Speedster7t 2D Network on Chip User Guide (UG089)

Speedster FPGAs

Preliminary Data

Achronix®
Data Acceleration

Copyrights, Trademarks and Disclaimers

Preliminary Data

This document contains preliminary information and is subject to change without notice. Information provided herein is based on internal engineering specifications and/or initial characterization data.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Table of Contents

Chapter - 1: Introduction . . . 7
2D NoC Features . . . 8
Chapter - 2: Speedster7t Peripheral 2D NoC . . . 10
Peripheral 2D NoC Features . . . 10
Modes of Operation . . . 11
Connections to 2D NoC Peripheral Ring . . . 12
FPGA Fabric Logic to GDDR6 or DDR4 Subsystems . . . 12
PCIe to GDDR6 or DDR4 Subsytems . . . 12
PCIe Endpoint to PCIe Endpoint . . . 12
PCIe Endpoint to FCU . . . 12
PCIe Endpoint to/from FPGA Fabric Logic . . . 13
FCU to All Endpoints . . . 13
Additional Features . . . 13
Addressing . . . 13
Clock Domain Crossing . . . 13
Transaction Arbitration . . . 13
Functional Prior to Configuration . . . 14
Chapter - 3: Speedster7t 2D NoC Rows and Columns . . . 15
Structure and Performance . . . 15
Modes of Operation . . . 16
AXI Mode . . . 17
Data Streaming . . . 17
Ethernet Packet Transfers . . . 18
Additional Features . . . 18
Clock Domain Crossing . . . 18
Transaction Arbitration . . . 18
Chapter - 4: Speedster7t 2D NoC Access Point . . . 20
AXI Responder NAP . . . 21
AXI Initiator NAP . . . 22
Horizontal NAP . . . 23
Vertical NAP . . . 24

Ethernet NAP 25

Chapter - 5: Speedster7t 2D NoC Connectivity 26

Interface-only Connections 26

Interface-to-Fabric Connections 27

Ethernet-to-Fabric Connections 28

Packet Mode 29

Quad-Segmented Mode 31

Fabric-to-Fabric Connections 32

AXI Transactions 33

Data Streaming 33

Chapter - 6: Speedster7t 2D NoC Address Mapping 36

Global Address Map 36

PCIe 37

DDR4 37

GDDR6 37

NAP 38

CSR Space 39

FCU 39

Control and Status Register Space 39

Address Translation 40

DDR4 41

GDDR6 41

NAP 41

Chapter - 7: Speedster7t 2D NoC Performance 42

Latency and Performance 42

Latency 42

AXI Burst Transactions 42

NAP Buffering 43

Outstanding Transactions Table 43

Flits and AXI Protocol 43

AXI Responses 44

Power 45

Chapter - 8: Speedster7t 2D NoC Simulation Support 46

NAP Bus Functional Model 46

ACX_NAP_AXI_SLAVE Responder Macro 47

ACX_NAP_AXI_MASTER Initiator Macro . . . 48
ACX_NAP_HORIZONTAL Macro . . . 48
ACX_NAP_VERTICAL Macro . . . 49
ACX_NAP_ETHERNET Macro . . . 49
Simulating 2D NoC with DSM . . . 50
Cycle-Accurate Simulations of Interface Subsystems . . . 50
Chapter - 9: Speedster7t 2D NoC Software Support . . . 51
Create Clocks and Configure the PLL . . . 51
Configure the 2D NoC . . . 52
Top-Level Configuration . . . 52
Individual Sections Clocking . . . 53
Access Controls . . . 54
Revision History . . . 56

Chapter - 1: Introduction

This guide introduces the concept of the network hierarchy feature of the Speedster®7t family of FPGAs.

The Speedster7t family of FPGAs has a network hierarchy that enables extremely high-speed data flow between the FPGA core and the interfaces around the periphery as well as between logic within the FPGA itself. This on-chip network hierarchy supports a cross-sectional bidirectional bandwidth exceeding 20 Tbps. It supports a multitude of interface protocols including GDDR6, DDR4/5, 400G Ethernet, and PCI Express Gen5 data streams while greatly simplifying access to memory and high-speed protocols. The Achronix two-dimensional network on chip (2D NoC) provides for read/write transactions throughout the device as well as specialized support for 400G Ethernet streams in selected columns.

The 2D NoC extends both vertically and horizontally over the FPGA fabric until it reaches the peripheral portion of the 2D NoC. This structure provides an easy-to-use, high-bandwidth method to communicate between various initiators and responders on a Speedster7t FPGA, including specialized connections between the Ethernet subsystem and 2D NoC access points (NAPs) on select 2D NoC columns in the FPGA fabric. In addition, the 2D NoC provides a connection from the FPGA fabric and interface subsystems to the FPGA configuration unit (FCU). The FCU receives bitstreams and is used to configure the FPGA fabric as well as the various interface subsystems on the device. The 2D NoC also provides read and write access to the control and status register (CSR) space. The CSR space includes control registers and status registers for the interface subsystems.

The features of the 2D NoC described in this user guide generally pertain to the entire Speedster7t family of FPGAs. To simplify understanding specific connections and features of the 2D NoC, this user guide focuses on the 2D NoC as implemented in the Speedster7t AC7t1500 FPGA.

Initiator Endpoints

- Up to 80 2D NoC access point (NAP) responders distributed throughout the FPGA core responding to the user-implemented initiator logic endpoint
- All PCI Express Interfaces
- FPGA configuration unit (FCU)

Responder Endpoints

- Up to 80 NAP initiators distributed throughout the FPGA core communicating with the user-implemented responder logic endpoint
- Up to 16x GDDR6 memory interfaces
- DDR4/5 controller
- All PCI Express Interfaces
- All control and status register (CSR) interfaces of all subsystem cores
- FCU (enables configuring of FPGA and interface subsystems)

Packet Endpoints

- Up to 80 vertical and 80 horizontal NAP packet interfaces distributed throughout the FPGA core for fabric-to-fabric transactions
- Up to 32 of the 80 vertical NAPs can send and receive data to/from the Ethernet subsystems, each Ethernet controller connects to two dedicated 2D NoC columns
- Up to two Ethernet subsystems, supporting a mix of up to 4× 400 Gbps Ethernet or 16× 100 Gbps Ethernet

2D NoC Features

While the main purpose of the 2D NoC is to provide high-bandwidth connections between various endpoints on a Speedster7t device, it also includes features for ease-of-use and flexibility. The NAPs that provide the 2D NoC-to-FPGA interface can operate in several different modes:

- 256-bit advanced extensible Interface (AXI) responder
- 256-bit AXI initiator
- Ethernet packet
- NAP-to-NAP data streaming

These different modes provide a built-in means of communication between endpoints without requiring the design of the needed logic. The 2D NoC also handles flow control internally such that data is never dropped. Additionally, each NAP has its own address translation table providing both flexibility in addressing, as well as security through the ability to block access to specific memory regions on a per-NAP basis.

The following figure illustrates the 2D NoC surrounded by high-speed interfaces on a Speedster7t AC7t1500 FPGA, and the rows and columns of the 2D NoC over the FPGA fabric.

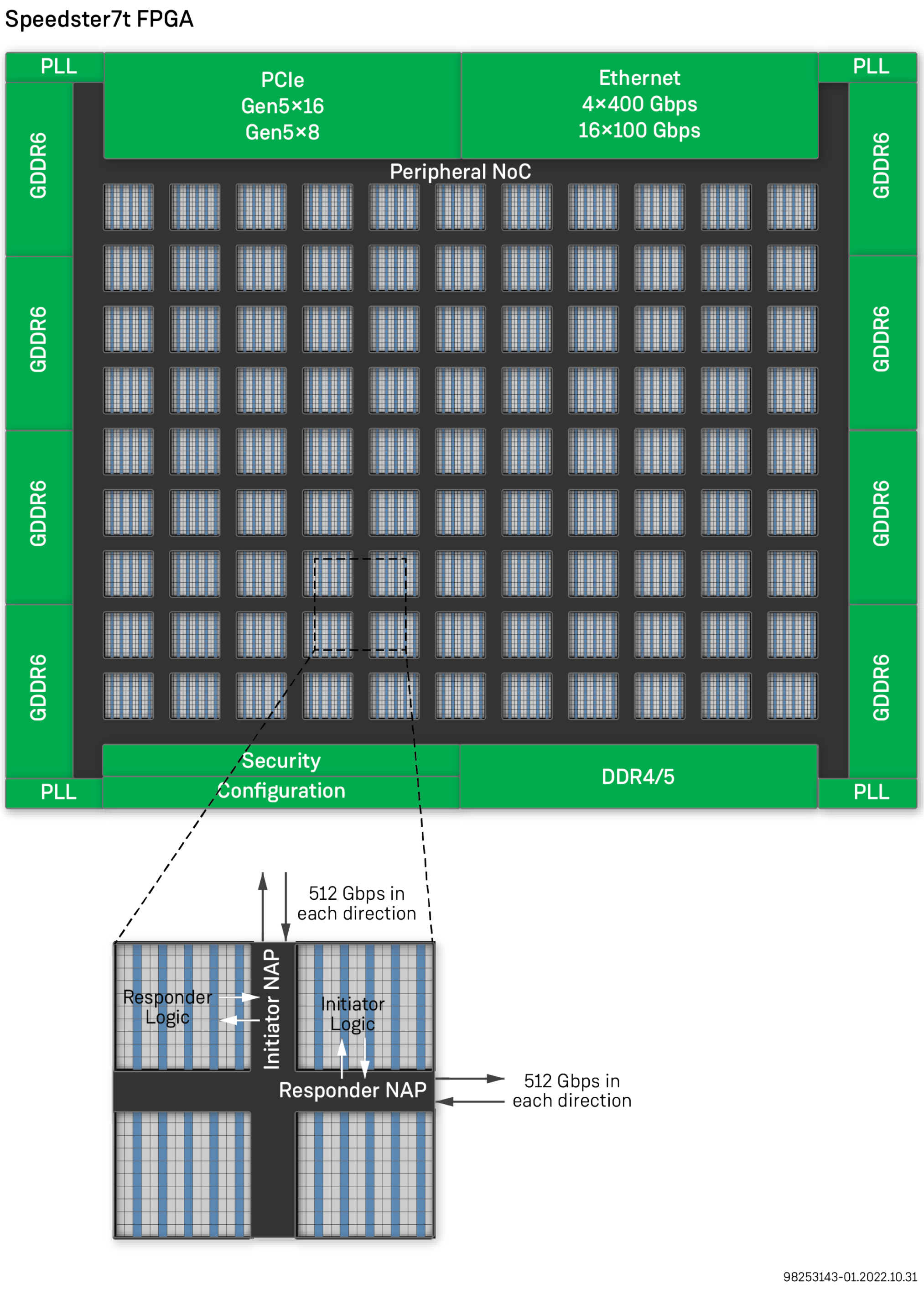


Figure 1: ***Speedcore7t 2D NoC Showing Initiator and Responder Endpoints***

Chapter - 2: Speedster7t Peripheral 2D NoC

The Achronix Speedster7t FPGA 2D NoC consists of two main parts:

- The peripheral ring around the fabric that connects to all the IP interfaces.
- The rows and columns that run over the top of the FPGA fabric.

This section describes the peripheral ring of the 2D NoC, along with its connections and features.

Peripheral 2D NoC Features

The peripheral portion of the 2D NoC forms a ring around the FPGA fabric, but operates entirely without consuming any FPGA resources. This ring provides a 256-bit wide primary data-path that runs at 2 GHz, implemented with six full crossbar switches allowing access to all endpoints connected to the 2D NoC. In addition, It has built-in clock domain crossing logic to handle the different endpoint frequencies, built-in address decoding using a global address map, and built-in arbitration to keep traffic moving at high speeds.

While the peripheral portion of the 2D NoC can be used without configuring the fabric, this portion also connects directly to the rows and columns of the 2D NoC that run over the FPGA fabric, providing access to initiator and responder logic in the FPGA. Additionally, the peripheral ring of the 2D NoC connects to the FPGA configuration unit (FCU), allowing the 2D NoC to aid in configuration of the FPGA fabric or the various interfaces. To see how the peripheral ring connects to the rows and columns, see the figure in the chapter Speedster7t 2D NoC Rows and Columns (see page 16). The following figure shows the peripheral portion of the 2D NoC as it surrounds the FPGA fabric and provides high-bandwidth connections to the memory and networking interfaces.

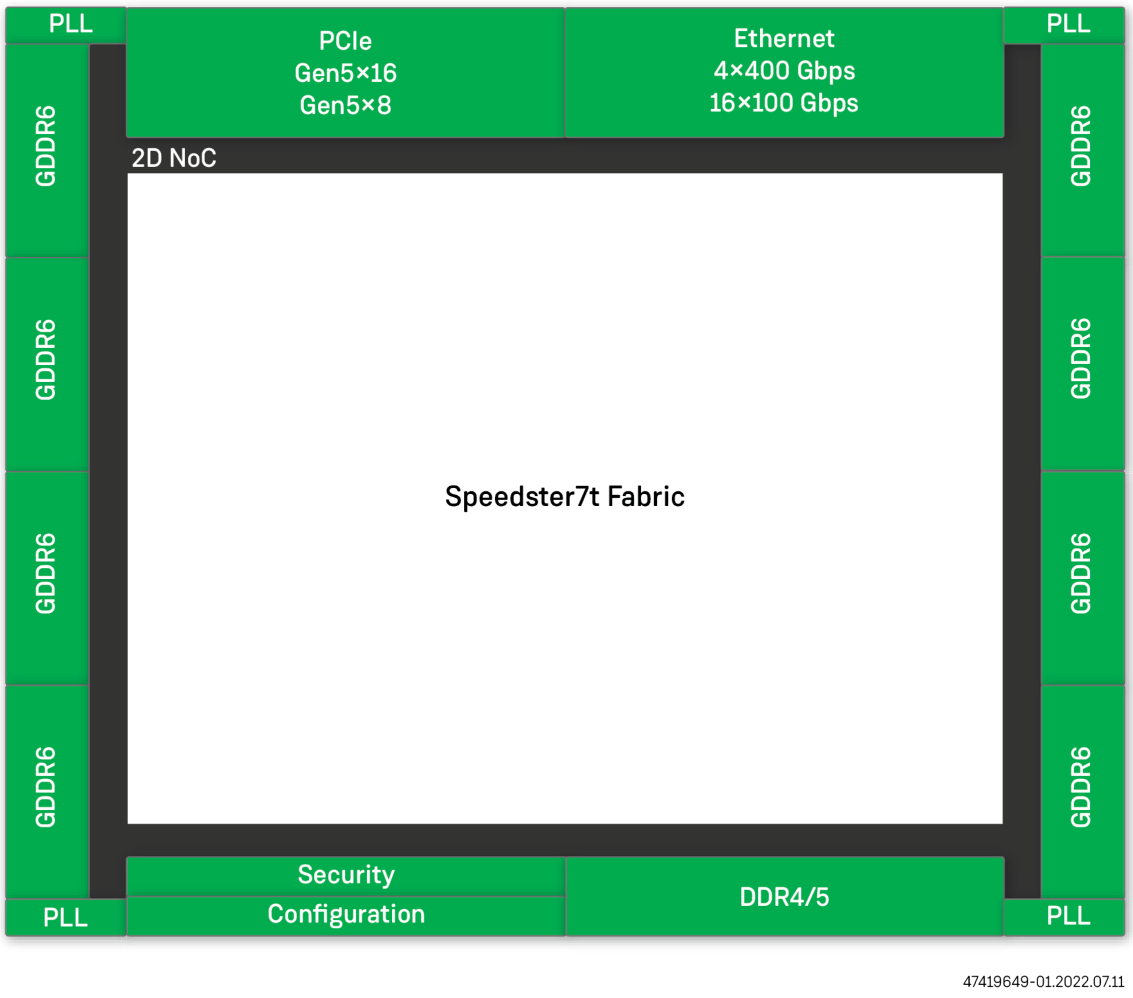


Figure 2: ***Speedster7t 2D NoC Peripheral Ring***

Modes of Operation

The 2D NoC supports AXI4 initiator/responder interfaces with read and write transactions. As implied, initiators initiate commands and responders respond to commands by either writing the provided data or sending the requested read data. This mechanism provides easy-to-use connections between all of the interfaces without needing to design complicated logic to communicate with each interface separately. As mentioned above, this mode of operation provides for a 256-bit main data path operating at up to 2 GHz.

Additionally, the 2D NoC connects to the advanced peripheral bus (APB) interface used to configure and collect status from all of the interface subsystem control and status registers (CSRs) in the device. While this interface operates at a lower frequency, it is expected to only be used in limited scenarios. If using the PCIe interface (e.g., to program CSRs), the 2D NoC handles all of the translation from AXI4 transactions to the APB.

Connections to 2D NoC Peripheral Ring

The 2D NoC allows designers to easily communicate between the various device interfaces, as well as connecting the fabric to any of the interfaces on the device, all without using logic or routing resources in the FPGA fabric. The peripheral portion of the 2D NoC connects endpoints using an initiator/responder model, where the initiator initiates transactions and the responder responds to transactions. The peripheral portion of the 2D NoC can connect the following endpoints:

- GDDR6
- DDR4
- PCIe
- Rows and columns of the 2D NoC to fabric logic
- FPGA configuration unit (FCU)
- CSRs in the entire FPGA

FPGA Fabric Logic to GDDR6 or DDR4 Subsystems

Initiator logic in the FPGA fabric can initiate transactions to any of the GDDR6 channels or the DDR4 interface. The user logic sends a transaction to the NAP connected to a row of the 2D NoC. This transaction then travels east or west on the row until it reaches the peripheral portion of the 2D NoC, and then to the destination GDDR6 or DDR4 channel.

PCIe to GDDR6 or DDR4 Subsytems

Either PCIe endpoint can initiate transactions to either GDDR6 or DDR4 directly using the 2D NoC. In this case, the PCIe endpoint is the initiator with either the GDDR6 or DDR4 as the responder. The 2D NoC is able to provide enough bandwidth to sustain PCIe Gen 5 traffic connecting to two channels of GDDR6. This high-bandwidth connection is achieved without consuming any FPGA fabric resources. It is only necessary to enable PCIe, GDDR6, and/or DDR4 to send transactions on the 2D NoC.

