AxAddr = 0x00**

Start Addr \ Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00																
0x10																
0x20																
0x30																

NoC Slave

**Table 9: Interface Width = 64-bit, AxSize = 3 (64-bit), AxLen = 7 (8), AxBurst = INCR, AxAddr = 0x00**

Start Addr \ Byte	7	6	5	4	3	2	1	0
0x00								
0x08								
0x10								
0x18								
0x20								
0x28								
0x30								
0x38								

## AXI Configuration Assumptions

Each NMU and NSU has a configurable data width to allow you to define the interface width of the NMU or NSU.

For the NMU AXI-MM interface, the AxSize of the master is defined by the AxSize input of the transaction.

For the the NSU AXI-MM interface, the AxSize of the slave is implied with the following assumptions:

- For modifiable transactions, the AxSize of the slave must be equal to the interface width of the slave. This defines the NSU packing width in modifiable transactions.
- For non-modifiable transactions, the slave must support the AxSize of the master or smaller. This is because the NSU would either narrow transfer or downsize the transfer from the AxSize of the Master to the Slave interface.
- AxSize of the DDRMC is fixed at 4 (128-bit). The NMU always upsizes or downsizes into the AxSize of 4 (128-bit) for DDR transactions.

For AXI4-Stream:

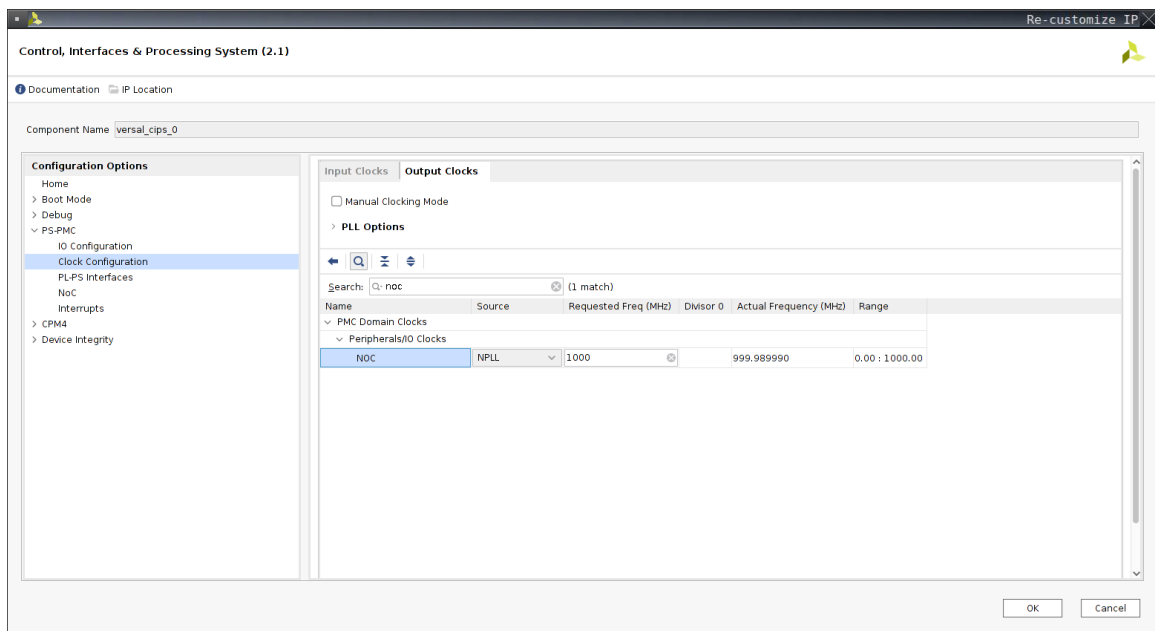
- Only PL and AI Engine interfaces use AXI-S.
- TDEST signal support for soft interconnect.

# Clocking

The NoC is clocked by a single clock for the entire chip. The NMUs/NSUs have asynchronous integrated FIFOs to ensure transition from the AXI clock domain of an individual master or slave to the NoC clock domain.

The NoC clock is controlled by a PLL inside the CIPS Control, Interface and Processing System (CIPS) IP. To Change the NoC Clock Frequency the CIPS IP must be opened and used. This can be done using a Tcl command or via the GUI as shown in the following figure.

Figure 10: CIPS: Output Clocks



For more details about the NPLL refer to the Clocks Chapter in the *Versal ACAP Technical Reference Manual* ([AM011](#)).



**IMPORTANT!** When using the NoC with the included DDRMC, the NoC clock frequency must be set greater than or equal to 800 MHz.

Table 10: Clocks

Clock	Description
aclkn	The AXI NoC and AXIS NoC IP can be configured to have up to $N$ independent AXI clocks, where $N$ is the sum of the number of AXI interfaces on the IP core. Each AXI interface of the NoC is associated with a clock by selecting the clock signal on the Inputs and Outputs tabs of the configuration screen.



Table 10: Clocks (cont'd)

Clock	Description
sys_clk	<p>If an instance of axi_noc is configured to include an integrated DDRMC, one <code>sys_clk</code> port appears on the boundary of the IP for each DDRMC. This port must be connected to a differential clock source whose frequency is user-selectable on the DDR Basic tab of the AXI NoC configuration dialog. The <code>sys_clk</code> differential clock source must be placed on a global clock (GC) input pin in any of the three XPIO banks used by a DDRMC. This clock is used internally by the DDRMC to generate various clocks for the controller and external DDR memory.</p> <p>For information on <code>sys_clk</code> input termination and I/O standards see AC Coupling Recommendations in the <i>Versal ACAP SelectIO Resources Architecture Manual</i> (AM010). The clock generator driving <code>sys_clk</code> should have jitter of less than 3 ps RMS.</p>

## Resets

The NoC and the integrated DDR memory controller (DDRMC) can only be reset from the Platform Management Controller (PMC). There are no user routable resets in the NoC or in the AXI interfaces to the NoC.

# Integrated Memory Controller (DDRMC) Architecture

---

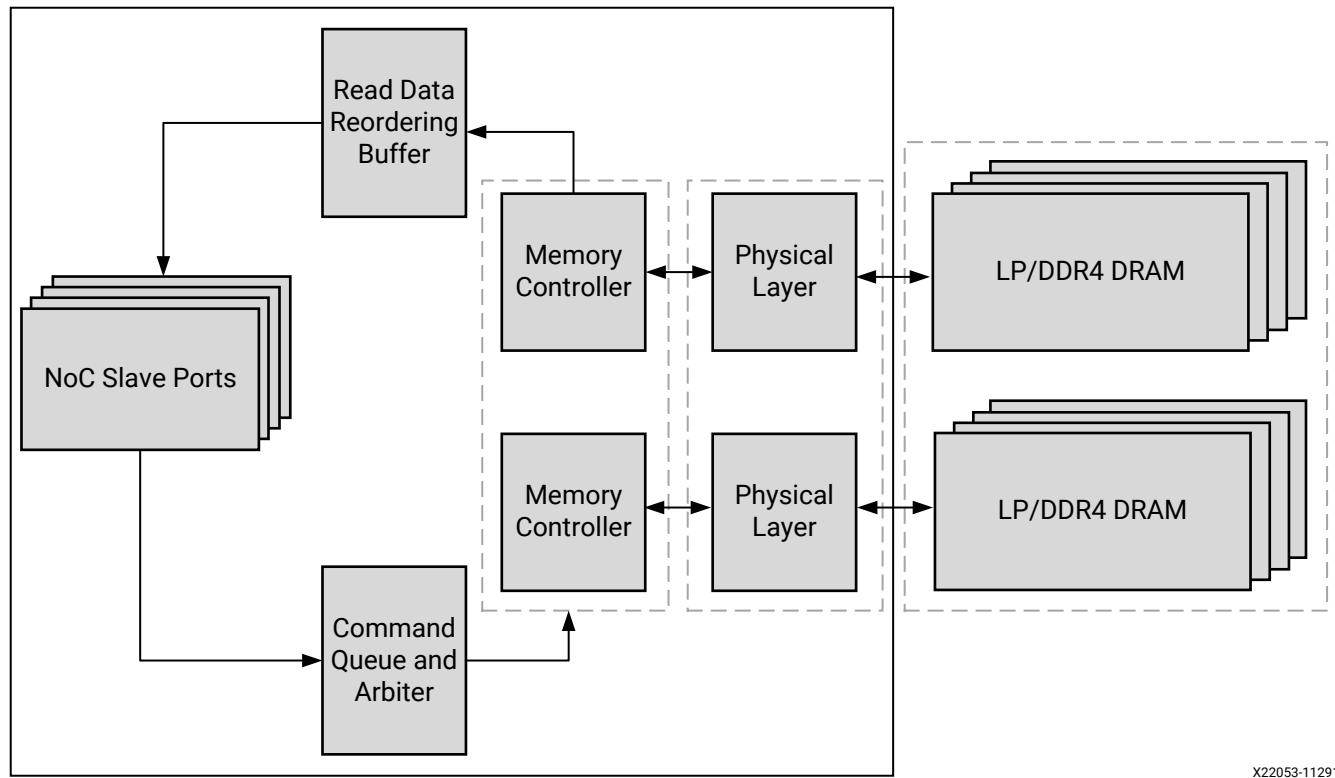
## Memory Controller

This section describes the Versal<sup>®</sup> architecture-based Memory Controller core with an overview of the modules and interfaces.

### Core Architecture

There are four NoC Slave ports to access the Memory Controller. The slave ports sort the commands to the appropriate command queue based on their QoS class. An arbiter then pulls commands from the queues based on priority and bandwidth requirements and pushes them to the Memory Controller. The Memory Controller reorders the commands to optimize for efficiency and then sends them to the Physical Layer which handles the DRAM interface timing and sequencing. The controller may be configured to act as two separate memory channels of up to 32-bit data width per channel.

Figure 11: Memory Controller Block Diagram



X22053-112918

## Reordering

The Memory Controller utilizes a state machine to determine reordering priority. Depending on the state of the transactions, it will either optimize for efficiency (reordering to take advantage of open pages) or it may service transactions which have been left idle for too long.

There are four states: Read Priority, Write Priority, Write/Read, and Starved.

- **Read Priority:** Read transactions are given priority by allowing Read page miss transactions to execute a precharge even when there is a pending write to the same bank. Write transactions cannot execute a precharge if there is a pending read page hit. This is the default state.
- **Write Priority:** The number of write commands have exceeded a threshold and so writes will take priority until the pressure is lowered.
- **Write/Read:** Efficient transactions are the priority in this state. Reordering commands for page hits are prioritized.
- **Starved:** One or more read transactions are starved and only starved Reads and any Writes coincident with starved Reads are selected.

There is a single read reordering buffer in each memory controller. When a memory controller is configured in dual-channel mode, half of the buffer is assigned to each channel. This reduced buffer depth will result in efficiency loss for read operations because of the reduced reordering flexibility. The amount is dependent on the address pattern of the reads.

## **ECC**

Error Correction (Single Error Correct, Double Error Detect) performs checks of read data without interrupting traffic. The ECC calculations are stored in an additional byte, regardless of the data width. For example, a 64-bit DRAM interface with ECC enabled needs an additional byte, making the entire interface 72 bits. Similarly, a 32-bit interface with ECC enabled also needs an additional byte, making the entire interface 40 bits.

For each burst single-bit errors are correctable. Double-bit errors are detectable but uncorrectable. Three or more bit errors may or may not be detected and are not correctable. If any errors are detected, correctable or uncorrectable, they are logged and can be configured to generate an interrupt. Correctable and uncorrectable errors can be injected on write operations.

**Note:** ECC adds two memory clock cycles of latency during writes when the CAS Write Latency is less than 11.

## **ECC Scrubbing**

On-the-fly scrubbing occurs when a correctable ECC error is detected on a read transaction. A Read-Modify-Write (RMW) operation is executed at the same memory address. A RMW is used in the event that a write had occurred after the correctable error was detected but before the controller had returned to complete the scrubbing. If an uncorrectable error is detected, on-the-fly scrubbing will not be performed. However if both correctable and uncorrectable errors are detected in a single burst, scrubbing will be performed.

Background scrubbing is the process of stepping through the DRAM doing RMW to each address to mitigate data loss via single event upset. The memory controller will utilize idle cycles to implement the scrubbing, and in the event of full traffic will periodically insert transactions to ensure progress is made. For DDR4, the background scrubbing period can be set by the user via the GUI. For LPDDR4/4X, the background scrubbing period is not a GUI option and is set to a fixed value of 20  $\mu$ s.

The memory can be initialized with the proper ECC values at the end of memory calibration. The amount of memory to be initialized is configurable, and the memory controller will not execute any user commands until this process is completed. Initialization is not required, however the user must ensure that no reads are issued to an address that hasn't been written to first.

## Address Parity

The Versal Memory Controller supports Address and Command Parity for DDR4. If a parity error is detected by the DRAM, `ALERT_n` will be driven low and detected by the memory controller and the controller will halt all traffic and attempt a re-try of the failed command. This requires significant effort by the controller and physical interface (PHY) to recover and re-try. The PHY buffers need to be flushed, which may have a latency impact of several microseconds. As a result, isochronous latency is not guaranteed, and bandwidth requirements may not be met in this scenario. If an `ALERT_n` is detected during the retry attempt, the memory controller will issue a fatal interrupt. The DDRMC cannot recover from an `ALERT_n` event during retry. A system reboot is needed and memory contents is lost.

## Address Parity Error Injection

A parity error can be injected on the DRAM Command/Address bus using the `add_par_err_inj` register. This register contains fields to inject a parity error on each DDRMC sub-channel. The injector can be set up to pick a Read CAS command, a Write CAS command, or a random command. The register logs the command the error was injected on and sets a “done” bit. After an error inject event occurs, the “done” bit can be cleared and the error injector can be re-enabled to inject another error.

1. Select the type of command for error injection by writing to `DDRMC_MAIN_n.add_par_err_inj[cmd_type{0,1}]`, where `n` represents the DDRMC number, and the 0 or 1 after `cmd_type` represents the DDRMC sub-channel number. By default, errors are injected on any DRAM command. Setting this register to 1 limits error injection to DRAM Read CAS commands, and setting it to 2 limits error injection to DRAM Write CAS commands.
2. Enable error injection by writing `1'b1` to `DDRMC_MAIN_n.add_par_err_inj[cmd_en{0,1}]`.
3. Access memory with the selected command type.
4. Read `DDRMC_MAIN_n.add_par_err_inj[cmd_done{0,1}]`. A value of `1'b1` indicates that error injection is done. The `dram_rank{0,1}` and `dram_cmd{0,1}` fields can be read to see which command and rank had an error injected.
5. Read `DDRMC_MAIN_n.DDRMC_ISR[DRAM_PARITYm]` to look for the DRAM Parity Error status. `n` = the memory controller number, and `m` = the channel number within a given memory controller.

## Periodic Reads

The Versal Memory Controller will periodically issue a read instruction to the DRAM to ensure that read clocks are returned for DQS gate tracking purposes. During calibration the DQS are aligned with an internal clock, and this alignment is adjusted during normal operation. In order for this adjustment to occur, a read DQS is required. An internal counter keeps track of the time elapsed since a read DQS has arrived, and if one has not arrived in the last 20  $\mu$ s, the controller will insert a 'dummy read' solely for the purpose of receiving the DQS for tracking. The data returned from the DRAM is discarded at the PHY and is not returned to the memory controller.

Memory access patterns with long continuous writes will have a small efficiency impact as a read operation will create a read-write turnaround delay. If the memory access contains a mix of read and write commands, provided two reads are within 20  $\mu$ s of each other, the controller will not need to insert any periodic reads.

## Refresh

The Versal Memory Controller issues refreshes to the DRAM automatically. The controller will issue refreshes as needed to maintain data integrity of the DRAM.

**Note:** The Versal Memory Controller does not automatically adjust refresh rate in response to temperature changes. When configuring the AXI NoC parameters on the DDR Memory tab, you must set  $t_{REFI}$  (ps) according to the DRAM refresh interval requirement at the maximum intended operating temperature. Consult the DRAM vendor datasheet for tREFI requirements.

For LPDDR4/4X, the Versal Memory Controller supports all-bank or per-bank refreshes; however, per-bank refresh is only supported for single-rank systems running at 1250 Mb/s or faster. Dual-rank systems do not support per-bank refresh.

For DDR4, fine granularity refresh modes of 1x, 2x, and 4x are supported in the following configurations:

- **1x:** All configurations
- **2x:** Configurations with equal or less than eight logical ranks (ranks \* stacks)
- **4x:** Configurations with equal or less than four logical ranks (ranks \* stacks)

On-the-fly refresh for DDR4 is not supported.

During normal DDR4 operation and when operating LPDDR4/4x in all-bank refresh mode the Versal DDRMC will block accesses to the rank which requires the refresh. Next it will issue a Precharge All command to the rank and then it will issue the Refresh command. The Versal DDRMC does not support Auto Precharge for Write/Read commands to opportunistically Precharge Banks before a Refresh command.

## User Refresh and ZQ Periodic Calibration

The MC has two main refresh modes, internal and user, the default being internal. There is no support for switching between these two modes on the fly. User refresh mode can be enabled from the GUI while configuring the memory IP. Refer to the DDR Advanced Tab in [Configuring the Memory Controller](#). This will create additional ports that can be controlled through PL.

**Table 11: User Refresh Port List**

Port Name	IO
ref_req_*	input
ref_rank_en_*	input
ref_ack_*	output
ref_usr_port_available/cal_done	output

In user mode, the refresh request can be made in rank-wise granularity, a four-bit request vector can be used to implement this from PL. In user refresh mode a PL interface controls when the controller's per-rank "pending refresh counters" are incremented.

Each request/acknowledge pair corresponds to a single physical rank. Each pair operates independently of the other pairs. The "ref\_rank\_en" signals are driven by PL and are asynchronous to the DDRMC clock. The "ref\_ack" signals are driven by the DDRMC and are asynchronous to the PL clock. It is up to the receiving clock domain to synchronize the signals before use to avoid metastability problems.

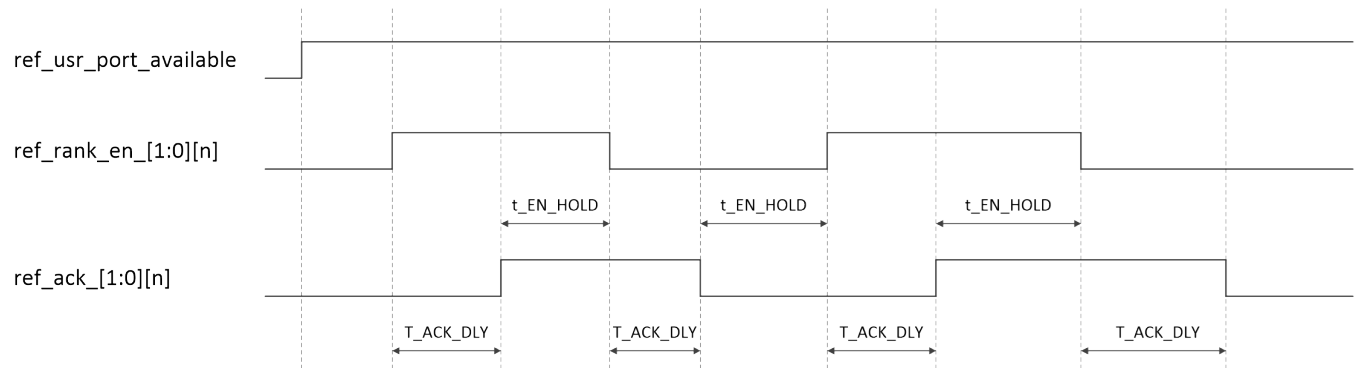
After the "ref\_usr\_port\_available / cal\_done" is asserted then "ref\_rank\_en" can be asserted high by the PL and needs to stay asserted until it detects a high on the corresponding "ref\_ack". Completing this handshake signifies that the DDRMC has incremented the associated "pending refresh counter" by one and will schedule the refresh as soon as the controller can halt system traffic and DRAM timing is safe. To initiate another user request, the PL must de-assert "ref\_rank\_en", wait for "ref\_ack" to de-assert, and then assert "ref\_rank\_en" again to start the handshake cycle over again. This process is broken down into individual steps below:

Following is the handshake flow to request one refresh command:

1. PL waits for `ref_usr_port_available / cal_done` to assert
2. PL asserts `ref_rank_en`
3. PL waits for corresponding `ref_ack` bit to assert
4. PL de-asserts `ref_rank_en`
5. PL waits for corresponding `ref_ack` bit to de-assert
6. PL can loop back to step 2 to request another refresh

The following timing diagram shows two refresh request handshake cycles, with two timing parameters,  $t_{EN\_HOLD}$  and  $t_{ACK\_DLY}$ .  $t_{EN\_HOLD}$  is a requirement for the PL, and  $t_{ACK\_DLY}$  is a requirement for the DDRMC.

Figure 12: Timing Diagram



Timing Parameter	Minimum	Maximum
$t_{EN\_HOLD}$	0	n/a
$t_{ACK\_DLY}$	0	16 mc_clk cycles

**Note:**

1. In user mode the MC does not track tREFI or postponed Autorefreshes. Instead, it issues Refreshes when it receives a request to a target Rank through the PL.
2. The maximum interval to schedule a refresh is  $7 \cdot t_{REFI}$ , namely the maximum number of postponed refreshes for a target is 6. Failure to meet this requirement will possibly violate the tRAS max.
3. In user refresh mode with LPDDR4, per-bank refresh is not allowed.
4. For a multi-rank system made up of DDR4 3DS devices, you must make a refresh request to all the ranks during every request.
5. While under single channel configuration, enable the clock gate for channel 1 or make a refresh request to both channels (even when channel 1 is not configured) to work successfully under the user refresh mode.
6. The number of refresh requests made to each of the ranks should be the same by every (tREFI/refresh\_rate).
7. When `ref_usr_port_available` / `cal_done` are de-asserted, the DDRMC will not execute any user requested refreshes.
8. The ZQCS/ZQCal\_START command will be issued internally at ZQ interval following a refresh command requested through PL. If you do not want periodic ZQ calibration then `block_periodic_cal_*` should be asserted.



## DRAM Power Down

The Versal Memory Controller supports activity-based DRAM Power Down. When the Memory Controller has been idle for a programmable amount of clock cycles, it will drive the DRAM `CKE` low to enter power down mode. The DRAM will be periodically taken out of power down mode as required for refresh operations or for ECC scrubbing. The DRAM will be taken out of power down mode automatically when memory transactions are received. This will have a delay of about 50 memory clock cycles.

## PHY

The PHY is considered the low-level physical interface to an external DDR4 or LPDDR4/4X SDRAM device as well as all calibration logic for ensuring reliable operation of the physical interface itself. The PHY generates the signal timing and sequencing required to interface to the memory device.

The PHY contains the following features:

- Clock/address/control-generation logic
- Write and read datapaths
- Logic for initializing the SDRAM after power-up

In addition, the PHY contains calibration logic to perform timing training of the read and write datapaths to account for system static and dynamic delays.

## PHY Architecture

The Versal architecture PHY is composed of dedicated blocks and integrated calibration logic. The dedicated blocks are structured adjacent to one another with back-to-back interconnects to minimize the clock and datapath routing necessary to build high performance physical layers. The Memory Controller and calibration logic communicate with this dedicated PHY in the slow frequency clock domain, which is either divided by four or divided by two. This depends on the DDR4 or LPDDR4/4X memory clock.

## Memory Initialization and Calibration Sequence

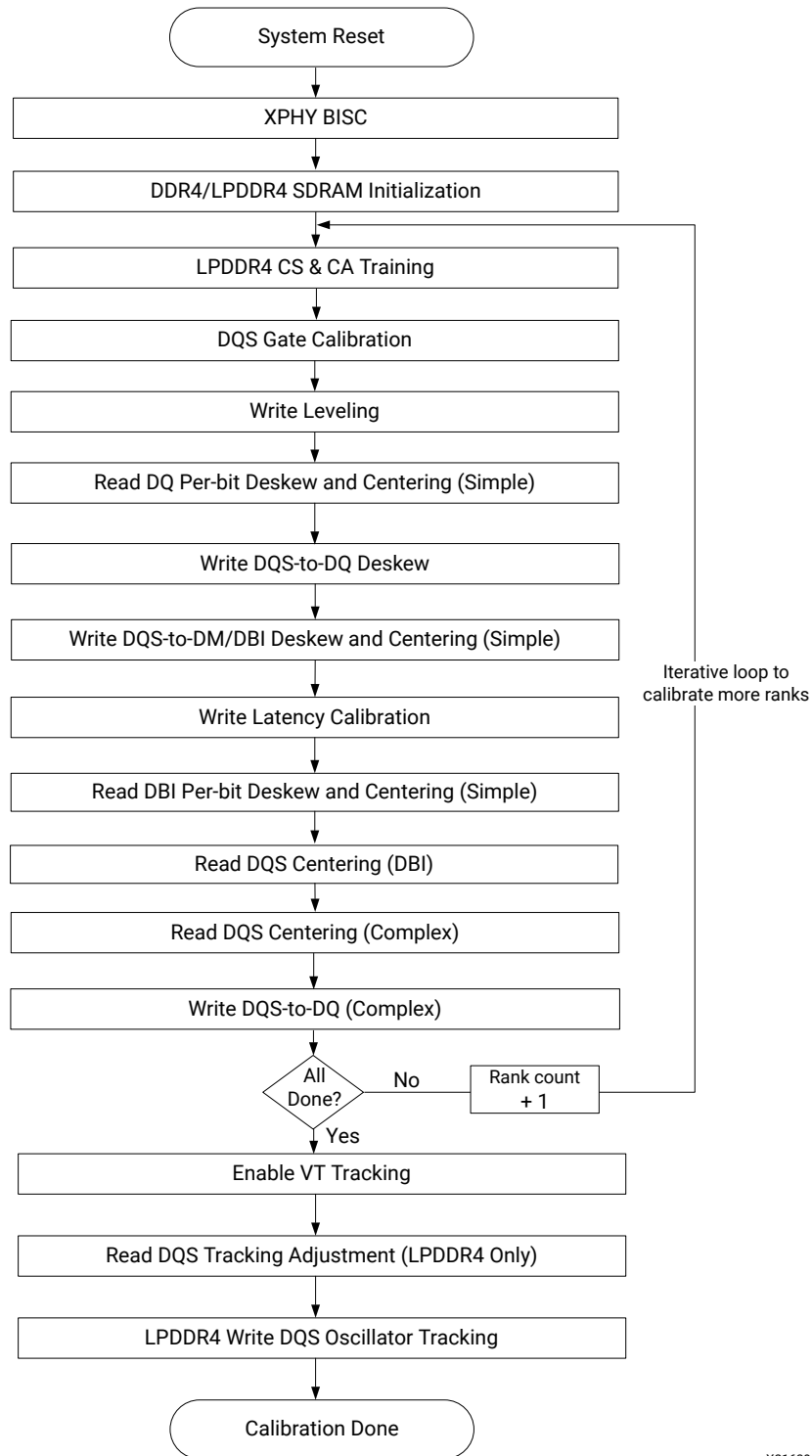
After deassertion of the system reset, the PHY performs the required internal calibration steps:

1. The built-in self-check of the PHY (BISC) is run. BISC is used in the PHY to compute internal skews for use in voltage and temperature tracking after calibration is completed.
2. After BISC is completed, calibration logic performs the required power-on initialization sequence for the memory.
3. This is followed by several stages of timing calibration for the write and read datapaths.

4. After calibration is completed the PHY calculates internal offsets to be used in voltage and temperature tracking.
5. PHY indicates calibration is finished and the controller begins issuing commands to the memory.

The following figure shows the overall flow of memory initialization and the different stages of calibration. While the figure shows an iterative loop to calibrate other ranks after all calibration stages have been completed for the first rank, in reality this iteration occurs on a stage-by-stage basis, after each stage is completed.

Figure 13: PHY Overall Initialization and Calibration Sequence



X21639-111720

## LPDDR4/4X CS and CA Training

LPDDR4/4X SDRAM provides a mechanism for training the command bus prior to enabling termination for high-frequency operation. For operation above 1866 Mb/s, this stage of training uses Command Bus Training mode to align the CS and CA signals at the DRAM relative to CK. In this mode, the DRAM uses CS and CK to capture the values on the CA pins, and it feeds back the result to the controller on the DQ[13:8] pins. The training is accomplished in two phases: CS Training and CA Training.

CS Training is completed first to align CS transitions with the falling edge of CK. A static pattern is driven on the CA pins, while the CS to CK timing is varied. By comparing the pattern received on the DQ pins relative to the pattern sent on the CA pins, the controller can identify the noise region. When this has been completed, the CS delay is adjusted to center the rising CK edge  $\frac{1}{2}$  clock cycle away from the center of the noise region. After aligning CS to CK, CA Training uses a toggling pattern on the CA lines to deskew and center the CA signals relative to CK.

## DQS Gate

During this stage of calibration, the read DQS preamble is detected and the gate to enable data capture within the FPGA is calibrated to be one clock cycle before the first valid data on DQ. The coarse and fine DQS gate taps (`RL_DLY_COARSE` and `RL_DLY_FINE`) are adjusted during this stage. Read commands are issued with gaps in between to continually search for the DQS preamble position. The DDR4/LPDDR4/4X preamble training mode is enabled during this stage to increase the low preamble period and aid in detection. During this stage of calibration, only the read DQS signals are monitored and not the read DQ signals. DQS Preamble Detection is performed in parallel for all bytes. During this stage of calibration, the coarse taps are first adjusted while searching for the low preamble position and the first rising DQS edge.

If the preamble is not found, the read latency is increased by one. The coarse taps are reset and then adjusted again while searching for the low preamble and first rising DQS edge. After the preamble position is properly detected, the fine taps are then adjusted to fine tune and edge align the position of the sample clock with the DQS.

## Write Leveling

Write leveling allows the controller to adjust each write DQS phase independently with respect to the CK forwarded to the DDR4/LPDDR4/4X SDRAM device. This compensates for the skew between DQS and CK and meets the `tDQSS` specification.

During write leveling, DQS is driven by the FPGA memory interface and DQ is driven by the DDR4/LPDDR4/4X SDRAM device to provide feedback. DQS is delayed until the 0 to 1 edge transition on DQ is detected. The DQS delay is achieved using both `ODELAY` and coarse tap delays.

After the edge transition is detected, the write leveling algorithm centers on the noise region around the transition to maximize margin. This second step is completed with only the use of `ODELAY` taps.

## ***Read DQ Per-Bit Deskew and Centering (Simple)***

Read DQ Per-bit Deskew and Centering is performed over multiple stages to maximize the data eye and center the internal read sampling clock in the read DQ window for robust sampling. To achieve this, Read Eye Training performs the following sequential steps:

1. Maximizes the DQ eye by removing skew and On-Chip Variation (OCV) effects using per-bit read DQ deskew.
2. Sweeps DQS across all DQ bits and finds the center of the data eye using both easy (Multi-Purpose register data pattern) and complex data patterns. Centering of the data eye is completed for both the DQS and DQS#.
3. Post calibration, continuously maintains the relative delay of DQS versus DQ across the VT range.

### **Read Per-Bit Deskew**

Read Per-Bit deskew is performed on a per-bit basis whereas Read DQS centering is performed on a per-nibble basis. During per-bit deskew, a predefined training pattern of `101010 . . .` is read back from the DRAM while DQS adjustments (PQTR and NQTR individual fine taps on DQS) and DQ adjustments (IDELAY) are made. At the end of this stage, the DQ bits are internally deskewed to the left edge of the incoming DQS.

### **Read DQS Centering (Simple)**

During Read DQS Centering (simple), the toggling `101010 . . .` MPR pattern is continuously read back while DQS adjustments (PQTR and NQTR individual fine taps on DQS) and DQ adjustments (IDELAY) are made. This is to establish an initial DQS center point using an easy pattern that does not rely on writing a pattern to the DRAM.

## ***Write DQS to DQ Deskew***

Write DQS to DQ calibration is required to center align the write DQS in the write DQ window per bit. At the start of Write DQS Centering and Per-Bit Deskew, DQS is aligned to CK but no adjustments on the write window have been made. Write window adjustments are made in the following two sequential stages:

1. Write Per-Bit Deskew
2. Write DQS Centering

## Write DQS-to-DQ Per-Bit Deskew

During write per-bit deskew, a toggling 10101010 pattern is continuously written and read back while making 90° clock phase adjustments on the write DQ along with individual fine ODELAY adjustments on DQS and DQ. At the end of per-bit write DQ deskew, the write DQ bits are aligned as they are transmitted to the memory.

## Write DQS-to-DQ Centering

During Write DQS Centering, the same toggling 10101010 pattern is continuously written and read back. ODELAY adjustments on DQS and DQ are also made but all of the DQ ODELAY adjustments for a given byte are made in step to maintain the previously deskewed alignment.

## Write DQS-to-DM/DBI Deskew and Centering (Simple)

When the write DBI option is selected for DDR4/LPDDR4, the pin itself is calibrated as a DM and write DBI are enabled at the end of calibration.

In all previous stages of calibration, data mask signals are driven low before and after the required amount of time to ensure they have no impact on calibration. Now, both the read and the writes have been calibrated and data mask can reliably be adjusted. If DM signals are not used within the interface, this stage of calibration is skipped.

During DM Calibration, a data pattern of 55555555\_55555555 is first written to address 0x000 followed by a write to the same address but with a data pattern of BBBB BBBB\_BBBBBBBB with DM asserted during the rising edge of DQS. A read is then issued where the expected read back pattern is all 0xBs except for the data where DM was asserted. In these masked locations, a 5 is expected. The same series of steps completed during Write Per-Bit Deskew and Write DQS Centering is then completed but for the DM bits.

## Write Latency Calibration

Write Latency Calibration is required to align DQS to the correct CK edge. During write leveling, DQS is aligned to the nearest rising edge of CK. However, this might not be the edge that captures the write command.

Depending on the interface type (UDIMM, RDIMM, LRDIMM, or component), the DQS could either be one CK cycle earlier than, two CK cycles earlier than, or aligned to the CK edge that captures the write command.

This is a pattern based calibration where coarse adjustments are made on a per byte basis until the expected on time write pattern is read back. The process is as follows:

1. Issue extended writes followed by a single read.
2. Check the pattern readback against the expected patterns.

3. If necessary add coarse adjustments.
4. Repeat until the on time write pattern is read back, signifying DQS is aligned to the correct CK cycle, or an incorrect pattern is received resulting in a Write Latency failure.

### ***Read DBI Per-Bit Deskew and Centering***

If the Read DBI option is selected for DDR4/LPDDR4, the DBI input pin is calibrated by writing a 101010 . . . pattern into the DRAM, enabling the read DBI functionality, and reading back the pattern from the DRAM. The DRAM sends the data back as 11111111, (or 00000000\_00000000 for LPDDR4), but the DBI pin itself has the 101010 . . . pattern, that is used to calibrate the DBI input pin itself.

### ***Read DQS Centering (DBI)***

If the Read DBI option is selected for DDR4/LPDDR4/4X, the position of the DQS in the data valid window must also use the timing information of the DBI pin itself, because the DBI pin can be the limit to the data valid window.

The 0F0F0F0F pattern is written to the DRAM and read back with read DBI enabled. The DRAM sends the data back as FFFFFFFF but the DBI pin has the clock pattern 01010101, which is used to measure the data valid window of the DBI input pin itself. The final DQS location is determined based on the aggregate window for the DQ and DBI pins.

### ***Read DQS Centering (Complex)***

The final stage of DQS read centering that is completed before normal operation is repeating the steps performed during MPR DQS read centering but with a more difficult/complex pattern. The purpose of using a complex pattern is to stress the system for SI effects such as ISI and noise while calculating the read DQS center position. This ensures that the read center position can reliably capture data with margin in a true system.

### ***Write DQS-to-DQ Centering (Complex)***

**Note:** The calibration step is only enabled for the first rank in a multi-rank system. Also, this is only enabled for data rates above 1,600 Mb/s.

For the same reasons as described in the Read DQS Centering (Complex), a complex data pattern is used on the write path to adjust the Write DQS-to-DQ alignment. The same steps as detailed in the Write DQS-to-DQ Centering are repeated just with a complex data pattern.

### **Related Information**

[Read DQS Centering \(Complex\)](#)

[Write DQS-to-DQ Centering](#)

### ***Enable VT Tracking***

After the DQS gate multi-rank adjustment (if required), a signal is sent to the XPHY to recalibrate internal delays and to start voltage and temperature tracking.

For multi-rank systems, when all nibbles are ready for normal operation the XPHY requires two write-read bursts to be sent to the DRAM before starting normal traffic. A data pattern of `F00FF00F` is used for the first and `0FF00FF0` for the second. The data itself is not checked and is expected to fail.

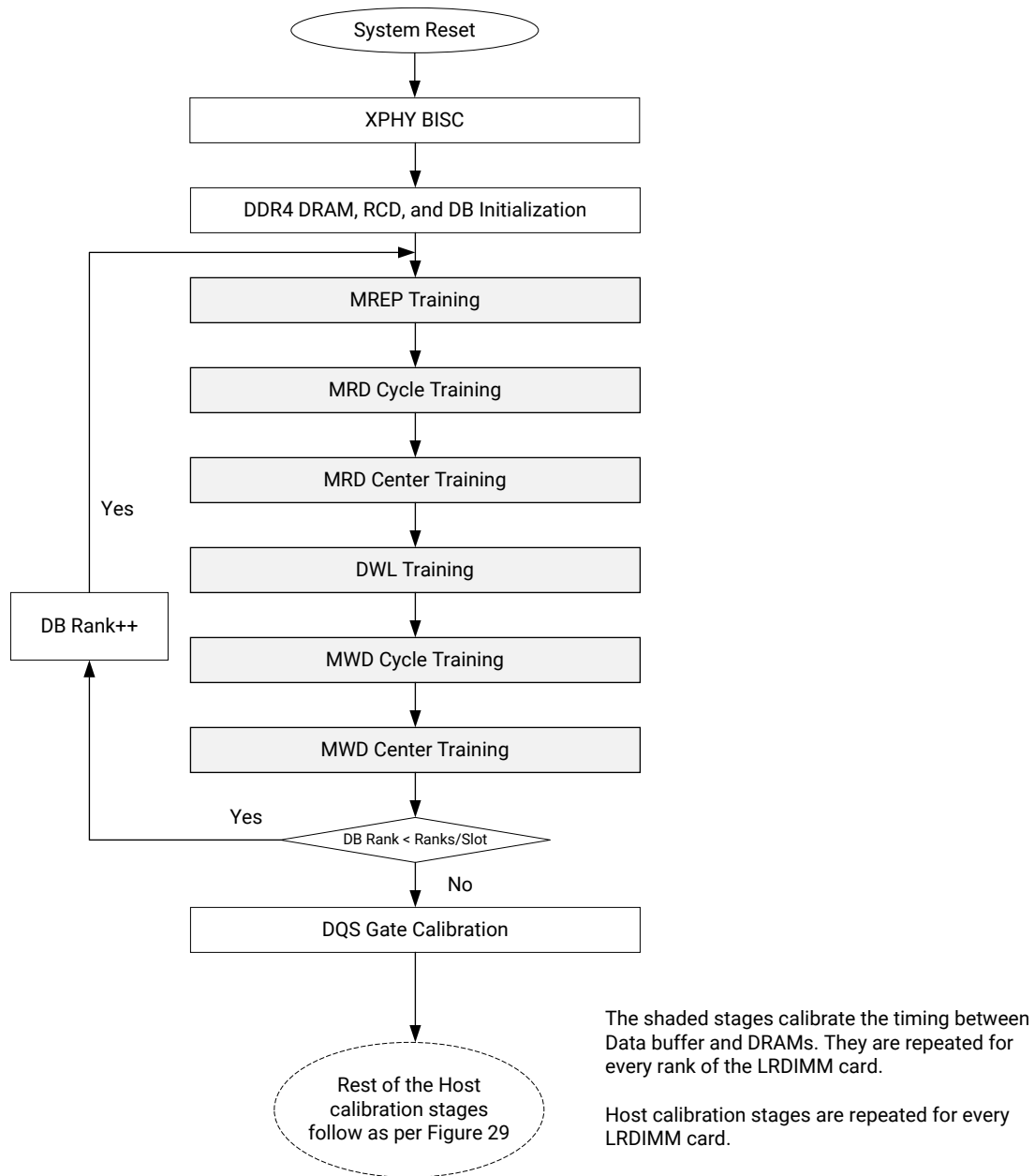
### ***DDR4 LRDIMM Memory Initialization and Calibration Sequence***

Most of the LRDIMM calibration sequence details are in line with the DDR4 core calibration sequence details as described in the previous Memory Initialization and Calibration Sequence section, unless otherwise stated below.

The following figure shows the overall flow of memory initialization and the different stages of the LRDIMM calibration sequence.



Figure 14: LRDIMM Calibration Sequence



X21646-111720

The following data buffer calibration stages are added to meet the timing between the data buffer and DRAMs and these are repeated for every rank of the LRDIMM card/slot.

- MREP Training
- MRD Cycle Training
- MRD Center Training
- DWL Training

- MWD Cycle Training
- MWD Center Training

The host side calibration stages exercise the timing between host and data buffer, and they are performed once for every LRDIMM card/slot.

All the calibration stages between data buffer and DRAMs are exercised first, and then the host side calibration stages are exercised.

At the end of each of the data buffer calibration stages, Per Buffer Addressing (PBA) mode is enabled to program the calibrated latency and delay values into the data buffer registers.

The following sections describe the data buffer calibration stages.

### **DRAM Interface MDQ Receive Enable Phase (MREP) Training**

This training aligns the Read MDQS phase with the data buffer clock. In this training mode, the host sends a sequence of read commands, the DRAM sends out the MDQS, the data buffer samples the strobe with the clock, and feeds back the result on DQ. Calibration continues to perform this training to find the 0 to 1 transition on Read MDQS sampled with the data buffer clock.

### **DRAM-to-DB Read Delay (MRD) Cycle Training**

This training finds the correct cycle to maintain the set Read Latency value at the data buffer. In this training mode, the host pre-programs the data buffer MPR registers with the expected pattern and issues read commands. The data buffer compares the read data with the expected data and feeds back the result on the DQ bus. Calibration picks the correct cycle based on the result of the comparison.

### **DRAM-to-DB Read Delay (MRD) Center Training**

This training aligns Read MDQS in the center of the Read MDQ window at the data buffer. In this training mode, the host pre-programs the data buffer MPR registers with the expected pattern and issues the commands. The data buffer compares the read data with the expected data and feeds back the result on the DQ bus. Calibration finds the left and right edges of the Read MDQ valid window and centers Read MDQS in it.

### **DRAM Interface Write Leveling (DWL) Training**

This training aligns the Write MDQS phase with the DRAM clock. In this training mode, the data buffer drives the MDQS pulses, the DRAM samples the clock with MDQS, and feeds back the result on MDQ. The data buffer forwards this result from MDQ to DQ. Calibration continues to perform this training to find a 0 to 1 transition on the clock sampled with the Write MDQS at the DRAM.

## DB-to-DRAM Write Delay (MWD) Cycle Training

This training finds the right cycle to maintain the set Write Latency value in the DRAM. In this training mode, the host pre-programs the data buffer MPR registers with the expected pattern, issues write commands to load the data into memory, and issues reads to the memory. The data buffer compares the read data with the expected data and feeds back the result on to the DQ bus. Calibration identifies the correct cycle based on the result of the comparison.

## DB-to-DRAM Write Delay (MWD) Center Training

This training center aligns Write MDQS in the Write MDQ window at the DRAM. In this training mode, the host pre-programs the data buffer MPR registers with the expected pattern, issues write commands to load the data into memory, and issues reads to the memory. The data buffer compares the read data with the expected data and feeds back the result on the DQ bus. Calibration finds the left and right edges of the MDQ valid window and centers MDQS in it.

## XPLL

Each IO Bank contains two XPLLs, one of which is required by the integrated DDRMC. If enabled, the DDRMC uses the left XPLL in the second bank of a triplet in all configurations. In addition, if DDRMC pins are assigned to other banks in that triplet, then the left XPLL for each used bank is also required.

For example, the first triplet of the VC1902 contain banks 700, 701, and 702.

- XPLL\_X0Y0 and XPLL\_X1Y0 are in bank 700
- XPLL\_X2Y0 and XPLL\_X3Y0 in bank 701
- XPLL\_X4Y0 and XPLL\_X5Y0 in bank 702

Bank 701 is the second bank in the triplet, and XPLL\_X2Y0 is the left XPLL in that IO bank. So XPLL\_X2Y0 is used by the DDRMC regardless of the configuration, and XPLL\_X0Y0 and XPLL\_X4Y0 may be used if DDR pins are used in banks 700 and 702 respectively.

# Designing with the Core

The Versal<sup>®</sup> ACAP tool flow introduces a pair of IP cores, the AXI NoC and the AXI4-Stream NoC (AXIS NoC). These IP cores act as logical representations of the Versal programmable network on chip (NoC). The `axi_noc` supports the memory mapped AXI protocol while the `axis_noc` supports the AXI4-Stream protocol. A Versal platform design might include multiple instances of each of these IP cores. Each instance specifies one or more connections to be mapped onto the physical NoC, along with the quality of service requirements for each connection. IP integrator automatically aggregates the connectivity and quality of service information from all of the logical NoC instances to form a unified traffic specification. This traffic specification is used to compute an optimal configuration for the NoC.

The integrated memory controllers (MCs) are integrated into the `axi_noc` IP core. An instance of the `axi_noc` can be configured to include one, two, or four instances of the integrated MC. If two or four instances of the MC are selected, they are configured to form a single interleaved memory. If multiple non-interleaved memory controllers are required, each controller requires a separate `axi_noc` instance.



---

**RECOMMENDED:** For guidance on the NoC/DDRMC design flow, refer to the [NoC DDRMC Design Flow Tutorial](#).

---

---

## Introduction to the Inter-NoC Interface

The Inter-NoC Interface (INI) provides a means of connecting two NoC instances (either `axi_noc` or `axis_noc`). An INI link represents a logical connection within the physical NoC that is resolved at the time the NoC compiler is called.

The INI does not directly represent NoC resources, instead the INI connectivity is implemented in the switch network by the NoC compiler. It represents a connection from an NMU in one AXI NoC instance to one or more NSUs or DDRMCs in another instance. An INI port can be an input or an output from an instance of the `axi_noc` or `axis_noc`. An INI output port on one NoC instance can connect only to an INI input port on another. An INI input or output port appears in the connectivity matrix like any AXI input or output port, allowing connectivity from an AXI input (NMU) to an INI output, or from an INI input to an AXI output or DDRMC port.

Quality of Service traffic-class for a path that includes an INI connection must be specified at the NMU. Quality of Service bandwidth and other per-path settings must be set in the axi\_noc that owns the path. The axi\_noc QoS page displays the ownership information.

## INI Topology

INI can be connected in a 1:N topology, which is one NMU with fanout to one or more NSU or DDRMC. Or a N:1 topology, which is one or more NMUs to a single NSU or DDRMC. The path ownership (where QoS is set) is typically at the 'N' side of the INI path. This allows QoS to be set separately for each INI path.

INI connections have an advanced option parameter called 'ini strategy'. This parameter can be set to 'driver' or 'load'. When set to 'driver' the INI represents the NMU (the driver). When set to 'load' the INI represents the NSU or DDRMC (the load). In some cases this parameter can be set to 'auto' for inference by the tool, however in other cases it must be explicitly set. The path ownership is at the driver to load transition.

