Speedster7t Network on Chip User Guide (UG089)

Speedster FPGAs

Achronix®
Data Acceleration

Copyrights, Trademarks and Disclaimers

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

The Speedster7t FPGA family of devices has a network hierarchy that enables extremely high-speed dataflow 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 of 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. Achronix's network on chip (NoC) provides for read/write transactions throughout the device, as well as specialized support for 400G Ethernet streams in selected columns.

The NoC extends both vertically and horizontally over the FPGA fabric until it reaches the peripheral portion of the NoC. This structure provides an easy-to-use, high-bandwidth method to communicate between various masters and slaves on a Speedster7t device, including specialized connections between the Ethernet subsystem and NoC access points (NAPs) on select NoC columns in the FPGA fabric. In addition, the NoC provides a connection from the FPGA fabric and IP interfaces to the FPGA configuration unit (FCU). The FCU receives bitstreams and is used to configure the FPGA fabric as well as the various IP interfaces on the device. The 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 IP interfaces.

The features of the NoC described in this user guide generally pertain to the entire Speedster7t family of devices. In order to help users understand specific connections and features of the NoC, this user guide focuses on the NoC as implemented in the AC7t1500 device.

Master Endpoints

- 80 NoC access point (NAP) masters distributed throughout the FPGA core for user-implemented masters
- Two PCI Express Interfaces
- FPGA configuration unit (FCU)

Slave Endpoints

- 80 NAP slaves distributed throughout the FPGA core for user-implemented slaves
- 16x GDDR6 slave interfaces
- DDR4/5 controller
- Two PCI Express Interfaces
- All control and status register (CSR) interfaces of all IP cores
- FCU (to enable configuring of FPGA and interface IP subsystems)

Packet Endpoints

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

NoC Features

While the main purpose of the 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 NoC-to-FPGA interface can operate in several different modes: 256-bit advanced extensible Interface (AXI) slave, 256-bit AXI master, Ethernet packet, and NAP-to-NAP data streaming mode. These different modes provide a built-in way to communicate between endpoints without the user needing to design the logic themselves. The 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 figure below shows the NoC surrounded by high-speed interfaces on a Speedster7t1500 device, and the rows and columns of the NoC over the FPGA fabric.

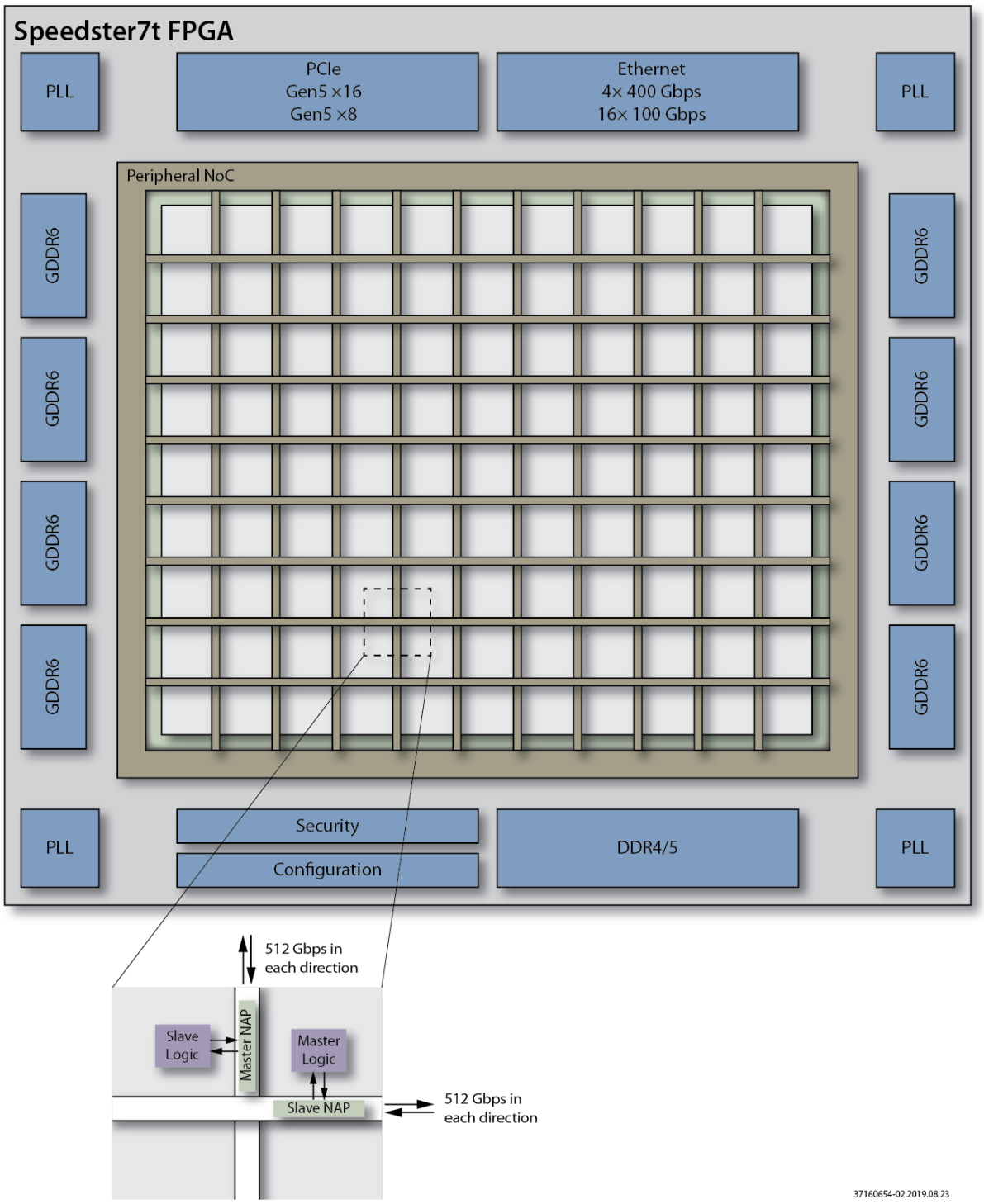

Figure 1: ***Speedcore7t NoC Showing Master and Slave Endpoints***

Chapter - 2: Speedster7t Peripheral NoC

Achronix's Speedster7t 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 NoC, along with its connections and features.

Peripheral NoC Features

The peripheral portion of the 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 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 NoC can be used without configuring the fabric, it also connects directly to the rows and columns of the NoC that run over the FPGA fabric, providing access to master and slave logic in the FPGA. Additionally, the peripheral ring of the NoC connects to the FPGA configuration unit (FCU), allowing the NoC to aid in configuration of the FPGA fabric, or the various interfaces. The figure below shows the peripheral portion of the NoC as it surrounds the FPGA fabric and provides high-bandwidth connections to the memory and networking interfaces.

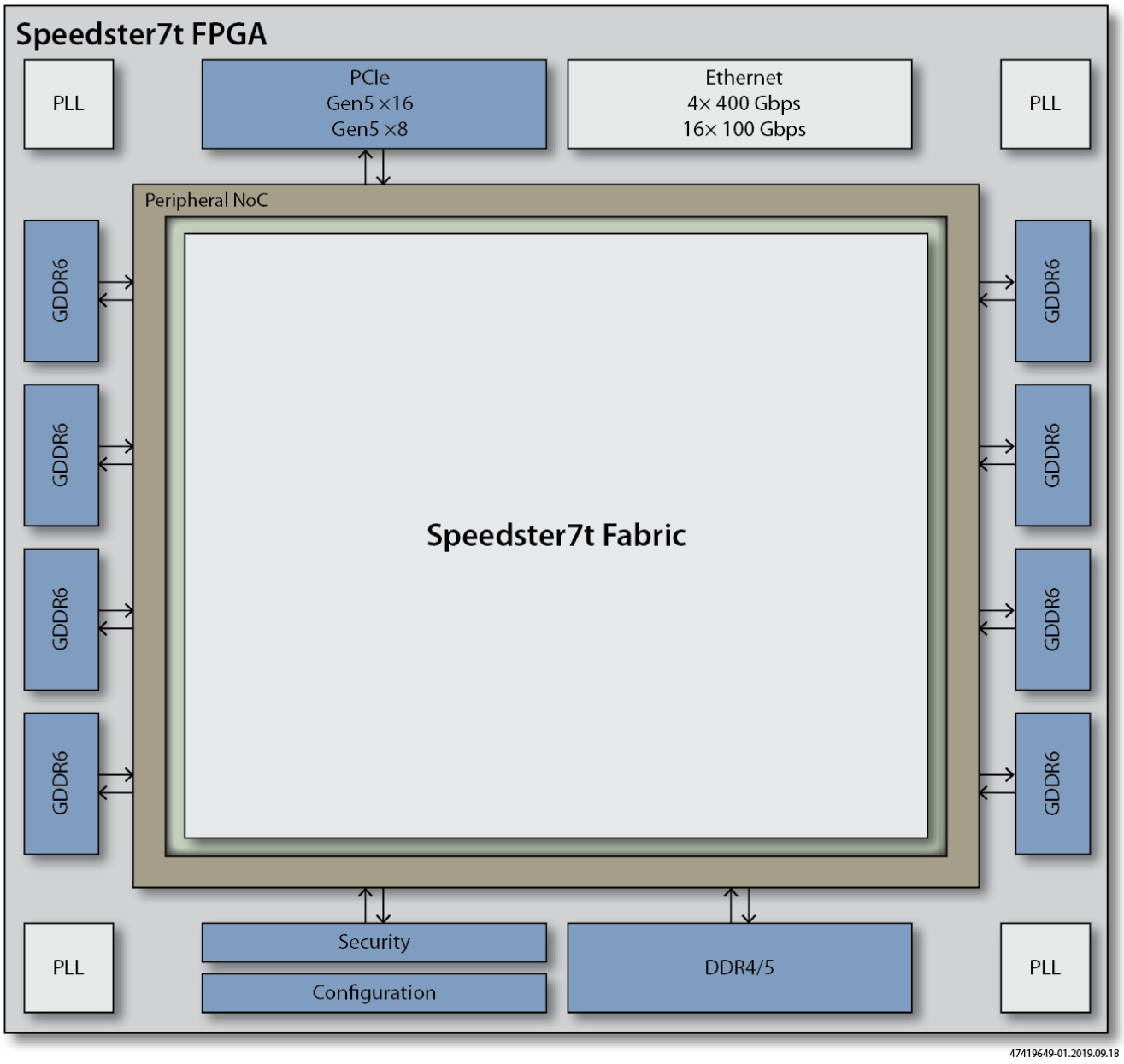

Figure 2: ***Speedster7t Peripheral NoC***

Modes of Operation

The NoC primarily uses AXI4 master/slave interfaces with read and write transactions. Masters initiate commands and slaves respond to commands by either writing the provided data or sending the requested read data. This mechanism provides easy-to-use connections between all the interfaces without the user 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 2 GHz.

Additionally, the NoC supports an advanced peripheral bus (APB) interface that can be used to program configuration 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 for example to program CSRs, the NoC handles all the translation from AXI4 transactions to APB.

Connections to NoC Peripheral Ring

The 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 NoC connects endpoints using a master/slave model, where the master initiates transactions and the slave responds to transactions. The peripheral portion of the NoC can connect the following endpoints.

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

PCIe to GDDR6 or DDR4

One PCIe endpoint can initiate transactions to either GDDR6 or DDR4 directly using the NoC. In this case, the PCIe endpoint is the master with either the GDDR6 or DDR4 as the slave. The 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. The user only needs to enable PCIe, GDDR6, and/or DDR4 in order to send transactions on the NoC.

