Creating PCI Express Links Using FPGAs

Course Agenda

- Building a Hard IP for PCI Express Design
- Simulating a Hard IP for PCI Express Design

Creating PCI Express Links Using FPGAs

Building a Hard IP for PCI Express Design

Building a Hard IP for PCI Express Design Section Agenda

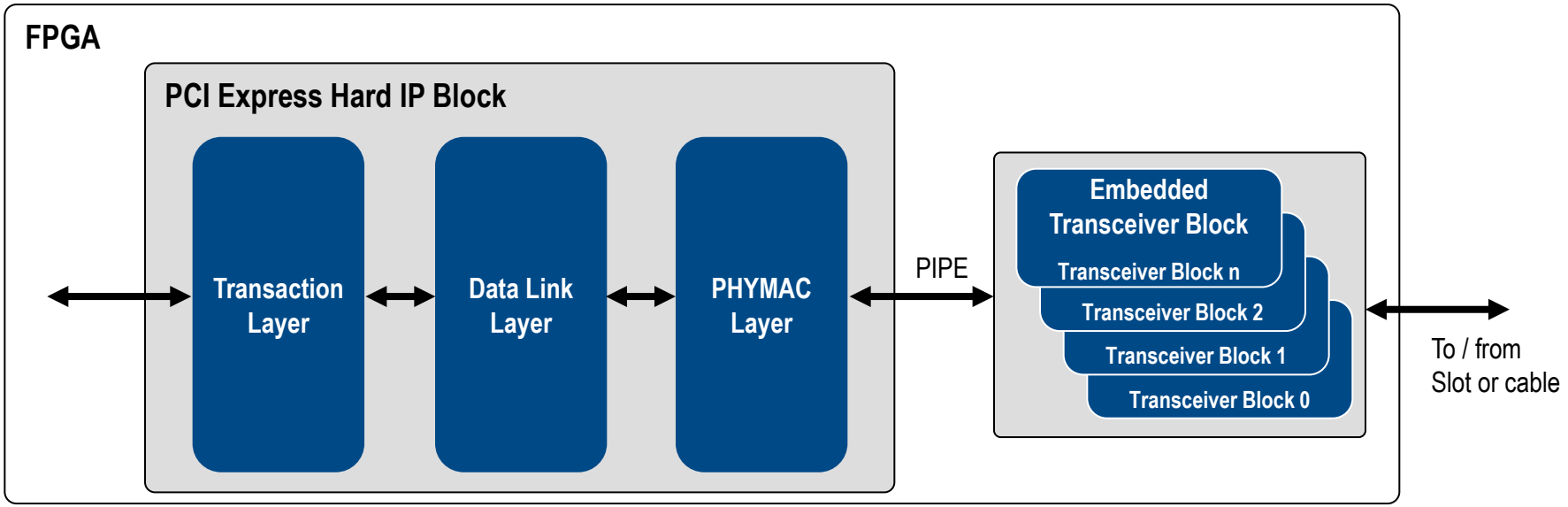
- Hard IP for PCI Express Features
- Customizing the Hard IP for PCI Express Settings
- Connecting the Hard IP for PCI Express Interfaces
- Design Implementation

Hard IP for PCI Express User Guides

- Referring to the User Guide is critical for a successful PCI Express design
- There are separate user guides for each device family, user interface and feature function:
  - Avalon-ST
    - *Arria® 10 Avalon-ST Interface for PCIe Solutions User Guide* (PDF)
    - *Stratix V Avalon-ST Interface for PCIe Solutions User Guide* (PDF)
    - *Arria V GZ Avalon-ST Interface for PCIe Solutions User Guide* (PDF)
    - *Arria V Avalon-ST Interface for PCIe Solutions User Guide* (PDF)
    - *Cyclone® V Avalon-ST Interface for PCIe Solutions User Guide* (PDF)
  - Avalon-MM
    - *Arria 10 Avalon-MM Interface for PCIe Solutions User Guide* (PDF)
    - *Stratix V Avalon-MM Interface for PCIe Solutions User Guide* (PDF)
    - *Arria V GZ Avalon-MM Interface for PCIe Solutions User Guide* (PDF)
    - *Arria V Avalon-MM Interface for PCIe Solutions User Guide* (PDF)
    - *Cyclone V Avalon-MM Interface for PCIe Solutions User Guide* (PDF)
  - Avalon-MM with DMA
    - *Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide* (PDF)
    - *V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide* (PDF)
  - Single Root I/O Virtualization
    - *Arria 10 Avalon-ST Interface with SR-IOV for PCIe Solutions User Guide (PDF)*
    - *Stratix V Avalon-ST Interface with SR-IOV for PCIe Solutions User Guide* (PDF)
  - CvP
    - *Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide* (PDF)

Hard IP for PCI Express

- Performs Transaction, Data Link, PHYMAC Layer and functionality
- Connects directly to FPGA embedded transceivers using internal PIPE interface for complete protocol stack solution
- Best solution for implementing PCIe interfaces in Altera FPGAs

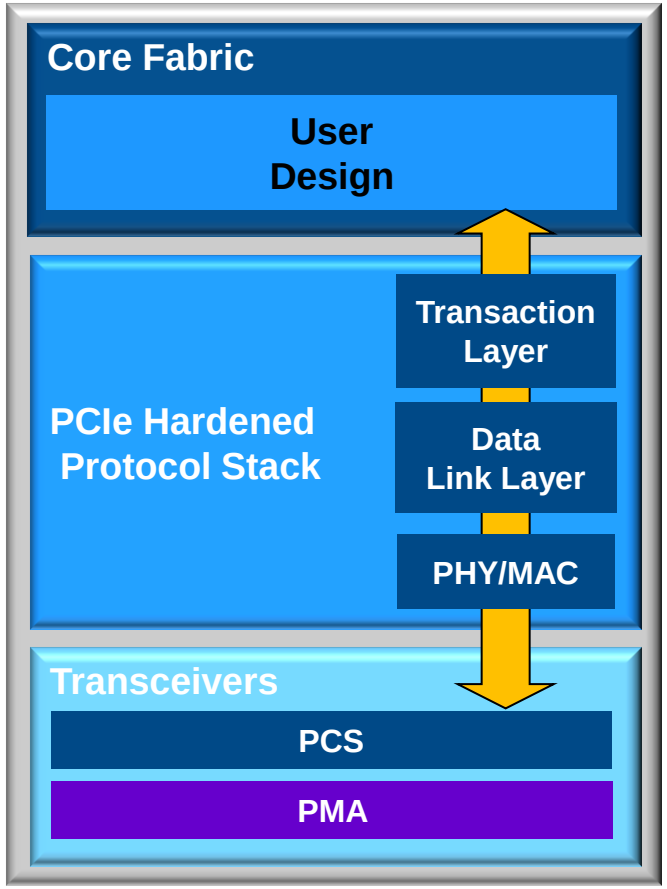


Hard IP for PCI Express Advantages

- Best-in-Class PCI Express performance IP solution
  - http://www.altera.com/b/best-in-class-pcie.html
  - Scatter-gather list based built-in direct memory access (DMA) engine
    - High-Performance DMA Reference Design
    - AN690 – PCIe DMA Reference Design for Stratix V Devices
  - Linux and Windows device drivers
- Low risk
  - Pre-verified PCI Express compliant core
  - Tested with industry-standard BFMs
  - Functional coverage with directed and random tests
  - PCI-SIG compliance hardware-tested with internal & external PHYs
- Variety of implementation options
  - x1, x2, x4, or x8 data widths
  - Gen1, Gen2 or Gen3
  - Root Port or Endpoint functionality
- Cost
  - $0 list price (Soft core requires a license fee)
  - ~0 LEs used, 0 core memory blocks

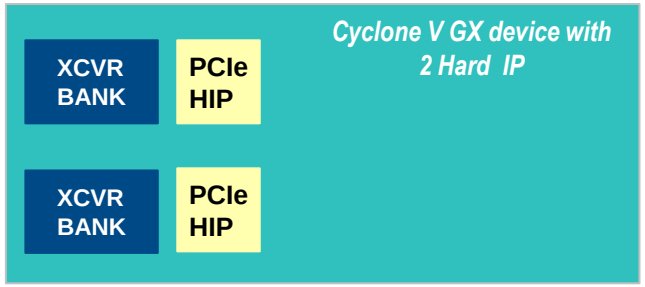
Benefits of Using Hard IP for PCIe

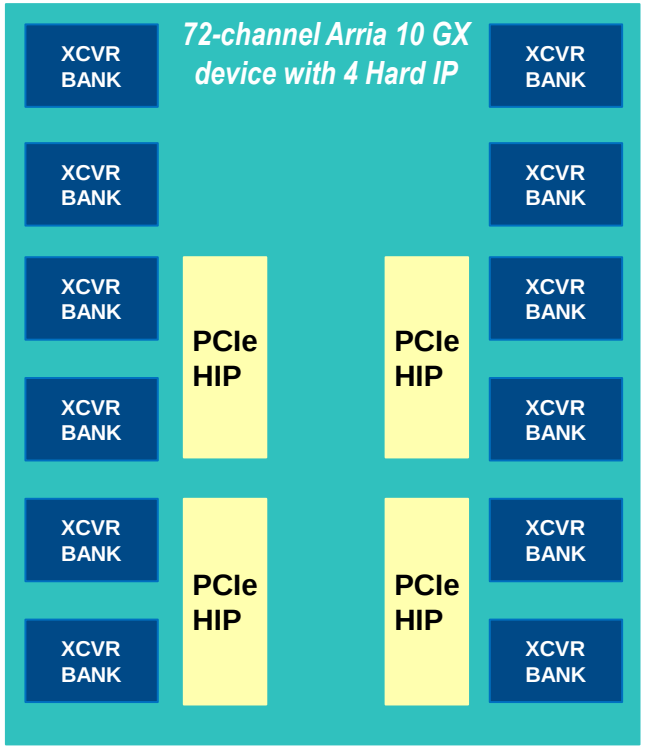
- Hardened PCIe Protocol Stack
  - Pre-verified
  - Guaranteed timing
- Reduced costs
  - Saves up to 100KLE's; Int Memory
  - Routing resources conserved
  - No license fees
- Variety of Implementation Options
  - x1, x2, x4, or x8 data widths
  - Gen1, Gen2 or Gen 3 data rates
  - Root Port or End Point functionality
- Easy to use parameter entry tool



Hard IP Availability

- Cyclone V GX/GT Devices
  - Up to 2 Hard IP (HIP) blocks on left side
- Arria V GX/GT/GZ Devices
  - Up to 2 Hard IP blocks (1 per side)
- Stratix V GX/GS/GT Devices
  - Up to 4 Hard IP blocks per FPGA (2 per side)
- Arria 10 SX/GX/GT Devices
  - Up to 4 Hard IP blocks per FPGA (2 per side)





Hard IP Features (1)

- Multiple configurations to support various data rates, link widths and use models
  - Avalon-ST configuration (datapath-centric)
  - Avalon-MM configuration (memory-mapped)
- Configurable maximum payload size
  - 128, 256, 512, 1024, or 2048 bytes
- Maximum number of Hard IP blocks
  - Cyclone V and Arria V devices : Up to 2 HIPs
  - Stratix V : Up to 4 HIPs
- One internal receive buffer per block
  - 6KB or 16KB
  - DPRAM (dual-port RAM) with configurable P/NP/CPL segment sizes
  - Store and forward behavior
- One internal retry buffer

Hard IP Features (2)

- Advanced Error Reporting (AER) support
  - Optional ECRC generation / check or forwarding
- Low-power mode using 62.5-MHz clock
  - Supported in Gen1 x1 only
- Multi-function Endpoint support
  - Single Hard IP can behave as if up to 8 unique endpoints (functions)
    - Cyclone V and Arria V (non-GZ) devices
  - Single-root I/O Virtualization (SR-IOV) allows up to 128 virtual functions (discussed later)
    - Stratix V devices
- System on a Chip (SoC) device support
  - Root Port example design available on rocketboards.org
    - Hardware and software code
    - Detailed instructions on getting setup and running

Hard IP Features (3)

- Extensive Interrupt support
  - Legacy PCI interrupt support
    - 1 (INTx) per Endpoint
    - 4 for Root Port
  - Up to 32 MSI
  - Up to 2048 MSI-X
- Power Management support
  - Responds to Power Management Event (PME) messages
  - Hard IP does not support Power Management states (L0s, L1, L2)
    - No FPGA power reduction in these modes
- Autonomous mode support
  - Hard IP released from reset before FPGA is configured
    - Hard IP can begin Link training
- Custom Configuration Space Registers (CSRs) support
  - Expand PCI Express functionality with unsupported, user-specific features

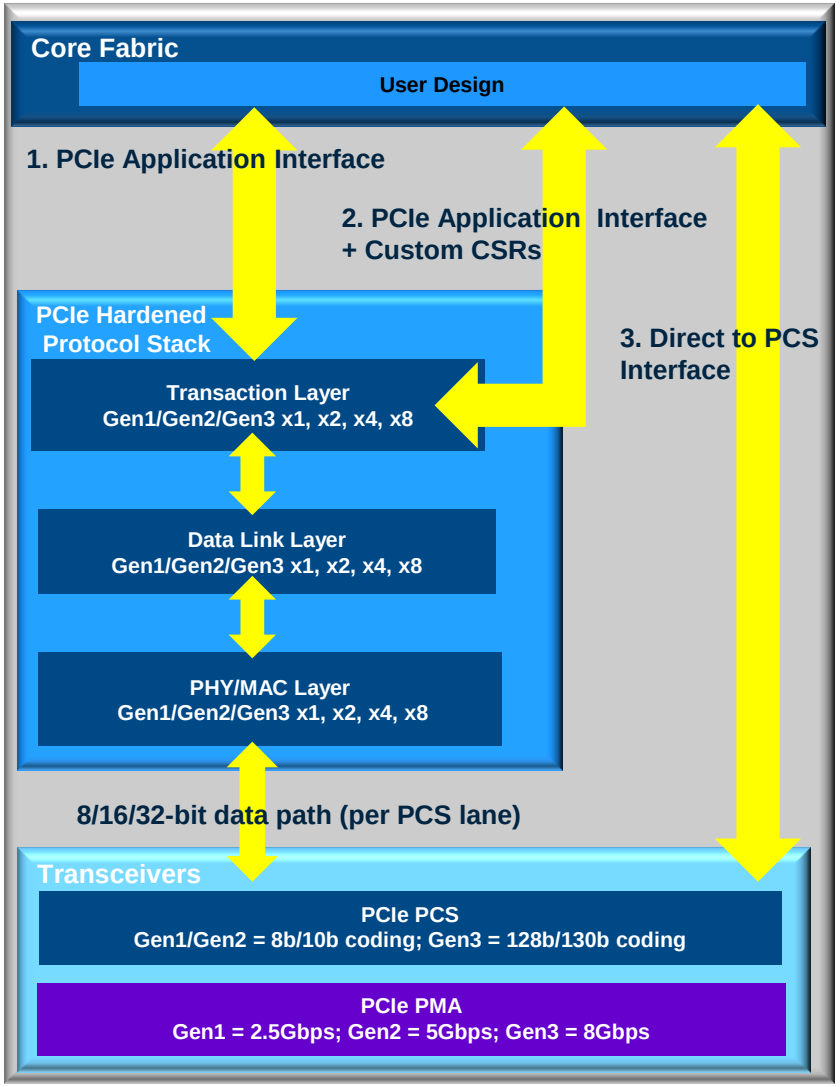
Arria 10 Hard IP Enhancements

- RX Credit Interface
  - Significant bandwidth improvement for read completions
  - Option for users to control the release of posted credits
    - Allows management of posted write storms
    - Makes number of outstanding read requests no longer limited by the size of the RX Completion buffer
- Enhanced RX buffer overflow protection
  - Prevents corruption of Rx Buffer when flow control rules are violated
- Latency reduction
  - Significant improvement for Gen1 and Gen2
- CvP supported in all operating modes
  - Including Gen3
- Altera Debug Master Endpoint (ADME) support added
  - Arria 10 devices: access to Native PHY, XCVR and other PLLs
  - User now has additional access points to perform test and debug

* *Support planned in a future version of the Quartus Prime software.*

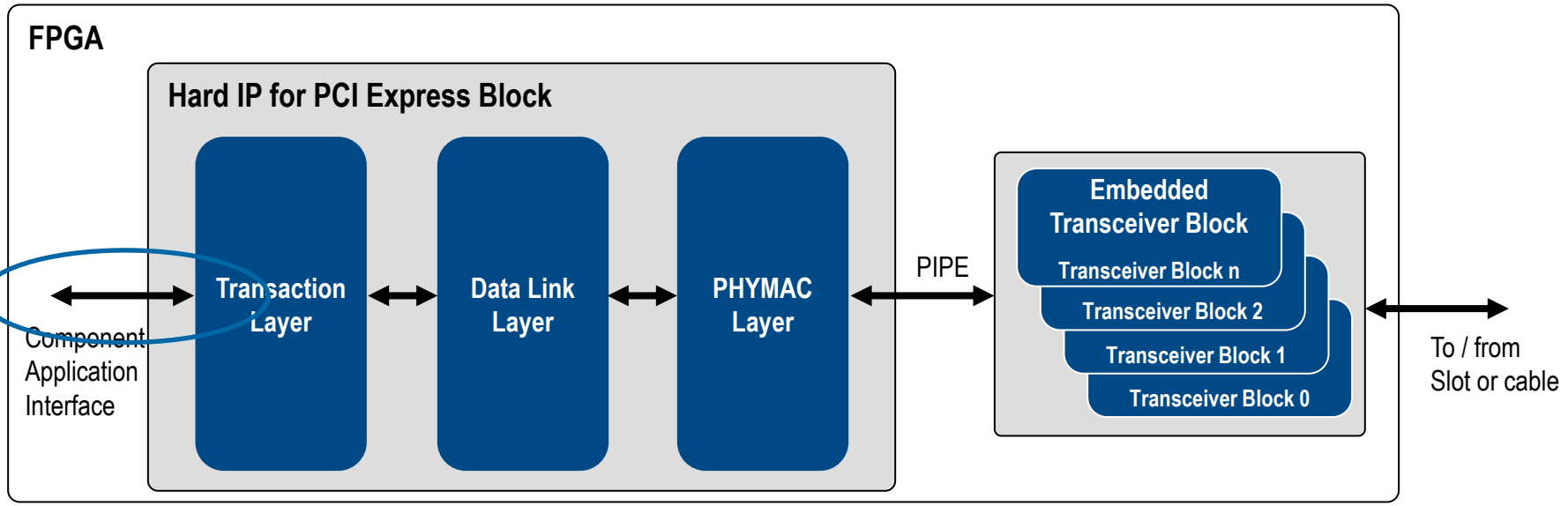
Generation 10 Hard IP Protocol Stack Solutions

- Flexible architecture allows for creation of PCIe solutions in one of three different ways
- PCIe Application Layer only
  - Typical use model
  - Users access Hard IP protocol stack via defined interfaces
- PCIe Application Layer + custom CSRs
  - Implement CSRs in FPGA fabric to add capabilities to the PCIe Application
- Complete soft IP protocol stack
  - Direct to PCS interface
  - Use soft IP built in FPGA fabric



Hard IP for PCI Express IP Core Variations

- Avalon-ST
- Avalon-ST with SR-IOV
- Avalon-MM
- Avalon-MM with DMA



HIP Overview - Component Interface Comparison

Avalon-ST interface

- Unidirectional, point to point
- **Advantage:** Maximize throughput
- **Advantage:** More PCI Express features supported
  - TLPs presented directly to/from application
  - Interrupts
  - Outstanding read requests
  - SR-IOV
  - All TLPs types supported
- Qsys and IP Catalog flow