The INI supports the following use cases:

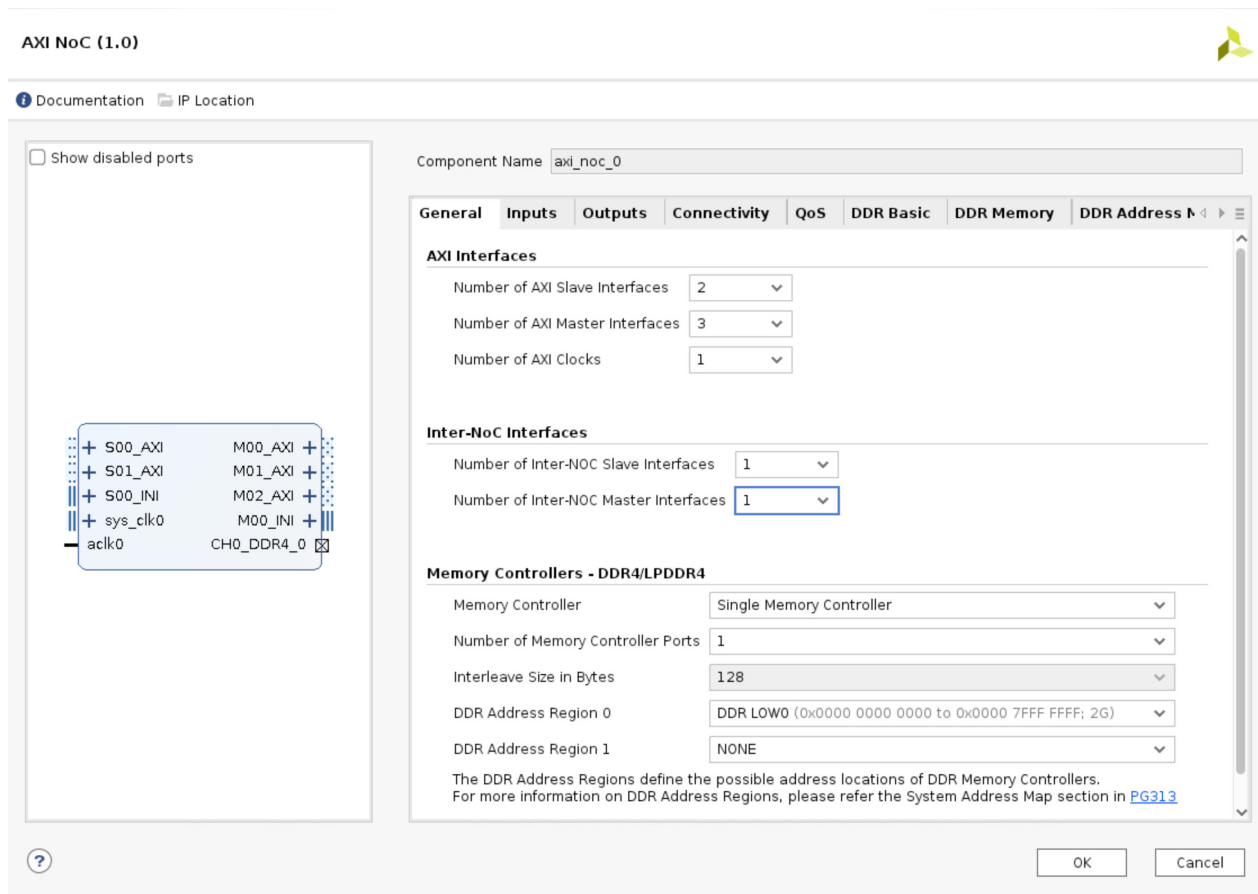
- **Connect one or more AXI masters to multiple, independent DDRMCs:** In this use case, the AXI master is connected to an AXI NoC IP configured with one or more INI outputs. Each INI output is connected to another AXI NoC configured to include a DDRMC.
- **Extend a NoC connection across levels of the design hierarchy:** In this use case, an AXI NoC in one level of hierarchy connects to an AXI NoC in another through INI ports at the hierarchy boundaries.

# Configuring the AXI NoC

## General Tab

The General tab of the Customize IP dialog box is shown in the following figure.

Figure 15: General Tab



- **Number of AXI Slave Interfaces:** This is the number of NoC ingress (NMU) ports.
- **Number of AXI Master Interfaces:** This is the number of NoC egress (NSU) ports.
- **Number of AXI Clocks:** This is the number of independent AXI clocks that will be used across the set of NMU and NSU ports. The association of clock signal with AXI port is made on the Inputs and Outputs tabs.
- **Number of Inter-NoC Slave Interfaces:** This is the number of Inter-NoC Interface ingress (INI) ports.

- **Number of Inter-NoC Master Interfaces:** This is the number of Inter-NoC Interface egress (INI) ports.
- **Number of Memory Controllers:** The number of integrated memory controllers connected to this axi\_noc instance. Must be 0, 1, 2, or 4. If 2 or 4, the memory controllers are interleaved.
- **Number of Memory Controller Ports:** The number of MC ports available to connect to in the connectivity tab. Must be 0-4. This corresponds to the number of NSU connections enabled for the MC.
- **Interleave Size:** When memory controller interleaving is enabled, set the number of bytes per interleave block. Must be one of {128, 256, 512, 1024, 2048, 4096}.
- **DDR Address Region 0/1:** This maps the DDRMC address with the unified system address map of the Versal device. For more information, refer to the System Address Map.

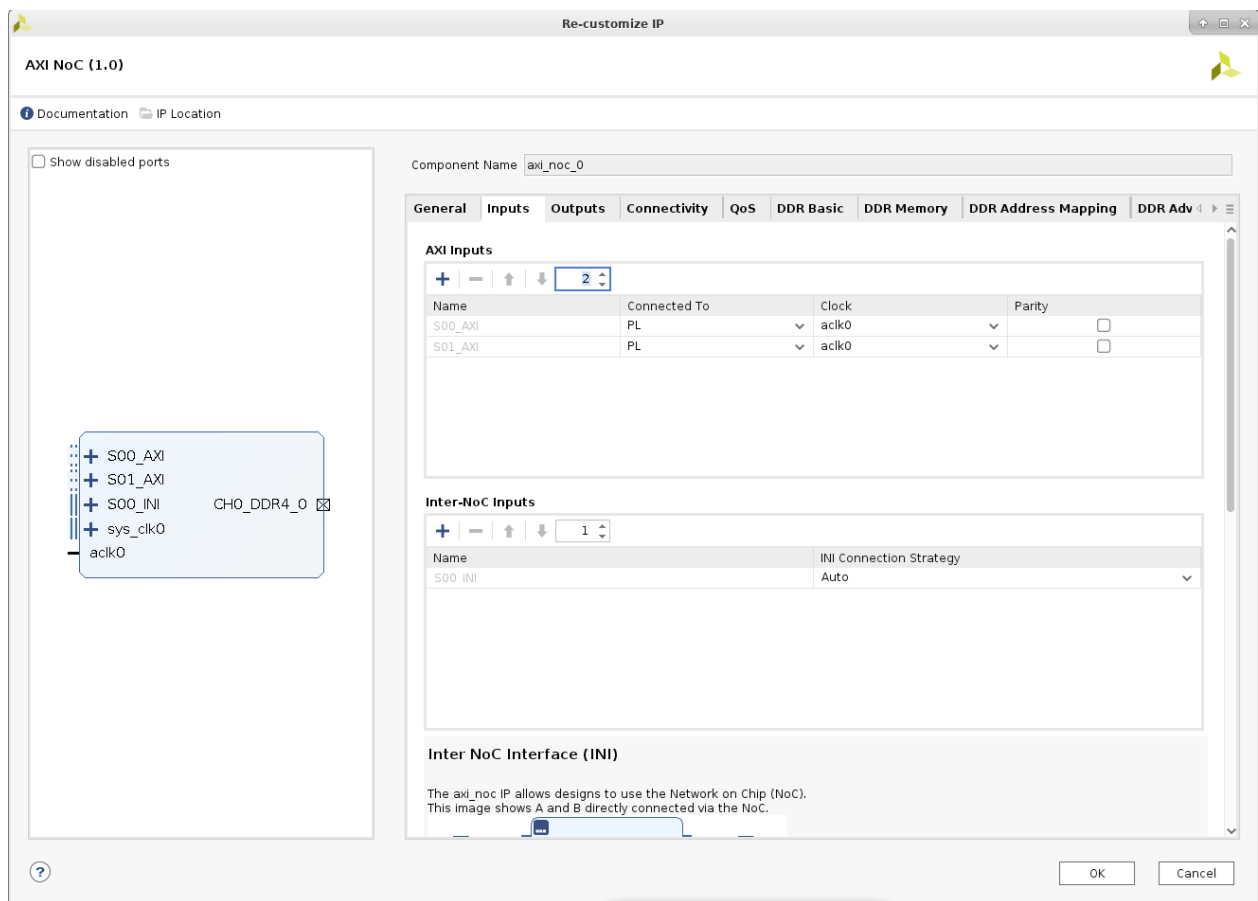
## Related Information

[Address Decoding and the System Address Map](#)

## Inputs Tab

The Inputs tab of the Customize IP dialog box is shown in the following figure and allows the configuration of the input ports, and associated clocks.

Figure 16: Inputs Tab



- **AXI Inputs:** Set the number of AXI inputs to this instance and configure each input. The configuration options are:
  - **Connected To:**
    - **AIE:** From the AI Engine array.
    - **PS Non-Coherent:** From one of the non-coherent interfaces of the PS.
    - **PS PMC:** From the Platform Management Controller of the PS.



- **PS RPU:** From the real-time processing unit of the PS.
- **PL:** From the programmable logic fabric.
- **PS Cache Coherent:** From one of the cache coherent interfaces of the PS.
- **PS PCIe:** From the PCIe® interface of the PS.
- **Clock:** Choose one of the AXI clocks defined on the General tab.
- **Parity:** Enables parity checking of the connection from the AXI master to the NMU. Even parity is used. Mapping of the parity bits is described in the following table:

**Table 12: NMU Parity Pin Mapping**

NMU Pin	Direction	Width	AXI Signal
AXI AWADDR Parity	IN	8	AWUSER[17:10]
AXI ARADDR Parity	IN	8	ARUSER[17:10]
AXI WDATA Parity	IN	64	WUSER[127],WUSER[125],WUSER[123],...,WUSER[3],WUSER[1]
AXI RDATA Parity	OUT	64	RUSER[127],RUSER[125],RUSER[123],...,RUSER[3],RUSER[1]
AXI WPOISON	IN	1	WUSER[0]
AXI RPOISON	OUT	1	RUSER[0]

**Note:** Parity checking is only available for inputs connected to PL. No parity checking is available from PS or AI Engine.

- **Interrupt:** Enable four bits of user-defined interrupt inputs. These are directed to the PMC for interrupt handling.
- **Inter-NoC Inputs:** Specify inputs from other axi\_noc instances using the Inter-NoC Interface (INI).

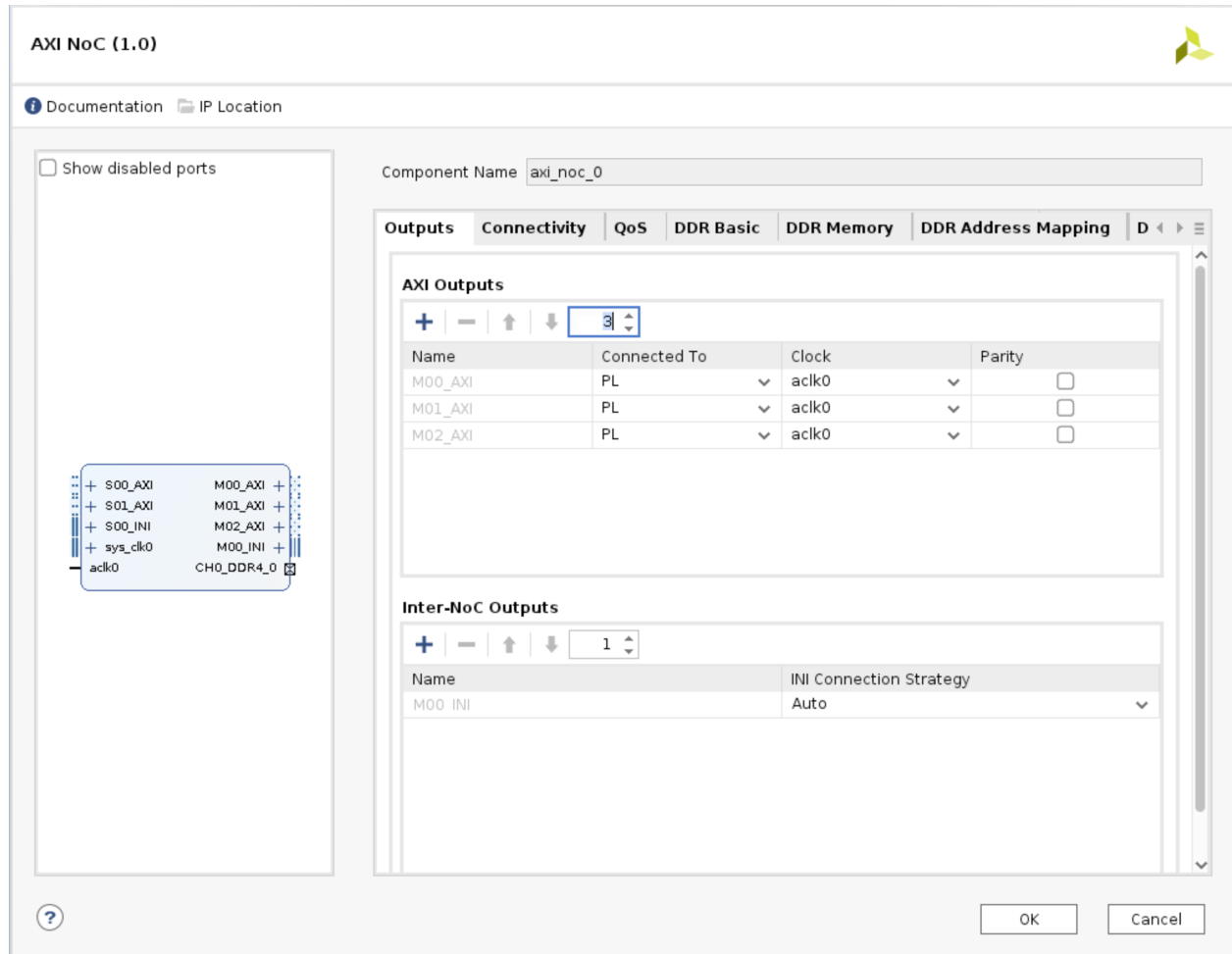
You can optionally set the INI Connection Strategy as follows:

- **Auto:** IP integrator determines the correct strategy. This is the default setting.
- **Single Driver:** Has a single driver and possibly multiple loads.
- **Single Load:** Has a single load and possibly multiple drivers.

## Outputs Tab

The Outputs tab of the Customize IP dialog box is shown in the following figure and allows the configuration of the output ports and associated clocks.

Figure 17: Outputs Tab



- **AXI Outputs:** Set the number of AXI outputs from this instance and configure each output. The configuration options are:
  - **Connected To:**
    - **AIE:** To the AI Engine array.
    - **PS Non-Coherent:** To one of the non-coherent interfaces of the PS.
    - **PS PMC:** To the Platform Management Controller of the PS.
    - **PL:** To the programmable logic fabric.

- **PS Cache Coherent:** To one of the cache coherent interfaces of the PS.
- **PS PCIe:** To the PCIe interface of the PS.
- **Clock:** Choose one of the AXI clocks defined on the General tab.
- **Parity:** Enables parity checking of the connection from the NSU to the AXI slave. Even parity is used. Mapping of the parity bits is described in the following table:

**Table 13: NSU Parity Pin Mapping**

NSU Pin	Direction	Width	AXI Signal
AXI AWADDR Parity	OUT	8	AWUSER[17:10]
AXI ARADDR Parity	OUT	8	ARUSER[17:10]
AXI WDATA Parity	OUT	64	WUSER[127],WUSER[125],WUSER[123],...,WUSER[3],WUSER[1]
AXI RDATA Parity	IN	64	RUSER[127],RUSER[125],RUSER[123],...,RUSER[3],RUSER[1]
AXI WPOISON	OUT	1	WUSER[0]
AXI RPOISON	IN	1	RUSER[0]

**Note:** Parity checking is only available for outputs connected to PL. No parity checking is available to PS or AI Engine.

- **Interrupt:** Enable four bits of user-defined interrupt inputs. These are directed to the PMC for interrupt handling.
- **Inter-NoC Outputs:** Specify outputs to other axi\_noc instances using the Inter-NoC Interface (INI).

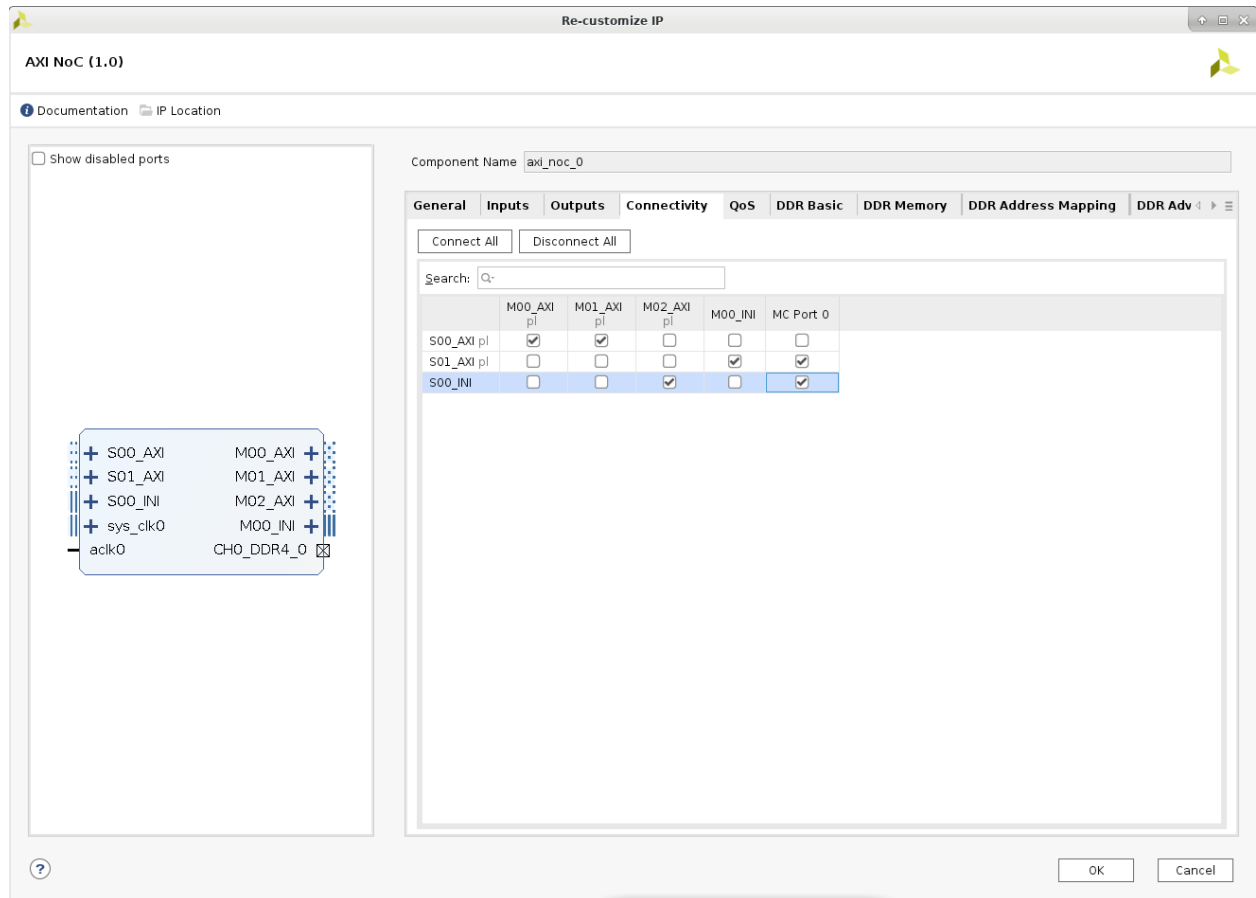
You might optionally set the INI Connection Strategy as follows:

- **Auto:** Allows IP integrator to determine the correct strategy. This is the default.
- **Single Driver (NMU). Load (NSU/MC) owns PR path and has QoS.:** Has a single driver and possibly multiple loads.
- **Single Load (NSU/MC). Driver (NMU) owns PR path and has QoS.:** Has a single load and possibly multiple drivers.

## Connectivity Tab

The Connectivity tab of the Customize IP dialog box is shown in the following figure.

Figure 18: Connectivity Tab



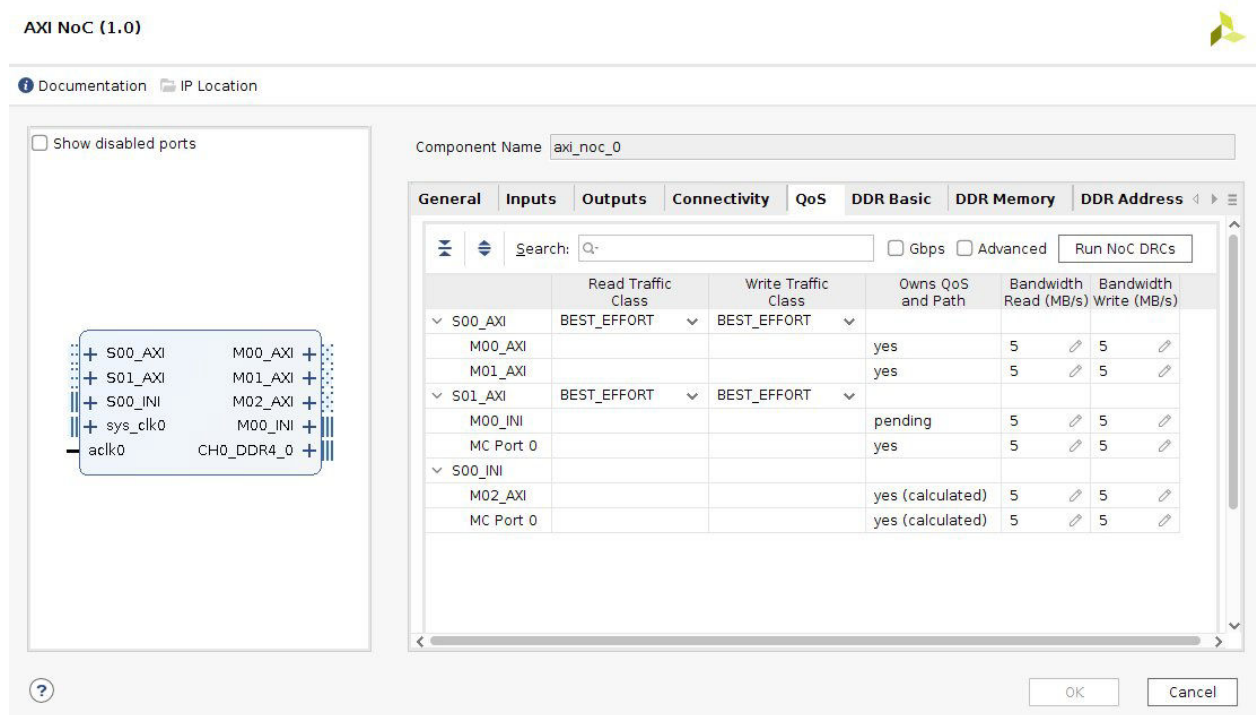
The Connectivity tab is used to define the connectivity through the axi\_noc. Connectivity is captured in the form of a connectivity matrix, as shown in the figure. The rows of the matrix correspond to the inputs to the axi\_noc instance, the columns correspond to the outputs. A check in the box at the intersection of a row and a column indicates a connection from the row input to the column output. The previous figure shows:

- Input S00\_AXI is connected to outputs M00\_AXI and M01\_AXI.
- Input S01\_AXI is connected to output M00\_INI and memory controller port 0.
- Inter-NoC input S00\_INI is connected to output M02\_AXI and to memory controller port 0.

## QoS Tab

The quality of service (QoS) tab of the Customize IP dialog box is shown in the following figure and allows the configuration of the quality of service properties of every path defined in the axi\_noc instance.

Figure 19: QoS Tab



- The first column of the QoS tab shows a tree structure of the connectivity defined in the Connectivity tab. The top of each tree (left aligned port names) are the NoC ingress ports; in the figure, S00\_AXI and S01\_AXI. Shown under each ingress port are the associated rows for each connected egress port.
- The second column defines the read traffic class. The possible values are LOW\_LATENCY, ISOCHRONOUS, and BEST\_EFFORT (default). The traffic class applies to all connections originating from the given ingress port.
- The third column defines the write traffic class. The possible values are ISOCHRONOUS and BEST\_EFFORT. Note that LOW\_LATENCY is not supported for write traffic. As with read traffic, the write traffic class applies to all connections originating at the given ingress port.

- The fourth column indicates whether this NoC instance owns the QoS setting for a given path. A NoC path may go through multiple NoC instances using INI. QoS settings are taken from the NoC instance that owns the QoS and path. QoS is ignored when a NoC instance does not own the path. Ownership is at the transition from an NMU or `strategy=Driver` to an NSU, MC, or `strategy=Load`. A value of 'pending' means the ownership will be calculated during validate, or by clicking the **Run NoC DRCs** button above. For a value of 'error', validate or click the **Run NoC DRCs** button above to see error details in the Tcl Console and messages window.
- The fifth column defines the Read bandwidth expressed in MB/s. Allowed values range from 0 (no read traffic is accepted) to the maximum data bandwidth of the NoC physical channel.

**Note:** Read bandwidth is expressed in Gb/s if the Gb/s option is checked ( $\text{Gb/s} = 8 * \text{MB/s} / 1000$ ).

- The sixth column defines the Write bandwidth expressed in MB/s. Allowed values range from 0 (no write traffic is accepted) to the maximum data bandwidth of the NoC physical channel.

**Note:** Read bandwidth is expressed in Gb/s if the Gb/s option is checked ( $\text{Gb/s} = 8 * \text{MB/s} / 1000$ ).

**Note:** The Gb/s check box allows bandwidth conversion between MB/s and Gb/s.

( $\text{Gb/s} = \text{MB/s} * 8 / 1000$ )

### Owns QoS and Path

A NoC path may go through multiple NoC instances using INI. QoS settings are taken from the NoC instance that owns the QoS and Path. QoS is ignored when a NoC does not own the path. Ownership is at the transition from an NMU or `strategy=Driver` to an NSU, MC, or `strategy=Load`.

A value of 'pending' means the ownership will be calculated during validate, or by clicking the **Run NoC DRCs** button above. For a value of 'error', validate or click the 'Run NoC DRCs' button above to see error details in the Tcl Console and messages window.

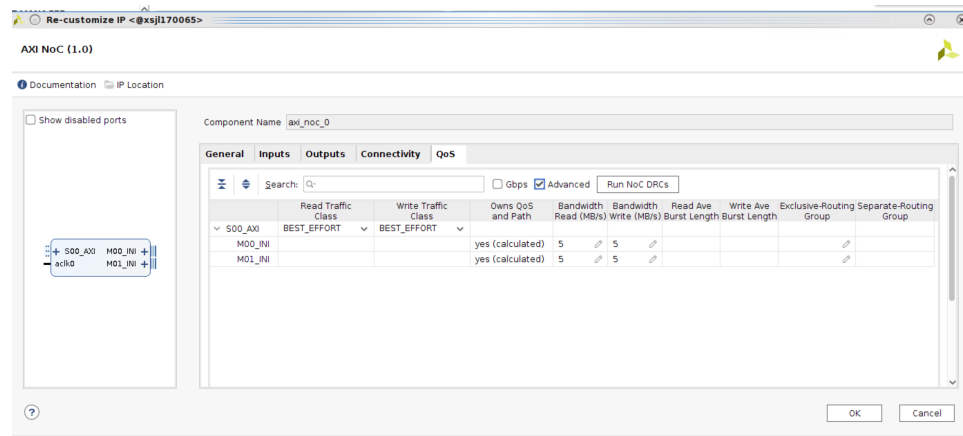
**Note:** If Owns QoS and Path is `no` then all settings BW and traffic class will be ignored.

### Run NoC DRCs

Run NoC DRCs across the design. Errors are listed in the Tcl Console and messages window. Update 'pending' entries in the 'Owns QoS' column by calculating ownership across the design. A NoC path may go through multiple NoC instances using INI. QoS settings are taken from the NoC instance that owns the QoS and Path. QoS is ignored when a NoC does not own the path. Ownership is at the transition from an NMU or `strategy=Driver` to an NSU, MC, or `strategy=Load`.

The QoS tab Advanced check box enables following columns:

Figure 20: Advanced QoS Tab



- Read/Write Ave Burst Length: Average burst length in beats:

**Note:** The NoC Compiler uses the average burst length to predict packet overhead. This influences the overall bandwidth required to meet QoS. See [Packetization Overhead](#) for more details.

- Lower Limit: 1
- Upper Limit:  $\text{average\_burst\_length} * \text{master\_data\_width} / 8 = 4096$  (4 KB data limit)
- Exclusive-Routing Group/Separate-Routing Group:
  - Both of these features are intended to support functional safety applications. The edit boxes allow you to enter arbitrary strings to define groups of NoC routing nets: nets with the same string are members of the same group. Nets in a given “exclusive routing group” are allowed to share physical resources with other members of the group, but are guaranteed by the compiler not to share resources with any nets outside of the group. Analogously, nets in a given “separate routing group” are guaranteed not to share resources with other members of the group, but may share resources with nets outside of the group.

# Configuring the Memory Controller

The axi\_noc IP core can be configured to include one or more integrated DDR memory controllers (MCs). In the current release of the tools, a given axi\_noc instance can be configured to include 0, 1, 2, or 4 MCs. The MC group appears as a single interleaved memory to all masters connected through the NoC.

In interleave mode, the application sees a single unified block of memory that spans the participating MCs. The NoC supports interleaving across two or four MCs by automatically chopping AXI requests into interleave block sized sub-requests and alternately sending the sub-requests to each of the participating MCs in turn.

If an axi\_noc instance is configured for one or more MCs, several additional tabs appear on the customize IP dialog box:

- **DDR Basic:** Allows the configuration of the controller/PHY mode and the clocking options. The controller type can be set to either DDR4 or LPDDR4. The clocking options include setting the memory frequency, and the system clock period and mode. In this release only differential mode clocks are supported.
- **DDR Memory Options:** Allows configuration of memory device options, the memory density parameters, the JEDEC timing parameters of the external DDR and the mode register settings.
- **DDR Address Mapping Options:** Allows the remapping of system address bits to DDR address bits. Three predefined mapping choices are available:
  - ROW, BANK, COLUMN
  - ROW, COLUMN, BANK
  - BANK, ROW, COLUMN

In addition to the predefined choices, you can construct a custom mapping of address bits to row, bank, and column bits.

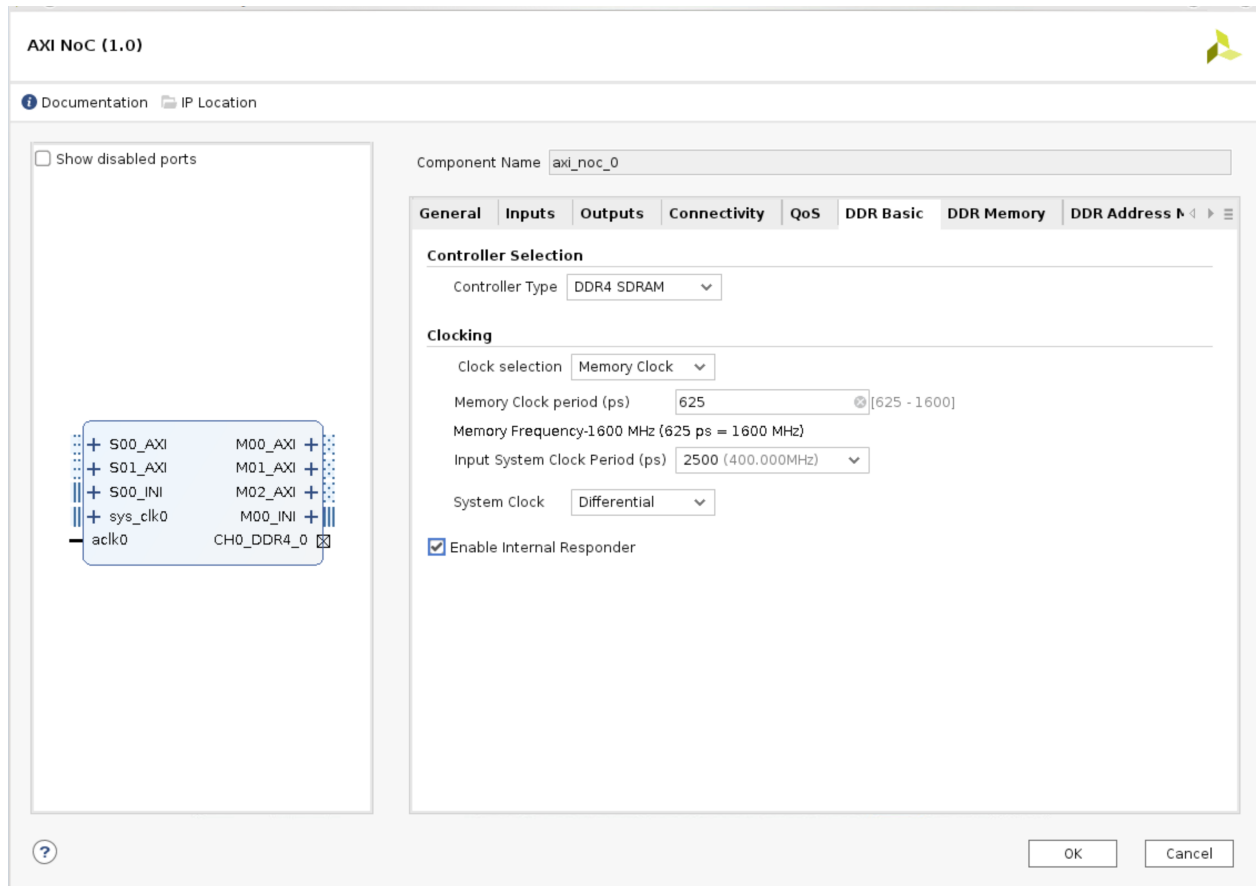
- **DDR Advanced:** Selects the system address regions and options for ECC, refresh, and power saving modes.



## DDR Basic Tab

The DDR Basic tab is shown in the following figure for reference.

Figure 21: DDR Basic Tab



- Controller Selection:** In the DDR Basic tab, use the Controller selection drop-down to select the required Controller Type.
  - DDR4 SDRAM
  - LPDDR4/4X SDRAM
- Clocking:** Clock selection provides two options; Memory Clock and System Clock. If Memory Clock is selected (as shown in the previous figure), a range of values in ps is provided for the Input System Clock Period for the provided Memory Clock period. The opposite will happen if System Clock is selected. A range of values in ps is provided for the Memory Clock Period for the provided System Clock Period. The range of values provided in is determined based on the clocking architecture. For most use cases it is easier to first set the Input System Clock Period and then select the desired Memory Clock period.
  - Memory Clock Period: Enter or select the desired memory interface clock period. This clock period is half the toggle rate for DDR4 or LPDDR4/4X memory interfaces.

- Input System Clock Period: Enter or select the desired Input System Clock Period. This clock is used to generate the clocking configuration and resulting memory interface options for the DDRMC.
- System Clock: Currently only Differential external input clocks are supported.
- Enable Internal Responder: Enables Xilinx DDR4 or LPDDR4/4X memory device models for simulation.
  - When the Enable Internal Responder option is unchecked there are additional pins exposed at the IP integrator top level which are hooked up to the memory model provided external to the AXI NoC IP. This default option enables customers to replace the Xilinx responder with their memory vendor model if desired.
  - When the Enable Internal Responder option is checked the Xilinx responder is included inside the AXI NoC IP, therefore no additional pins are added at the IP integrator level. To include the pins in simulation, search for “u\_ddr\_responder” and add it to waveform window.

## DDR Memory Options Tab

The DDR Memory Options tab is shown in the following figure for reference

Figure 22: DDR Memory Options Tab 1/2

AXI NoC (1.0)

Documentation IP Location

Component Name: `axi_noc_0`

**DDR Memory Options**

Device Type: Components

Speed Bin (Monolithic/3DS): DDR4-3200AC(24-24-24)

Base Component Width: x8

Memory Device Width: x8

**Memory Density Parameters**

Row Address Width: 16 Bank Group Width: 2

Bank Address Width: 2

Component Density: 8Gb, Memory Device Density: 8Gb, Memory Density per Channel: 8GB

Number of Channels: Single

Data Width per Channel (including ECC bits if enabled): 72

Data bits per Channel : 64, ECC bits per Channel: 8

Ranks: 1

Stack Height: 1

Slot: Single

Number of Memory Clocks: 1

**Timing Parameters**

tFAW (ps): 21000

tRRD\_S (nCK): 4

tRRD\_L (nCK): 8

tRAS (ps): 32000

tRCD (ps): 15000

tRP (ps): 15000

tRC (ps): 47000

tRFC (ps): 350000

tREFI (ps): 7800000

tRTP (ps): 7500

tWR (ps): 15000

tWTR\_S (ps): 2500

tWTR\_L (ps): 7500

txPR (nCK): 576

tCCD\_L (nCK): 8

OK Cancel

Figure 23: DDR Memory Options Tab 2/2

- **DDR Memory Options:** Select device type, speed bin, base component width and memory device width.
  - **Device Type:** Select from the following memory form factor options:
    - Components
    - UDIMMs
    - SODIMMs
    - RDIMMs
    - LRDIMMs
  - **Speed Bin (Monolithic/3DS):** Options to select a JEDEC supported speed bin for a selected DDR4 or LPDDR4/4X SDRAM controller type. For example, if a selection of x8 and 2 ranks is made then the tool infers it as DDP. When DDR4 is selected for the Controller Type on the DDR Basic tab, the second half of the pull-down has 3DS devices.
  - **Base Component Width:** The width of the base die on a selected Memory Device type.
  - **Memory Device Width:** DDR4 only. This option is applicable only for Component Memory Devices with a base component width of x8. Use this option to differentiate between a x8 device vs. a x16 with two x8 die.

- **Memory Density Parameters:**

- **Row Address Width:** Consult the memory data sheet addressing table.
- **Bank Group Width:** (DDR4 only) Automatically calculated.
- **Bank Address Width:** Automatically calculated.
- **Number of Channels:** One or two independent sub-channels within a single memory controller.
- **Data width:** The data width per channel for selected memory configuration.
- **Ranks:** Select the number of ranks based on the memory topology.
- **Stack Height:** (DDR4 only) Select the stack height of the 3DS memory device.
- **Slot:** (DDR4 only) Set the number of slots for the RDIMM/LRDIMM topology.
- **Number of Memory Clocks:** (DDR4 only) Set to 2 when using more than four DDR4DDP deep components at or above 1866 Mbps. For details refer to [DDR4 Dual CK Configuration](#).
- **ECC:** Select to enable ECC for a selected memory configuration.
- **Write DBI/Read DBI:** Data bus inversion selection on memory and controller side. The Write Data Mask (DM) option is also available in this drop down when applicable.
- **Channel Interleaving:** The DDRMC supports interleaving across two memory channels by automatically chopping AXI requests into interleave block-sized sub-requests and alternatively sending sub-requests to each of the two channels.
- **DRAM Command/Address Parity:** (DDR4 only) Check this option to enable Command/Address Parity.
- **CA Mirror (DDR4 only):** (DDR4 only) Enables address mirroring as required by memory topology.
- **Clamshell:** (DDR4 only) Enables Clamshell topology when selected. For details refer to [Clamshell Topology](#).

**Note:** Options 10, 13, 14, and 15 are supported in hardware only and are not reflected in simulations.

- **LP4 Pin Efficient:** (LPDDR4/X Only): Reduces the number of required pins at the expense of performance. This option is limited to single rank memory topologies.
- **Future Expansion for PCB Design (set per Interleaved MC):** Multiple options are available based on the memory configuration selection. Refer to [Pinout Options for Future Expansion](#). Following are the available options:
  - **Optimum:** Gives the standard pin out for a selected memory configuration.

- **Ranks Expansion:** Supports the selected memory configuration and creates pins for future rank expansion.
- **Slot Rank and Stack Height Expansion:** Supports the selected memory configuration and creates pins for future expansion.
- **CLK Expansion:** Set to 2 when using more than four DDR4 DDP deep components at or above 1866 Mb/s.



**IMPORTANT!** When an Expansion option is selected the tools will only generate the additional pin sites based on the current configuration. For example if future designs may require a Dual Rank DDR4 topology, then set the Ranks option to 2 so those pins are generated in the pinout.

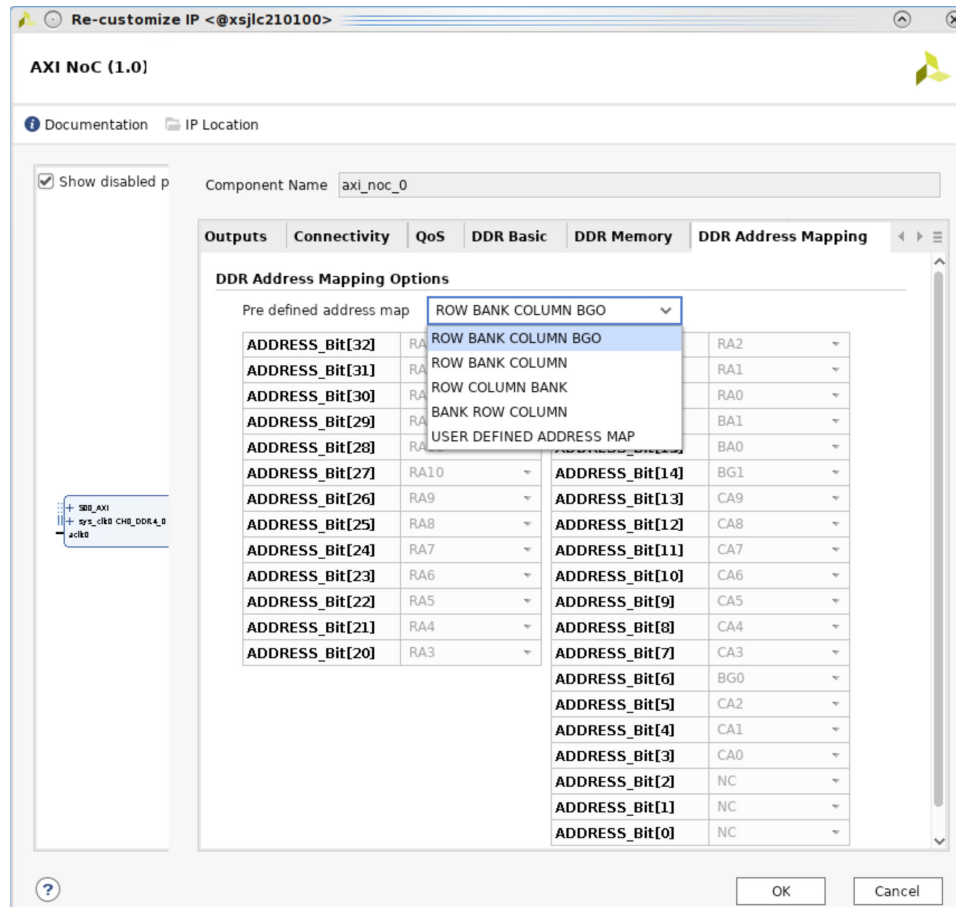
- **Flipped Pinout (set per Interleaved MC):** MC0 Flipped Pinout: Flipping reverses the ordering of all the nibbles within the triplet of banks. Some of the nibbles end up being 'shadowed' by hardened blocks on the die. Nibbles that are shadowed can only be used by the DDRMC. These cannot be re-purposed. For detailed information refer to [Pinout Rules](#).
- **Timing Parameters:** Default values are based on the JEDEC specification per selected speed bin and operating point.
- **Mode Register Settings:**
  - **CAS latency(nCK):** The CAS latency options for a selected memory configuration.
  - **CAS Write latency(nCK):** The CAS Write latency for a selected memory configuration.
- **Operating Temperature:** Available only for LPDDR4 SDRAM. Two options are available; Standard or High. Selecting **High** derates certain timing parameters.

The Timing Parameters selections under DDR4 Memory Options need to be selected from the memory data sheet based on the required memory speed bin. Default values are based on the JEDEC specification per selected speed.

## DDR Address Mapping Options Tab

The DDR Address Mapping Options tab is shown in the following figure for reference.

Figure 24: DDR Address Mapping

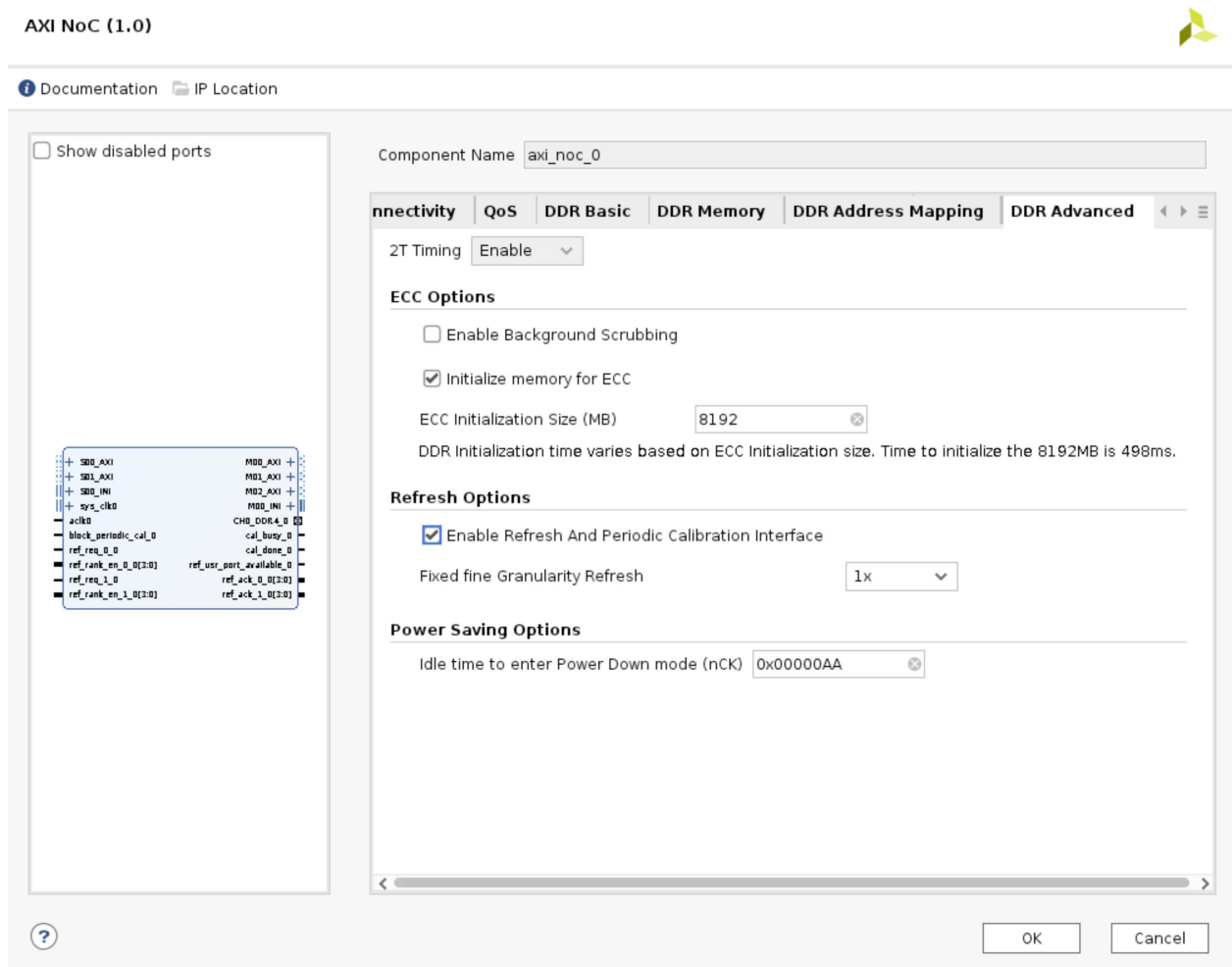


On the DDR4 Address Mapping Options tab, a predefined address map can be selected from the drop-down list, or individual address bits can be updated from the drop-down list after selecting "USER DEFINED ADDRESS MAP".

## DDR Advanced Tab

The DDR Advanced tab is shown in the following figure for reference.

Figure 25: DDR Advanced



Use the DDR Advanced tab to select the required advanced options listed.

**2T Timing (DDR4 Only):** Enabled by default for DDR component configurations. For other DDR configurations such as RDIMM/LRDIMM/UDIMM/SODIMM you can either enable or disable this option.

**ECC Options:** These options are only applicable for ECC enabled designs.

- Enable Background Scrubbing (DDR4 only): Refer to the [ECC Scrubbing](#) section for details. When ECC is enabled for LPDDR4, Background Scrubbing is automatically enabled.
- Initialize Memory for ECC: This option is enabled by default.



- ECC Initialization Size (MB): Initialization size can be changed by the user from the default value.

**Note:** You are responsible for initializing memory if memory initialization is disabled.

#### **Refresh Options:**

- Enable Refresh and Periodic Calibration Interface : Refer to Refresh section under "Integrated Memory Controller (DDRMC) Architecture"→"Memory Controller"→"Core Architecture".
- Fixed fine Granularity Refresh (DDR4 Only) : 1x, 2x, 4x. Fixed Fine Granularity Refresh modes can help reduce refresh protocol overhead in high temperature environments. Refer to vendor data sheets for JESD79-4C for more details.
- Refresh Type (LPDDR4 Only) : Select between All Bank or Per Bank refresh modes.