PCIe to PCIe

Because the AC7t1500 contains two independent PCIe controllers, each PCIe can send transactions to the other via the NoC. Similar to connections with GDDR6 and DDR4, this high-bandwidth connection is achieved without consuming any FPGA fabric resources — the user only needs to enable both PCIe controllers in order to send transactions between them using the NoC.

PCIe to FCU

The PCIe endpoint can also connect directly to the FCU via the 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 to/from FPGA Fabric Logic

The PCIe endpoint can connect to logic in the FPGA fabric through the NoC. In this case, the PCIe endpoint can be the master or slave, and similarly the logic in the FPGA can be either the master or slave. If the PCIe endpoint is initiating transactions, the peripheral portion of the NoC sends transactions down the columns of the 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 NoC. If the logic in the FPGA fabric is acting as the master, it can initiate transactions to the NAP on a row of the NoC, which sends the transaction to the peripheral portion and then to the PCIe.

FPGA Fabric Logic to GDDR6 or DDR4

Master 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 NoC. This transaction travels east or west on the row until it reaches the peripheral portion of the NoC, and then to the destination GDDR6 or DDR4 channel.

FCU to All Endpoints

The FCU can act as a master to all other endpoints of the NoC, allowing the FCU to program the entire FPGA, including the interfaces. For example, using the 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 the memory of GDDR6 or DDR4. 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 NoC, refer to the Speedster7t NoC Connectivity (see page 23) section in this user guide.

Additional Features

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

Addressing

The NoC provides address decoding using a global address map to ensure transactions are sent to their intended destination. Additionally, the NoC supports address translation for flexibility and added security. For more information on the address map and address translation features, see the Speedster7t NoC Address Mapping (see page 33) section in this user guide.

Clock Domain Crossing

To make logic design easier, the NoC handles all clock domain crossing and transaction arbitration internally. This capability significantly simplifies user design while providing a way to easily transfer data operating at lower frequencies compared to the NoC. The transaction arbitration capability keeps data moving through the NoC without causing major congestion.

Functional Prior to Configuration

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

Chapter - 3: Speedster7t NoC Rows and Columns

The rows and columns of the 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 NoC through NoC access points (NAPs). The rows and columns are connected to the peripheral portion of the NoC, which communicates with the IP interfaces 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 NoC is placed in rows and columns at regular intervals over the FPGA fabric. The user logic connects to the NoC by way of NAPs and does not interfere with the connectivity of other logic within the fabric. Each row and column has a main 256-bit data path that operates at 2 GHz, delivering 512 Gbps of bidirectional bandwidth in all directions. While there are no direct connections between the rows and the columns, both connect to the peripheral ring of the NoC which allows for connections between points in the fabric. Master logic in the FPGA fabric connects to NAPs on the horizontal rows, and slave logic in the FPGA fabric connects to NAPs on the vertical columns of the NoC. The figure below shows an example of the NoC as constructed in the 7t1500 device. 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.

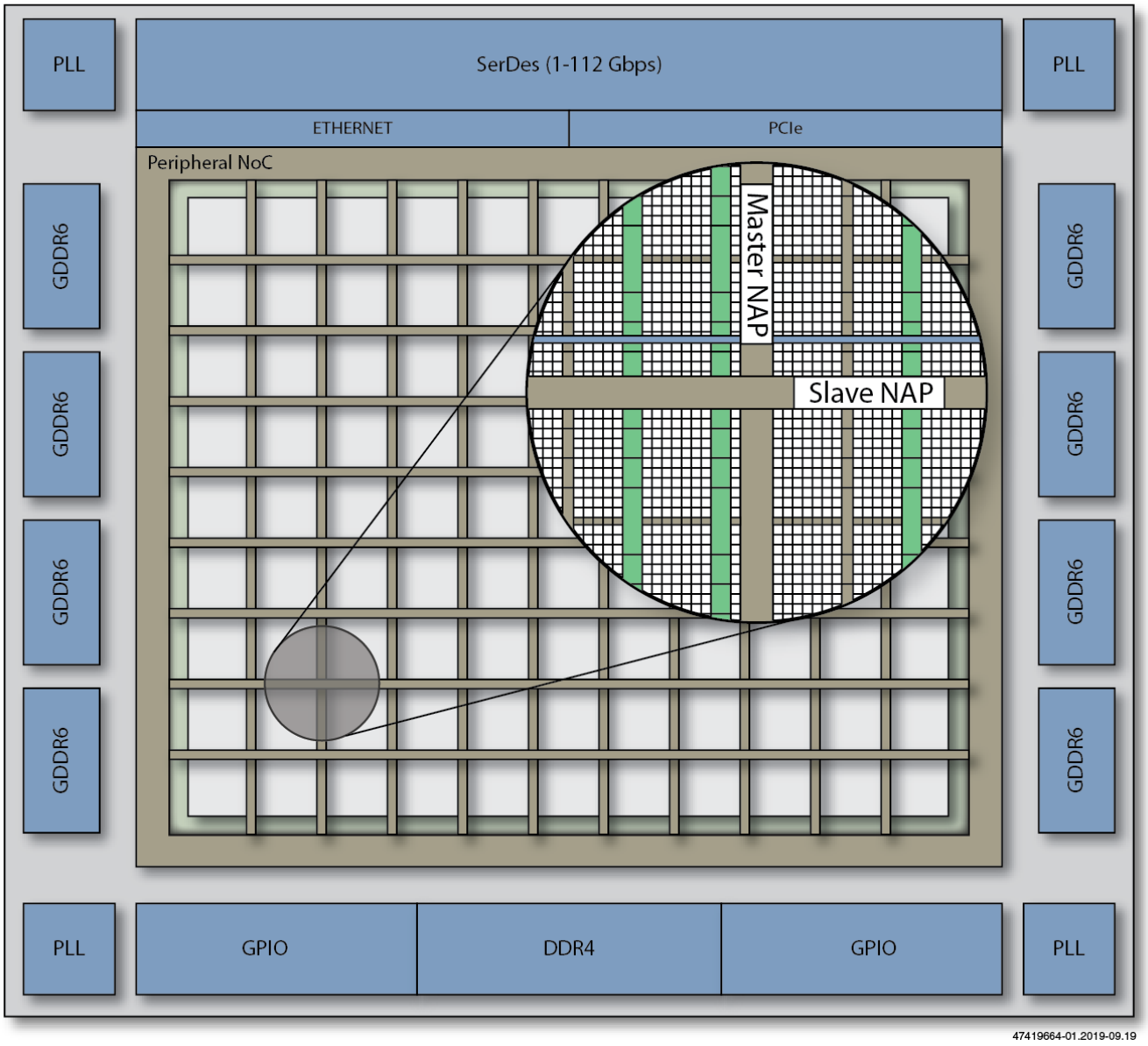

Figure 3: ***NoC Rows and Columns***

Modes of Operation

There are three main modes of operation in the rows and columns of the NoC. The mode which is used to communicate with most of the interface IP connected at the periphery of the NoC is AXI4, using the industry-standard AXI-4 interface protocol. Additionally, the internal fabric can connect to points on the same NoC row or same column using data streaming. Finally, the Ethernet interface is connected via dedicated columns using Ethernet packet transfers. These three modes are described in more detail below.

AXI Mode

AXI mode operates using slave and master logic with the master initiating transactions, and the slave responding to transactions. Master logic in the fabric FPGA can initiate commands to slave NAPs on NoC rows, which send the commands to slave endpoints such as GDDR6, DDR4, PCIe, and the FCU. Similarly, slave user logic in the FPGA fabric can respond to transactions from a master NAP on a column of the 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 may or may not be located on the same row or column of the NoC. For more details on AXI transactions, see the AMBA AXI Protocol Specification.

Data Streaming

User logic in the FPGA can communicate to another piece of logic in the fabric using data streaming on a single row or single column of the NoC. In this case, FPGA-to-FPGA transfers act like a FIFO. The start point and endpoint must be on the same row or same column of the NoC, using a simple signaling protocol. This protocol uses a valid signal to indicate valid data being sent for the transfer, a ready signal used to accept the data or signal back pressure, and a destination/source signal to indicate which NAP on the column or row is transmitting or receiving data. Data streaming uses all 288 bits of the data bus.

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

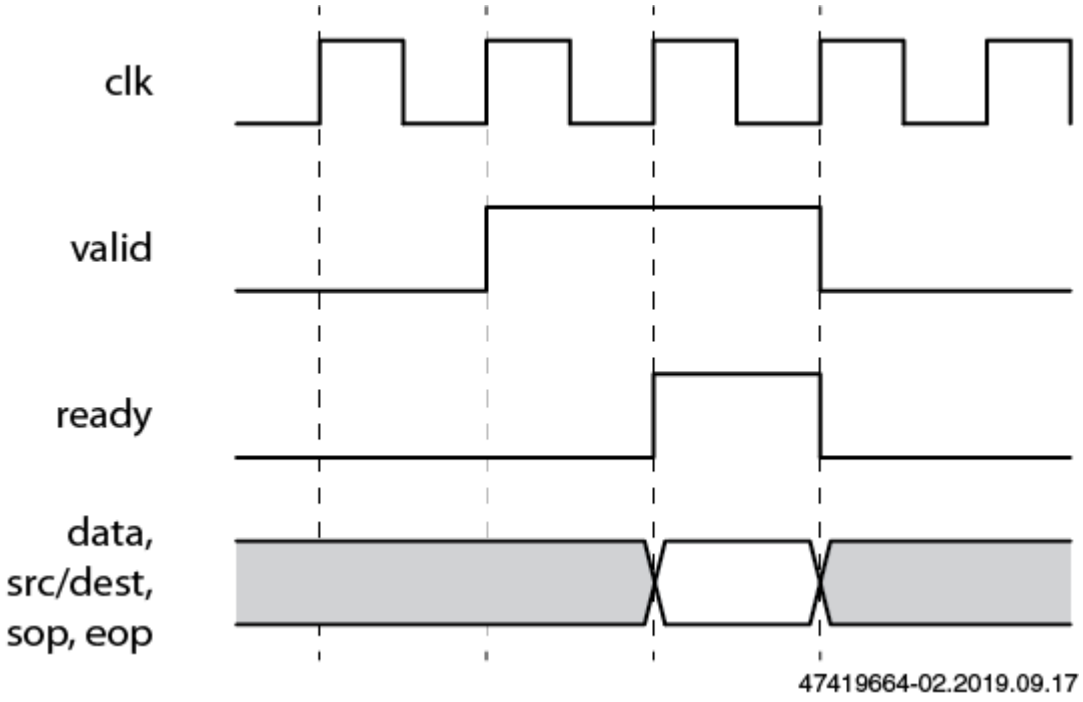

Figure 4: ***Data Streaming Timing Diagram with Valid Asserted First***

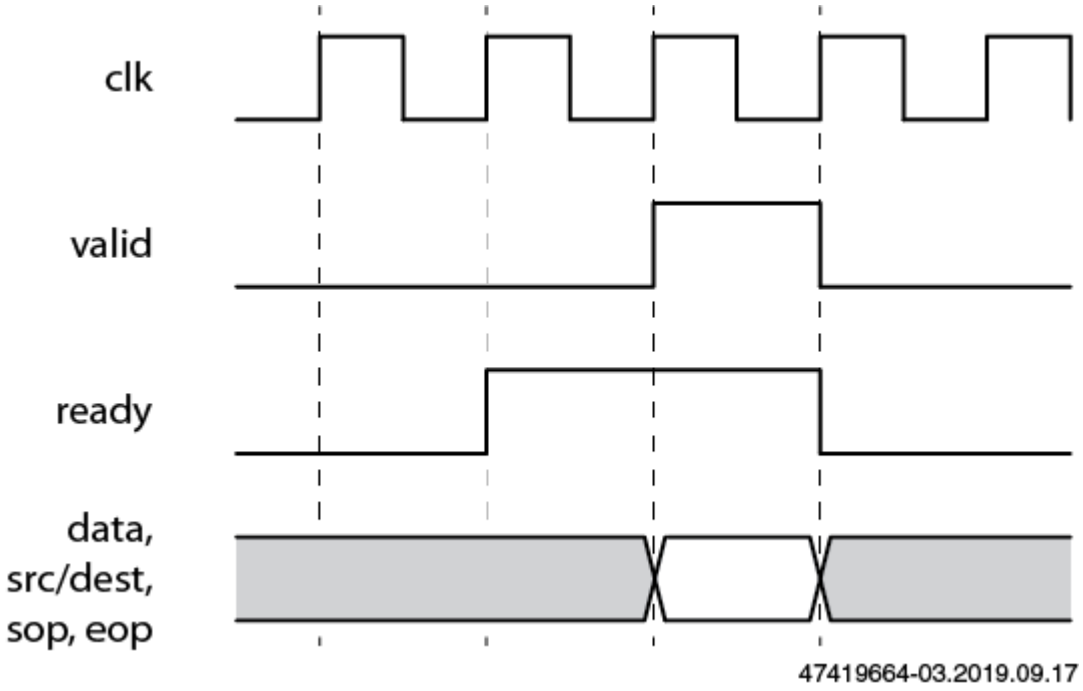

Figure 5: ***Data Streaming Timing Diagram with Ready Asserted First***

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 NoC. For more details on data streaming, see the sections, Speedster7t NoC Access Point (see page 18) or Speedster7t NoC Connectivity (see page 23).