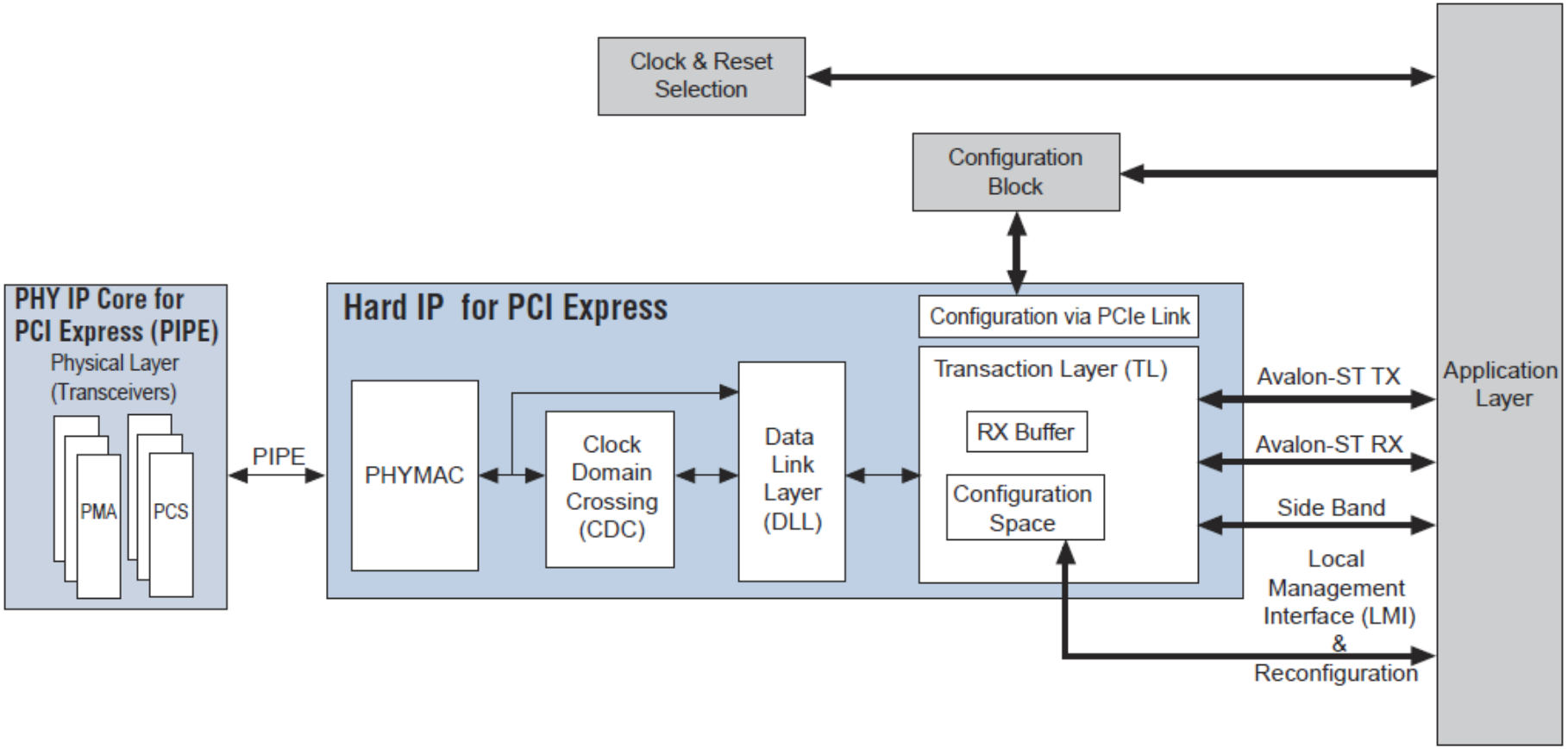
Avalon-MM interface

- Address mapped
- DMA Engine IP provides similar throughput to Avalon-ST interface core
- >90% of theoretical max throughput without DMA
- **Advantage:** ease of use
  - Reduces engineering effort required
  - PCIe packets converted from/into Avalon-MM transaction
  - Some TLPs not supported
  - Other complexities removed
- Qsys design flow only

Avalon-ST Interface Definition

- Targets data plane applications
- Unidirectional & point-to-point
  - Source interface sends data directly to a sink interface
- Source interface
  - launches data on rising edges of associated clock
- Sink interface
  - latches data on rising edges of associated clock
- Both interfaces may stall data flow when correct interface signals used
  - Must be ready to send/receive on each rising clock otherwise
- Data format/definition controlled by application or component
- See Avalon Interface Specification for more information

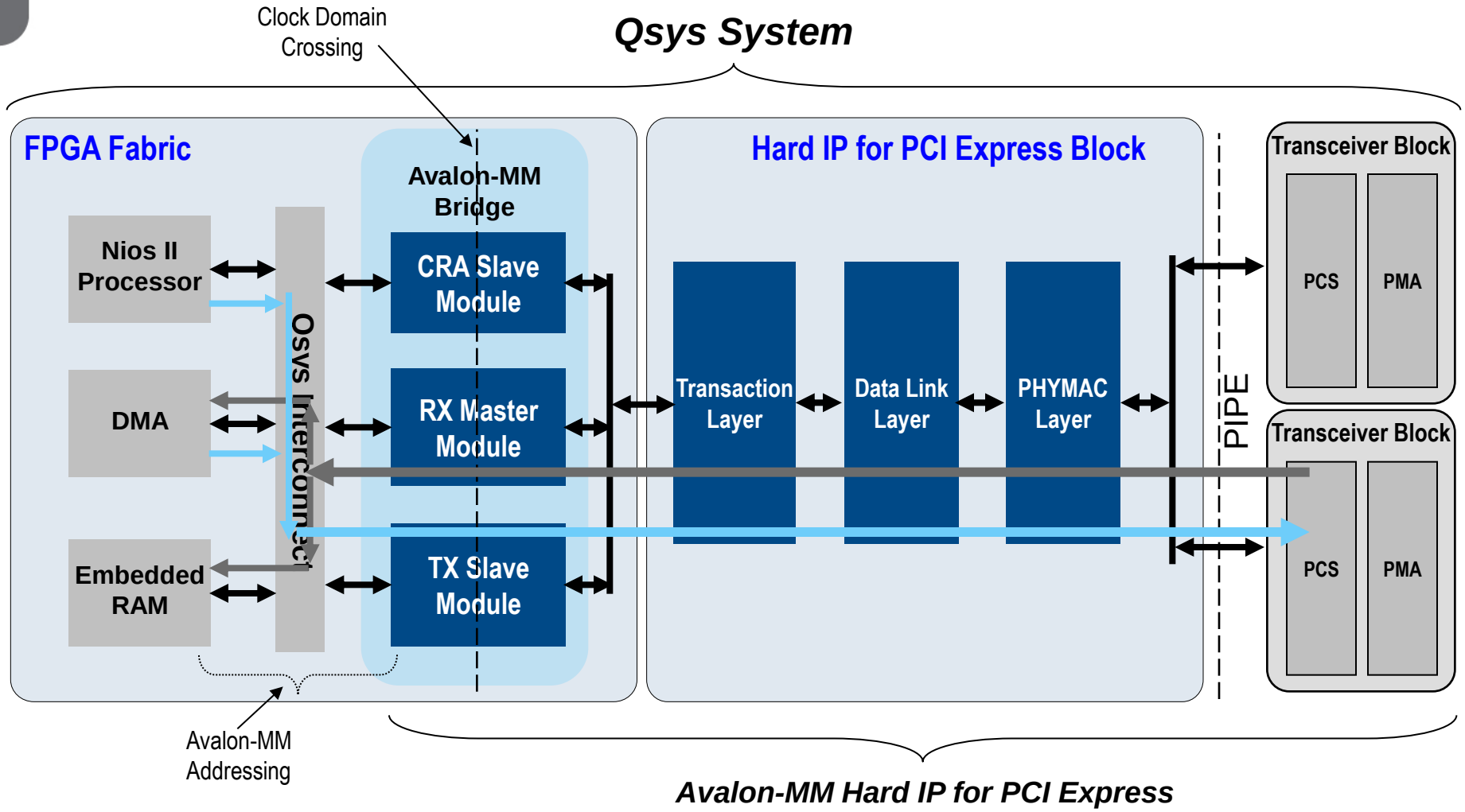
Avalon-ST Component – Block Diagram



Avalon-MM Interface Definition

- Address-based (memory-mapped) protocol
  - Components to communicate using read/write transfer requests
- Slave interfaces
  - Mapped to location(s) in address space
  - Respond to transfer requests from Qsys interconnect
- Master interfaces
  - Communicate with specific slave by issuing requests to slave address(es)
  - Initiate read/write transfers with Qsys interconnect targeting address in address space
- See Avalon Interface Specification for more information

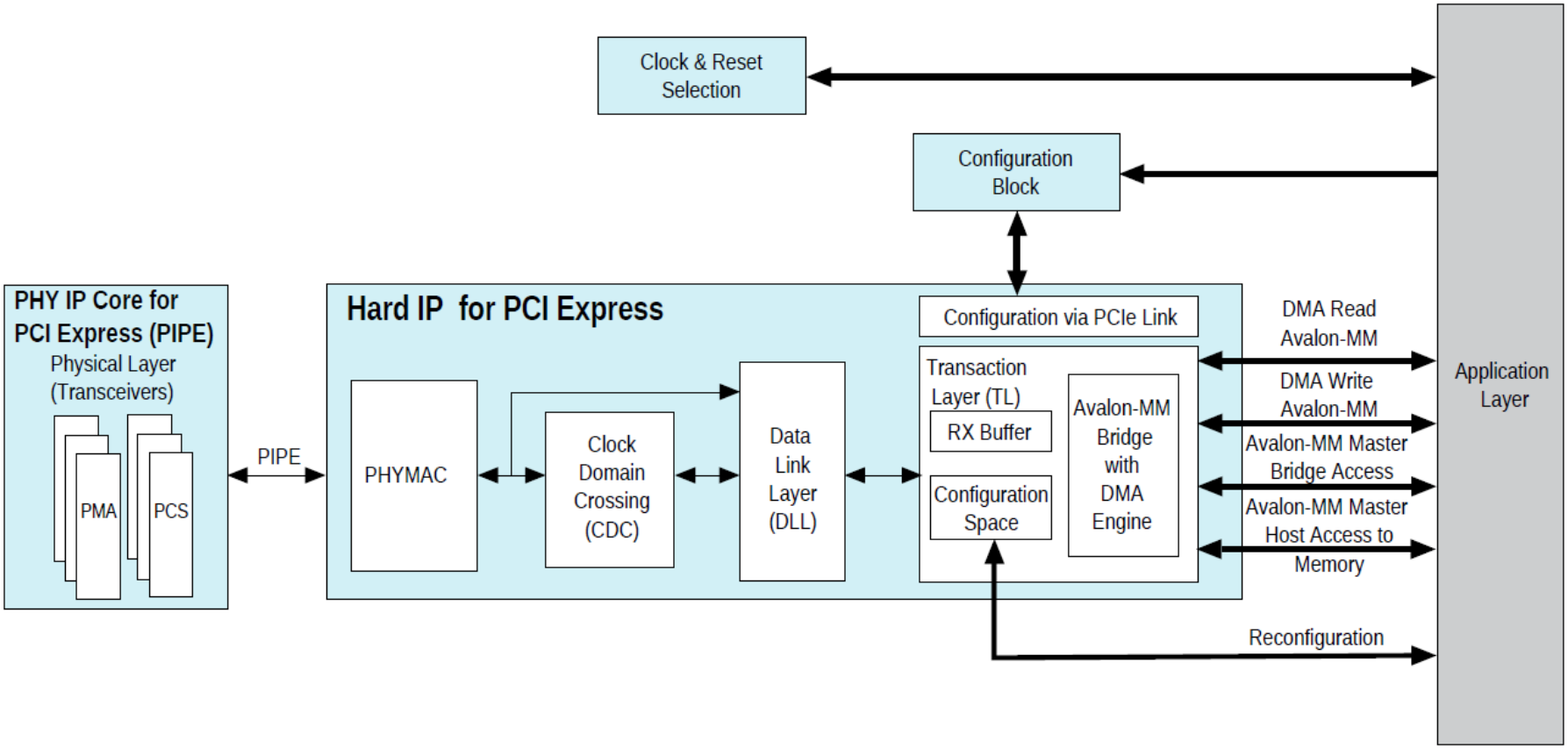
Avalon-MM Component – Block Diagram



Avalon-MM with DMA Engine

- > 6.4 GB/s DMA reads & writes for Generation 3 x8 configuration
- Added RX burst master
  - Provides high performance peer-to-peer communication
- Extended data path support to include 128-bit
  - Optimal for Gen 2 x8 & Gen 3 x4
- Linux & Windows drivers available to expedite evaluation & design-in
  - Altera Express Driver Development Kit (AXDDK) User Guide
  - Open source code available

Avalon-MM interface with DMA Block Diagram



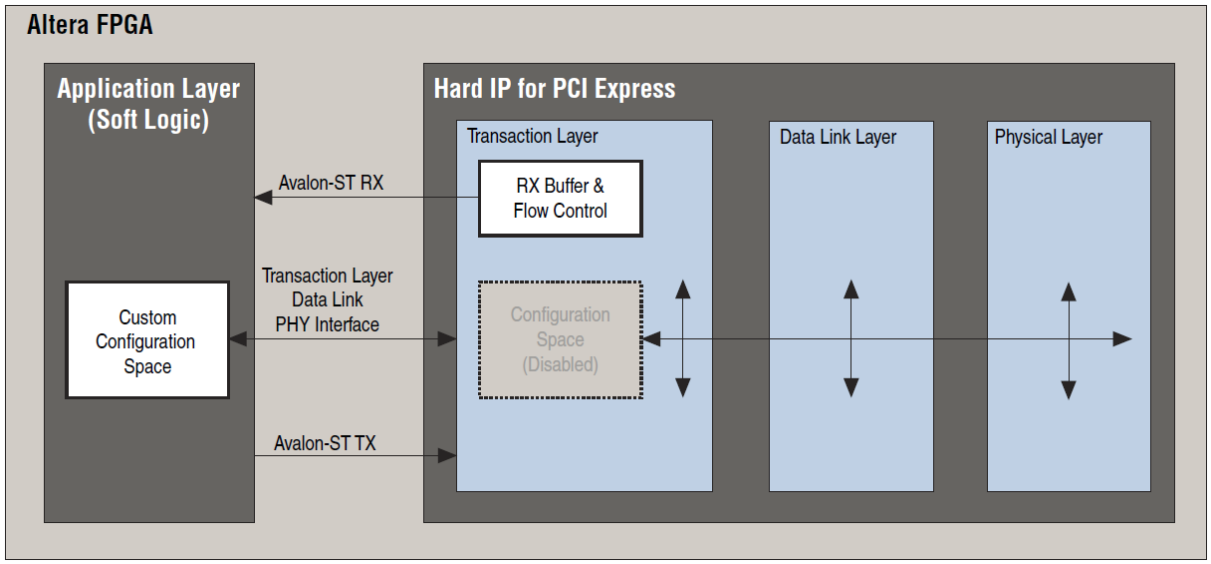
2. PCIe Application Layer + Custom CSRs

- Custom CSRs can be added to the Hard IP protocol stack using one of two modes
  - Configuration Shadow/Extension Bus (CSEB) Mode(1)
    - Custom (user-defined) Configuration Space Registers (CSRs) can be added into user logic
  - Configuration Space Bypass mode
    - Users implement complete custom Configuration Space within FPGA logic
- Allows designers to add functionality or features not supported by Hard IP
- Takes advantage of the hardened protocol stack without having to create/implement fully customized protocol stacks in FPGA soft logic
- Example functionality
  - Single-root I/O Virtualization (SR-IOV) EP
  - Transparent bridge or switch port
  - Multi-function EP
  - Optional protocol extensions

1. Support planned in a future version of the Quartus Prime software.

Configuration Bypass Mode

- Bypasses the Transaction Layer Configuration Space registers included as part of the hard IP
- Allows substitution with a custom Configuration Space implemented in soft logic as shown below



Configuration Bypass Mode (2)

- Functions retained in the Hard IP
  - RX Buffer, Flow Control, DL and PHY layers
  - Passing well formed TLPs to the Application Layer
  - Detecting and dropping malformed TLPs
- Functions disabled in the Hard IP
  - TLP BAR matching and completion tag matching
  - Power Management
  - MSI and legacy interrupts
  - Completion Errors
  - Configuration Interfaces
- Application Layer logic must
  - Implement disabled features
  - Detect and handle unsupported requests and unexpected completions
  - Generate all completions and messages

Single Root I/O Virtualization (SRIOV)

- One physical device that supports multiple physical functions and virtual functions (*next slide*)
- Advantages
  - System software
    - Allow each virtual machine (VM) to control a specific virtual function (VF)
    - Reduce arbitration bottleneck in software
    - Route packets via address mapping
  - Hardware
    - Allow multiple applications to share the same hardware resources
    - Save gate counts compared to multi-functions
- Disadvantages
  - Require new software driver

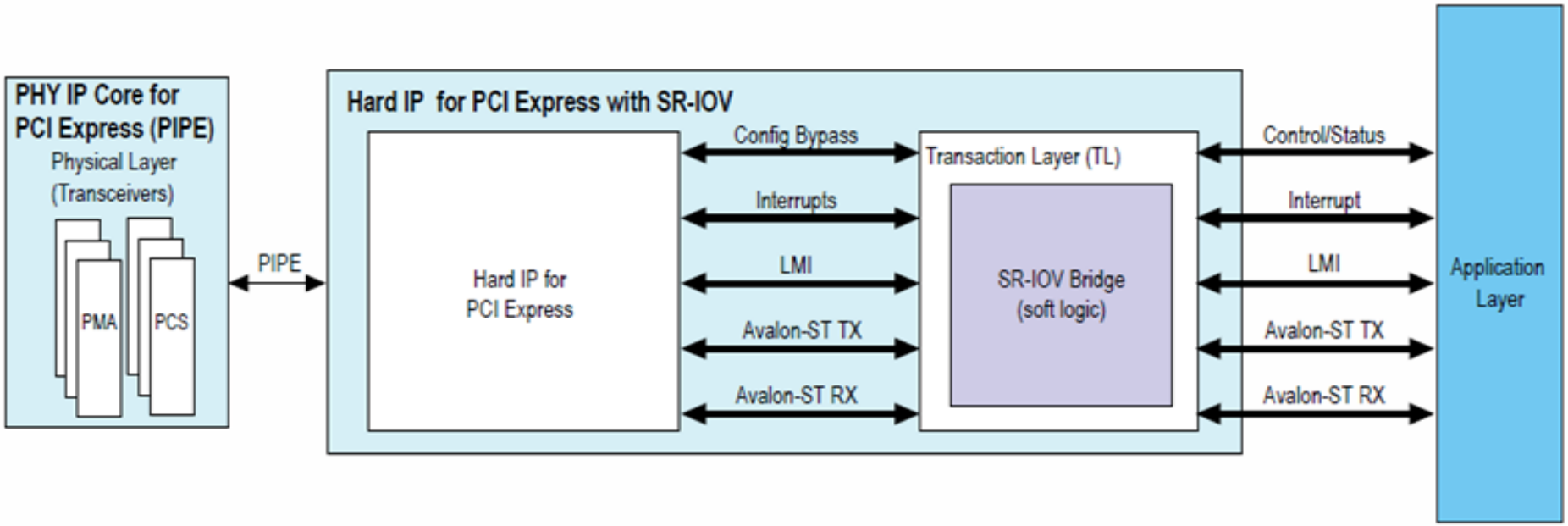
Physical Function vs. Virtual Function

- Physical Function (PF)
  - Is a “mission” function
  - Can be used for boot or work
  - Used for managing and configuring up to 128 Virtual Functions (next slide)
  - Errors CAN be attributed to this PF, but CANNOT be attributed directly to a particular Virtual Function within this physical function
- Virtual Function (VF)
  - Implemented through a PF’s capability registers
  - Each VF has unique configuration space
  - Minimum of 4 VFs required
  - All the VFs within a PF have the same size bars
    - BAR0 = BAR0, BAR1 = BAR1, but BAR0 does not have to equal BAR1
  - VFs use Alternative Requester ID (ARI) for routing ID
  - May have PCIe memory space, but not Legacy (I/O) space

Avalon-ST Interface with SR-IOV IP Core

- Stratix V & Arria 10 devices only
- Uses Configuration Bypass Mode
- Support
  - Gen2 and Gen3
  - x2, x4 and x8
  - 128-bit and 256-bit Avalon Streaming
- 2 Physical Functions (PF) / 128 Virtual Functions (VF)
- Advanced Error Reporting (AER) support
- New BAR hit signals
- New MSI-X interface
- Design examples available

Stratix V SR-IOV Block Diagram



3. Complete PCIe Soft IP Protocol Stack

- Allows for 3rd party or custom-made PCIe protocol stack
- Accesses transceiver PCS via PIPE (or Gen3 PIPE-like) interface
- Useful for designers who have specific protocol requirements not implemented in HIP
  - TLP Prefixes
    - Multi-root I/O virtualization (MR-IOV)
  - Custom monitoring and testing requirements

HIP Comparison and Summary (Cyclone V & Arria V)

Features	Cyclone V and Arria V (non GZ) Families		
	Avalon-ST Component	Avalon-MM Component	
		No DMA	w/DMA
Root Port support	Yes	Yes	No
Gen1 (Up to)	x1, x2, x4, x8*	x1, x2, x4, x8(1)	x8*
Gen2 (Up to)	x1, x2, x4	x1, x2, x4	x4
Max Payload Size	128-512 B	128–256 B	
TLP types	All	• Memory Read/Write • I/O Read/Write • Configuration Read/Write • Completions with or without data • Single-word Memory Read/Write	• Memory Read/ Write • Completions with or without data
64/128-bit Application Layer interface	Yes/Yes	Yes/Yes	Yes/No
62.5 MHz Input Clock	Yes	Yes	No
Num of tags (non-posted reqs)	32 or 64	16	
Multi-function support	Up to 8	Single	
ECRC Forwarding (TX/RX)	Yes	No	
Out of order completions	No	Yes	