## Configuring the HBM

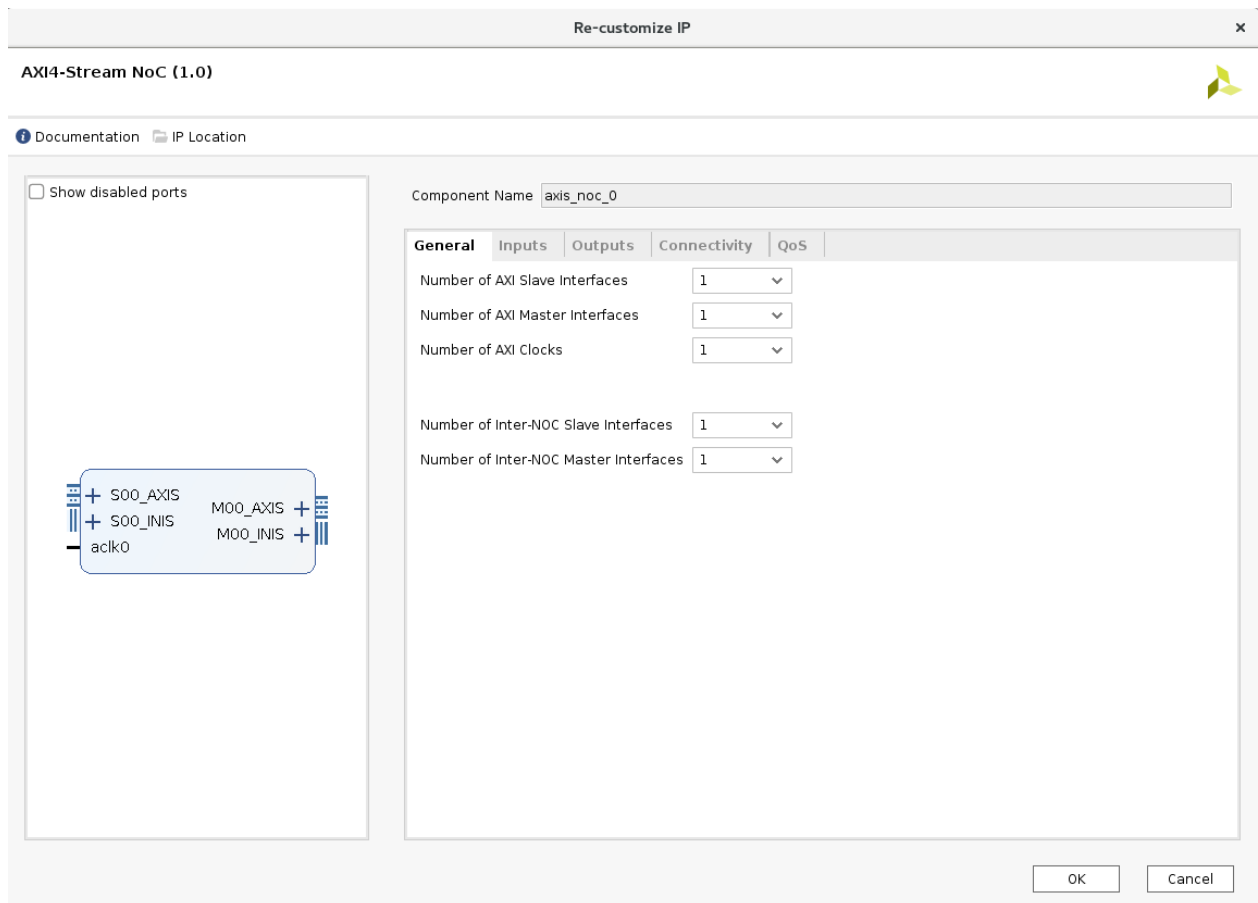
For information on configuring HBM, see Module 6 of the tutorial *Introduction to the NoC DDRMC Design Flow*, available in the Versal Premium [Lounge](#).

## Configuring the AXIS NoC

### General Tab

The General tab of the Customize IP dialog box is shown in the following figure.

Figure 26: General Tab



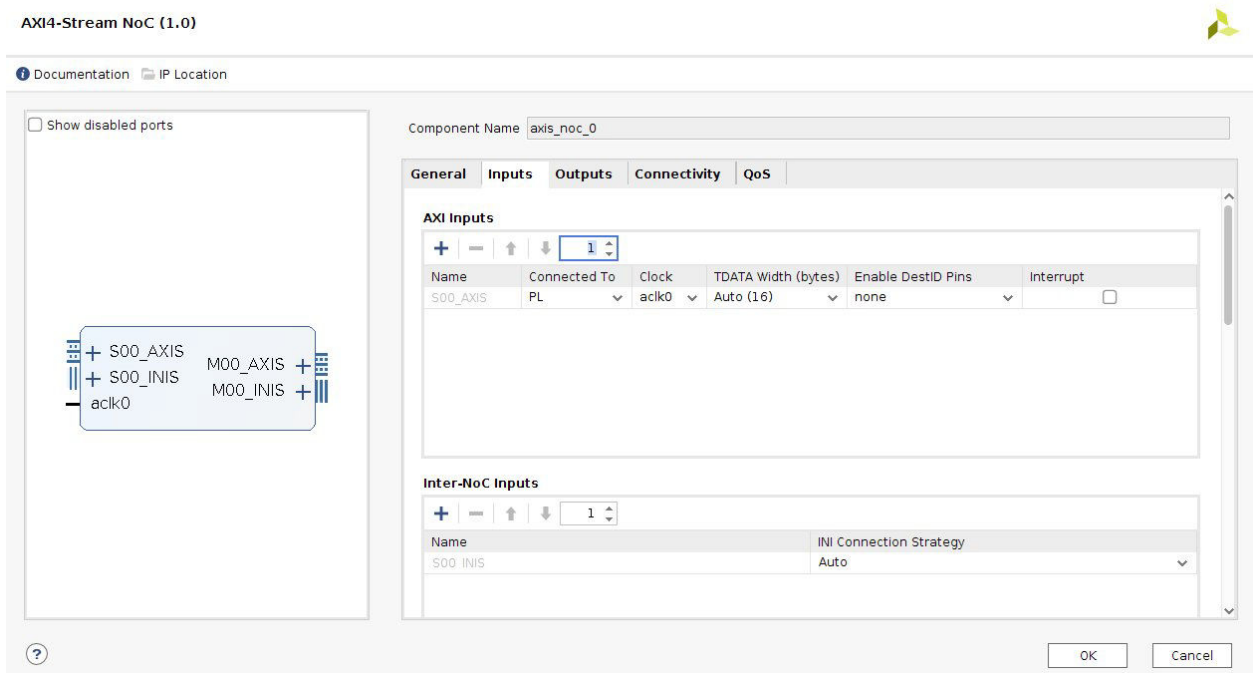
- **Number of AXI Stream Slave Interfaces:** This is the number of NoC ingress (NMU) ports.
- **Number of AXI Stream Master Interfaces:** This is the number of NoC egress (NSU) ports.

- **Number of AXI Clocks:** This is the number of AXI clocks that will be used across the set of NMC and NSU ports. The association of clock signal with AXI port is made on the Inputs and Outputs tabs.
- **Number of Inter-NoC Stream Slave Interfaces:** This is the number of Inter-NoC ingress ports.
- **Number of Inter-NoC Stream Master Interfaces:** This is the number of Inter-NoC egress ports.

## Inputs Tab

The Inputs tab of the Customize IP dialog box is shown in the following figure. This tab allows the configuration of the input ports, associated clocks, and destination ID pins.

Figure 27: Inputs Tab



- **AXI Inputs:** Set the number of AXI4-Stream inputs to this instance and configure each input. The configuration options of the Inputs tab are the same as the options on the Inputs tab of the AXI NoC.
- **Inter-NoC Inputs:** Set the number of INI input ports to this NoC.

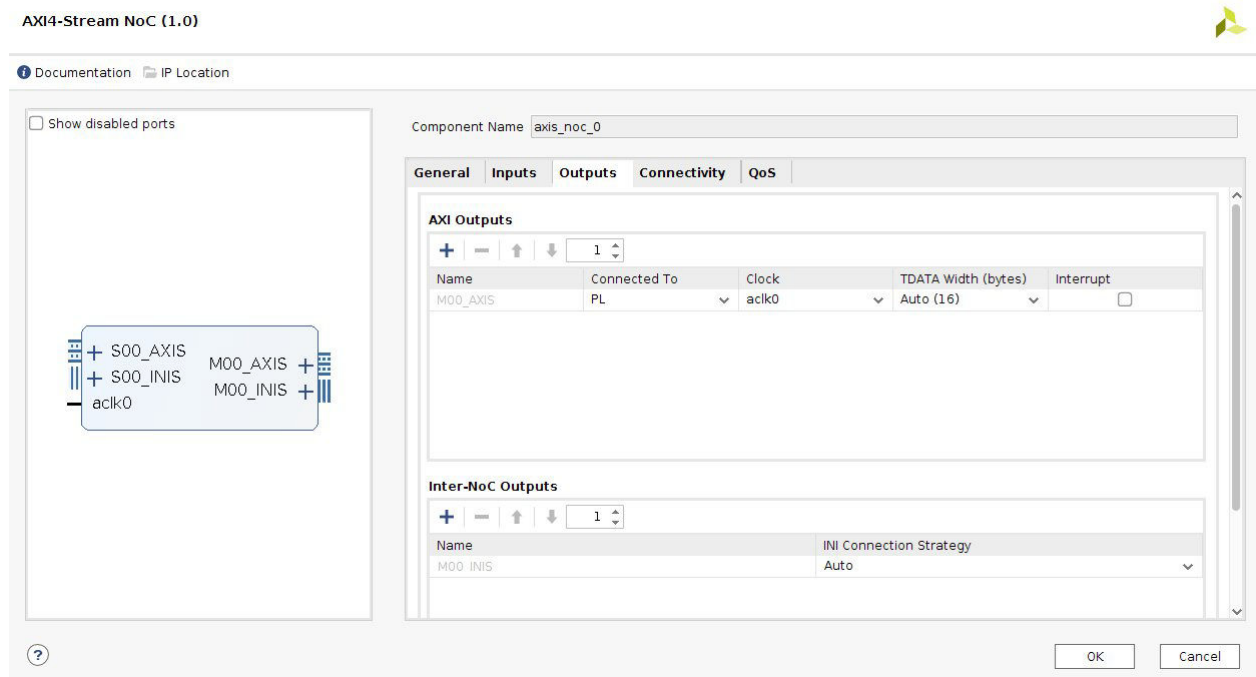
You can optionally set the INI Connection Strategy as follows:

- **Auto:** Allows IP integrator to determine the correct strategy. This is the default.
- **Single Driver:** Has a single driver and possibly multiple loads.
- **Single Load:** Has a single load and possibly multiple drivers.

## Outputs Tab

The Outputs tab of the Customize IP dialog box is shown in the following figure and allows the configuration of the output ports and associated clocks.

Figure 28: Outputs Tab



- **AXI Outputs:** Set the number of AXI4-Stream outputs to this instance and configure each output. The configuration options of the Outputs tab are the same as the options on Outputs tab of the AXI NoC.
- **Inter-NoC Outputs:** Set the number of INI output ports to other NoCs.

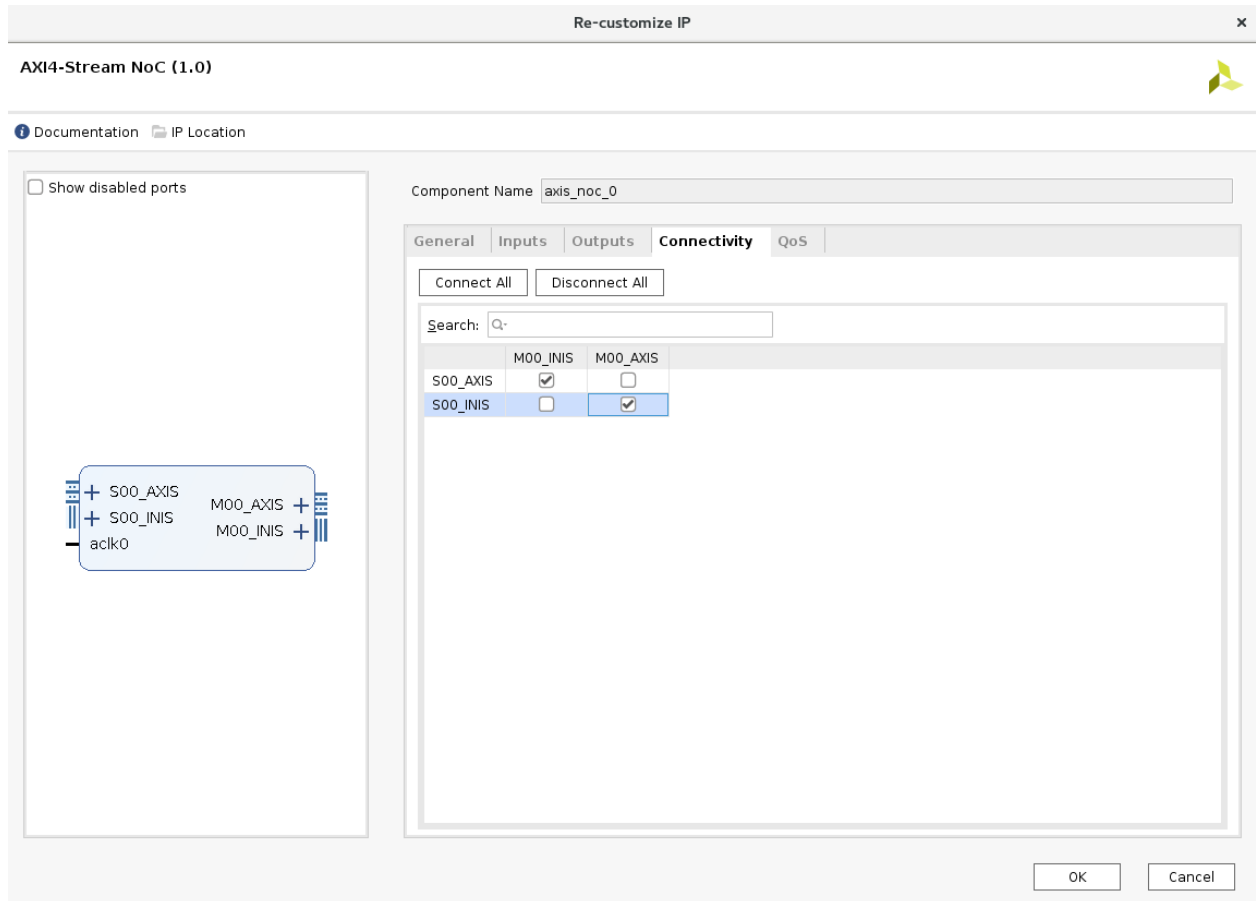
You might optionally set the INI Connection Strategy as follows:

- **Auto:** Allows IP integrator to determine the correct strategy. This is the default.
- **Single Driver:** Has a single driver and possibly multiple loads.
- **Single Load:** Has a single load and possibly multiple drivers.

## Connectivity Tab

The Connectivity tab of the Customize IP dialog box is shown in the following figure.

Figure 29: Connectivity Tab



The Connectivity tab is used to define the connectivity through the AXIS NoC. Connectivity is captured in the form of a matrix, as shown in the previous figure. The rows of the matrix correspond to the inputs of the `axis_noc` instance, the columns correspond to the outputs. A check in the box at the intersection of a row and a column indicates a connection from the row input to the column output.

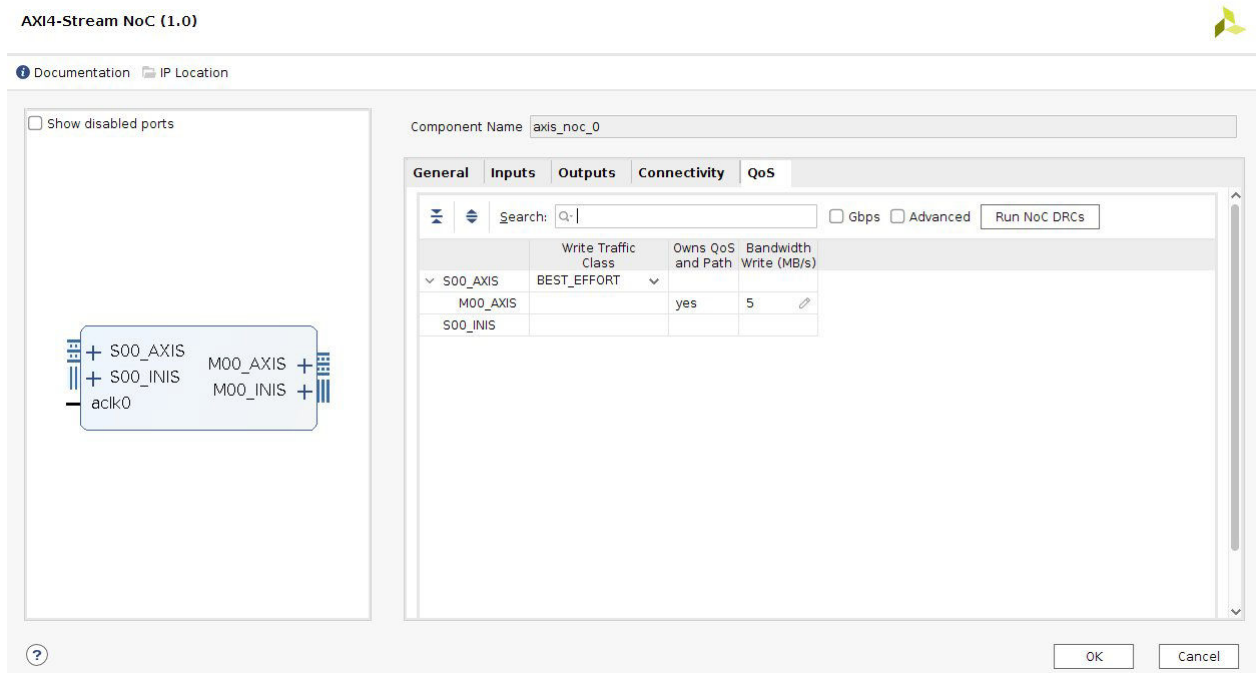
The previous figure shows that Input `S00_AXIS` is connected to output `M00_INI`, and inter-NoC input `S00_INI` is connected to AXIS output `M00_AXIS`.

**Note:** In this release, multiple connections from a single input are not supported.

## QoS Tab

The Quality of Service (QoS) tab of the Customize IP dialog box is shown in the following figure. This tab allows the configuration of the quality of service properties of every stream path defined in the `axis_noc` instance.

Figure 30: QoS Tab



The left column of the QoS tab shows a tree structure of the connectivity defined in the Connectivity tab. The top of each tree (left aligned port names) are the NoC ingress ports; in the figure `S00_AXIS`. Shown under each ingress port are the associated rows for each connected egress port.

The second column defines the write traffic class. The possible values are `LOW_LATENCY`, `ISOCRONOUS`, and `BEST_EFFORT` (default). The traffic class applies to all connections originating from the given ingress port.

The third column indicates whether this NoC instance owns the QoS setting for a given path. A NoC path may go through multiple NoC instances using INI. QoS settings are taken from the NoC instance that owns the QoS and path. QoS is ignored when a NoC instance does not own the path. Ownership is at the transition from an NMU or `strategy=Driver` to an NSU or `strategy=Load`. A value of 'pending' means the ownership will be calculated during validate, or by clicking the **Run NoC DRCs** button above. For a value of 'error', validate or click the **Run NoC DRCs** button above to see error details in the Tcl Console and messages window.

The fourth column defines the Write Bandwidth expressed in MB/s or Gb/s. Allowed values range from 0 (no write traffic is accepted) to the maximum data bandwidth of the NoC physical channel.

## Placement of NoC NMU/NSU

In order to meet QoS requirements the NoC compiler is invoked during IP integration. For further placement optimization, you can provide physical location constraints as detailed in the “Assigning Locations to NoC Endpoints” section of this [tutorial](#).

# NoC and Memory Controller Simulation

NoC and Integrated Memory Controller simulation support is provided with behavioral models in either System Verilog (RTL in GUI) or SystemC (TLM in GUI). The simulation time with SystemC model is much faster but less accurate compared to the System Verilog model. While both the SystemC and System Verilog models can be used to verify functionality, the System Verilog model should be used for performance analysis. Performance includes both bandwidth and latency. Performance analysis using the System Verilog model is within +/- 5% of hardware. All supported memory densities can be simulated using the System Verilog model.

The System Verilog model is a behavioral model developed for performance analysis. The following features have minimal impact on performance and are hence not modeled:

- Calibration algorithm
- Memory initialization sequence – Model gets Mode Register values through parameters
- ECC - Check Bit calculation is unsupported, however performance impact due to Check Bit calculation and Read Modify Write (RMW) is modeled
- Initializing DRAM with data patterns
- ECC Poisoning
- Scrubbing – Performance impact is insignificant because it is a background activity
- 2T timing – Performance impact is insignificant
- DRAM Command/Address Parity – Retry resulting from Command/Address parity error not modeled
- Write/Read DBI – Performance impact is modeled
- Exclusive transactions – All transactions treated equally by the model
- Programmable preamble and postamble for read and write
- Self-Refresh, User Refresh
- Page closed policy
- CA Mirror



## RTL versus SystemC Models

The Vivado® tool suite supports two simulation models for the NoC: a SystemC transaction level model, (tlm), and a System Verilog register transfer model, (rtl). The tlm model is fast and efficient while the rtl model is near cycle-accurate (typically within 5% of hardware).

Because the NoC routing solution is compiled from an aggregation of the traffic specifications of all of the NoC instances in the design, a single simulation model is constructed to represent the NoC as configured by the compiler. It is not possible to mix rtl simulation for some NoC instances and tlm for others. If there is a mismatch in simulation models an error will be issued during `validate_bd_design`.

The simulation model selection can be made by setting the `SELECTED_SIM_MODEL` property on each of the `axi_noc` and `axis_noc` instances. The possible values of `SELECTED_SIM_MODEL` are `rtl` (the default) and `tlm`. On the IP integrator canvas select each `axi_noc` instance and set the `SELECTED_SIM_MODEL` property in the Block Properties menu. From Tcl issue the following command.

```
set_property SELECTED_SIM_MODEL tlm [get_bd_cells /axi-noc_0]
```

## The noc\_sim\_wrapper

When netlisting for rtl simulation the constructed model is split across different components of the design. The NoC endpoints (for example NMUs, NSUs, DDRMCs) are instantiated in the containing `axi_noc` and `axis_noc` instances, while a new block called "xlnoc" is constructed to contain the set of connection components (for example NPSs, NIDBs) required to implement the NoC routing solution.

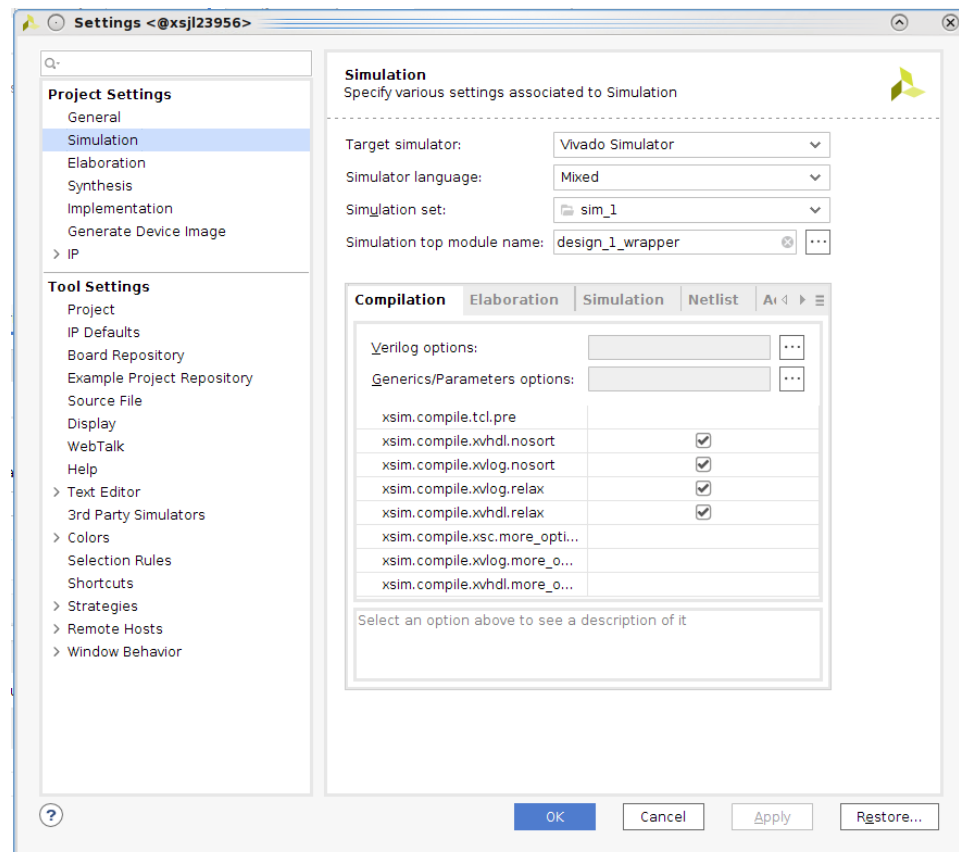
Vivado creates a new top-level wrapper file called `<design>_wrapper_sim_wrapper.v` to stitch the `xlnoc` block into the simulation netlist. The original `<design>_wrapper` is not modified but is instantiated in the new top-level along with a single instance of the constructed `xlnoc`. `<design>_wrapper_sim_wrapper` connects the ports of `xlnoc` to the corresponding NoC endpoint ports by reaching into the design hierarchy using Verilog hierarchical references.

The file `<design>_wrapper_sim_wrapper.v` is constructed in the project sources directory by the `launch_simulation` command. The **Simulation top module name** on the Simulation Settings page will be set to `<design>_wrapper_sim_wrapper`. To create the wrapper file without invoking the Vivado simulator specify the `-scripts_only` option to `launch_simulation`.

# Simulation Settings

In the Flow Navigator click **Simulation** → **Simulation Settings**. This opens the Project Settings menu on the Simulation tab as shown.

Figure 31: Simulation Settings



Set the **Target simulator** to **Vivado Simulator**.

**Note:** The wrapper created above is now the Simulation top module name.

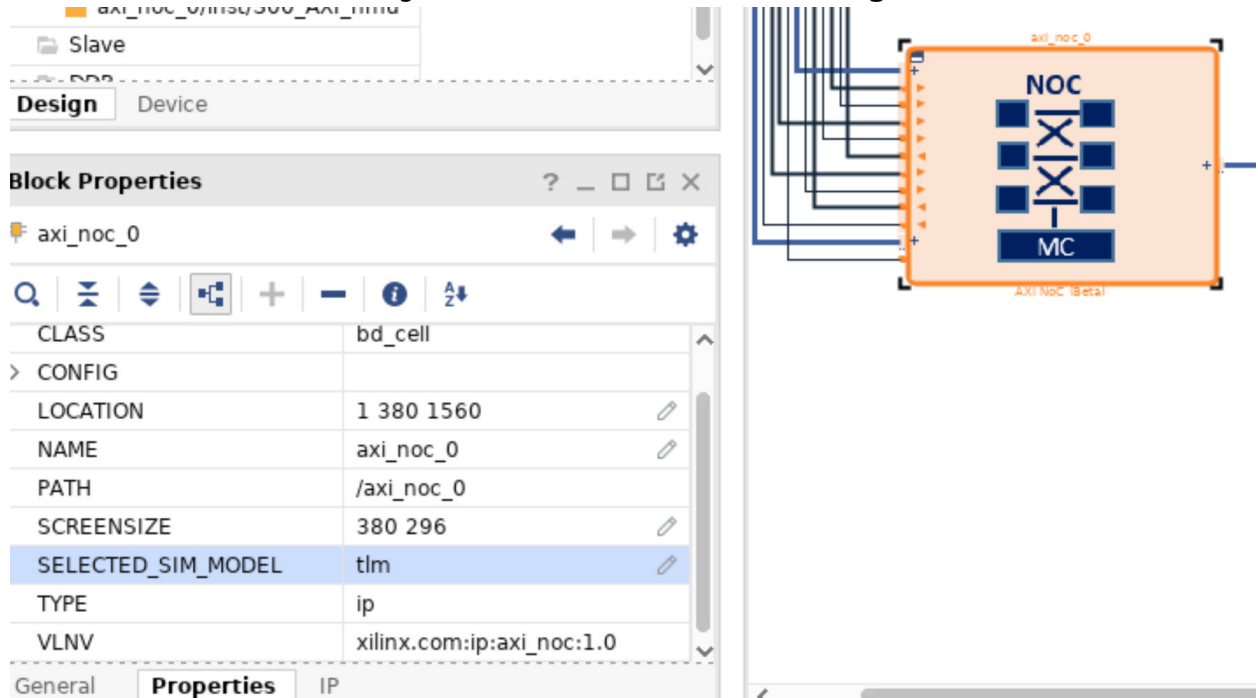
## SystemC Simulation of the NoC

By default, Run Simulation uses a System Verilog behavioral model of the NoC and DDRMC. Vivado® has the ability to use a much faster SystemC model of the NoC and DDRMC. To enable SystemC modeling, every instance of the AXI NoC and AXIS NoC needs to have the property `SELECTED_SIM_MODEL` set to `tlm`. The following figure shows changing the selected sim model to System C by setting it to `tlm` (by default, the value is set to `rtl`). Note that the SystemC model is much faster but less accurate compared to the System Verilog model. The SystemC model can be used to verify functionality, but for modeling performance System Verilog should be used.

The Tcl command to select a simulation model is as follows. Set it to `tlm` for SystemC or `rtl` for System Verilog.

```
set_property SELECTED_SIM_MODEL tlm [get_bd_cells/axi_noc_0]
```

Figure 32: Simulation Model Setting



**Note:** TLM model supports simulation only with Internal Responder.

---

## Simulating the Design

Details on building a design and running simulation are provide in the tutorial available on [GitHub](#).

## NoC Performance Tuning

This chapter discusses the key performance measures of bandwidth, latency, and system design trade-offs affecting performance, and how to optimize performance of the NoC and the integrated DDR Memory Controllers.



**RECOMMENDED:** For guidance on NoC/DDRMC performance tuning, refer to the [Performance Tuning Tutorial](#).

## Performance Metrics

As previously discussed in the Quality of Service section, every connection through the NoC has an associated QoS requirement consisting of three components: traffic class; read and write bandwidth; and read and write latency. The NoC compiler combines the requirements of all connections through the NoC to arrive at an optimal NoC configuration. Various factors combine to limit the maximum achievable bandwidth and minimum achievable latency through the NoC.

### Related Information

[Quality of Service](#)

## Physical Link (Raw) Bandwidth

At the most basic level, each physical link through the NoC is limited by the raw possible bandwidth of a physical link, ignoring packetization overheads. Read responses have no packet header overheads and can achieve this bandwidth. Write requests have a header flit and so need some derating.

The following table shows the maximum operating frequency of the NoC packet transport for each Versal® speed grade.

*Table 14: NoC Clock Frequency per Speed Grade*

Speed Grade	-1LP	-1MP	-2LP	-2MP	-2HP	-3HP
NoC Frequency (MHz)	960	960	1000	1000	1080	1080

The following table shows the physical link bandwidth before any packetization overheads are considered.

Table 15: Raw Link Physical Bandwidth

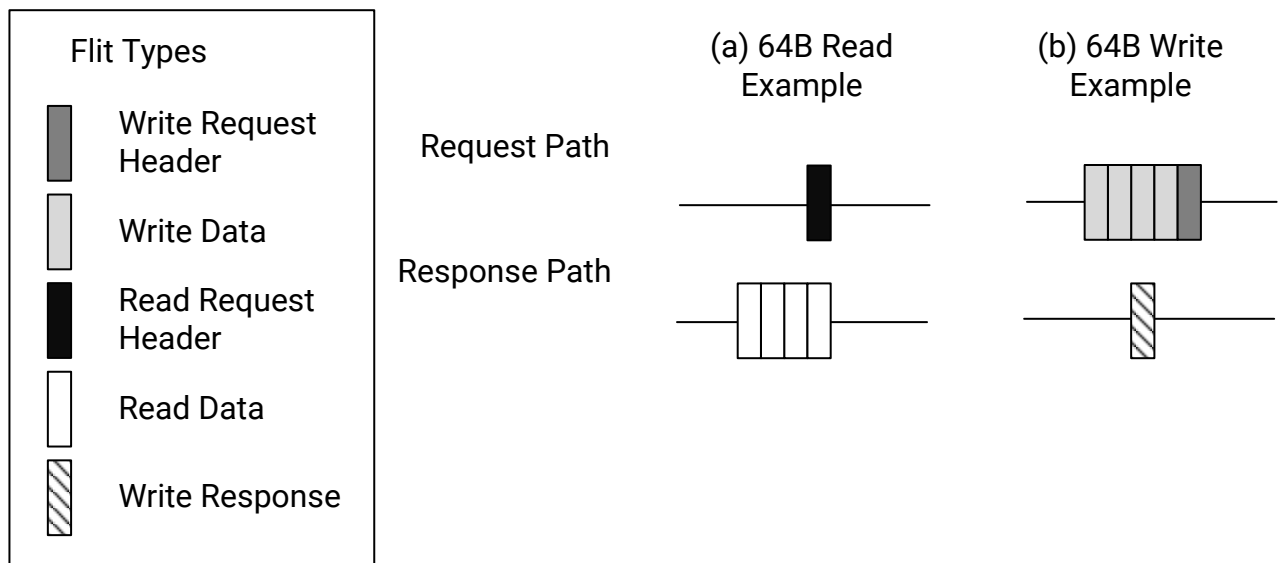
NoC Clock Frequency MHz	Read BW GBytes/sec 256 Byte Transfer	Write BW GBytes/sec 256 Byte Transfer	Read BW GBytes/sec 64 Byte Transfer	Write BW GBytes/sec 64 Byte Transfer
960	15.36	14.46	15.36	12.29
1000	16.0	15.06	16.0	12.8
1080	17.28	16.26	17.28	13.82

## Packetization Overhead

The amount of data that can be moved through the NoC data paths is affected by packetization overhead. Some flits transport data while others carry protocol related information such as the transaction address or the packet response.

The following figure shows the packet types and the overhead incurred for each type. The NoC packet domain data paths are all 128 bits (16 bytes) wide. Each read request consumes one header flit in the request path and  $n$  data flits in the response path. This is shown in (a). A write request has one header flit and  $n$  data flits. For example, a 64-byte transfer that is aligned to the burst start consumes five data path flits (one header flit and four data flits). The write response consumes one flit in the response path. This is shown in (b).

Figure 33: Packet Flit Overheads



X23209-100419

If the AXI request is less than the chop size (minimum of 256 bytes or interleave granularity in the case of interleaved memory channels), the transfer ends early and the unused flits are available for another command.

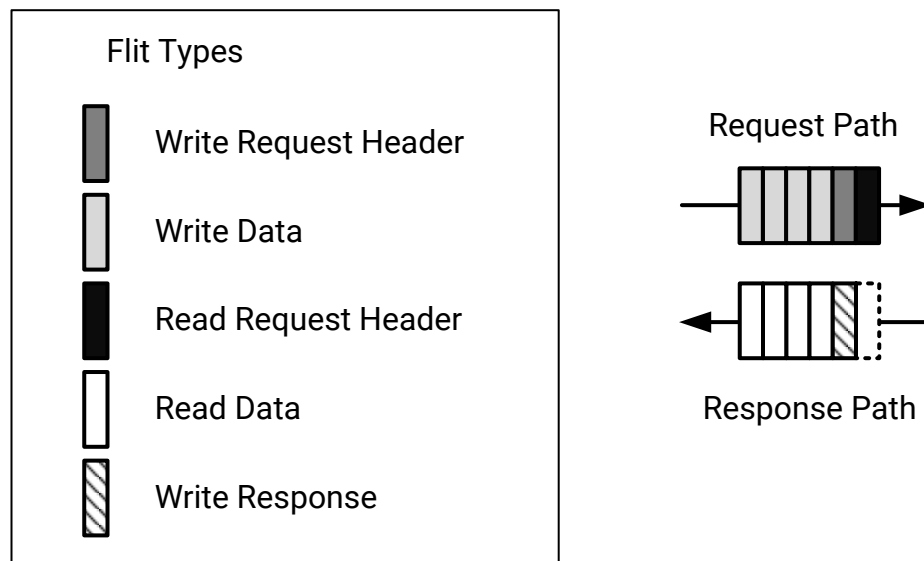
A data path with bandwidth  $X$  GBytes/sec of raw bandwidth will have  $(n/n+1) X$  Gbytes/sec of bandwidth where  $n$  is the burst length. Burst length is set in the QoS tab of the NoC GUI under advanced features. To minimize overhead, select a burst length such that (burst length  $\times$  data width in Bytes = chop size).

Each write request generates one response flit. Note that the NoC NMU has a default chop size of 256 bytes, or interleave granularity if smaller. A write request may get chopped into multiple write bursts and each one will have a write response. The NMU handles coalescing the responses so the original AXI write bursts only see one write response.

For a full duplex NoC link sending a mix of read and write requests in one direction and receiving responses, extra flits will be generated which will degrade the peak bandwidth.

The following figure shows an example. The request and response paths of one physical link are shown, with the order of flits sent over each physical link in each direction. This example shows 64B packets with a 50/50 read/write mix. For this traffic mix and packet size there will be groups of six flits in the request path. These six flits are one read request header flit, one write request header flit, and four write data flits. In the response path there will be four read data response flits and one write response flit. There is also one dead cycle, since this traffic mix involves a 50/50 mix between read and write data (that is, the read and write bandwidth are the same. With this mix there will be five response flits for every six request flits and the request path is the bottleneck (the response path has some spare capacity).

Figure 34: Link Utilization for Mixed Read/Write Traffic



X23208-092419

**Note:** The NoC compiler does not necessarily share the same physical link when routing read and write traffic from one master. The following examples illustrate bandwidth achievable when read and write traffic is mixed on the same physical link. The NoC compiler factors flit overheads when calculating available bandwidth. This section is provided to help understand possible bottlenecks when contention occurs.

The following table provides the expected peak bandwidths for a given read/write traffic mix. These examples are for a 1 GHz NoC clock frequency and the numbers scale linearly with the NoC clock.

**Table 16: Read/Write Bandwidth for Mixed Traffic at a 1 GHz NoC Clock Frequency**

Bytes per Transaction/ Frequency	0/100 (%Rd/%Wr)	30/70	50/50	70/30	100/0
32B@1000 MHz	0/10.67	4.0/9.33	8.0/8.0	13.17/5.65	16.0/0
64B@1000 MHz	0/12.8	5.05/11.79	10.67/10.67	14.45/6.19	16.0/0
256B@1000 MHz	0/15.06	6.30/14.69	14.22/14.22	15.58/6.68	16.0/0

## Latency

The latency through the NoC comprises two components:

- **Structural latency:** This is ‘best case latency’ which is the time taken for packets to progress through the pipeline stages inside the NoC.
- **Queuing latency:** This is the delay caused by queuing in the network. If a flit from a packet is placed in a queue, the flits ahead of it need to drain out before the flit will be serviced.

It is important to understand the role of queuing latency. Depending on the workload sometimes queuing latency has a much bigger impact on the average transaction latency than structural latency, especially in a heavily loaded system.

## Throughput, Latency, and Outstanding Transactions

Assuming masters and slaves are capable of issuing and accepting multiple transactions, there is a relationship between throughput, latency, and outstanding transactions (OTs).



Consider the following example. Suppose you wish to achieve maximum throughput in a single-clock-domain system, using single-cycle requests. You need to issue a single-cycle request every cycle. Since response latency is non-zero, you must have multiple OTs. In fact, the number of OTs required is a function of the slave's latency. If the slave's latency is less than or equal to the master's maximum OTs, full throughput is possible. However, if the slave's latency is longer, the master will reach maximum OT and stop issuing requests, and it will be able to issue new requests only in exchange for arriving responses. In such a case, the request 'pipeline' has gaps and the achieved bandwidth is lower than the maximum by an amount proportional to the slave's latency.

The OT limits are as follows:

- **NMU:** 64 outstanding writes (up to 256 bytes), and up to 64 outstanding reads of 32 bytes each. The read reorder buffer, RROB, holds 64 32-byte entries. A read that is >32 bytes consumes multiple of these entries.
- **NSU:** 32 outstanding writes and 32 outstanding reads.
- **DDRMCMC:** 32 outstanding transactions per channel, where each transaction is a burst length  $n$  read or write where  $n=8$  for DDR4, and 16 for LPDDR4.

## AXI ID: Single or Multiple IDs?

Versal® architecture and interconnect is consistent with the AMBA® AXI standard, and as such, each transaction is accompanied by an AXI ID. The master may use a single AXI ID for all its transactions, or may issue different transactions with different IDs. Since the same AXI ID is returned with the transaction response, it can be used by the master to forward the response to the appropriate internal destination if there is more than one. An example of the use of the AXI ID within a complex master is an 8-channel DMA. For data transfers eight different AXI IDs are used, one per channel, and an additional eight AXI IDs are used, one per channel, for the descriptor access. By inspecting the AXI ID in the response, the DMA logic knows how to handle the arriving response.

An important aspect of AXI ID is that transactions which have the same AXI ID must be returned in the same order they were issued, while transactions with different IDs may be returned in any order. Suppose the master is a simple function with a single thread of data access. Using a single ID ensures that transactions are returned in the order issued. Using multiple IDs may cause transactions to be returned out of order, and may require special handling in the master to understand the sequence, and possibly reorder the arriving responses. It may therefore be preferable to use a single ID in such cases. However, in most cases better performance is achieved using multiple IDs. It is therefore advisable to assess the possible performance improvement against the extra design complexity in the master.

One exception to this rule is transactions from an NMU to an NSU into PS or PL. At the NoC exit points into the PS and PL, the NSU only supports two bits of ID. The original AXI ID undergoes a simple form of ID compression down to two bits. This is equivalent to an outstanding ID limit of four. The limited number of outstanding IDs can limit the number of OT. The two extreme examples are:

- All transactions use the same AXI ID for a maximum OT of 64. This is assuming a single master via NoC NMU).
- Each transaction uses a unique ID for a max OT of four.

These limits are independent for the read and write channels.

## The Single Slave per ID Rule

The Single Slave per ID (SSID) rule is often used by interconnect components as a way of simplifying implementation at the cost of some performance loss. The rule applies whenever:

- A master routes transactions to more than one output port (slave), and
- Multiple transactions are outstanding, and
- Transactions use the same ID

According to AXI ordering rules, transactions using the same ID must be responded to in the same order they were issued. However, under the conditions stated above, it is possible that different slaves have different latencies, and as a result, the responses may be returned out of order. To prevent that occurring, the component does the following:

- If a transaction destined to port A with ID x arrives, and there are already some pending transactions with ID x to port B, the master will wait until all pending transactions with ID x to port B have completed before routing the new request to port A.

In effect, the master ensures that any ID value can be pending only to a single slave destination, thus ensuring that responses cannot be returned out of order. Hence the name “single slave per ID”. In some cases, particularly with interleaved memories, this can have a negative impact on performance.

---

## System Design Considerations

There are many system design trade-offs that impact NoC performance. This section explains many of those options and their effect on performance.

## DDR4 Component Choice

DDR4 DRAM components are available in 3 variants: x4, x8, and x16. The choice of component is often determined by factors such as cost, power, board space, etc. Here, only the performance impact is considered. The following table compares various features of the DRAM components and their performance impact.

**Note:** Data shown is for 8 Gb components, 3200 speed grade.

**Table 17: DDR4 Component Choice Trade-Offs**

Feature	x4	x8	x16	Performance Implication
Page size (x64)	8 KB	8 KB	8 KB	None, all are the same.
Banks	16	16	8	Fewer banks reduces the performance of random and multi-thread linear traffic patterns.
Bank Groups	4	4	2	Fewer bank groups reduces the MC's ability to reorder and achieve "short" timing.
Data Mask	No	Yes	Yes	Without data mask, any partial write requires a read-modify-write operation, reducing overall performance.
tFAW	13 ns	21 ns	30 ns	Shorter tFAW improves performance of short random traffic pattern.
tRRD_L	4.9 ns	4.9 ns	6.4 ns	Shorter parameter improves overall performance.
tRRD_S	3.3 ns	3.3 ns	5.3 ns	Shorter parameter improves overall performance.

## Multi-Rank and 3DS DRAM Configurations

Multi-rank and 3DS DRAM configurations primarily provide more storage, but may also provide performance benefits in some use cases. For example, a short-transaction random-access traffic pattern is often performance-limited by the number of DRAM banks or tFAW. By using multiple physical or logical ranks it is possible to achieve better performance.

## Single x64 vs. Dual x32 DDR4 Configuration

The DDRMC can be configured in Single- or Dual-Channel mode. For a 64-bit DDR4 interface, this means the controller can support a single x64 channel, or dual x32 channels. This selection is made on the DDR Memory tab of the AXI NoC configuration dialog. This choice affects performance as follows.

**Table 18: Single- vs. Dual-Channel Trade-offs**

Feature	1 x64	2 x32	Notes
Storage	-	-	Same
Access Quantum	64 bytes	32 bytes	For some 32-byte use cases, 2 x32 can be significantly more efficient.
Read efficiency	Good	Fair	Both channels share the same read reorder buffer, causing some performance limitations.

Table 18: Single- vs. Dual-Channel Trade-offs (cont'd)

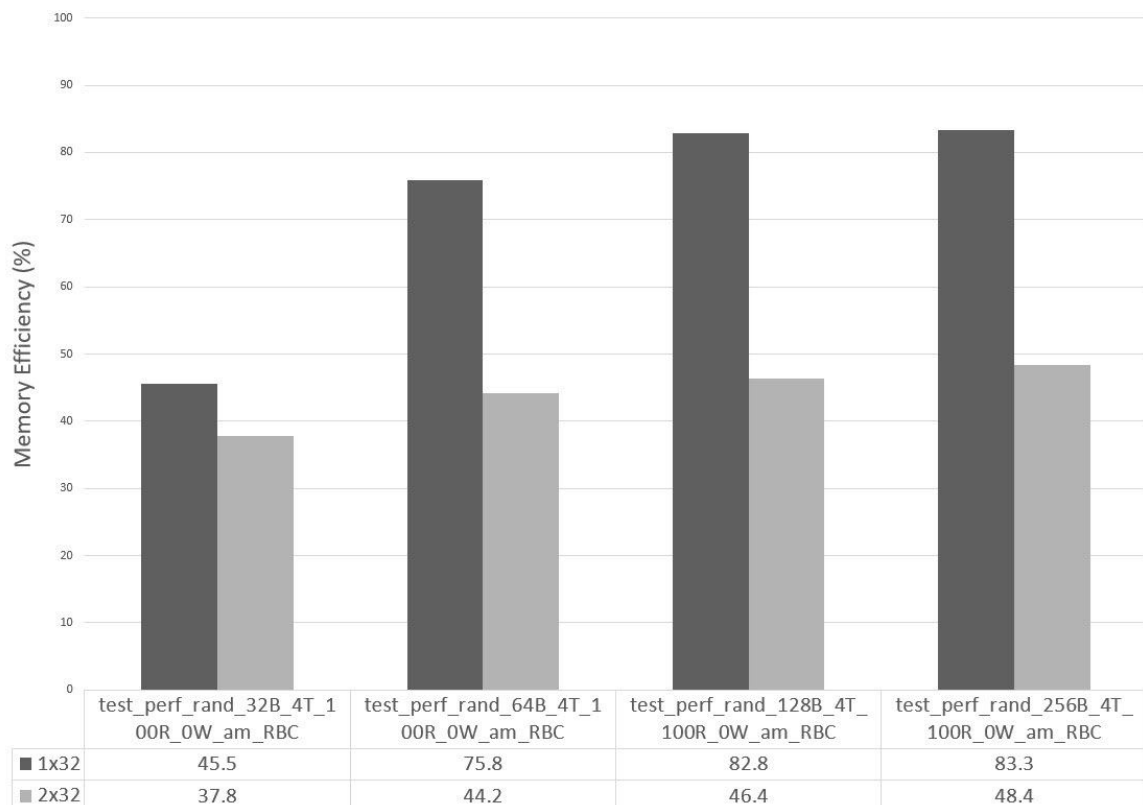
Feature	1 x64	2 x32	Notes
Write efficiency	Good	Fair	Shared resources cause some performance loss.

Likewise, an LPDDR4 controller can support a single 1x32 channel, or a dual 2x32 channel.

The next few figures show examples of read and write efficiency differences between single and dual-channels for random and linear addressing. All the data shown is for DDR4 at 3200 Mb/s or LPDDR4 at 4267 MB/s. Actual efficiency may differ from the values illustrated depending on traffic type, address mapping, data rate, and other variables.