Ethernet Packet Transfers

The Ethernet MAC interface is connected to dedicated columns in the 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 described above. For more details on Ethernet packet transfers on the NoC, see the section on Speedster7t NoC Connectivity (see page 23).

Additional Features

Clock Domain Crossing

The NoC handles clock domain crossing for any endpoints on the NoC. This feature allows user logic operating at a slower frequency to easily connect to the NoC without having to spend time to design resource-intensive and complicated clock domain crossing logic.

Transaction Arbitration

The NoC also handles arbitration of transactions internally, with time domain multiplexing used to interleave traffic from AXI transactions, Ethernet packets, and/or data streaming. This arbitration will not only keep traffic moving and prevent backups, but also keeps the NoC operating at its peak capacity.

Chapter - 4: Speedster7t NoC Access Point

The NoC access point (NAP) is the connection point from user logic in the fabric to the NoC. Users instantiate NAPs in their logic to connect to the rows and columns of the NoC. Depending on the function, users instantiate the appropriate NAP instance in their design. The figure below shows an example of how the NAPs connect to the NoC.

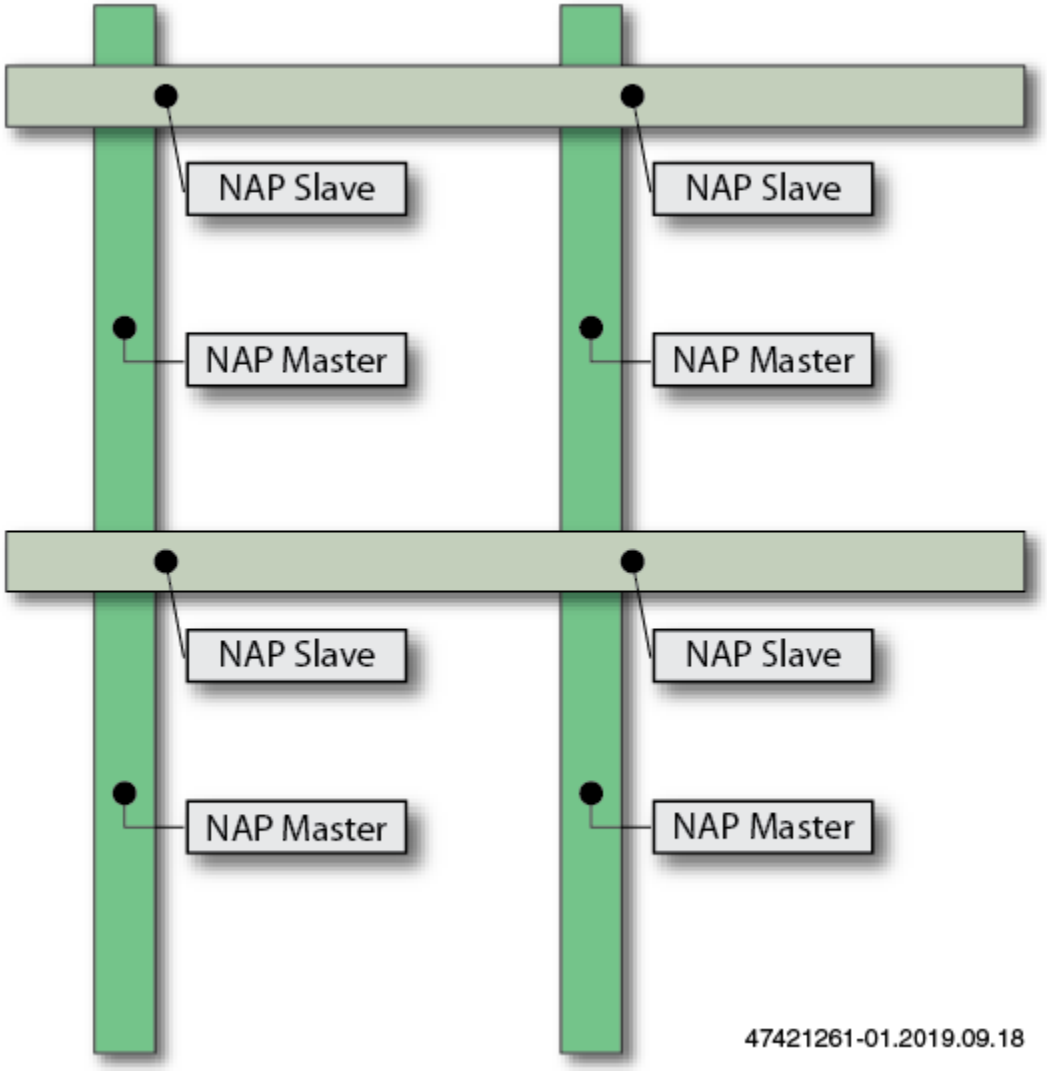

Figure 6: ***NoC Access Points in FPGA Fabric***

AXI Slave NAP

The ACX_NAP_AXI_SLAVE macro presents a 256-bit AXI slave to master user logic in the fabric and connects to the rows of the NoC. The resulting connection uses standard AXI4 protocol for read and write transactions and connects the user logic to any peripherals on the NoC, including the IP interfaces, as well as other user logic in the FPGA fabric connected through a NAP. The input clock is the clock used in the user logic in the FPGA fabric. The NoC uses this clock for any clock crossing logic. Below is a block diagram of the AXI slave NAP.

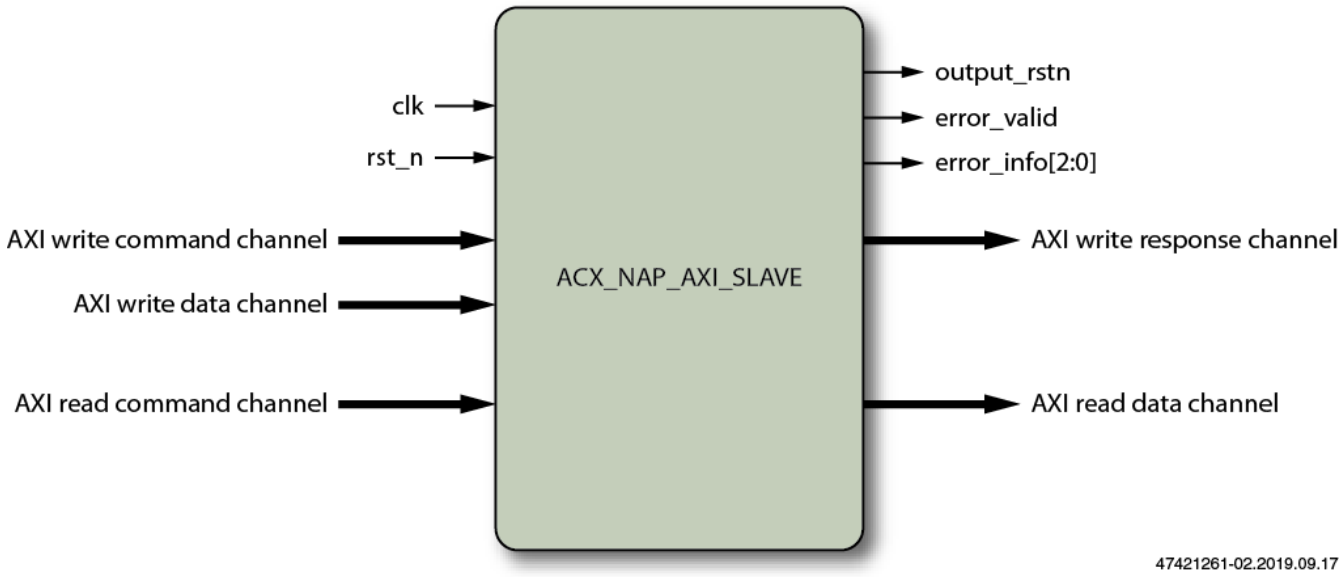

Figure 7: ***AXI Slave NAP Block Diagram***

For further details on the port names and instantiating the component, see the section, Speedster7t Network on Chip Primitives, in the *Speedster7t IP Component Library User Guide* (UG068).

AXI Master NAP

The ACX_NAP_AXI_MASTER macro presents a 256-bit AXI master to slave user logic in the fabric and connects to the columns of the NoC. The resulting connection uses standard AXI4 protocol for read and write transactions and connects the user logic to peripherals on the NoC, including the IP interfaces, as well as other user logic in the FPGA fabric connected through a NAP. The input clock is the clock used in the user logic in the FPGA fabric. The NoC uses this clock for any clock crossing logic. Below is a block diagram of the AXI master NAP.

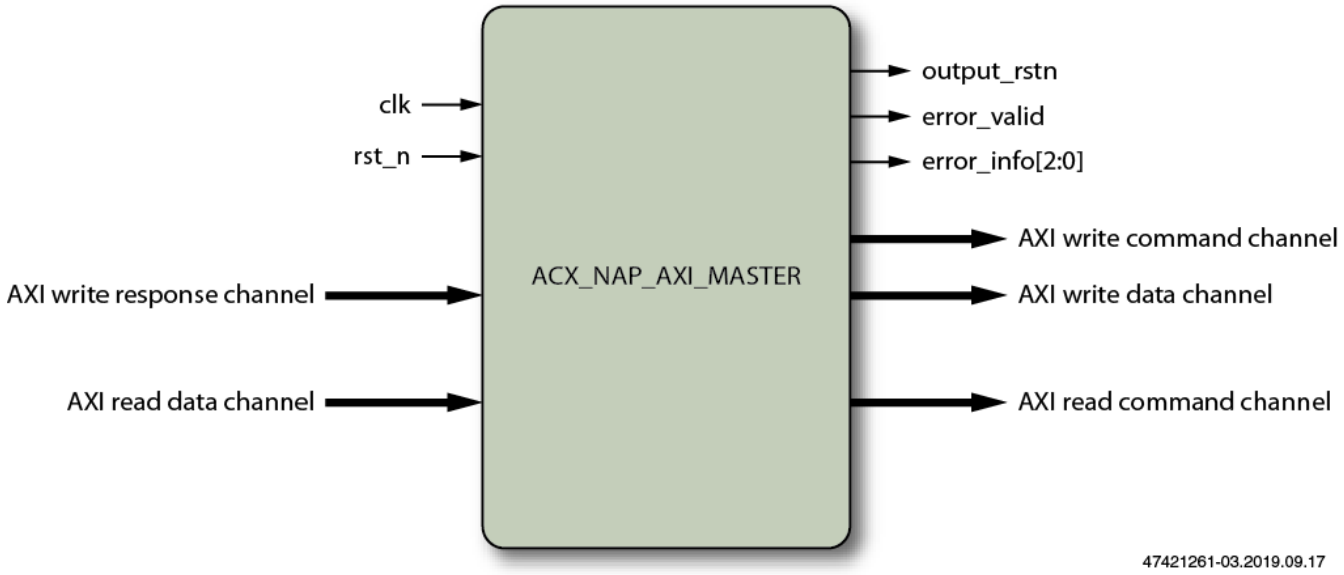

Figure 8: ***AXI Master NAP Block Diagram***

For further details on the port names and instantiating the component, see the section, Speedster7t Network on Chip Primitives, in the *Speedster7t IP Component Library User Guide* (UG068).