PCIe Endpoint to PCIe Endpoint

Because the Speedster7t AC7t1500 FPGA contains two independent PCIe controllers, each PCIe controller can send transactions to the other via the 2D NoC. Similar to connections with GDDR6 and DDR4, this high-bandwidth connection is achieved without consuming any FPGA fabric resources — it is only necessary to enable both PCIe controllers to send transactions between them.

PCIe Endpoint to FCU

The PCIe endpoint can also connect directly to the FCU via the 2D NoC without using FPGA fabric resources. This feature allows the PCIe endpoint to send a bitstream directly to the FCU, which then configures the FPGA fabric with the bitstream.

PCIe Endpoint to/from FPGA Fabric Logic

The PCIe endpoint can connect to logic in the FPGA fabric through the 2D NoC. In this case, the PCIe endpoint can be the initiator or responder, and similarly the logic in the FPGA can be either the initiator or responder. If the PCIe endpoint is initiating transactions, the peripheral portion of the 2D NoC sends transactions down the columns of the 2D NoC to reach NAPs in the fabric. These NAPs send the transaction to the logic in the fabric and then send the responses back onto the 2D NoC. If the logic in the FPGA fabric is acting as the initiator, it can initiate transactions to the NAP on a row of the 2D NoC, which sends the transaction to the peripheral portion and then to the PCIe endpoint.

FCU to All Endpoints

The FCU can act as an initiator to all other endpoints of the 2D NoC, allowing the FCU to program the entire FPGA, including configuring the interface subsystems. For example, using the 2D NoC, the FCU can read and write the control and status register (CSR) space in the entire FPGA and can even be used to load GDDR6 or DDR4 memory. For details on how the FCU performs these transactions, refer to the appropriate interface user guide.

For additional details on all the connectivity available in the 2D NoC, refer to the chapter, Speedster7t 2D NoC Connectivity (see page 26).

Additional Features

The 2D NoC provides several features that make it easy to use without sacrificing on area, congestion, or design time.

Addressing

The 2D NoC provides address decoding using a global address map to ensure transactions are sent to their intended destination. Additionally, the 2D NoC supports address translation for flexibility and added security. For more information on the address map and address translation features, see the chapter, Speedster7t 2D NoC Address Mapping (see page 36).

Clock Domain Crossing

To make logic design easier, the 2D NoC handles all clock domain crossing internally. This capability significantly simplifies user designs while providing a way to easily transfer data operating at lower frequencies compared to the 2D NoC. Specifically, on the peripheral portion of the 2D NoC, this handles the clock crossing needed between the 2D NoC and PCIe, GDDR6, DDR4, and FCU.

Transaction Arbitration

Transaction arbitration is also handled internally by the 2D NoC. This capability keeps data moving through the 2D NoC without causing major congestion. For the peripheral portion of the 2D NoC, a FIFO-based arbitration scheme is used, meaning transactions are handled on a first-come, first-served basis. If multiple transactions arrive on the same 2D NoC clock cycle, a least-recently-serviced policy is used to order the transactions. This scheme guarantees that no endpoint is starved, and all transactions complete. The arbitration scheme in the peripheral portion of the 2D NoC is not configurable and differs somewhat from the arbitration scheme used in the rows and columns of the 2D NoC. For more information on transaction arbitration in the rows and columns, refer to the Transaction Arbitration section in the chapter, Speedster7t 2D NoC Rows and Columns (see page 18).

Functional Prior to Configuration

Additionally, the peripheral portion of the 2D NoC is operational without needing to first configure the FPGA fabric. This feature allows a host to use the PCIe endpoint to program the FPGA fabric, and further, this capability also makes possible partial reconfiguration through the peripheral portion of the 2D NoC.

Chapter - 3: Speedster7t 2D NoC Rows and Columns

The rows and columns of the 2D NoC are placed over the FPGA fabric and do not break the connectivity within the fabric. This structure allows the logic in the FPGA fabric to connect to the 2D NoC through NoC access points (NAPs). The rows and columns are connected to the peripheral portion of the 2D NoC, which communicates with the interface subsystems such as GDDR6, PCIe, and DDR4. The columns also have direct connections to the Ethernet MAC, and thus connect easily to user logic in the FPGA fabric.

Structure and Performance

The 2D NoC is placed in rows and columns at regular intervals over the FPGA fabric. The user logic connects to the 2D NoC by way of NAPs and does not interfere with the connectivity of other logic within the fabric. Each row and column has a primary 256-bit data path and additional control signals that operate at 2 GHz, delivering 512 Gbps of bidirectional bandwidth. While there are no direct connections between the rows and the columns, both connect to the peripheral ring of the 2D NoC which allows for connections between points in the fabric. Initiator logic in the FPGA fabric connects to NAPs on the horizontal rows, and responder logic in the FPGA fabric connects to NAPs on the vertical columns of the 2D NoC. The following figure shows an example of the 2D NoC as constructed in the Speedster7t AC7t1500 FPGA. As shown, there are eight rows and ten columns, providing a total of 80 NAPs on the horizontal rows and 80 NAPs on the vertical columns to which the fabric logic can connect. The result is 10 Tbps of bidirectional bandwidth going north-south and 8 Tbps of bidirectional bandwidth going east-west.

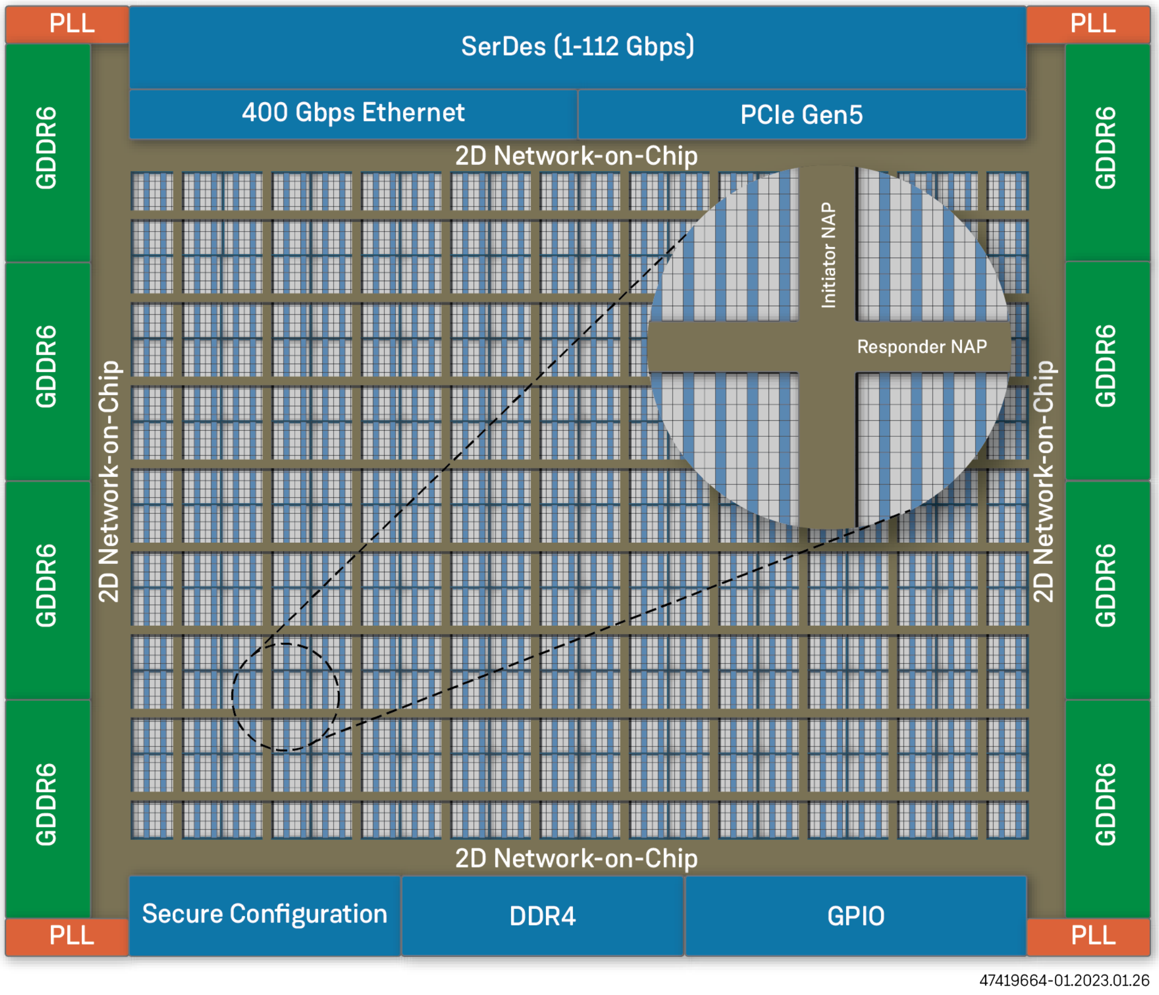


Figure 3: ***2D NoC Rows and Columns***

Modes of Operation

There are three main modes of operation in the rows and columns of the 2D NoC:

1. Industry-standard AXI-4 interface protocol is used to communicate from the fabric to most of the interface subsystems connected at the periphery of the 2D NoC, as well as within the fabric.
2. The internal fabric can connect to points on the same 2D NoC row or column using data streaming.
3. The Ethernet interface is connected via specific columns using Ethernet packet transfers.

These three modes are described in more detail in the following sections.

AXI Mode

AXI mode operates using initiator and responder logic. Initiator logic in the FPGA fabric can initiate commands to responder NAPs on 2D NoC rows, which send the commands to responder endpoints such as GDDR6, DDR4, PCIe, and the FCU. Similarly, responder user logic in the FPGA fabric can respond to transactions from an initiator NAP on a column of the 2D NoC sent by the PCIe or FCU endpoints. Additionally, this mode is used to send transactions from FPGA fabric user logic to other endpoints in the FPGA fabric that might not be located on the same row or column of the 2D NoC. Generally, AXI mode follows the AXI-4 standard, but there is a burst length limit of 16 for a single transaction. For more details on AXI transactions, see the AMBA AXI Protocol Specification.

Data Streaming

User logic in the FPGA can communicate with another logic block in the fabric using data streaming on a single row or column of the 2D NoC. In this case, intra-FPGA transfers act like a distributed FIFO. The start point and endpoint must be on the same row or same column of the 2D NoC, using a simple signaling protocol. This protocol uses a "valid" signal to indicate valid data being sent for the transfer and a "ready" signal to accept the data or signal back pressure. The start point sends a destination ID to indicate which NAP on the column or row receives data, and the endpoint receives a source ID to indicate which NAP on the column or row transmitted the data. The receiving NAP does not send back any acknowledgement to the transaction, it simply accepts the data, and knows which NAP sent the data. Optionally, a start-of-packet and end-of-packet signal can be used as well. Data streaming uses all 288 bits of the data bus for rows and 293 bits of the data bus for columns.

Example timing diagrams of data streaming transactions follow, showing how transfers are captured when both the associated `ready` and `valid` signals are high.

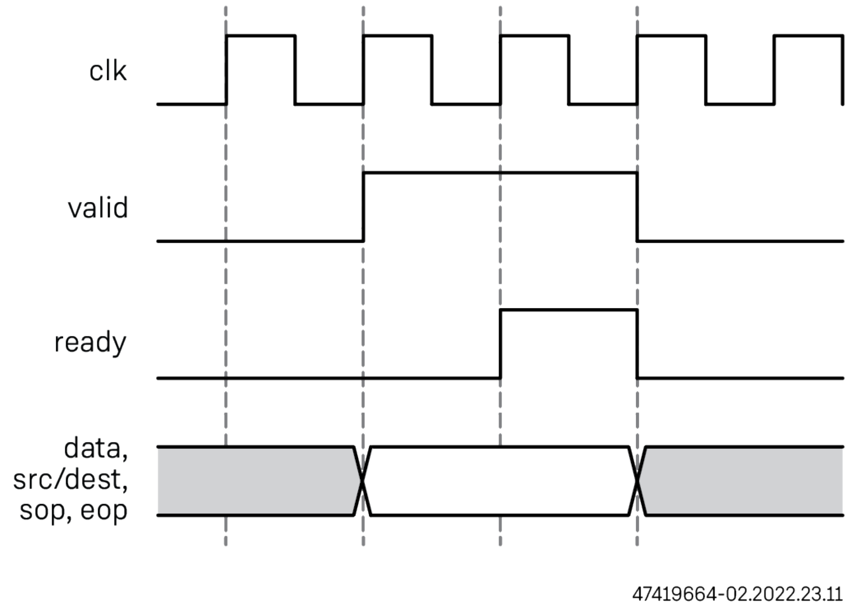


Figure 4: ***Data Streaming Timing Diagram With Valid Asserted First***

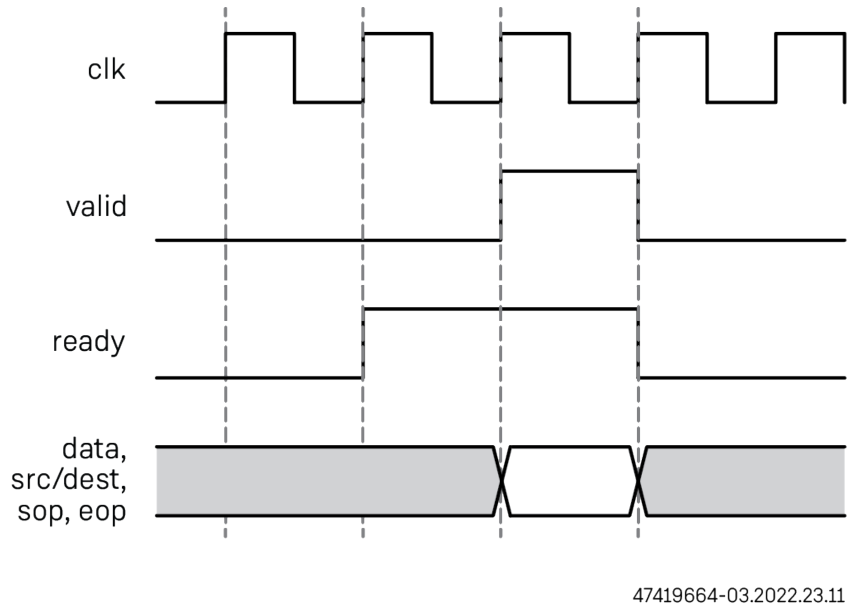


Figure 5: ***Data Streaming Timing Diagram With Ready Asserted First***

In data streaming mode, any point in the fabric can initiate the transfer, but the two points *must* reside on the same row or same column of the 2D NoC. For more details on data streaming, see the sections, Speedster7t 2D NoC Access Point (see page 20) or Speedster7t 2D NoC Connectivity (see page 26).

Ethernet Packet Transfers

The Ethernet MAC interface is connected to specific columns in the 2D NoC. The user logic in the fabric can connect to NAPs in these columns to communicate with the Ethernet MAC using Ethernet packets. This mode is very similar to the data streaming mode previously described. For more details on Ethernet packet transfers on the 2D NoC, refer to the section, Speedster7t 2D NoC Connectivity (see page 26).

Additional Features

Clock Domain Crossing

The 2D NoC handles clock domain crossing for any endpoints on the 2D NoC. This feature allows user logic operating at a lower frequency to easily connect to the 2D NoC without the need to design resource-intensive and complicated clock domain crossing logic. Simply connect the slower fabric clock to the NAP, and the rest of the clock crossing logic is handled by the 2D NoC.

Transaction Arbitration

The 2D NoC also handles transaction arbitration internally, and interleaves traffic from AXI transactions, Ethernet packets, and/or data streaming. This arbitration not only keeps traffic moving and prevents backups, but also keeps the 2D NoC operating at its peak capacity. The rows and columns use a configurable, round-robin arbitration scheme where the arbitration schedule can be configured at each NAP. The schedule values are passed via parameters when the NAP is instantiated and remain static after configuration of the fabric. For the NAPs on columns, there is a parameter for the north-to-south direction as well as the south-to-north direction. Similarly, the rows have a parameter for the east-to-west direction as well as west-to-east.

Each arbitration parameter consists of a 32-bit value used to initialize the arbitration schedule mechanism. Bit 0 of the arbitration schedule vector is used to determine if the local NAP transaction entering the 2D NoC wins arbitration when there is competing traffic from the upstream NAP on the row or column. If bit 0 has a value of `1`, the local traffic entering the 2D NoC wins, while if `0`, the upstream transaction on the row or column wins. After each 2D NoC clock cycle where both the local transaction and the upstream transaction are competing for access, the value in the schedule register rotates to the left. For example, a value of `32'h`AAAA_AAAA means that the local NAP transaction has high priority on every second 2D NoC cycle.

ACE chooses default values for the arbitration schedule to create fairness on the rows and columns, but those values can be overridden if a particular NAP needs to have higher priority in a design. It is recommended that the arbitration schedule values are not overridden, as the default values set fairness for all NAPs on a row or column. The default value for each NAP is based on the number of instantiated NAPs along a row or column, and the location of the particular NAP. The formula used for the values on each row or column that instantiates N number of NAPs is 1/N for the last NAP in that direction, 1/(N-1) to next upstream NAP, and so on until the first instantiated NAP in the row or column. For example, if three NAPs are on a row, the westernmost NAP has priority every third cycle, the next upstream NAP has priority every second cycle, and the easternmost NAP always has priority as there are no further competing NAPs in the east-to-west direction. Both ACE and the simulation environment enforce the default arbitration value unless explicitly overridden by a user value. For more information on the arbitration schedules, refer to the "Speedster7t Network on Chip Primitives" chapter in the *Speedster7t Component Library User Guide* (UG086).

Chapter - 4: Speedster7t 2D NoC Access Point

Note

Achronix is aware of the use of historical terms within this document with regards to initiator and responder logic. Achronix is working hard to remove all such references from within its code base and documentation.