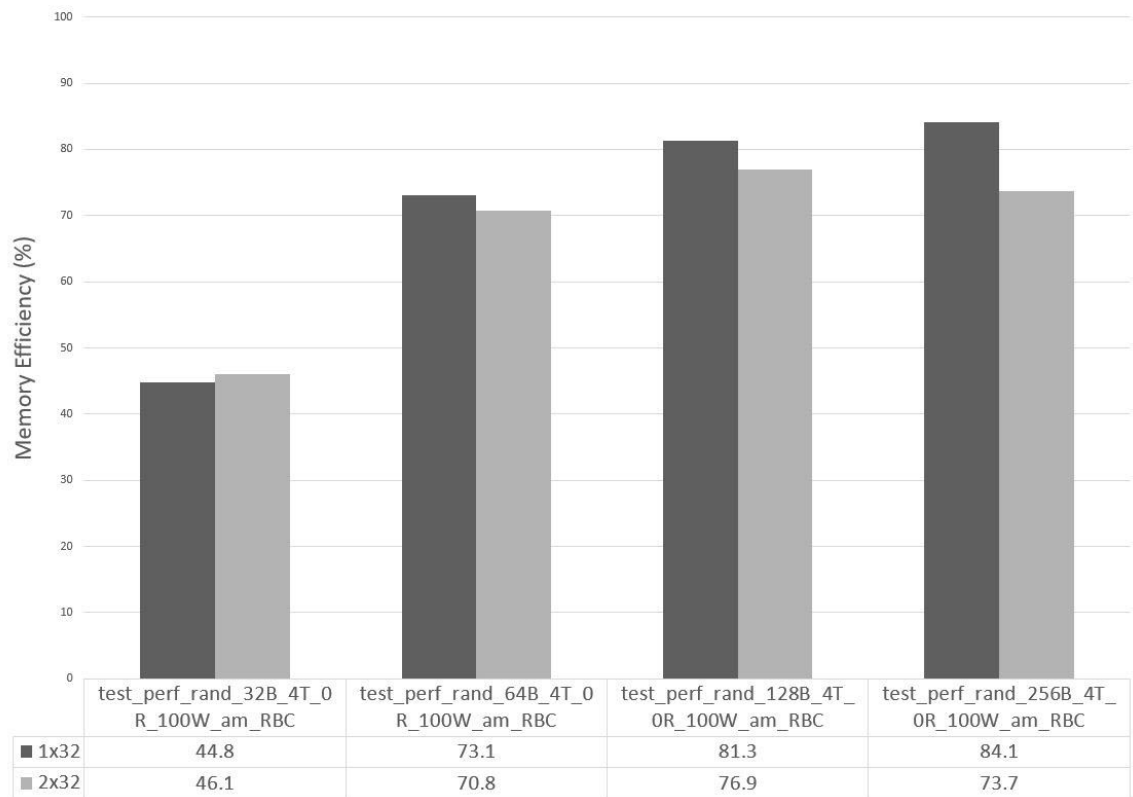
The following figure illustrates efficiency differences for DDR4 random reads. Single-channel efficiency is shown in black, while corresponding dual-channel efficiency is shown in gray. The shared read reorder buffer causes a decrease in efficiency for dual-channel compared to single-channel.

Figure 35: DDR4 Random Read Efficiency



By contrast, the following figure illustrates that for DDR4 random writes, there is only a small efficiency difference between dual-channel and single-channel operation.

Figure 36: DDR4 Random Write Efficiency



Similarly, for LPDDR4 the random read efficiency is impacted by the shared read reorder buffer, while random write efficiency is not significantly affected. This is illustrated in the following two figures.

Figure 37: LPDDR4 Random Read Efficiency

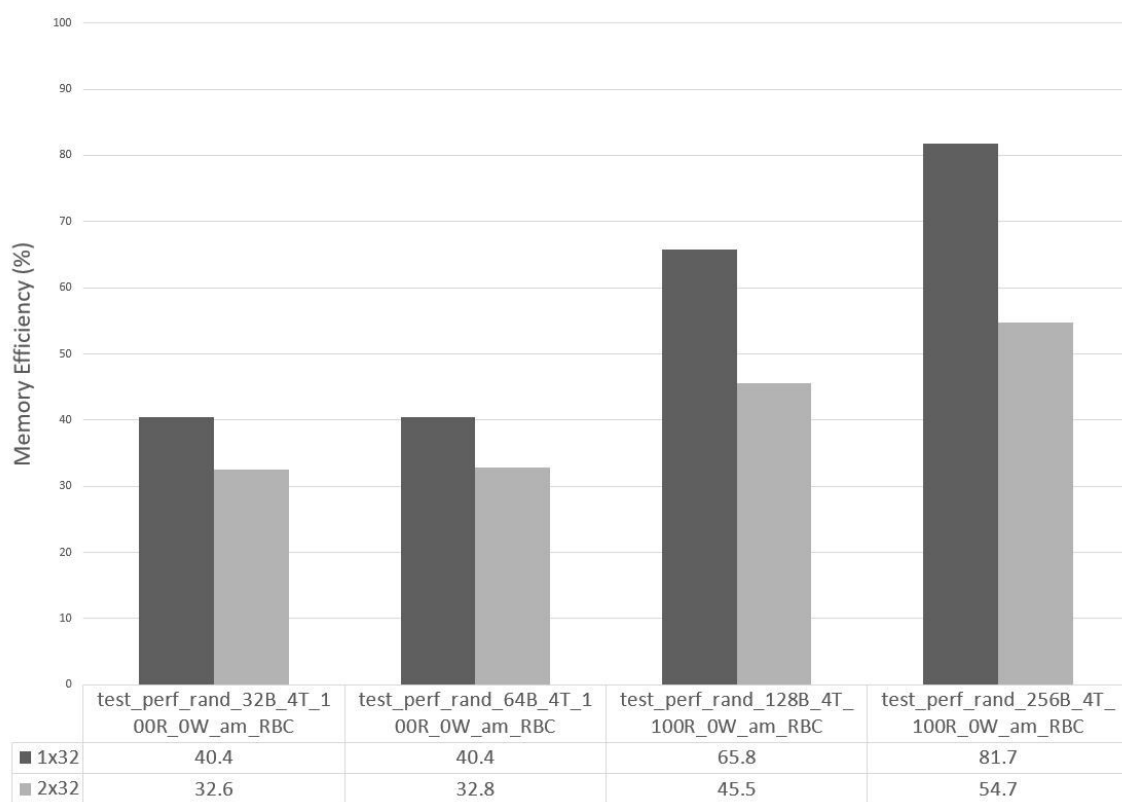
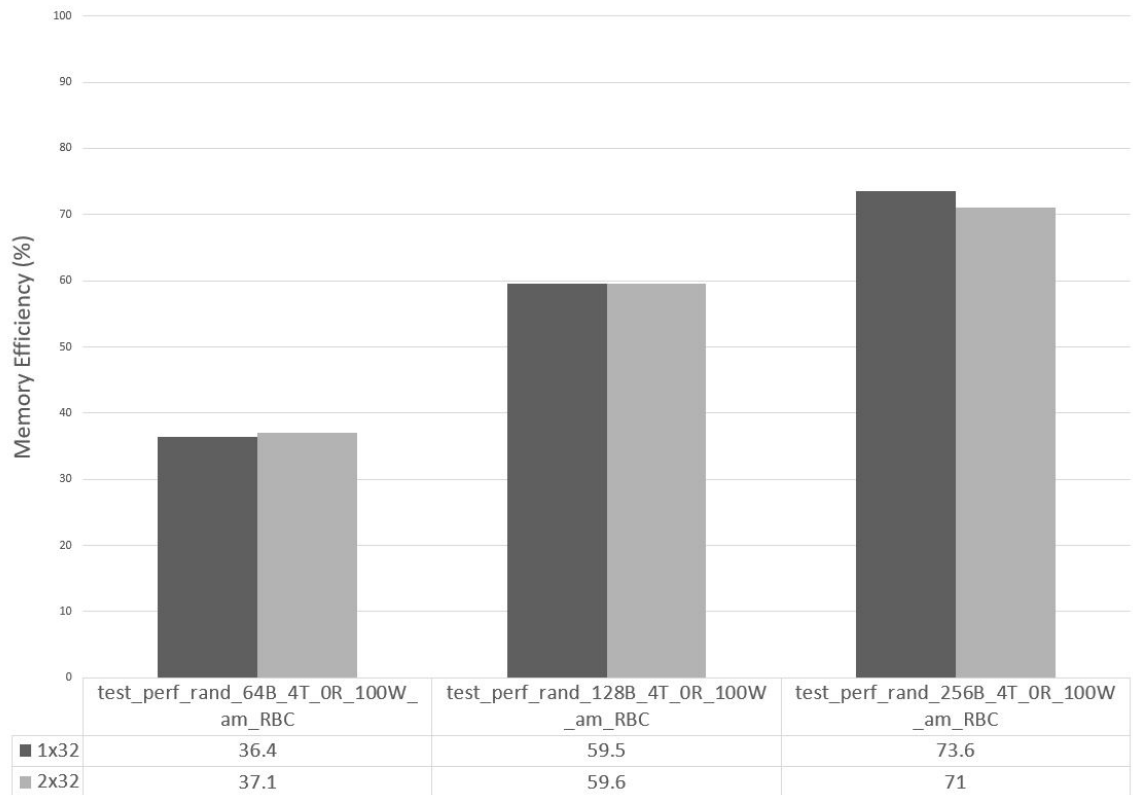
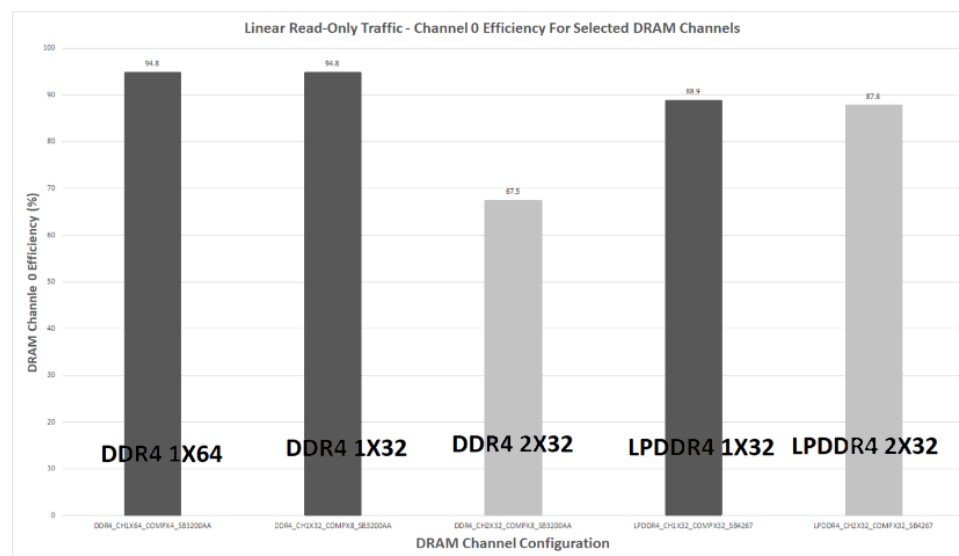


Figure 38: LPDDR4 Random Write Efficiency



Linear read traffic also shows some efficiency degradation for DDR4 dual-channel operation. This efficiency loss is less pronounced for LPDDR4. The following figure illustrates some sample efficiencies for DDR4 and LPDDR4 linear read traffic.

Figure 39: Linear Read Efficiency



## Memory Interleaving

When more than one DDR controller is available on a device, you can choose between two mapping modes:

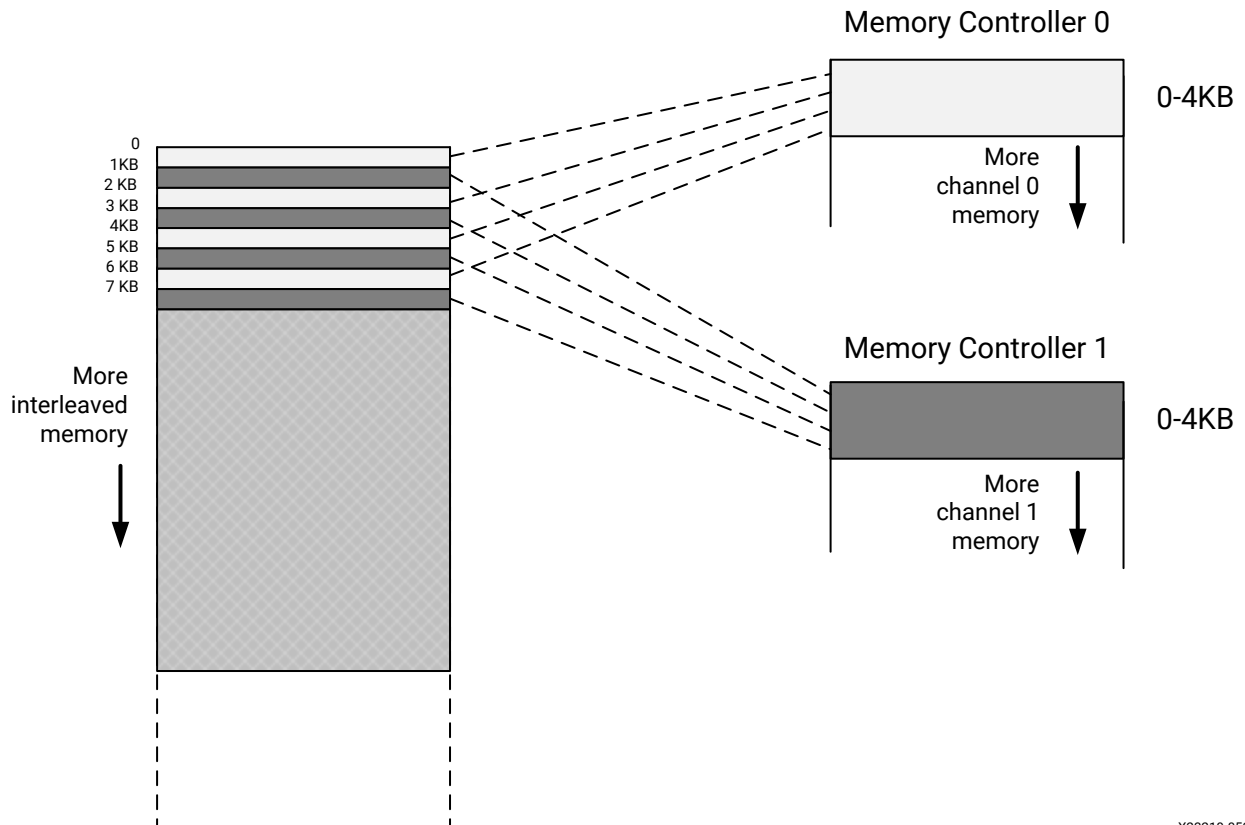
- Each DRAM controller is one contiguous address space which may be targeted independently.
- Two or four DRAM controllers may be interleaved to present a single unified address space.

Memory interleaving makes the participating memory controllers appear as one large pool of memory. The memory may still be spread across discontinuous physical regions like a single DDR controller, with a small region in 4 GB address space and a larger region somewhere higher in the address map. Memory traffic is balanced across the participating DDR controllers in hardware and software does not need to determine how to place data to effectively use more than one DDR controller.

The following figure shows an example of interleaved memory across two DDR controllers. The interleave granularity shown is 1K. The NoC manages interleaving at each NoC entry point (NMU). Interleaving is a property of a memory region. If a memory region is interleaved across two memory controllers, then half of the memory in the region goes to one controller and half goes to the other. The mapping is arranged in strided fashion such that alternate 1K regions go to different DDR controllers. If a burst transaction is sent to an NMU and crosses an interleave boundary (in this case 1K), the transaction is chopped at the interleave boundary. This ensures that a single transaction never crosses an interleave boundary.



Figure 40: Interleaved Memory across Two DDR Controllers



X23210-052120

Some benefits of independent controllers include:

- Explicit partitioning of the workload based on traffic type, function, or other considerations.
- Independent power mode switching and/or frequency changes.

Some benefits of interleaved controllers include:

- A unified address space.
- Automatic load balancing.
- 2x or 4x the bandwidth of a single 'pipe'.

## SSID Effect in Memory Interleaving

An important attribute of the NoC is that the NMU includes a 2 KB read reorder buffer and each read transaction is tagged with a unique tag which allows the transactions to be executed and returned out of order, thereby improving performance in various ways. One of those ways is avoiding the SSID effect seen previously. However, the same does not apply to write transactions. The NoC NMU does not tag or reorder writes, and the SSID rule is applied. Since MC interleaving involves sending transactions to two or four slaves concurrently, SSID may become a factor limiting write performance if transactions share the same ID or a limited number of IDs.

## Short Transactions: Read vs. Write

When a read transaction size is smaller than the access quantum of the DRAM, bandwidth is always wasted. For example, a 32-byte read request to a 1x64 DDR4 DRAM, will always read 64 bytes and discard half the data, imposing a 50% efficiency reduction on top of the normal DRAM efficiency.

When a write transaction size is smaller than the access quantum of the DRAM, two possibilities exist. The first is behavior similar to the read case where 50% of the efficiency is lost. The second possibility is that while the transaction is pending in the queue, a subsequent transaction to the next address will merge with the current transaction to make it a whole access quantum, thereby losing no efficiency. This behavior is attributed to the “write combine” feature of the DDRMC. Thus, linear access writes will be efficient even when using 32B access in a 1x64 configuration, but random access writes will not.

## Transaction Size

The choice of transaction size may affect performance. Comparing transaction sizes of 64, 128, and 256 bytes (1, 2, and 4 access quanta for a x64 DDR4 or x32 LPDDR4), the key things to consider are outstanding transactions (OT) and access locality.

- For linear or random block access, transaction size does not directly impact performance but may indirectly impact it due to OT limitations. Therefore, longer transactions are preferred in these cases.
- For random access, short transactions suffer the most DRAM access overhead, while longer transactions benefit from some locality and lower average overhead. Therefore, longer transactions will typically see higher efficiency.

## LPDDR4 Refresh Options

The DDRMC supports two DRAM refresh options for LPDDR4:

- All-bank refresh
- Per-bank Refresh

When all-bank refresh is used, all banks must be precharged before a refresh is issued. This means all banks are unavailable for the duration of the refresh operation (tRFC, e.g., 280 ns for 4267 speed). All-bank refresh makes the entire DRAM unavailable for about 7% of the time.

When per-bank refresh is used, each bank is issued a separate refresh command. This means while one bank is refreshed, the other banks are available. The duration of the per-bank refresh is shorter (tRFCpb, e.g., 140 ns for 4267 speed). Per-bank refresh makes each bank unavailable for about 3.5% of the time.

Using per-bank refresh may reduce and even eliminate the performance loss due to refresh. However, results heavily depend on traffic pattern and address mapping, and may in some cases be worse than all-bank refresh.

## DRAM Addressing

Unlike static RAM (SRAM) in which access to any location is treated equally, DRAM has a specific structure which makes access to different parts incur different overhead. The structure of a typical DRAM includes rows, columns, banks, bank groups, etc.

A DRAM row (or 'page') is a block of DRAM space that shares some internal resources and has to be 'opened' before access to it is possible. A row contains many columns. A typical DRAM may contain 64K rows. Switching between rows incurs a performance penalty due to the required closing of the current row and opening the next row using 'Precharge' and 'Activate' DRAM commands, and the mandatory minimum allowed time interval between such commands and the next read or write operation.

A DRAM column is a single addressable memory location. A typical DRAM row contains 1024 columns.

A DRAM bank is a group of rows. Within each bank, only a single row can be 'open' at any time, and switching between rows is costly. By having multiple banks, multiple rows can be open simultaneously, and switching between rows in different banks can be very efficient. Precharge and Activate commands to different banks can be scheduled concurrently. Typical DDR4 components have 16 banks.

As components got faster and more sophisticated, the concept of bank groups was added (for example in DDR4 and HBM). For example, a DDR4 DRAM has 16 banks arranged in 4 bank groups. A new restriction was added: when switching between banks belonging to *different* bank groups, there is no performance penalty, but when switching between banks belonging to *the same* bank group, there is a mandatory wait time of a few clock cycles, leading to lower performance.

Other DRAM partitions such as rank, logical rank (in 3DS), and HBM stack ID (SID) also have some performance implications similar to banks and bank groups.

## DRAM Address Mapping

DRAM address mapping is the way the system physical address (for example a 32-bit byte address delivered via AMBA AXI bus) is mapped to the physical DRAM address bits which are partitioned into rows, columns, banks, and bank groups, etc. The address mapping is usually programmable in the DRAM controller and is selected at initialization time. There is no '*one-size-fits-all*' mapping. Different mappings will maximize performance of different workloads or traffic patterns.

Factors affecting choice of mapping include:

- Address progression: linear, random, rectangular, etc.
- Transaction size (64, 128, 256 bytes, etc.)
- Number of threads (that is, the number of individual devices accessing memory)
- Multi-thread memory space partitioning

It should be noted that it is possible to pick a default address mapping that is a '*middle-of-the-road*' solution which provides reasonable performance. But for specific use cases that have well-defined traffic patterns and demand for the highest possible performance, the correct choice is crucial.

### Address Mapping Notation

The notation system used in this document is explained in the following example.

A certain DRAM component has the following properties:

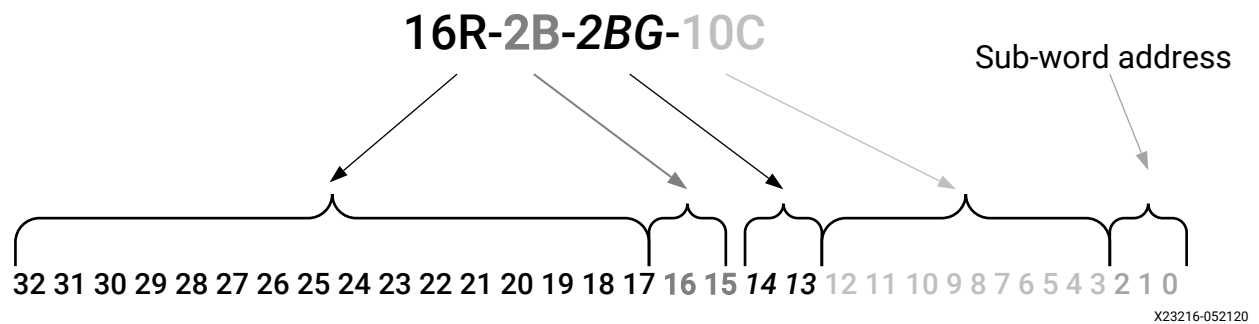
- 16 row address bits (64K rows per bank)
- 10 column address bits (1024 columns per row)
- 2 bank address bits (4 banks per bank group)
- 2 bank group address bits (4 bank groups)

The DRAM bus width is 64 bits. The basic DRAM access unit is 64 bytes. The desired address mapping is:

- System\_addr[2:0] is not used by the DRAM due to the 64-bit width
- Column\_addr[9:0] = system\_addr[12:3]
- Bank\_group\_addr[1:0] = system\_addr[14:13]
- Bank\_addr[2:0] = system\_addr[16:15]
- Row\_addr[15:0] = system\_addr[32:17]

The notation for this mapping is: 16R-2B-2BG-10C.

Figure 41: Example Address Mapping



## Examples of Traffic Patterns and Choice of Mapping

Performance results in the following examples are based on modeling with a python model.

Results are for 64-bit DDR4-3200.

### Single-Thread Linear Read

A single device accesses the entire memory linearly. Access within a page (row) is very efficient, but switching pages can cause loss of efficiency. To minimize the loss, choose a mapping that will switch banks at page boundaries and, furthermore, alternate between different bank groups after each 64-byte DRAM command.

The chosen mapping is therefore: 16R-2B-1BG-7C-1BG-3C.

This mapping places a bank group bit after three column bits. This ensures that back-to-back DRAM commands target different bank groups and avoid same-group penalty. For comparison, the following table shows performance with other possible mappings.

Table 19: Single-Thread Linear Read Address Mapping

Mapping	Efficiency	Mapping Type
16R-2B-1BG-7C-1BG-3C	95%	Row-bank-column with bank group optimization
16R-2B-2BG-10C	54%	Simple row-bank-column

## Multi-Thread Linear Read/Write

This example uses two read devices and two write devices, all operating concurrently, and each linearly accessing a different memory region. Similar to the previous example, it is beneficial for each device to open a page and access it linearly without page miss interruptions. To achieve this, 'place' each device in a separate bank, while keeping one bank group bit in the low order bits as before to avoid same-group penalty.

To achieve this, decide how to partition the memory between the four devices. For example, allocate 32 MB for each device. To ensure that each device is accessing a different bank, place the bank bits at the position corresponding to 32 MB boundary: 6R-2B-1BG-11R-7C-1BG-3C.

Table 20: Multi-Thread Linear Read/Write Address Mapping

Mapping	Efficiency	Mapping Type
6R-2B-1BG-11R-7C-1BG-3C	88%	32 MB multi-thread
16R-2B-1BG-7C-1BG-3C	40%	Row-bank-column with bank group optimization
16R-2B-2BG-10C	29%	Simple row-bank-column

This approach works well with up to eight devices because each device occupies two banks. With more than eight devices, some performance degradation is expected. Note that the 32MB choice is just an example. Another possibility is to place the three bank bits as most-significant bits, thus partitioning the entire DRAM space into eight equal regions.

## Random Addressing

Recall that any system transaction larger than the DRAM's basic access unit (typically 64 bytes) is chopped into individual DRAM commands. If addressing is random and transaction size is equal to the DRAM basic access unit, then no address mapping has an advantage. The DRAM controller will reorder transactions to hide page access overhead as much as possible. Expected efficiency is around 40%.

Longer transactions create some short-term linear progression between commands and can benefit from mappings that are similar to the single-thread linear case. Below are results for 128-byte transactions.

**Table 21: Random Traffic Address Mapping**

Mapping	Efficiency	Mapping Type
16R-2B-1BG-7C-1BG-3C	78%	Row-bank-column with bank group optimization
16R-2B-2BG-10C	63%	Simple row-bank-column

The ‘Random Block’ addressing is a generalization of random addressing. Once a random address is chosen, an entire block is read linearly. For example, in a badly fragmented virtual address system, linear access in the virtual domain may look like random block addressing in the physical domain, with block size equal to the address translation table.

## Video Decompression

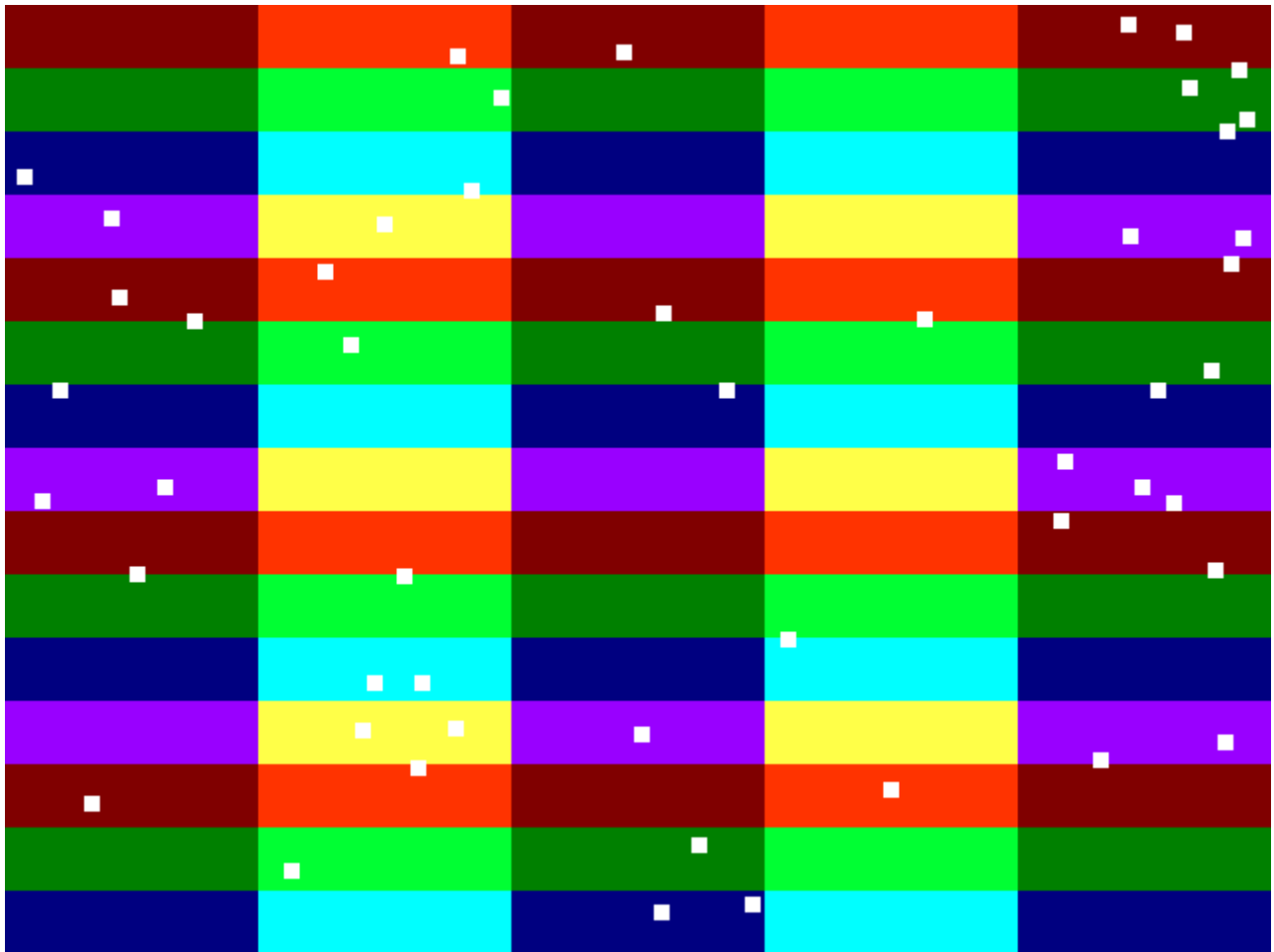
Video compression standards often use block-based motion estimation. To decode a 16x16 pixel block in the current frame, the decoder may receive a motion vector relative to a past or future reference frame from which the best approximation of the current block can be fetched. As a result the majority of DRAM access done by a video decoder is in the form of rectangular block fetch from an image frame buffer, with arbitrary offset or alignment. The result depends on the speed and direction of the motion in the video.

To maximize efficiency of video decode traffic, the mapping should shape DRAM pages into rectangular image regions rather than ‘raster’ image lines. An optimal mapping has the following properties:

- DRAM pages are shaped into rectangular image regions, referred to as tiles.
- Each tile is large enough to maximize the probability of a block fetch being fully contained within the tile.
- Adjacent tiles in any image direction (left, right, top, bottom) belong to alternate bank groups.

The following figure illustrates video decode frame buffer tiling. In the buffer, 50 8x8 blocks are randomly placed to model a video decoder motion-compensated block fetch.

Figure 42: Video Decode Frame Buffer Tiling



It can be seen that most blocks fall within a single tile and therefore exhibit very efficient DRAM access. Some blocks span two tiles either horizontally or vertically, and exhibit somewhat reduced efficiency. On rare occasions, a block may fall on a corner of four tiles and will exhibit yet lower efficiency.

Note that such mapping imposes restrictions on how the video frame buffer is allocated and used. Key requirements include:

- The frame buffer start address must be page-aligned.
- If the video decode line size is not a power-of-2 value (as in HD: 1920), the line length is rounded up to the next power-of-2 value (2048 in this example), and therefore some portion of the line is unused. This is referred to as *Line Stride*, being larger than the line length.

The results shown in the following table demonstrate video decode traffic sensitivity to address mapping. The optimal mapping is highlighted in yellow.



Table 22: Video Decompression Address Mapping

Address Mapping	Efficiency [%]
BG-optimized RBC (16R-2B-1BG-7C-1BG-3C)	64
RBC (16R-2B-2BG-10C)	64
D4_64t4k (14R-2B-5C-1BG-3R-2C-1BG-3C)	75
D4_64t2k (15R-2B-5C-1BG-2R-2C-1BG-3C)	58
RCB (16R-7C-2B-2BG-3C)	40

## Overview of Performance Tuning

Following is an overview of performance tuning for the NoC and integrated DDRMC:

- Choose the traffic class for each NoC master.
  - If the master has a cache, reads are low latency (LL) and writes are best effort.
  - If the master has any hard or soft real-time requirements (for example video applications), the traffic class is isochronous (ISOC).
  - Otherwise, choose best effort (BE). If unsure, this should be the default.
- Select bandwidth allocation for each flow. The bandwidth allocations determine what happens when all masters are trying to get service and there is contention.
  - For LL traffic, the bandwidth allocation should be a small fraction of the available DDR bandwidth.
  - For ISOC traffic, the bandwidth needed/generated by the master is typically known, and the allocation should match that bandwidth.
  - For BE traffic, choose a minimum desired bandwidth.
- Configure masters to send the longest bursts possible. This will have the best DDR efficiency.
- Tune the DDR performance first before the NoC. Ensure row/bank/column mappings are optimized.
- If DDR performance is optimal but the overall NoC traffic specification is still not met:
  - Examine shared routes through the NoC. Some masters may get less bandwidth than expected due to sub-optimal sharing of routes.

- b. A BE master may get less bandwidth than expected on reads due to finite read reorder buffer entries in the NMU.
  - i. Moving the master closer to DDR may help.
  - ii. Increase the bandwidth allocation of the master. For BE traffic this may not always help. The NoC applies some traffic shaping to BE based on allocations, but the memory controller treats all BE traffic with equal priority.
  - iii. Other options are to make the master ISOC, or to rate limit other BE masters.

# Memory Controller Pinout Rules and Future Expansion Options

## Pinout Rules

Pinouts or pin mapping for DDR4 and LPDDR4/4X memory interfaces are fixed and are output by the Vivado® tools as per the selected configuration. Some pinout flexibility is provided for DQ bits within a byte and is described in the pin rules section. Even when following the pin rules the recommendation is to validate the final pinout in Vivado. Strict adherence to all documented PCB guidelines is required for successful operation. For more information on PCB guidelines, see the *Versal ACAP PCB Design User Guide* ([UG863](#)).



### IMPORTANT!

*LPDDR4/4X protocol uses feedback on the DQ bits during CA Training and Write Leveling calibration stages so any pin swapping needs to be done and validated in the tools. Additionally the DQ mapping from the ACAP to the LPDDR4/4X component channels needs to maintain an exact one-to-one mapping. For example, LPDDR4\_DQ\_A[0] must be connected to DQ0 of Channel A of the LPDDR4 component, LPDDR4\_DQ\_A[1] must be connected to DQ1 of Channel A, through LPDDR4\_DQ\_B[15] that must be connected to DQ15 of Channel B of the LPDDR4 component. You are encouraged to try the Obtaining and Verifying Versal ACAP Memory Pinouts [tutorial](#) available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.*

Each DDRMC has three XPIO Banks (known as a triplet) associated with it. An XPIO Bank comprises twenty seven differential pin pairs (LOP/N to L26P/N). There are nine Nibbles in a Bank and each Nibble has six pins. The package pin name `IO_NiPj_MxPy` describes the fields listed below.

- **NiPj**: Ni is the Nibble number within a Bank (where i = 0 to 8)  
Pj is the pin number within the Nibble (where j = 0 to 5).
- **MxPy**: Mx is the DDRMC triplet number (where x = 0 to (Total number of DDRMCs - 1))  
Py is the pin number within the triplet (where y = 0 to 161).

- MxP0 through MxP53 are pins in the first Bank of the triplet.
- MxP54 through MxP107 are pins in the second Bank of the triplet.
- MxP108 through MxP161 are pins in the third Bank of the triplet.

**Table 23: Mapping of Triplet Pin to Nibble Pin in a Bank**

Triplet#Pin#	Nibble#Pin# in package pin name	Notes
M0P0	IO_L0P_XCC_N0P0	First pin in first Bank of triplet maps to first pin of Nibble 0 Pin 0
M0P1	IO_L0N_XCC_N0P1	
M0P2	IO_L1P_N0P2	
M0P3	IO_L1N_N0P3	
M0P4	IO_L2P_N0P4	
M0P5	IO_L2N_N0P5	Last pin of Nibble 0 in this Bank
M0P6	IO_L3P_XCC_N1P0	First pin of Nibble 1 in this Bank
M0P7	IO_L3N_XCC_N1P1	
M0P8	IO_L4P_N1P2	
M0P9	IO_L4N_N1P3	
M0P10	IO_L5P_N1P4	
M0P11	IO_L5N_N1P5	Last pin of Nibble 1 in this Bank
M0P12	IO_L6P_GC_XCC_N2P0	First pin of Nibble 2 in this Bank
M0P13	IO_L6N_GC_XCC_N2P1	
M0P14	IO_L7P_N2P2	
M0P15	IO_L7N_N2P3	
M0P16	IO_L8P_N2P4	
M0P17	IO_L8N_N2P5	Last pin of Nibble 2 in this Bank
M0P18	IO_L9P_GC_XCC_N3P0	First pin of Nibble 3 in this Bank
M0P19	IO_L9N_GC_XCC_N3P1	
M0P20	IO_L10P_N3P2	
M0P21	IO_L10N_N3P3	
M0P22	IO_L11P_N3P4	
M0P23	IO_L11N_N3P5	Last pin of Nibble 3 in this Bank
M0P24	IO_L12P_GC_XCC_N4P0	First pin of Nibble 4 in this Bank
M0P25	IO_L12N_GC_XCC_N4P1	
M0P26	IO_L13P_N4P2	
M0P27	IO_L13N_N4P3	
M0P28	IO_L14P_N4P4	
M0P29	IO_L14N_N4P5	Last pin of Nibble 4 in this Bank
M0P30	IO_L15P_XCC_N5P0	First pin of Nibble 5 in this Bank
M0P31	IO_L15N_XCC_N5P1	
M0P32	IO_L16P_N5P2	
M0P33	IO_L16N_N5P3	

Table 23: Mapping of Triplet Pin to Nibble Pin in a Bank (cont'd)

Triplet#Pin#	Nibble#Pin# in package pin name	Notes
M0P34	IO_L17P_N5P4	
M0P35	IO_L17N_N5P5	Last pin of Nibble 5 in this Bank
M0P36	IO_L18P_XCC_N6P0	First pin of Nibble 6 in this Bank
M0P37	IO_L18N_XCC_N6P1	
M0P38	IO_L19P_N6P2	
M0P39	IO_L19N_N6P3	
M0P40	IO_L20P_N6P4	
M0P41	IO_L20N_N6P5	Last pin of Nibble 6 in this Bank
M0P42	IO_L21P_XCC_N7P0	First pin of Nibble 7 in this Bank
M0P43	IO_L21N_XCC_N7P1	
M0P44	IO_L22P_N7P2	
M0P45	IO_L22N_N7P3	
M0P46	IO_L23P_N7P4	
M0P47	IO_L23N_N7P5	Last pin of Nibble 7 in this Bank
M0P48	IO_L24P_GC_XCC_N8P0	First pin of Nibble 8 in this Bank
M0P49	IO_L24N_GC_XCC_N8P1	
M0P50	IO_L25P_N8P2	
M0P51	IO_L25N_N8P3	
M0P52	IO_L26P_N8P4	
M0P53	IO_L26N_N8P5	Last pin of Nibble 8 in this Bank, Last pin in first Bank of triplet

## Non-Flipped versus Flipped

All the supported configurations have two versions of the pinout; *Non-Flipped* and *Flipped*. The Flipped version of the pinout is available for the following reasons:

- To provide the option to free up as many pins as possible that are not under the transceivers and processor systems for user system designs.
- To provide flexibility for PCB layout engineers for better signal routing.
- To provide the option to have contiguous XPIO Banks for user system designs.

The following illustration of the XPIO Banks in an XCVC1902VSVA2197 device will help explain the need for the flipped version of the pinout.

Figure 43: XPIO Banks in XCVC1902VSVA2197

GTU Quad 106 X0Y12-X0Y15 CD [L]	HDIO Bank 306 AA	HDIO Bank 406 AB	GTU Quad 206 X1Y24-X1Y27 BG [R]
GTU Quad 105 X0Y8-X0Y11 CC [L] (RCAL)	PCIE X0Y2	PCIE X1Y3	GTU Quad 205 X1Y20-X1Y23 BF [R]
GTU Quad 104 X0Y4-X0Y7 CB [L] (RCAL)	PCIE X0Y1	MRMAC X0Y3	GTU Quad 204 X1Y16-X1Y19 BE [R]
GTU Quad 103 X0Y0-X0Y3 CA [L]	CPM-G4	MRMAC X0Y2	GTU Quad 203 X1Y12-X1Y15 BD [R] (RCAL)
CPM-G4	CPM-G4	PCIE X1Y0	GTU Quad 202 X1Y8-X1Y11 BC [R] (RCAL)
LPD MIO 502	PMC FIO 503	MRMAC X0Y1	GTU Quad 201 X1Y4-X1Y7 BB [R]
PMC MIO 500	PMC MIO 501	MRMAC X0Y0	GTU Quad 200 X1Y0-X1Y3 BA [R]

XPIO Bank 700 A	XPIO Bank 701 B	XPIO Bank 702 C	XPIO Bank 703 D	XPIO Bank 704 E	XPIO Bank 705 F	XPIO Bank 706 G	XPIO Bank 707 H	XPIO Bank 708 I	XPIO Bank 709 J	XPIO Bank 710 K	XPIO Bank 711 L
-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------

X22193-100220

XPIO Banks 703 through 710 are fully fabric accessible and can be used for integrated memory controller designs as well as non-memory designs. XPIO Banks 700, 701, part of 702, and 711 can only be used for integrated memory controller designs. As an example, if the system requires a single rank L/RDIMM integrated memory controller and multiple non-memory designs then it would necessitate optimal pin utilization. As shown in the non-flipped single rank L/RDIMM pinout targeting XPIO Banks 709, 710, and 711 there are Free nibbles in Bank 711 which cannot be used for non-memory designs.

Figure 44: Non-Flipped Single Rank L/RDIMM pinout

	709										710										711									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	Free	Free			

X22832-050319

In this scenario using the flipped single rank L/RDIMM pinout would result in Free nibbles in XPIO Bank 709 allowing the use of these Free nibbles for non-memory designs.

Figure 45: Flipped Single Rank L/RDIMM pinout

	709										710										711									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ			

X22206-050319

The selection of Non-Flipped vs. Flipped pinout is controlled by the Pinout Swapping check box in the DDR Memory tab of the AXI NoC configuration dialog. For interleaved memory controllers, pinout swapping can be selected separately for each interleaved controller.

## DDR4

### DDR4 Pin Rules

**Note:** You are encouraged to try the *Obtaining and Verifying Versal ACAP Memory Pinouts* [tutorial](#) available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.

1. All Command, Address, Control, and Clock pins are fixed.
  - a. Command/Address/Control pins: ACT\_N, RAS\_N(A[16]), CAS\_N(A[15]), WE\_N(A[14]), A, BA, BG, CKE, CS\_N, ODT, PAR.
  - b. Clock pins: CK\_T, CK\_C.
2. All DQ/DM/DQS pins are fixed with the following swapping allowed:
  - a. DDR4 DQ bit swap.
    - i. DQ bits within a nibble can be swapped for x4 based interfaces.

**Table 24: Example of DQ bits swapped within Nibble**

Triplet #Pin#	Package pin name	Un-swapped x4	Swapped x4	Notes
MxP0	IO_L0P_XCC_N0P0	DQS0_t		Cannot swap
MxP1	IO_L0N_XCC_N0P1	DQS0_c		Cannot swap
MxP2	IO_L1P_N0P2	DQ3	DQ0	
MxP3	IO_L1N_N0P3	DQ1	DQ2	
MxP4	IO_L2P_N0P4	DQ2	DQ1	
MxP5	IO_L2NN0P5	DQ0	DQ3	

- ii. DQ bits within a byte can be swapped for x8/x16 based interfaces.

**Table 25: Example of DQ bits swapped within Byte**

Triplet#Pin#	Package pin name	Un-swapped x8/x16	Swapped x8/x16	Notes
MxP0	IO_L0P_XCC_N0P0	DM1		Cannot swap
MxP1	IO_L0N_XCC_N0P1	A12		Cannot swap
MxP2	IO_L1P_N0P2	DQ11	DQ8	
MxP3	IO_L1N_N0P3	DQ9	DQ12	
MxP4	IO_L2P_N0P4	DQ10	DQ14	
MxP5	IO_L2N_N0P5	DQ8	DQ15	
MxP6	IO_L3P_XCC_N1P0	DQS1_t		Cannot swap
MxP7	IO_L3N_XCC_N1P1	DQS1_c		Cannot swap
MxP8	IO_L4P_N1P2	DQ12	DQ9	



Table 25: Example of DQ bits swapped within Byte (cont'd)

Triplet#Pin#	Package pin name	Un-swapped x8/x16	Swapped x8/x16	Notes
MxP9	IO_L4N_N1P3	DQ14	DQ13	
MxP10	IO_L5P_N1P4	DQ13	DQ10	
MxP11	IO_L5N_N1P5	DQ15	DQ11	

b. DDR4 DQ nibble/byte swap.

- i. Any byte including the ECC byte (CB) can be swapped with any other byte for x8/x16 based component and DIMM interfaces. DQ bits within the swapped bytes can also be swapped.

Table 26: Example of Byte1 swapped with Byte0

Triplet#Pin#	Package pin name	Un-swapped x8	Swapped x8	Notes
MxP0	IO_L0P_XCC_N0P0	DM1	DM0	Data Mask only
MxP1	IO_L0N_XCC_N0P1	A12		Cannot swap
MxP2	IO_L1P_N0P2	DQ11	DQ3	
MxP3	IO_L1N_N0P3	DQ9	DQ1	
MxP4	IO_L2P_N0P4	DQ10	DQ2	
MxP5	IO_L2N_N0P5	DQ8	DQ0	
MxP6	IO_L3P_XCC_N1P0	DQS1_t	DQS0_t	DQSx_t only
MxP7	IO_L3N_XCC_N1P1	DQS1_c	DQS0_c	DQSx_c only
MxP8	IO_L4P_N1P2	DQ12	DQ4	
MxP9	IO_L4N_N1P3	DQ14	DQ6	
MxP10	IO_L5P_N1P4	DQ13	DQ7	
MxP11	IO_L5N_N1P5	DQ15	DQ5	
MxP12	IO_L6P_GC_XCC_N2P0	DM0	DM1	Data Mask only
MxP13	IO_L6N_GC_XCC_N2P1	A7		cannot swap
MxP14	IO_L7P_N2P2	DQ3	DQ11	
MxP15	IO_L7N_N2P3	DQ1	DQ9	
MxP16	IO_L8P_N2P4	DQ2	DQ10	
MxP17	IO_L8N_N2P5	DQ0	DQ8	
MxP18	IO_L9P_GC_XCC_N3P0	DQS0_t	DQS1_t	DQSx_t only
MxP19	IO_L9N_GC_XCC_N3P1	DQS0_c	DQS1_c	DQSx_c only
MxP20	IO_L10P_N3P2	DQ4	DQ12	
MxP21	IO_L10N_N3P3	DQ6	DQ14	
MxP22	IO_L11P_N3P4	DQ7	DQ13	
MxP23	IO_L11N_N3P5	DQ5	DQ15	

- ii. Any byte including the ECC byte (CB) with any other byte and nibbles within bytes can be swapped for x4 based DIMM interfaces. DQ bits within the swapped nibbles can also be swapped.

**Table 27: Example of swapping Nibbles within a Byte**

Triplet#Pin#	Package pin name	Un-swapped x4	Swapped x4	Notes
MxP0	IO_L0P_XCC_N0P0	DQS4_t	DQS13_t	DQSx_t only
MxP1	IO_L0N_XCC_N0P1	DQS4_c	DQS13_c	DQSx_c only
MxP2	IO_L1P_N0P2	DQ35	DQ37	
MxP3	IO_L1N_N0P3	DQ34	DQ36	
MxP4	IO_L2P_N0P4	DQ33	DQ39	
MxP5	IO_L2N_N0P5	DQ32	DQ38	
MxP6	IO_L3P_XCC_N1P0	DQS13_t	DQS4_t	DQSx_t only
MxP7	IO_L3N_XCC_N1P1	DQS13_c	DQS4_c	DQSx_c only
MxP8	IO_L4P_N1P2	DQ37	DQ35	
MxP9	IO_L4N_N1P3	DQ36	DQ34	
MxP10	IO_L5P_N1P4	DQ39	DQ33	
MxP11	IO_L5N_N1P5	DQ38	DQ32	

- iii. Any nibble with any other nibble can be swapped for x4 based component interfaces. DQ bits within the swapped nibbles can also be swapped.