Horizontal NAP

The `ACX_NAP_HORIZONTAL` macro is used for data streaming along the rows of the NoC. The `ACX_NAP_HORIZONTAL` macro presents a 288-bit datapath to another `ACX_NAP_HORIZONTAL` instance on the same row of the 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's receiver interface. The input clock is the clock used in the user logic in the FPGA fabric. The NoC uses this clock for any clock crossing logic. Below is a block diagram of the horizontal NAP.

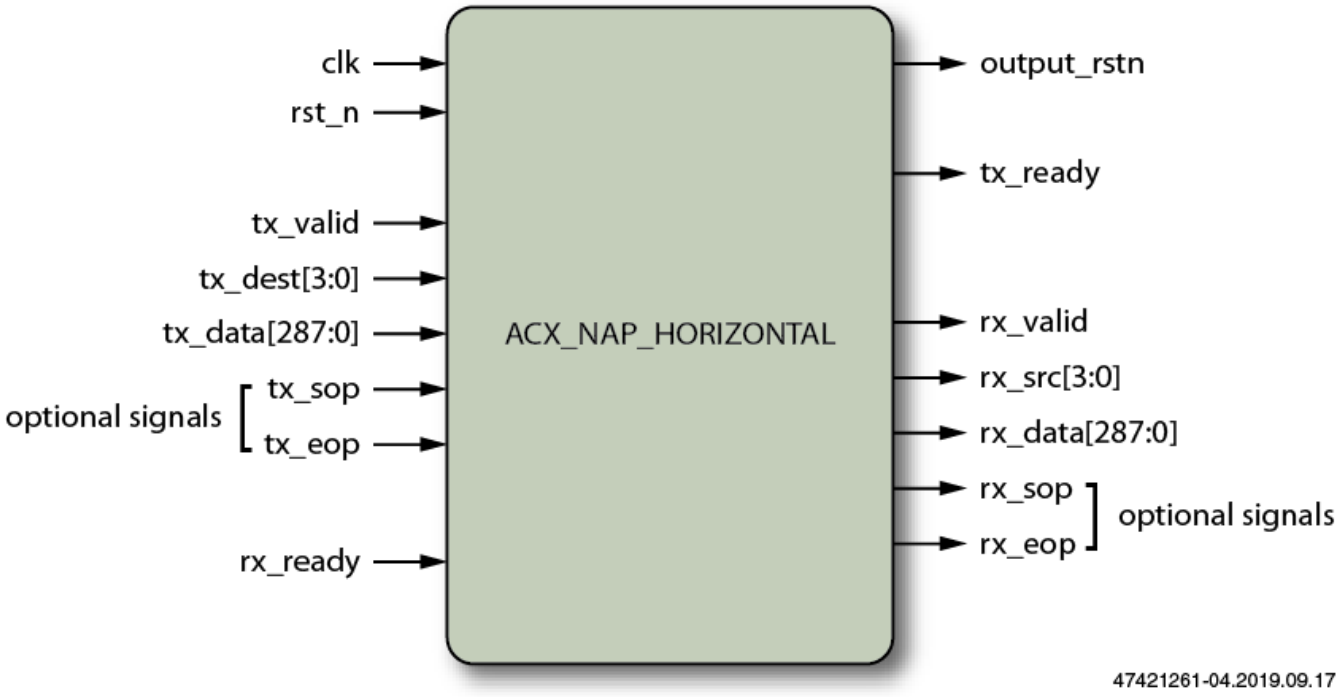

Figure 9: ***Horizontal NAP Block Diagram***

For further details on the port names and instantiating the component, see the section, Speedster7t Network on Chip Primitives, in the *Speedster7t IP Component Library User Guide* (UG068).

Vertical NAP

The `ACX_NAP_VERTICAL` macro is used for data streaming along the columns of the NoC. The `ACX_NAP_VERTICAL` macro presents a 293-bit datapath to another `ACX_NAP_VERTICAL` instance on the same column of the 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's receiver interface. The input clock is the clock used in the user logic in the FPGA fabric. The NoC uses this clock for any clock crossing logic. Below is a block diagram of the vertical NAP.

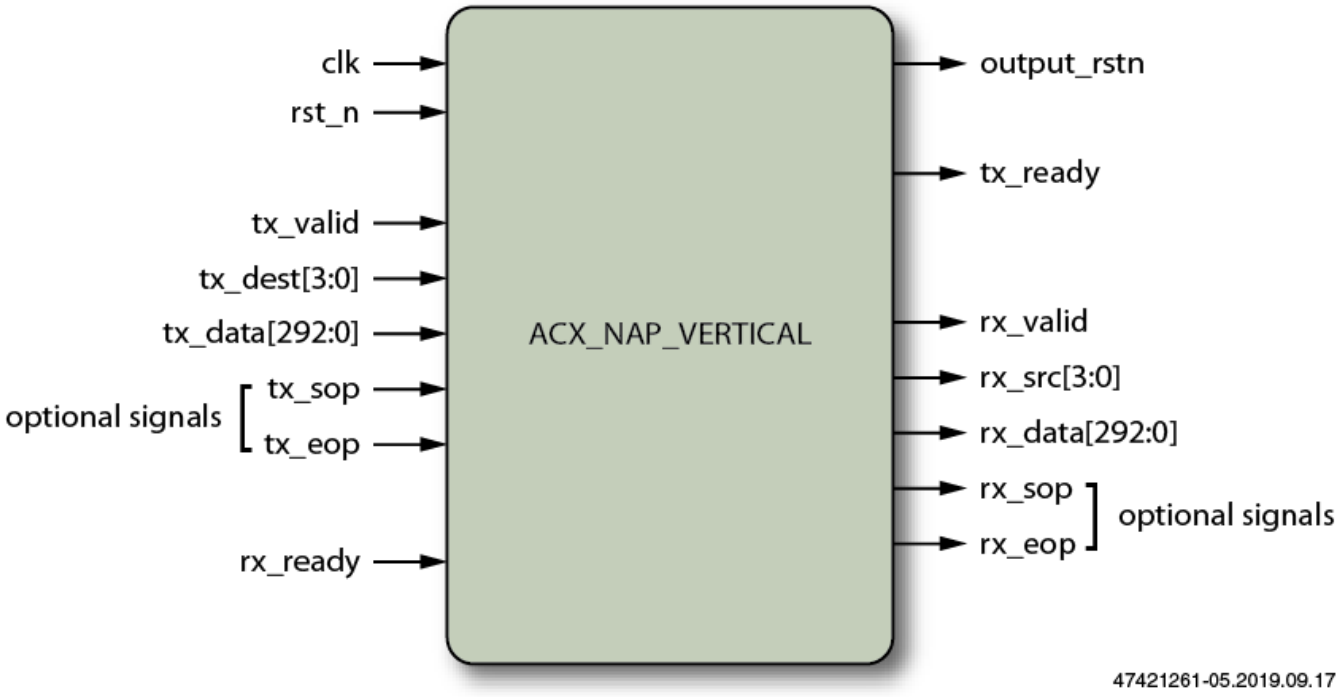

Figure 10: ***Vertical NAP Block Diagram***

For further details on port names and instantiating the component, see the section, Speedster7t Network on Chip Primitives, in the *Speedster7t IP Component Library User Guide* (UG068).

Chapter - 5: Speedster7t NoC Connectivity

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

Interface-only Connections

The NoC connects certain interface endpoints without using the FPGA fabric. Endpoint connections makes use of only the peripheral portion of the NoC and connects PCIe to GDDR6, PCIe to DDR4, PCIe to the FCU, and the FCU to PCIe, GDDR6, and DDR4. The connections between the PCIe, FCU, GDDR6, and DDR4 use the AXI4 protocol to send transactions. GDDR6 and DDR4 endpoints can only act as slaves, while the PCIe and FCU endpoints can act as either a master or slave. The figure below shows an example of the PCIe endpoint sending read or write transactions to a GDDR6 channel.

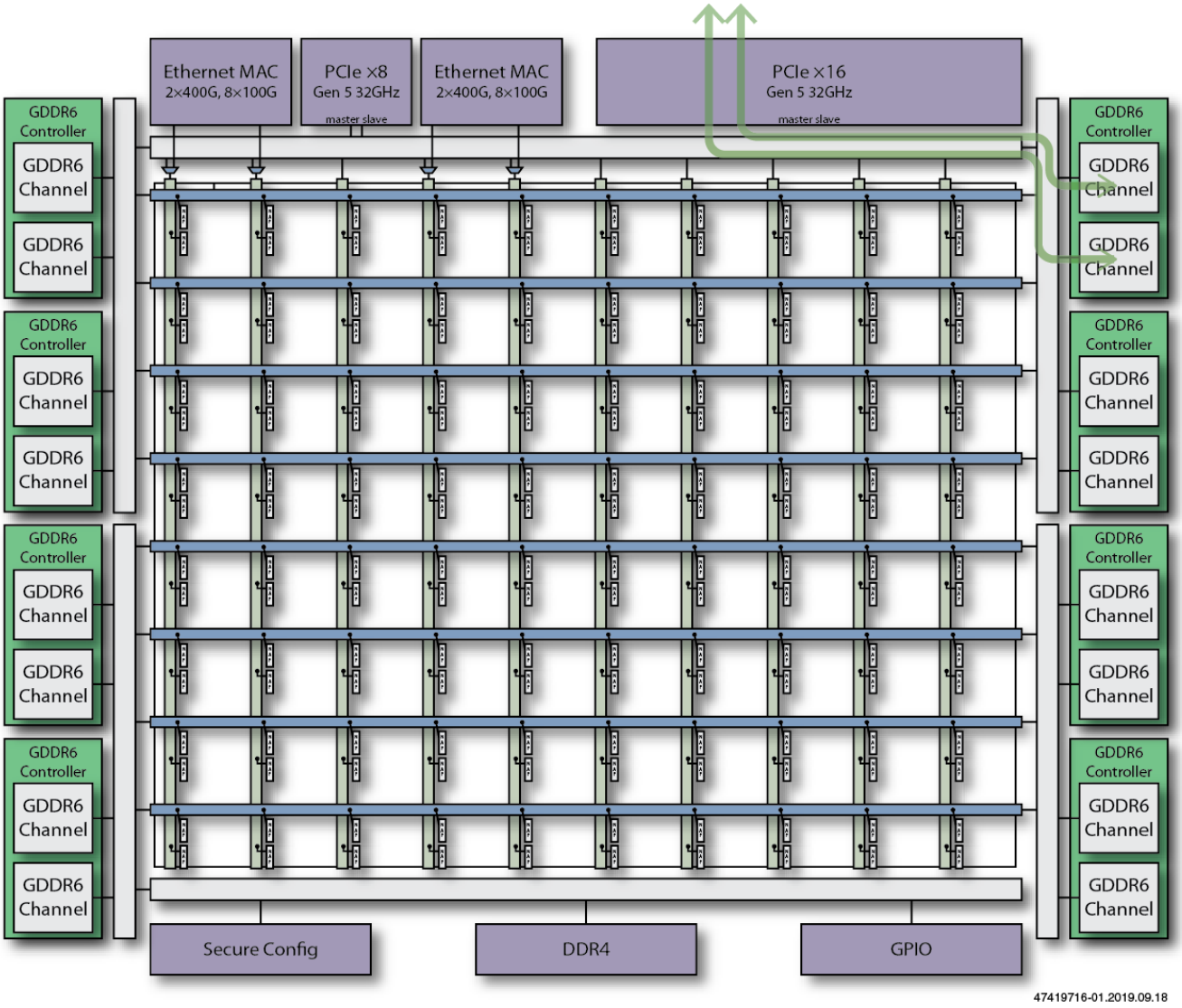

Figure 11: ***PCIe-to-GDDR6 Transactions***

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 NoC handles any clock domain crossing internally, as well as flow control and transaction arbitration; however, the user does need to consider the traffic flow to expected endpoints so as to optimize for low congestion on the peripheral portion of the NoC. For example, if the user is 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 to both the east and west side rather than sending all traffic down one side, thus reducing congestion on the NoC.