1. Arria V family support only

HIP Comparison and Summary (Arria GZ & Stratix V)

Features	Arria V GZ and Stratix V Families		
	Avalon-ST Component	Avalon-MM Component	
		No DMA	w/DMA
Root Port support	Yes	Yes	No
Gen1 (Up to)	x1, x2, x4, x8	x1, x2, x4, x8	-
Gen2 (Up to)	x1, x2, x4, x8	x1, x2, x4, x8	x4, x8
Gen3 (Up to)	x1, x2, x4, x8	x1, x2, x4	x4, x8
Max Payload Size	128 B – 2 KB	128–256 B	128–512 B
TLP types	All	• Memory Read/Write • I/O Read/Write • Configuration Read/Write • Completions with or without data • Single-word Memory Read/Write	• Memory Read/ Write • Completions with or without data
64/128/256-bit Application Layer interface	Yes/Yes/Yes	Yes/Yes/No	Yes/No/Yes
62.5 MHz Input Clock	Yes	Yes	No
Num of tags (non-posted reqs)	256	8	16
Configuration Space Bypass/SR-IOV*	Yes	No	Yes
ECRC Forwarding (TX/RX)	Yes	No	
Out of order completions	No	Yes	

HIP Comparison and Summary (Arria 10)

Features	Arria 10 Families			
	Avalon-MM Component		Avalon-ST Component	
	No DMA	w/ DMA	No SR-IOV	w/ SR-IOV
Root Port support	Yes	No	Yes	No
Gen1	x1, x2, x4, x8	-	x1, x2, x4, x8	x8
Gen2	x1, x2, x4, x8	x4, x8	x1, x2, x4, x8	x4, x8
Gen3	x1, x2, x4	x2, x4, x8	x1, x2, x4, x8	x2, x4, x8
Max Payload Size	128 / 256 B	128 / 256 B	128 B – 2 KB	128 / 256 B
TLP types supported (for transmit)*	• Memory Read/Write • I/O Read/Write • Configuration Read/Write • Completion without Data • Single-word Memory Read/Write • Message Request (incl. with Data) • Completion (w/o Data)	• Memory Read/ Write • Completion with or without Data	• Same as Avalon-MM (no DMA) plus • Memory Read Lock Request • Completion with Data • Completion Lock with or without Data • Fetch and Add Atomic	• Same as Avalon-MM (no DMA) plus • Memory Read Lock Request • Completion wth Data • Completion Lock with or without Data
64/128/256-bit Application Layer interface	Yes/Yes/No	No/Yes/Yes	Yes/Yes/Yes	No/Yes/Yes
62.5 MHz Application Layer clock	Yes (Gen1 x1)	No	Yes (Gen1 x1)	No
Num of tags (non-posted reqs)	8	16	256	256
Configuration Space Bypass	No	No	Yes	Yes
ECRC forwarding (TX/RX)	No	No	Yes	Yes
Automatic out of order completion handling	Yes	Yes	No	No

Configuration via Protocol (CvP) using PCI Express

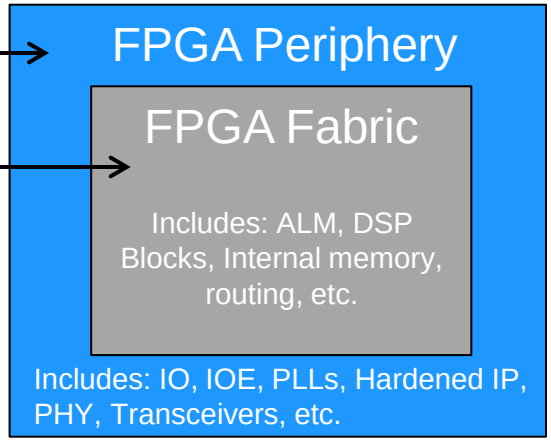
- A special configuration scheme using PCI Express protocol
- All V-Series and Arria 10 devices supported
- Creates separate images for the periphery and core logic
  - Guarantee PCIe link up within 100 ms specification
  - Allows reprogramming core image without requiring a system power down or disturbing the PCIe link
  - Improves security for the proprietary core bitstream by ensuring that only the PCIe host can access the FPGA core image
  - Reduces system costs by reducing the size of the flash device to store the periphery configuration data (larger core image may be stored in external memory)
- Two modes available
  - CvP Initialization Mode
  - CvP Update Mode
- Only available for Endpoint variants (not Root Port)

Autonomous PCIe Hardened IP

- Hardened PCIe IP blocks are “autonomous” from the FPGA fabric
  - Hardened IP is capable of link training prior to the FPGA fabric being configured & ensures L0 state is entered within 100 ms of reset
- Two steps to full FPGA configuration
  - “Periphery” configuration
    - Periphery always configured first (includes hardened PCIe IP block)
  - then “Fabric” configuration
    - Fabric configuration bits can be delivered via any standard programming method (FPP, PS, etc.) OR bits can be delivered across the PCIe bus

Step 1. FPGA Periphery is configured

Step 2. FPGA Fabric is configured



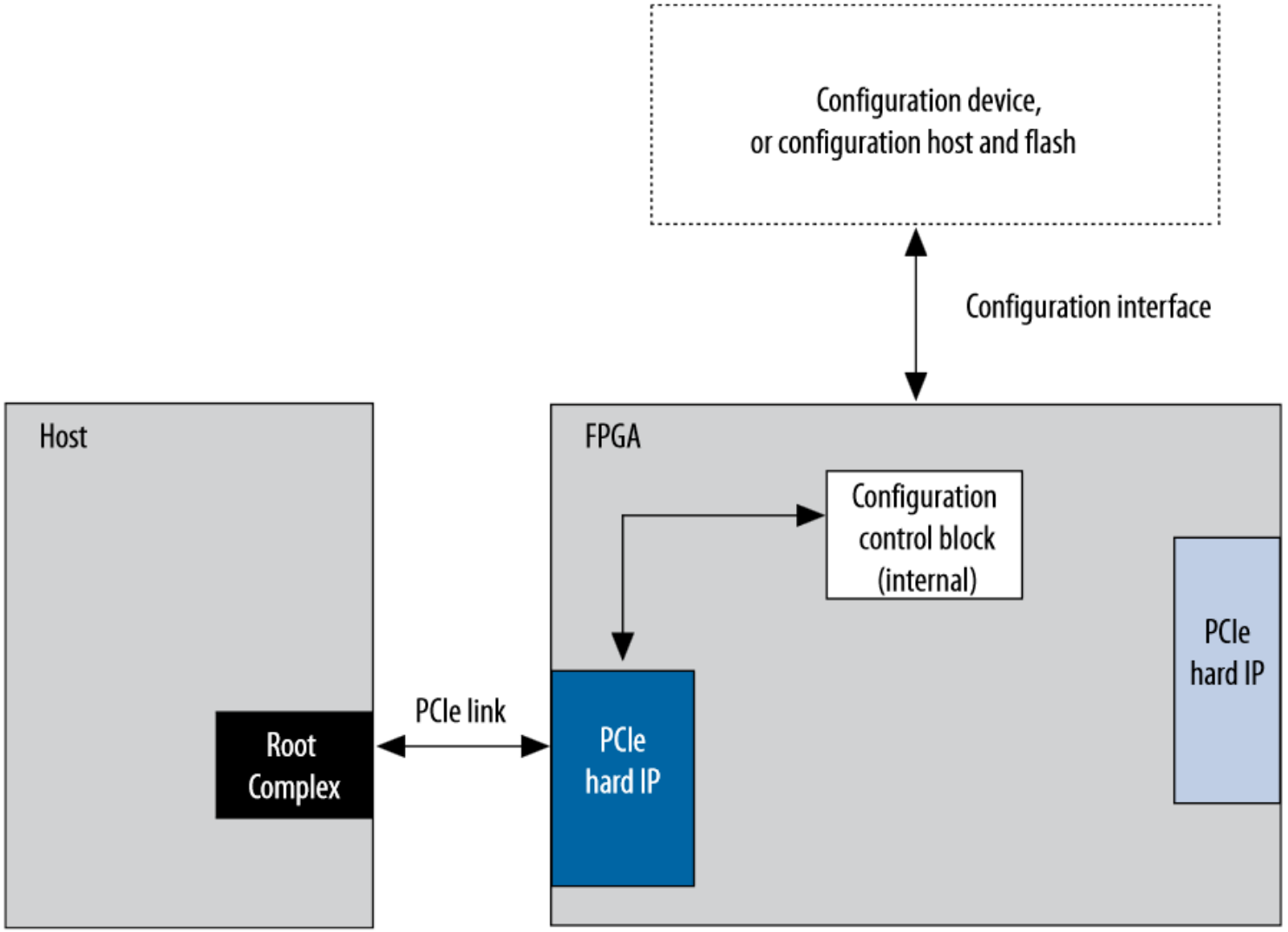
CvP Initialization Mode

- Leverages I/O connected to Flash / EEPROM for configuration bits to do initial programming at power-up
- CvP Init enables systems to communicate and access the hardened PCIe IP block without having the FPGA fabric programmed
  - Ensured via entering L0 (Link Active) within 100 ms window
- CvP Initialization functionality is identical across multiple device families
  - Cyclone V, Arria V, Stratix V, and Arria 10 devices

CvP Update Mode

- Leverages PCIe bus as a channel to move configuration bits to re-program the FPGA fabric to support different functionality
  - Different image used vs. CvP Init
  - Both images must have the same periphery setup
  - Occurs at some time later than right at power-up
- CvP Update is replaced by a simpler version of partial reconfiguration in the Arria 10 family (*discussed later in this section*)

CvP Block Diagram

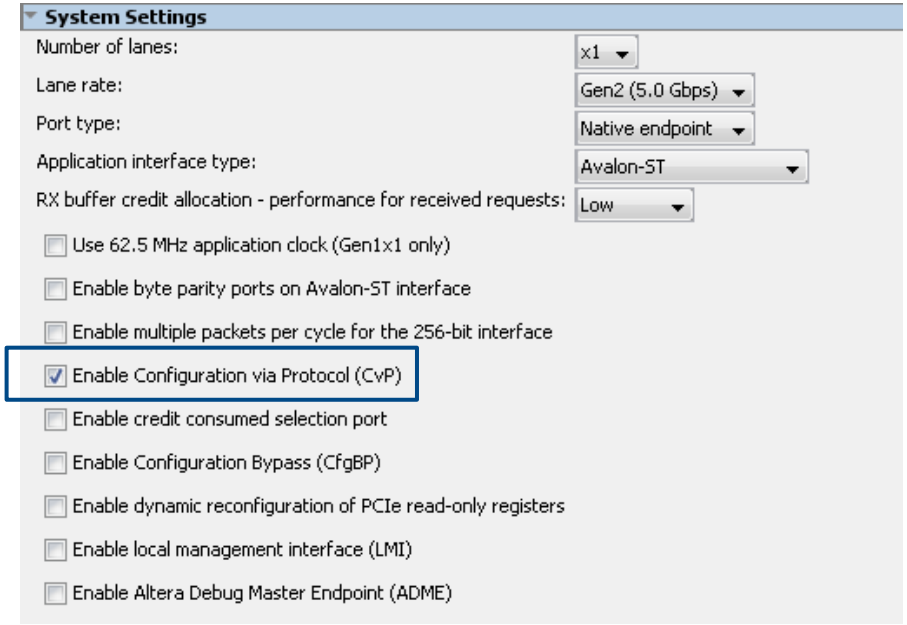


CvP Design Flow

1. Enable CvP in Hard IP parameter editor
2. Set Device and Pin Options for CvP
    1. Turn on "Auto-restart configuration after error”
    2. Select CvP mode
3. Split .SOF file into two separate programming images using Convert Programming File utility
    1. Small periphery image (type depends on configuration method)
    2. Larger core image (.RBF)

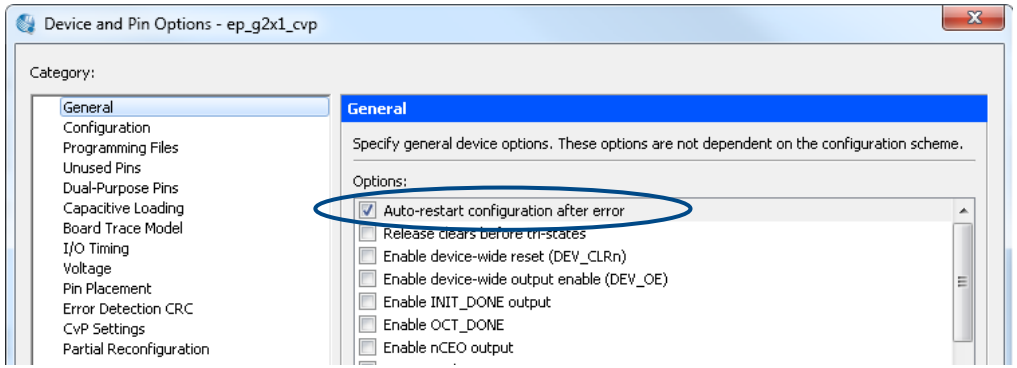
Enable CvP in Hard IP Parameter Editor

- Turn on "Enable Configuration via Protocol" in the System Settings section of the Hard IP parameter editor
  - Autonomous PCIe HIP mode enable automatically
  - Tells Fitter to place Hard IP in correct location

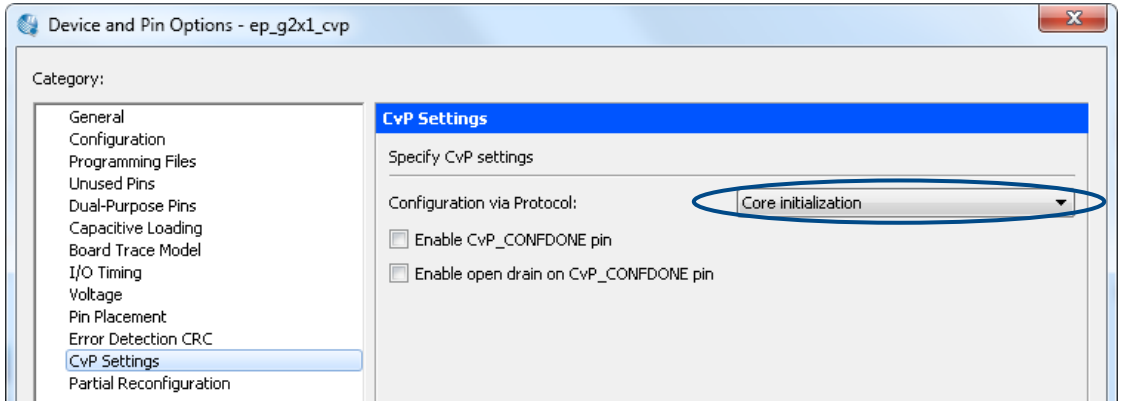


Set Device and Pin Options (Assignments menu → Device)

- Turn on "Auto-restart configuration after error"
  - Restart CvP if error detected

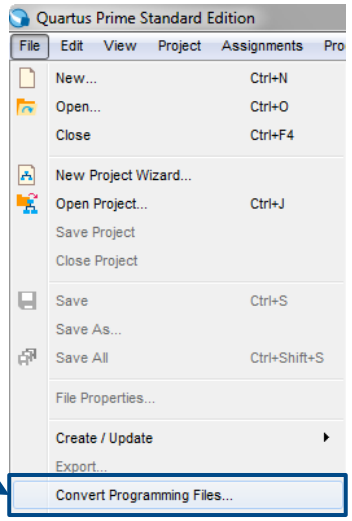


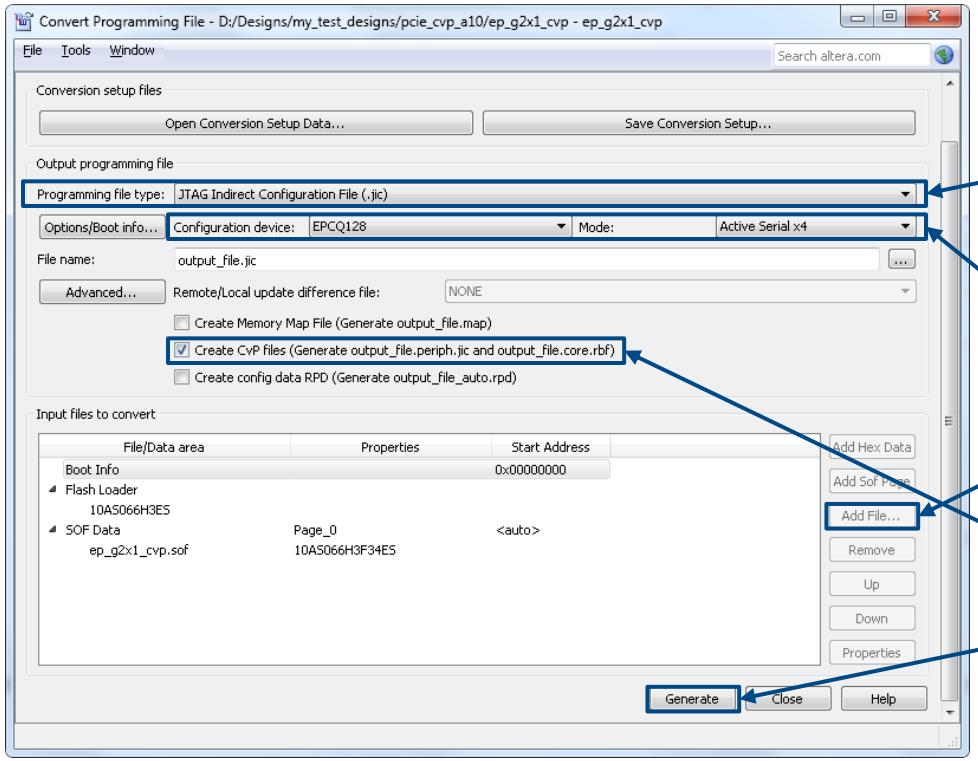
- Enable CvP mode



Split SOF Configuration File

1. Launch *Convert Programming Files* Utility





2. Select the configuration file type for periphery
3. Select configuration device and mode
4. Add SOF File
5. Enable Create CvP files
6. Click Generate

Partial Reconfiguration over Protocol (PRoP)(1)

- PRoP replaces CvP Update in Arria 10 family
- PRoP enables a user to redefine & update the FPGA 'fabric' to support a different functionality
- Simpler version of existing partial reconfiguration methodology
- PRoP provides more flexibility and robustness
  - Peripheral components aren't impacted
    - E.g. DDR maintains state / operation
  - XCVR and DDR re-calibrations are not necessary
- Similarities to CvP Update
  - Using PCIe bus as data conduit to move configuration bits into the FPGA
  - Writes configuration bits into the hardened PCIe IP block

1. Beta Feature in Quartus Prime version 15.1
Contact an Altera sales representative for further information.

CvP and PRoP Summary

- CvP Initialization (CvP Init)
  - Refers to the FPGA periphery configuration bits from a flash / eeprom interface to complete initial programming at power-up only
    - Initialization means this is the first time the FPGA will go into user mode since power was applied to the system
- CvP Update (28 nm devices only: Cyclone V, Arria V, Stratix V)
  - Refers to the FPGA fabric configuration bits being sent across the PCIe bus to update the FPGA fabric with new functionality (different image)
    - Periphery must remain the same between the FPGA images
  - Not supported on Arria 10
- Partial Reconfiguration over Protocol (PRoP)
  - Updating the FPGA fabric while the device remains in user mode
  - New configuration scheme provides higher flexibility to precisely hone into where RTL changes are needed

CvP Help?