The NoC access point (NAP) is the connection point from user logic in the fabric to the 2D NoC. NAPs are instantiated in user logic to connect to the rows and columns of the 2D NoC. Depending on the function, the appropriate NAP instance is instantiated in the design. The following figure shows an example of how the NAPs connect to the 2D NoC.

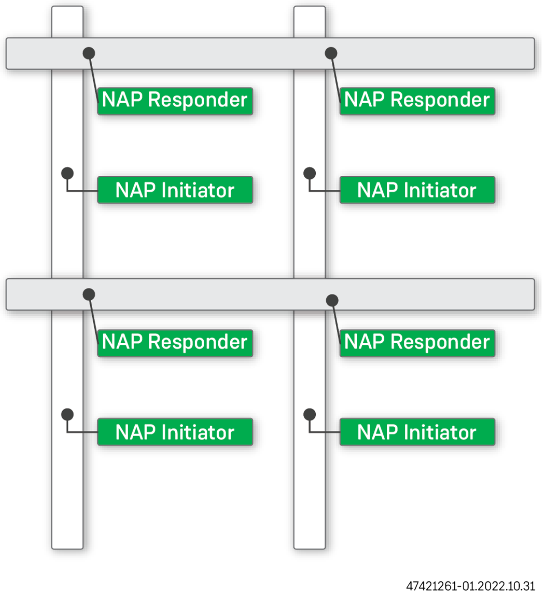


Figure 6: ***NoC Access Points in FPGA Fabric***

AXI Responder NAP

The `ACX_NAP_AXI_SLAVE` macro presents a 256-bit AXI responder to initiator user logic in the fabric and connects to the rows of the 2D NoC. The resulting connection uses standard AXI4 protocol for read and write transactions and connects the user logic to any peripherals on the 2D NoC, including the interface subsystems, as well as other user logic in the FPGA fabric connected through a NAP. The input clock drives the user logic in the FPGA fabric. The 2D NoC uses this clock for any clock-crossing logic. The following is a block diagram of the AXI responder NAP.

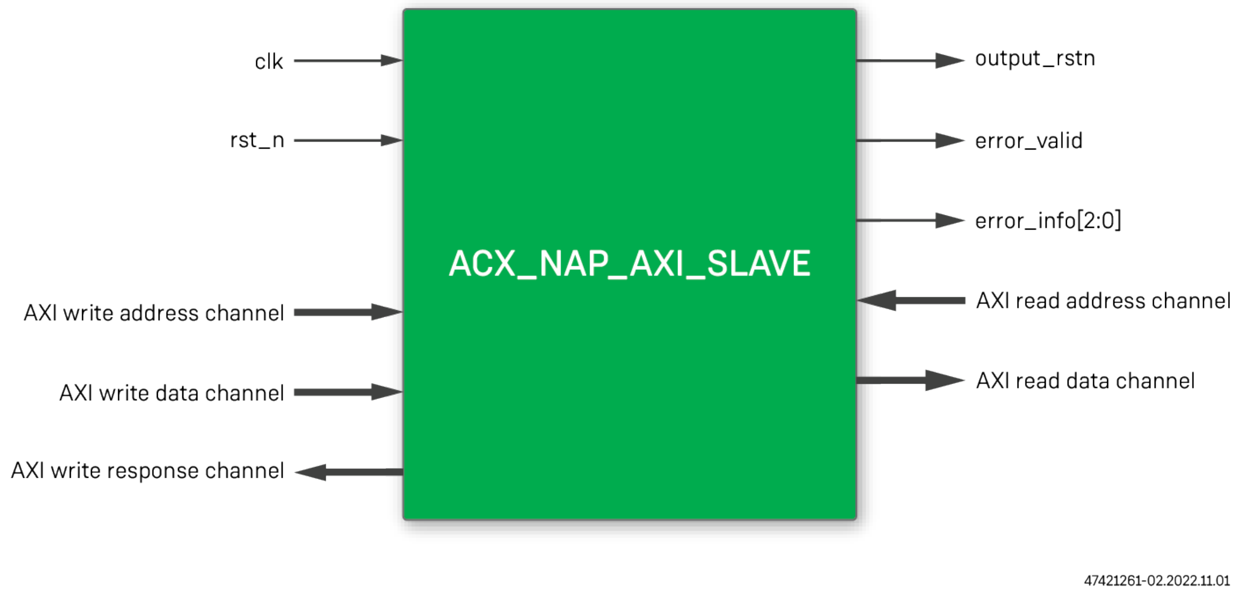


Figure 7: *AXI Responder NAP Block Diagram*

For details on port names and instantiating the component, see the "Speedster7t Network on Chip Primitives" chapter in the *Speedster7t Component Library User Guide* (UG086).

AXI Initiator NAP

The `ACX_NAP_AXI_MASTER` macro presents a 256-bit AXI initiator to responder user logic in the fabric and connects to the columns of the 2D NoC. The resulting connection uses standard AXI4 protocol for read and write transactions and connects the user logic to peripherals on the 2D NoC, including the interface subsystems, as well as other user logic in the FPGA fabric connected through a NAP. The input clock drives the user logic in the FPGA fabric. The 2D NoC uses this clock for any clock crossing logic. The following is a block diagram of the AXI initiator NAP.

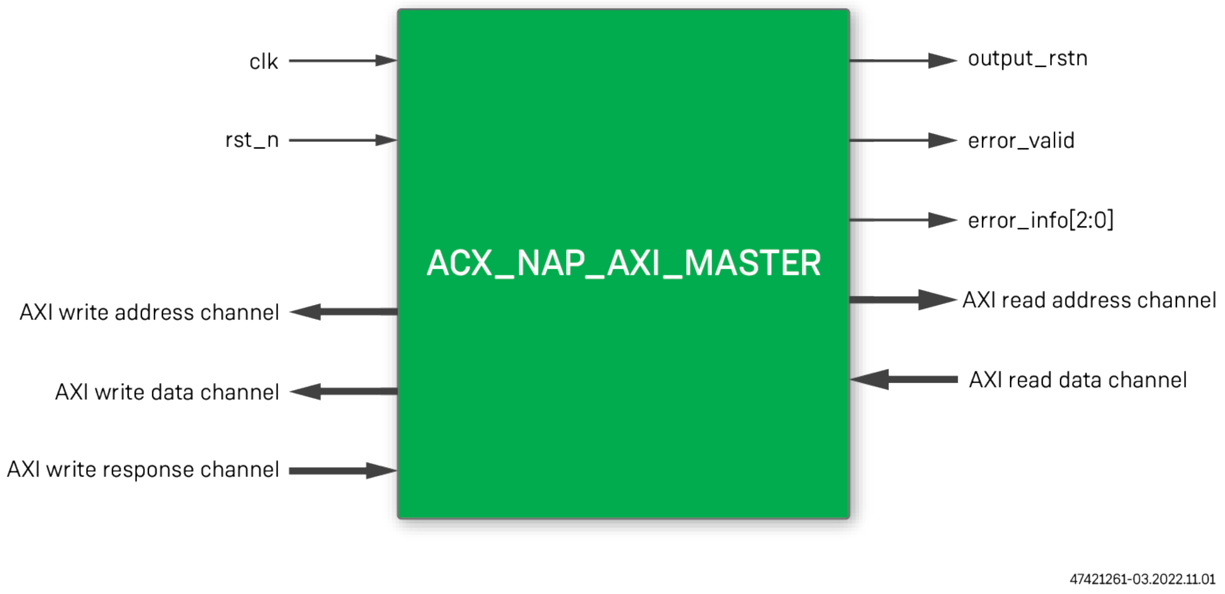


Figure 8: ***AXI Initiator NAP Block Diagram***

For details on port names and instantiating the component, see the "Speedster7t Network on Chip Primitives" chapter in the *Speedster7t Component Library User Guide* (UG086).

Horizontal NAP

The `ACX_NAP_HORIZONTAL` macro is used for data streaming along the rows of the 2D NoC. The `ACX_NAP_HORIZONTAL` macro presents a 288-bit datapath to another `ACX_NAP_HORIZONTAL` instance on the same row of the 2D NoC using transactions similar to a FIFO. User logic presents data to the interface along with a destination ID. The data and other fields are captured and sent to the destination NAP as indicated by `tx_dest [3:0]`, which then is sent to the FPGA logic using the destination NAP receiver interface. The input clock drives the user logic in the FPGA fabric. The 2D NoC uses this clock for any clock crossing logic. The following is a block diagram of the horizontal NAP.

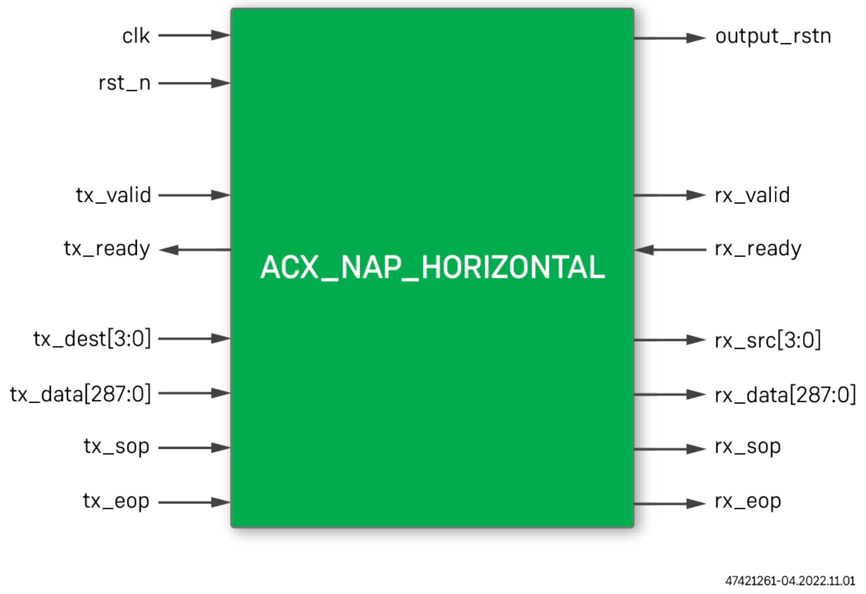


Figure 9: *Horizontal NAP Block Diagram*

For details on port names and instantiating the component, see the "Speedster7t Network on Chip Primitives" chapter in the *Speedster7t Component Library User Guide* (UG086).

Vertical NAP

The `ACX_NAP_VERTICAL` macro is used for data streaming along the columns of the 2D NoC. The `ACX_NAP_VERTICAL` macro presents a 293-bit datapath to another `ACX_NAP_VERTICAL` instance on the same column of the 2D NoC using transactions similar to a FIFO. User logic presents data to the interface along with a destination ID. The data and other fields are captured and sent to the destination NAP as indicated by `tx_dest [3:0]`, which then is sent to the FPGA logic using the destination NAP receiver interface. The input clock drives the user logic in the FPGA fabric. The 2D NoC uses this clock for any clock crossing logic. The following is a block diagram of the vertical NAP.

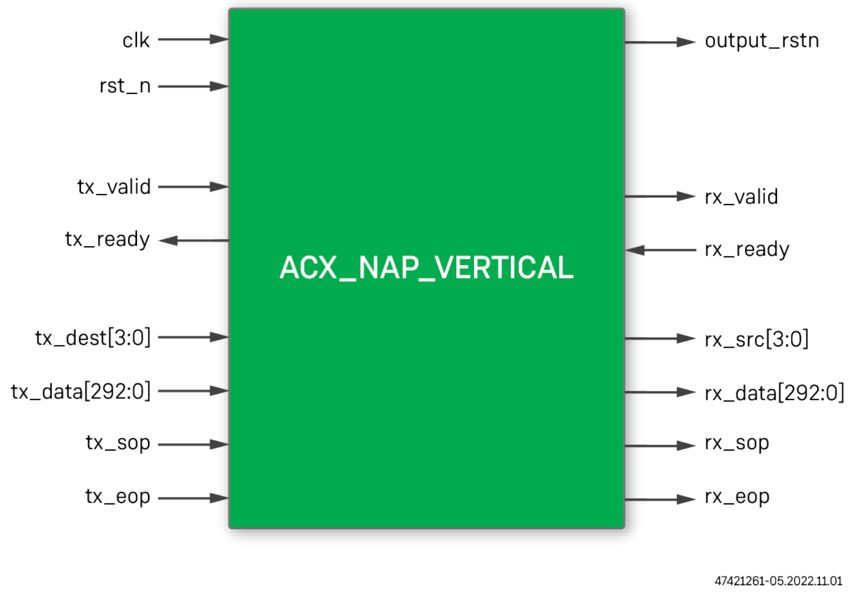


Figure 10: *Vertical NAP Block Diagram*

For details on port names and instantiating the component, see the "Speedster7t Network on Chip Primitives" chapter in the *Speedster7t Component Library User Guide* (UG086).

Ethernet NAP

The `ACX_NAP_ETHERNET` macro is used for data streaming of Ethernet packets along the columns of the 2D NoC. The `ACX_NAP_ETHERNET` macro presents a 293-bit datapath to the user logic. The `ACX_NAP_ETHERNET` is configured via parameters to connect to the Ethernet Interface Unit (EIU) at the top of the column. Ethernet packets are input and output to the `ACX_NAP_ETHERNET` using the streaming interface. For packet transmission, the destination ID must be set to `4'hf`. In addition only certain columns connect to the EIUs in a device.

Note

Although it is possible to directly instantiate the `ACX_NAP_ETHERNET`, there are a number of additional design constraints, specific to Ethernet, and the integration between the NAP, EIU and Ethernet subsystem. It is, therefore, recommended that the `ACX_ETHERNET_NODE` wrapper be instantiated. This wrapper is freely available in any Achronix reference or demonstration design that includes Ethernet. The node directly instantiates the `ACX_NAP_ETHERNET`.

The input clock drives the user logic in the FPGA fabric. The 2D NoC uses this clock for any clock crossing logic. The following is a block diagram of the Ethernet NAP.

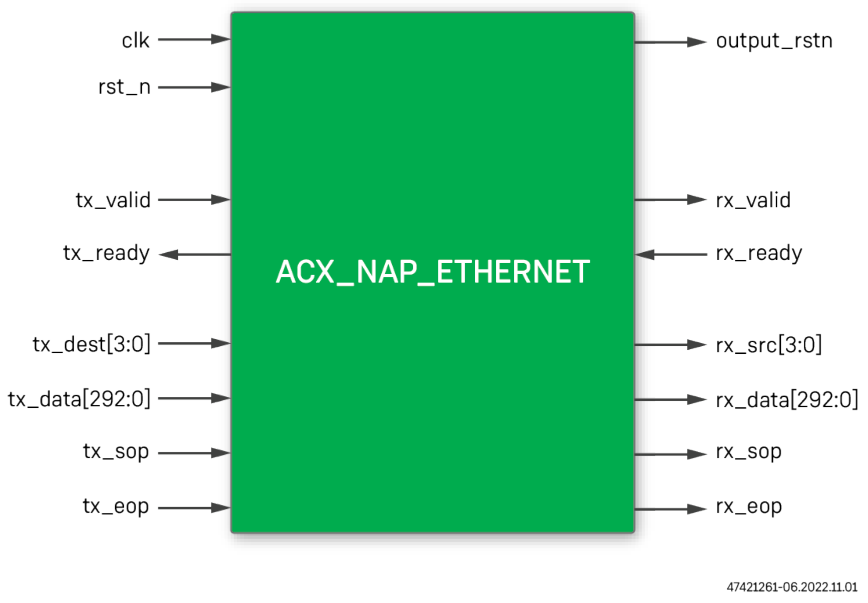


Figure 11: *Ethernet NAP Block Diagram*

For details on port names and instantiating the component, see the "Speedster7t Network on Chip Primitives" chapter in the *Speedster7t Component Library User Guide* (UG086). For details of the Ethernet subsystem, and how the ACX_NAP_ETHERNET is used to send and receive Ethernet traffic, refer to the *Speedster7t Ethernet User Guide* (UG097).

Chapter - 5: Speedster7t 2D NoC Connectivity

This section describes how the 2D NoC connects the various endpoints together, how traffic moves, and the role of the designer in optimizing a design for low congestion, low latency, and high performance. The 2D NoC connects interface-only endpoints, interface to fabric, fabric to fabric, and Ethernet to fabric.

Interface-only Connections

The 2D NoC connects certain interface endpoints without using the FPGA fabric. Interface-only connections make use of only the peripheral portion of the 2D NoC which connects PCIe to GDDR6, DDR4, and FCU. Additionally, the FCU uses the 2D NoC to access the CSR space of all interface subsystems including PCIe, GDDR6, DDR4, and Ethernet. The connections between the PCIe, FCU, GDDR6, and DDR4 use the AXI4 protocol to send transactions. GDDR6 and DDR4 endpoints can only act as responders, while the PCIe and FCU can act as both initiator and responder. The following figure shows an example of the PCIe endpoint sending read or write transactions to a GDDR6 channel.