Interface-to-Fabric Connections

Interface IP can connect to master or slave logic in the FPGA fabric. The user needs to instantiate the appropriate ACX_NAP_AXI_SLAVE macro or ACX_NAP_AXI_MASTER macro depending on the type of logic in the fabric. The user logic only needs to use standard AXI4 protocol to communicate with the NAP through read and write transactions, which in turn connects the user logic through the NoC to the various IP interfaces. Master logic in the fabric can send transactions to the PCIe, GDDR6, DDR4, FCU, or CSR space. Additionally, the PCIe and FCU can talk to slave logic in the FPGA fabric. The figure below shows an example of the PCIe endpoint sending transactions to slave logic in the FPGA fabric.

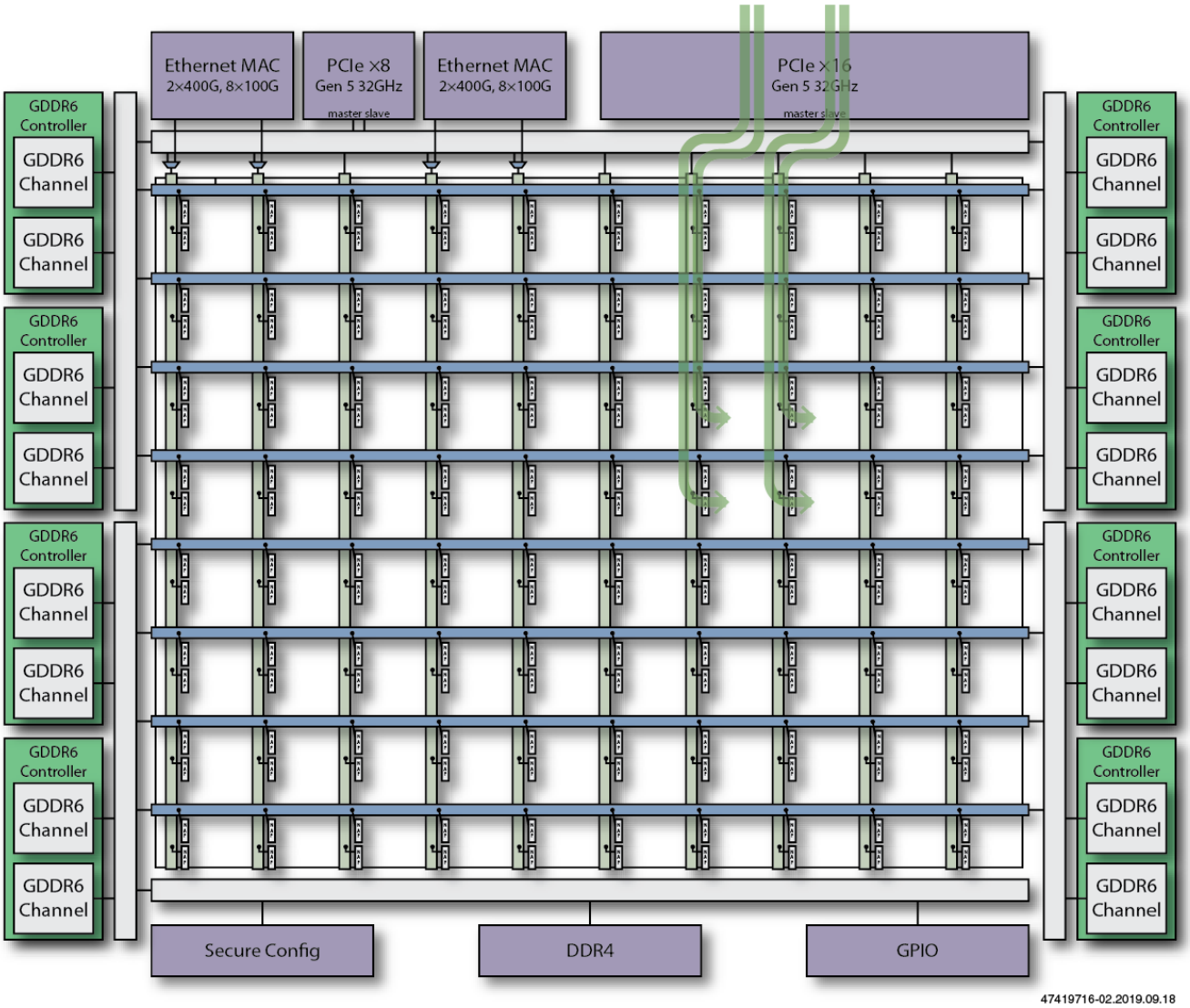

Figure 12: ***PCIe-to-FPGA Fabric Transactions***

The NoC handles any clock crossing logic and transaction arbitration internally, eliminating the need for the user to design this logic in the FPGA fabric. The user does need to consider placement of the NAPs in the fabric with respect to the IP interfaces if latency and congestion is a concern. 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. Additionally, if the user places logic along a single column or row in the FPGA, the traffic is concentrated on that one row or column. To reduce congestion, the user should consider expected traffic patterns and choose NAP locations that spread the transaction traffic across several rows or columns when possible.

Ethernet-to-Fabric Connections

The Ethernet MAC connects directly to specific columns on the NoC and can communicate to FPGA fabric logic connected to NAPs along those specific columns using Ethernet packets. Each Ethernet MAC has two dedicated columns and can send transactions to NAPs placed only on those two specific columns. The table below lists the specific columns connected to the Ethernet MACs.

Table 1: ***NoC Columns for Ethernet MACs***

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

Table Note

NoC Columns are numbered 1 at the west-most column and increment going east.

There are a few modes available, depending on how the user wishes to handle the Ethernet packets 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 NoC rearranges the 1024-bit data bus into four narrower data paths, funneling a separate packet to each of four NAPs and spliting the full 1024-bit data bus into four 256-bit (32-byte) data paths. This solution results in less congestion in the fabric because the user logic can reside in four separate engines distributed down the NoC columns rather than a single large engine immediately next to the Ethernet MAC. 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 NoC sending a sequence number along with each packet. The user logic is then responsible for reordering the packets (if necessary), in order to completely retrieve the original data sequence. The figure below shows how the Ethernet MAC data bus is rearranged into four separate 256-bit wide data buses. Each packet can take multiple cycles to complete.

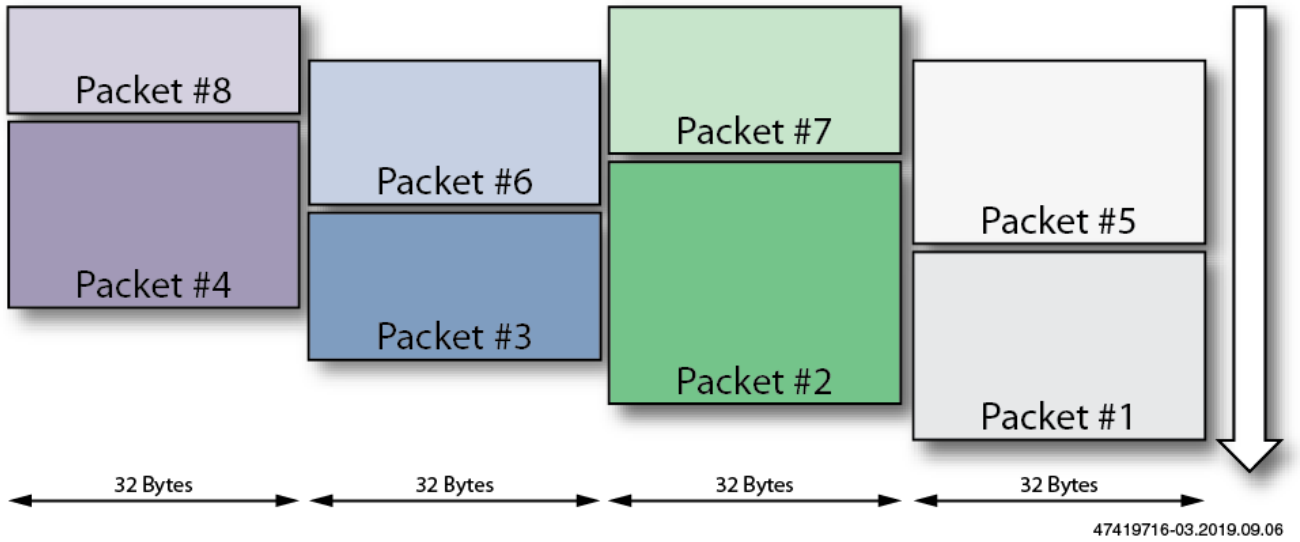

Figure 13: ***Data Bus Rearrangement for Packet Mode***

The four packets shown above are sent to four separate NAPs distributed down the designated NoC columns. Each NAP talks to an individual packet processing engine in the FPGA fabric that can run at a lower frequency than a single engine processing the full 1024-bit bus, thus simplifying the system design. The 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.