- See online training Configuring Altera FPGAs for more details on
  - Using the Convert Programming Files utility
  - Understanding various configuration methods and file types for each
- See the Configuration via Protocol landing page for
  - CvP User Guide
    - CvP topologies, software, design considerations and example designs
    - CvP debugging check list
  - Open source driver code for Linux

Building a Hard IP for PCI Express Design Section Agenda

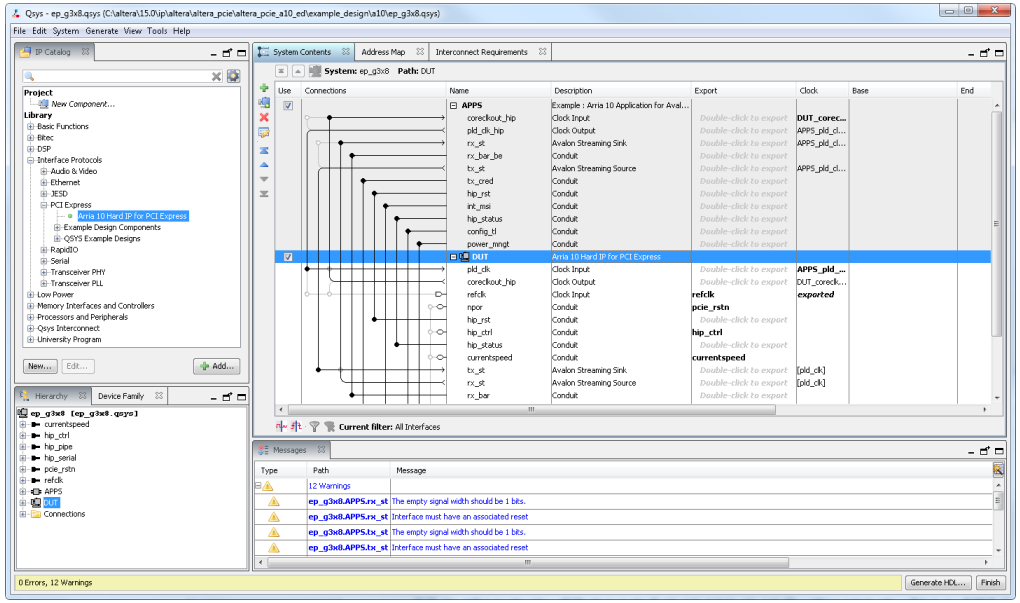
- Hard IP for PCI Express Features
- Customizing the Hard IP for PCI Express
- Connecting the Hard IP for PCI Express
- Design Implementation

Hard IP for PCI Express Design Flows

- Qsys and IP Catalog flows
  - Similar PCI Express features
  - Different component and interconnect options
- Qsys flow
  - Both Avalon-ST and Avalon-MM components
  - Graphical Qsys interconnect
- IP Catalog flow
  - Avalon-ST components only
  - Hardware Description Language (HDL) interconnect

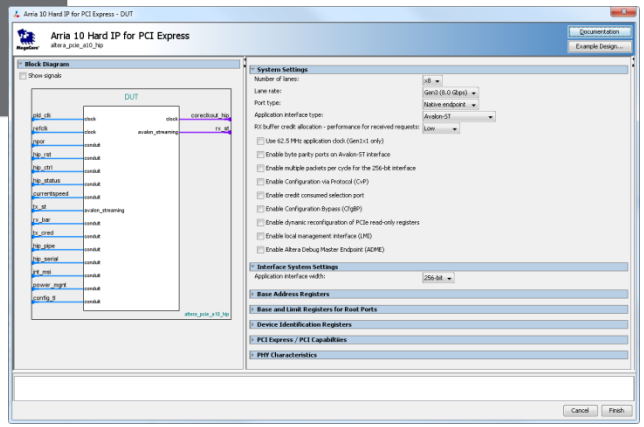
Qsys System Design Tool

- Describes system in design blocks
- Automates IP Block connectivity
- Supports multiple memory-mapped and streaming-type interfaces
- Connects both standard and custom IP blocks
- Supports hierarchical system design

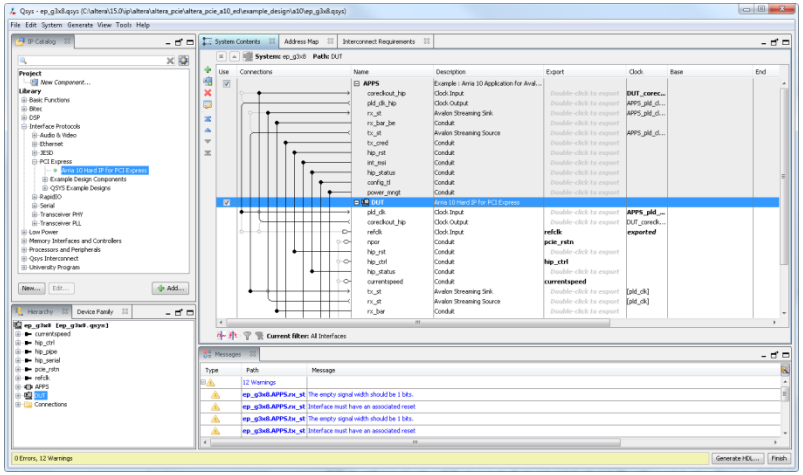


Reduce Development Time by Weeks!

Qsys Design Flow



2. Hard IP for PCI Express parameter editor gets called from Qsys to configure Hard IP block and add to system



1. Open Qsys from the Quartus Prime software to build system

Quartus Prime Design Software

3. Qsys generates HDL that is added to Quartus II project for compilation into FPGA

Note: This material will focus on the Qsys flow because it supports both Avalon-MM and Avalon-ST interfaces

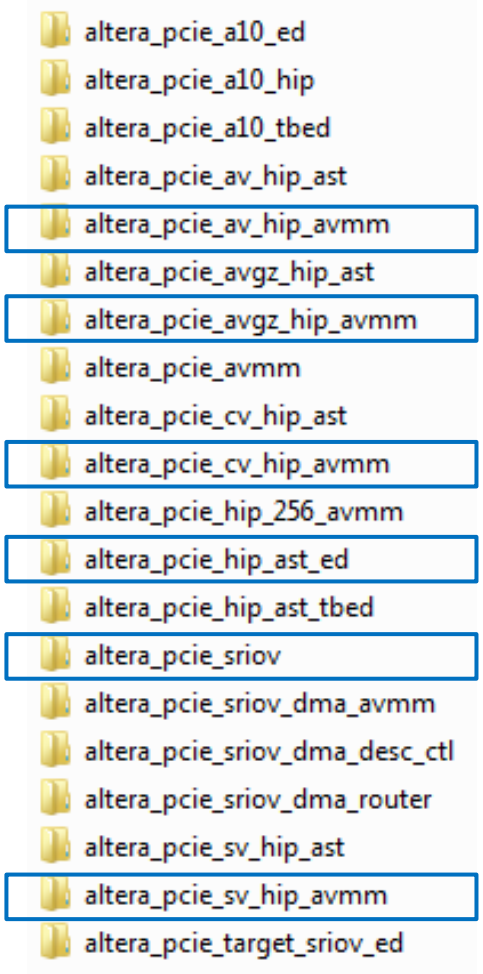
Qsys System Design Flow

1. Create Qsys system
2. Generate HDL files for synthesis in Generate window
   - Placed by default into **synth** subdirectory
3. Set top-level Quartus Prime project entity
   - Instantiate Qsys system in another design file or specify Qsys system name as top-level design entity
4. Add generated .qip (Quartus Prime IP) file to Quartus Prime project
   - Adds all required generated HDL files
5. Constrain design
   - Hard IP requires only `derive_pll_clocks -create_base_clocks` SDC command
6. Compile design

- For more information on using Qsys, see the following online training
  - Introduction to Qsys
  - Advanced System Design Using Qsys

Starting with Pre-Built Hard IP Example Designs

- Navigate to <installation_directory>/ip/altera/altera_pcie
  - Organized in folders by device family and Hard IP core version/interface type
    - Avalon MM interface vs. Avalon-ST interface
    - SR-IOV support
    - With or without built-in DMA
    - 256-bit wide Application Layer interface
- Copy .QSYS file into project directory
- Customize Hard IP instance per system requirements
- Modify .QSYS file
  - e.g. Adding or substituting in user Application Layer logic

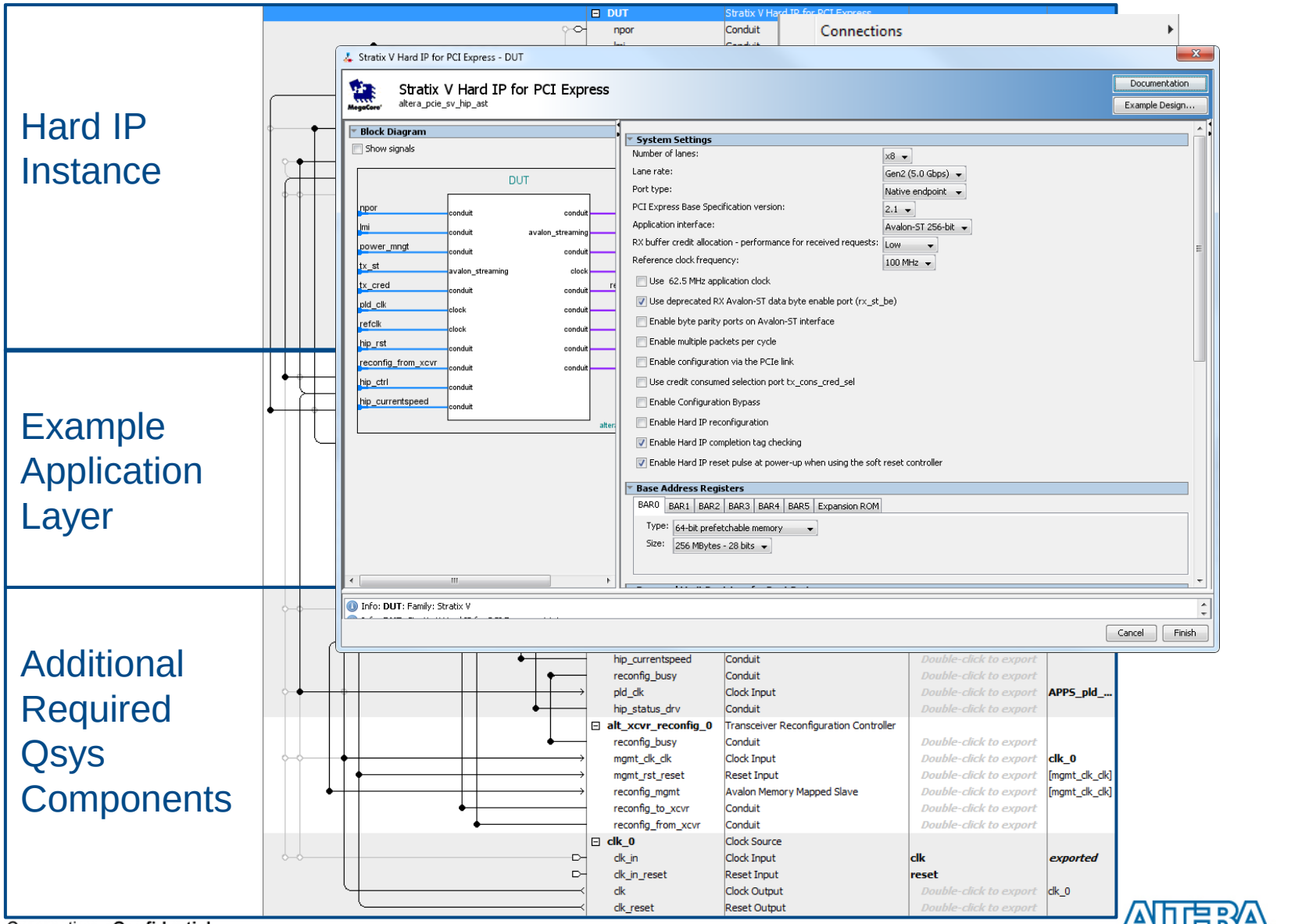


Hard IP Example Design

Hard IP Instance

Example Application Layer

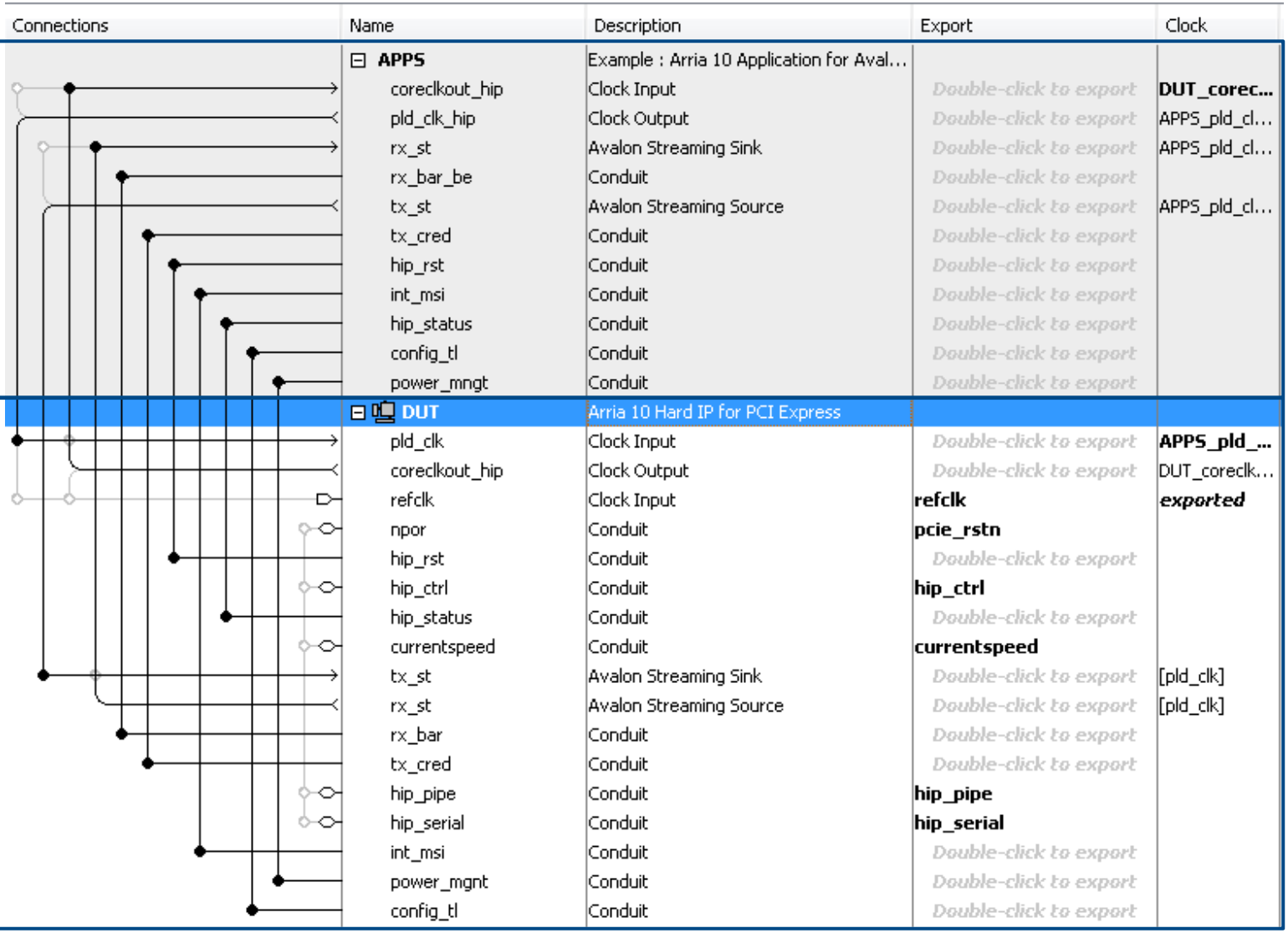
Additional Required Qsys Components



Hard IP Example Design - Avalon-ST Interface Core

Example Application Layer

Hard IP Instance

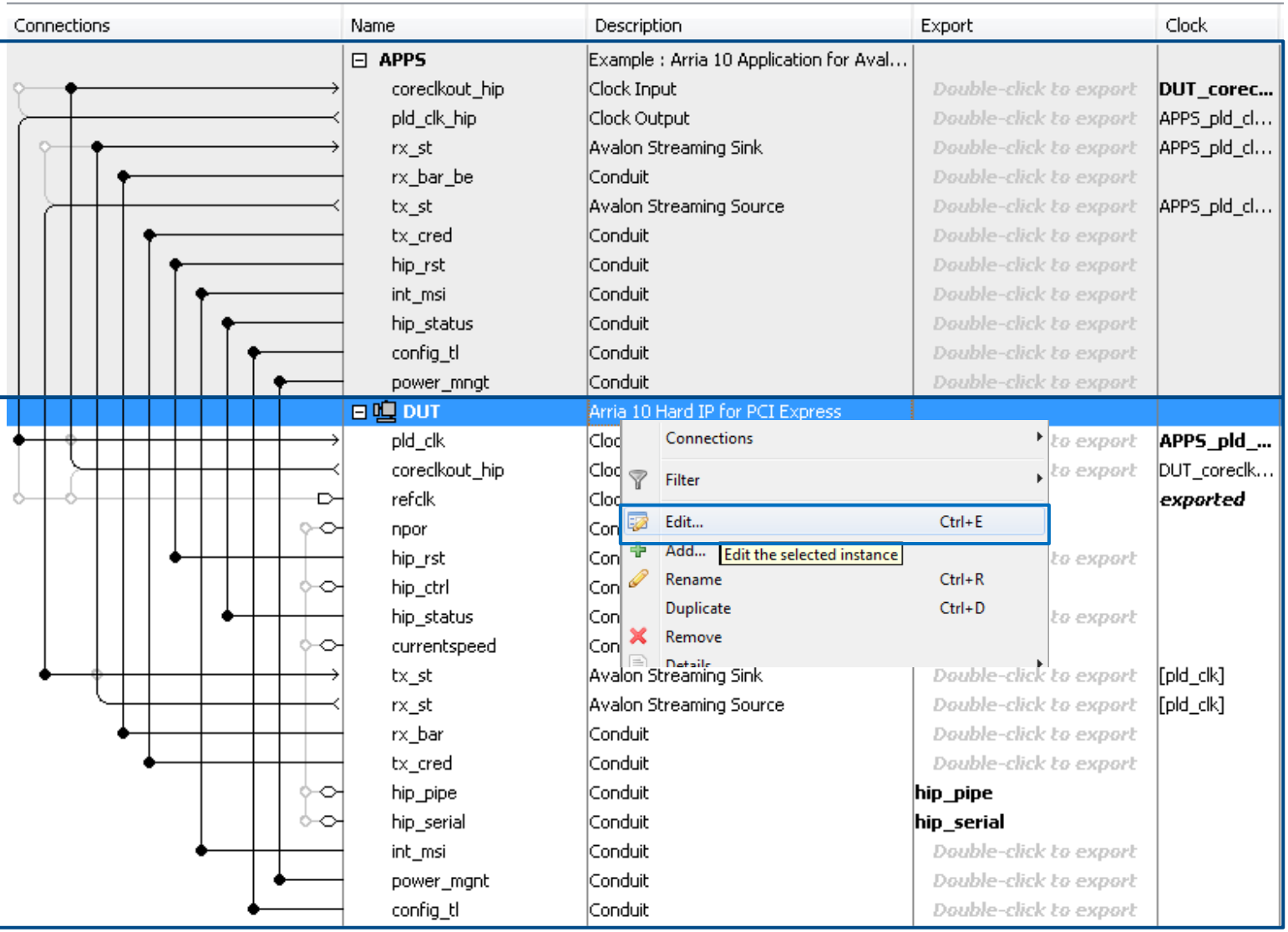


Connections	Name	Description	Export	Clock
	APPS	Example : Arria 10 Application for Aval...		
	coreclkout_hip	Clock Input	Double-click to export	**DUT_corec...**
	pld_clk_hip	Clock Output	Double-click to export	APPS_pld_cl...
	rx_st	Avalon Streaming Sink	Double-click to export	APPS_pld_cl...
	rx_bar_be	Conduit	Double-click to export	
	tx_st	Avalon Streaming Source	Double-click to export	APPS_pld_cl...
	tx_cred	Conduit	Double-click to export	
	hip_rst	Conduit	Double-click to export	
	int_msi	Conduit	Double-click to export	
	hip_status	Conduit	Double-click to export	
	config_tl	Conduit	Double-click to export	
	power_mngt	Conduit	Double-click to export	
	DUT	Arria 10 Hard IP for PCI Express		
	pld_clk	Clock Input	Double-click to export	**APPS_pld_...**
	coreclkout_hip	Clock Output	Double-click to export	DUT_coreclk...
	refclk	Clock Input	**refclk**	***exported***
	npor	Conduit	**pcie_rstn**	
	hip_rst	Conduit	Double-click to export	
	hip_ctrl	Conduit	**hip_ctrl**	
	hip_status	Conduit	Double-click to export	
	currentspeed	Conduit	**currentspeed**	
	tx_st	Avalon Streaming Sink	Double-click to export	[pld_clk]
	rx_st	Avalon Streaming Source	Double-click to export	[pld_clk]
	rx_bar	Conduit	Double-click to export	
	tx_cred	Conduit	Double-click to export	
	hip_pipe	Conduit	**hip_pipe**	
	hip_serial	Conduit	**hip_serial**	
	int_msi	Conduit	Double-click to export	
	power_mgnt	Conduit	Double-click to export	
	config_tl	Conduit	Double-click to export	