**Table 28: Example of swapping any Nibble with any other Nibble**

Triplet#Pin#	Package pin name	Un-swapped x4	Swapped x4	Notes
MxP6	IO_L3P_XCC_N1P0	DQS1_t	DQS2_t	DQSx_t only
MxP7	IO_L3N_XCC_N1P1	DQS1_c	DQS2_c	DQSx_c only
MxP8	IO_L4P_N1P2	DQ4	DQ8	
MxP9	IO_L4N_N1P3	DQ6	DQ11	
MxP10	IO_L5P_N1P4	DQ7	DQ10	
MxP11	IO_L5N_N1P5	DQ5	DQ9	
MxP54	IO_L0P_XCC_N0P0	DQS2_t	DQS1_t	DQSx_t only
MxP55	IO_L0N_XCC_N0P1	DQS2_c	DQS1_c	DQSx_c only
MxP56	IO_L1P_N0P2	DQ8	DQ4	
MxP57	IO_L1N_N0P3	DQ11	DQ6	
MxP58	IO_L2P_N0P4	DQ10	DQ7	
MxP59	IO_L2N_N0P5	DQ9	DQ5	

- c. Swap across memory channels is not allowed. MC channel-0 must not be swapped with MC channel-1.

3. Reduced DDR4 data width configurations must remove from the highest numbered DQs in order.
  - 72-bit converts to 64-bit by removing DQ71 down to DQ64
4. In XP I/O bank 700 and bank 800 (not present in all devices), there is an additional bank pin that is used as a reference to calibrate internal on-die termination. The IO\_VR\_700/IO\_VR\_800 pin must be externally connected to a 240Ω resistor on the PCB and pulled up to the bank VCCO voltage.
5. Pinout for RDIMM/LRDIMM is not pin compatible with that of UDIMMs.
6. Pinout for UDIMM is not pin compatible with that of components.
7. Pinouts are compatible between x4 and x8 RDIMM/LRDIMM. If PCB compatibility between x4 and x8 RDIMM/LRDIMM is desired, the following pin swapping restrictions apply.
  - DQ bits can only be swapped within a nibble
  - Nibbles cannot be swapped with any other nibble
  - Bytes can be swapped with any other byte
8. One differential system clock source (sys\_clk) is required per integrated DDR MC. Refer to Clocking for details.
9. If an entire nibble is free then it can be used for non-memory designs.
10. Multi-component interfaces must use same data width components. For example, a 72-bit interface can use nine x8 components or five x16 components.

## Related Information

[Clocking](#)

## DDR4 Pinouts for Supported Configurations

Supported configurations for DDR4 components, UDIMM/SODIMM, and L/RDIMM are listed under Memory Configuration Support in the [DDR Memory Controller](#) section of this Product Guide.



**RECOMMENDED:** You are encouraged to try the *Obtaining and Verifying Versal ACAP Memory Pinouts tutorial* available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.

## Related Information

[DDR Memory Controller](#)

## DDR4 Pinout for Multi-Rank L/RDIMM Interfaces (Non-Flipped)

Nibble utilization for 72-bit, Multi-Rank L/RDIMM with ECC in the non-flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 46: Nibble utilization for 72-bit Multi-rank L/RDIMM with ECC (Non-flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ			

X22210-050319

## DDR4 Pinout for Multi-Rank L/RDIMM Interfaces (Flipped)

Nibble utilization for 72-bit, Multi-Rank L/RDIMM interface with ECC in the flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 47: Nibble utilization for 72-bit Multi-Rank L/RDIMM with ECC (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ			

X22211-050319

## DDR4 Pinout for Single Rank L/RDIMM Interfaces (Non-Flipped)

Nibble utilization for 72-bit, Single Rank L/RDIMM interface with ECC in the non-flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 48: Nibble utilization for 72-bit Single rank L/RDIMM with ECC (Non-flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	Free	Free			

X22212-050319

## DDR4 Pinout for Single Rank L/RDIMM Interfaces (Flipped)

Nibble utilization for 72-bit, Single Rank L/RDIMM interface with ECC in the flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 49: Nibble utilization for 72-bit Single rank L/RDIMM with ECC (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ			

X22213-050319

## DDR4 Pinout for UDIMM/SODIMM Interfaces (Non-Flipped)

Nibble utilization for a 72-bit, Single Rank UDIMM/SODIMM interface with ECC in the non-flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, `sys_clk` indicates a nibble comprising the System Clock pair, `RESET_n`, and `ALERT_n`. Note that for a 64-bit, Single Rank UDIMM/SODIMM interface without ECC, nibbles 4 and 5 in addition to nibbles 6, 7, and 8 in the third Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 50: Nibble utilization for 72-bit Single rank UDIMM/SODIMM with ECC (Non-flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	Free	Free			

X22214-050319

Nibble utilization for a 72-bit, Multi-rank (Single Slot, 2 Rank) UDIMM/SODIMM interface with ECC in the non-flipped configuration is shown in the following figure. Note that for a 64-bit Multi-rank (Single Slot, 2 Rank) UDIMM/SODIMM interface without ECC nibbles 0, and 1 in addition to nibble 8 in the third bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.



**Figure 51: Nibble utilization for 72-bit Multi-rank (single slot, 2 rank) UDIMM/SODIMM with ECC (Non-flipped)**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ			

X22215-050319

## DDR4 Pinout for UDIMM/SODIMM Interfaces (Flipped)

Nibble utilization for a 72-bit, Single rank UDIMM/SODIMM interface with ECC in the flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n. Note that for a 64-bit, Single rank UDIMM/SODIMM interface without ECC nibbles 2 and 3 in addition to nibbles 0, 1, and 8 in the first Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 52: Nibble utilization for 72-bit Single rank UDIMM/SODIMM with ECC (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ			

X22216-050319

Nibble utilization for a 72-bit, Multi-rank (Single Slot, 2 Rank) UDIMM/SODIMM interface with ECC in the flipped configuration is shown in the following figure. Note that for a 64-bit, Multi-rank (Single Slot, 2 Rank) UDIMM/SODIMM interface without ECC, nibbles 6, and 7 in addition to nibble 8 in the first bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

**Figure 53: Nibble utilization for 72-bit Multi-rank (single slot, 2 rank) UDIMM/SODIMM with ECC (Flipped)**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ			

X22217-050319

## DDR4 Pinout for Component Interfaces (Non-Flipped)

Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 1CK pair in the non-flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, `sys_clk` indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a reduced data width of 64-bits, nibbles 4 and 5 in addition to nibbles 6, 7, and 8 in the third Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 54: Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 1CK (Non-Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ 1	DQ 1	DQ 0	DQ 0	AC	AC	AC	AC	AC	DQ 3	DQ 3	DQ 4	DQ 4	sys_clk	DQ 2	DQ 2	DQ 5	DQ 5	DQ 6	DQ 6	DQ 7	DQ 7	Free	DQ 8	DQ 8	Free	Free			

X22218-102320

Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 2CK pairs in the non-flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, `sys_clk` indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a reduced data width of 64-bits, nibbles 6, and 7 in addition to nibble 8 in the third Bank and nibble 2 in the first Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

**Figure 55: Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 2CK (Non-Flipped)**

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ 0	DQ 0	Free	AC	AC	AC	AC	AC	AC	DQ 1	DQ 1	DQ 2	DQ 2	sys_clk	DQ 3	DQ 3	DQ 4	DQ 4	DQ 5	DQ 5	DQ 6	DQ 6	Free	DQ 7	DQ 7	DQ 8	DQ 8

X22219-102320

Nibble utilization for 40-bit interface using x4 DDP (2 Ranks) or 3DS components with 2CK pairs in the non-flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

**Figure 56: Nibble utilization for 40-bit interface using x4 DDP (2 rank) or 3DS components with 2CK (Non-Flipped)**

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ 0	DQ 0	AC	AC	AC	AC	AC	AC	AC	DQ 1	DQ 1	DQ 2	DQ 2	sys_clk	DQ 3	DQ 3	DQ 4	DQ 4	Free	Free	Free	Free	Free	Free	Free	Free	Free

X22220-102320

## DDR4 Pinout for Component Interfaces (Flipped)

Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 1CK pair in the flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a reduced data width of 64-bits, nibbles 2 and 3 in addition to nibbles 0, 1, and 8 in the first Bank would be free.



 **IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 57: Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 1CK (Flipped)

Nibble #	First Bank									Second Bank									Third Bank								
	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ

X22221-050319

Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 2CK pairs in the flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a reduced data width of 64-bits, nibbles 0 and 1 in addition to nibble 8 in the first Bank and nibble 5 in the third Bank would be free.

 **IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

**Figure 58: Nibble utilization for 72-bit interface using x8, x16, x8 DDP 2 rank, or 3DS components with 2CK (Flipped)**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	Free	DQ	DQ			

X22222-050319

Nibble utilization for 40-bit interface using x4 DDP (2 Ranks) or 3DS components with 2CK pairs in the flipped configuration is shown in the following figure. DQ indicates a data nibble, AC indicates an Address/Command/Control nibble, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

**Figure 59: Nibble utilization for 40-bit interface using x4 DDP (2 rank) or 3DS components with 2CK (Flipped)**

	First Bank										Second Bank								Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	Free	Free	Free	Free	Free	Free	Free	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ

X22223-050319

## DDR4 Pinout for 2x Component Interfaces (Non-Flipped)

The DDRMC can also be configured as two independent DDR interfaces of 16, 24, or 32 data bits each. The non-flipped 2x component pinout configurations are included in this section.

Nibble utilization for 2x32 interface using SDP, DDP (2 Ranks) or 3DS components in the non-flipped configuration is shown in the following figure. DQL and DQR indicate data nibbles for the left and right interfaces respectively, ACL and ACR indicate Address/Command/Control nibbles for the left and right interfaces respectively, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a 1x32 interface all nibbles in the second Bank and nibbles 2, 3, 6, and 7 in the third Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 60: Nibble utilization for 2x32 interface using SDP, DDP (2 rank), or 3DS components (Non-Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ			
	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH	CH		CH	CH	CH	CH			
	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1		0	0	1	1			

X22224-101420

Nibble utilization for 2x16 interface using SDP, DDP (2 Ranks) or 3DS components in the non-flipped configuration is shown in the following figure. DQL and DQR indicate data nibbles for the left and right interfaces respectively, ACL and ACR indicate Address/Command/Control nibbles for the left and right interfaces respectively, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a 1x16 interface all nibbles in the second Bank would be free in addition to the free nibbles in the third Bank.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.



**Figure 61: Nibble utilization for 2x16 interface using SDP, DDP (2 rank) or 3DS components (Non-Flipped)**

Nibble #	First Bank										Second Bank										Third Bank									
	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ CH 0	DQ CH 0	DQ CH 0	DQ CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	DQ CH 1	DQ CH 1	DQ CH 1	DQ CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free			

X22225-101420

Nibble utilization for 2x24 interface using SDP, DDP (2 Ranks) or 3DS components in the non-flipped configuration is shown in the following figure. DQL and DQR indicate data nibbles for the left and right interfaces respectively, ACL and ACR indicate Address/Command/Control nibbles for the left and right interfaces respectively, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a 1x24 interface all nibbles in the second Bank and nibbles 2 and 3 in addition to nibbles 4, 5, 6, and 7 in the third bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

**Figure 62: Nibble utilization for 2x24 interface using SDP, DDP (2 rank) or 3DS components (Non-Flipped)**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ CH 0	DQ CH 0	DQ CH 0	DQ CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	DQ CH 1	DQ CH 1	DQ CH 1	DQ CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	DQ CH 0	DQ CH 0	DQ CH 1	DQ CH 1	sys_clk	Free	Free	Free	Free			

X22226-101420

## DDR4 Pinout for 2x Component Interfaces (Flipped)

The DDRMC can also be configured as two independent DDR interfaces of 16, 24, or 32 data bits each. The flipped 2x component pinout configurations are included in this section.

Nibble utilization for 2x32 interface using SDP, DDP (2 Ranks) or 3DS components in the flipped configuration is shown in the following figure. DQL and DQR indicate data nibbles for the left and right interfaces respectively, ACL and ACR indicate Address/Command/Control nibbles for the left and right interfaces respectively, `sys_clk` indicates a nibble comprising the System Clock pair, `RESET_n`, and `ALERT_n` signals. For a 1x32 interface all nibbles in the second Bank and nibbles 0, 1, 4, and 5 in the first Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 63: Nibble utilization for 2x32 interface using SDP, DDP (2 rank), or 3DS components (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ			
	CH 1	CH 1	CH 0	CH 0		CH 1	CH 1	CH 0	CH 0	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0			

X22227-101420

Nibble utilization for 2x16 interface using SDP, DDP (2 Ranks) or 3DS components in the flipped configuration is shown in the following figure. DQL and DQR indicate data nibbles for the left and right interfaces respectively, ACL and ACR indicate Address/Command/Control nibbles for the left and right interfaces respectively, `sys_clk` indicates a nibble comprising the System Clock pair, `RESET_n`, and `ALERT_n` signals. For a 1x16 interface all nibbles in the second Bank would be free Bank in addition to the free nibbles in the first Bank.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 64: Nibble utilization for 2x16 interface using SDP, DDP (2 rank), or 3DS components (Flipped)

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free	AC CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	DQ CH 1	DQ CH 1	DQ CH 1	DQ CH 1	AC CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	DQ CH 0	DQ CH 0	DQ CH 0	DQ CH 0

X22228-101420

Nibble utilization for 2x24 interface using SDP, DDP (2 Ranks) or 3DS components in the flipped configuration is shown in the following figure. DQL and DQR indicate data nibbles for the left and right interfaces respectively, ACL and ACR indicate Address/Command/Control nibbles for the left and right interfaces respectively, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. For a 1x24 interface all nibbles in the second Bank would be free and nibbles 4 and 5 in addition to nibbles 0, 1, 2, and 3 in the first Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.


Figure 65: Nibble utilization for 2x24 interface using SDP, DDP (2 rank), or 3DS components (Flipped)

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	Free	Free	sys_clk	DQ CH 1	DQ CH 1	DQ CH 0	DQ CH 0	AC CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	DQ CH 1	DQ CH 1	DQ CH 1	DQ CH 1	AC CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	DQ CH 0	DQ CH 0	DQ CH 0	DQ CH 0


X22229-101420

## LPDDR4/4X

### LPDDR4/4X Pin Rules

 **IMPORTANT!** LPDDR4/4X protocol uses feedback on the DQ bits during CA Training and Write Leveling calibration stages so any pin swapping needs to be done and validated in the tools. Additionally the DQ mapping from the ACAP to the LPDDR4/4X component channels needs to maintain an exact one-to-one mapping. For example, LPDDR4\_DQ\_A[0] must be connected to DQ0 of Channel A of the LPDDR4 component, LPDDR4\_DQ\_A[1] must be connected to DQ1 of Channel A, through LPDDR4\_DQ\_B[15] that must be connected to DQ15 of Channel B of the LPDDR4 component. You are encouraged to try the Obtaining and Verifying Versal ACAP Memory Pinouts [tutorial](#) available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.

1. All Command, Address, Control, and Clock pins are fixed.
  - a. Command/Address/Control pins: CA\_A, CA\_B, CKE\_A, CKE\_B, CS\_A, CS\_B.
  - b. Clock pins: CK\_T, CK\_C.
2. All DQ/DM/DQS pins are fixed with the following swapping allowed:
  - a. LPDDR4/4X DQ bit swap.

 **IMPORTANT!** LPDDR4/4X protocol uses feedback on the DQ bits during CA Training and Write Leveling calibration stages so any pin swapping needs to be done and validated in the tools. Additionally the DQ mapping from the ACAP to the LPDDR4/4X component channels needs to maintain an exact 1:1 mapping. For example, LPDDR4\_DQ\_A[0] must be connected to DQ0 of Channel A of the LPDDR4 component. It follows LPDDR4\_DQ\_A[1] is connected to DQ1 of Channel A, through LPDDR4\_DQ\_B[15] is connected to DQ15 of Channel B of the LPDDR4 component.

- i. DQ bits within a byte can be swapped.

Table 29: Example of DQ Bits swapped within a Byte

Triplet#Pin#	Package pin name	Un-swapped	Swapped	Notes
MxP0	IO_L0P_XCC_N0P0	DMI3_0		Cannot swap
MxP1	IO_L0N_XCC_N0P1	CS0_B_0		Cannot swap
MxP2	IO_L1P_N0P2	DQ30_0	DQ25_0	
MxP3	IO_L1N_N0P3	DQ31_0	DQ28_0	
MxP4	IO_L2P_N0P4	DQ27_0	DQ29_0	
MxP5	IO_L2N_N0P5	DQ26_0	DQ30_0	
MxP6	IO_L3P_XCC_N1P0	DQS3_T_0		Cannot swap
MxP7	IO_L3N_XCC_N1P1	DQS3_C_0		Cannot swap
MxP8	IO_L4P_N1P2	DQ29_0	DQ24_0	
MxP9	IO_L4N_N1P3	DQ28_0	DQ27_0	

Table 29: Example of DQ Bits swapped within a Byte (cont'd)

Triplet#Pin#	Package pin name	Un-swapped	Swapped	Notes
MxP10	IO_L5P_N1P4	DQ24_0	DQ31_0	
MxP11	IO_L5N_N1P5	DQ25_0	DQ26_0	

b. LPDDR4/4X DQ byte swap.

- i. Swap of bytes between x16 halves (within channel A or B of LPDDR4/4X device). As shown in the example, DQ[7:0] and associated DQS0 can be swapped with DQ[15:8] and associated DQS1. However, DQ[15:0] (channel A) must not be swapped with DQ[31:16] (channel B).

Table 30: Example of Bytes swapped within each channel

Triplet#Pin#	Package pin name	Un-swapped	Swapped	Notes
MxP78	IO_L12P_GC_XCC_N4P0	DQS0_T_0	DQS1_T_0	DQSx_T only
MxP79	IO_L12N_GC_XCC_N4P1	DQS0_C_0	DQS1_C_0	DQSx_C only
MxP80	IO_L13P_N4P2	DQ0_0	DQ13_0	
MxP81	IO_L13N_N4P3	DQ1_0	DQ12_0	
MxP82	IO_L14P_N4P4	DQ2_0	DQ9_0	
MxP83	IO_L14N_N4P5	DQ3_0	DQ14_0	
MxP84	IO_L15P_XCC_N5P0	DMI0_0	DMI1_0	Data Mask only
MxP85	IO_L15N_XCC_N5P1	NC		
MxP86	IO_L16P_N5P2	DQ7_0	DQ8_0	
MxP87	IO_L16N_N5P3	DQ6_0	DQ15_0	
MxP88	IO_L17P_N5P4	DQ4_0	DQ10_0	
MxP89	IO_L17N_N5P5	DQ5_0	DQ11_0	
MxP12	IO_L6P_GC_XCC_N2P0	DQS1_T_0	DQS0_T_0	DQSx_T only
MxP13	IO_L6N_GC_XCC_N2P1	DQS1_C_0	DQS0_C_0	DQSx_C only
MxP14	IO_L7P_N2P2	DQ13_0	DQ0_0	
MxP15	IO_L7N_N2P3	DQ12_0	DQ1_0	
MxP16	IO_L8P_N2P4	DQ9_0	DQ2_0	
MxP17	IO_L8N_N2P5	DQ14_0	DQ3_0	
MxP18	IO_L9P_GC_XCC_N3P0	DMI1_0	DMI0_0	Data Mask only
MxP19	IO_L9N_GC_XCC_N3P1	CS1_B_0		Cannot swap
MxP20	IO_L10P_N3P2	DQ8_0	DQ7_0	
MxP21	IO_L10N_N3P3	DQ15_0	DQ6_0	

Table 30: Example of Bytes swapped within each channel (cont'd)

Triplet#Pin#	Package pin name	Un-swapped	Swapped	Notes
MxP22	IO_L11P_N3P4	DQ10_0	DQ4_0	
MxP23	IO_L11N_N3P5	DQ11_0	DQ5_0	

- c. Swap across memory channels is not allowed. MC channel-0 must not be swapped with MC channel-1.
3. In XP I/O bank 700 and bank 800 (not present in all devices), there is an additional bank pin that is used as a reference to calibrate internal on-die termination. The IO\_VR\_700/IO\_VR\_800 pin must be externally connected to a 240Ω resistor on the PCB and pulled up to the bank VCCO voltage..
4. One differential system clock source (`sys_clk`) is required per integrated DDR MC. Refer to Clocking for details.
5. If an entire nibble is free then it can be used for non-memory designs.

## Related Information

[Clocking](#)

## LPDDR4/4X Pinout for Supported Configurations

Supported configurations for LPDDR4/4X are listed under Memory Configuration Support in the DDR Memory Controller section of this Product Guide.



**RECOMMENDED:** You are encouraged to try the Obtaining and Verifying Versal ACAP Memory Pinouts [tutorial](#) available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.



**IMPORTANT!** LPDDR4/4X protocol uses feedback on the DQ bits during CA Training and Write Leveling calibration stages so any pin swapping needs to be done and validated in the tools. Additionally the DQ mapping from the ACAP to the LPDDR4/4X component channels needs to maintain an exact 1:1 mapping. For example, LPDDR4\_DQ\_A[0] must be connected to DQ0 of Channel A of the LPDDR4 component. It follows LPDDR4\_DQ\_A[1] is connected to DQ1 of Channel A, through LPDDR4\_DQ\_B[15] is connected to DQ15 of Channel B of the LPDDR4 component.

## Related Information

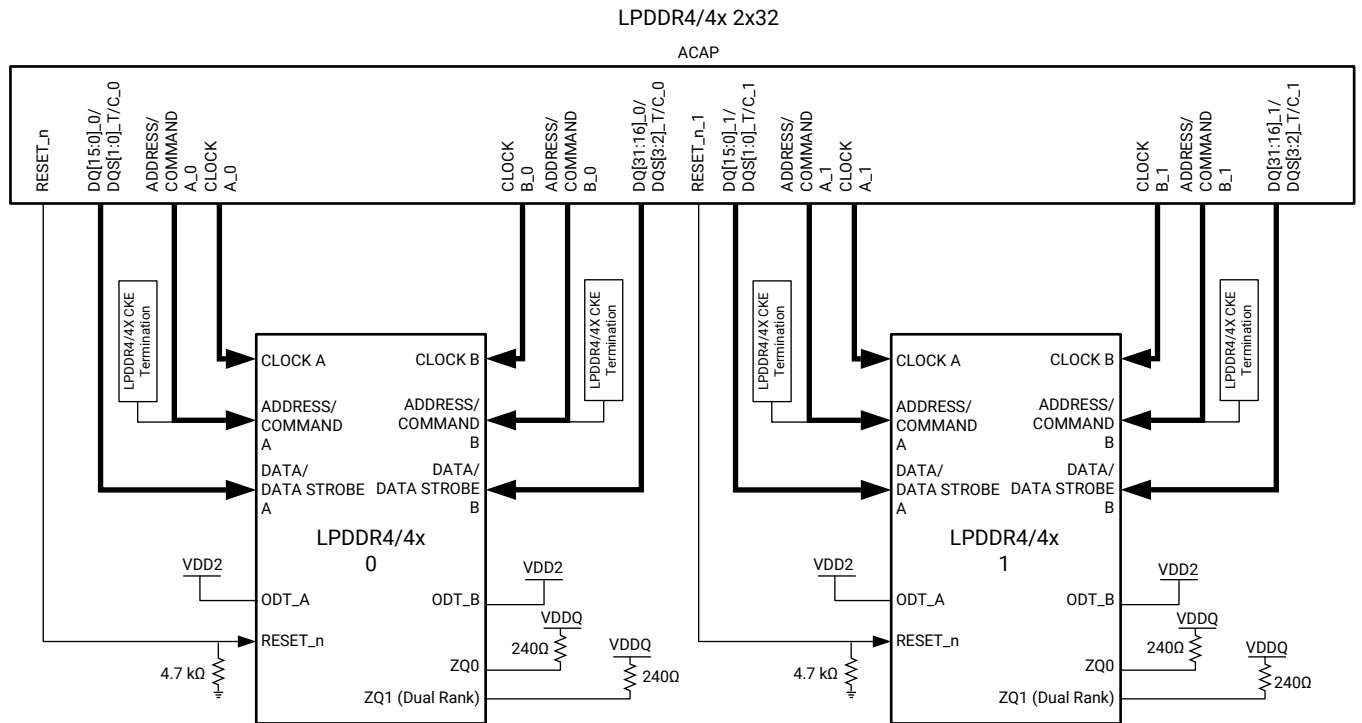
[DDR Memory Controller](#)

## LPDDR4/4X Pinout for Component Interfaces (Non-Flipped)

### 2x32 Component Interface (Non-Flipped)

The integrated DDRMC can also be configured as two independent DDR interfaces of 16 or 32 data bits each. This section describes the Non-Flipped 2x32 LPDDR4/4X interface.

Figure 66: Connections for a 2x32 LPDDR4/4x Interface



X21603-062120

Nibble utilization for 2x32 interface using two x32 components in the non-flipped configuration is shown in the following figure. This pinout provides the maximum interface performance compared to the 2x32 pin efficient version. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to [Clocking](#) for System Clock details.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 67: Nibble utilization for 2x32 component interface (Non-Flipped)

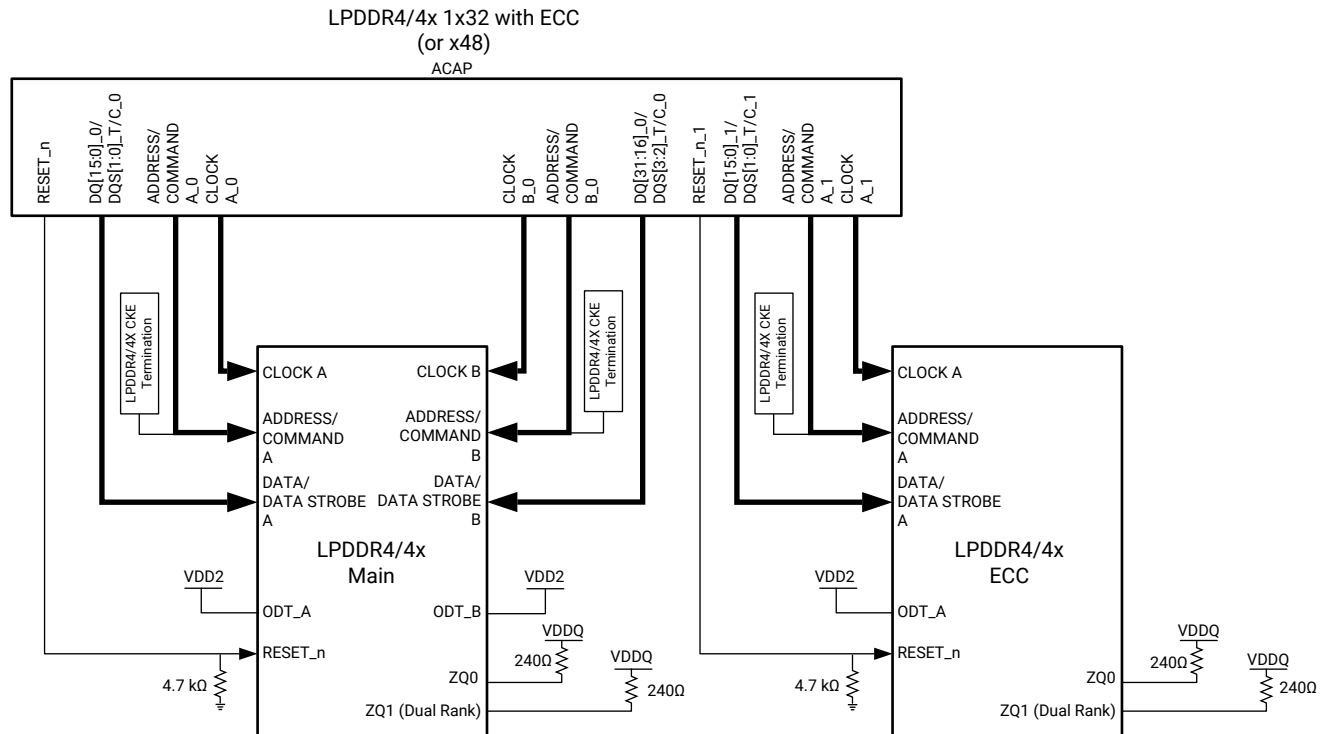
	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q  CH 0 B	D Q  CH 0 B	D Q  CH 0 A	D Q  CH 0 A	 sys_clk	A C  CH 0 B	A C  CH 0 B	A C  CH 0 A	A C  CH 0 A	D Q  CH 0 B	D Q  CH 0 B	D Q  CH 1 B	D Q  CH 1 B	 Free	D Q  CH 0 A	D Q  CH 0 A	D Q  CH 1 A	D Q  CH 1 A	A C  CH 1 A	A C  CH 1 A	D Q  CH 1 A	D Q  CH 1 A	 Free	A C  CH 1 B	A C  CH 1 B	D Q  CH 1 B	D Q  CH 1 B

X22230-101520



## 1x48 Component Interface (Non-Flipped)

Figure 68: Connections for a 1x32 (or x48) with ECC LPDDR4/4x Interface



X21604-062120

Nibble utilization for 1x48 interface (or 1x32 with ECC) in the non-flipped configuration is shown in the following figure. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to Clocking for System Clock details. Note that the AC nibbles for the ECC device are on nibbles 0 and 1 in the third Bank and DQ nibbles for the ECC device are on nibbles 2 and 3 in the third Bank and nibble 6 and 7 in the second Bank. For a 1x32 interface without ECC all nibbles in the third Bank and nibbles 2, 3, 6, 7, and 8 in the second Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 69: Nibble utilization for 1x48 component interface (Non-Flipped)

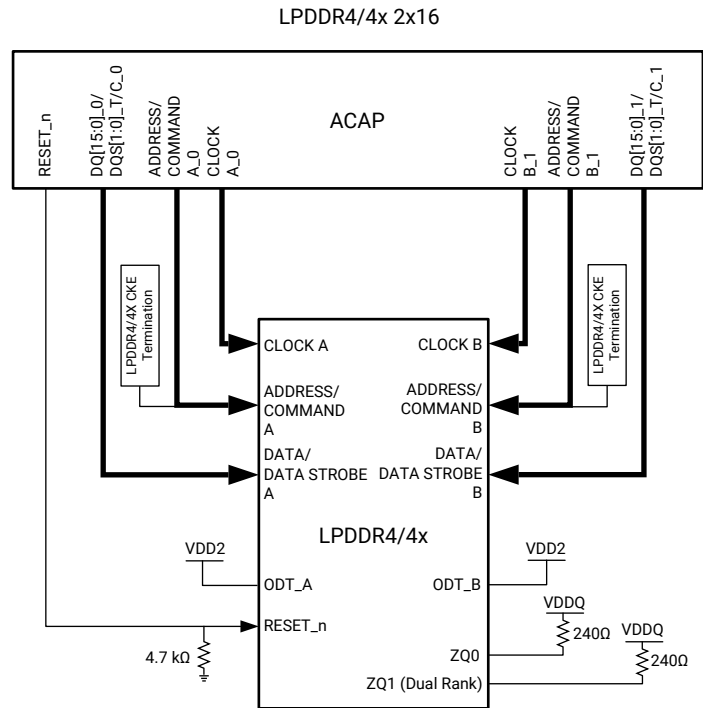
Nibble #	First Bank								Second Bank								Third Bank										
	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ B	DQ B	DQ A	DQ A	sys_clk B	AC B	AC B	AC A	AC A	DQ B	DQ B	Free	Free	Free	DQ A	DQ A	DQ A 2	DQ A 2	AC A 2	AC A 2	DQ A 2	DQ A 2	Free	Free	Free	Free	Free

X22231-101520

## 2x16 Component Interface (Non-Flipped)

The integrated DDRMC can also be configured as two independent DDR interfaces of 16 or 32 data bits each. This section describes the Non-Flipped 2x16 LPDDR4/4X interface.

Figure 70: Connections for a 2x16 LPDDR4/4x Interface



X21605-062120

Nibble utilization for 2x16 interface using a x32 component in the non-flipped configuration is shown in the following figure. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to [Clocking](#) for System Clock details.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 71: Nibble utilization for 2x16 component interface (Non-Flipped)

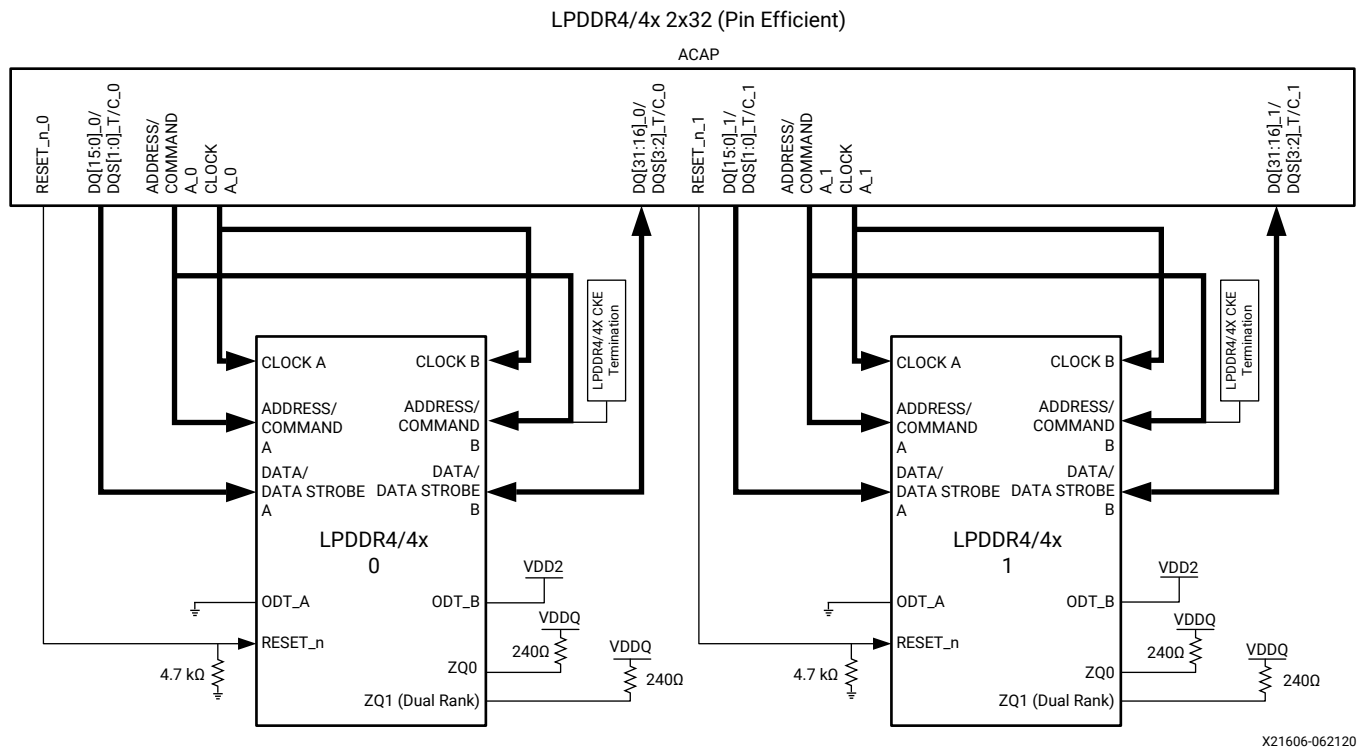
	First Bank									Second Bank								Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q  CH 1	D Q  CH 1	D Q  CH 0	D Q  CH 0	sys_clk	A C  CH 1	A C  CH 1	A C  CH 0	A C  CH 0	D Q  CH 1	D Q  CH 1	Free	Free	Free	D Q  CH 0	D Q  CH 0	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free

X22232-101520

## Pin Efficient 2x32 Component Interface (Non-Flipped)

The integrated DDRMC can also be configured as two independent DDR interfaces of 16 or 32 data bits each. This section describes the Non-Flipped pin efficient 2x32 LPDDR4/4X interface.

Figure 72: Connections for a Pin Efficient 2x32 LPDDR4/4x Interface



Nibble utilization for pin efficient 2x32 interface using two x32 components in the non-flipped configuration is shown in the following figure. For maximum interface performance use the 2x32 pinout described in [Figure 67: Nibble utilization for 2x32 component interface \(Non-Flipped\)](#). DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to [Clocking](#) for System Clock details.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

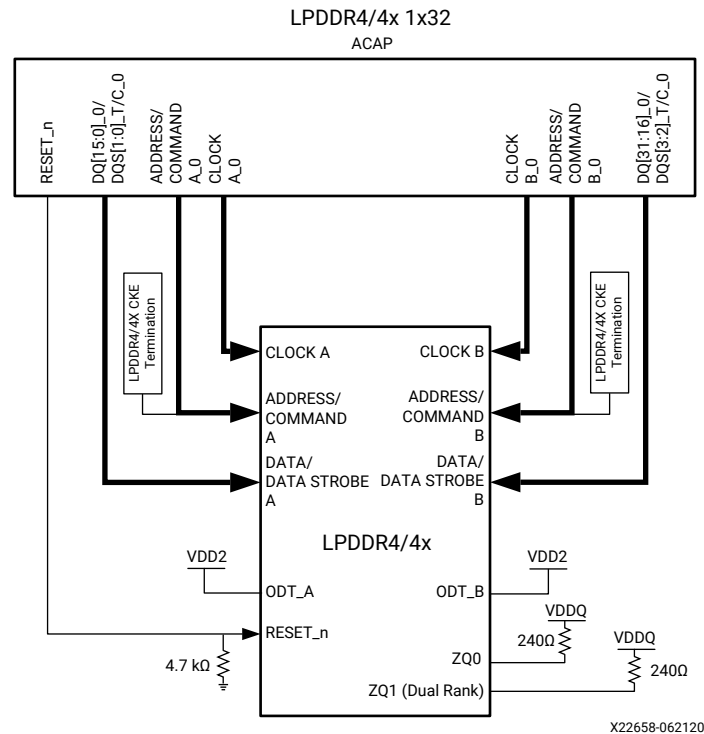
Figure 73: Nibble utilization for pin efficient 2x32 component interface (Non-Flipped)

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q	D Q	D Q	D Q	A C	D Q	D Q	D Q	D Q	D Q	D Q	D Q	D Q	A C	D Q	D Q	D Q	D Q									
	CH 0 B	CH 0 B	CH 0 B	CH 0 B	CH 0	CH 0 A	CH 0 A	CH 0 A	CH 0 A	CH 1 A	CH 1 A	CH 1 B	CH 1 B	CH 1	CH 1 A	CH 1 A	CH 1 B	CH 1 B	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free

X22233-101520

## 1x32 Component Interface (Non-Flipped)

Figure 74: Connections for a 1x32 LPDDR4/4x Interface



Nibble utilization for 1x32 interface using one x32 component in the non-flipped configuration is shown in the following figure. This pinout provides the maximum interface performance compared to the 1x32 pin efficient version. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to [Clocking](#) for System Clock details.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 75: Nibble utilization for 1x32 component interface (Non-Flipped)

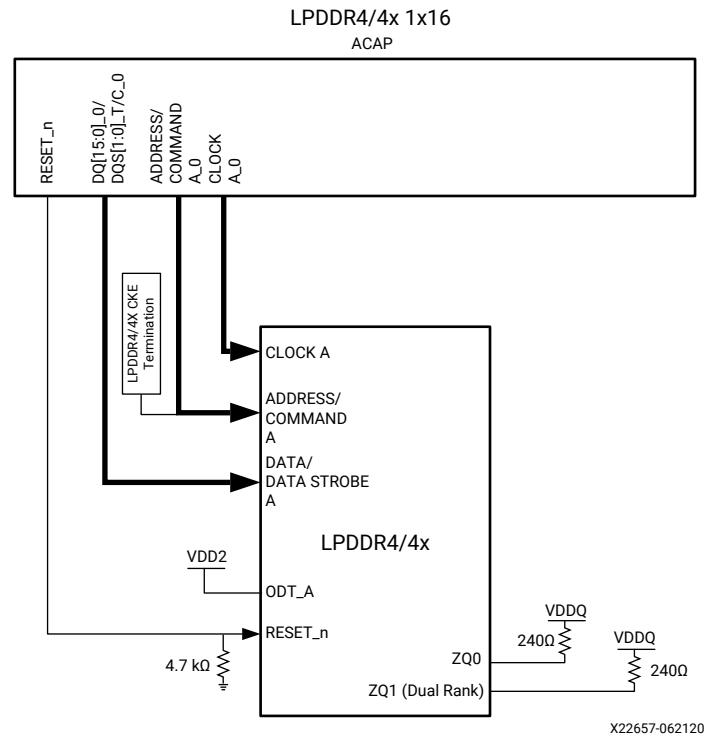
Nibble #	First Bank								Second Bank								Third Bank										
	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q B	D Q B	D Q A	D Q A	sys_clk	A C B	A C B	A C A	A C A	D Q B	D Q B	Free	Free	Free	D Q A	D Q A	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free

X22573-101520



## 1x16 Component Interface (Non-Flipped)

Figure 76: Connections for a 1x16 LPDDR4/4x Interface



Nibble utilization for 1x16 interface in the non-flipped configuration is shown in the following figure. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to [Clocking](#) for System Clock details.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

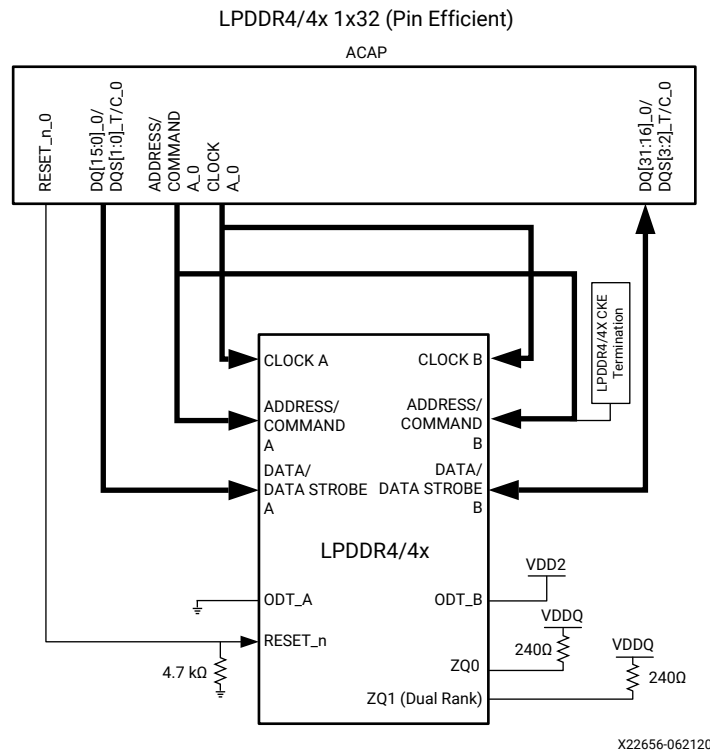
Figure 77: Nibble utilization for 1x16 component interface (Non-Flipped)

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	D Q A	D Q A	sys_clk	Free	Free	A C A	A C A	Free	Free	Free	Free	Free	D Q A	D Q A	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free

X22574-101520

## Pin Efficient 1x32 Component Interface (Non-Flipped)

Figure 78: Connections for a Pin Efficient 1x32 LPDDR4/4x Interface



Nibble utilization for pin efficient 1x32 interface using one x32 component in the non-flipped configuration is shown in the following figure. For maximum interface performance use the 1x32 pinout described in the Figure titled "Nibble utilization for 1x32 component interface (Non-Flipped)". DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals. Refer to [Clocking](#) for System Clock details.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 79: Nibble utilization for pin efficient 1x32 component interface (Non-Flipped)

Nibble #	First Bank								Second Bank								Third Bank										
	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q B	D Q B	D Q B	D Q B	A C	D Q A	D Q A	D Q A	D Q A	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free

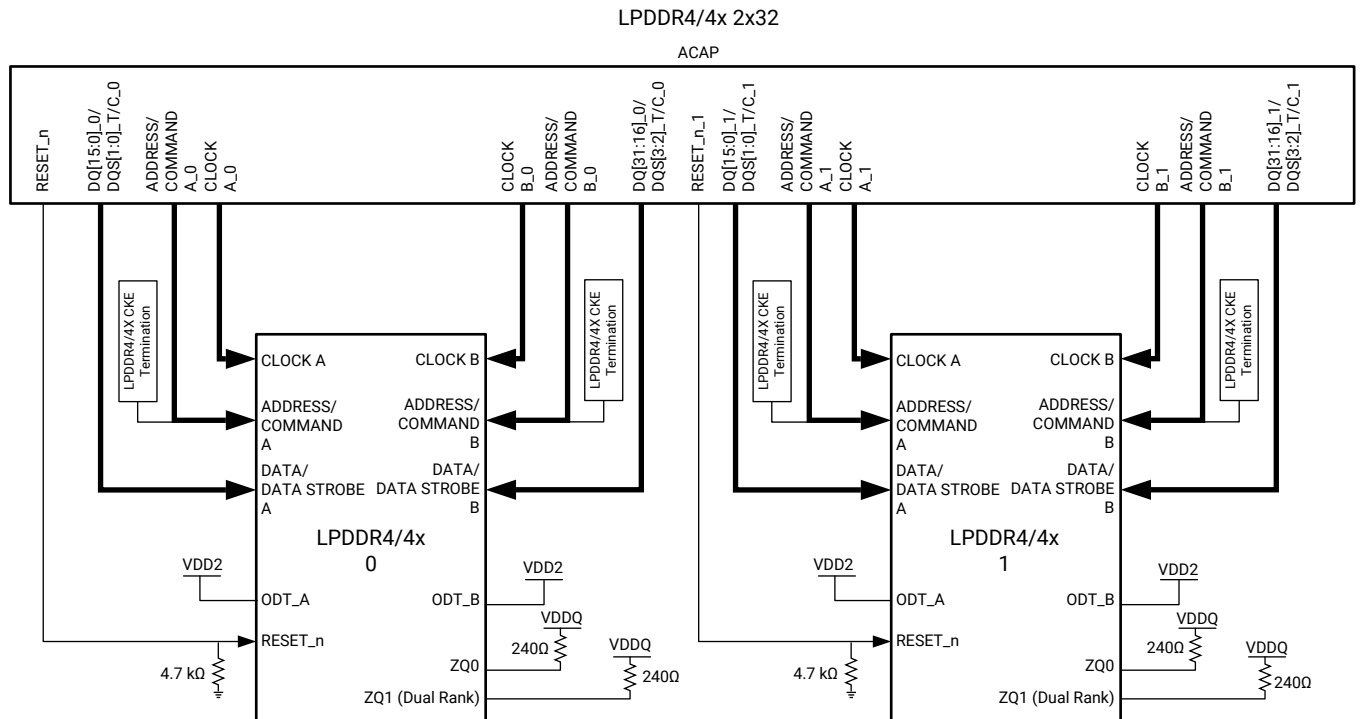
X22581-101520

## LPDDR4/4X Pinout for Component Interfaces (Flipped)

### 2x32 Component Interface (Flipped)

The integrated DDRMC can also be configured as two independent DDR interfaces of 16 or 32 data bits each. This section describes the Flipped 2x32 LPDDR4/4X interface.