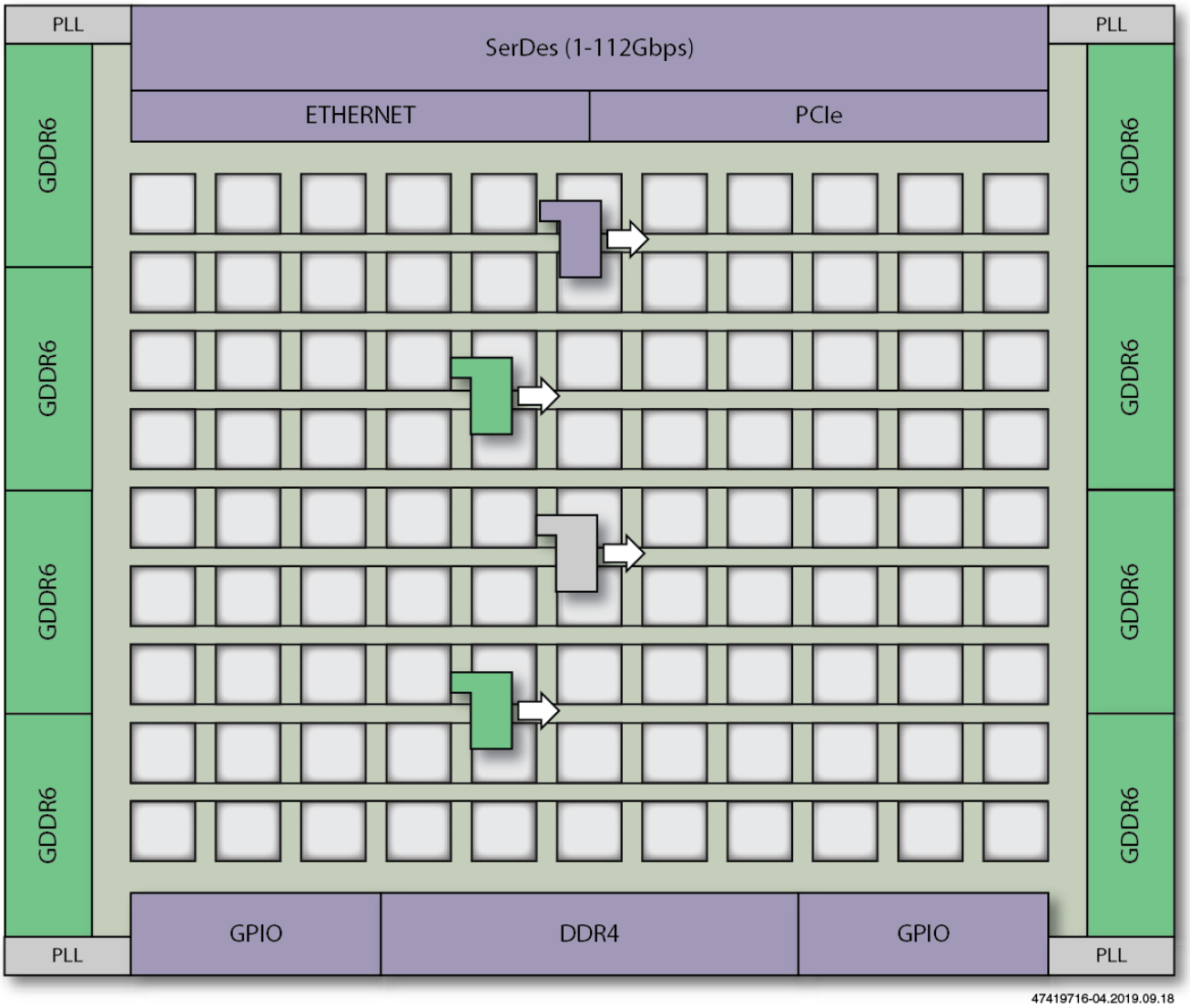

Figure 14: ***Ethernet Packet Mode on the NoC***

Quad-Segmented Mode

In quad-segmented mode, the NoC sends a 1024-bit bus that is segmented across 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 described above. 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 cycle.

Similar to the packet mode above, 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 MAC. 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. The figure below shows how the packets are arranged and segmented for the quad-segmented mode.

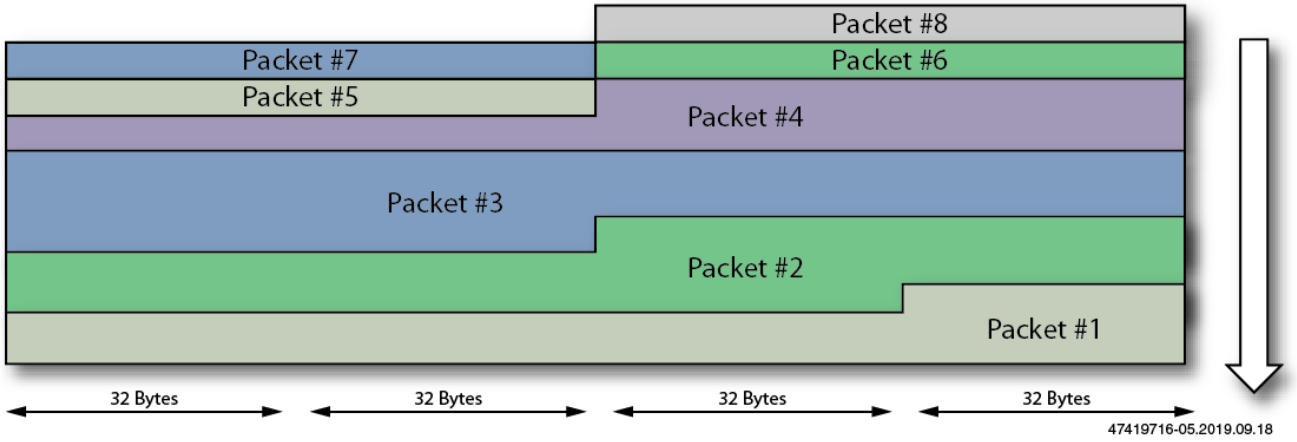

Figure 15: ***Packet Segmentation for Quad-Segmented Mode***

Each packet is distributed across four NAPs located on the designated columns of the NoC. The packet processing engine should be located close to the four NAPs.

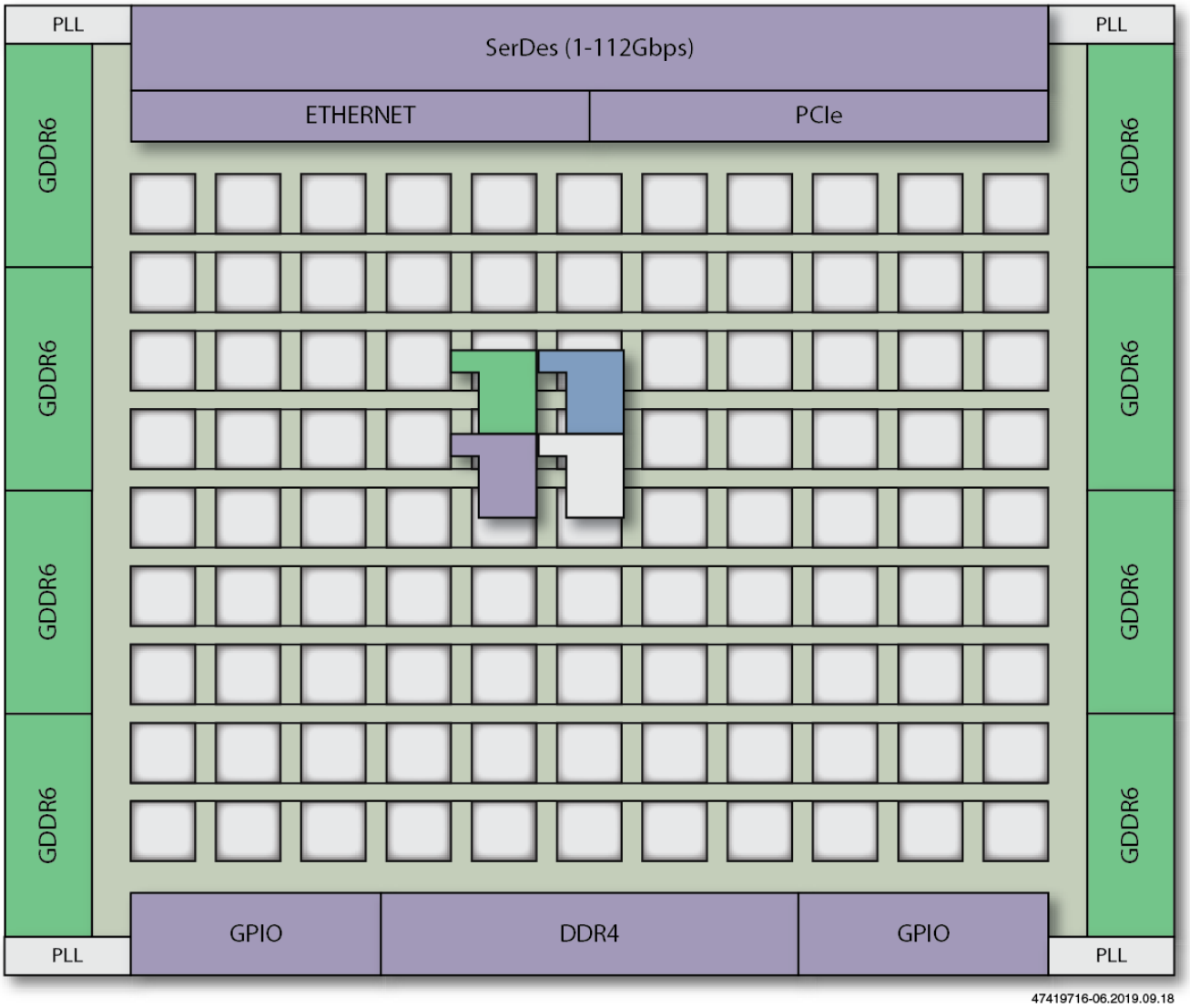

Figure 16: ***Quad-segmented Mode on the NoC***

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

Fabric-to-Fabric Connections

While logic in the FPGA fabric can communicate to other logic in the fabric in a traditional way using the FPGA's conventional routing resources, the NoC now enables designs to communicate between points within the FPGA fabric on a large, high-speed bus, without using the fabric routing resources. Depending on where the endpoints are located, and the style of transfer the user wishes to use, there are two methods to using the NoC for fabric-to-fabric communication: AXI commands or data streaming.

AXI Commands

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

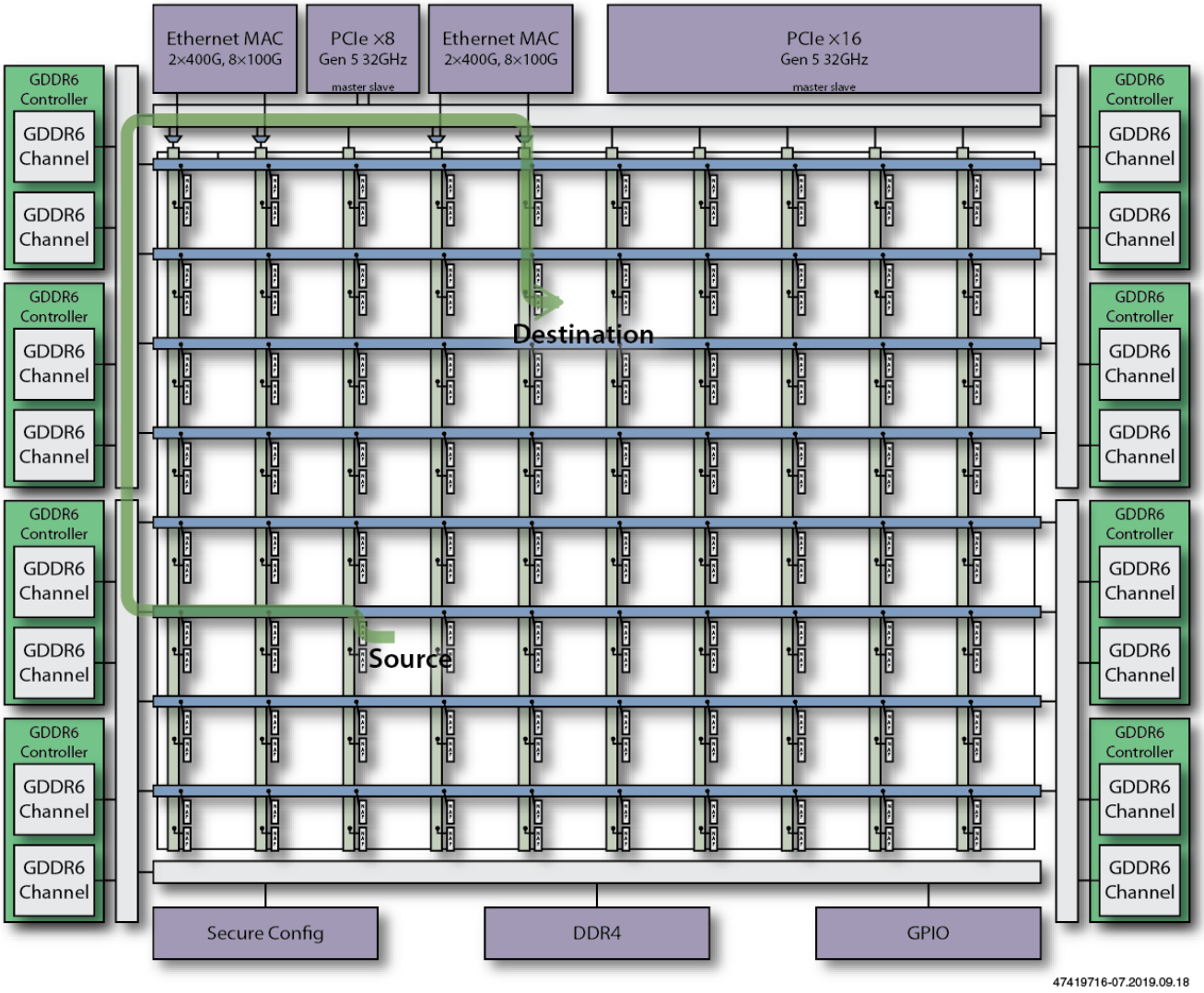

Figure 17: ***AXI Mode Fabric-to-Fabric Transaction***

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 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 is also a `tx_dest[3:0]` and `rx_src[3:0]` that indicates the transfer's destination and source, respectively. The location ID is a static number along the row or column.