Hard IP Example Design – Arria 10 Avalon-ST Core

Example Application Layer

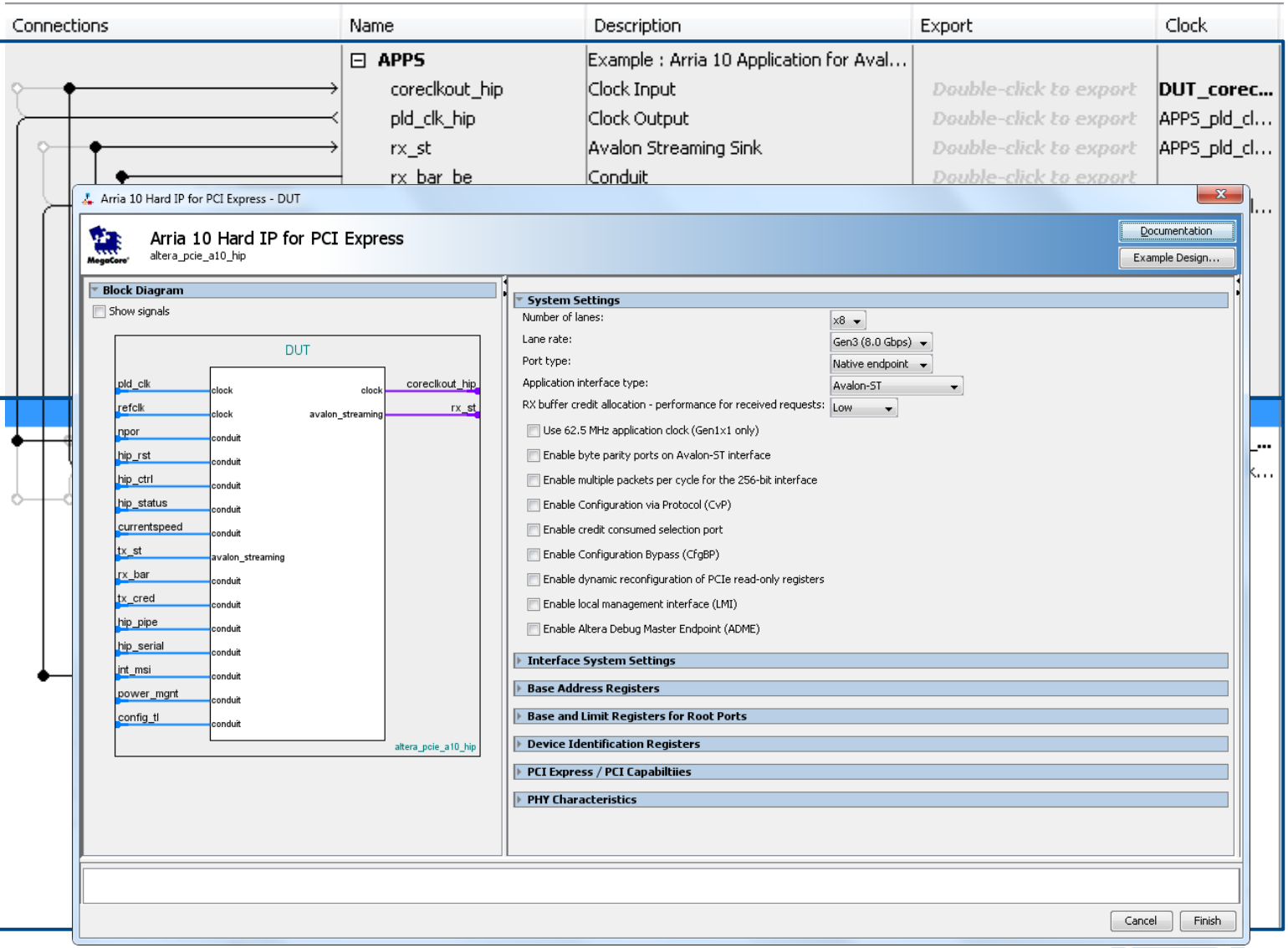
Hard IP Instance



Hard IP Example Design – Arria 10 Avalon-ST Core

Example Application Layer

Hard IP Instance

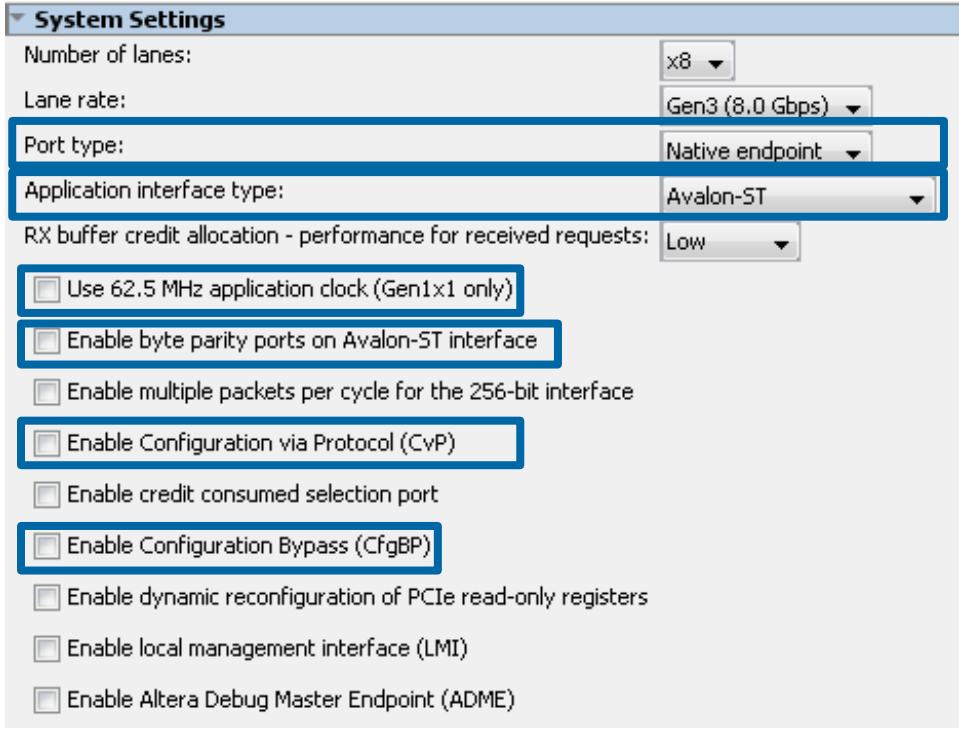


Customizing the Hard IP Using the Parameter Editor

- System Settings
- Interface System Settings
- SR-IOV System Settings
- Base Address Registers or SR-IOV Base Address Settings
- Base and Limit Registers for Root Ports
- Device ID Registers
- Device Capabilities
- Error Reporting
- Link Capabilities
- MSI/MSI-X
- Slot Capabilities
- Power Management

Customizing the HIP – System Settings

- Port type
  - Native Endpoint or Root Port
- RX Buffer Credit allocation(1)
  - Based on expected RX traffic
  - Allocates RX credits between requests and completions
    - Lower setting for receiving more read completions
    - Higher setting for receiving more read requests
- Configuration via Protocol
  - Enables FPGA CvP



Note:
Avalon-ST component settings shown above
Avalon-MM component has a subset of settings

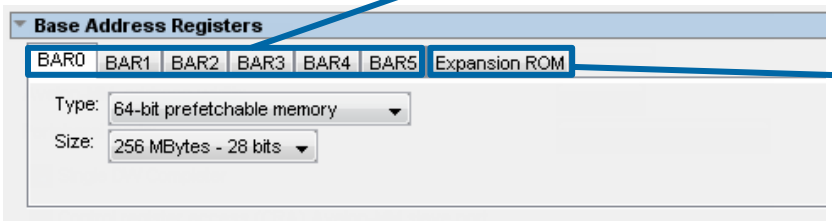
(1) The actual RX buffer allocation (%) appears in the Message Window of the parameter editor.

Customizing the HIP – Avalon-ST Interface System Settings

- Select the width of the Application Layer data interface
- Choose between
  - 64 bits wide
  - 128 bits wide
  - 256 bits wide

Customizing the HIP – Base Address Registers

- BARx
  - Define type and size
  - 64-bit prefetchable (Uses up 2 BARs)
  - 32-bit non/prefetchable
- Expansion ROM
  - Up to 16 Mbytes
  - Not available for the Avalon-MM component

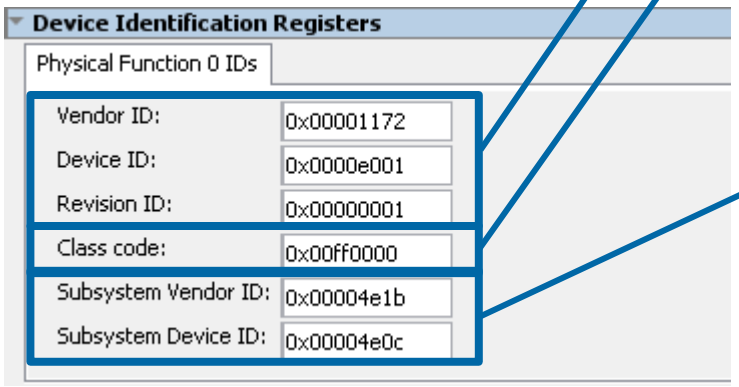


Type 0 Configuration Header

Byte			
3	2	1	0
Device ID		Vendor ID	
Status Register		Command Register	
Class Code			Revision ID
BIST	Type	Latency Timer	Cache Line Size
Base Address 0			
Base Address 1			
Base Address 2			
Base Address 3			
Base Address 4			
Base Address 5			
CardBus CIS Pointer			
Subsystem ID		Subsystem Vendor ID	
Expansion ROM Base Address			
Reserved			Capabilities Pointer
Reserved			
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

Customizing the HIP – Device Identification Registers

- Vendor ID
  - Identifies the device vendor
  - Allocated by PCI-SIG
- Device ID
  - Identifies the device model
  - Assigned by vendor
- Subsystem Vendor ID
  - Usually assigned by the chip manufacturer
- Subsystem Device ID
  - Usually assigned by the board manufacturer

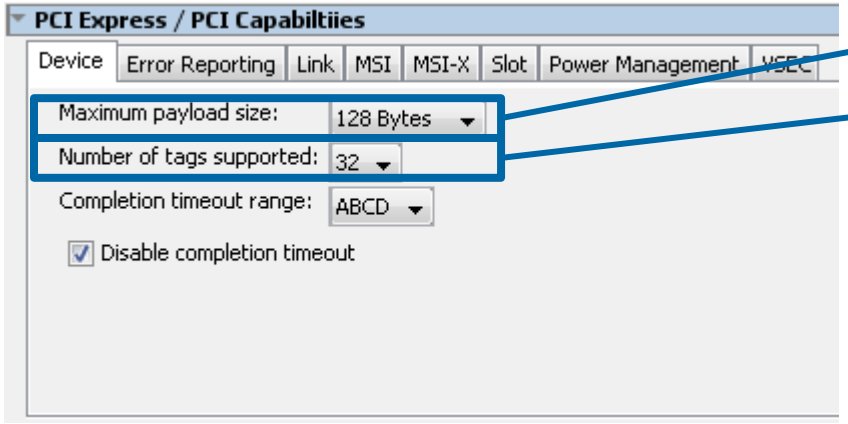


Type 0 Configuration Header

Byte			
3	2	1	0
Device ID		Vendor ID	
Status Register		Command Register	
Class Code			Revision ID
BIST	Type	Latency Timer	Cache Line Size
Base Address 0			
Base Address 1			
Base Address 2			
Base Address 3			
Base Address 4			
Base Address 5			
CardBus CIS Pointer			
Subsystem ID		Subsystem Vendor ID	
Expansion ROM Base Address			
Reserved			Capabilities Pointer
Reserved			
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

Customizing the HIP – PCI/PCIe Capabilities (Device)

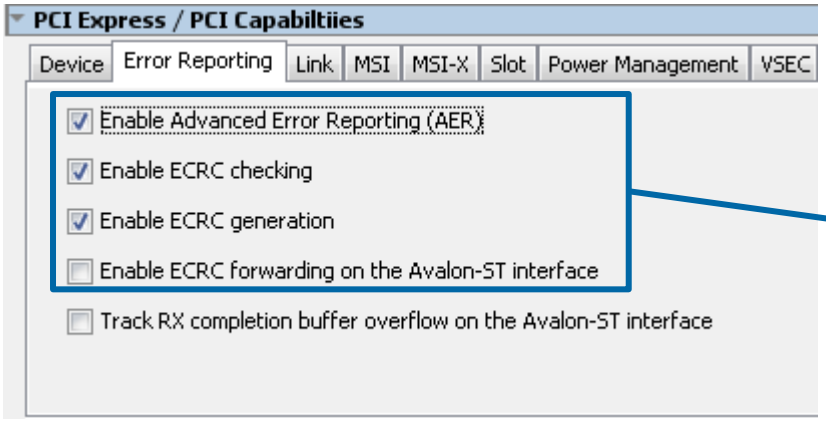
- Maximum Payload Size
  - 128/256/512/1024/2048 bytes
- Number of tags
  - For non-posted requests
  - Avalon-ST cores:
    - 32 or 64
  - Avalon-MM cores: 8 or 16



Byte Offset	31:16	15:8	7:0
0x080	PCI Express Capabilities Register	Next Cap Pointer	PCI Express Cap ID
0x084	Device Capabilities		
0x088	Device Status	Device Control 2	
0x08C	Link Capabilities		
0x090	Link Status	Link Control	
0x094	Slot Capabilities		
0x098	Slot Status	Slot Control	
0x09C	Root Capabilities	Root Control	
0x0A0	Root Status		
0x0A4	Device Capabilities 2		
0x0A8	Device Status 2	Device Control 2 Implement completion timeout disable	
0x0AC	Link Capabilities 2		
0x0B0	Link Status 2	Link Control 2	
0x0B4	Slot Capabilities 2		
0x0B8	Slot Status 2	Slot Control 2	

Customizing the HIP – PCI/PCIe Capabilities (Error Reporting)

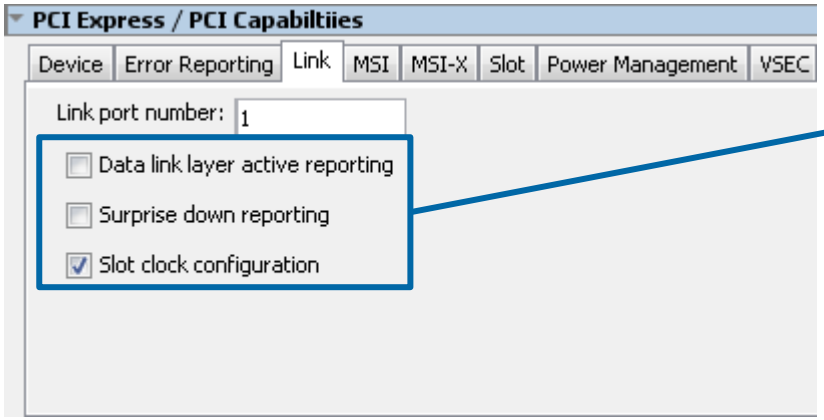
- Advanced error reporting
  - Enables the AER capability
- ECRC checking
  - Sets ECRC checking bit in control register
- ECRC generation
  - Set ECRC generation bit in control register
- Track RX Completion Buffer Overflow
  - Enables signals to track the RX posted completion buffer overflow status
  - Not available for the Avalon-MM component



Byte Offset	31:24	23:16	15:8	7:0
0x800	PCI Express Enhanced Capability Header			
0x804	Uncorrectable Error Status Register			
0x808	Uncorrectable Error Mask Register			
0x80C	Uncorrectable Error Severity Register			
0x810	Correctable Error Status Register			
0x814	Correctable Error Mask Register			
0x818	Advanced Error Capabilities and Control Register			
0x81C	Header Log Register			
0x82C	Root Error Command			
0x830	Root Error Status			
0x834	Error Source Identification Register		Correctable Error Source ID Register	

Customizing the HIP – PCI/PCIe Capabilities (Link Capabilities)

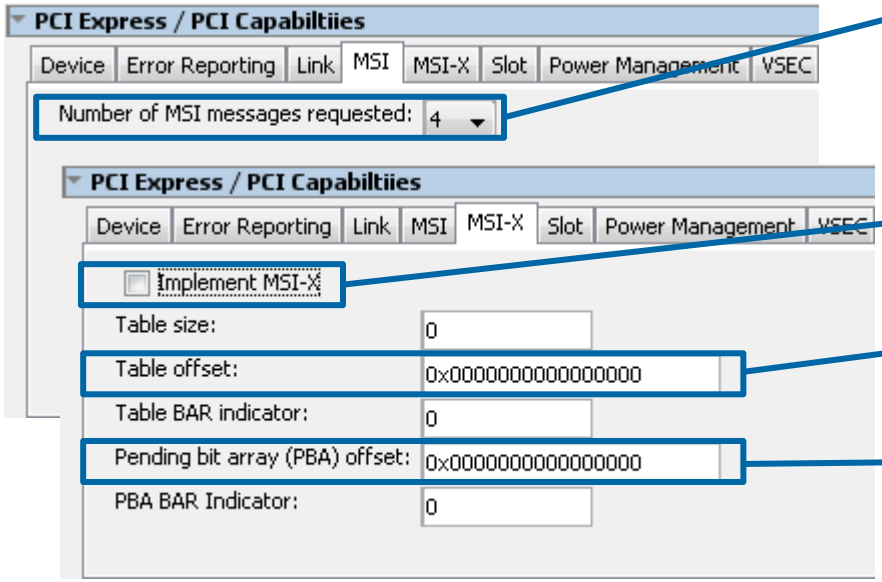
- Data Link Layer active reporting
  - For a hot-plug capable port this parameter must be turned on
  - Root Port mode only
- Surprise down
  - Supports the optional capability of detecting and reporting the surprise down error condition
  - Root Port mode only
- Slot clock
  - When enabled, the port uses the physical reference clock provided on the connector
  - When disabled, the IP core always uses an independent clock



Byte Offset	31:16	15:8	7:0
0x080	PCI Express Capabilities Register	Next Cap Pointer	PCI Express Cap ID
0x084	Device Capabilities		
0x088	Device Status	Device Control 2	
0x08C	Link Capabilities		
0x090	Link Status	Link Control	
0x094	Slot Capabilities		
0x098	Slot Status	Slot Control	
0x09C	Root Capabilities	Root Control	
0x0A0	Root Status		
0x0A4	Device Capabilities 2		
0x0A8	Device Status 2	Device Control 2 Implement completion timeout disable	
0x0AC	Link Capabilities 2		
0x0B0	Link Status 2	Link Control 2	
0x0B4	Slot Capabilities 2		
0x0B8	Slot Status 2	Slot Control 2	

Customizing the HIP – PCI/PCIe Capabilities (MSI/MSI-X Capabilities)

- Number of MSI Messages requested
  - Number of MSI messages the Application Layer can request
  - Sets the Multiple Message Capable field of the Message Control register
- Implement MSI-X
  - Enables the MSI-X functionality
- Table size
  - Sets the MSI-X Table size <n>
  - Legal range is 0–2047

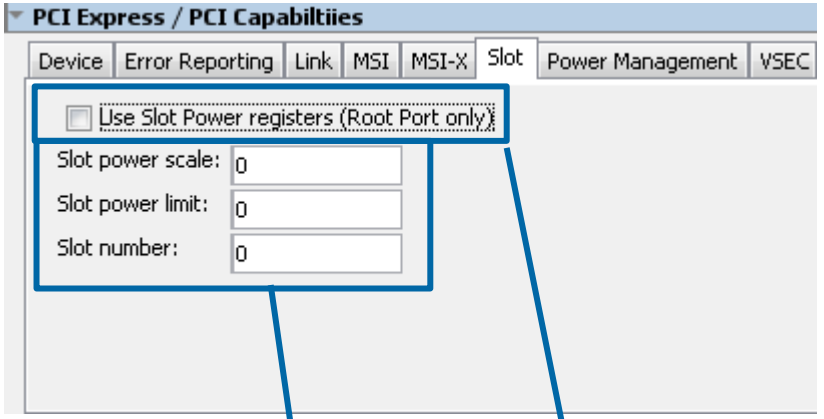


Byte Offset (1)	31:24	23:16	15:8	7:0
0x050	Message Control Configuration MSI Control Status Register Field Descriptions		Next Cap Ptr	Capability ID
0x054	Message Address			
0x058	Message Upper Address			
0x05C	Reserved		Message Data	

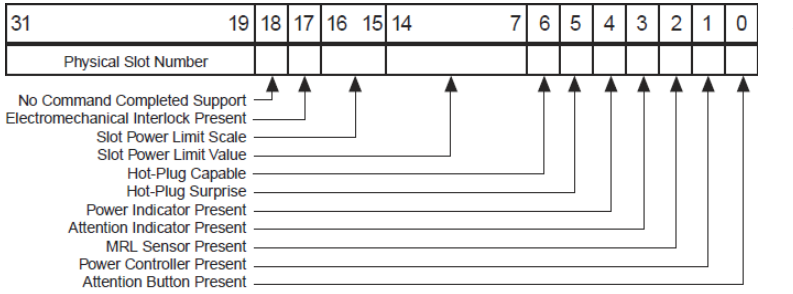
Byte Offset	31:24	23:16	15:8	7:3	2:0
0x068	Message Control		Next Cap Ptr	Capability ID	
0x06C	MSI-C Table Offset	MSI-X Table Offset			MSI-X Table BAR Indicator
0x070	MSI-X Pending Bit Array (PBA) Offset MSI-X Pending Bit Array (PBA) Offset				MSI-X Pending Bit Array - BAR Indicator MSI-X Pending Bit Array – BAR Indicator