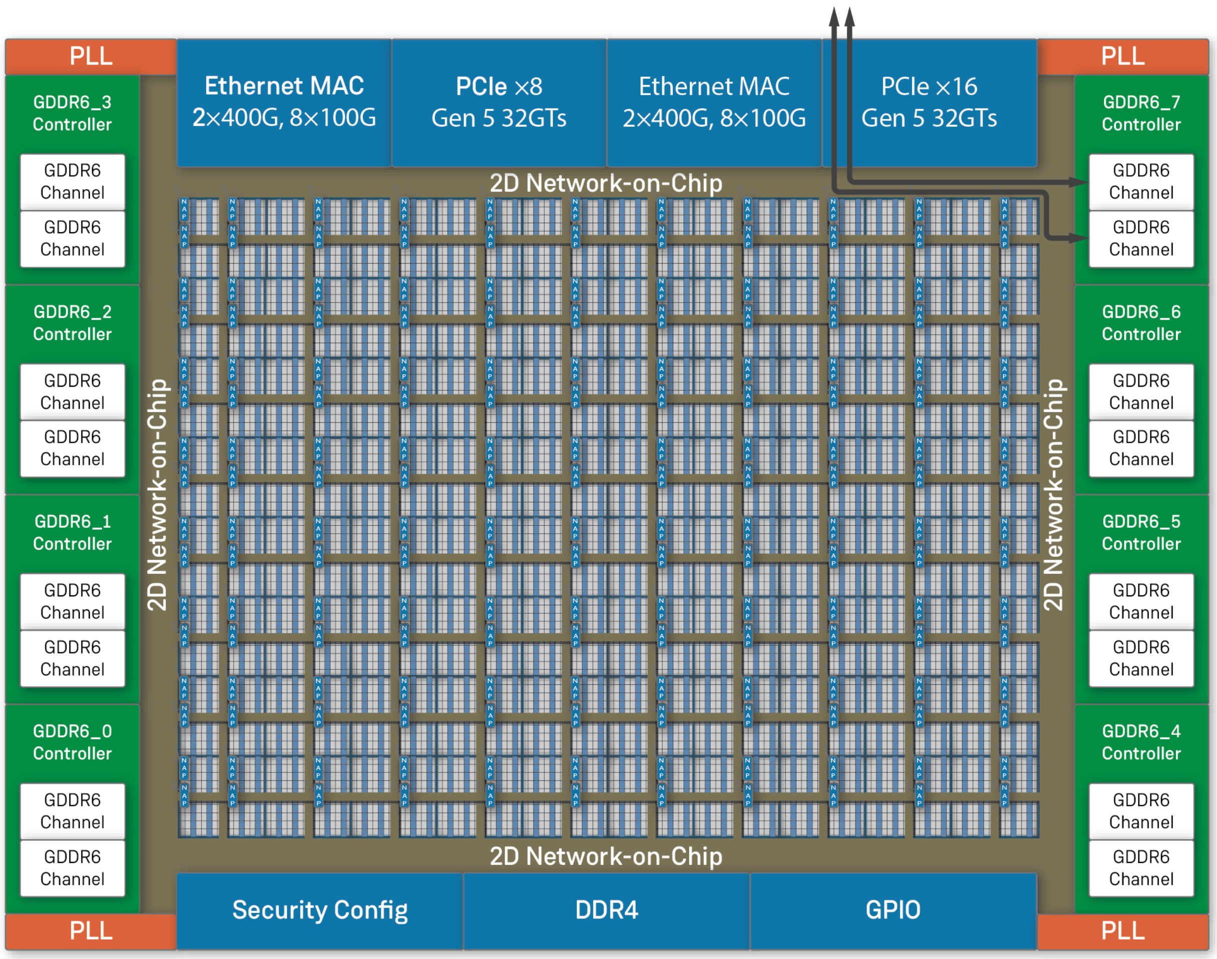


Figure 12: ***PCIe-to-GDDR6 Transactions***

Because these connections do not consume any FPGA fabric resources, there is no impact on routing, area, or timing of the logic in the FPGA. The 2D NoC handles any clock domain crossing internally, as well as flow control and transaction arbitration. However, the traffic flow to expected endpoints must be considered so as to optimize for latency and congestion on the peripheral portion of the 2D NoC. For example, when sending transactions from the PCIe endpoint to several channels of GDDR6, choosing channels on both the east and west side of the FPGA can spread out the traffic rather than sending all traffic down one side, thus reducing congestion on the 2D NoC.

Interface-to-Fabric Connections

Interface subsystems can connect to initiator or responder logic in the FPGA fabric. The appropriate `ACX_NAP_AXI_SLAVE` or `ACX_NAP_AXI_MASTER` macro must be instantiated depending on the type of logic in the fabric. Standard AXI4 protocol is all that is needed to communicate with the NAP through read and write transactions, which in turn connects the user logic through the 2D NoC to the various interface subsystems. Initiator logic in the fabric can send transactions to the PCIe, GDDR6, DDR4, FCU, or CSR space. Additionally, the PCIe and FCU can talk to responder logic in the FPGA fabric. The following figure shows an example of the PCIe endpoint sending transactions to four NAPs with connected responder logic in the FPGA fabric. The transactions go from the PCIe, through the 2D NoC, and into the FPGA fabric via NAPs. Responses travel the same path back to the PCIe endpoint.

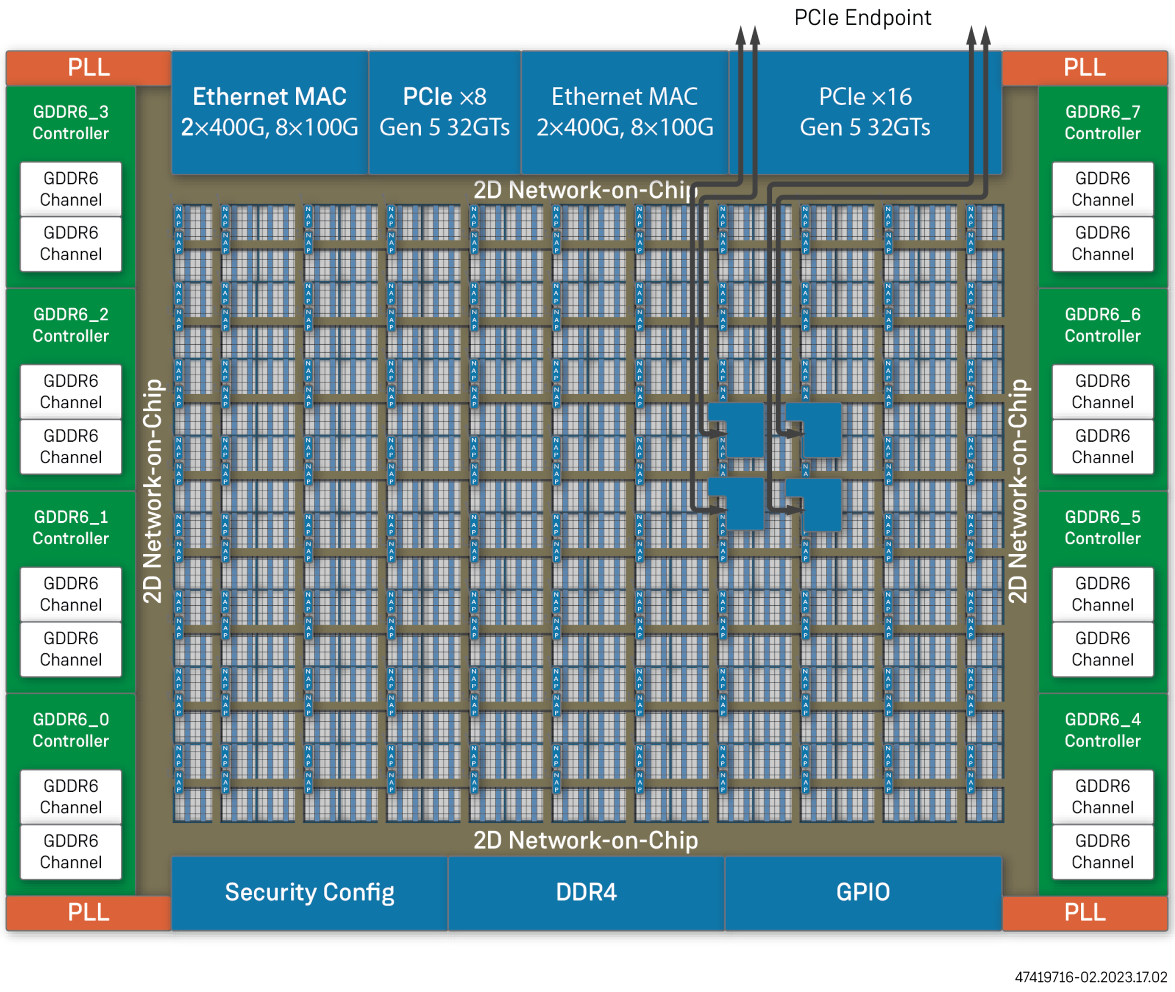


Figure 13: *PCIe-to-FPGA Fabric Transactions*

The 2D NoC handles any clock crossing logic and transaction arbitration internally, eliminating the need to design this logic in the FPGA fabric. Placement of the NAPs does need to be considered in the fabric with respect to the interface subsystems if latency and congestion are concerns. For example, when sending transactions from the PCIe endpoint to a NAP in the fabric, there is more latency to reach a NAP that is physically further away from the PCIe endpoint. Similarly, if a NAP located on the west side of the device sends a transaction to a GDDR6 channel on the east side of the device, the latency is longer than if a NAP on the east side of the device sends the transaction. To help with placement of initiator logic using NAPs that initiate transactions, it is important for a designer to know the direction a transaction takes when traversing the 2D NoC row to the peripheral ring.

The following table lists the direction a transaction takes on the row to arrive at the various interface targets. The direction is based solely on the target destination, and not on the location of the initiating NAP.

Table 1: ***Direction of Transaction Based on Target***

Interface Target	Direction on Row
GDDR6_0	west
GDDR6_1	west
GDDR6_2	west
GDDR6_3	west
GDDR6_4	east
GDDR6_5	east
GDDR6_6	east
GDDR6_7	east
DDR4	east
PCIe ×16	east
PCIe ×8	west
CSR space	west

Additionally, if logic and multiple NAPs are placed along a single column or row in the FPGA, the traffic is concentrated on that one row or column. To reduce congestion, consider expected traffic patterns in the design and choose NAP locations that spread the transaction traffic across several rows or columns when possible.

Ethernet-to-Fabric Connections

The Ethernet subsystem connects directly to specific columns on the 2D NoC and can communicate to FPGA fabric logic connected to vertical NAPs along those specific columns using Ethernet packets. Each Ethernet subsystem has two dedicated columns and can send transactions to NAPs placed only on those two specific columns. The following table lists the specific columns connected to the Ethernet subsystems.

Table 2: ***2D NoC Columns for Ethernet Subsystems***

Ethernet Subsystem location	Ethernet Subsystem 0 (West)	Ethernet Subsystem 1 (East)
2D NoC Column 1	1	4
2D NoC Column 2	2	5

Table Notes

- 2D NoC Columns are numbered 1 at the westernmost column and increment going east.

There are a few modes available, depending on how the Ethernet packets are to be handled in the FPGA fabric. For interfaces using 100GE or slower, the Ethernet sends 256-bit packets down the columns directly to NAPs. For interfaces running 200GE or 400GE, there are two modes to choose from: packet mode or quad-segmented mode.

Packet Mode

The 2D NoC rearranges the 1024-bit data bus into either four (400GE) or two (200GE) narrower data paths, funneling a separate packet to each of the four (or two) NAPs and splitting the full 1024-bit data bus into either four 256-bit (32-byte) or two 256-bit (32-byte) data paths. This solution results in less congestion in the fabric because the user logic can reside in multiple separate engines distributed down the 2D NoC columns rather than a single large engine immediately next to the Ethernet subsystem. This mode also reduces the needed frequency in the FPGA fabric design and makes the design easier because each NAP can have its own individual packet processing engine.

Packet mode can result in larger latency as each packet can take more cycles to transfer. Importantly, packets can arrive out of order, with the 2D NoC sending a sequence number along with each packet. The user logic is responsible for reordering the packets, if necessary, in order to retrieve the original data sequence. The following figure shows how the Ethernet subsystem data bus is rearranged into four separate 256-bit wide data buses for 400G mode. Each packet can take multiple cycles to complete. In 200G mode, the subsystem data bus is rearranged into two separate 256-bit data buses. However, the same principles of a sequence number, and additional latency apply.

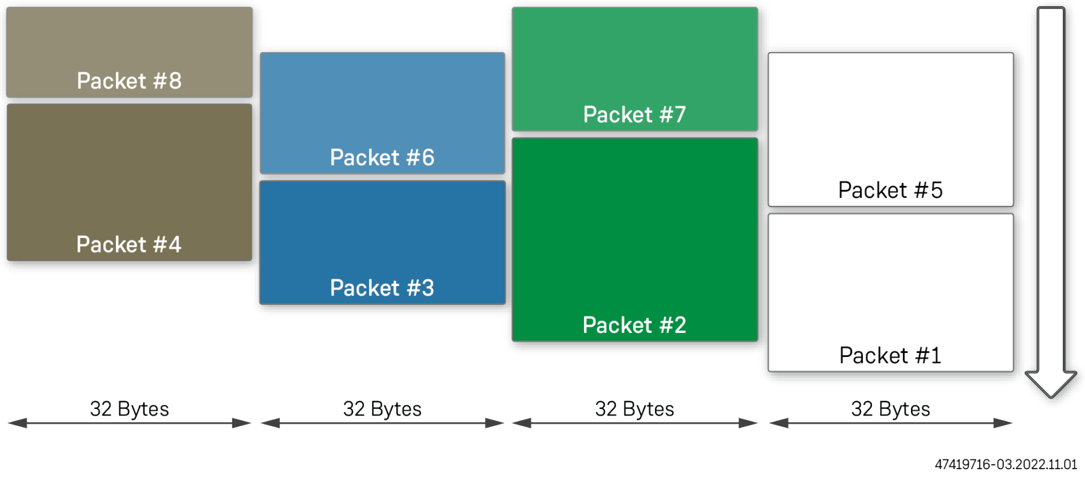


Figure 14: ***Data Bus Rearrangement for Packet Mode***

The four packets shown above are sent to four separate NAPs distributed down the designated 2D NoC columns. Each NAP can communicate with an individual packet processing engine. This arrangement allows each NAP and processing engine to be run at a lower frequency than that required of a single processing engine with the full 1024-bit bus, thus simplifying the system design. For example, a single processing engine for a 400GE solution would require a 1024-bit bus running at approximately 728 MHz, whereas the packet mode for 400GE uses four NAPs and requires four 256-bit buses running at 507 MHz. The 2D NoC automatically handles the load balancing, sending the next available packet to the next free NAP. For more details on Ethernet packet mode, refer to the *Speedster7t Ethernet User Guide* (UG097).

In the following figure, the four NAPs are distributed in different locations along two columns. The specific placement of the NAPs is a design choice. It is equally possible to have all four NAPs located on a single column, or grouped closer together.

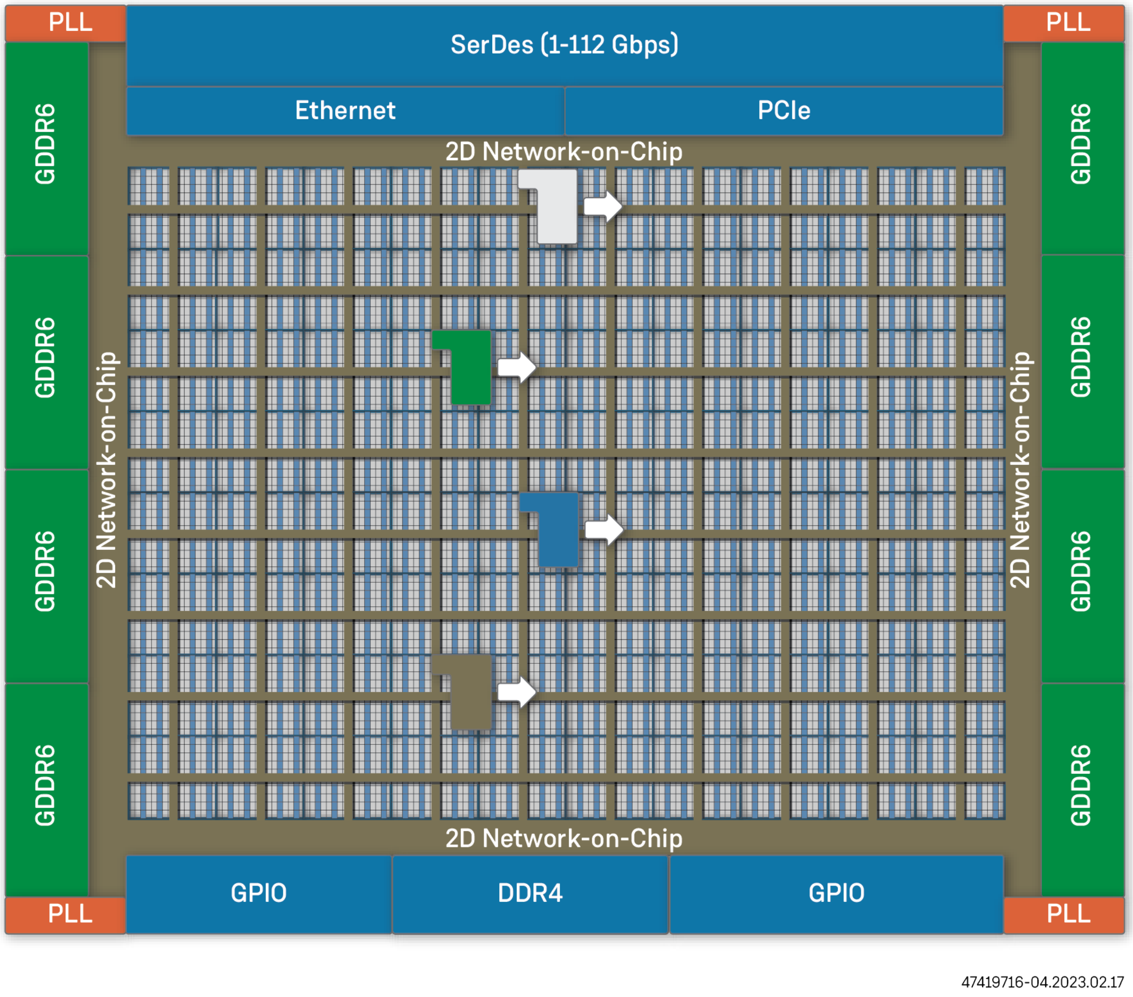


Figure 15: *Ethernet Packet Mode on the 2D NoC*

Quad-Segmented Mode

In quad-segmented mode, the 2D NoC sends a 1024-bit bus that is segmented across four NAPs. This applies to both 200G and 400G modes, quad mode always uses four NAPs. This mode makes the user logic a little more complex as the design logically is one large packet processing engine distributed across the four NAP locations. This mode does guarantee in-order packet arrival, and larger packets arrive with less latency than in packet mode as previously described. Because the bus is segmented, packets can potentially start at any of the four NAPs, and up to two packets can arrive in a single fabric clock cycle.