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. Any clock crossing logic is automatically handled in the NoC.

The figure below shows transactions between various points in the NoC. The logic at points 1 and 2 have each instantiated a horizontal NAP. The NAPs can both send and receive data, but each individual data stream is just in one direction. Similarly, the logic at points 3 and 4 both instantiate a vertical NAP and can send data streams between each other.

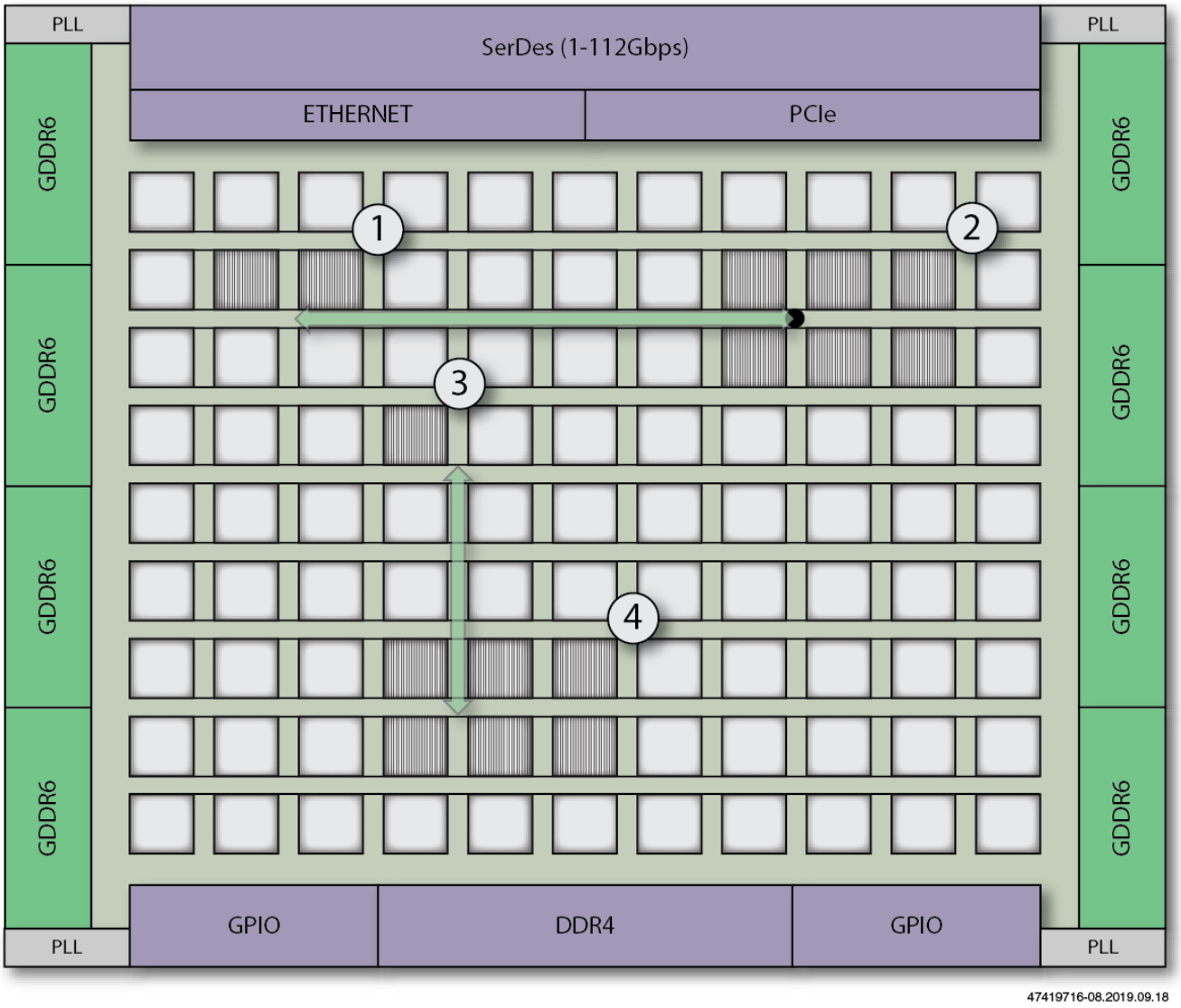

Figure 18: ***Data Streaming***

The NoC also handles transaction arbitration internally and can interleave the data streaming with AXI transactions. The user needs be aware of 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, the user needs 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 NoC Address Mapping

Global Address Map

The NoC has a global address map that is used to address all the endpoints in the FPGA. It uses a 42-bit address space, and includes regions that can be remapped with an address translation table for each NAP (see section on Address Translation (see page 36) below). The figure below shows the NoC address space and how each portion of the 42-bit address space is distributed.

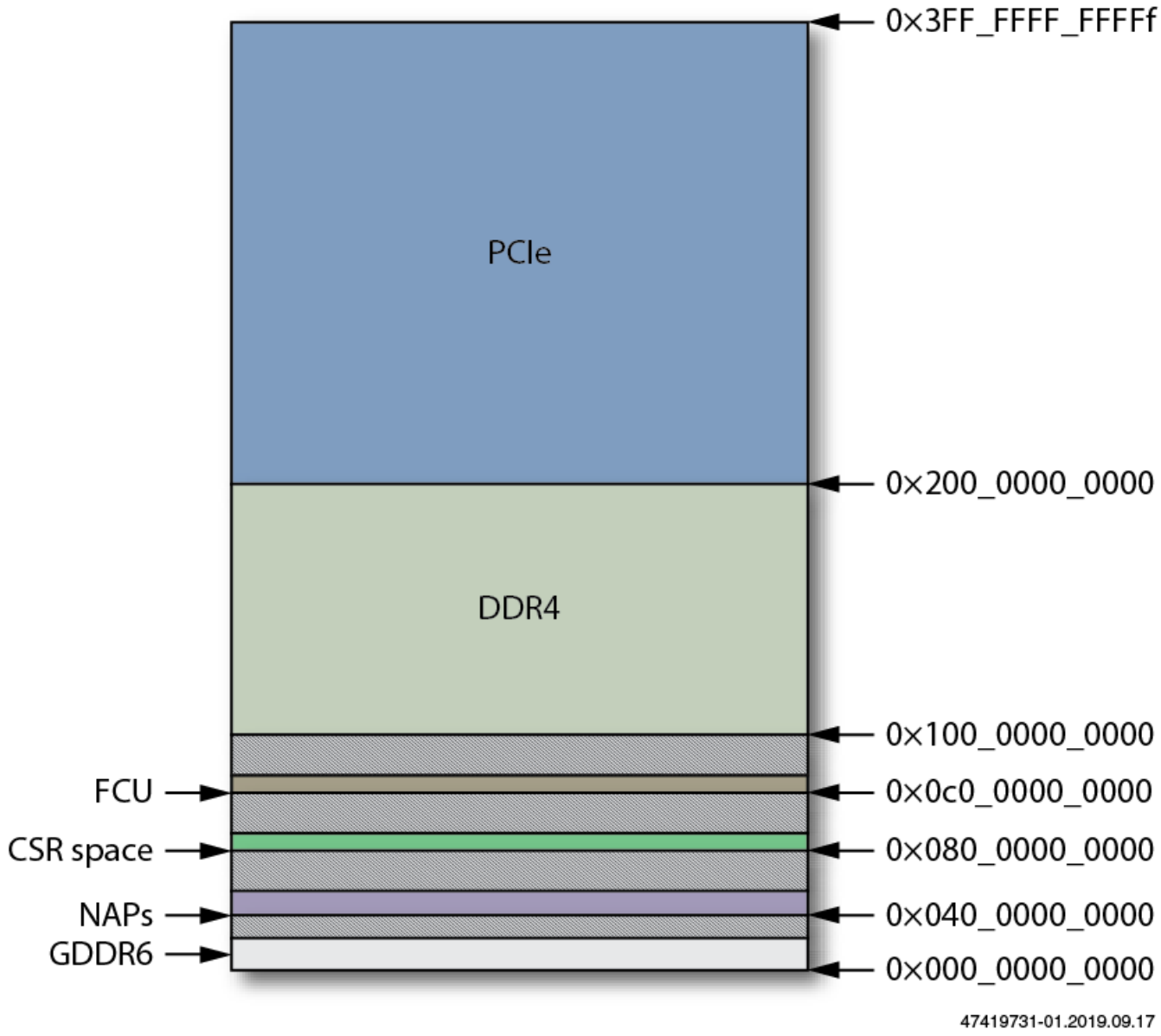

Figure 19: ***NoC Address Space***

Each of the endpoints on the NoC has its own address space. Below is the global address space table describing the details for each of the endpoints available on the NoC.

Table 2: ***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 NoC uses the most significant bits (MSB) in the address to identify the destination space of a transaction. Below is a description of each address space.

PCIe

- Addr[41] = `1'b1`
- Addr[40] = **ID** - Selects between the two PCIe IP cores.
- Addr[39:0] = **Memory Address** - This address is passed to the PCIe core.

DDR4

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

GDDR6

- Addr[41:37] = `5'b00000`
- Addr[36:33] = **Ctrl ID** - Selects which of the eight GDDR6 controllers the transaction is destined for. The three most significant bits of this field select the controller, the least significant bit selects between the two channels on each controller.
- Addr[32:0] = **Memory Address** - The memory address for the specific controller and channel.

NAP

- Addr[41:35] = `7'b0001000` – This space reaches any NAP endpoint in the device

- Addr[34:31] = **NAP Column** - Indicates which column number; 1 indicates the west-most column and the numbers increment going east.
- Addr[30:28] = **NAP Row** - Indicates which row number; 1 indicates the north-most row and the numbers increment going south.
- Addr[27:0] = **Memory Address** - This address is passed to the FPGA fabric logic.

CSR Space

- Addr[41:34] = `8'b00100000` – This space reaches all the control and status registers in the FPGA.
- Addr[33:28] = **Target ID** - Selects the space (DDR4, PCIe, Ethernet, 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.