Figure 80: Connections for a 2x32 LPDDR4/4x Interface



X21603-062120

Nibble utilization for 2x32 interface using two x32 components in the flipped configuration is shown in the following figure. This pinout provides the maximum interface performance compared to the 2x32 pin efficient version. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the Reference Clock pair, RESET\_n, and ALERT\_n signals. For a 1x32 interface all nibbles in the first Bank and nibbles 0, 1, 4, 5, and 8 in the second Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

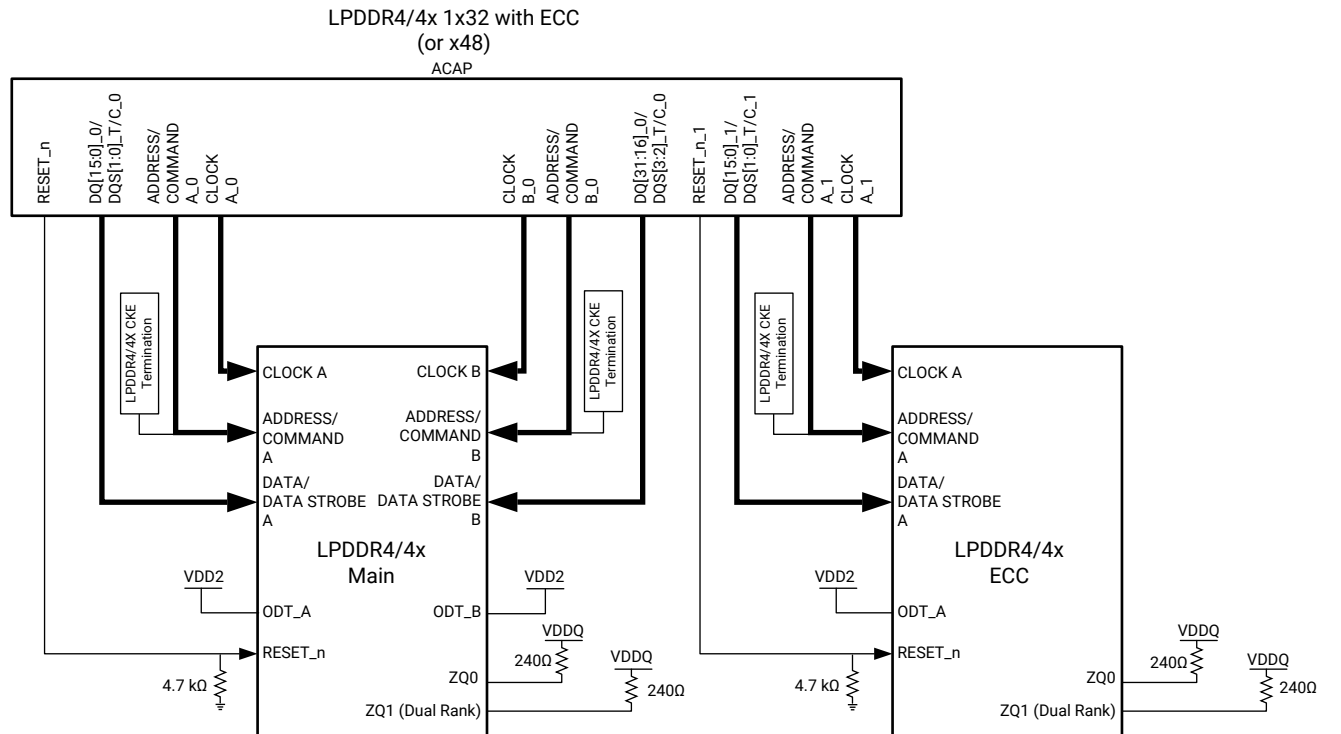
Figure 81: Nibble utilization for 2x32 component interface (Flipped)

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q  CH 1 B	D Q  CH 1 B	A C  CH 1 B	A C  CH 1 B	 Free  																						

X22234-101520

## 1x48 Component Interface (Flipped)

Figure 82: Connections for a 1x32 (or x48) with ECC LPDDR4/4x Interface



X21604-062120

Nibble utilization for 1x48 interface (or 1x32 with ECC) in the flipped configuration is shown in the following figure. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the Reference Clock pair, RESET\_n, and ALERT\_n signals. Note that the AC nibbles for the ECC device are on nibbles 6 and 7 in the first Bank and DQ nibbles for the ECC device are on nibbles 4 and 5 in the first Bank and nibble 0 and 1 in the second Bank. For a 1x32 interface all nibbles in the first Bank and nibbles 0, 1, 4, 5, and 8 in the second Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 83: Nibble utilization for 1x48 component interface (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	Free	Free	Free	DQ A 2	DQ A 2	AC A 2	AC A 2	DQ A 2	DQ A 2	DQ A	DQ A	Free	Free	Free	DQ B	DQ B	AC A	AC A	AC B	AC B	sys_clk	DQ A	DQ A	DQ B	DQ B			

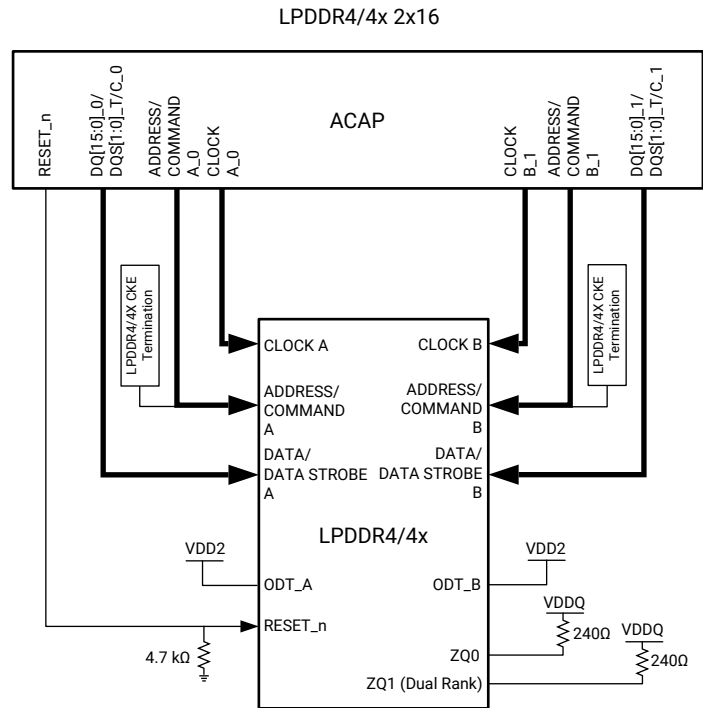
X22235-101520



## 2x16 Component Interface (Flipped)

The integrated DDRMC can also be configured as two independent DDR interfaces of 16, or 32 data bits each. This section describes the Flipped 2x16 LPDDR4/4X interface.

Figure 84: Connections for a 2x16 LPDDR4/4x Interface



X21605-062120

Nibble utilization for 2x16 interface using a x32 component in the flipped configuration is shown in the following figure. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the Reference Clock pair, RESET\_n, and ALERT\_n signals. For a 1x16 interface nibbles 4, 5, 6, 7, and 8 in the second Bank and nibbles 2, 3, 6, and 7 in the third Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 85: Nibble utilization for 2x16 component interface (Flipped)

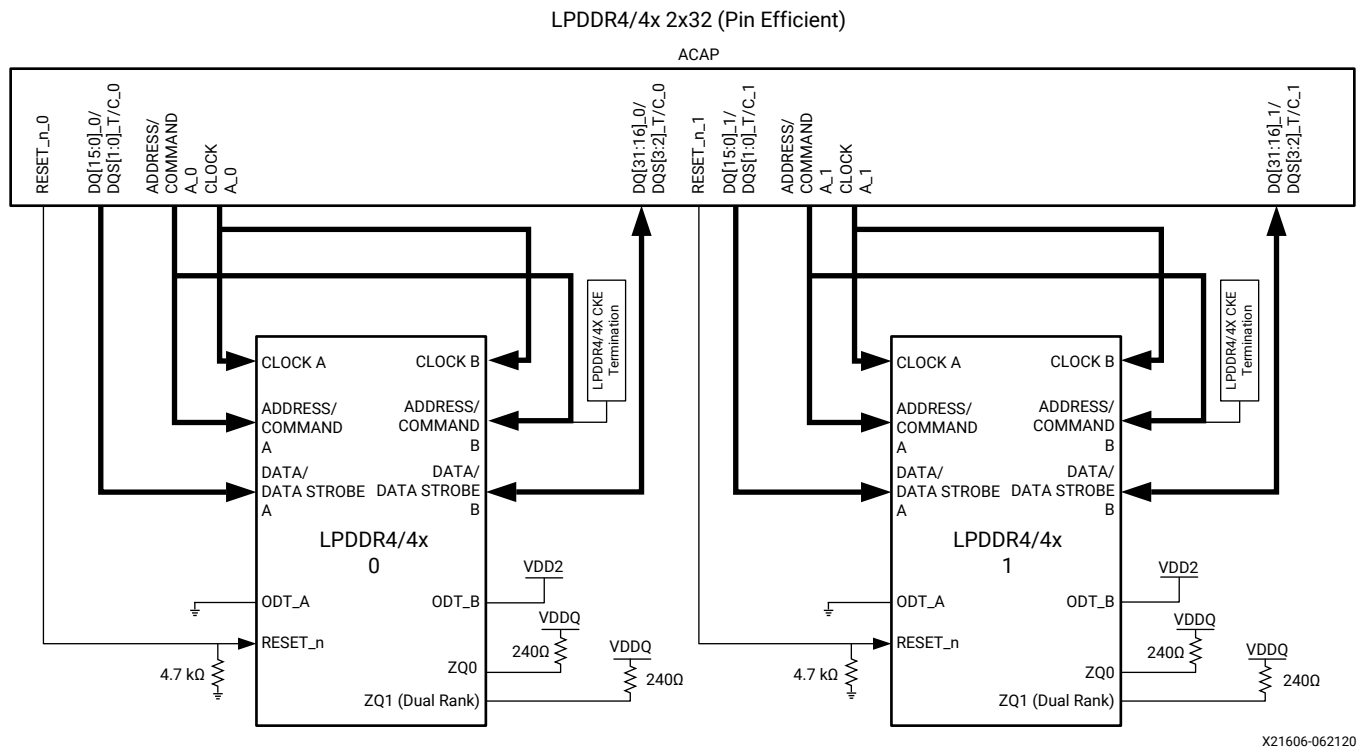
	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	D Q	D Q	Free	Free	Free	D Q	D Q	A C	A C	A C	A C	sys_clk	D Q	D Q	D Q	D Q			
												CH 0	CH 0				CH 1	CH 1	CH 0	CH 0	CH 1	CH 1		CH 0	CH 0	CH 1	CH 1			

X22236-101520

## Pin Efficient 2x32 Component Interface (Flipped)

The integrated DDRMC can also be configured as two independent DDR interfaces of 16 or 32 data bits each. This section describes the Flipped pin efficient 2x32 LPDDR4/4X interface.

Figure 86: Connections for a Pin Efficient 2x32 LPDDR4/4x Interface



Nibble utilization for pin efficient 2x32 interface using two x32 components in the flipped configuration is shown in the following figure. For maximum interface performance use the 2x32 pinout described in Figure 30. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the Reference Clock pair, RESET\_n, and ALERT\_n signals. For a 1x32 interface all nibbles in the second Bank and all nibbles except 8 in the first Bank would be free.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

Figure 87: Nibble utilization for pin efficient 2x32 component interface (Flipped)

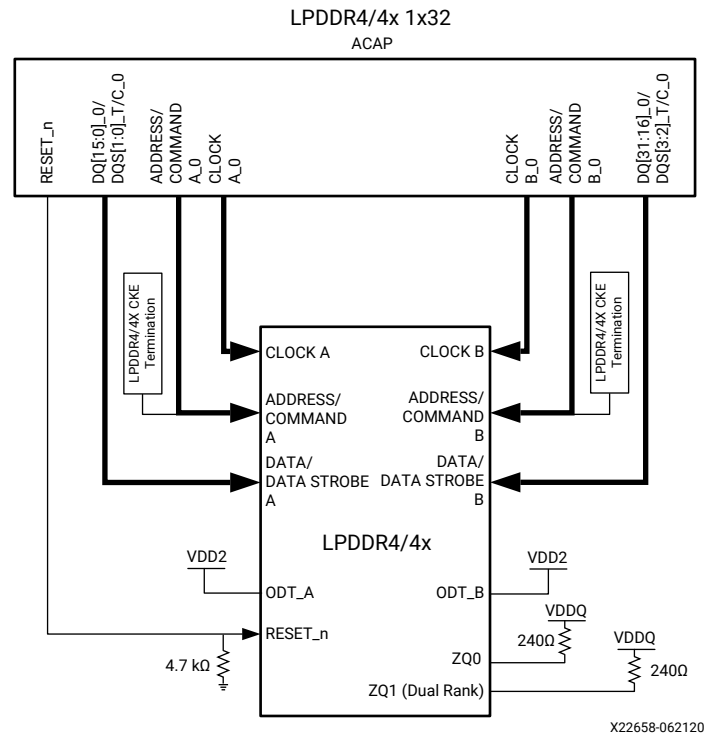
	First Bank								Second Bank								Third Bank										
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free	D Q CH 1 B	D Q CH 1 B	D Q CH 1 A	D Q CH 1 A	A C CH 1 A	D Q CH 1 B	D Q CH 1 B	D Q CH 1 A	D Q CH 1 A	D Q CH 0 A	D Q CH 0 A	D Q CH 0 A	D Q CH 0 A	A C CH 0 A	D Q CH 0 B	D Q CH 0 B	D Q CH 0 B	D Q CH 0 B

X22237-101520

## LPDDR4/4X Component Pinout (Flipped)

### 1x32 Component Interface (Flipped)

Figure 88: Connections for a 1x32 LPDDR4/4x Interface



Nibble utilization for 1x32 interface using one x32 component in the flipped configuration is shown in the following figure. This pinout provides the maximum interface performance compared to the 1x32 pin efficient version. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

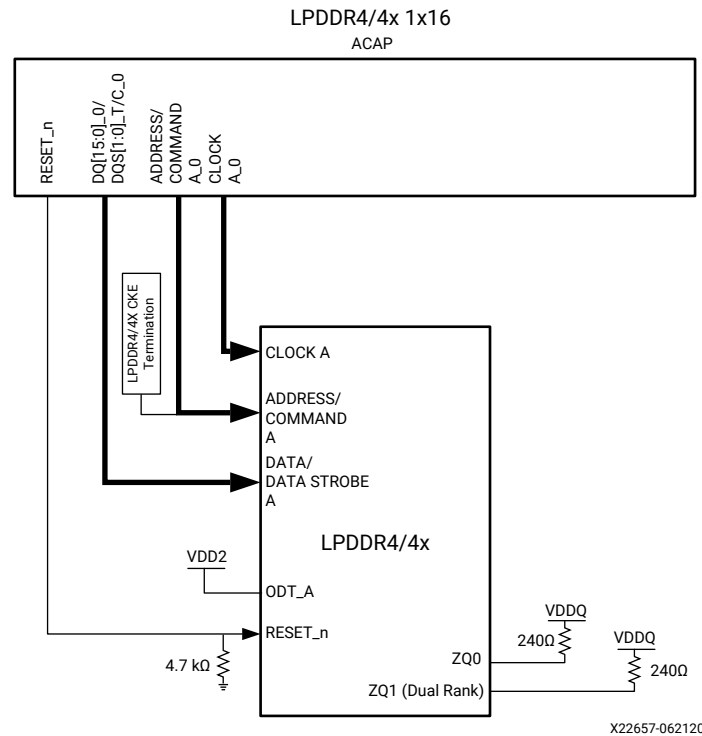
Figure 89: Nibble utilization for 1x32 component interface (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	D Q A	D Q A	Free	Free	Free	D Q B	D Q B	A C A	A C A	A C B	A C B	sys_clk	D Q A	D Q A	D Q B	D Q B			

X22575-101520

## 1x16 Component Interface (Flipped)

Figure 90: Connections for a 1x16 LPDDR4/4x Interface



X22657-062120

Nibble utilization for 1x16 interface in the flipped configuration is shown in the following figure. DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.

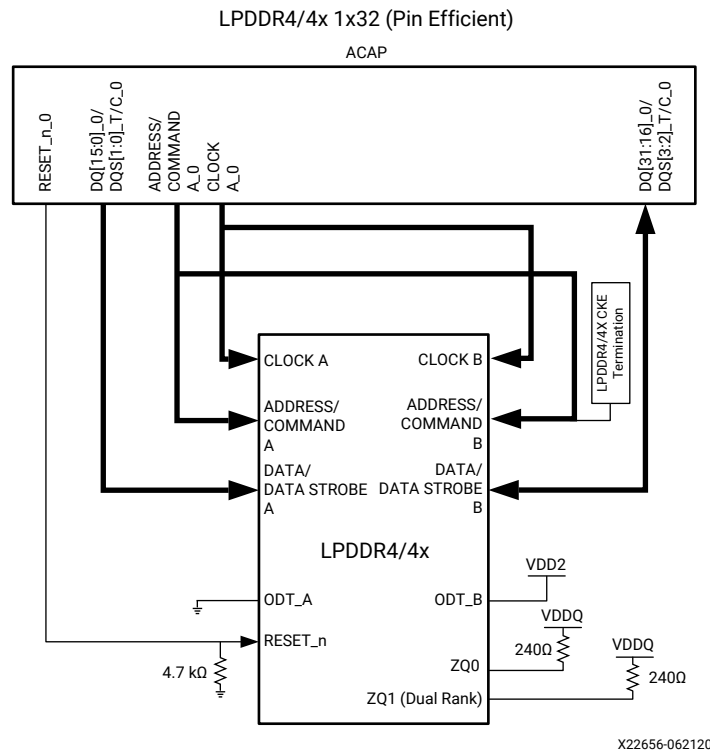
Figure 91: Nibble utilization for 1x16 component interface (Flipped)

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	D Q A	D Q A	Free	Free	Free	Free	Free	A C A	A C A	Free	Free	sys_clk	D Q A	D Q A	Free	Free			

X22579-101520

## Pin Efficient 1x32 Component Interface (Flipped)

Figure 92: Connections for a Pin Efficient 1x32 LPDDR4/4x Interface



Nibble utilization for pin efficient 1x32 interface using one x32 component in the flipped configuration is shown in the following figure. For maximum interface performance use the 1x32 pinout described in the Figure titled "Nibble utilization for 1x32 component interface (Flipped)". DQ indicates data nibbles, AC indicates Address/Command/Control nibbles, sys\_clk indicates a nibble comprising the System Clock pair, RESET\_n, and ALERT\_n signals.



**IMPORTANT!** The nibble utilization figure is based on the fixed pinout output by Vivado for this configuration.



Figure 93: Nibble utilization for pin efficient 1x32 component interface (Flipped)

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	D Q A	D Q A	D Q A	D Q A	A C B	D Q B	D Q B	D Q B	D Q B

X22580-101520

## Pinout Options for Future Expansion

This section provides PCB designers with guidance on how to ensure their DDRMC pinout is sufficient when considering future memory topology expansions like additional ranks, slots, or transitioning to 3DS devices. The Versal® DDRMC pinout behavior is a departure from previous generations as each memory technology has a unique set of fixed pinouts and the final pin map is dependent on the memory topology and the expansion options. This section describes the different pinouts supported for different memory technologies, how they may change per the topology, and how to ensure the DDRMC pinout going to fabrication is correct with future expansion in mind.



**RECOMMENDED:** You are encouraged to try the *Obtaining and Verifying Versal ACAP Memory Pinouts tutorial* available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.

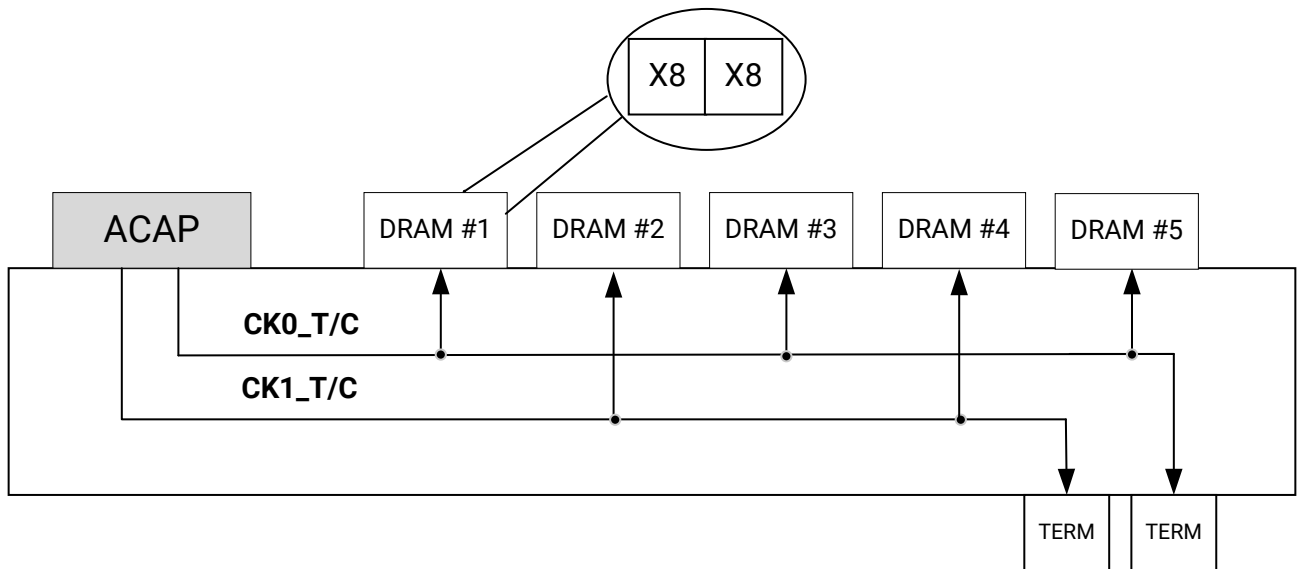
### DDR4 Dual CK Configuration

DDR4 single-rank component interfaces with more than nine CK loads will require two CK pairs to be routed on the PCB. Two CK pairs are required to mitigate signal integrity issues resulting from ten or more loads. The integrated memory controller IP outputs two CK pin pairs for single-rank component interfaces resulting in ten or more CK loads. Refer to the *Versal ACAP PCB Design User Guide* ([UG863](#)) for details on PCB design guidelines.

A 72-bit single-rank, x16 twin die (or dual die) component interface will have ten CK loads. Each x16 twin die (or dual-die) component has two x8 die resulting in two CK loads. The following figure shows the routing guideline for the two CK pairs.

Figure 94: **DDR4 2CK Single-Rank Configuration**

2 CK pairs with DDR4 Single-Rank, x16 DDP (x8 per die) Component Interface



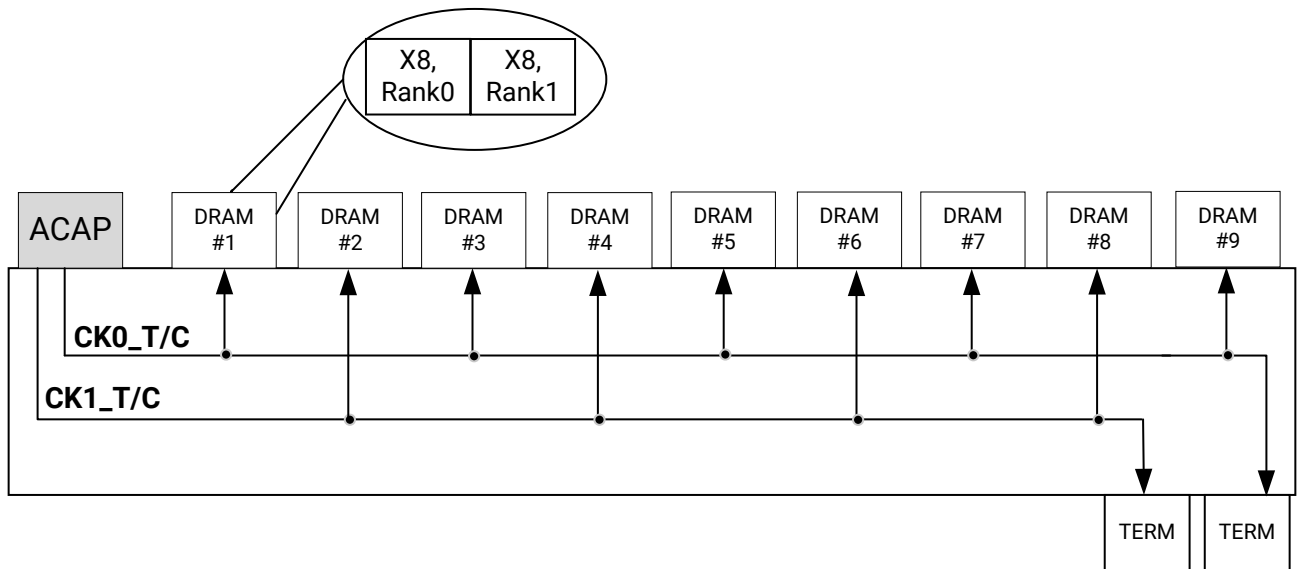
X25088-021721

DDR4 dual-rank component interfaces with more than ten CK loads will require two CK pairs to be routed on the PCB. Two CK pairs are required to mitigate signal integrity issues resulting from more than ten loads. The integrated memory controller IP outputs two CK pin pairs for dual-rank component interfaces resulting in ten or more CK loads. Refer to the *Versal ACAP PCB Design User Guide* ([UG863](#)) for details on PCB design guidelines.

A 72-bit dual-rank, x8 twin die (or dual-die) component interface will have eighteen CK loads. Each x8 twin die (or dual-die) component has two x8 die resulting in two CK loads. The following figure shows the routing guideline for the two CK pairs.

Figure 95: **DDR4 2CK Dual-Rank Configuration**

2 CK pairs with DDR4 Dual-Rank, x8 DDP (x8 per die) Component Interface



X25089-021721

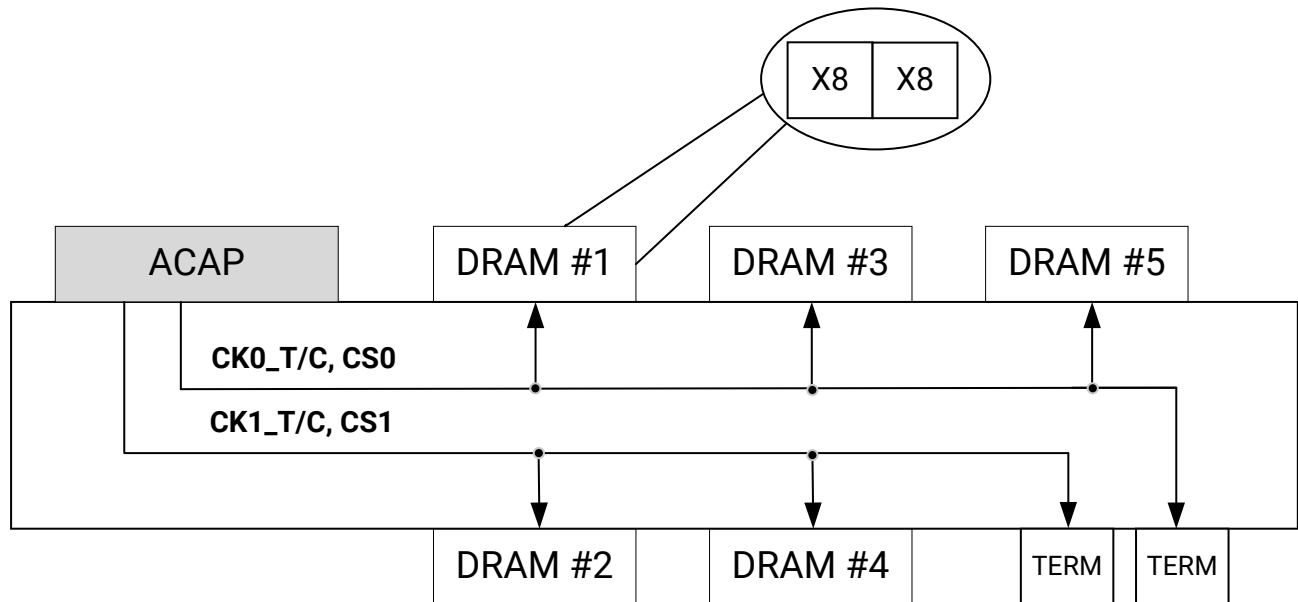
## Clamshell Topology

Clamshell topology saves the component area by placing them on both sides (top and bottom) of the board to mimic the address mirroring concept of multi-rank DIMMs. Address mirroring improves the signal integrity of the address and control ports while also making PCB routing easier. The clamshell topology is only supported for DDR4 single-rank components. The Clamshell option is available on the DDR Memory tab of the NoC configuration GUI.

The components are split into two categories called non-mirrored and mirrored. One additional chip select signal is added to the design for the mirrored components as shown in the following figure. Note that the figure in this example shows two CK pairs in addition to two chip select signals because the load on one CK pair would be greater than nine die with five single-rank, x16 DDP devices. Greater than nine loads on a CK pair will result in signal integrity issues. Refer to the *Versal ACAP PCB Design User Guide (UG863)* for details on PCB design guidelines.

Figure 96: Clamshell Configuration Example

Clamshell Configuration: 2 CS and 2 CK pairs with DDR4 Single-Rank, x16 DDP (x8 per die) Component Interface



X25057-021721

## DDR4 Interfaces

The Versal DDRMC supports multiple DDR4 topologies and with each configuration there is a different set of PCB expansion options. Fundamentally each DDR4 topology has a unique pin map, so when it comes to PCB expansion paths you must use the same basic topology type like Dual Channel Components, Single Channel Components, UDIMM/SODIMM, or RDIMM/LRDIMM. For example, a Dual Channel Component interface must remain a Dual Channel Component interface in the future or an RDIMM/LRDIMM interface must remain an RDIMM/LRDIMM interface in the future. The pin maps do not support switching a topology from Single Channel Components to UDIMM, or UDIMM to an LRDIMM, or Dual Channel Components to Single Channel Components. It is also important to consider different DDR4 controller options if the future use case requires a new feature like Command/Address Parity, ECC, Pinout Swapping, or if the DM/DBI functionality will change.

### Dual Channel DDR4 Component Interfaces

Available options in dual channel topology:

- Dual Channel 16-bit

- Dual Channel 24-bit as 16-bit Data with 8-bit ECC
- Dual Channel 32-bit

PCB Expansion Options:

- Optimum

All Dual Channel DDR4 Component interfaces will always have the Optimum option. This is because all configurations follow the same basic pinout while more Nibbles are used as the data widths get larger. The Dual Channel DDR4 Component pinouts are not compatible with Single Channel DDR4 Component pinouts because of the differences in the input reference clock Nibble and the Command/Address/Control (CAC) Nibbles. The CAC Nibbles for Dual Channel interfaces map to Data Nibbles for Single Channel interfaces so there is no way for one hardware design to satisfy the needs for both types of topologies. In addition, Dual Channel topologies only support a single DDR4 interface clock output since any supported configuration will be within the DDP Deep loading guidelines. The default Optimum pinout will automatically support DDP Deep rank expansion or 3DS device support. There is no support for DDP Deep 3DS Single Channel or Dual Channel component interfaces. DDP Deep 3DS components are supported in RDIMM/LRDIMM applications.

The Dual Channel DDR4 Component interface pinouts optimally pack the Triplet by placing the utilized Nibbles starting first in the Left bank and expanding to the Right bank as the data width gets larger. The figure below shows the Dual Channel 16-bit bank packing with the unused Nibbles on the Right side of the bank.

**Figure 97: Dual Channel 16-Bit (2 x 16) Component Pinout with DDP Deep and 3DS Support**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free			
Signals	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 0	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	CH 1	Free	Free	Free	Free	Free	Free	Free	Free	Free			

X22225-101420

The Dual Channel 24-bit (16-bit Data + 8-bit ECC) interface pinout follows the Dual Channel 16-bit pinout with the additional Data Nibbles populating in the Third Bank.

**Figure 98: Dual Channel 24-Bit (2 x 24) Component Pinout with DDP Deep and 3DS Support**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ CH 0	DQ CH 0	DQ CH 0	DQ CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	DQ CH 1	DQ CH 1	DQ CH 1	DQ CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	DQ CH 0	DQ CH 0	DQ CH 1	DQ CH 1	sys_clk	Free	Free	Free	Free			

X22226-101420

The Dual Channel 32-bit interface pinout follows the Dual Channel 24-bit pinout with the additional Data Nibbles populating in the Third Bank.

**Figure 99: Dual Channel 32-bit (2 x 32) Component Pinout with DDP Deep and 3DS Support**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ CH 0	DQ CH 0	DQ CH 0	DQ CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	AC CH 0	DQ CH 1	DQ CH 1	DQ CH 1	DQ CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	AC CH 1	DQ CH 0	DQ CH 0	DQ CH 1	DQ CH 1	sys_clk	DQ CH 0	DQ CH 0	DQ CH 1	DQ CH 1			

X22224-101420

## Single Channel DDR4 Component Interfaces

Available options in single channel topology:

- Single Channel 16-bit
- Single Channel 24-bit as 16-bit Data with 8-bit ECC
- Single Channel 32-bit
- Single Channel 40-bit as 32-bit Data with 8-bit ECC
- Single Channel 64-bit

- Single Channel 72-bit as 64-bit Data with 8-bit ECC

PCB Expansion Options:

- Optimum
- CLK Expansion

For Single Channel DDR4 Component interfaces there are two distinct pinouts. The first is Optimum which efficiently packs the Triplet based on the data width. The other is CLK Expansion which is fundamentally a different pinout than the Optimum with two DDR4 CLK outputs generated for DDP Deep topologies. For most applications the Optimum pinout is acceptable and will allow for future expansion if adding ECC, 3DS devices, or expanding the data width. The CLK Expansion option is only required when migrating to DDP Deep devices in the future. The current Versal DDR4 solution only supports x8 DDP Deep devices and two DDR4 CLK outputs are only required when five or more components are present as with data widths of 40-bits, 64-bits, or 72-bits. It is not possible to generate any other Single Channel topology which requires two CLK outputs at this time. The logical ranks found in 3DS devices do not add additional loading like DDP Deep devices so there is no need for CLK expansion with these components. There is no support for DDP Deep 3DS Single Channel or Dual Channel component interfaces. DDP Deep 3DS components are supported in RDIMM/LRDIMM applications.

The Optimum single clock output pinout leaves free Nibbles open on the far Right Bank, and depending on your data width, more Nibbles will be available moving Right to Left. The CLK Expansion pinout has two free Nibbles, one in the first Bank and one in the last Bank. The CLK Expansion pinout will have more free Nibbles starting in the Right bank and moving Left as your data width decreases. Both pinouts support 3DS and Rank expansion, but the CLK Expansion pinout must be used for DDP Deep devices with five or more placements. If the DDRMC configuration already has two CLKs enabled then the only pinout option will be Optimum which is the CLK Expansion pinout.

**Figure 100: Single Channel up to 72-Bit DDR4 Component Pinout with Single DDR4 Clock Output**

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ 1	DQ 1	DQ 0	DQ 0	AC	AC	AC	AC	AC	DQ 3	DQ 3	DQ 4	DQ 4	sys_clk	DQ 2	DQ 2	DQ 5	DQ 5	DQ 6	DQ 6	DQ 7	DQ 7	Free	DQ 8	DQ 8	Free	Free			

X22218-102320



**Figure 101: CLK Expansion Single Channel up to 72-Bit DDR4 Component Pinout with Two DDR4 Clocks**

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ 0	DQ 0	Free	AC	AC	AC	AC	AC	AC	DQ 1	DQ 1	DQ 2	DQ 2	sys_clk	DQ 3	DQ 3	DQ 4	DQ 4	DQ 5	DQ 5	DQ 6	DQ 6	Free	DQ 7	DQ 7	DQ 8	DQ 8

X22219-102320

## DDR4 UDIMM and SODIMM Interfaces

PCB Expansion Options:

- Optimum
- Rank Expansion

The Versal DDR4 controller supports single slot DDR4 UDIMM and SODIMM topologies which can be single rank or dual rank. From a pinout and memory topology perspective the DDR4 UDIMM and SODIMM implementations are identical and will be referred to as UDIMM/SODIMM since there is no difference aside from the physical slot and form factor. UDIMM/SODIMM pinouts differ from component interfaces in the CAC bus signals placement in the Center Bank for easier routing to the connector. For UDIMM/SODIMM topologies there are two distinct pin maps. The first is Optimum which only supports single rank devices and leaves three free Nibbles open in the Right bank. The second is Rank Expansion which supports both single rank and dual rank UDIMM/SODIMM devices. If there is a need for increased memory capacity in the future then the Rank Expansion option must be selected, otherwise, if only single rank devices will be used then this can be left at Optimum. If the UDIMM/SODIMM configuration is dual rank then the only pinout option will be Optimum which is the Rank Expansion pinout. UDIMM/SODIMM standards do not support 3DS devices.

Figure 102: Single Rank DDR4 UDIMM/SODIMM 64-bit or 72-bit Pinout

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	Free	Free			

X22214-050319

Figure 103: Rank Expansion Single or Dual Rank DDR4 UDIMM/SODIMM 64-bit or 72-bit Pinout

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ				

X22215-050319

## DDR4 RDIMM and LRDIMM Interfaces

PCB Expansion Options:

- Optimum
- Rank Slot and Stack Height Expansion

The Versal DDR4 controller supports single slot, dual slot, single rank, and dual rank RDIMM/LRDIMM topologies as well as single slot quad rank LRDIMM devices. For these buffered DIMMs there are two pin maps; Optimum or Rank Slot and Stack Height Expansion.

Regardless of the pinout option, the existing pinout will be compatible with x4 and x8 devices without any user intervention.

When the DDRMC is configured for a single rank, single slot, non-3DS based RDIMM device, the default pinout is an efficiently packed Optimum which will only work for these types of devices. The single rank single slot RDIMM is the most efficient buffered DIMM pinout but does not support any additional rank, slot, or 3DS stack height expansion, nor does it support LRDIMM devices. If future hardware revisions may require additional ranks, slots, switch to 3DS devices, or switch to an LRDIMM device then the Rank Slot and Stack Height Expansion option must be selected.

The Rank Slot and Stack Height Expansion option supports all multi-rank, multi-slot, 3DS or standard component based RDIMM/LRDIMM topologies. When an LRDIMM device is selected the only pinout option is Optimum which is the Rank Slot and Stack Height Expansion pinout. When any dual rank or dual slot RDIMM device is selected the only pinout option is Optimum which is the Rank Slot and Stack Height Expansion pinout. When any 3DS based RDIMM or LRDIMM topology is selected then the only pinout option is Optimum which is the Rank Slot and Stack Height Expansion pinout. If your existing hardware design used an LRDIMM device or any multi-rank, or multi-slot, or 3DS based RDIMM device then the hardware already has all the necessary sites allocated for future PCB expansion.

**Figure 104: Single Rank Single Slot Standard Component RDIMM Pinout**

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	Free	Free

X22212-050319

**Figure 105: Rank Expansion Single/Dual Slot Single/Multi-Rank 3DS/Standard RDIMM/LRDIMM Pinout**

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ	DQ	DQ	DQ	sys_clk	DQ	DQ	DQ	DQ	AC	AC	AC	AC	AC	AC	AC	DQ	DQ	DQ	DQ	DQ	Free	DQ	DQ	DQ	DQ	

X22210-050319

## Additional Considerations for DDR4 Interfaces

### ECC Enabled Interfaces

When ECC is enabled then an additional data byte will be generated in the DDRMC. The ECC byte includes 8-bits of data, a DQS pair, and a DM/DBI pin for x8 or x16 devices. If future hardware designs will be using ECC, then the pinout must be generated with ECC enabled. During hardware layout ensure these are routed to the correct locations on the board. When running in application the ECC function can be disabled and hardware will operate correctly as the ECC byte is a functional "don't care".

### DRAM Command/Address Parity

The Command/Address Parity function enables two additional pins in the DDRMC. These are the output only PAR pin and the input only ALERT\_N pin. If future hardware designs will be using the Command/Address Parity function, then generate the pinout with this feature enabled. During hardware layout ensure these sites are routed to the correct locations on the DDR4 components or DIMM connectors. When running in application the Command/Address parity feature can be disabled and hardware will operate correctly as these signals are a functional "don't care".

## DM/DBI

If the DDRMC configuration has Write DM/DBI or Read DBI functionality disabled, then the IP will not generate these ports in the pinout. In the scenario when future hardware applications need Write Data Mask or Write/Read DBI functionality, then the pinout must be generated with these functions enabled. Write DM and Write/Read DBI functionality is only available for x8 and x16 devices. For RDIMM/LRDIMM devices the JEDEC standard connector handles the mapping for x4 and x8-based devices which includes the DM/DBI pin for x8 components, so no additional care is required in this buffered DIMM scenario. If required for future applications then these functions must be enabled when generating the pinout and ensure these signals are routed to the correct sites on the DDR4 components or DIMM connector.

## Pinout Swapping

If any pinout was generated with Pinout Swapping enabled, then any future hardware compatible expansions must also have Pinout Swapping enabled.

## *Generating Future Expansion Pinouts*

When designing a board with the expectation there will be changes to the memory topology in the future, then the pinout must be generated targeting the future memory topology and application features. After the hardware is designed the DDRMC can be configured for the current memory topology and application features but it also must select the appropriate pinout option.

### Example

An example of this would be generating a pinout for a 72-bit (64-bit + 8-bit ECC) dual rank SODIMM but the current application only requires a 64-bit single rank SODIMM. The pinout was generated to target the maximum future configuration which was then implemented in hardware. When hardware arrives and the first use case only requires a 64-bit single rank SODIMM, then the DDRMC would be configured to a single rank 64-bit SODIMM with the Rank Expansion option selected. If the DDRMC were configured to a 64-bit single rank SODIMM with the Optimum option selected, then this would be a fundamentally different pinout and it would not pass calibration. If the DDRMC were configured correctly for Rank Expansion and the single rank SODIMM is installed, then the hardware will function correctly.

## Dual Channel Component Interfaces

The minimum Dual Channel DDR4 Component interface configuration is 2 x 16-bit while the maximum is 2 x 32-bit. For all Dual Channel DDR4 Component interface configurations the default pinout is Optimum because fundamentally they all follow the same pinout except for the number of enabled data bytes or the additional Control signals for DDP Deep or 3DS based devices. If future expansion is expected, then generate the pinout based on the maximum 2 x 32-

bit interface configuration with the appropriate ECC (if applicable), Command/Address Parity, DM/DBI, and Pinout Swapping options. If using 3DS or DDP Deep devices in the future, then the maximum pinout must have the appropriate options enabled to ensure the additional control signals are generated and routed to the appropriate package pins. JEDEC standards include compatibility for DDP Deep and 3DS based packages with a single hardware design.

## Single Channel Component Interfaces

The minimum Single Channel DDR4 Component interface configuration is 16-bits while the maximum is 72-bits with ECC. For most configurations there will be two pinout options; either Optimum or CLK Expansion. CLK Expansion is only required for DDP Deep applications with five or more devices. If the DDRMC configuration already has two DDR4 output clocks enabled then the only pinout option will be Optimum. When planning for future memory topology expansions the maximum configuration will depend on whether DDP Deep devices are required. If they are not required then a single rank 72-bit interface with the appropriate 3DS, ECC, Command/Address Parity, DM/DBI, and Pinout Swapping options will allow for DQ width, 3DS device support, and ECC expansion when selecting the Optimum pinout option. If the future topology requires DDP Deep devices then the maximum pinout must be generated with a Dual Rank configuration to ensure the additional Control signals are included in the pinout. If CLK expansion is required then generate the pinout as Dual Rank with two DDR4 output clocks enabled if it is not already enabled in the tools. DDP Deep 3DS Component interfaces are not supported. JEDEC standards include compatibility for DDP Deep and 3DS based packages with a single hardware design so it is possible to generate a single pinout compatible with either type of device.

## DDR4 UDIMM and SODIMM Interfaces

The minimum UDIMM/SODIMM interface configuration is single rank 64-bits while the maximum is dual rank 72-bits. For all single rank configurations there will be two pinout options; either Optimum or Rank Expansion. For all dual rank configurations the only option will be Optimum because it is already using the Rank Expansion pinout. If future applications will only use single rank devices then the pinout can be left at Optimum. If future applications require single or dual-rank devices then Rank Expansion is required. Generate the pinout for a 72-bit DQ width for future ECC support. Ensure the appropriate Command/Address Parity, DM/DBI, and Pinout Swapping options are selected when generating the pinout. Generate the maximum future pinout as Dual-Rank to ensure that the additional DDR4 output clock and control signals required for the second rank are included in the hardware design.

## DDR4 RDIMM and LRDIMM Interfaces

The minimum RDIMM/LRDIMM interface configuration is single rank, single slot RDIMM with 72-bits while the maximum is multi-rank multi-slot RDIMM or any LRDIMM with 72-bits. There are two pinout options for buffered DIMMs; either Optimum or Rank Slot and Stack Height Expansion. The minimum and most efficient Optimum pinout option only applies for single rank single slot non-3DS based RDIMMs. If future applications will still only use single rank single slot non-3DS based RDIMMs then the Optimum option can be selected, otherwise for any other

application the Slot Rank and Stack Height option must be selected. When any multi-rank, multi-slot or 3DS based RDIMM configuration is selected the only option is Optimum because it already uses the Rank Slot and Stack Height Expansion pinout. When any LRDIMM device is selected the only option is Optimum because it already uses the Rank Slot and Stack Height Expansion pinout. When using any multi-rank multi-slot or 3DS based RDIMM or any LRDIMM device then the existing pinout automatically supports all future RDIMM and LRDIMM topology expansions as well as x4 and x8 device compatibility.

To ensure every future signal is routed correctly the Write DM or any Write/Read DBI function must be enabled to generate the DM/DBI ports. These are critical for x4/x8 compatibility. Also note any requirements for 3DS devices, quad rank, or dual slot topologies and generate the maximum pinout appropriately to ensure the additional DDR4 output clocks and control signals are generated and included in the hardware design. Enable the Command Address parity feature when generating the pinout if it is required.

## LPDDR4/4X Interfaces

The Versal DDRMC supports either Single Channel or Dual Channel LPDDR4 interfaces. Most Single Channel and Dual Channel LPDDR4 pinouts are compatible with each other for future expansion but additional care needs to be taken in some configurations. It is also important to consider different LPDDR4 controller options like DM/DBI functionality, ECC, and Pinout Swapping when planning for future PCB expansion. A typical 32-bit LPDDR4 device operates as two channels, each 16-bits wide, and each has its own Command/Address bus. When using a 16-bit LPDDR4 device it has a single 16-bit data channel. When using a 48-bit LPDDR4 topology there are three 16-bit data channels. These must not be confused with the DDRMC configuration with regard to Single or Dual Channel. Single Channel refers to the DDRMC operating as a single LPDDR4 interface. Dual Channel refers to the DDRMC operating as two LPDDR4 interfaces. A Single Channel 48-bit configuration is one LPDDR4 interface with three 16-bit data channels. A Dual Channel 32-bit configuration is two LPDDR4 interfaces, each with two 16-bit data channels.

### *Single Channel LPDDR4/4X Interfaces*

Available options in single channel topology:

- Single Channel 16-bit
- Single Channel 32-bit
- Single Channel 32-bit as 16-bit Data with 16-bit ECC
- Single Channel 48-bit as 32-bit Data with 16-bit ECC

PCB Expansion Options:

- Optimum

All Single Channel LPDDR4/4X interfaces will always give the Optimum option. This is because all configurations follow the same basic pinout while more Nibbles are used as the data width increases. As the data width expands from 16-bits to 32-bits or 32-bits to 48-bits additional Address/Command signals are generated for the additional LPDDR4/4X data channel. The Optimum pinout supports single or dual rank topologies. It is also possible for Single Channel interfaces to expand to Dual Channel interfaces. This is because the non-pin efficient Dual Channel 32-bit pinout is compatible with the individual LPDDR4/4X data channels along with their respective Command/Address bus. All Single Channel LPDDR4/4X interfaces will not be able to select the LP4 Pin Efficient option in the GUI. This ensures forward compatibility with other topologies.

Figure 106: Single Channel 16-bit (1x16) LPDDR4/4X Interface Pinout

	First Bank									Second Bank								Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	Free	Free	D Q A	D Q A	sys_clk	Free	Free	A C A	A C A	Free	Free	Free	Free	Free	D Q A	D Q A	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free

X22574-101520

Figure 107: Single Channel 32-bit LPDDR4/4X Interface Pinout

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q B	D Q B	D Q A	D Q A	sys_clk	A C B	A C B	A C A	A C A	D Q B	D Q B	Free	Free	Free	D Q A	D Q A	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free	Free

X22573-101520



**Figure 108: Single Channel 48-bit LPDDR4/4X Interface Pinout with DQA2 and ACA2 being the ECC Channel**

Nibble #	First Bank								Second Bank								Third Bank										
	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	DQ B	DQ B	DQ A	DQ A	sys_clk B	AC B	AC B	AC A	AC A	DQ B	DQ B	Free	Free	Free	DQ A	DQ A	DQ A 2	DQ A 2	AC A 2	AC A 2	DQ A 2	DQ A 2	Free	Free	Free	Free	Free

X22231-101520

## Dual Channel LPDDR4/4X Interfaces

Available options in dual channel topology:

- Dual Channel 16-bit
- Dual Channel 32-bit
- Dual Channel 32-bit as 16-bit Data with 16-bit ECC

PCB Expansion Options:

- Optimum

All Dual Channel LPDDR4/4X interfaces will always give the Optimum option but it is possible to get a unique pinout with a Dual Channel 32-bit Pin Efficient option. Fundamentally there are two different Dual Channel LPDDR4/4X pinouts. One is the Dual Channel 32-bit Pin Efficient which only supports single rank devices and the other is all Dual Channel pinouts. The Pin Efficient pinout is unique as it only supports single rank 2 x 32-bit interfaces. The standard Dual Channel pinouts will support single channel and dual channel, single rank or dual rank, topologies. The fundamental pinouts for the standard Dual Channel configuration supports 16-bit, 32-bit, and 32-bit as 16-bit Data and 16-bit ECC configurations. This is because all configurations follow the same basic pinout while more Nibbles are used as the data width increases. As the data width expands from 16-bit to 32-bit or 32-bit to 48-bit additional Address/Command signals are generated for the additional LPDDR4/4X data channel. The Optimum pinout supports single or dual rank topologies. It is also possible for Single Channel interfaces to expand to Dual Channel interfaces. This is because the non-pin efficient Dual Channel 32-bit pinout is compatible with the individual LPDDR4/4X data channels along with their respective Command/Address bus. All Single Channel LPDDR4/4X interfaces will not be able to select the LP4 Pin Efficient option in the GUI to ensure forward compatibility with other topologies.