Similar to the previously described packet mode, the FPGA logic can be spread across the space of four NAPs on the designated columns, rather than having to be placed immediately next to the Ethernet subsystem. This arrangement helps ease congestion, and because the design can be split across four NAPs, the frequency can be reduced similar to the packet mode. For example, a single processing engine for a 400GE solution would require a 1024-bit bus running at approximately 728 MHz, whereas the quad-segmented mode for either 400G or 200G uses four NAPs and requires four 256-bit buses running at 507 MHz. The following figure shows how the packets are arranged and segmented for the quad-segmented mode.

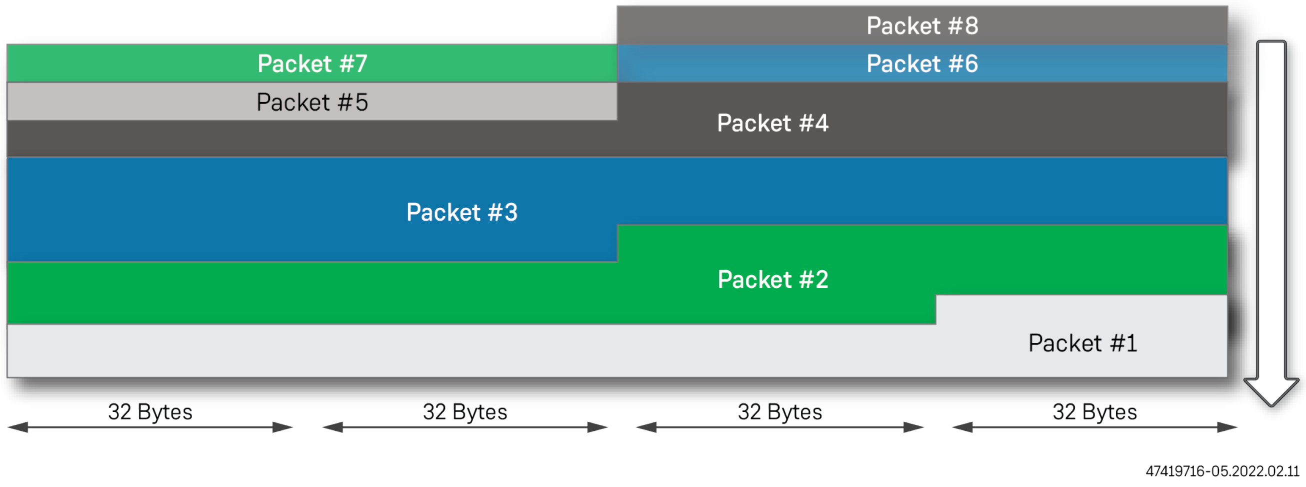


Figure 16: ***Packet Segmentation for Quad-Segmented Mode***

Each packet is distributed across four NAPs located on the designated columns of the 2D NoC. Each 32-byte segment is dedicated to a specific NAP in the group of four. The packet processing engine should be located close to the four NAPs. The following figure shows the four NAPs distributed in two columns, but placed close together. The specific placement of the NAPs is a design choice. It is equally possible to have all four NAPs located on a single column, or grouped farther apart.

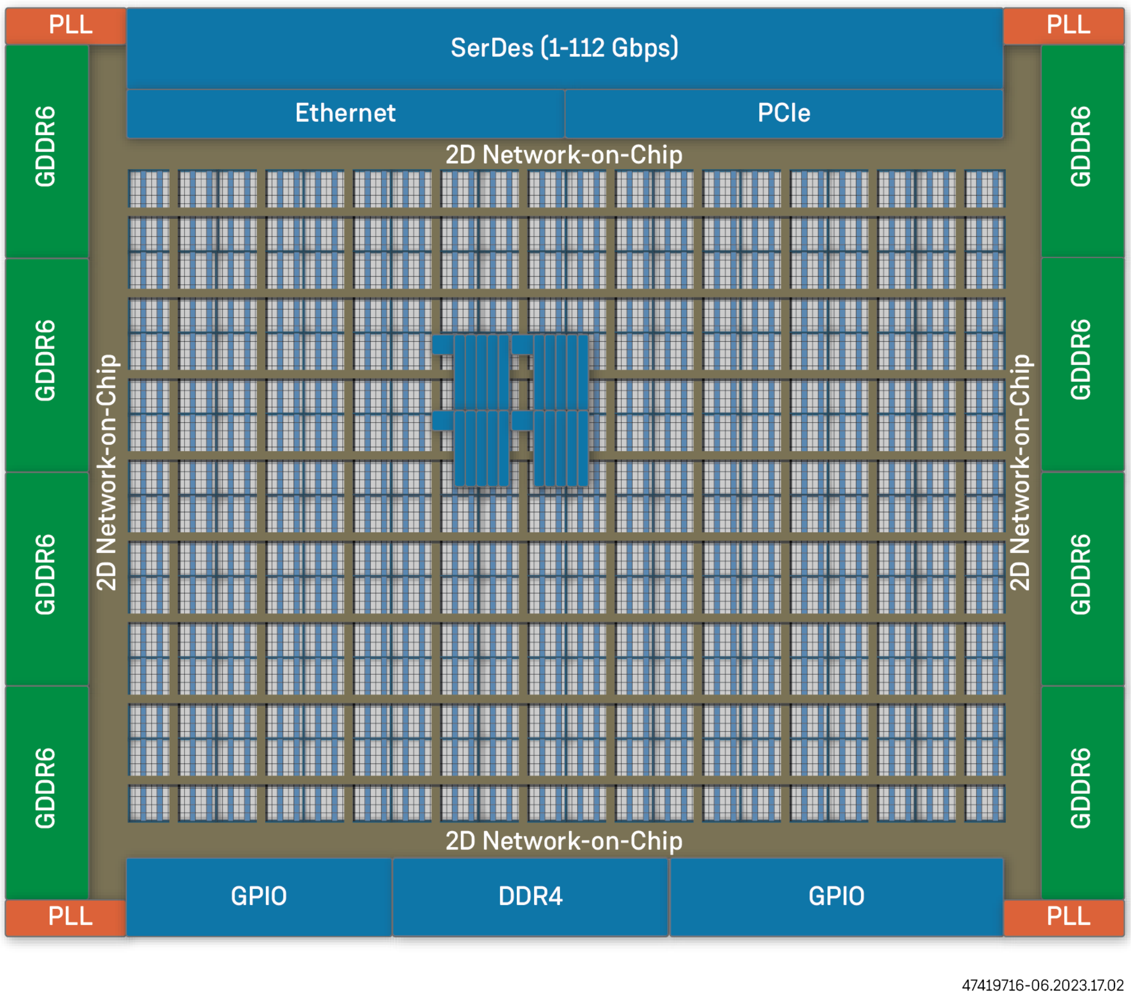


Figure 17: ***Quad-Segmented Mode on the 2D NoC***

For full details on the Ethernet modes and the Ethernet MAC, refer to the *Speedster7t Ethernet User Guide (UG097)*.

Fabric-to-Fabric Connections

While logic in the FPGA fabric can communicate to other logic in the fabric in a traditional manner using conventional routing resources in the FPGA, the 2D NoC enables designs to communicate between points within the FPGA fabric on a wide, high-speed bus without using the fabric routing resources. Depending on where the endpoints are located, and the style of transfer desired, there are two methods to using the 2D NoC for fabric-to-fabric communication:

- AXI transactions
- Data streaming

AXI Transactions

Two points in the FPGA can communicate with each other via the 2D NoC through AXI NAPs. In this case, initiator logic using an AXI responder NAP on a row can send transactions east or west to the peripheral portion of the 2D NoC, and then down a column to an AXI initiator NAP that connects to responder logic in the fabric. As previously mentioned, the AXI NAPs send read and write commands using the AXI4 standard. This method of connecting FPGA points is not optimized for latency, but can easily transfer read and write data. The following figure shows an example of connecting two points via the 2D NoC using AXI mode.

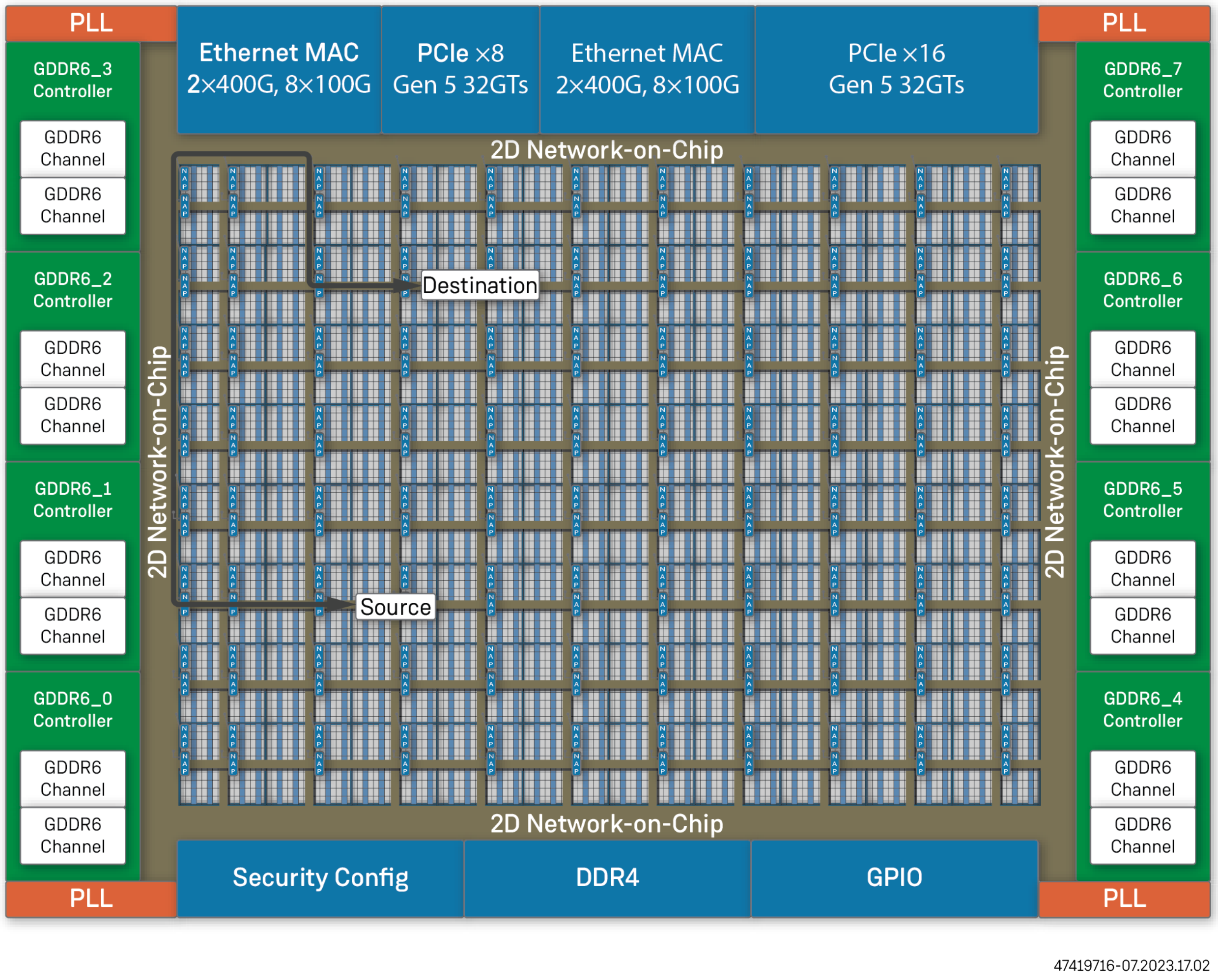


Figure 18: ***AXI Mode Fabric-to-Fabric Transaction***

Data Streaming

Two points within the FPGA fabric along the same row or the same column can communicate via data streaming. These transfers behave like pushing or popping data to or from a FIFO. The transactions use a "ready" signal to indicate that the logic or the NAP can accept data and a "valid" signal to indicate when data is being transmitted. There are also `tx_dest[3:0]` and `rx_src[3:0]` signals that indicate the transfer destination and source, respectively. The location ID is a static number along the row or column. For example, on a row, the NAP number starts at 1 with the westernmost NAP and increments to 10 with the easternmost NAP. Similarly, on a column the NAP number starts at 1 with the southernmost NAP and increments to 8 with the northernmost NAP.

Data streaming provides a simple method to push data across a single row or column without using FPGA routing resources. Each NAP endpoint can both send and receive data, although each individual transfer is one way. The receiving NAP does not send an acknowledgement of receiving the data. Additionally, any number of NAPs on the same row or column can send data streaming transactions between each other; however, transactions are only point to point. There is no broadcast option built into the 2D NoC. If it is necessary to broadcast data down a row or column, the design must take this into account and send the transaction along to the next NAP.

The following figure shows transactions between various points in the 2D NoC. The logic at points 1 and 2 have each instantiated a horizontal NAP. The NAPs can both send and receive data, as indicated by the arrows in the figure, but each individual data stream transaction is unidirectional. For example, the NAP at location 1 can send a data stream transaction to the NAP at location 2, and the NAP at location 2 can send a separate data stream transaction to the NAP at location 1. Similarly, the logic at points 3 and 4 both instantiate a vertical NAP and can send data streams between each other.

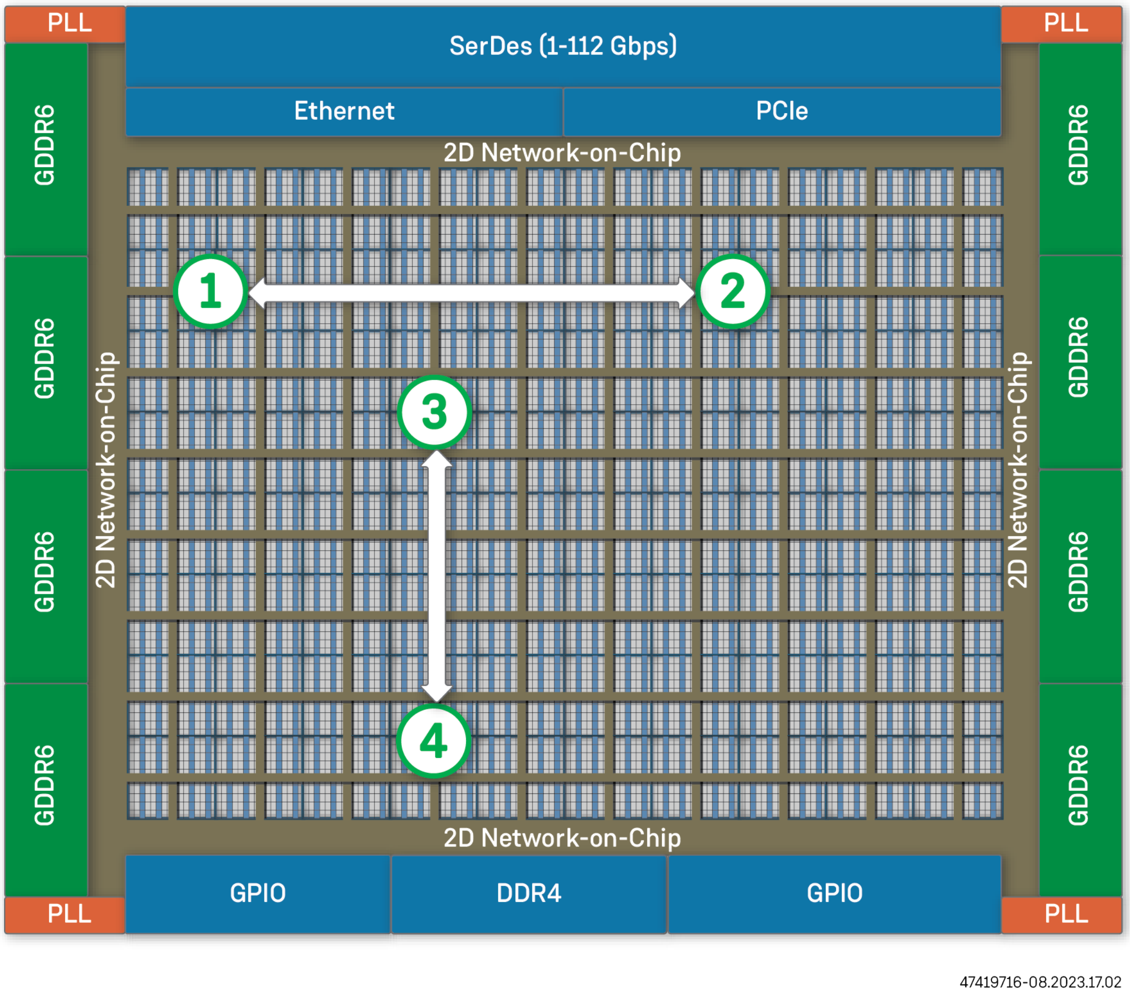


Figure 19: *Data Streaming*

Any clock domain crossing logic is automatically handled in the 2D NoC. The 2D NoC also handles transaction arbitration internally and can interleave the data streaming with AXI transactions. Keep in mind the full design when using multiple NAPs on a row or column such that traffic congestion is considered. Since data streaming requires a single column or row for the NAPs communicating with each other, it is necessary to be aware of traffic to AXI NAPs on the same row or column. AXI transactions and data streaming can be interleaved and add to latency.

Chapter - 6: Speedster7t 2D NoC Address Mapping

Global Address Map

The 2D NoC has a global address map used to address all of the endpoints in the FPGA. The address map uses a 42-bit address space and includes regions that can be remapped with an address translation table for each NAP. Refer to the section on Address Translation (see page 40). The following figure shows the 2D NoC address space and how each portion of the 42-bit address space is distributed.