Customizing the HIP – PCI/PCIe Capabilities (Slot Capabilities)

- Use Slot Register
  - Root Port mode only
  - Required if a slot is implemented on the port
- Slot Power scale (0-3)
  - Scale for the slot power limit
  - 0 = 1.0x, 1 = 0.1x, 2 = 0.01x, 3 = 0.001x
- Slot power limit (0–255)
  - Upper power limit supplied by the slot (watts)

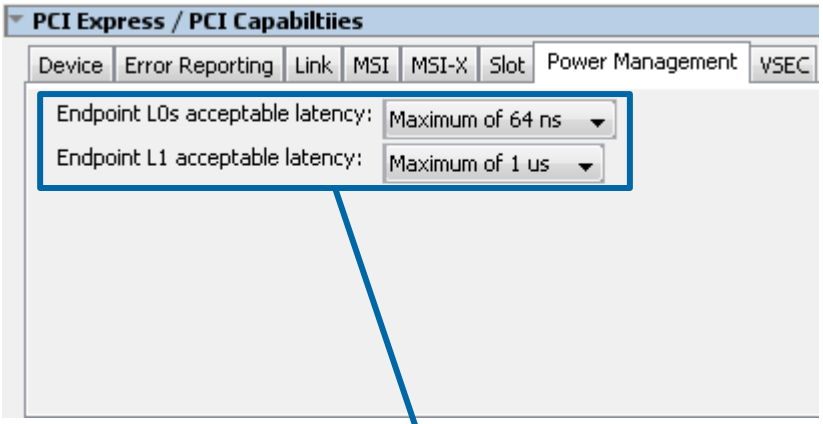


Byte Offset	31:16	15:8	7:0
0x080	PCI Express Capabilities Register	Next Cap Pointer	PCI Express Cap ID
0x084	Device Capabilities		
0x088	Device Status	Device Control 2	
0x08C	Link Capabilities		
0x090	Link Status	Link Control	
0x094	Slot Capabilities		
0x098	Slot Status	Slot Control	
0x09C	Root Capabilities	Root Control	
0x0A0	Root Status		
0x0A4	Device Capabilities 2		
0x0A8	Device Status 2	Device Control 2 Implement completion timeout disable	
0x0AC	Link Capabilities 2		
0x0B0	Link Status 2	Link Control 2	
0x0B4	Slot Capabilities 2		
0x0B8	Slot Status 2	Slot Control 2	



Customizing the HIP – PCI/PCIe Capabilities (Power Mgmt)

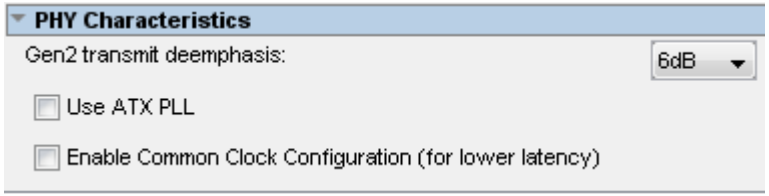
- Low-power modes are not supported by the PCIe HIP
  - These parameters allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM)
- Endpoint L0s acceptable latency
  - Maximum acceptable latency device can tolerate to exit the L0s
  - Sets the Endpoint L0s acceptable latency field of the Device Capabilities Register
  - Default 64 ns is the safest setting for most designs
- Endpoint L1 acceptable latency
  - Maximum acceptable latency device can tolerate to exit the L1 state
  - Sets the Endpoint L1 acceptable latency field of the Device Capabilities Register
  - Default 1 µs is the safest setting for most designs



Byte Offset	31:16	15:8	7:0
0x080	PCI Express Capabilities Register	Next Cap Pointer	PCI Express Cap ID
0x084	Device Capabilities		
0x088	Device Status	Device Control 2	
0x08C	Link Capabilities		
0x090	Link Status	Link Control	
0x094	Slot Capabilities		
0x098	Slot Status	Slot Control	
0x09C	Root Capabilities	Root Control	
0x0A0	Root Status		
0x0A4	Device Capabilities 2		
0x0A8	Device Status 2	Device Control 2 Implement completion timeout disable	
0x0AC	Link Capabilities 2		
0x0B0	Link Status 2	Link Control 2	
0x0B4	Slot Capabilities 2		
0x0B8	Slot Status 2	Slot Control 2	

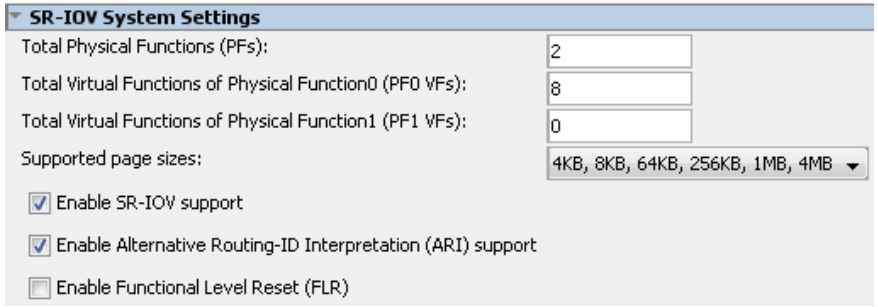
Parameter Editor – PHY Characteristics

- Stratix V / Arria 10 devices only
  - Avalon-ST interface only
- Gen2 transmit de-emphasis
  - 3.5dB: recommended for short PCB traces
  - 6.0dB: recommended for long PCB traces
- Use ATX Phase Lock Loop (PLL)
  - The ATX PLL is used instead of the CMU PLL *(discussed later)*
  - Requires the use of the soft reset controller *(discussed later)*
  - Does not support the CvP flow
- Enable Common Clock
  - Causes the Application Layer and Transaction Layer to use a common clock.
  - Reduces data path latency



Customizing the HIP – SR-IOV System Settings

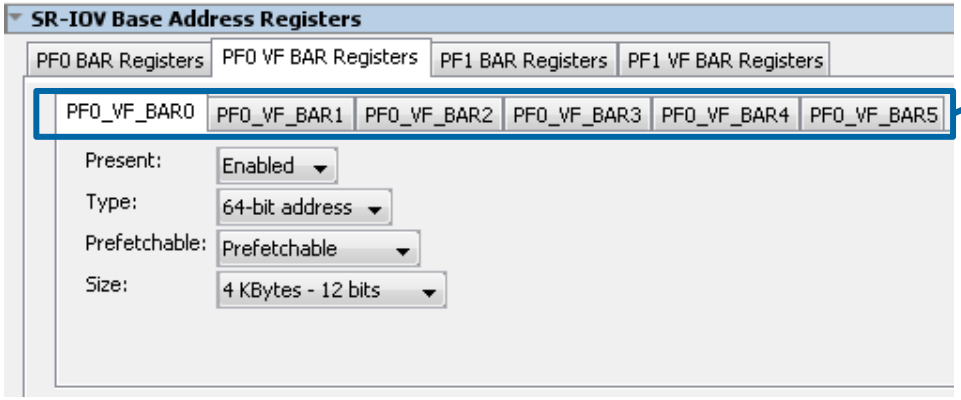
- ARI
  - Alternative Routing-ID Interpretation
  - Uses device number and function # from packet header
- The following are supported
  - 1 PF and 4-7 VFs with no ARI
  - 1 PF and 4-128 VFs in multiples of 4 with ARI
  - 2 PFs with 4-6 VFs and no ARI
  - 2 PFs with 4-128 VFs in multiples of 4 with ARI
- Total Physical Functions
  - This core supports 1 or 2 Physical Functions
- Total VF of PF
  - Total number of VFs for PF0
  - 4 to 128 VFs are supported
  - If PF1 is enabled, the sum of this field and PF1 VFs should not exceed 128
- Enable Function Level Reset
  - Allow each function to be individually reset



Customizing the HIP – SR-IOV Base Address Registers

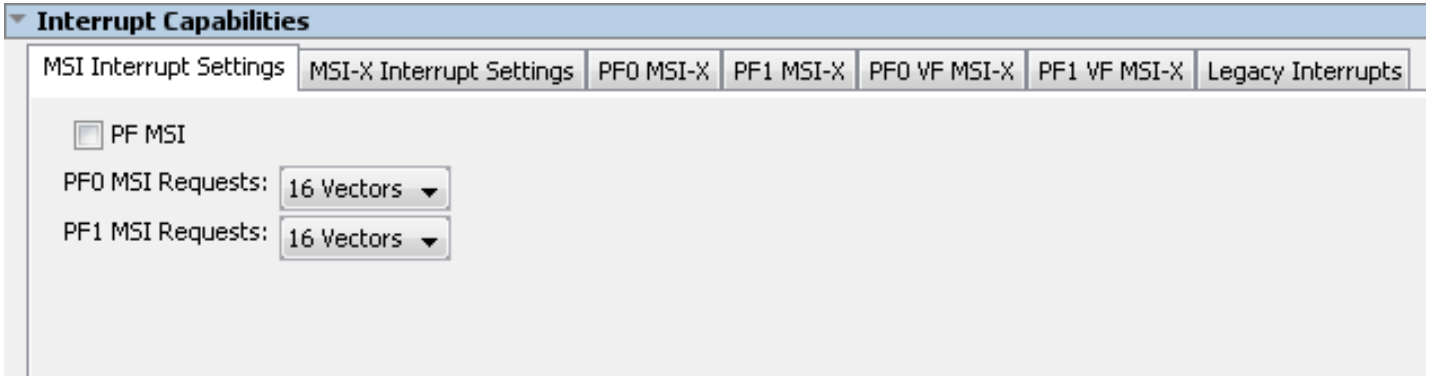
- PFx_VF_BARx
  - Define type and size for each BAR for all VFs in each PFx
  - 64-bit prefetchable (Uses up 2 BARs)
  - 32-bit non/prefetchable

	31 – 24	23 – 20	19 – 16	15 – 0
0x180	Next Capability Offset		Capability Version	PCI Express Extended Capability ID
0x184	SR-IOV Capabilities			
0x188	SR-IOV Status			SR-IOV Control
0x18C	TotalVFs (RO)			InitialVFs (RO)
0x190	RsvdP	Function Dependency Link (RO)		NumVFs (RW)
0x194	VF Stride (RO)			First VF Offset (RO)
0x198	VF Device ID (RO)			RsvdP
0x19C	Supported Pages Sizes (RO)			
0x1A0	System Page Size (RW)			
0x1A4	VF BAR0 (RW)			
0x1A8	VF BAR1 (RW)			
0x1AC	VF BAR2 (RW)			
0x1B0	VF BAR3 (RW)			
0x1B4	VF BAR4 (RW)			
0x1B8	VF BAR5 (RW)			
0x1BC	VF Migration State Array Offset (RO)			



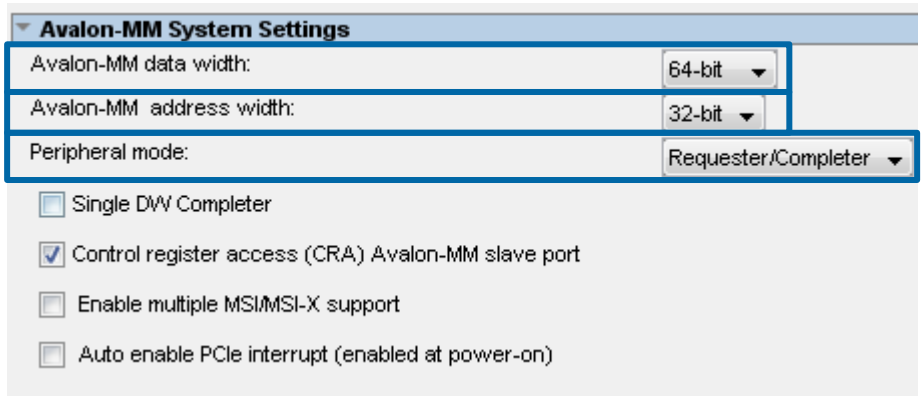
Customizing the HIP – Interrupt Capabilities (SR-IOV)

- Enable/disable MSI for all physical functions
- Enable/disable legacy interrupts for each physical function individually
- Enable/disable MSI-X for all physical and/or all virtual functions
- Set up MSI-X table
  - For each physical function
  - For all virtual functions under a physical function



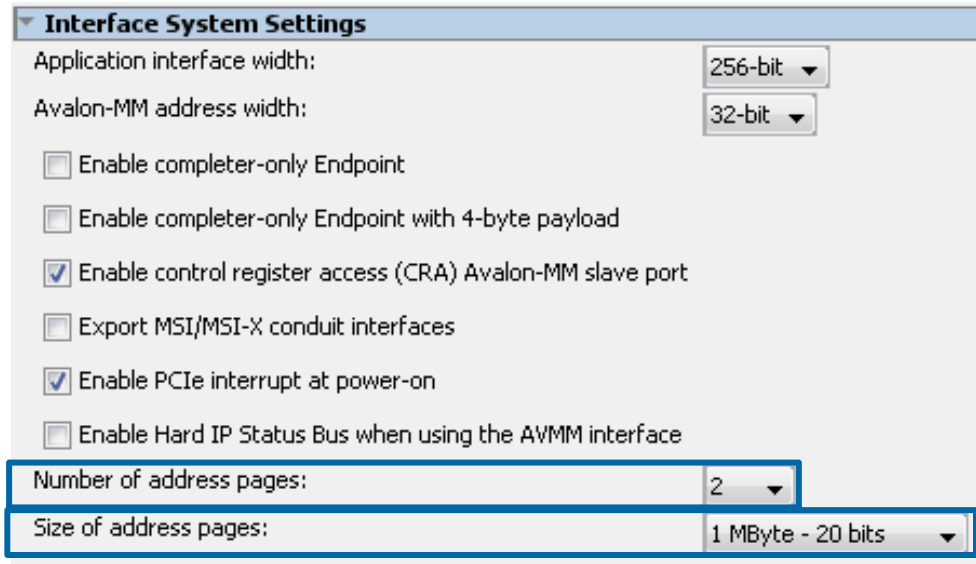
Customizing the HIP – Avalon-MM System Settings

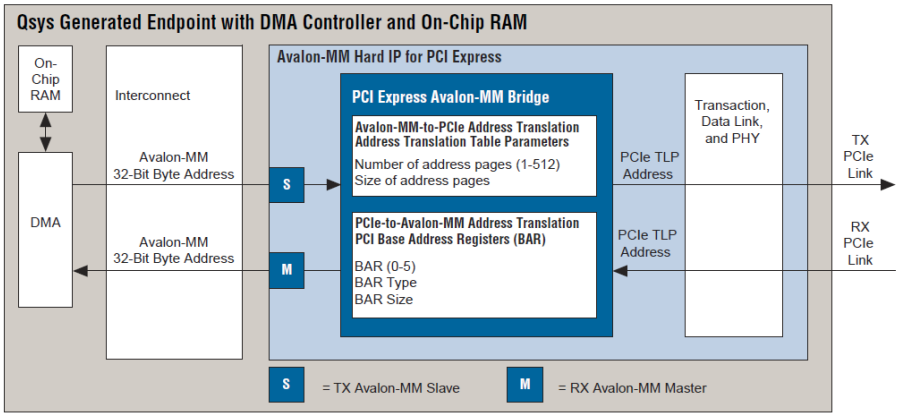
- Avalon-MM data width
  - 64-bit, 128-bit or 256-bit
  - Data width between the Transaction Layer and the Application Layer
- Avalon-MM address width
  - 32-bit or 64-bit
  - Address width for Avalon-MM RX ports that access Avalon-MM slaves
  - If 32-bit addressing is selected, address translation is performed
- Peripheral mode
  - Requester/Completer — can send requests and receive packets
  - Completer-Only — cannot send requests because the Avalon-MM TX slave port is removed to save logic



Customizing the HIP – Avalon-MM Address Translation

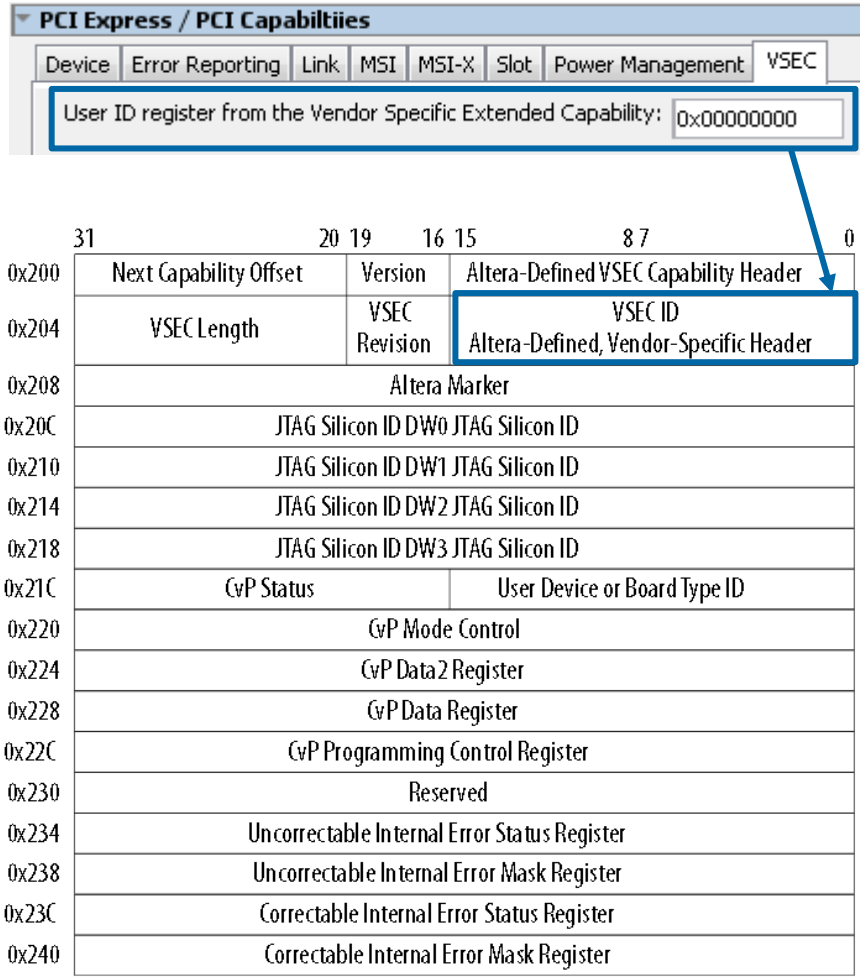
- Number of address pages
  - Specifies the number of pages required to translate Avalon-MM addresses to PCI Express addresses before a request packet is sent to the Transaction Layer
  - 1,2,4,8,16,32,64, 128,256,512
- Size of address pages
  - Specifies the size of each memory segment
  - Each memory segment must be the same size
  - 4 KByte –4 Gbytes
- 32-bit Address Avalon-MM components only
  - If 64-bit addressing is used, no address translation is necessary





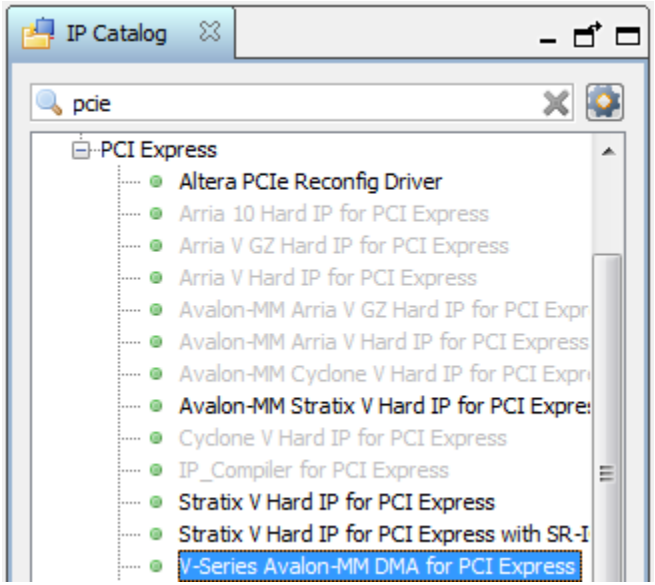
Customizing the HIP – Vendor Specific Extended Capability

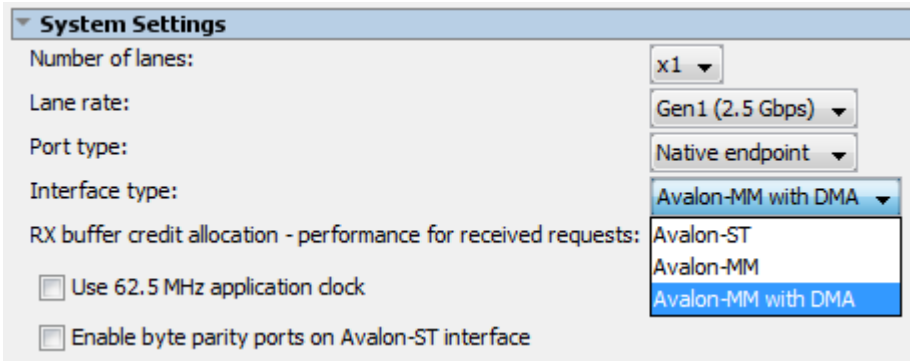
- Provides the 16-bit user ID for the Vendor Specific Extended Capability register
  - Remaining VSEC register values defined by Altera
- Supports CvP and detailed internal error reporting
- Stratix V and Arria 10 devices only