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 a master on the NoC. The master initiates an AXI transaction to the particular address of a register in the CSR space, allowing the master 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. Below is the target IP spaces address map.

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

Table 3: *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

Target ID							Description
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 a user wishes to have several engines accessing GDDR6 and wants to reuse the same RTL for each engine, this can be done easily. A module can be written to access GDDR6 0, but then the user can configure the translation tables 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, the user can prevent access to certain endpoints, for example, to add security such that a user can prevent two engines from accessing the same memory. The I/O Designer Toolkit's NoC configuration GUI in ACE provides a simple way to disable access per 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 the user to remap pages in the memory.

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 of the Ctrl ID can be used in address translation allowing the user to remap which GDDR6 controller receives a transaction.

Bits[28:26] of the GDDR6 memory address can be used in address translation allowing the user to remap pages in the memory.

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

NAP column (bits[34:31]) and NAP row (bits[30:28]) can be used in address translation, allowing users to remap the location of the NAP transaction.

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 NoC Performance

The NoC is optimized for high bandwidth and supports a cross-sectional bidirectional bandwidth of 20 Tbps. The 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 NoC can run at high speeds, while the FPGA fabric and IP interfaces can run at slower frequencies as needed.

Latency and Performance

In order to increase flexibility for user designs, the 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 NoC. Each NAP has a small asynchronous FIFO adding a few clock cycles in each direction, adding a small amount of latency to transactions. Additionally, there is some latency added to traverse a 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 3 × 2 GHz, or 1.5 ns per NAP along the column.

Power

The NoC has different aspects of its power consumption that users should understand. For each NoC access point (NAP) there are two portions that consume power:

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

The high-speed portion of each NAP and the rows and columns of the NoC itself can be turned off completely if the user does not connect a clock to it. In that case, the rows and columns of the NoC do not contribute to dynamic power, but it also cannot be used to connect to the FPGA fabric. The peripheral portion of the NoC always has a slower configuration clock connected to it, so it can remain in a lower power state if not being used by the IP interfaces.

Chapter - 8: Speedster7t NoC Simulation Support

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

- A bus functional model of the NAP for simple functional simulations
- A model of the full rows and columns of the NoC to simulate latency and congestion between NAPs
- A full cycle-accurate model of the entire system including IP interfaces

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 bus functional models (BFMs) in each instance of a NAP macro. Each NAP includes simple tasks, and the user can call these tasks 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 example below 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 Achronix's mlp_conv2d reference design.

NAP Task BFM Binding Example

```
// Testbench has to connect to NAP slave 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_slave_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_slave_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
```

NAP_AXI_SLAVE Macro

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

Table 4: *NAP AXI Slave Tasks*

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

NAP_AXI_MASTER Macro

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

Table 5: *NAP AXI Master Tasks*

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

NAP_HORIZONTAL Macro

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

Table 6: *NAP Horizontal Tasks*

Task Name	Description
`issue_rx`	Issue a flit transfer and wait for it to be accepted.

Task Name	Description
get_tx	Receive a flit transfer request, assert ready when ready and waits for a valid.

NAP_VERTICAL Macro

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

Table 7: *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 waits for a valid.

Simulating NoC and Full System

Currently ACE libraries provide support for the NAPs with BFMs to simulate the FPGA logic sending/receiving transactions to or from a single NAP. In order to accurately simulate latency and congestion within the NoC, the user needs a full model of the rows and columns of the NoC. Additionally, for logic sending/receiving transactions to or from the IP interfaces, a user may need a full system model including accurate delays and traffic on the peripheral portion of the NoC, as well as IP interface models. Detailed descriptions of how to use these models will be included in this user guide once available.

> **Caution**
>
> The simulation model of the full chip including the NoC and its rows and columns is not currently available in the ACE tool suite.

Chapter - 9: Speedster7t NoC Software Support

The I/O Designer Toolkit allows users to configure the IP interfaces, clocks, PLLs, GPIO, and the NoC. This section describes the steps needed to configure the NoC.

Create Clocks and Configure the PLL

The first step in configuring the NoC is to provide a global clock running at 200 MHz. The simplest method is to first create a clock using the GPIO IP configuration in the I/O Designer Toolkit. Then create a PLL that uses the new clock as a reference input, and set the output frequency to 200 MHz. Once this clock is configured to the user's specifications, the next step is to configure the NoC itself.

Configure the NoC

First, using the I/O Designer Toolkit create a new IP configuration of the NoC. This operation only needs to be performed once as there is only one NoC in a Speedster7t device. First, the user connects the clock from the PLL. As can be seen in the figure below, the user sets the target device from a pull-down menu. Additionally, the user sets the reference clock name chosen from a pull-down list of valid clocks available in the design. In this case, it must be a 200 MHz clock. Then click **Next** to continue to the next configuration tab.

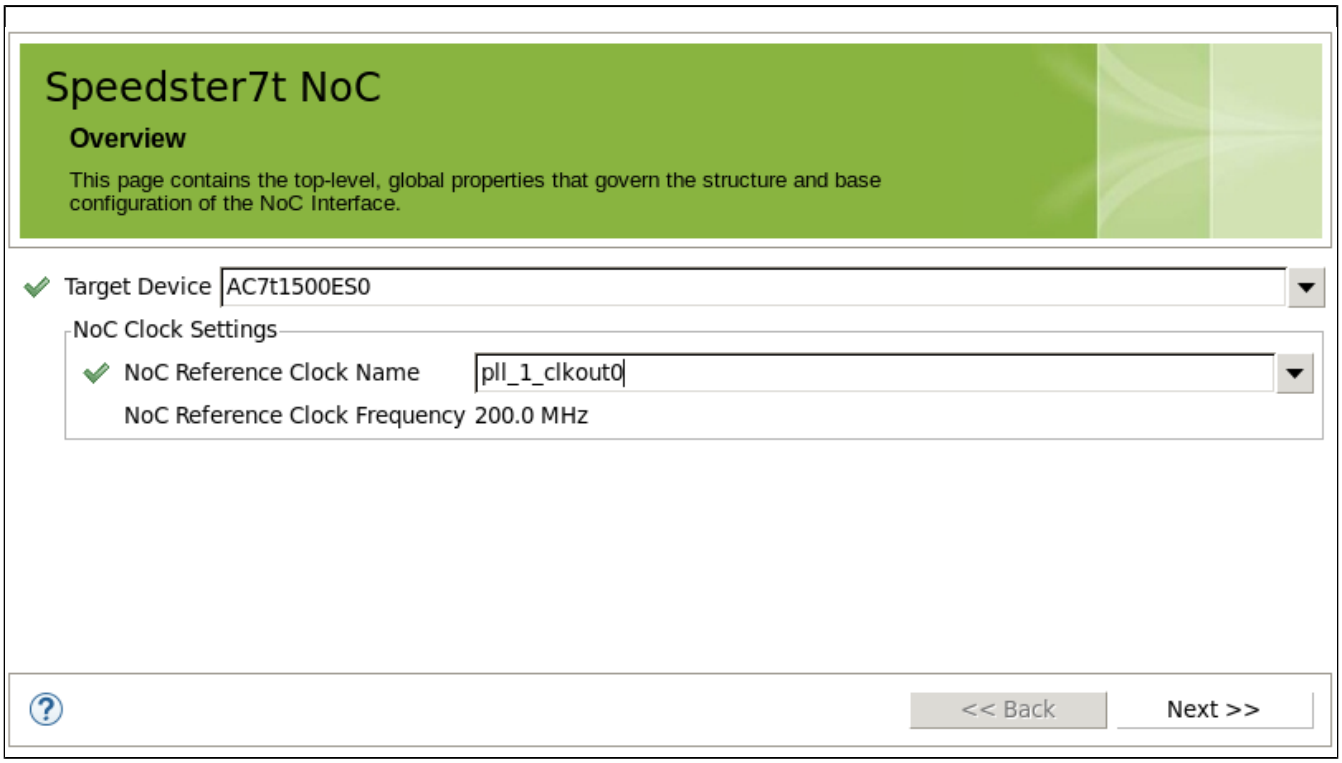

Figure 20: ***NoC Configuration of Clock***

This page in the IP Configuration GUI allows the user to enable or disable access to different endpoints per NoC row. For the AC7t1500 there are access controls for all eight rows. This dialog screen is where the user can turn on/off access to the entire GDDR6, DDR4, FCU, CSR spaces, plus PCIe 0, PCIe 1, or the entire NAP space for

that row of the NoC. In other words, for any NAP on that row, the NAP can only access the spaces that are checked for that row. Once all options are set, save the IP configuration as a `*.acxip` file and add it to the design project.

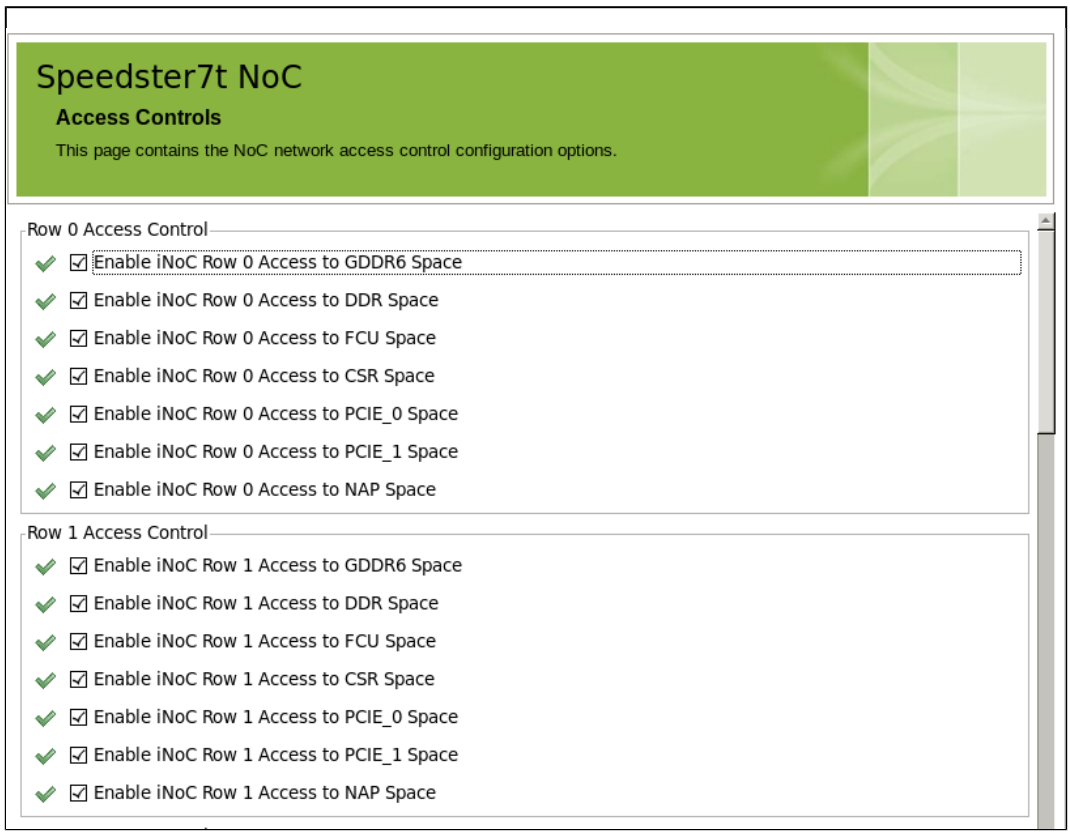

Figure 21: ***NoC Configuration Row Enable***

> **Caution**
>
> Generating output files such as a bitstream or simulation models is not supported in the current version of the I/O Designer Toolkit.

Revision History

Version	Date	Description
1.0	19 Sep 2019	• Initial Achronix release.