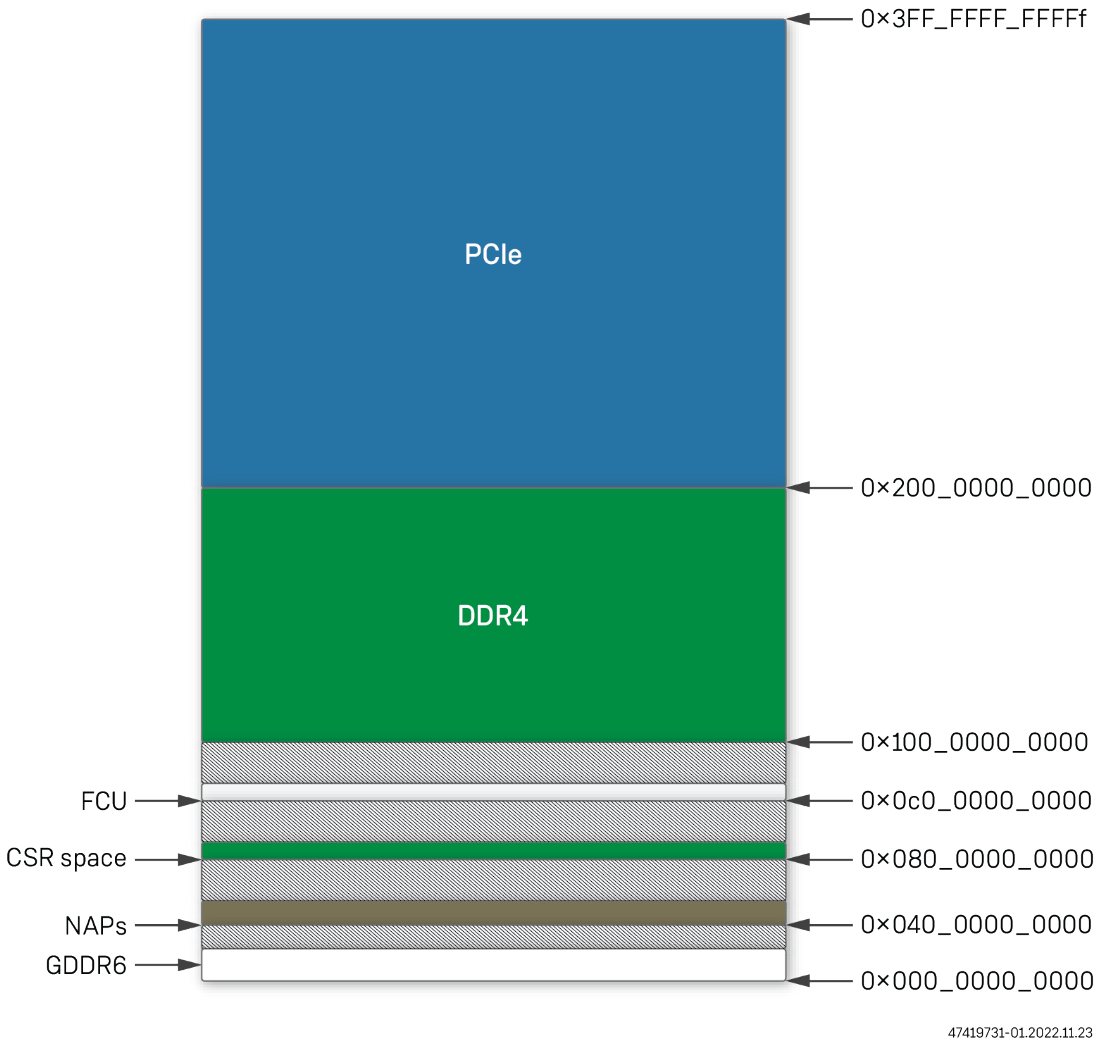


Figure 20: *2D NoC Address Space*

Each of the endpoints on the 2D NoC has its own address space. The following global address space table describes the details of each of the endpoints available on the 2D NoC.

Table 3: ***2D NoC Global Address Map***

Address Bit	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	…	0
Destination																						
PCIe	1	ID	Memory Address																			
DDR4	0	1	Memory Address																			
GDDR6	0	0	0	0	0	Ctrl ID				Memory Address												
NAP	0	0	0	1	0	0	0	NAP Column				NAP Row			Memory Address							
CSR Space	0	0	1	0	0	0	0	0	Target ID						IP ID				Memory Address			
FCU	0	0	1	1	0	0	0	0	0	0	0	0	FCU Address									

The 2D NoC uses the most significant bits of the address to identify the destination space of a transaction. A description of each address space follows.

PCIe

- Addr[41] = `1'b1`
- Addr[40] = **ID** – Selects between the two PCIe IP cores:
    - 1'b0 – PCIE_1 IP, PCIex16 port
    - 1'b1 – PCIE_0 IP, PCIex8 port
- Addr[39:0] = **Memory Address** – Address passed to the PCIe core

DDR4

- Addr[41:40] = `2'b01`
- Addr[39:0] = **Memory Address** – Address passed directly to the DDR4 controller

GDDR6

- Addr[41:37] = `5'b00000`
- Addr[36:33] = **Ctrl ID** – Selects which of the sixteen GDDR6 channels the transaction is destined for:
    - Addr[36:34] – Selects the controller
    - Addr[33] – Selects between the two channels on each controller
- Addr[32:0] = **Memory Address** – The memory address for the specific controller and channel

GDDR6 Ctrl ID Mappings

The mappings from **Ctrl ID** to individual GDDR6 memory controllers address space vary according to the particular device. As an example, the mappings for the Speedster7t AC7t1500 FPGA are shown in the following table.

Table 4: ***Speedster7t AC7t1500 FPGA GDDR Memory Mapping***

Ctrl ID				Controller	Channel
36	35	34	33		
0	0	0	0	GDDR6 2	0
0	0	0	1	GDDR6 2	1
0	0	1	0	GDDR6 6	1
0	0	1	1	GDDR6 6	0
0	1	0	0	GDDR6 1	0
0	1	0	1	GDDR6 1	1
0	1	1	0	GDDR6 5	1
0	1	1	1	GDDR6 5	0
1	0	0	0	GDDR6 3	0
1	0	0	1	GDDR6 3	1
1	0	1	0	GDDR6 7	1
1	0	1	1	GDDR6 7	0
1	1	0	0	GDDR6 0	0
1	1	0	1	GDDR6 0	1
1	1	1	0	GDDR6 4	1
1	1	1	1	GDDR6 4	0

Note

- The address mappings apply only to the memory address space. For configuration space (CSR) address mappings, please see the appropriate CSR mapping table.
- The channel selection (LSB of **Ctrl ID**) is reversed for channels on the east side of the device. This is shown in the table, above.

NAP

- Addr[41:35] = `7'b0001000` – Accesses any NAP endpoint in the device.
- Addr[34:31] = **NAP Column**– Valid values for this field are 0 to 9, west to east. Columns on the 2D NoC are numbered 1 to 10, west to east. In order to save bits in the address, the number for a column "N" becomes "N-1" for this value (column 3 uses the value of 2).
- Addr[30:28] = **NAP Row**– Valid values for this field are 0 to 7, south to north. Rows on the 2D NoC are numbered 1 to 8, south to north. In order to save bits in the address, the number for a row "N" becomes "N-1" for this value (row 5 uses the value of 4).
- Addr[27:0] = **Memory Address** – Passed to the FPGA fabric logic.

Note

The row and column fields in the address for AXI transactions start numbering from 0, whereas placement constraints in ACE use the actual row and column numbers, starting from 1.

CSR Space

- Addr[41:34] = `8'b00100000` – Accesses all of the control and status registers in the FPGA.
- Addr[33:28] = **Target ID** – Selects the space (PCIe, DDR4, GDDR6, etc.) where the control and status register(s) reside.
- Addr[27:24] = **IP ID** – Indicates a specific target space internal to the IP. This ID is unique for each IP and is described in the associated user guide.
- Addr[23:0] = **Memory Address** – Byte address for the specific space in the IP.
  See the table, Control and Status Register Map (see page 40).

FCU

- Addr[41:30] = `12'b001100000000`
- Addr[29:0] = **FCU Address** - This address is passed directly to the FCU block.

Control and Status Register Space

The control and status register (CSR) space can receive read or write transactions from an initiator on the 2D NoC, which initiates an AXI transaction to the particular address of a register in the CSR space, allowing the initiator to write to a control register or read a status register in one of the GDDR6 controllers or DDR4 controller, for example. The CSR space uses a 34-bit address, with the most significant bits indicating the target IP space. The target IP spaces address map follows.

For more information on each individual register space, consult the associated user guide.

Table 5: *Control and Status Register Map*

Target ID							Description
CSR Space	**33**	**32**	**31**	**30**	**29**	**28**	
GDDR6_0	0	0	0	0	0	0	GDDR6 0 control and status registers.
GDDR6_1	0	0	0	0	0	1	GDDR6 1 control and status registers.
GDDR6_2	0	0	0	0	1	0	GDDR6 2 control and status registers.
GDDR6_3	0	0	0	0	1	1	GDDR6 3 control and status registers.
DDR4	0	0	1	0	0	1	DDR4 control and status register space.
GPIO south	0	0	1	0	1	1	General-purpose I/O on south side.
Temp Sensor	0	0	1	1	0	0	Temperature sensor.
GDDR6_4	0	1	0	0	0	0	GDDR6 4 control and status registers.
GDDR6_5	0	1	0	0	0	1	GDDR6 5 control and status registers.
GDDR6_6	0	1	0	0	1	0	GDDR6 6 control and status registers.
GDDR6_7	0	1	0	0	1	1	GDDR6 7 control and status registers.
PCIe x16	0	1	1	0	0	1	PCIe ×16 control and status registers.
PCIe x8	0	1	1	0	1	0	PCIe ×8 control and status registers.
Ethernet 0	0	1	1	0	1	1	Ethernet 0 control and status registers.
Ethernet 1	0	1	1	1	0	0	Ethernet 1 control and status registers.
GPIO north	0	1	1	1	0	1	General-purpose I/O on north side.

Address Translation

Each NoC access point (NAP) has its own private address translation table that is configured through the bitstream. The address translation table allows the NAP to remap various endpoints. For example, the NAP can remap the address of each GDDR6 controller, along with pages within each controller memory space. Similarly, each NAP can remap pages within the DDR4 memory space, and can even remap other NAP endpoints.

Address translation can be useful for a number of reasons. For example, if it is desired to have several engines accessing GDDR6 and to reuse the same RTL for each engine, this can be easily accomplished. A module can be written to access GDDR6 0, but then the translation tables can be configured to point to the particular GDDR6 that is closest to each instance of the engine.

Caution

Configuration of the address translation table in the NAP is not currently available in the ACE tool suite.

Additionally, access to certain endpoints can be prevented, for example, to add security such that two engines can be prevented from accessing the same memory. The I/O Designer Toolkit 2D NoC configuration GUI in ACE provides a simple way to disable access per 2D NoC row to various endpoints such as GDDR6, DDR4, PCIe 0, PCIe 1, FCU, CSR space, and the NAPs.

The following tables list the bits available for address translation within the specific address spaces. Bits that are available for address translation are highlighted in yellow.

DDR4

- Bits[32:26] of the DDR4 memory address can be used in address translation allowing pages in the memory to be remapped.
  Table 6: *DDR4 Address Translation*

Address Bit	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	…	0
DDR4	0	1	Memory Address							🟨										

GDDR6

- Addr[36:33] = **Ctrl ID** – All bits can be used in address translation allowing remapping to determine which GDDR6 controller receives a transaction.
- Bits[28:26] – Can be used in address translation allowing pages in the memory to be remapped.
  Table 7: *GDDR6 Address Translation*

Address Bit	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	…	0
GDDR6	0	0	0	0	0	Ctrl ID				Mem Address				🟨						

NAP

- Bits[34:31] = **NAP Column** – Can be used in address translation.
- Bits[30:28] = **NAP Row** – Can be used in address translation.
  Both fields can be used in address translation allowing the location of the NAP transaction to be remapped.
  Table 8: *NAP Address Translation*

Address Bit	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	…	0
NAP	0	0	0	1	0	0	0	NAP Column				NAP Row			Memory Address					

Chapter - 7: Speedster7t 2D NoC Performance

The 2D NoC is optimized for high bandwidth and supports a cross-sectional bidirectional bandwidth of 20 Tbps. The 2D NoC provides a 256-bit wide primary datapath that runs at 2 GHz, thus delivering 512 Gbps of bidirectional bandwidth in all directions. Because it includes clock-crossing logic internally, the main buses of the 2D NoC can run at high speeds, while the FPGA fabric and IP interfaces can run at lower frequencies as needed.

Latency and Performance

Latency

In order to increase user design flexibility, the 2D NoC includes clock domain crossing logic to transmit data from the logic operating at the FPGA fabric speed to the 2 GHz data path of the 2D NoC. Each NAP has a small asynchronous FIFO adding a few fabric clock cycles in each direction, which adds a small amount of latency to transactions. Additionally, there is some latency added to traverse a 2D NoC row or column. In the east-west direction there is latency of 2 × 2 GHz, or 1 ns per NAP along the row. In the north-south direction, there is latency of 7 × 2 GHz, or 3.5 ns per NAP along the column.

Note

The 2D NoC may be operated at frequencies other than the default of 2GHz. In these instances, the latency values increase according to the clock period of the selected 2D NoC frequency. See the section, Power (see page 45).

AXI Burst Transactions

One method to increase performance is to make use of burst transactions. For a single AXI transaction, Speedster7t AC7t1500/1550 FPGAs support a maximum burst length of up to 16 beats, equivalent to 4Kb or 512B of data. While this is smaller than the full 256 beats supported by the AXI standard, this does provide for more efficient use of each AXI transaction, thus increasing performance when AXI bursts are used.

There are two exceptions to the limit above:

- When reading from external memory, DDR4 or GDDR6, it is possible to request a burst greater than 16 beats. However, the east/west buffers in an AXI Responder NAP have 16 entries. These are written at the 2D NoC frequency, (up to 2.027GHz), however they are read out at the lower user design frequency. This causes the NAP to back-pressure the 2D NoC row, causing any other traffic on that row to be stalled. Therefore, read bursts of greater than 16 beats should only be considered for a NAP that is on its own row, with no other NAPs present on the same row. For bursts of 16 beats or less, the NAP FIFO can absorb all the traffic at full speed with no back pressure to the rest of the 2D NoC row.
- It is possible for an interface subsystem which initiates traffic, such as the PCIe core, to generate bursts greater than 16 beats. Therefore, a user design should ensure that any data buffering from such a source should be able to support longer burst sizes. Particularly for data from the PCIe core as that is sent from north to south, and the shallow NAP north to south buffers (see below), can only buffer 4 entries.

Note

- For writing to external memory, using the AXI Responder NAP, the limit is 16-beats. This is supported by the DSM simulation which indicates an error if a burst of greater than 16 beats is attempted.
- For detailed information on the AXI initiator and responder NAPs, please refer to the appropriate chapters in the *Speedster7t Soft IP User Guide* (UG103).

NAP Buffering

Each NAP contains two asynchronous FIFO buffers, one in each 2D NoC direction. These support the buffering of data as it is transferred between the user design clock domain, and the 2D NoC clock domain. For horizontal and AXI responder NAPs, the east to west and west to east buffers are 16 entries deep. For vertical, AXI initiator and Ethernet NAPs, the south to north buffers are 16 entries deep, however the north to south buffers are only 4 entries deep.

As the ratio of the user design and 2D NoC clock domains can span a wide variance, it is possible that for certain combinations of the two clocks, traffic throughput can be impacted, particularly by the shallow north to south buffers, as clock domain crossing of the data is performed.

Note

The buffers in each NAP are designed to absorb traffic from the 2D NoC, and to support CDC crossing from the 2D NoC clock and the user design clock. They should not be used, by design, to store incoming transactions or data. In this scenario, it is possible for the buffers to fill, and then back-pressure their respective row or column.

User designs should ensure that they are able to service any read or write request from a NAP in a timely manner. For example, a PCIe DMA transaction may issue multiple read requests. The user design NAP receiving these read requests should accept them from the NAP as they arrive, even if the user logic is not yet able to service the read request. The read requests can then be processed by the user logic and responses returned in due course.

Outstanding Transactions Table

Each NAP contains an outstanding transactions table (OTT) which is used to ensure in-order responses. According to the AXI specification, if two transactions are issued with the same transaction ID, the responses must be returned in the same order. If two transactions with the same ID are issued to the same direction (east or west), downstream logic ensures that they are returned in order. But if the transactions are issued in different directions, the responses might be returned out of order. The outstanding transaction table prevents this scenario by allowing transactions to be issued in one direction only if there are no outstanding transactions in the other direction. To accomplish this, a counter is maintained of the number of outstanding transactions for every transaction ID. The counter has a maximum value of 16, meaning that no more than 16 transactions can be outstanding for each transaction ID.

Since this can cause a performance issue, the OTT can be disabled, allowing more transactions to be outstanding at any given time, but also allowing out-of-order responses for transactions in different directions. If a NAP is only ever sending transactions in one direction, or is using different IDs for each transaction, then there is no risk to disabling the OTT. While this can give greater throughput, this parameter should not be disabled unless the NAP is only sending in a single direction, or is using unique IDs, by design.

Flits and AXI Protocol

All transactions on the 2D NoC are carried as small data packets called flits. These are carried serially over the 2D NoC, traversing rows, columns and exiting to the peripheral 2D NoC surrounding the core programmable fabric.

The AXI protocol defines five asynchronous channels:

1. Write request
2. Write data
3. Write response
4. Read request
5. Read data

According to the AXI specification, these channels should be considered independent, and can issue requests and responses independently of each other.

When using an AXI NAP, these channels are presented as the five independent channels. However, within the NAP, each channel transaction is converted to or from flits, which are sent serially and in order through the 2D NoC. This creates a dependency between the AXI channels. In the case that an AXI NAP blocks one of the AXI channels by not responding, this blocks the other channels as the flit awaiting a response is held, causing "head of line" blocking.

For example, if an AXI Responder NAP is both reading and writing, then having issued a write request, write data and read request, it should ensure that `rready` and `bready` are both asserted in order to accept the write response and read data. If `bready` is not asserted, and the write response is received first at the NAP, that flit is held waiting `bready`. This blocks any other flits, such as the those with the read data. In this way it would appear that the read never returned any data and, at the same time, the read data could be overflowing the NAP buffer on to the 2D NoC row, blocking any other NAPs. Equally, if `rready` is not asserted, and the read data is received first, it is held, and blocks the write response flit.