Selecting Avalon-MM with DMA IP Core

- V-Series Devices
  - Select V-Series Avalon-DMA for PCI Express in IP Catalog
- Arria 10 Devices
  - Select Arria 10 Hard IP for PCI Express in IP Catalog
  - Select Avalon-MM with DMA in the System Settings category

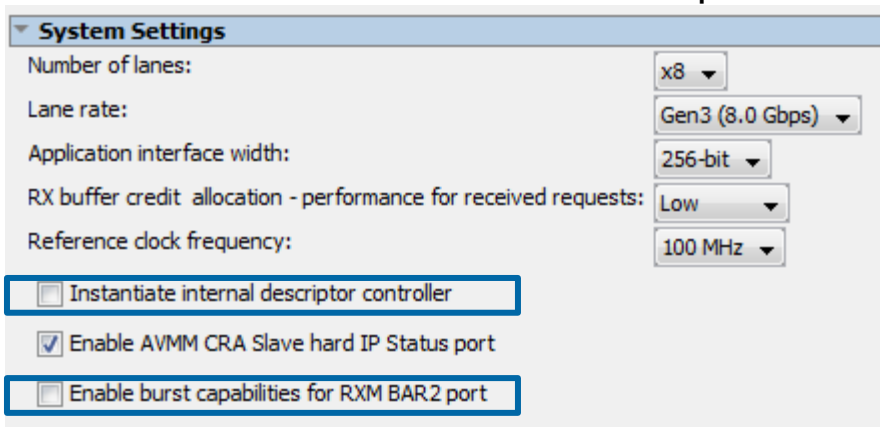




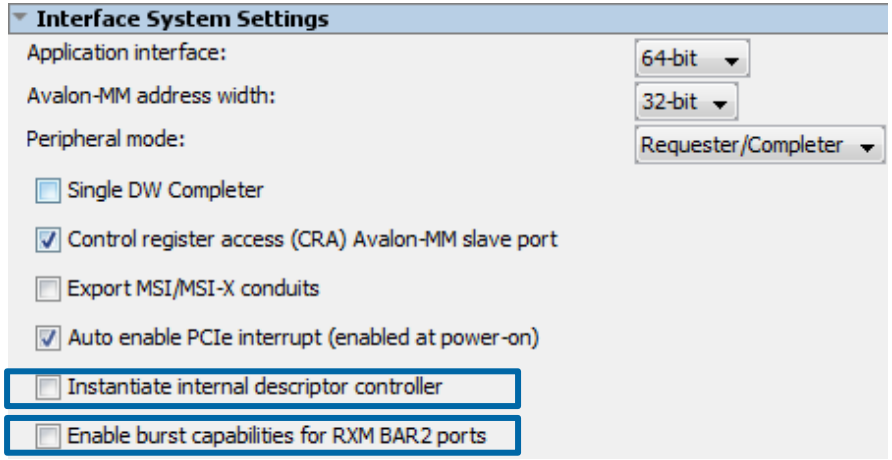
Avalon-MM with DMA specific parameters

- Instantiate internal descriptor controller
  - Turn this option on, if you plan to use the Altera-provided descriptor controller in your design.
  - Turn this option off if you plan to modify or replace the descriptor controller logic in your design.
- Enable burst capabilities
  - When you turn on this option, the BAR2 RX Avalon-MM master is burst capable
  - If BAR2 is 32 bits and Burst capable, then BAR3 is not available for other use
  - If BAR2 is 64 bits, the BAR3 register holds the upper 32 bits of the address

V-Series Avalon-DMA for PCI Express

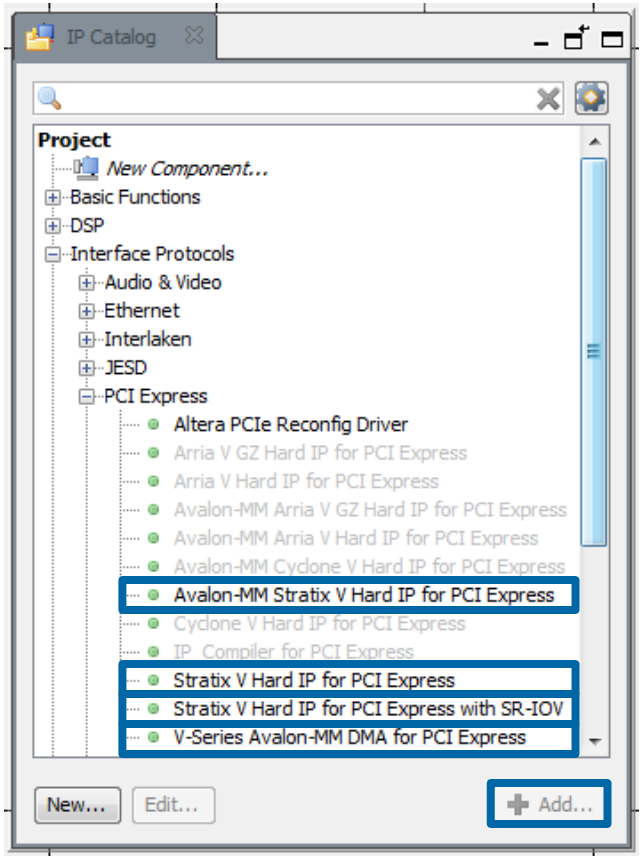


Arria 10 Hard IP for PCI Express



Building a Brand New Qsys System

- Select from the IP Catalog
  - (device family) Hard IP for PCI Express (Avalon-ST) component
  - (device family) Hard IP for PCI Express with SR-IOV (Avalon-ST) component
  - Avalon-MM (device family) Hard IP for PCI Express component
  - V-Series Avalon-MM DMA for PCI Express component
- Select +Add
- Use Qsys to connect Hard IP to
  - Application Layer
  - Control logic
  - Debugging/test logic
  - Clocks and resets



IP Catalog Flow

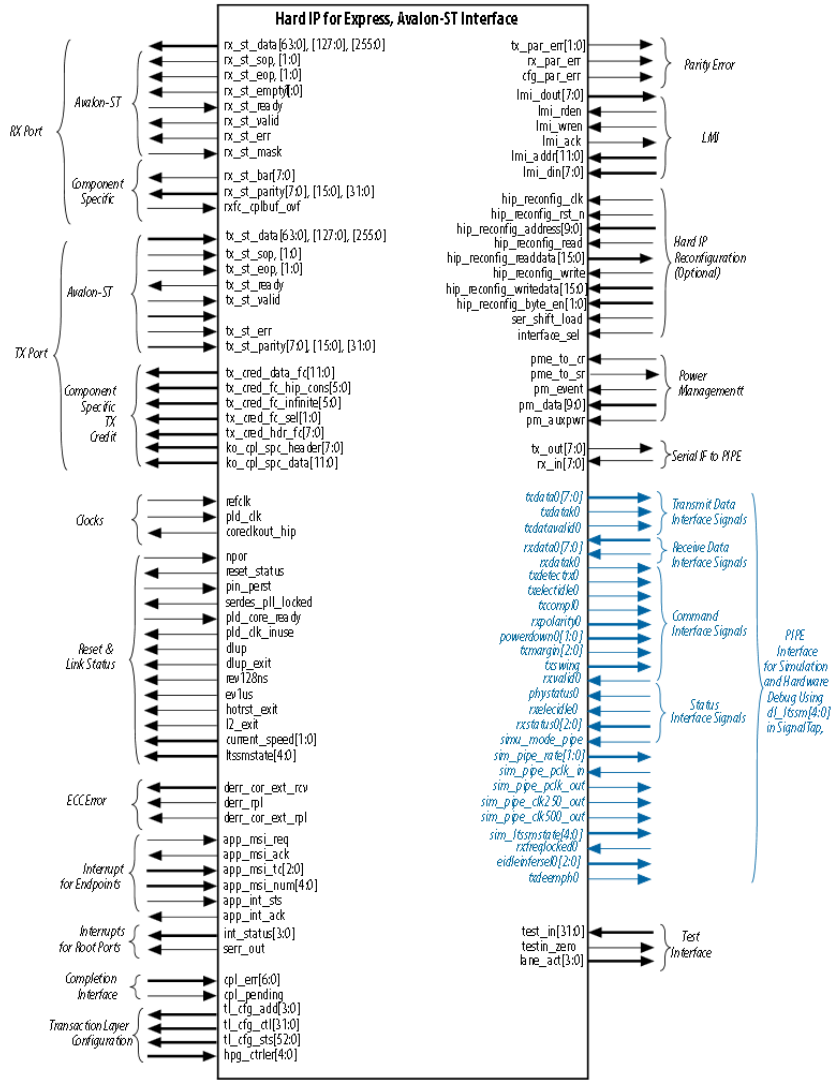
- Select and parameterize the Hard IP using the IP Catalog
- Instantiate and connect the Hard IP in HDL
  - Avalon-ST Hard IP interface cores only
  - Same feature set as Qsys flow
  - Same parameter editor as Qsys flow
  - Generates an example Qsys system design
- Manually add timing constraint to the top-level design SDC file
  - Only **`derive_pll_clocks -create_base_clocks`** command needed to constrain entire Hard IP

Building a Hard IP for PCI Express Design Section Agenda

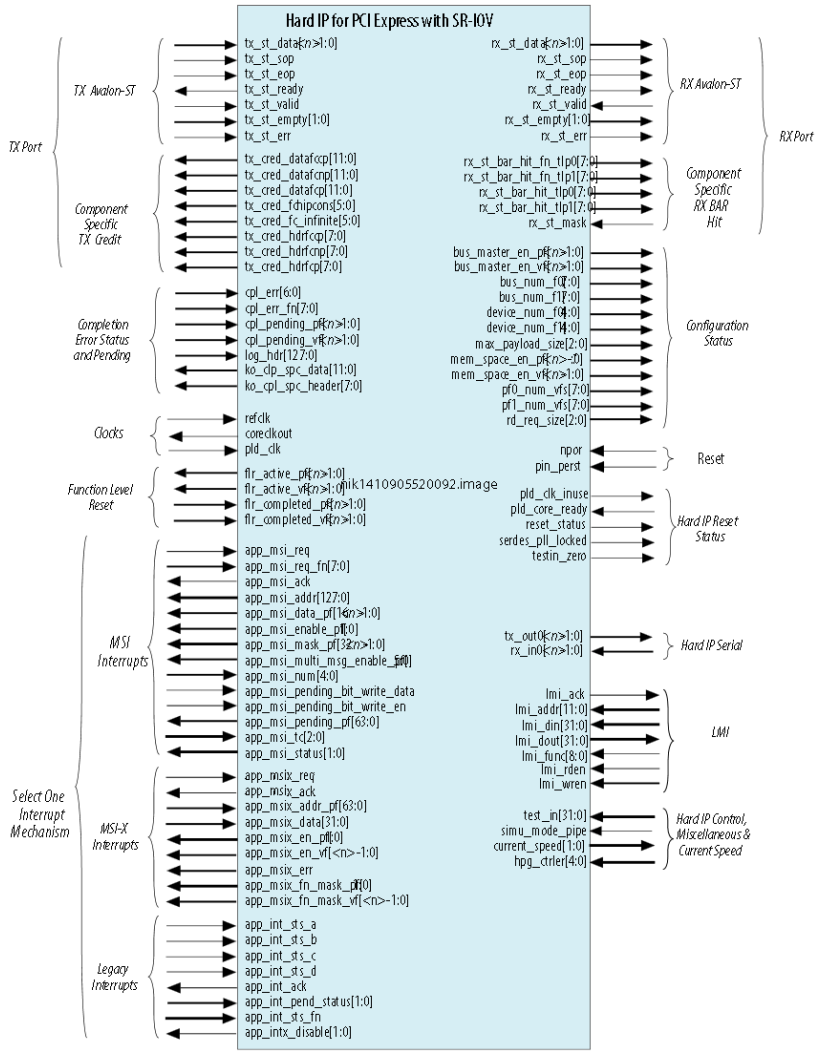
- Hard IP for PCI Express Features
- Customizing the Hard IP for PCI Express Settings
- Connecting the Hard IP for PCI Express Interfaces
- Design Implementation

Connecting the HIP – Avalon-ST Component Symbols

Avalon-ST



Avalon-ST with SR-IOV

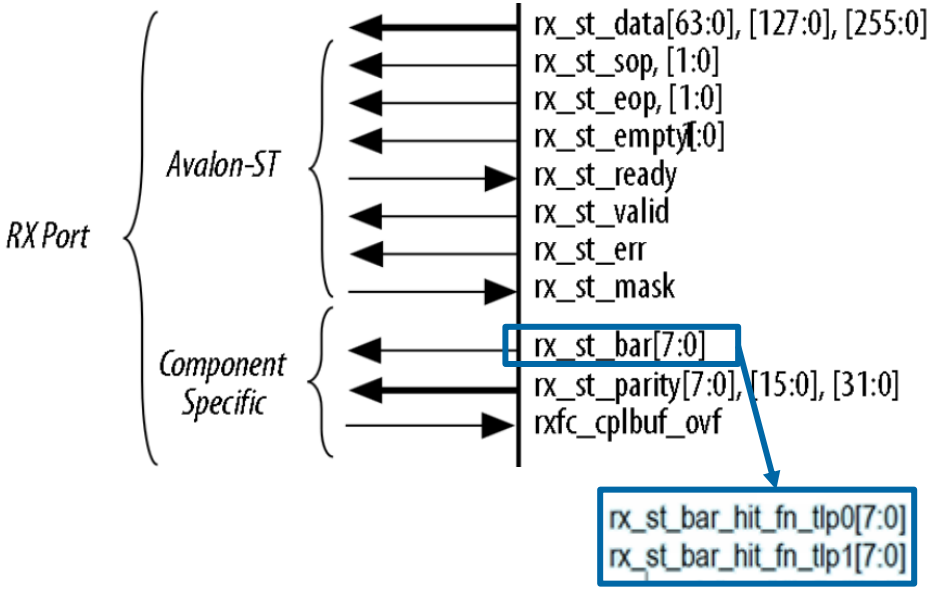


Connecting the HIP - Avalon-ST Component Interfaces

- Avalon ST RX Port
- Avalon ST TX Port
- Transaction Layer Configuration
- Link Management Interface (LMI)
- Hard IP Reconfiguration
- Power Management
- Transceiver Reconfiguration
- Clocks
- Reset & Link Status
- Error Correcting/Correction Code (ECC) Errors
- Interrupts
- Completion Interface
- SR-IOV Interfaces

Connecting the HIP - RX Avalon-ST interfaces signals

- Standard Avalon-ST signals
  - Data (64, 128, or 256 bits)
  - Data valid
  - Ready
  - Start of packet
  - End of packet
  - Empty
  - Error
- Non-standard Avalon-ST signals
  - Parity
  - Decoded BAR (standard IP)
  - Byte enable (deprecated(1))
  - Mask - stop non-posted requests

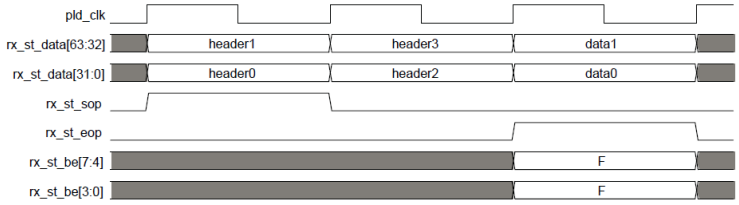


- *Decoded BAR (SR-IOV IP)*
  - Identifies the Function number that was hit by a TLP
  - Valid in the first cycle of a TLP Valid for MRd, MWr and Atomic Op TLPs and to be ignored for all other TLPs
  - rx_st_bar_hit_fn_tlp0[7:0] applies to the TLP that starts on bits [127:0]
  - rx_st_bar_hit_fn_tlp1[7:0] applies to the TLP that starts on bits [255:128]

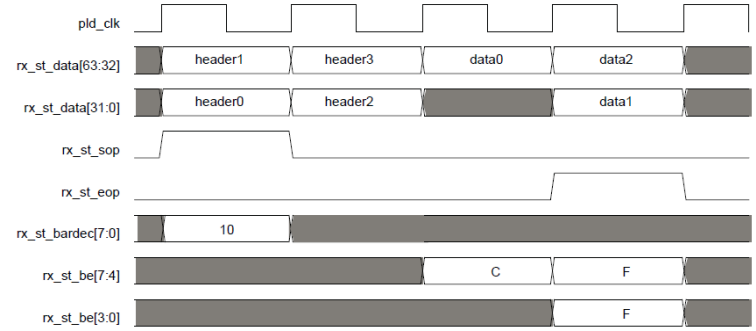
(1) The same information can be found by decoding the FBE and LBE fields in the TLP header.

Connecting the HIP - Avalon-ST RX Port Examples

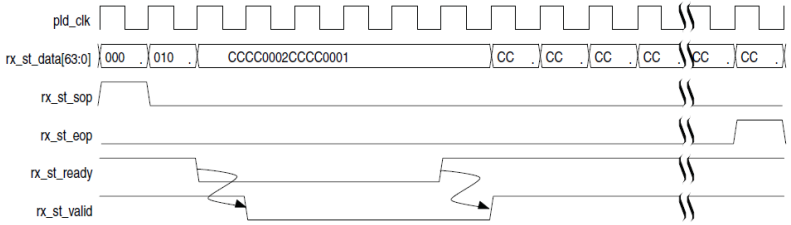
64-bit Data, 4 DW Header, 64-bit aligned address



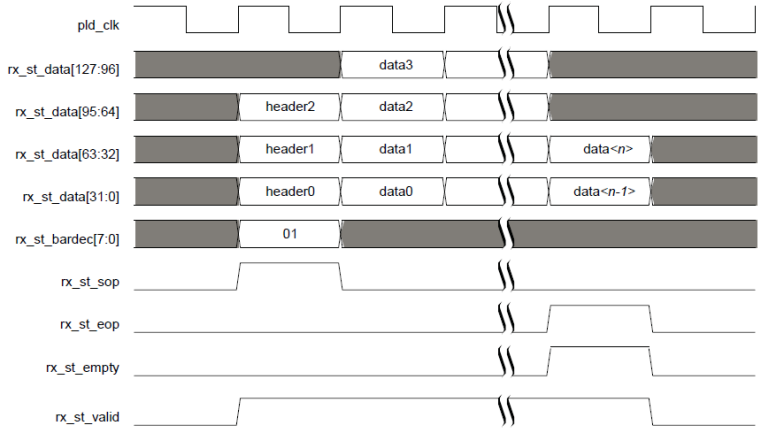
64-bit Data, 4 DW Header, non-64-bit aligned address



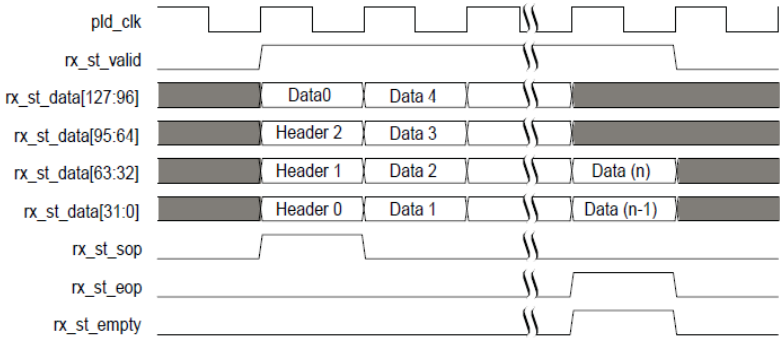
64-bit Data, application backpressures the IP