Figure 109: Standard Dual Channel 2 x 16-bit LPDDR4/4X Interface Pinout

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	D Q  CH 1	D Q  CH 1	D Q  CH 0	D Q  CH 0	sys_clk  sys_clk 1	A C  CH 1	A C  CH 1	A C  CH 0	A C  CH 0	D Q  CH 1	D Q  CH 1																			

X22232-101520

Figure 110: Standard Dual Channel 2 x 32-bit LPDDR4/4X Interface Pinout

	First Bank									Second Bank									Third Bank								
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7
Signals	D Q	D Q	D Q	D Q	sys_clk	A C	A C	A C	A C	D Q	D Q	D Q	D Q	Free	D Q	D Q	D Q	D Q	A C	A C	D Q	D Q	Free	A C	A C	D Q	D Q
	CH 0 B	CH 0 B	CH 0 A	CH 0 A		CH 0 B	CH 0 B	CH 0 A	CH 0 A	CH 0 B	CH 0 B	CH 1 B	CH 1 B		CH 0 A	CH 0 A	CH 1 A	CH 1 A	CH 1 A	CH 1 A	CH 1 A	CH 1 A		CH 1 A	CH 1 A	CH 1 B	CH 1 B

X22230-101520

Figure 111: Pin Efficient Dual Channel 2 x 32-bit Single Rank LPDDR4/4X Interface Pinout

	First Bank										Second Bank										Third Bank									
Nibble #	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7	0	1	2	3	8	4	5	6	7			
Signals	D Q	D Q	D Q	D Q	A C	D Q	D Q	D Q	D Q	D Q	D Q	D Q	D Q	A C	D Q	D Q	D Q	D Q												
	CH 0 B	CH 0 B	CH 0 B	CH 0 B	CH 0	CH 0 A	CH 0 A	CH 0 A	CH 0 A	CH 1 A	CH 1 A	CH 1 B	CH 1 B	CH 1	CH 1 A	CH 1 A	CH 1 B	CH 1 B	Free	Free	Free	Free	sys_clk	Free	Free	Free	Free			

X22233-101520

The connection guidance figures shown below must be followed for feasibility of PCB expansion options.

Figure 112: **Connections for a 2x16 LPDDR4/4X Interface**

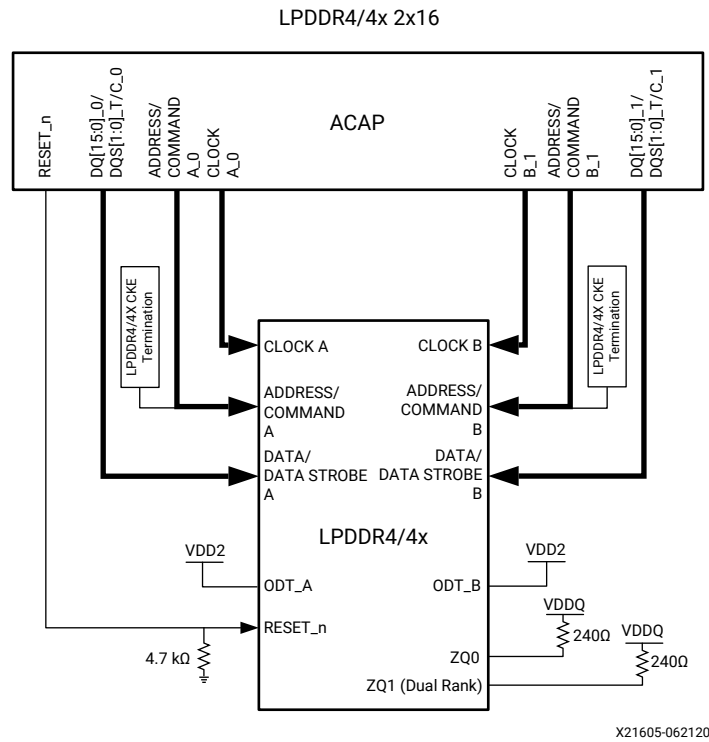
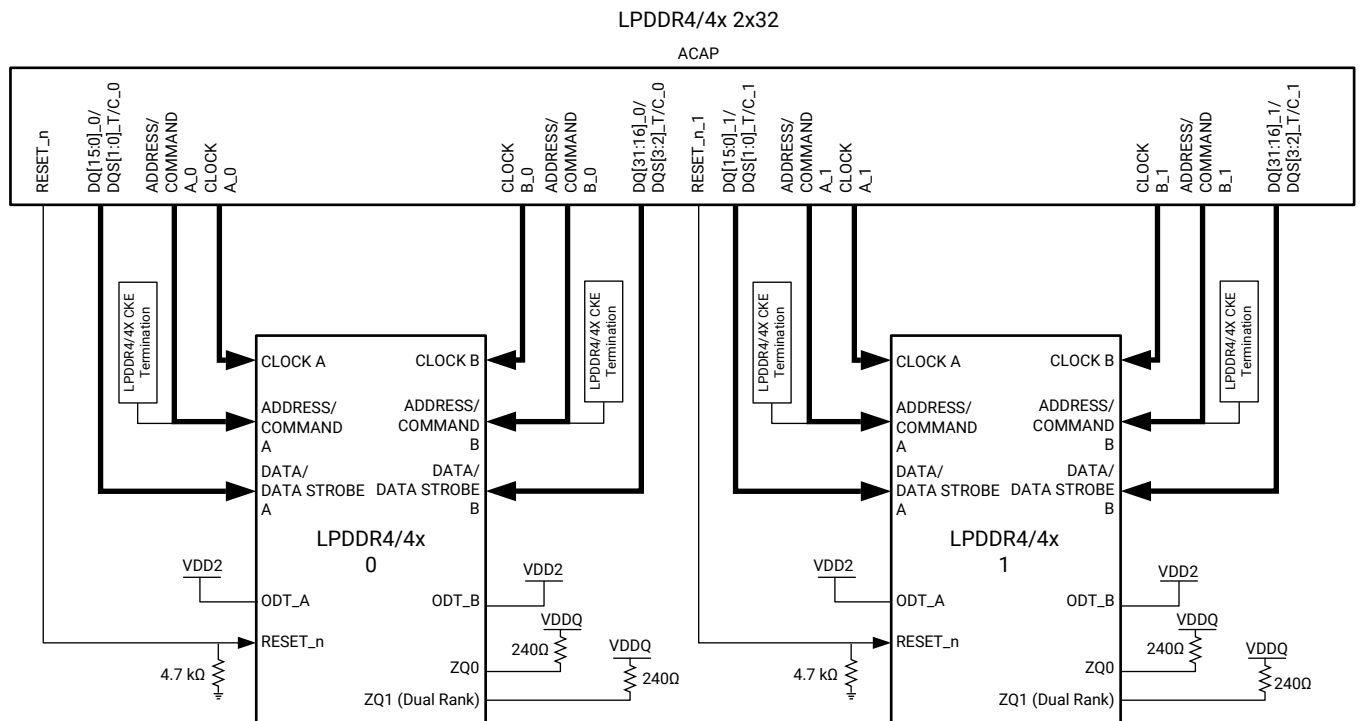
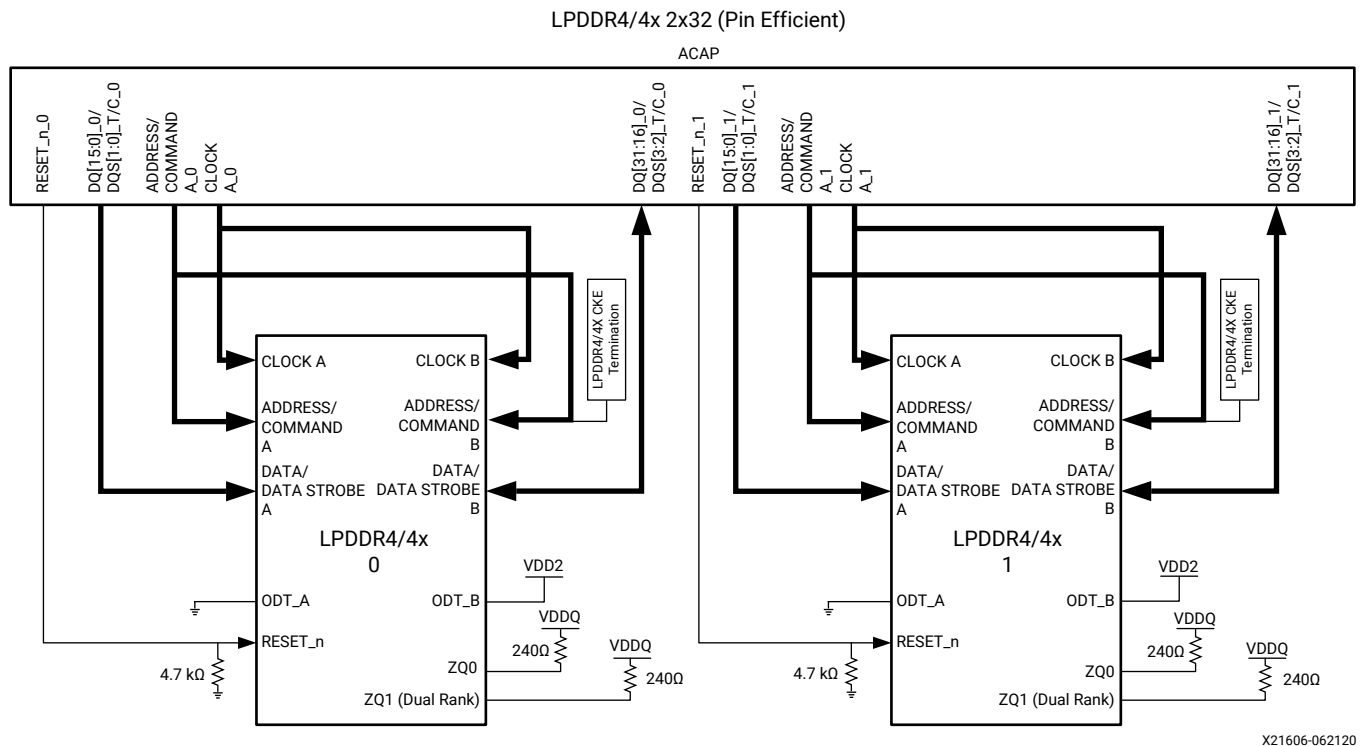


Figure 113: Connections for a 2x32 LPDDR4/4X Interface



X21603-062120

Figure 114: Connections for a Pin Efficient 2 x 32 Single Rank LPDDR4/4X Interface



## Generating Future Expansion Pinouts

Generating future expansion pinouts for LPDDR4/4X designs is a simpler than for DDR4 based designs. Only the Dual Channel 2 x 32-bit LPDDR4/4X Single Rank Pin Efficient pinout is unique and doesn't support other LPDDR4/4X configurations. The standard Optimum pinout which is not Pin Efficient will support future expandability for dual rank devices, enabling an ECC device, expanding single channel data widths, expanding dual channel data widths, and expanding from single channel to dual channel topologies. As long as the connection examples are followed multiple topologies can be supported with a single hardware design. While the logical net names change the hardware can remain unchanged. When generating the maximum pinout ensure settings like ECC, Write DM, DBI, and DDRMC Pinout Swapping settings are set as expected for the application. Refer to the connection diagrams for the first configuration and the expanded configuration to visualize the commonalities between them to ensure hardware is designed correctly. An easy way to demonstrate this concept it to look at the Single Channel 16-bit connection diagram, then the Dual Channel 2 x 16-bit connection diagram, then the Single Channel 1 x 32-bit connection diagram, then the Single Channel 1 x 48-bit connection diagram, and then finally a Dual Channel 2 x 32-bit connection diagram.

Figure 115: Connections for a 1x16 LPDDR4/4x Interface

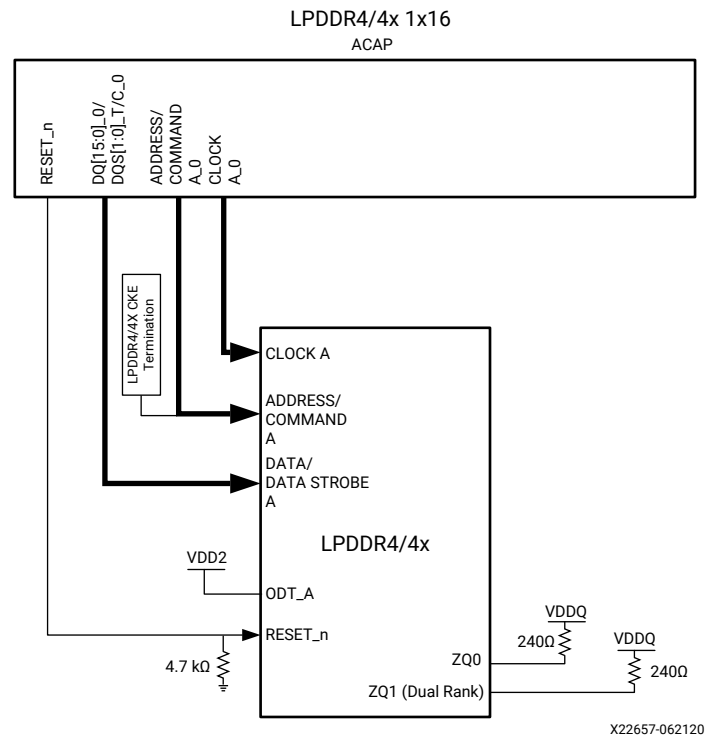
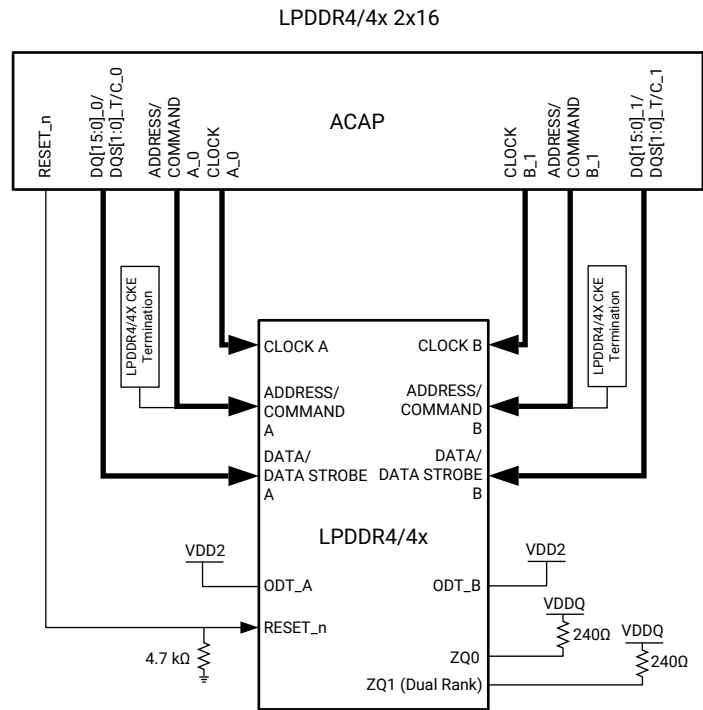
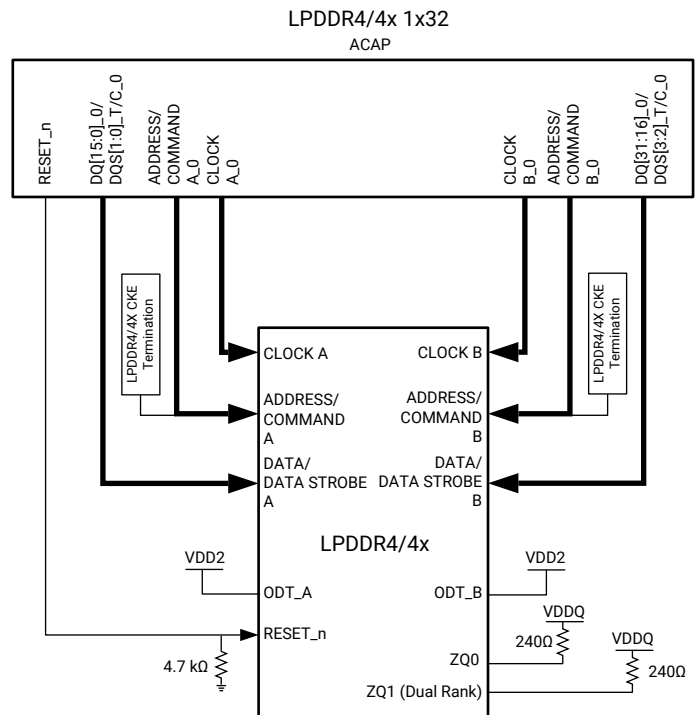


Figure 116: Connections for a 2x16 LPDDR4/4x Interface



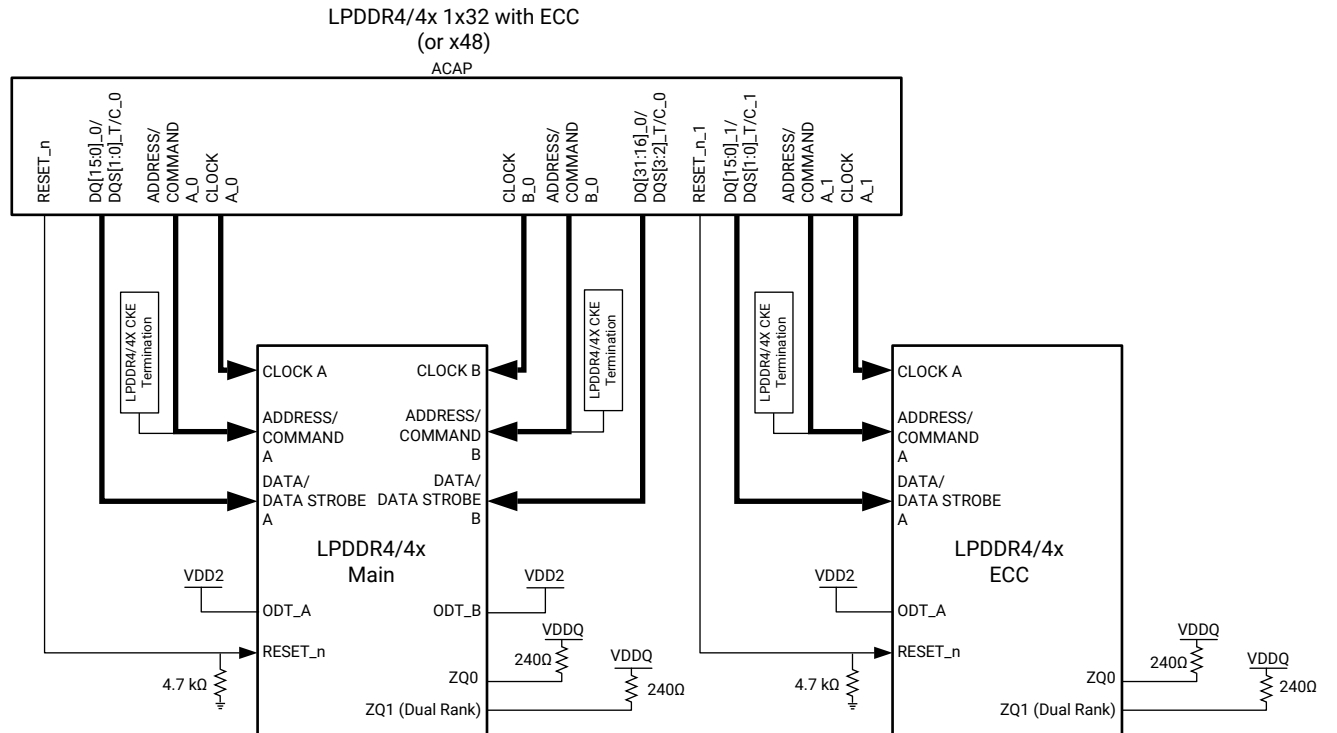
X21605-062120

Figure 117: Connections for a 1x32 LPDDR4/4x Interface



X22658-062120

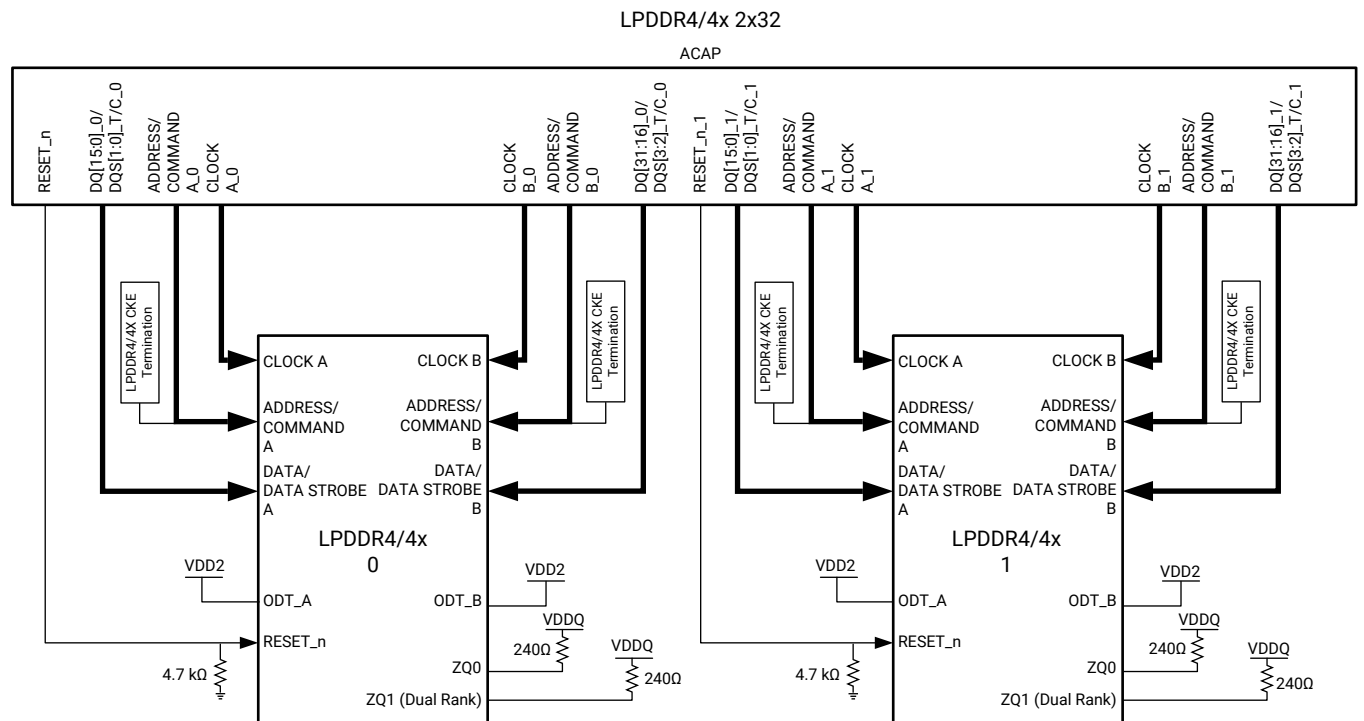
Figure 118: Connections for a 1x32 with ECC (or 1x48) LPDDR4/4x Interface



X21604-062120



Figure 119: Connections for a 2x32 LPDDR4/4x Interface



## System Address Map

### Address Decoding and the System Address Map

The Versal® ACAP programmable NoC system address map defines the default address locations of slaves in the Versal device. The address map is built into the integrated interconnect of the NoC. The NoC provides some capabilities to perform address re-mapping which allow the address map to be customized to the target application.

The following table shows the top level of the system address map from the NoC perspective.

**Table 31: System Address Map**

Name	Start Address	End Address	Description
DDR_LOW0	0x0000_0000_0000	0x0000_7FFF_FFFF	DDR Channel 0 subregion 0
LPD_AFI_FS	0x0000_8000_0000	0x0000_9FFF_FFFF	Access to the PL via the low power domain AFI port
reserved	0x0000_A000_0000	0x0000_A3FF_FFFF	Not decoded
FPD_AFI_0	0x0000_A400_0000	0x0000_AFFF_FFFF	Access to the PL via the full power domain AFI port
FPD_AFI_1	0x0000_B000_0000	0x0000_BFFF_FFFF	Access to the PL via the full power domain AFI port
QSPI	0x0000_C000_0000	0x0000_DFFF_FFFF	Access to the QSPI/OSPI interface
PCIe_0	0x0000_E000_0000	0x0000_EFFF_FFFF	Access to PCIe region 0
PMC	0x0000_F000_0000	0x0000_F7FF_FFFF	Access to the PMC slave devices
STM_CORESIGHT	0x0000_F800_0000	0x0000_F90F_FFFF	Access to the CoreSight STM and the GIC programming interface
reserved	0x0000_F910_0000	0x0000_FBFF_FFFF	Not decoded
CPM	0x0000_FC00_0000	0x0000_FCFE_FFFF	Access to the CPM block
FPD_SLAVES	0x0000_FD00_0000	0x0000_FDFE_FFFF	Access to slave devices in the full power domain
LPD_SLAVES	0x0000_FE00_0000	0x0000_FFEF_FFFF	Access to slave devices in the low power domain
PMC_ALIAS_0	0x0001_0000_0000	0x0001_07FF_FFFF	Access to the PMC on the master die from other die on SSIT devices
PMC_ALIAS_1	0x0001_0800_0000	0x0001_0FFF_FFFF	Access to the PMC on the die 1 from other die on SSIT devices
PMC_ALIAS_2	0x0001_1000_0000	0x0001_17FF_FFFF	Access to the PMC on the die 2 from other die on SSIT devices

Table 31: System Address Map (cont'd)

Name	Start Address	End Address	Description
PMC_ALIAS_3	0x0001_1800_0000	0x0001_1FFF_FFFF	Access to the PMC on the die 3 from other die on SSIT devices
reserved	0x0001_2000_0000	0x0003_FFFF_FFFF	Not decoded
PS_TO_PL_0	0x0004_0000_0000	0x0005_FFFF_FFFF	Access from the PS to the PL via the AFI interface
PCIE_1	0x0006_0000_0000	0x0007_FFFF_FFFF	Access to PCIe region 1
DDR_LOW1	0x0008_0000_0000	0x000F_FFFF_FFFF	DDR Channel 0 subregion 1
reserved	0x0010_0000_0000	0x007F_FFFF_FFFF	Not decoded
PCIE_2	0x0080_0000_0000	0x00BF_FFFF_FFFF	Access to PCIe region 2
DDR_LOW2	0x00C0_0000_0000	0x00FF_FFFF_FFFF	DDR Channel 0 subregion 2
DDR_LOW3	0x0100_0000_0000	0x01B7_7FFF_FFFF	DDR Channel 0 subregion 3
reserved	0x01B7_8000_0000	0x01FF_FFFF_FFFF	Not decoded
AIE	0x0200_0000_0000	0x0200_3FFF_FFFF	Access to the AI Engine array
Reserved	0x0200_8000_0000	0x0200_FFFF_FFFF	Not decoded
PL_LO	0x0201_0000_0000	0x03FF_FFFF_FFFF	Access to slave devices in the PL, low address region
reserved	0x0400_0000_0000	0x04FF_FFFF_FFFF	Not decoded
DDR_CH1	0x0500_0000_0000	0x057F_FFFF_FFFF	Low half of DDR Channel 1
DDR_CH1_1	0x0580_0000_0000	0x05FF_FFFF_FFFF	High half of DDR Channel 1
DDR_CH2	0x0600_0000_0000	0x067F_FFFF_FFFF	Low half of DDR Channel 2
DDR_CH2_1	0x0680_0000_0000	0x06FF_FFFF_FFFF	High half of DDR Channel 2
DDR_CH3	0x0700_0000_0000	0x077F_FFFF_FFFF	Low half of DDR Channel 3
DDR_CH3_1	0x0780_0000_0000	0x07FF_FFFF_FFFF	High half of DDR Channel 3
PL_HI	0x0800_0000_0000	0x0FFF_FFFF_FFFF	Slave devices in the PL, high address region
reserved	0x1000_0000_0000	0xFFFF_FFFF_FFFF	Not decoded

## Address Regions

- **DDR\_CH0:** Address region DDR\_CH0 comprises multiple sub-regions: DDR\_LOW0, DDR\_LOW1, DDR\_LOW2, and DDR\_LOW3, totaling 1 TB of space. The low 2 GB of that space, DDR\_LOW0, is addressable by 32-bit address masters. Address regions DDR\_CH1, DDR\_CH2, and DDR\_CH3 each may be divided into two sub-regions that can map to independent memory controllers.
- **CIPS:** The various CIPS slave regions (LPD\_AFI\_FS, FPD\_AFI\_0, FPD\_AFI\_1, QSPI, PCIe\_0, PMC, STM\_CORESIGHT, CPM, FPD\_SLAVES, and LPD\_SLAVES) have fixed address regions within the low 4 GB of the address space to allow for access by 32-bit masters.
- **PMC\_ALIAS:** The PMC\_ALIAS regions provide access to the address space of the PMC block in one SLR from masters in other SLRs. For example, access to the PMC in SLR1 from SLR0 would be through the PMC\_ALIAS\_1 address region.
- **PS\_to\_PL:** The PS\_to\_PL address region provides direct access from masters in the PS to slaves in the PL through the AFI interface. This region is not accessible from the NoC.

- **AIE, PL\_LO and PL\_HI:** The AIE, PL\_LO, and PL\_HI regions do not have dedicated address decoders. Transactions to these regions must use the fixed destination ID, a master defined destination ID, or an address remap register.

If a transaction is being routed using the AXI address and the incoming address does not hit in any address matcher or fall in the AIE, PL\_LO or PL\_HI spaces, an AXI DECERR is generated and an interrupt status bit is set to indicate an address map error. Transactions that fall in the AIE, PL\_LO, or PL\_HI spaces but do not hit in a remap register and hence do not receive a valid destination ID may still be injected into the NoC routing fabric. In this case an NSU may be configured as an error slave to receive such invalid transactions. The error slave is configured to return a DECERR instead of a SLVERR.

# Memory Interface Debug

Hardware issues can range from an interface failing to calibrate at power-on to data errors that occur hours into endurance testing. This section describes methods and tools to use for isolating and resolving these issues.

---

## General Memory Debug Checklist

1. Verify that the system clock frequency on hardware matches the IP setting (Input System Clock Period).
2. Verify all guidelines in the Memory Interface chapter of the *Versal ACAP PCB Design User Guide* ([UG863](#)) have been followed.
3. Check the rules on the pin and bank options in DDR Memory Controller (see Pinout Rules).



**RECOMMENDED:** You are encouraged to try the *Obtaining and Verifying Versal ACAP Memory Pinouts* [tutorial](#) available on GitHub. This is a fast and effective way to quickly generate pinouts for Versal DDRMCs. All pins swaps must be captured in the design's XDC and validated before generating hardware. PCB level pin swaps not captured in the tools may lead to hardware failures if pin rules are not followed.

4. Measure voltages on the board during idle and non-idle times to ensure the voltages are set appropriately and the noise is within specifications.
  - Ensure the termination voltage regulator ( $V_{tt}$ ) is powered to  $V_{cco}/2$ .



**IMPORTANT!** LPDDR4/4X protocol uses feedback on the DQ bits during CA Training and Write Leveling calibration stages so any pin swapping needs to be done and validated in the tools. Additionally the DQ mapping from the ACAP to the LPDDR4/4X component channels needs to maintain an exact 1:1 mapping. For example, LPDDR4\_DQ\_A[0] must be connected to DQ0 of Channel A of the LPDDR4 component. It follows LPDDR4\_DQ\_A[1] is connected to DQ1 of Channel A, through LPDDR4\_DQ\_B[15] is connected to DQ15 of Channel B of the LPDDR4 component.

5. Scope the clock input to verify frequency and signal quality.
6. Check the termination registers for the proper values. These are detailed in the Memory Interface chapter of the *Versal ACAP PCB Design User Guide* ([UG863](#)).

7. Perform general signal integrity analysis:
  - a. Observe DQ and DQS signals using a scope at the memory. View the alignment of the signals and the Vil/Vih levels during both reads and writes and the overall signal integrity.
  - b. Observe the Address and Command signals on a scope at the memory. View the alignment of the signals and the Vil/Vih levels and the overall signal integrity.
8. Verify the memory parts on the board match the settings set in the Memory IP. The timing parameters must match between the IP and the physical part.
9. Measure CK/CK\_N and DQS/DQS\_N and the system clock for duty cycle distortion and general signal integrity.
10. Verify timing constraint rules (trace matching) are being met as documented in the Memory Interface chapter of the *Versal ACAP PCB Design User Guide* ([UG863](#)).

## Related Information

[Pinout Rules](#)

---

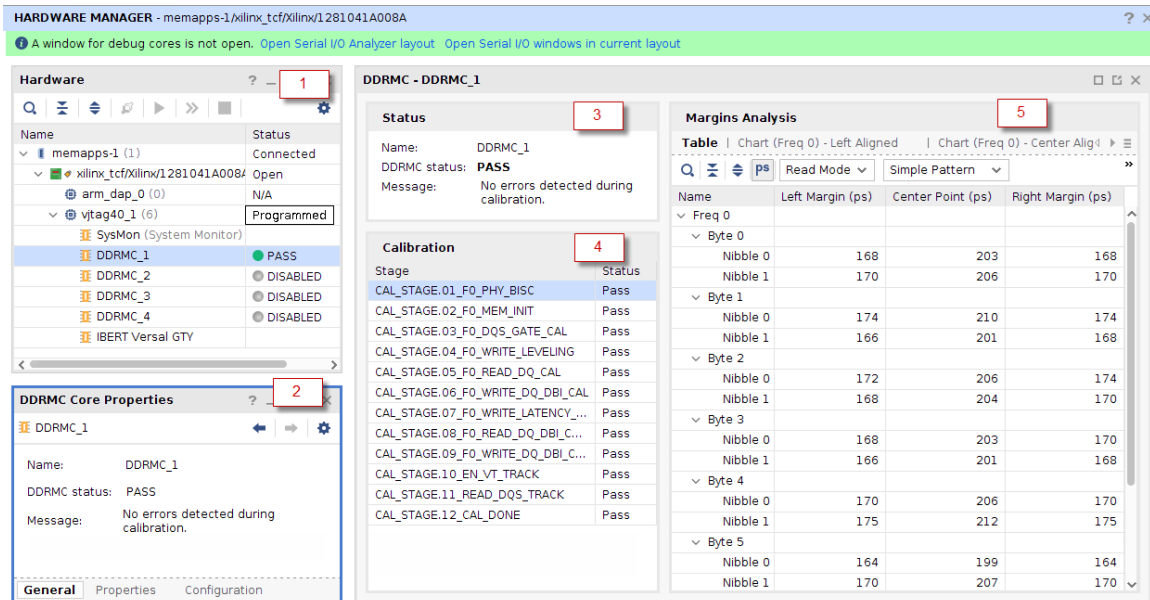
# Vivado Hardware Manager – Memory Debug

After programming, the Vivado® Hardware Manager displays all debug IP which includes integrated memory interfaces. This provides information such as memory IP configuration, status of calibration, calibration errors, and read/write window margin data. This information can be viewed through the GUI or extracted to text files via Tcl.

**Note:** These properties are read back only once, either upon completion of programming, or initial connection to an already programmed device. They are not regularly updated, so if any actions have been done to change the configuration after initial programming, you should refresh the device to re-download the latest properties. To do this, right-click on the device and select **Refresh Device**.

# Memory Debug: GUI Usage

Figure 120: Debug GUI



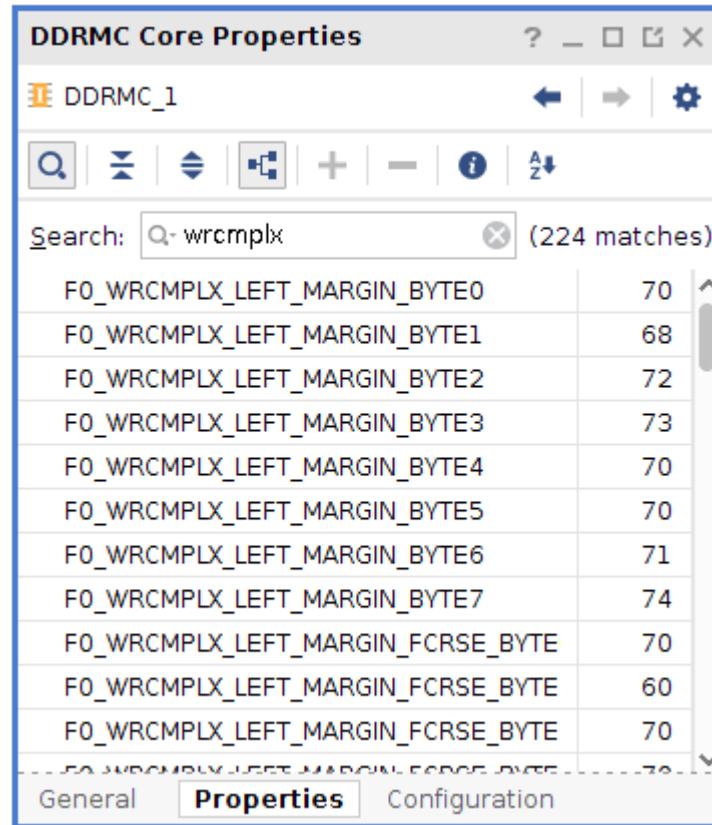
- 1. **Debug Tree:** This is the list of all of the debug elements in the design. For the integrated memory interfaces, all four are displayed with status listed to the right. A label of 'DISABLED' means that particular memory controller is not used in the current configuration. Selecting a DDRMC will cause all windows to update with the appropriate data.

The DDRMC\_x is a label that notes the physical location of the memory controller on the device. DDRMC\_1 is the memory controller on the far left, underneath the Processor. DDRMC\_2 is the next on the right, and so on.

- 2. **Core Properties:** This window contains three tabs: General, Properties, and Configuration. The General tab displays basic status of the memory controller.

The Properties tab, shown in the following figure, lists out all of the internal registers for the memory controller.

Figure 121: **Memory Controller Core Properties: Properties Tab**

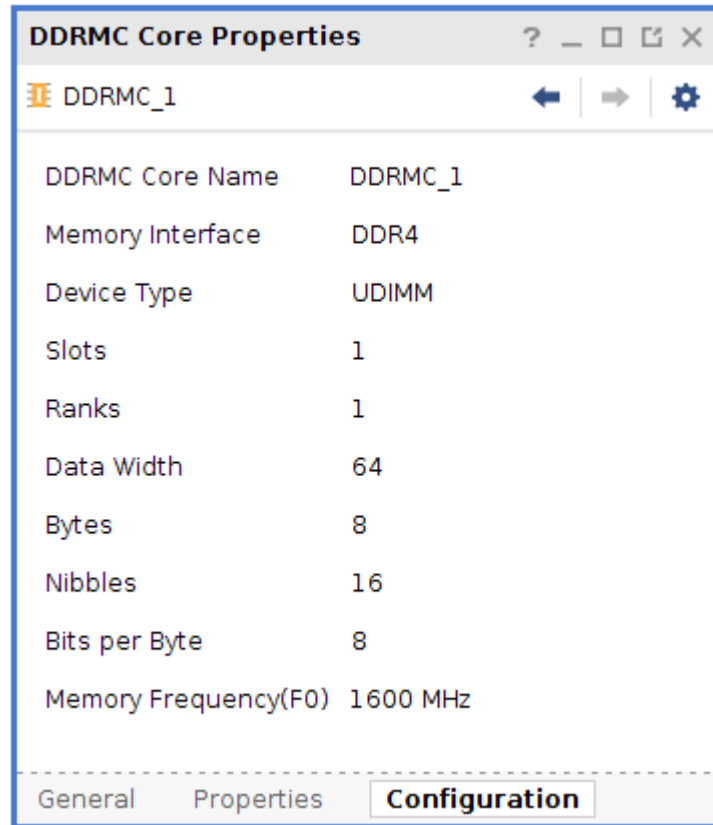


A search is available by selecting the magnifying glass icon in the upper left. This is helpful to minimize the hundreds of properties when searching for specific information.

The Configuration tab, shown in the following figure, provides information on the configuration programmed into the memory controller such as data width, ranks, bytes, and memory frequency.



Figure 122: Memory Controller Core Properties: Configuration Tab



- **3. Memory Controller Status:** This window provides the overall status of the memory controller. If there is an error in calibration, detailed information describing the issue will be displayed here, as well as the location of the failure such as which byte lanes or nibbles (if appropriate).
- **4. Calibration Stages:** This window lists all of the calibration states that are needed to bring up the memory interface for a given configuration. Depending on the memory type (DDR4 vs. LPDDR4) or the operating frequency of the memory, the stages will vary.

In the event of a calibration error, the failing stage will be shown.

- **5. Margin Data:** This window, shown in the following figure, displays the margin data obtained from calibration. There is margin data for both read and write modes. For each mode there are also simple and complex patterns. This represents the aggressiveness of the data pattern used. It is expected that complex data patterns will result in a smaller window than the simple pattern due to more bits transitioning and therefore generating more noise. There is also a selection for the rising and falling edges of the DQS.

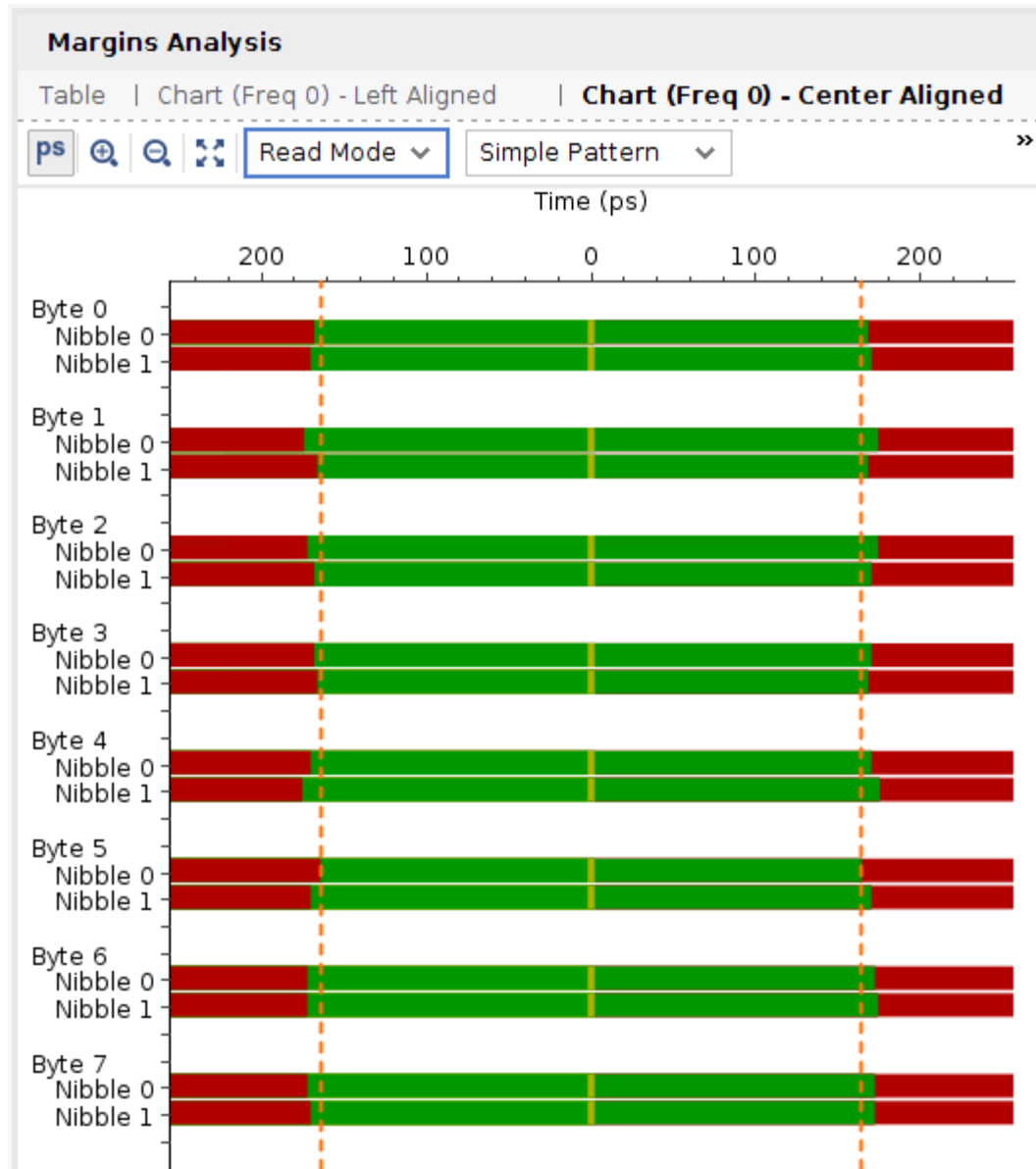
Figure 123: Memory Controller Margin Window: Table View

Margins Analysis			
<div> <div>Table</div> <div>Chart (Freq 0) - Left Aligned</div> <div>Chart (Freq 0) - Center Aligned</div> </div>			
<div> <div>Q</div> <div>≡</div> <div>≡</div> <div>ps</div> <div>Read Mode</div> <div>Simple Pattern</div> <div>Rising Clock Edge</div> </div>			
Name	Left Margin (taps)	Center Point (taps)	Right Margin (taps)
▼ Freq 0			
▼ Byte 0			
Nibble 0	86	104	86
Nibble 1	86	104	86
▼ Byte 1			
Nibble 0	88	106	88
Nibble 1	86	104	87
▼ Byte 2			
Nibble 0	89	107	90
Nibble 1	84	102	85
▼ Byte 3			
Nibble 0	86	104	87
Nibble 1	85	103	86
▼ Byte 4			
Nibble 0	86	104	86
Nibble 1	85	103	85
▼ Byte 5			
Nibble 0	85	103	85
Nibble 1	84	102	84
▼ Byte 6			
Nibble 0	86	104	86
Nibble 1	86	104	87
▼ Byte 7			
Nibble 0	86	104	86
Nibble 1	85	103	86

Margin data can also be viewed in a graph format (as shown in the following figure). It is easier to compare window margins for the various byte lanes this way. The dotted vertical line shows the size of the smallest window for comparison purposes. It does not signify a minimum margin for proper operation.

The graph view is helpful to identify any issues that may result from poor layout. If one nibble or byte lane is much smaller than the others, it is worthwhile to examine the layout of that lane for any routing rule violations.

Figure 124: Memory Controller Margin Window: Chart View



## Memory Debug: Tcl Usage

The following Tcl commands are available from the Vivado Tcl Console when connected to the hardware in Hardware Manager.

- **get\_hw\_ddrmcs:** This command returns a list of all the integrated memory controllers. Use this command with 'lindex' to select an item in the list.
- **refresh\_hw\_device:** This command refreshes the whole device and all debug cores if no target is provided.
- **report\_property:** This command displays all the properties for a given target.

For example, to report all of the properties for the second enabled DDRMC:

```
report_property [lindex [ get_hw_ddrmcs] 1 ]
```

**Note:** The memory controller numbering in Hardware Manager starts from 1, but the Tcl lindex command starts from 0. If there are two enabled DDRMCs, their index numbers will be 0 and 1, regardless of their physical location on the ACAP device.

- **report\_hw\_ddrmc:** This command provides a formatted text output summarizing data that is provided in the GUI.

Following is an example of the output.