Note

- User designs should ensure that AXI NAPs assert their respective "channel ready" signals whenever they issue a transaction (AXI Responder NAP) or are designed to accept transactions (AXI Initiator NAP).
- For detailed information on the AXI initiator and responder NAPs, please refer to the appropriate chapters in the *Speedster7t Soft IP User Guide* (UG103).

AXI Responses

All AXI transactions must receive a response. If a read or write request is received at an AXI NAP, then it must issue a response. If the NAP does not respond, then, as described above, the respective flit is held and the NAP is blocked. Any future transactions to that NAP fail, possibly causing the whole system to fail. If the design cannot respond due to error, it should still respond, setting one of the error codes supported by `rresp` or `bresp`.

Power

With the 2D NoC, there are different aspects of power consumption that should be understood. For each NoC access point (NAP) there are two portions that consume power:

1. The high-frequency portion connected to the row or column of the 2D NoC that operates at 2 GHz. This portion is always active while the 2D NoC is in use.
2. The lower-frequency portion of the NAP that operates at the fabric logic frequency of the user design. This portion is only used if the NAP is instantiated in the design. In this case, the NAP portion operating at the fabric logic frequency does not contribute to dynamic power if it is unused.

The 2D NoC along with the NAPs *cannot* be powered down, but it can be configured to run with a lower frequency, which results in a lower power consumption. A lower frequency clock can be configured down to 200 MHz. In this case, the 2D NoC is still functional, but operates at the selected lower frequency rather than the default of 2 GHz. When set to the minimum frequency of 200MHz, the 2D NoC consumes the lowest power possible, however at these low frequencies it is not recommended to drive high bandwidth transactions through the 2D NoC to any interface subsystems.

Chapter - 8: Speedster7t 2D NoC Simulation Support

With the introduction of a two-dimensional network on chip (2D NoC) interacting with logic in the FPGA fabric, it is important to have methods for simulating the user design to understand how the design interacts with the 2D NoC. Achronix provides three levels of simulation models to support different phases in the design process:

1. A bus functional model (BFM) of the NAP for simple functional simulations.
2. A cycle-accurate model of the full rows, columns and peripheral ring of the 2D NoC to simulate latency and congestion between NAPs, with BFMs of the interface subsystems.
3. Ability to extended the cycle-accurate model of the 2D NoC with cycle-accurate RTL models of individual interface subsystems.

NAP Bus Functional Model

The first phase of simulation with a design is to functionally communicate with a NoC access point (NAP) in the fabric. The Achronix library includes simple BFMs in each instance of a NAP macro. Each NAP includes simple tasks which can be called to simulate sending or receiving a transaction. The tasks depend on the type of NAP macro used and the direction of the transaction. The testbench calls these tasks in the BFMs by using bind statements.

The following example shows how to bind to the BFM tasks in a NAP and use a testbench to respond to requests from the FPGA fabric logic initiating transactions. These examples are only snippets of code. For a more detailed example of how to use the NAP BFMs in a simulation, refer to the *Speedster7t DDR4 Reference Design Guide* (RD018), in STANDALONE simulation mode.

NAP Task BFM Binding Example

```
// Testbench has to connect to NAP responder via tasks
// When binding, the module is inside the target module, so gets
// parameters and signal names from that module - not this module

 bind dut.i_axi_responder_nap_wrapper.x_NAP_AXI_SLAVE
 tb_noc
   inst_noc (
        // Inputs
        .i_clk                 (clk), // bound to signal in AXI_NAP_SLAVE
        .i_reset_n             (rstn) // bound to signal in AXI_NAP_SLAVE
    );

  // DUT
  my_design_with_nap
    dut (
        // Inputs
        .i_clk                 (clk),
        .i_reset_n             (reset_n)
    );

//----------------------------

// The DUT that instantiates the NAP

 module my_design_with_nap (
  input i_clk,
  input i_reset_n
);

ACX_NAP_AXI_SLAVE i_axi_responder_nap_wrapper (
        .clk             (i_clk),
        .rstn            (i_reset_n),

//----------------------------

// The ACX_NAP_AXI_SLAVE instantiates the NAP_AXI_SLAVE which has the BFM tasks
NAP_AXI_SLAVE x_NAP_AXI_SLAVE (
        .clk            (i_clk),
        .rstn           (i_reset_n),


//----------------------------

// the testbench that is bound to the NAP calls the tasks

module tb_noc
(
    // Inputs
    input  wire             i_clk,
    input  wire             i_reset_n    // Negative synchronous reset
);

    // Support read requests by calling tasks in NAP
    initial
    begin
        #1000       // Allow NAP simulations models to reset first
        while(1)
        begin
            // Blocking call.  Task will only complete when request made
            get_AR(t_arid, t_araddr, t_arlen, t_arsize, t_arburst, t_arlock, t_arqos);
            begin
                // Read request logged
                for( i=t_arlen; i>0; i=i-1 )
                begin
                    issue_R(t_arid,mem_array_out,2'b00,1'b0);
                    t_araddr = t_araddr + 42'h01;
                    @(posedge i_clk);
                end
                issue_R(t_arid,mem_array_out,2'b00,1'b1);
                @(posedge i_clk);
            end
            @(posedge i_clk);
        end
    end
```

ACX_NAP_AXI_SLAVE Responder Macro

Initiator logic in the FPGA fabric communicates with a NAP AXI responder macro. In this case, the transactions initiate in the FPGA user logic and the NAP responds. The user testbench can call the tasks in the BFM by using bind statements. The following tasks are available to functionally model AXI transactions to initiator logic in the FPGA fabric.

Table 9: ***NAP AXI Responder Tasks***

Task Name	Description
`get_AR`	Waits for a valid read request and returns the relevant AXI fields to accept the transaction.
`get_AW`	Waits for a valid write request and returns the relevant AXI fields to accept the transaction.
`get_W`	Waits for valid write data and returns relevant AXI fields to accept the data.
`issue_R`	Issues valid read data and waits until the read data is accepted.
`issue_B`	Issues valid write response/acknowledge and waits until the response is accepted.

ACX_NAP_AXI_MASTER Initiator Macro

Responder logic in the FPGA fabric communicates with a NAP AXI initiator macro. In this case, the transactions initiate from the NAP, and the FPGA user logic responds. The user testbench can call these tasks in the BFM by using bind statements. The following tasks are available to functionally model AXI transactions to responder logic in the FPGA fabric.

Table 10: ***NAP AXI Initiator Tasks***

Task Name	Description
`issue_AR`	Issue a valid read request and wait until the request is accepted.
`issue_AW`	Issue a valid write request and wait until the write request is accepted.
`issue_W`	Send valid write data and wait until the write data is accepted.
`get_R`	Receive read data when valid read data is available.
`get_B`	Receive write response/acknowledge when valid.

ACX_NAP_HORIZONTAL Macro

If user logic sends or receives raw data streams (or flit transfers) along a single row, there must be two horizontal NAP macros which communicate with each other. Each horizontal NAP implements a simple BFM to model the functionality of the data transfer. The user testbench can call the tasks in the BFM by using bind statements. The following tasks are available to functionally model the flit transfers.

Table 11: ***NAP Horizontal Tasks***

Task Name	Description
`issue_rx`	Issue a flit transfer and wait for it to be accepted.
`get_tx`	Receive a flit transfer request, assert ready when ready and wait for a "valid" signal.

Note

Transactions between `ACX_NAP_HORIZONTAL` instances can only be simulated when using the full cycle-accurate model of the 2D NoC, as it requires traffic to pass between multiple NAPs.

ACX_NAP_VERTICAL Macro

If user logic sends or receives raw data streams (or flit transfers) along a single column, there must be two vertical NAP macros that communicate with each other. Each vertical NAP implements a simple BFM to model the functionality of the data transfer. The user testbench can call the tasks in the BFM by using bind statements. The following tasks are available to functionally model the flit transfers.

Table 12: ***NAP Vertical Tasks***

Task Name	Description
`issue_rx`	Issue a flit transfer and wait for it to be accepted.
`get_tx`	Receive a flit transfer request, assert ready when ready and wait for a "valid" signal.

Note

Transactions between `ACX_NAP_VERTICAL` instances can only be simulated when using the full cycle-accurate model of the 2D NoC, as it requires traffic to pass between multiple NAPs.

ACX_NAP_ETHERNET Macro

If user logic sends or receives Ethernet streams along a column, an Ethernet NAP macro must be used to communicate with the Ethernet subsystem. Each Ethernet NAP implements a simple BFM to model the functionality of the data transfer. The user testbench can call the tasks in the BFM by using bind statements. The following tasks are available to functionally model the transfers.

Table 13: ***NAP Ethernet Tasks***

Task Name	Description
`issue_rx`	Issue a transfer and wait for it to be accepted.
`get_tx`	Receive a transfer request, assert ready when ready and wait for a "valid" signal.

Note

Transactions using `ACX_NAP_ETHERNET` can only be simulated when using the full cycle-accurate model of the 2D NoC, as it requires traffic to pass between the `ACX_NAP_ETHERNET` and the Ethernet interface subsystem.

Simulating 2D NoC with DSM

ACE includes a full chip simulation model of each of the Speedster7t FPGAs, known as the Device Simulation Model (DSM). The DSM provides a full cycle-accurate model of the 2D NoC along with BFMs of the interface subsystems. The combination of the BFMs for the interface subsystems and the cycle-accurate 2D NoC creates a balance between faster compile and simulation times, while also accurately modeling latency and traffic congestion on the 2D NoC. If multiple NAPs are used in a design, simulating at this level is a critical step to understanding if there are bottlenecks in the 2D NoC usage and allows determining if the efficiency can improve by placing the NAPs on different rows or columns.

When using the DSM, the Achronix-defined text macro is then used to attach the NAP in the user design to the specific NAP location in the 2D NoC hierarchy. An example follows of how to instantiate the Speedster7t AC7t1500 FPGA, and to connect four types of NAPs in a design to specific NAP locations in the device.

Full 2D NoC Simulation Binding NAPs

```
//Instantiate Speedster7t1500
ac7t1500 ac7t1500( );

// horizontal NAP at col=1, row=3
`ACX_BIND_NAP_HORIZONTAL(DUT.i_nap_row_3.i_nap_horizontal,1,3);

// vertical NAP at col=3, row=1
`ACX_BIND_NAP_VERTICAL(DUT.i_nap_col_3.i_nap_vertical,3,1);

// AXI responder NAP at col=1, row=1 (south-west corner)
`ACX_BIND_NAP_AXI_SLAVE(DUT.i_axi_responder_wrapper_in.i_axi_responder,1,1);

// AXI initiator NAP at col=9, row=8 (north-east corner)
`ACX_BIND_NAP_AXI_MASTER(DUT.i_axi_bram_rsp.i_axi_initiator_nap.i_axi_initiator,9,8);
```

For an example of using the DSM, refer to the *Speedster7t 2D NoC Reference Design Guide* (RD022). This reference design provides example source code for instantiating and using NAPs, and includes a full testbench along with constraint and project files for implementation.

Cycle-Accurate Simulations of Interface Subsystems

A final step for simulation is to use the DSM, while switching the target interface subsystems from a BFM to an RTL model. This step allows accurately modeling traffic in the 2D NoC and to/from any interface subsystems such as PCIe, GDDR6, DDR4, or Ethernet. It provides a cycle-accurate method to model delays, latency, and traffic congestion in the entire system. However, this accuracy comes at the cost of increased compile and simulation time. As each interface subsystem can be run in full RTL or BFM mode, it is possible, for example, to simulate the DDR4 interface subsystem as an RTL model, while choosing to run the remaining interface subsystems in BFM mode.

Caution

Refer to the specific interface subsystem user guide for details on running the particular subsystem using full RTL simulation.

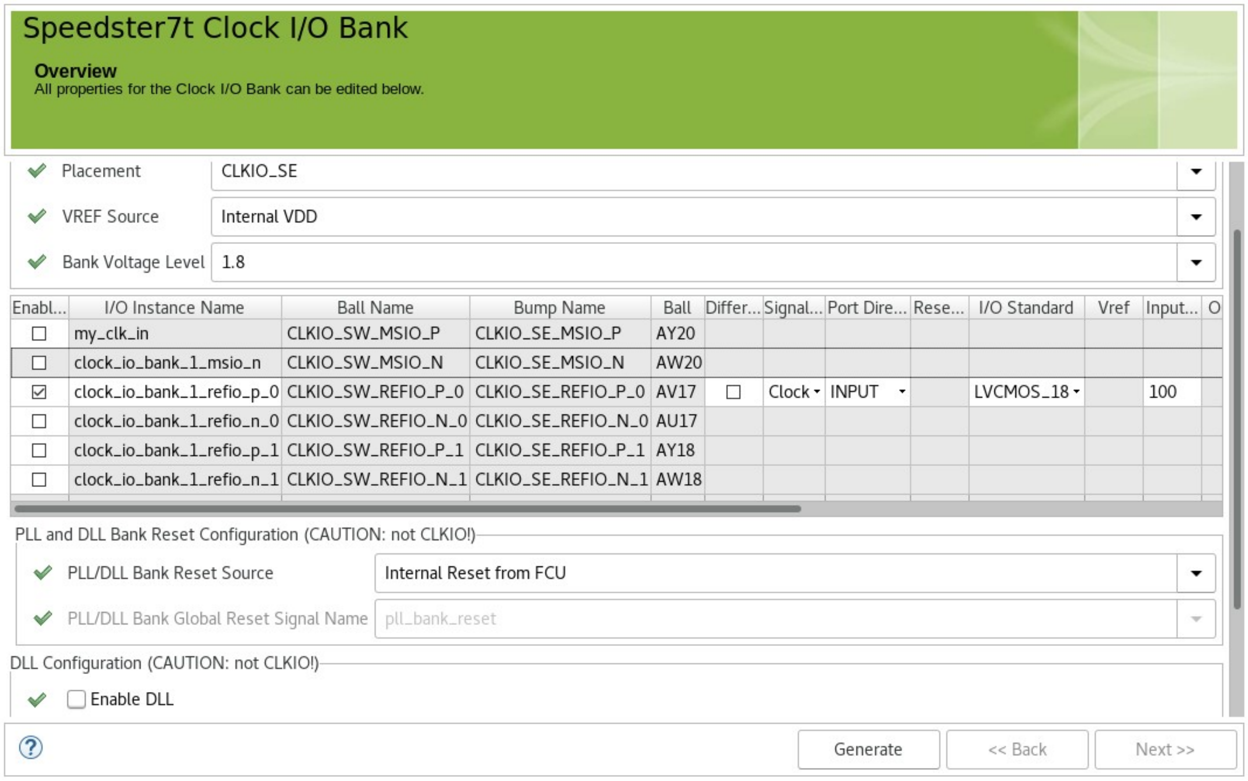
Chapter - 9: Speedster7t 2D NoC Software Support

The I/O Designer Toolkit allows configuring the interface subsystems, clocks, PLLs, GPIO, and the 2D NoC. This section details the steps needed to configure the 2D NoC.

Create Clocks and Configure the PLL

The first step in configuring the 2D NoC is to provide a global clock running at 200 MHz:

1. Connect a clock input using the clock I/O bank configuration in the I/O Designer Toolkit.


Speedster7t Clock I/O Bank

Overview

All properties for the Clock I/O Bank can be edited below.

✓ Placement	CLKIO_SE
✓ VREF Source	Internal VDD
✓ Bank Voltage Level	1.8

Enabl...	I/O Instance Name	Ball Name	Bump Name	Ball	Differ...	Signal...	Port Dire...	Rese...	I/O Standard	Vref	Input...	O
☐	my_clk_in	CLKIO_SW_MSIO_P	CLKIO_SE_MSIO_P	AY20								
☐	clock_io_bank_1_msio_n	CLKIO_SW_MSIO_N	CLKIO_SE_MSIO_N	AW20								
☑	clock_io_bank_1_refio_p_0	CLKIO_SW_REFIO_P_0	CLKIO_SE_REFIO_P_0	AV17	☐	Clock	INPUT		LVCMOS_18		100	
☐	clock_io_bank_1_refio_n_0	CLKIO_SW_REFIO_N_0	CLKIO_SE_REFIO_N_0	AU17								
☐	clock_io_bank_1_refio_p_1	CLKIO_SW_REFIO_P_1	CLKIO_SE_REFIO_P_1	AY18								
☐	clock_io_bank_1_refio_n_1	CLKIO_SW_REFIO_N_1	CLKIO_SE_REFIO_N_1	AW18								

PLL and DLL Bank Reset Configuration (CAUTION: not CLKIO!)

✓ PLL/DLL Bank Reset Source	Internal Reset from FCU
✓ PLL/DLL Bank Global Reset Signal Name	pll_bank_reset

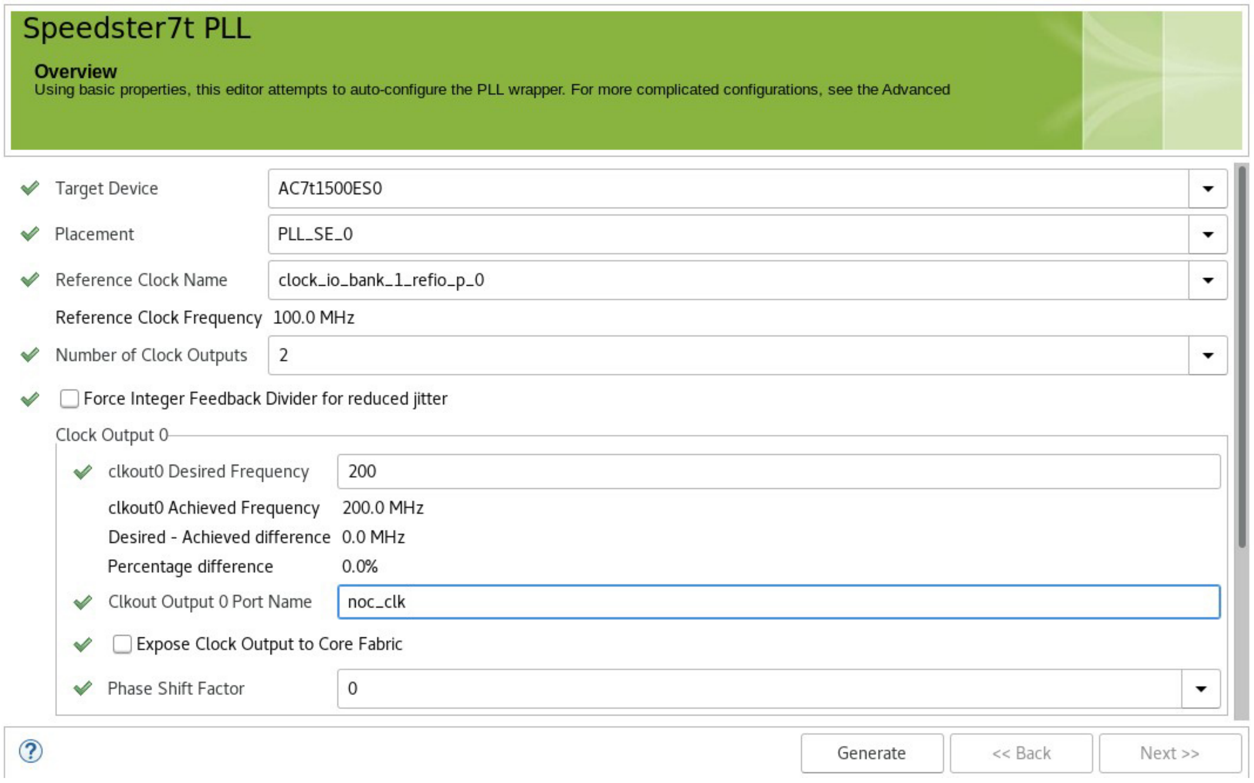
DLL Configuration (CAUTION: not CLKIO!)

✓ ☐ Enable DLL

Figure 21: *Clock I/O Bank Configuration*

2. Create a PLL using the PLL configuration GUI.
3. Configure the PLL so that it uses the new input clock as a reference input, and set the output frequency to 200 MHz. The output clock of the PLL can be renamed `noc_clk`, for example, to make it easier to identify.

4. If the clock for the 2D NoC is not used in the FPGA fabric, uncheck **Expose Clock Output to Core Fabric** so that it does not consume a clock resource in the FPGA fabric.



Speedster7t PLL

Overview
Using basic properties, this editor attempts to auto-configure the PLL wrapper. For more complicated configurations, see the Advanced

Target Device	AC7t1500ES0
Placement	PLL_SE_0
Reference Clock Name	clock_io_bank_1_refio_p_0
Reference Clock Frequency	100.0 MHz
Number of Clock Outputs	2
☐ Force Integer Feedback Divider for reduced jitter	

Clock Output 0

clkout0 Desired Frequency	200
clkout0 Achieved Frequency	200.0 MHz
Desired - Achieved difference	0.0 MHz
Percentage difference	0.0%
Clkout Output 0 Port Name	noc_clk
☐ Expose Clock Output to Core Fabric	
Phase Shift Factor	0

Figure 22: ***PLL Configuration***

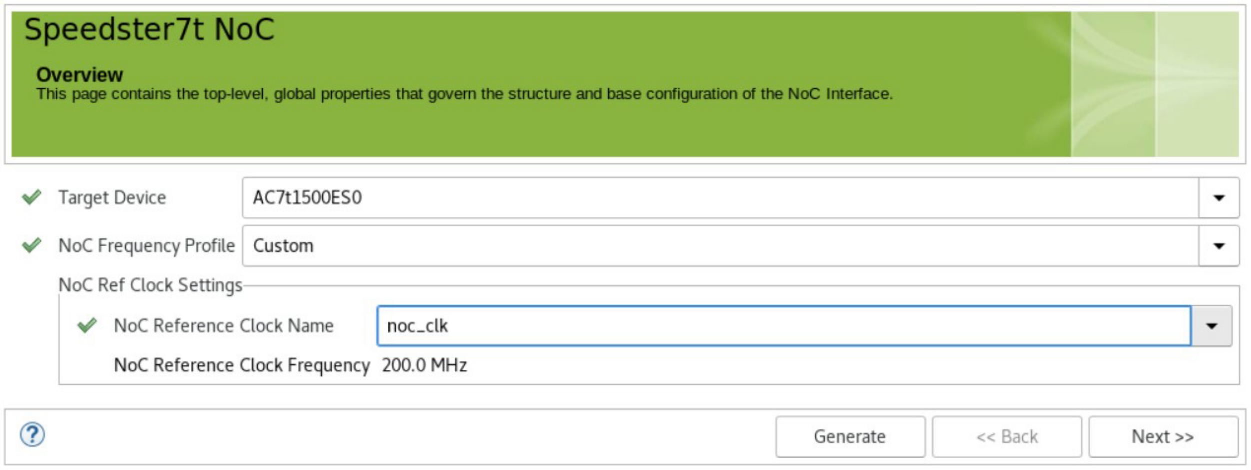
5. When this clock is configured to the desired specifications, configure the 2D NoC itself.

Configure the 2D NoC

Top-Level Configuration

1. Using the I/O Designer Toolkit, create a new 2D NoC IP configuration. This operation only needs to be performed once as there is only one 2D NoC in a Speedster7t FPGA.

2. As can be seen in the following figure, the target device is selected from a pull-down menu.



Speedster7t NoC

Overview
This page contains the top-level, global properties that govern the structure and base configuration of the NoC Interface.

Target Device: AC7t1500ES0
NoC Frequency Profile: Custom
NoC Ref Clock Settings
NoC Reference Clock Name: noc_clk
NoC Reference Clock Frequency: 200.0 MHz

Figure 23: ***2D NoC Configuration of Clock***

3. The frequency profile for the 2D NoC is also selected from the pull-down menu of several frequency options, or a **Custom** option can be selected.
4. If the **Custom** option is chosen, frequencies can be set on the next configuration page.
5. The reference clock name is selected from a pull-down list of valid clocks available in the design. In this case, it must be a 200 MHz clock.
6. Click **Next** to continue to the next configuration page.

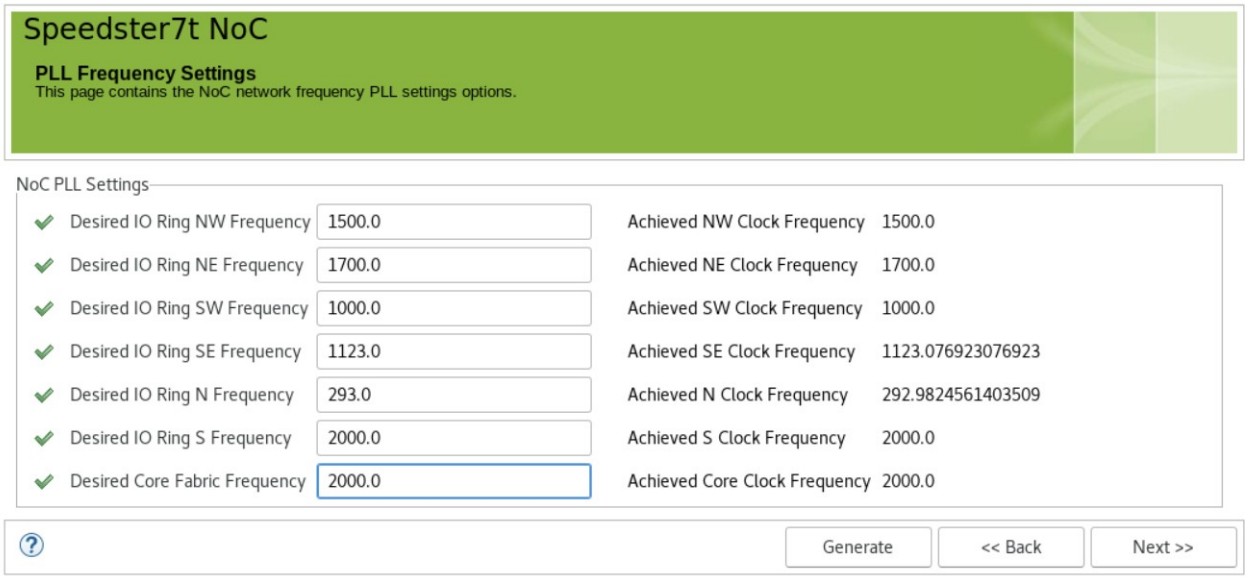
Individual Sections Clocking

This page in the IP Configuration GUI allows setting different frequencies for the different sections of the 2D NoC if the **Custom** option was chosen on the previous page.

If it is necessary to reduce power, and it is known that a portion of the 2D NoC is not used in the design, the frequency of certain segments of the 2D NoC may be reduced. In this case, the frequency of the six segments of the peripheral ring of the 2D NoC can be choosen, as well as the rows and columns, which is listed as **Core Fabric Frequency**. Possible frequency values range from 200 MHz up to 2.027 GHz. It is not possible to set the frequency to 0 MHz to turn off power. The frequency of each segment should be chosen such that it is fast enough to support the required design throughput.

1. The desired frequency is entered in the box on the left, and the achieved frequency is displayed on the right.

2. If a set frequency was chosen on the prior page, this page simply lists the achieved frequencies on each portion of the 2D NoC.



Speedster7t NoC

PLL Frequency Settings
This page contains the NoC network frequency PLL settings options.

NoC PLL Settings

✓ Desired IO Ring NW Frequency	1500.0	Achieved NW Clock Frequency	1500.0
✓ Desired IO Ring NE Frequency	1700.0	Achieved NE Clock Frequency	1700.0
✓ Desired IO Ring SW Frequency	1000.0	Achieved SW Clock Frequency	1000.0
✓ Desired IO Ring SE Frequency	1123.0	Achieved SE Clock Frequency	1123.076923076923
✓ Desired IO Ring N Frequency	293.0	Achieved N Clock Frequency	292.9824561403509
✓ Desired IO Ring S Frequency	2000.0	Achieved S Clock Frequency	2000.0
✓ Desired Core Fabric Frequency	2000.0	Achieved Core Clock Frequency	2000.0

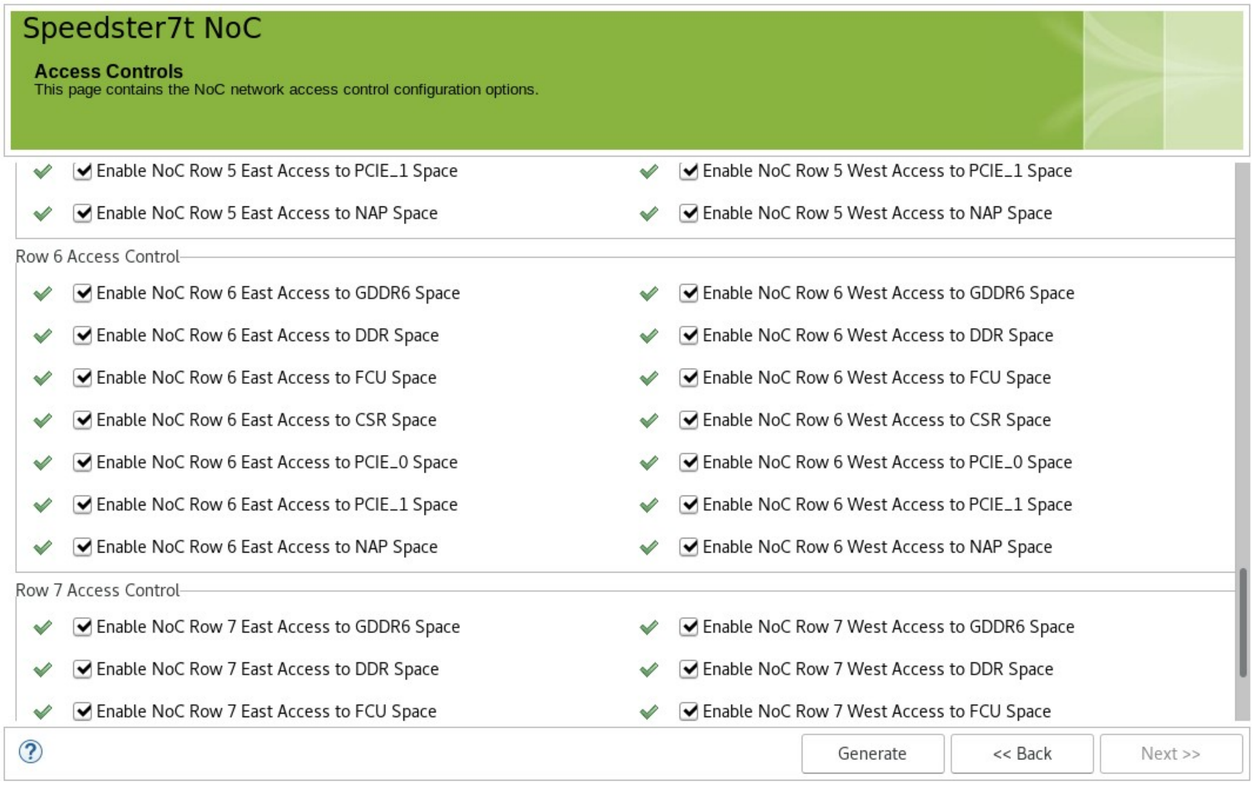
Figure 24: ***2D NoC Frequency Configuration***

3. Click **Next** to continue to the next configuration page.

Access Controls

This page in the IP Configuration GUI allows enabling or disabling access to different endpoints per 2D NoC row. For the Speedster7t AC7t1500 FPGA, there are access controls for all eight rows. This is where access to the entire GDDR6, DDR4, FCU, CSR spaces, plus PCIe 0, PCIe 1, or the entire NAP space can be turned on or off for transactions traveling east or west along that row of the 2D NoC. In other words, for any NAP on that row, the NAP can only access the spaces that are checked for that row.

1. After all configurations for the I/O Designer Toolkit are complete, click **Generate** to create all of the necessary output files for the I/O ring portion of the Speedster7t FPGA.



Speedster7t NoC

Access Controls
This page contains the NoC network access control configuration options.

✓ ☑ Enable NoC Row 5 East Access to PCIE_1 Space ✓ ☑ Enable NoC Row 5 West Access to PCIE_1 Space
✓ ☑ Enable NoC Row 5 East Access to NAP Space ✓ ☑ Enable NoC Row 5 West Access to NAP Space

Row 6 Access Control

✓ ☑ Enable NoC Row 6 East Access to GDDR6 Space ✓ ☑ Enable NoC Row 6 West Access to GDDR6 Space
✓ ☑ Enable NoC Row 6 East Access to DDR Space ✓ ☑ Enable NoC Row 6 West Access to DDR Space
✓ ☑ Enable NoC Row 6 East Access to FCU Space ✓ ☑ Enable NoC Row 6 West Access to FCU Space
✓ ☑ Enable NoC Row 6 East Access to CSR Space ✓ ☑ Enable NoC Row 6 West Access to CSR Space
✓ ☑ Enable NoC Row 6 East Access to PCIE_0 Space ✓ ☑ Enable NoC Row 6 West Access to PCIE_0 Space
✓ ☑ Enable NoC Row 6 East Access to PCIE_1 Space ✓ ☑ Enable NoC Row 6 West Access to PCIE_1 Space
✓ ☑ Enable NoC Row 6 East Access to NAP Space ✓ ☑ Enable NoC Row 6 West Access to NAP Space

Row 7 Access Control

✓ ☑ Enable NoC Row 7 East Access to GDDR6 Space ✓ ☑ Enable NoC Row 7 West Access to GDDR6 Space
✓ ☑ Enable NoC Row 7 East Access to DDR Space ✓ ☑ Enable NoC Row 7 West Access to DDR Space
✓ ☑ Enable NoC Row 7 East Access to FCU Space ✓ ☑ Enable NoC Row 7 West Access to FCU Space

Figure 25: ***2D NoC Configuration Row Enable***

2. Save the IP configuration as a `.acxip` file and add it to the design project.

Caution!

The NAPs located in the south half of the Speedster7t AC7t1500ES0 FPGA disable all transactions to the west by default.

Chapter - 10: Revision History

Version	Date	Description
1.0	19 Sep 2019	• Initial Achronix release.
1.1	03 Jun 2020	**Additions:** • Added details on the arbitration schemes in the peripheral ring of the NoC in Speedster7t Peripheral 2D NoC Transaction Arbitration (see page 13). • Included extra details about supported burst lengths in Speedster7t 2D NoC Rows and Columns AXI Mode (see page 17). • Added details on the arbitration schemes and information on how to configure arbitration weights in Speedster7t 2D NoC Rows and Columns Transaction Arbitration (see page 18). • Included further details on transactions and NAP placement in Speedster7t 2D NoC Connectivity Interface-to-Fabric Connections (see page 27). • Added new details in Speedster7t 2D NoC Simulation Support Simulating 2D NoC with DSM (see page 50). **Updates and Corrections:** • Minor updates and clarifications to Speedster7t 2D NoC Connectivity Ethernet-to-Fabric Connections (see page 28). • Updates and clarification to Speedster7t 2D NoC Connectivity Data Streaming (see page 33). • Corrected NAP numbering in Speedster7t 2D NoC Address Mapping (see page 36). • Updated and corrected details in Speedster7t 2D NoC Performance (see page 42). • Minor updates to Speedster7t 2D NoC Software Support (see page 51).
1.2	13 Apr 2023	• Add ACX_NAP_ETHERNET and updated all NAP figures in Speedster7t 2D NoC Access Point (see page 20). • Correct obsolete terminology. • Added GDDR6 address map in Speedster7t 2D NoC Address Mapping (see page 36). • Updates to Speedster7t 2D NoC Connectivity (see page 26), Speedster7t 2D NoC Performance (see page 42), Speedster7t 2D NoC Simulation Support (see page 46), and Speedster7t 2D NoC Software Support (see page 51).