Figure 125: report\_hw\_ddrmc

```

-----
Memory Configuration
-----
DDRM Core Name:  DDRMC_1
Memory Interface: DDR4
Device Type:  UDIMM
Slots:  1
Ranks:  1
Data Width:  64
Bytes:  8
Nibbles:  16
Bits per Byte:  8
Memory Frequency(F0):  1600 MHz

-----
Calibration Stages Information
-----
CAL_STAGE.01_F0_PHY_BISC:  Pass
CAL_STAGE.02_F0_MEM_INIT:  Pass
CAL_STAGE.03_F0_DQS_GATE_CAL:  Pass
CAL_STAGE.04_F0_WRITE_LEVELING:  Pass
CAL_STAGE.05_F0_READ_DQ_CAL:  Pass
CAL_STAGE.06_F0_WRITE_DQ_DBI_CAL:  Pass
CAL_STAGE.07_F0_WRITE_LATENCY_CAL:  Pass
CAL_STAGE.08_F0_READ_DQ_DBI_CAL_COMPLEX:  Pass
CAL_STAGE.09_F0_WRITE_DQ_DBI_CAL_COMPLEX:  Pass
CAL_STAGE.10_EN_VT_TRACK:  Pass
CAL_STAGE.11_READ_DQS_TRACK:  Pass
CAL_STAGE.12_CAL_DONE:  Pass

-----
Calibration Window Margin Analysis
-----

Frequency 0 - Read Margin - Simple Pattern - Rising Edge Clock in pS and (delay taps):
Byte 0 Nibble 0 - Left Margin: 168 (86) Center Point: 203 (104) Right Margin: 168 (86)
Byte 0 Nibble 1 - Left Margin: 170 (86) Center Point: 206 (104) Right Margin: 170 (86)
Byte 1 Nibble 0 - Left Margin: 174 (88) Center Point: 210 (106) Right Margin: 174 (88)
Byte 1 Nibble 1 - Left Margin: 166 (86) Center Point: 201 (104) Right Margin: 168 (87)
Byte 2 Nibble 0 - Left Margin: 172 (89) Center Point: 206 (107) Right Margin: 174 (90)
Byte 2 Nibble 1 - Left Margin: 168 (84) Center Point: 204 (102) Right Margin: 170 (85)
Byte 3 Nibble 0 - Left Margin: 168 (86) Center Point: 203 (104) Right Margin: 170 (87)
Byte 3 Nibble 1 - Left Margin: 166 (85) Center Point: 201 (103) Right Margin: 168 (86)
Byte 4 Nibble 0 - Left Margin: 170 (86) Center Point: 206 (104) Right Margin: 170 (86)
Byte 4 Nibble 1 - Left Margin: 175 (85) Center Point: 212 (103) Right Margin: 175 (85)
Byte 5 Nibble 0 - Left Margin: 164 (85) Center Point: 199 (103) Right Margin: 164 (85)
Byte 5 Nibble 1 - Left Margin: 170 (84) Center Point: 207 (102) Right Margin: 170 (84)
Byte 6 Nibble 0 - Left Margin: 172 (86) Center Point: 208 (104) Right Margin: 172 (86)
Byte 6 Nibble 1 - Left Margin: 172 (86) Center Point: 208 (104) Right Margin: 174 (87)
Byte 7 Nibble 0 - Left Margin: 172 (86) Center Point: 208 (104) Right Margin: 172 (86)
Byte 7 Nibble 1 - Left Margin: 170 (85) Center Point: 206 (103) Right Margin: 172 (86)

```

## UART Debug

In the event of calibration failure, the recommended debug method is to use the Hardware Manager to view the calibration status. When this is not possible, there is information provided in the UART0 output (enabled via CIPS). The output looks similar to the following:

```
[timestamp] ====DDRMC Register Dump Start=====
[timestamp] DDRMC_0 (UB 0xF6110000)
[timestamp] PCSR Control: 0x3
[timestamp] PCSR Status: 0x27
[timestamp] Calibration Error: 0xE
[timestamp] Nibble Location 1: 0x0
[timestamp] Nibble Location 2: 0xA
[timestamp] Nibble Location 3: 0x0
[timestamp] Calibration Stage: 0x1CB
```

Depending on the Versal device, the number of DDRMC\_x reported will vary. There will be one set of data per DDRMC instance.

- **DDRMC\_x:** This defines the DDRMC instance and the base register information for the DDRMC\_UB\_x block.
- **PCSR Control:** The only relevant bit is bit 0. A value of b1 indicates that the Memory Controller is enabled in the design.
- **PCSR Status:**

Table 32: Bit Field Descriptions

Bit Field	Description
5	CALERROR : 1 if there is an error during calibration
4	CALDONE : 1 if calibration has completed
3	INCAL : 1 if calibration is still in progress

### Calibration Error

Convert the hex code to decimal, and refer to the following table.

Table 33: Calibration Error Codes

CODE	ERROR
0	No error code generated
1	XPLL Timeout waiting for lock
2	XPLL found calibration error
3	XPHY BISC timeout waiting for BISC fixdly rdy
4	XPHY BISC found r1 dly qtr value to be 0

Table 33: Calibration Error Codes (cont'd)

CODE	ERROR
5	MEM INIT timeout waiting for memory initialization to complete
6	DQS gating timeout waiting for XPHY gate training to complete
7	DQS gating reached maximum read latency limit
8	DQS gating found rank to rank skew greater than expected
9	Write leveling failed to find rising edge using coarse and fine offsets
10	Write leveling failed in stable 0 confirmation stage
11	Write leveling reached maximum taps to find noise width by incrementing DQS odelay
12	Read DQ failed to find rising edge valid window by incrementing PQTR
13	Read DQ failed to find rising edge valid window by incrementing DQ idelay
14	Read DQ failed to find rising edge noise region by incrementing DQ idelay by higher taps
15	Read DQ failed to find rising edge noise region by incrementing DQ idelay
16	Read DQ failed to find noise region by incrementing DQ idelay
17	Read DQ failed to find left edge by incrementing PQTR and NQTR
18	Read DQ failed to find right edge by decrementing PQTR and NQTR
19	Read DQ failed in negative test post sanity check
20	Read DQ failed in positive test post sanity check
21	Write DQ-DBI failed to find valid window by moving DQS odelay first and then DQ odelay
22	Write DQ-DBI failed to find the noise region for per-bit deskew by incrementing DQ odelay by higher taps
23	Write DQ-DBI failed to find the noise region for per-bit deskew by incrementing DQ odelay
24	Write DQ-DBI failed to find valid window by moving DQS odelay first and then DBI odelay
25	Write DQ-DBI failed to find the noise region for per-bit deskew by incrementing DBI odelay by higher taps
26	Write DQ-DBI failed to find the noise region for per-bit deskew by incrementing DBI odelay
27	Write DQ-DBI failed to find the right edge by incrementing DQS odelay by higher taps
28	Write DQ-DBI failed to find the right edge by incrementing DQS odelay
29	Write DQ-DBI failed in post sanity check
30	Write DQ-DBI failed to find the valid window by incrementing DQ odelay
31	Write DQ-DBI doesn't have enough taps to decrement DQ odelay
32	Write DQ-DBI failed to find the valid window by incrementing DBI odelay
33	Write DQ-DBI failed to find the left edge by incrementing DQ odelay by higher taps
34	Write DQ-DBI failed to find the left edge by incrementing DQ odelay
35	Write DQ-DBI read DQS oscillator value to be 0
36	Write DQ-DBI computed tDQS2DQ value to be out of range
37	Write latency failed to find expected data pattern
38	Write latency found larger rank to rank skews than is supported
39	Write latency failed in post sanity check
40	Write latency failed in sanity check 1
41	Write latency failed in sanity check 2
42	Write latency failed in read complex pattern sanity check
43	Write latency failed in write complex pattern sanity check

Table 33: Calibration Error Codes (cont'd)

CODE	ERROR
44	Write latency failed in PRBS read pattern sanity check
45	Write latency failed in PRBS write pattern sanity check
46	Write latency read DQS oscillator value unexpectedly found to be 0
47	Write latency computed tDQS2DQ value to be out of range
48	Read DBI failed to find valid window by incrementing DBI idelay
49	Read complex failed to find noise region by incrementing DQ idelay
50	Read complex failed to find left edge by incrementing PQTR and NQTR using short complex burst
51	Read complex failed to find left edge by incrementing PQTR and NQTR
52	Read complex failed to find right edge by decrementing PQTR and NQTR using short complex burst
53	Read complex failed to find right edge by decrementing PQTR and NQTR
54	Read complex failed in positive test post sanity check
55	Write complex failed to find the left edge by incrementing DQ odelay by higher taps
56	Write complex failed to find the left edge by incrementing DQ odelay
57	Write complex failed to find the right edge by incrementing DQS odelay by higher taps
58	Write complex failed to find the right edge by incrementing DQS odelay
59	Write complex failed in post sanity check
60	Write complex failed to find the right edge by decrementing DQ odelay by higher taps
61	Write complex failed to find the right edge by decrementing DQ odelay
62	Write complex failed in post sanity check 1
63	Write complex read DQS oscillator value unexpectedly found to be 0
64	Write complex computed tDQS2DQ value to be out of range
65	Read PRBS failed in pre sanity check
66	Read PRBS failed to find left edge by incrementing PQTR and NQTR
67	Read PRBS failed to find right edge by decrementing PQTR and NQTR
68	Read PRBS failed in positive test post sanity check
69	Write PRBS failed to find left margin by incrementing DQ odelays by higher taps
70	Write PRBS failed to find left margin by incrementing DQ odelays
71	Write PRBS failed to find right margin by incrementing DQS odelays by higher taps
72	Write PRBS failed to find right margin by incrementing DQS odelays
73	Write PRBS failed to find right margin by decrementing DQ odelays by higher taps
74	Write PRBS failed to find right margin by decrementing DQ odelays
75	Write PRBS failed in post sanity check 1
76	Calibration enable VTC failed in post sanity check 1
77	Calibration read DQS tracking failed in post sanity check 1
78	Calibration read DQS tracking found RLDLYRNK coarse underflow
79	Calibration read DQS tracking found RLDLYRNK coarse overflow
80	Calibration LP4 oscillator tracking failed in pre sanity check 1
81	Calibration LP4 oscillator tracking failed in sanity check 1
82	Calibration LP4 oscillator tracking failed in post sanity check 1



Table 33: Calibration Error Codes (cont'd)

CODE	ERROR
83	Calibration LP4 oscillator tracking read DQS oscillator value to be 0
84	Calibration LP4 oscillator tracking computed tDQS2DQ value to be out of range
85	Read DQS gate tracking found RLDLYRNK coarse underflow
86	Read DQS gate tracking found RLDLYRNK coarse overflow
87	LP4 oscillator tracking read DQS oscillator value to be 0
88	LP4 oscillator tracking computed tDQS2DQ value to be out of range
89	LP4 oscillator tracking rank switching did not take effect
90	Self refresh exit DQS gating timeout waiting for XPHY gate training done signal
91	Self refresh exit DQS gating reached maximum read latency limit
92	Self refresh exit DQS gating found rank to rank skew greater than expected
93	Self refresh exit read DQS tracking failed in sanity check ddr
94	Self refresh exit read DQS tracking found RLDLYRNK coarse underflow
95	Self refresh exit read DQS tracking found RLDLYRNK coarse overflow
96	Frequency switching waiting for BISC pause ready signal
97	Frequency switching timeout waiting for XPLL lock signal
98	Frequency switching XPLL found calibration error
99	Frequency switching timeout waiting for BISC fixdly rdy signal
100	Frequency switching found zero rl dly qtr value after XPHY BISC
101	Frequency switching failed in enable VTC post sanity check 1
102	Parity error occurred during transaction re-transmission
103	Watchdog timeout while polling scrub busy after scrub being stopped
104	Watchdog timeout while polling scrub busy after scrub being enabled
105	Watchdog timeout while polling self refresh entry request from DC FSM
106	Watchdog timeout while polling self refresh exit request from DC FSM
107	Watchdog timeout while polling self refresh exit done from DC FSM
108	Calibration enable VTC timeout waiting for phy_rdy
109	Frequency switching enable VTC timeout waiting for phy_rdy
110	Restore calibration timeout waiting for XPLL lock signal
111	Restore calibration XPLL found calibration error
112	Restore calibration timeout waiting for BISC fixdly rdy signal
113	Restore calibration found zero rl dly qtr value after XPHY BISC
114	Restore calibration failed in enable VTC post sanity check 1
115	Restore calibration enable VTC timeout waiting for phy_rdy
116	CA calibration failed to find noise right edge while incrementing CA odelay by higher taps
117	CA calibration failed to find noise right edge while incrementing CA odelay
118	CA calibration failed in CA odelay centering stage
119	CA calibration failed to find noise right edge while incrementing CS odelay by higher taps
120	CA calibration failed to find noise right edge while incrementing CS odelay
121	CA calibration failed in CS odelay centering stage

Table 33: Calibration Error Codes (cont'd)

CODE	ERROR
122	Write DQ-DBI calibration failed to find right edge deep noise by incrementing DQ odelay
123	Write DQ-DBI calibration failed to find right edge deep noise by incrementing DBI odelay
124	Write DQ-DBI calibration failed to find right edge valid window by incrementing DBI odelay
125	Write DQ-DBI calibration failed to find right edge low noise by decrementing DBI odelay
126	Write DQ-DBI calibration failed to find right edge valid window by incrementing DQ odelay
127	Write DQ-DBI calibration failed to find right edge low noise by decrementing DQ odelay

### Nibble location 1,2,3:

These three fields each contain nine bits of data. One bit for each nibble in an IO bank in the given triplet. Nibble location 3 [8:0] for the third IO bank. Nibble location 2 [8:0] for the second IO bank, and Nibble location 1 [8:0] for the first IO bank. A '1' in any of these bit locations represents the nibble where a failure was detected associated with the error code defined earlier. Some calibration stages are done in parallel, so it is possible for multiple nibbles to be flagged.

### Calibration Status/Stage:

This hex code represents nine bits of data.

See [Table 35: CAL\\_SEQ\\_STATUS and CAL\\_POINTER Encoding](#) for information on the contents.

## DDRMC Calibration Debug

All references to register names in the following sections can be obtained from the Vivado Hardware Manager Properties window and Properties tab, or via the Tcl command described in the previous section.

The register names provided in this section can be matched to register names in the [NoC and Integrated Memory Controller NPI Register Reference](#) (AM019). This document provides the NPI address values for each register, enabling you to implement your own method to check the calibration status. In this document, there are two sections, DDRMC\_DDR4\_XRAM Module and DDRMC\_LPDDR4\_XRAM Module. Select the appropriate module based on the target memory interface type, as the register contents and locations vary.

### General Configuration

Confirm that the IP settings are as expected. Access the settings in the **Hardware Manager GUI** → **Core Properties** → **Configuration** tab.

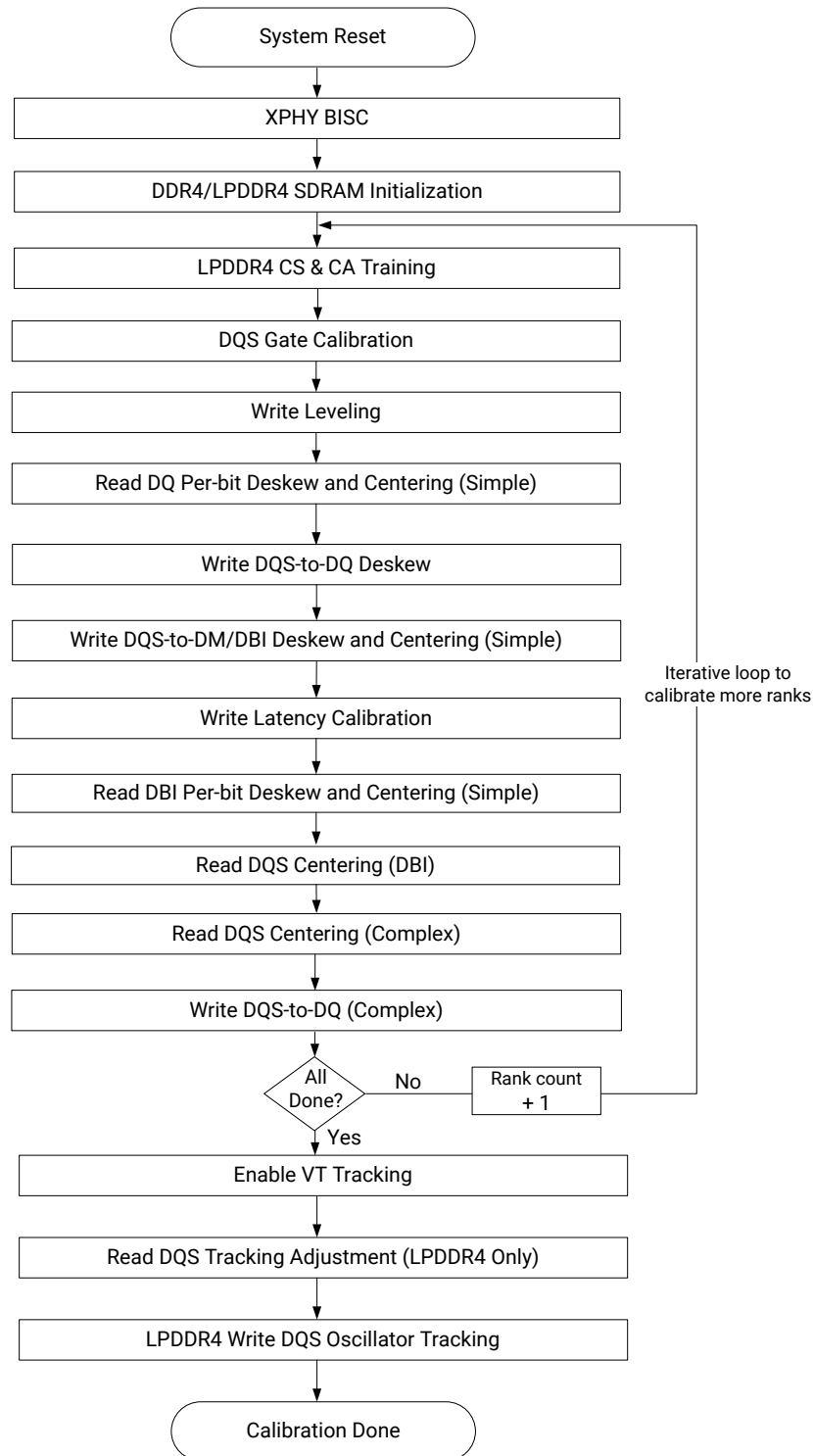
Table 34: Debug Registers

Register Name	Description (in decimal)
MEM_TYPE	1: Integrated Memory Controller DDR4 2: Integrated Memory Controller LPDDR4 3: Soft IP DDR4
PHY_RANKS	Number of XPHY Ranks
MEM_RANKS	Number of Memory Ranks
BYTES	Number of Bytes in the Memory Interface
NIBBLES	Number of Nibbles in the Memory Interface
BITS_PER_BYTE	Bits per byte (4 or 8)
DBI_PINS	Number of DBI pins
SLOTS	Number of Slots in the Memory Interface
DIMM_TYPE	0: Component 1: UDIMM 2: RDIMM 3: LRDIMM
LRANKS	Number of Logical Ranks
NUM_CHANNELS	Number of Memory Channels
DRAM_SIZE	Size of the DRAM components
SYS_CLK_8_0	Concatenate the two nine bit registers , SYS_CLK_17_9 and SYS_CLK_8_0 to represent the system clock period in pS
SYS_CLK_17_9	Concatenate the two nine bit registers , SYS_CLK_17_9 and SYS_CLK_8_0 to represent the system clock period in pS
ECC_EN	1 if ECC is enabled
REF_EN	1 if User Controlled Refresh is enabled
PER_RD_EN	1 if Periodic Reads are enabled. 0 if ECC Scrubbing is enabled.
SCRUB_EN	1 if ECC Scrubbing is enabled

## Calibration Stages

The following figure shows the calibration steps for DDR4 designs.

Figure 126: DDR4 Designs Calibration Steps



X21639-111720

Refer to the Calibration Stage window in the Vivado Hardware Manager to identify if there is a stage of calibration that has failed. The software utilizes the CAL\_SEQ\_STATUS and CAL\_POINTER registers to populate the table.

There are 64 CAL\_SEQ\_STATUS registers which contain the calibration stage codes as well as the status of each calibration stage. Calibration starts with CAL\_SEQ\_STATUS\_0. There are 64 registers allowing for a maximum of 64 stages, but in practice only a subset is utilized.

The CAL\_POINTER register indicates the current status and stage of calibration. By the time the Hardware Manager refreshes these registers it is likely that calibration has already completed, so you should not expect to see it change values. In the event of a calibration error, it will point at the stage where the error occurred. In the event of an error during calibration, the Vivado Hardware Manager will display additional information regarding the reason for failure.

The encoding of the CAL\_SEQ\_STATUS and CAL\_POINTER register is shown in the following table.

**Table 35: CAL\_SEQ\_STATUS and CAL\_POINTER Encoding**

Register Bits	Description
CAL_SEQ_STATUS [8..6]	Stage Status 0: Stage enabled, but not yet started 1: Stage not yet started, but will be skipped (invalid or unnecessary due to lower frequency operation) 2: Reserved 3: Calibration in progress 4: Calibration stage has been skipped 5: Reserved 6: Calibration stage completed successfully 7: Calibration stage failed

Table 35: CAL\_SEQ\_STATUS and CAL\_POINTER Encoding (cont'd)

Register Bits	Description
CAL_SEQ_STATUS [5..0]	<p>Stage encoding. The following list represents the encoding of the stages, but does not imply the same sequence is used for calibration.</p> <p>0x00: F0_PHY_BISC</p> <p>0x01: F0_MEM_INIT</p> <p>0x02: F0_LPDDR4_CS_CA_TRAIN</p> <p>0x03: F0_LPDDR4_CA_VREF_TRAIN</p> <p>0x04: F0_LRDIMM_DB_MREP</p> <p>0x05: F0_LRDIMM_DB_MRD_CYCLE</p> <p>0x06: F0_LRDIMM_DB_MRD_CENTER</p> <p>0x07: F0_LRDIMM_DB_DWL</p> <p>0x08: F0_LRDIMM_DB_MWD_CYCLE</p> <p>0x09: F0_LRDIMM_DB_MWD_CENTER</p> <p>0x0a: F0_DQS_GATE_CAL</p> <p>0x0b: F0_READ_DQ_CAL</p> <p>0x0c: F0_WRITE_LEVELING</p> <p>0x0d: F0_WRITE_DQ_DBI_CAL</p> <p>0x0e: F0_WRITE_LATENCY_CAL</p> <p>0x0f: F0_READ_DBI_CAL</p> <p>0x10: F0_READ_DQ_VREF_CAL</p> <p>0x11: F0_READ_DQ_DBI_CAL_COMPLEX</p> <p>0x12: F0_WRITE_DQ_VREF_CAL</p> <p>0x13: F0_WRITE_DQ_DBI_CAL_COMPLEX</p> <p>0x1a: F0_PRBS_READ</p> <p>0x1b: F0_PRBS_WRITE</p> <p>0x1f: FREQ_SWITCH</p> <p>0x20: F1_PHY_BISC</p> <p>0x21: F1_MEM_INIT</p> <p>0x22: F1_LPDDR4_CS_CA_TRAIN</p> <p>0x23: F1_LPDDR4_CA_VREF_TRAIN</p> <p>0x24: F1_LRDIMM_DB_MREP</p> <p>0x25: F1_LRDIMM_DB_MRD_CYCLE</p> <p>0x26: F1_LRDIMM_DB_MRD_CENTER</p> <p>0x27: F1_LRDIMM_DB_DWL</p> <p>0x28: F1_LRDIMM_DB_MWD_CYCLE</p> <p>0x29: F1_LRDIMM_DB_MWD_CENTER</p> <p>0x2a: F1_DQS_GATE_CAL</p> <p>0x2b: F1_READ_DQ_CAL</p> <p>0x2c: F1_WRITE_LEVELING</p> <p>0x2d: F1_WRITE_DQ_DBI_CAL</p> <p>0x2e: F1_WRITE_LATENCY_CAL</p> <p>0x2f: F1_READ_DBI_CAL</p> <p>0x30: F1_READ_DQ_VREF_CAL</p> <p>0x31: F1_READ_DQ_DBI_CAL_COMPLEX</p> <p>0x32: F1_WRITE_DQ_VREF_CAL</p> <p>0x33: F1_WRITE_DQ_DBI_CAL_COMPLEX</p> <p>0x3a: F1_PRBS_READ</p> <p>0x3b: F1_PRBS_WRITE</p> <p>0x3c: EN_VT_TRACK</p> <p>0x3d: READ_DQS_TRACK</p> <p>0x3e: LPDDR4_WRITE_DQS_OSC_TRACK</p> <p>0x3f: CAL_DONE</p>

If calibration has completed successfully, CAL\_POINTER Status will be b'110 (Calibration stage completed successfully) and the Stage will be x3f (CAL\_DONE). The concatenated value will be x1bf. If calibration encounters an error, the Status will be b'111 (Calibration stage failed) and the Stage will indicate the failing stage.

If a calibration error occurs, Hardware Manager extracts the error code from the register CAL\_ERROR. The error codes can be decoded from the [Table 33: Calibration Error Codes](#).

## PHY BISC

In this calibration stage, internal delays are measured to determine the number of fine taps required to equal a quarter clock memory period. This is referred to as a coarse tap.

**Table 36: PHY BISC Calibration Register**

Register Name	Quantity	Description
Fx_CALBISC_RL_DLY_QTR	1	Number of fine taps to equal ¼ memory clock period.

## Memory Initialization

The PHY issues commands to load the DDR4 Mode registers with values defined in the user IP settings. There are no debug registers associated with this calibration stage.

## LPDDR4 CS and CA Training

LPDDR4/4X SDRAM provides a mechanism for training the command bus prior to enabling termination for high-frequency operation. For operation above 1866 Mb/s, this stage of training uses the Command Bus Training mode to align the CS and CA signals at the DRAM relative to CK. In this mode, the DRAM uses CS and CK to capture the values on the CA pins, and it feeds the result back to the controller on the DQ[13:8] pins. The training is accomplished in two phases: CS Training and CA Training.

CS Training is completed first to align CS transitions with the falling edge of CK. A static pattern is driven on the CA pins, while the CS to CK timing is varied. By comparing the pattern received on the DQ pins relative to the pattern sent on the CA pins, the controller can identify the noise region. When this has been completed, the CS delay is adjusted to center the rising CK edge ½ clock cycle away from the center of the noise region. After aligning CS to CK, CA Training uses a toggling pattern on the CA lines to deskew and center the CA signals relative to CK.

There are no debug registers for this calibration stage.

## LRDIMM MREP Training

MREP training aligns the Read MDQS phase with the data buffer clock. In this training mode the memory controller sends a sequence of read commands, the DRAM sends out the MDQS, the data buffer samples the strobe with the clock and feeds the result back on DQ. Calibration continues to perform this training to find the 0 to 1 transition on the Read MDQS sampled with the data buffer clock.

Table 37: MREP Training Register

Register Name	Quantity	Description
Fx_DB_MREP_MRD_LAT	Rank	MRD latency [8:6] and MREP phase [5:0]

## LRDIMM MRD Cycle Training

This training finds the correct cycle to maintain the set Read Latency value at the data buffer. In this training mode, the controller pre-programs the data buffer MPR registers with the expected pattern and issues read commands. The data buffer compares the read data with the expected data and feeds back the result on the DQ bus. Calibration selects the correct cycle based on the result of the comparison.

There are no debug registers associated with this calibration stage.

## LRDIMM MRD Center Training

This training aligns Read MDQS in the center of the Read MDQ window at the data buffer. In this training mode, the controller pre-programs the data buffer MPR registers with the expected pattern and issues the commands. The data buffer compares the read data with the expected data and feeds back the result on the DQ bus. Calibration finds the left and right edges of the Read MDQ valid window and centers Read MDQS in it.

Table 38: MRD Center Training Register

Register Name	Quantity	Description
Fx_DB_MRD_CENTER	Rank	Data Buffer MRD taps [4:0]

## LRDIMM DWL Training

This training aligns the Write MDQS phase with the DRAM clock. In this training mode, the data buffer drives the MDQS pulses, the DRAM samples the clock with MDQS and feeds back the result on MDQ. The data buffer forwards this result from MDQ to DQ. Calibration continues to perform this training to find a 0 to 1 transition on the clock sampled with the Write MDQS at the DRAM.



Table 39: DWL Training Register

Register Name	Quantity	Description
Fx_DB_DWL_MWD_LAT	Rank	Data Buffer phase [8:6] and Data Buffer to DRAM latency [5:0]

### LRDIMM MWD Cycle Training

This training finds the correct cycle to maintain the set Write Latency value in the DRAM. In this training mode, the controller pre-programs the data buffer MPR registers with the expected pattern, issues write commands to load the data into memory, and issues reads to the memory. The data buffer compares the read data with the expected data and feeds back the result on to the DQ bus. Calibration identifies the correct cycle based on the result of the comparison.

Table 40: MWD Cycle Training Register

Register Name	Quantity	Description
Fx_DB_DWL_MWD_LAT	Rank	Data Buffer to DRAM latency [8:6] and Data Buffer phase [5:0]

### LRDIMM MWD Center Training

This training center aligns Write MDQS in the Write MDQ window at the DRAM. In this training mode, the controller pre-programs the data buffer MPR registers with the expected pattern, issues write commands to load the data into memory, and issues reads to the memory. The data buffer compares the read data with the expected data and feeds back the result on the DQ bus. Calibration finds the left and right edges of the MDQ valid window and centers MDQS in it.

Table 41: MWD Center Training Register

Register Name	Quantity	Description
Fx_DB_MWD_CENTER	Rank	Data Buffer MWD Taps [4:0]

### DQS Gate

The internal clock strobe is adjusted to align with the DQS returning from the memory. This is performed on all byte lanes in parallel.

This is accomplished as follows:

1. Search for the 3rd DQS edge using the initial read latency setting -2 and incrementing coarse taps.
2. Search for the noise region using fine taps, and then center on the noise region.
3. Adjust the read latency setting (+2) to restore alignment with the first DQS edge as well as account for the detected preamble type (1 or 2 TCK).
4. Repeat for each memory rank.

- Adjust the coarse and read latency values to account for the longest delay of all the ranks.

**Table 42: DQS Gate Registers**

Register Name	Quantity	Description
Fx_DQSGATE_STG1_OVERFLOW	Rank and Byte	1 of 16 coarse taps have been used to find the 3rd DQS edge
Fx_DQSGATE_STG1_READ_LAT	Rank and Byte	The read latency value after centering on the noise region
Fx_DQSGATE_STG1_RLDLYRNK_CRSE	Rank and Byte	The coarse tap value for the 3rd DQS edge
Fx_DQSGATE_STG1_RLDLYRNK_FINE	Rank and Byte	The fine tap value for the 3rd DQS edge
Fx_DQSGATE_STG2_READ_LAT	Rank and Byte	Read Latency after adjusting for Preamble type (1 or 2TCK)
Fx_DQSGATE_STG2_RLDLYRNK_CRSE	Rank and Byte	The coarse tap value after adjusting for Preamble type
Fx_DQSGATE_MAX_READ_LAT	1	The largest read latency value for all calibrated ranks. For a single rank interface, this value will match DQSGATE_STG2_READ_LAT
Fx_DQSGATE_READ_LAT_FINAL	Byte	Final Read Latency value
Fx_DQSGATE_RLDLYRNK_CRSE_FINAL	Rank and Byte	Final coarse tap value

## Write Leveling

Each DQS is adjusted to be aligned with CK. The memory is placed in read levelling calibration mode, so the DQ values returned are the CK being sampled by the DQS at the memory. DQS is then adjusted to align with the CK at the memory.

This is accomplished as follows:

- Search for a stable 1 on the DQ by incrementing coarse taps.
- Decrement coarse taps to find a stable 0.
- Use fine taps to center on the noise region.

**Table 43: Write Leveling Registers**

Register Name	Quantity	Description
Fx_WRLVL_CRSE_STG1	Rank and byte	Is this the Coarse Tap setting after finding the stable 1
Fx_WRLVL_OFFSET	Rank and byte	90 degree offset in fine taps
Fx_WRLVL_CRSE_FINAL	Rank and byte	Stable 0 in coarse taps
Fx_WRLVL_NOISE_FCRSE	Rank and byte	Output delay value for valid-to-noise
Fx_WRLVL_FINE_LEFT	Rank and byte	The Fine Tap setting for the left side of the noise window
Fx_WRLVL_FINE_RIGHT	Rank and byte	The Fine Tap setting for the right side of the noise window
Fx_WRLVL_FINE_FINAL	Rank and byte	The final Fine Tap setting

## Read DQ Per-Bit Deskew and Centering (Simple)

Each DQ bit is sampled by the internal clock strobe calibrated during the earlier DQS gate stage. The read data is passed through delay elements and are shifted to properly read the data, then the strobes are shifted to center in the data region.

This is accomplished as follows:

1. Read the pre-written 101010 pattern from the DRAM Mode Register (MR32 and MR40 for LPDDR4, MPR0 for DDR4).
2. Increment the internal clock coarse taps until the proper data is seen on rising and falling clock edges.
3. Use the delay elements of all bits in parallel to find the noise region for both the rising and falling edges.
4. Increment coarse taps to find the noise-to-valid for all bits.
5. Increment coarse taps until the valid-to-noise region is found for any bit.
6. Use fine taps to set the rising and falling clocks in the center of the data window.

**Table 44: Read DQ Per-Bit Deskew and Centering (Simple) Registers**

Register Name	Quantity	Description
Fx_RDDQ_QTR_DESKEW	Nibble	Coarse taps needed to find the valid region of rising edge clock
Fx_RDDQ_IDELAY_FINAL	DQ bit	Final IDELAY values for each data bit in the read path
Fx_RDDQ_PQTR_LEFT	Nibble	Left edge of valid window for rising edge clock
Fx_RDDQ_NQTR_LEFT	Nibble	Left edge of valid window for falling edge clock
Fx_RDDQ_PQTR_RIGHT_FCRSE	Nibble	Valid-to-noise coarse tap value for rising edge clock
Fx_RDDQ_NQTR_RIGHT_FCRSE	Nibble	Valid-to-noise coarse tap value for falling edge clock
Fx_RDDQ_PQTR_RIGHT	Nibble	Right edge of valid window for rising edge clock
Fx_RDDQ_NQTR_RIGHT	Nibble	Right edge of valid window for falling edge clock
Fx_RDDQ_PQTR_FINAL	Nibble	Final delay value for rising edge clock strobe
Fx_RDDQ_NQTR_FINAL	Nibble	Final delay value for falling edge clock strobe

## Write DQ/DBI Per-bit Deskew and Centering (Simple)

The data and clock strobe need to be centered at the receiving memory to ensure the more reliable data transfer. The starting point is a 90 degree offset from DQS and from there adjust the delays on the DQ and DBI pins until the valid data window is found, then center the delays.

This is accomplished as follows:

1. Find the valid DQ window using DQS delay.
2. Revert the DQS delay if valid window not found.
3. Find the valid DQ window using the DQ delay and then center the DQ bits.
4. Find the valid DBI window using DQS delay.
5. Revert the DQS delay if valid window not found.
6. Find the valid DBI window using the DBI delay and then center the DBI bits.

**Table 45: Write DQ/DBI Per-bit Deskew and Centering (Simple) Registers**

Register Name	Quantity	Description
Fx_WRDQDBI_STG1_DQS_DELAY	Byte	DQS odelay after finding valid window by moving DQS
Fx_WRDQDBI_STG1_BYTE_STATUS	2	Value of 1 if a valid window is found. One bit per byte lane
Fx_WRDQDBI_STG1_BIT_STATUS	DBI pins	Value of 1 if a valid window is found
Fx_WRDQDBI_STG2_DQS_ODLY	Byte	Odelay for DQS after reverting
Fx_WRDQDBI_STG2_DQ_ODLY	DQ bit	Odelay for DQ after reverting
Fx_WRDQDBI_STG3_DQ_ODLY	DQ bit	Odelay value for the valid window using DQ delay
Fx_WRDQDBI_DESKEW_DQ_ODLY_FCRSE	DQ bit	Deskew delay 10-tap increments
Fx_WRDQDBI_DESKEW_DQ_ODLY	DQ bit	Deskew delay
Fx_WRDQDBI_STG4_DQS_DELAY	DBI pin	DQS Odelay value for the valid window
Fx_WRDQDBI_STG4_BYTE_STATUS	2	1 if a valid window is found. 1 bit per byte lane
Fx_WRDQDBI_STG5_DQS_ODLY	DBI pin	Odelay after reverting DQS
Fx_WRDQDBI_STG6_DBI_ODLY	DBI pin	DBI Odelay value for valid window using
Fx_WRDQDBI_DESKEW_DBI_ODLY_FCRSE	DBI pin	Deskew delay 10-tap increments
Fx_WRDQDBI_DESKEW_DBI_ODLY	DBI pin	Deskew delay
Fx_WRDQDBI_LEFT_MARGIN	Byte	Smallest margin left of center for the byte in Odelay taps
Fx_WRDQDBI_LEFT_EDGE_DQ	DQ bit	Odelay value for left edge of DQ
Fx_WRDQDBI_LEFT_EDGE_DBI	DBI bit	Odelay value for left edge of DBI
Fx_WRDQDBI_RIGHT_MARGIN_FCRSE	Byte	Right margin DQS Odelay with 10-tap increments
Fx_WRDQDBI_RIGHT_MARGIN	Byte	Smallest margin right of center for the byte in Odelay taps
Fx_WRDQDBI_RIGHT_EDGE_DQS	Byte	DQS Odelay value for the right edge
Fx_WRDQDBI_ODLY_DQS_FINAL	Byte	Final DQS Odelay value
Fx_WRDQDBI_ODLY_DQ_FINAL	DQ bit	Final DQ Odelay value
Fx_WRDQDBI_ODLY_DBI_FINAL	DBI bit	Final DBI Odelay value

## Write Latency Calibration

The data has been aligned to the DQS strobe, however the address and command use a fly-by topology which allows for varying arrival times at the first and the last memory component. This stage addresses the possible mismatch caused by this.

This is accomplished as follows:

1. Send a data pattern of all 0's, with 1's before the expected data and 1's after the expected data.
2. Read the data back and compare with expected 0's.
3. Add or subtract coarse taps depending on the presence and location of 1's in the read back data.

**Table 46: Write Latency Calibration Registers**

Register Name	Quantity	Description
Fx_WRLAT_INIT_LATENCY	1	Initial write latency value before calibration
Fx_WRLAT_MATCH	Rank and byte	Write latency value
Fx_WRLAT_MIN_LATENCY	1	Minimum latency for the interface
Fx_WRLAT_XPI_OE_ALL_FINAL	1	Final Output Enable latency for the interface
Fx_WRLAT_XPI_WRDATA_ALL_FINAL	1	Final write data latency for the interface
Fx_WRLAT_PHY_OE_NIB_FINAL	Byte	Final Output Enable latency
Fx_WRLAT_PHY_DATA_NIB_FINAL	Byte	Final write data latency
Fx_WRLAT_WLDLYRNK_CRSE_FINAL	Rank and byte	Final coarse delay

## Read DBI Calibration

The DBI pins need to be centered for the same reason as the DQ pins. A small DBI margin affects the validity of the entire byte line.

This is accomplished as follows:

1. A toggling data pattern is written that will create a 101010 pattern on the DBI pin. Delay the DBI if the toggling pattern is not observed.
2. After the toggling pattern is detected, continue to shift the DBI until the noise region is found. This establishes the left margin.
3. Remove the left margin taps, and increment the clock strobe until the noise region is found. This establishes the right margin.
4. Revert the clock strobe delays, and compare left and right margins to determine the ideal centering.

There are no debug registers associated with this calibration stage.

## Read Centering (Complex)

The final stage of DQS read centering that is completed before normal operation repeats the steps performed during MPR DQS read centering but with a difficult/complex data pattern. The purpose of using a complex pattern is to stress the system for SI effects such as ISI and noise while calculating the read DQS center position. This ensures that the read center position can reliably capture data with margin in a true system.

This is accomplished as follows:

1. Reset PQTR and NQTR delays to 0 and verify that data is being properly read.
2. Delay the DQ/DBI until the noise region is found. Use short read bursts until noise region is found, then use long reads for accurate edge detection.
3. Delay the clock strobe to return to the valid region, continue until the noise region on the other side is found. Use short read bursts until noise region is found, then use long reads for accurate edge detection.
4. Center the strobe in the valid region.

**Table 47: Read Centering (Complex) Registers**

Register Name	Quantity	Description
Fx_RDCMPLX_IDELAY_OFFSET	Nibble	Idelay at left edge. (in noise region)
Fx_RDCMPLX_PQTR_LEFT_SHORT	Nibble	Tap value for left side of window with short complex pattern, rising edge (noise to valid)
Fx_RDCMPLX_NQTR_LEFT_SHORT	Nibble	Tap value for left side of window with short complex pattern, falling edge (noise to valid)
Fx_RDCMPLX_PQTR_LEFT	Nibble	Tap value for left side of window with long complex pattern, rising edge (noise to valid)
Fx_RDCMPLX_NQTR_LEFT	Nibble	Tap value for left side of window with long complex pattern, falling edge (noise to valid)
Fx_RDCMPLX_PQTR_RIGHT_SHORT_FCRSE	Nibble	Short pattern with 10-increment steps for right edge, rising edge (valid to noise)
Fx_RDCMPLX_NQTR_RIGHT_SHORT_FCRSE	Nibble	Short pattern with 10-increment steps for right edge, falling edge (valid to noise)
Fx_RDCMPLX_PQTR_RIGHT_SHORT	Nibble	Tap value for right side of window with short complex pattern, rising edge (valid to noise)
Fx_RDCMPLX_NQTR_RIGHT_SHORT	Nibble	Tap value for right side of window with short complex pattern, falling edge (valid to noise)
Fx_RDCMPLX_PQTR_RIGHT	Nibble	Tap value for right side of window with long complex pattern, rising edge (valid to noise)
Fx_RDCMPLX_NQTR_RIGHT	Nibble	Tap value for right side of window with long complex pattern, falling edge (valid to noise)
Fx_RDCMPLX_PQTR_FINAL	Nibble	Final centered tap value for PQTR
Fx_RDCMPLX_NQTR_FINAL	Nibble	Final centered tap value for NQTR

## Write Centering (Complex)

A more aggressive (complex) data pattern is used to stress the system for SI effects such as ISI and noise while centering the DQS within the DQ window at the memory.

This is accomplished as follows:

1. Clear any delays from the DQ and DBI.
2. Delay DQ and DBI to find a valid window for the byte lane (left side of valid window).

3. Continue delaying DQ and DBI until the right side of noise region is found (right side of valid window).
4. Calculate and center the DQ and DBI delays.

**Table 48: Write Centering (Complex) Registers**

Register Name	Quantity	Description
Fx_WRCMPLX_RIGHT_MARGIN_FCRSE	Byte	Taps of right side margin - 10 tap increments
Fx_WRCMPLX_RIGHT_MARGIN	Byte	Taps of right side margin
Fx_WRCMPLX_LEFT_MARGIN_FCRSE	Byte	Taps of left side margin - 10 tap increments
Fx_WRCMPLX_LEFT_MARGIN	Byte	Taps of left side margin
Fx_WRCMPLX_ODLY_DQS_FINAL	Byte	Final delay value for DQS
Fx_WRCMPLX_ODLY_DQ_FINAL	DQ Pin	Final delay value for DQ bit
Fx_WRCMPLX_ODLY_DBI_FINAL	DBI Pin	Final delay value for DBI bit

## VT Tracking

Calibration is performed at device configuration, but during normal operation the voltage and temperature may change. Voltage and temperature (VT) drift can have an effect on the alignment of the DQS with the DQ bits. VT tracking is a feature where the internal clocks that have been matched with the memory clock strobes (DQS) are monitored and adjusted to track any variations from the original calibrated values.

**Table 49: VT Tracking Registers**

Register Name	Quantity	Description
VTTRACK_RLDLYQTR	Nibble	The number of fine taps in a quarter clock period
VTTRACK_RLDLYQTR_MAX	Nibble	The largest number of fine taps in a quarter clock period since the tracking has started
VTTRACK_RLDLYQTR_MIN	Nibble	The smallest number of fine taps in a quarter clock period since the tracking has started

## DQS Gate Tracking

The DQS sent from the memory is aligned with an internal clock at calibration time, however voltage and temperature may change during operation which will affect the alignment. DQS gate tracking monitors the returning DQS and makes adjustments to the internal clock as necessary to track any variations.

**Table 50: DQS Gate Tracking Registers**

Register Name	Quantity	Description
DQSTRACK_RLDLYRNK_CRSE	Byte	The number of coarse taps for the internal clock
DQSTRACK_RLDLYRNK_FINE	Byte	The number of fine taps for the internal clock

Table 50: DQS Gate Tracking Registers (cont'd)

Register Name	Quantity	Description
DQSTRACK_RLDLYRNK_CRSE_MAX	Byte	The largest number of coarse taps for the internal clock since the tracking started
DQSTRACK_RLDLYRNK_FINE_MAX	Byte	The largest number of fine taps for the internal clock since the tracking started
DQSTRACK_RLDLYRNK_CRSE_MIN	Byte	The smallest number of coarse taps for the internal clock since the tracking started
DQSTRACK_RLDLYRNK_FINE_MIN	Byte	The smallest number of fine taps for the internal clock since the tracking started

## LPDDR4 OSC Tracking

The memory controller will periodically read the oscillator values from the DRAM and if enough variation is detected, adjust the write latency values to compensate. This adjustment is done to one rank while activity is being performed on the other.

Table 51: LPDDR4 OSC Tracking Registers

Register Name	Quantity	Description
LP4DQSOSCTRACK_WLDLYRNK0_FINE	Nibble	Fine taps for WLDLYRANK0
LP4DQSOSCTRACK_WLDLYRNK0_FINE_MAX	Nibble	Largest value of fine taps for WLDLYRANK0 since calibration completed
LP4DQSOSCTRACK_WLDLYRNK0_FINE_MIN	Nibble	Smallest value of fine taps for WLDLYRANK0 since calibration completed
LP4DQSOSCTRACK_WLDLYRNK1_FINE	Nibble	Fine taps for WLDLYRANK1
LP4DQSOSCTRACK_WLDLYRNK1_FINE_MAX	Nibble	Largest value of fine taps for WLDLYRANK1 since calibration completed
LP4DQSOSCTRACK_WLDLYRNK1_FINE_MIN	Nibble	Smallest value of fine taps for WLDLYRANK1 since calibration completed



# Additional Resources and Legal Notices

---

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

---

## References

These documents provide supplemental material useful with this product guide:

1. *Versal ACAP SelectIO Resources Architecture Manual* ([AM010](#))
  2. *Versal ACAP Technical Reference Manual* ([AM011](#))
  3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))
  4. *AMBA AXI4-Stream Protocol Specification* ([ARM IHI 0051A](#))
  5. *Versal ACAP PCB Design User Guide* ([UG863](#))
  6. *Versal ACAP Schematic Review Checklist* ([XTP546](#))
  7. [Versal ACAP Design Process Documentation](#)
  8. [Getting Started with Versal Memory Interfaces](#)
  9. [Introduction to NoC DDRMC Design Flow](#)
  10. [Obtaining and Verifying Versal ACAP Memory Pinouts](#)
- 

## Revision History

The following table shows the revision history for this document.

Section	Revision Summary
<b>11/08/2021 Version 1.0</b>	
General Updates	Updates for release 2021.2.
<a href="#">Data Poisoning</a>	New section.
<a href="#">Periodic Reads</a>	New section.
<a href="#">XPLL</a>	New section.
<a href="#">Chapter 9: System Address Map</a>	Section title change.
<a href="#">UART Debug</a>	New section.
<b>08/12/2021 Version 1.0</b>	
<a href="#">Chapter 3: NoC Architecture</a>	<ul style="list-style-type: none"> <li>Added AXI Conversion Section</li> <li>General updates for clarification. Restructured the section to split out NoC and DDRMC Architectures.</li> </ul>
<a href="#">Chapter 6: NoC and Memory Controller Simulation</a>	Added Simulation content.
<b>04/08/2021 Version 1.0</b>	
<a href="#">Chapter 3: NoC Architecture</a>	General updates/clarifications.
<a href="#">Versal Programmable Network on Chip Overview</a>	<ul style="list-style-type: none"> <li>Rewrite for clarification</li> <li>Updated NoC Functions</li> </ul>
<a href="#">Chapter 6: NoC and Memory Controller Simulation</a>	New Chapter.
General updates	Restructuring for improved coherency.
<b>11/23/2020 Version 1.0</b>	
<a href="#">DDR Memory Controller</a>	<ul style="list-style-type: none"> <li>Updated Design Generation Flow for 2020.2 release.</li> <li>Updated: <a href="#">Pinout Rules</a></li> <li>Added: <a href="#">Pinout Options for Future Expansion</a></li> </ul>
<a href="#">Appendix A: Memory Interface Debug</a>	Added: <a href="#">DDRMC Calibration Debug</a>
Customizing and Generating the Core	Added information on Interrupt and Parity options.
<b>07/16/2020 Version 1.0</b>	
Initial release	N/A

## Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any

action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

### **AUTOMOTIVE APPLICATIONS DISCLAIMER**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

### **Copyright**

© Copyright 2020-2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. PCI, PCIe and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.