128-bit Data, 3 DW Header, 64-bit aligned address

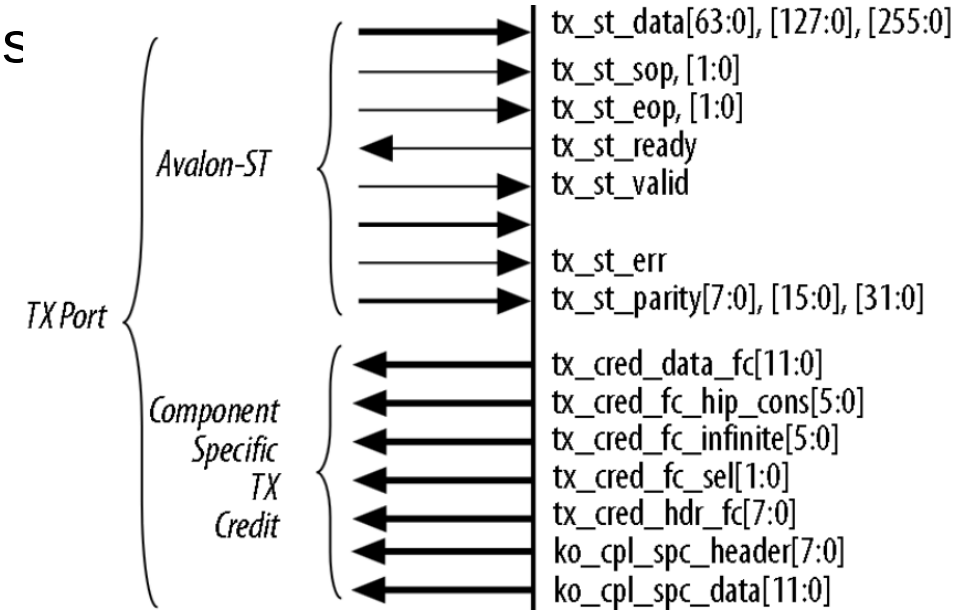


128-bit Data, 3 DW Header, non-64-bit aligned address



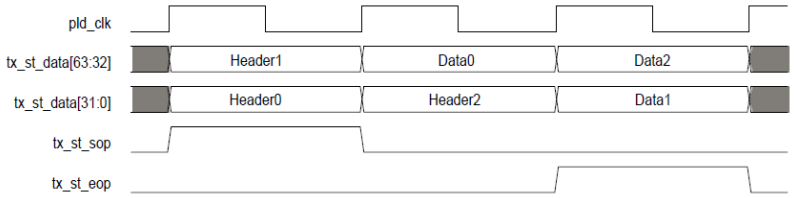
Connecting the HIP - TX Avalon-ST Interfaces Signals

- Standard Avalon-ST Signals
  - Data (64, 128, or 256 bits)
  - Data valid
  - Ready
  - Start of packet
  - End of packet
  - Empty
  - Error
- Non-standard signals
  - TX credit interface
  - TX completion interface

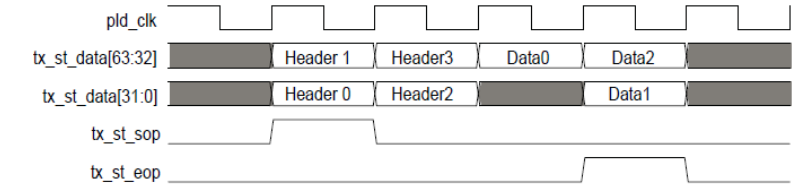


Connecting the HIP - Avalon-ST TX Examples

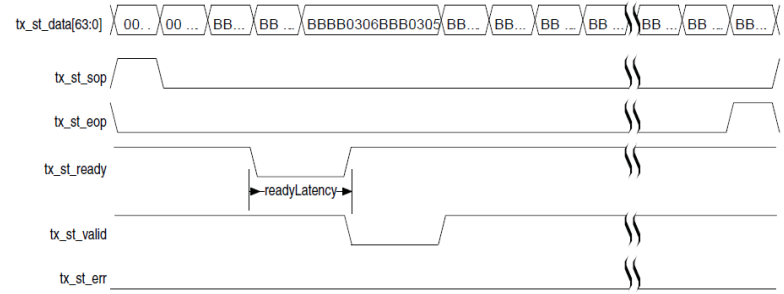
64-bit Data, 3 DW Header, 64-bit aligned address



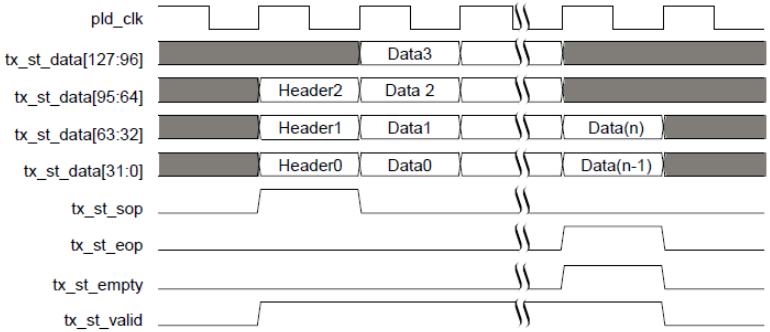
64-bit Data, 4 DW Header, non-64-bit aligned address



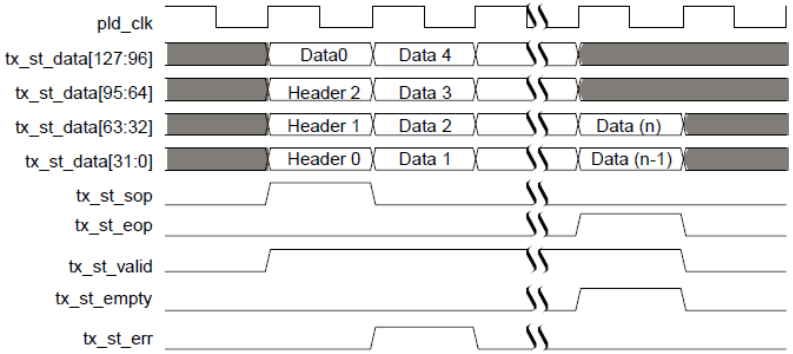
64-bit Data, IP backpressures the application



128-bit Data, 3 DW Header, 64-bit aligned address



128-bit Data, 3 DW Header, non-64-bit aligned address



Avalon-ST Cores - Avalon-ST TX Credit Interface (28-nm Devices)

- Produces credit information for each packet type

Signal	Description
tx_cred_datafccp[11:0]	Data credit limit for the received FC completions
tx_cred_datafcnp[11:0]	Data credit limit for the non-posted requests
tx_cred_datafcp[11:0]	Data credit limit for the FC posted writes.
tx_cred_fchipcons[5:0]	Single clock cycle pulse indicates that the corresponding credit type has been consumed
tx_cred_fc_infinite[5:0]	Indicates that the corresponding credit type has infinite credits available
tx_cred_hdrfccp[7:0]	Header credit limit for FC completions
tx_cred_hdrfcnp[7:0]	Header limit for the non-posted requests
tx_cred_hdrfcp[7:0]	Header limit for the posted writes
ko_cpl_spc_header[7:0]	Total number of completion headers that can be stored in the RX buffer.
ko_cpl_spc_data	Total number of 16 byte completion data units that can be stored in the completion RX buffer

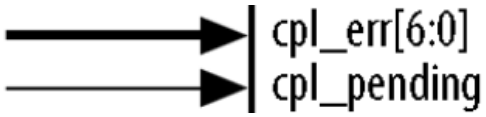
Avalon-ST Cores - Avalon-ST TX Credit Interface (Arria 10 Devices)

- Produces credit information for each packet type

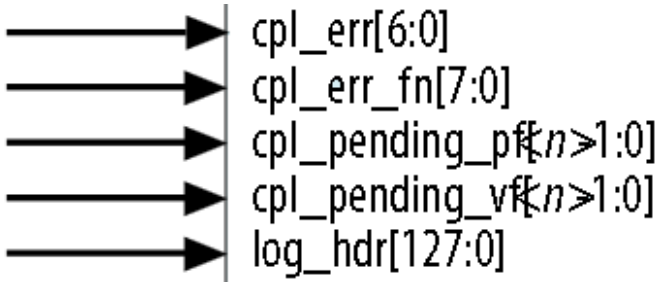
Signal	Description
tx_cred_data_fc[11:0]	Data credit limit for the flow control
tx_cred_hdr_fc[11:0]	Header credit limit for the flow control
tx_cred_fc_sel[5:0]	Input signal to select which credit information is provided on *tx_cred_data_fc* and *tx_cred_hdr_fc* (00:Posted; 01:Non-posted; 02:Completions)
tx_cred_fc_hip_cons[5:0]	Single clock cycle pulse indicates that the corresponding credit type has been consumed by the Hard IP (not the Application Layer); Must be added to Application Layer credits consumed
tx_cred_fc_infinite[5:0]	Indicates that the corresponding credit type has infinite credits available
ko_cpl_spc_header[7:0]	Total number of completion headers that can be stored in the RX buffer.
ko_cpl_spc_data	Total number of 16 byte completion data units that can be stored in the completion RX buffer

Avalon-ST Cores - Completion Interface

- When the Application Layer detects an error, it can assert the appropriate *cpl_err* bit to indicate what kind of error to log
  - The Hard IP sets the appropriate status bits and sends error messages in accordance with the PCI Express specification
  - Examples
    - *cpl_err[0]*: Completion timeout error with recovery
- *cpl_err_fn* (SR-IOV)
  - Function number associated with the completion error on cpl_err[6:0]
- *log_hdr* (SR-IOV)
  - Application Layer provides header of errored TLP
- *cpl_pending*, *cpl_pending_pf*, *cpl_pending_vf*
  - The Application Layer must keep asserted when the Hard IP is waiting for completion



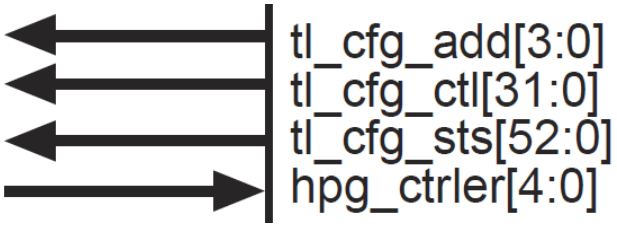
Completion Interface (standard)



Completion Interface (SR-IOV)

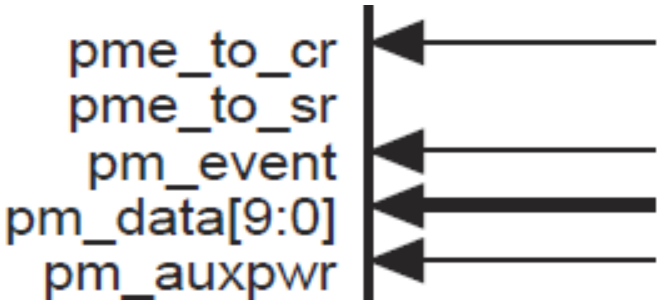
Avalon-ST Cores - TL Configuration Interface

- Primary method for reading the Configuration Space during normal operation
  - e.g. reading the system supported Maximum Payload Size
  - tl_cfg_add, tl_cfg_ctl
    - CSR information mapped to a 16 DW address space
    - Two busses provide the address and contents of each DW in a round robin fashion
    - See the user guide for CSR to address mapping
  - hpg_ctrler
    - Used by Root Port to define slot characteristics
    - e.g. Attention button pressed, power fault detected
- Not available when Transaction Layer is bypassed



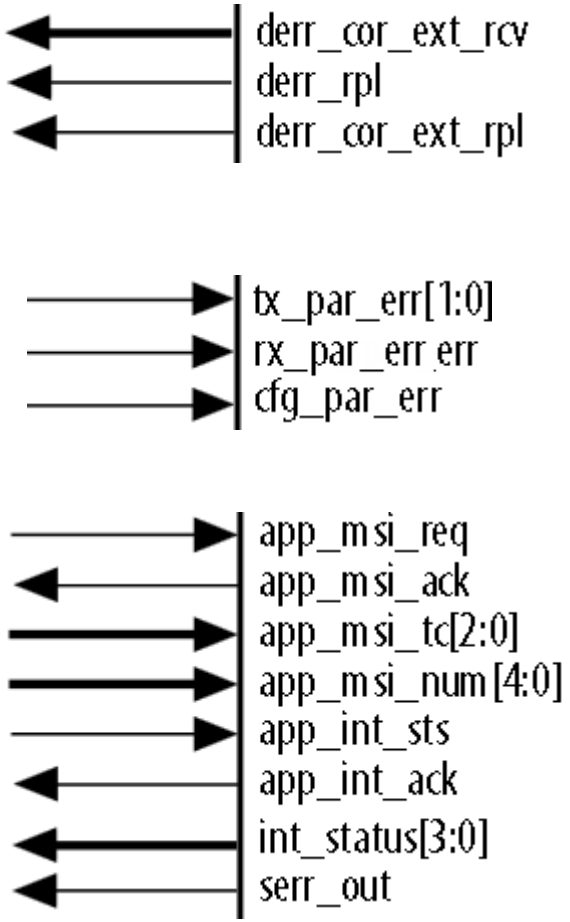
Avalon-ST Cores - Power Management

- Simple status and request signals
- Data bus for power consumption reporting
- Handles the transition between normal operation and the supported low power modes
- Creates and responds to power management DLLPs
  - Root Port sends power change requests
  - Endpoint responds to power change requests
  - Endpoint reports power consumption and power events



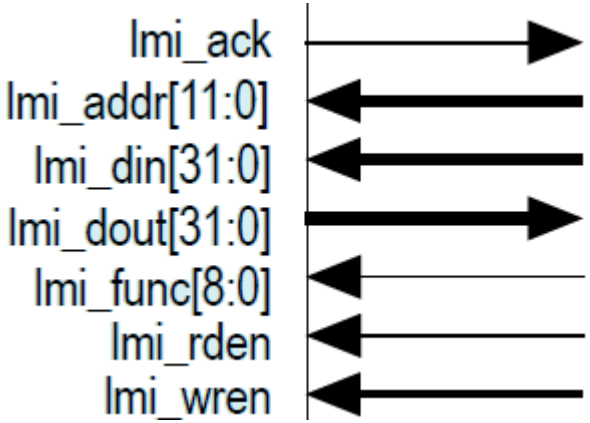
Avalon-ST Cores - Errors and Interrupts

- Error correcting code (ECC) error interface
  - Error correction status for RX and retry buffers
- Parity error signals
  - Parity error detected in sent or received TLP
- Interrupt (legacy, MSI & MSI-X) control signals
  - Process and control interrupt requests and responses



Connecting the HIP - Avalon-ST Link Management (LMI)

- Avalon-MM interface
  - 12- or 15-bit address
  - 32-bit data
  - Synchronized to pld_clk up to 250MHz
  - Maps directly to Configuration Space
- LMI reads
  - Obtain the contents of any Configuration Space register
  - May be issued at any time
- LMI writes
  - Provided for error report header logging, and debugging purposes only
  - Not for use during normal operation
- When an LMI write has a timing conflict with configuration TLP access, the configuration TLP accesses have higher priority
  - Use lmi_ack to verify completion of request
- lmi_func[8:0]
  - Bits [7:0] specify the function number corresponding to the LMI
  - SR-IOV IP only - Bit [8] directs the LMI read or write operation to either the HIP or the Function configuration

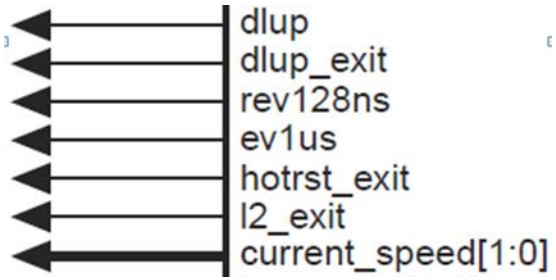


Avalon-ST Cores - Hard IP Reconfiguration

- Avalon-MM interface
  - 10-bit address
  - 16-bit data
- Allows changing "**read-only"** configuration registers
  - e.g. Link configuration, maximum payload size, power management settings/capabilities, MSI & MSI-X capabilities, BAR setup
  - Allows dynamic changes during device operation without reconfiguring entire FPGA
  - **CAUTION: Changing read-only configuration registers during operation can cause errors and other unexpected results**
  - **To ensure proper system operation, reset or repeat device enumeration of the link after changing the values of these registers**
- Example procedure
  1. Hold Hard IP in reset
  2. Change configuration register settings using reconfiguration block
  3. Bring Hard IP out of reset

Avalon-ST Cores - Link Status Signals

Status Signal	Description
serdes_pll_locked	Indicates coreclkout_hip clock signal is locked
pld_core_ready	Indicates Application Layer is ready
pld_clk_inuse	Transaction Layer is using the pld_clk and is ready
dlup	Indicates that the Hard IP block is in the up state
dlup_exit	Asserted low for one pld_clk cycle if the Data Link Layer has loses link
ev128ns	Asserted every 128 ns to create a time base aligned activity
ev1us	Asserted every 1 µs to create a time base aligned activity
hotrst_exit	Hot reset exit - Asserted for 1 clock cycle when the LTSSM exits hot reset state
l2_exit	Asserted low for one cycle after the LTSSM transitions from l2.idle to detect.
lane_act[3:0]	Each bit indicates the number of lanes that were enabled during link training
currentspeed[1:0]	00: Undefined, 01: Gen1, 10: Gen2, 11: Gen3



Avalon-ST Cores - LTSSM Link Status Signals

ltssmstate [4:0]	LTSSM State
00000	Detect.Quiet
00001	Detect.Active
00010	Polling.Active
00011	Polling.Compliance
00100	Polling.Configuration
00101	Polling.Speed
00110	config.Linkwidthstart
00111	Config.Linkaccept
01000	Config.Lanenumaccept
01001	Config.Lanenumwait
01010	Config.Complete
01011	Config.Idle
01100	Recovery.Rcvlock
01101	Recovery.Rcvconfig
01110	Recovery.Idle

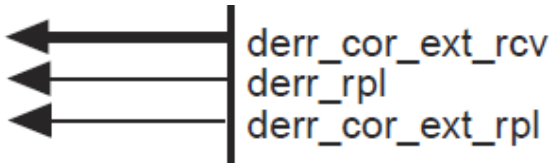
ltssmstate [4:0]	LTSSM State
01111	L0
10000	Disable
10001	Loopback.Entry
10010	Loopback.Active
10011	Loopback.Exit
10100	Hot.Reset
10101	L0s
11001	L2.transmit.Wake
11010	Recovery.Speed
11011	Recovery.Equalization, Phase 0
11100	Recovery.Equalization, Phase 1
11101	Recovery.Equalization, Phase 2
11110	Recovery.Equalization, Phase 3

Connecting the HIP – Transceiver Reconfiguration

- Transceiver Reconfiguration Controller IP
  - Required for Hard IP for PCI Express designs for V-Series Devices
    - Must be connected to transceiver interface on the Hard IP for PCI Express
    - **Not required for Arria 10 devices**
  - Dynamic reconfiguration may be necessary to compensate for process variations
    - Controller provides access to transceiver PMA settings via Avalon-MM interface
  - Automatically performs calibration for certain transceiver resources
    - RX channel buffers
    - ATX PLLs
- PCI Express Reconfiguration Driver IP
  - Automatic Gen3 equalization
  - Connect to the Avalon-MM bus on the controller for Gen3
  - May modify the Verilog file if necessary to meet your system requirements
  - Additional custom master may be connected to same Avalon-MM slave port for custom configuration

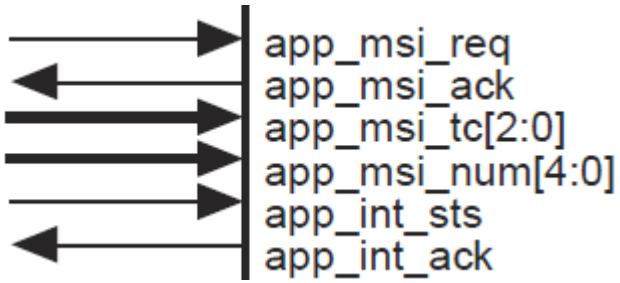
Connecting the HIP - Avalon-ST – ECC Error Interface

- ECC for the RX and retry buffers is implemented with MRAM
- Error signals are flags
- Valid for one clock cycle
- For debug only
- derr_cor_ext_rcv
  - Indicates a corrected error in the RX buffer
  - Hardware recovers from correctable ECC error occurs without any loss of information
  - No Application Layer intervention is required
- derr_core_ext_rpl
  - Indicates a corrected error in the retry buffer
  - Hardware recovers from correctable ECC error occurs without any loss of information
  - No Application Layer intervention is required
- derr_rpl
  - Indicates an uncorrectable error in the retry buffer
  - Resetting the core is recommended



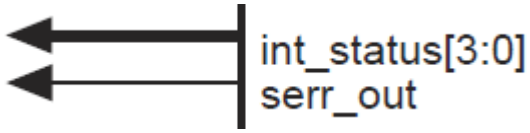
Connecting the HIP – Avalon-ST Interrupt Interface

Signal	IO	Description
app_msi_req	I	Causes an MSI posted write TLP to be generated
app_msi_ack	O	Acknowledges the request for an MSI interrupt
app_msi_tc[2:0]	I	Traffic class used to send the MSI
app_msi_num[4:0]	I	Low order bits in the message data field of the MSI
app_int_sts	I	Causes an Assert_INTA message to be generated
app_int_ack	O	Acknowledges assertion of the app_int_sts signal



Connecting the HIP – Avalon-ST Interrupts for Root Ports

- Drives legacy interrupts to the Application Layer
  - int_status[0]: interrupt signal A
  - int_status[1]: interrupt signal B
  - int_status[2]: interrupt signal C
  - int_status[3]: interrupt signal D



- System Error (serr_out)
  - Asserted for a single clock cycle when a system error occurs
  - Only asserted when that specific system error detection is enabled in the Root Control register and the Device Control register

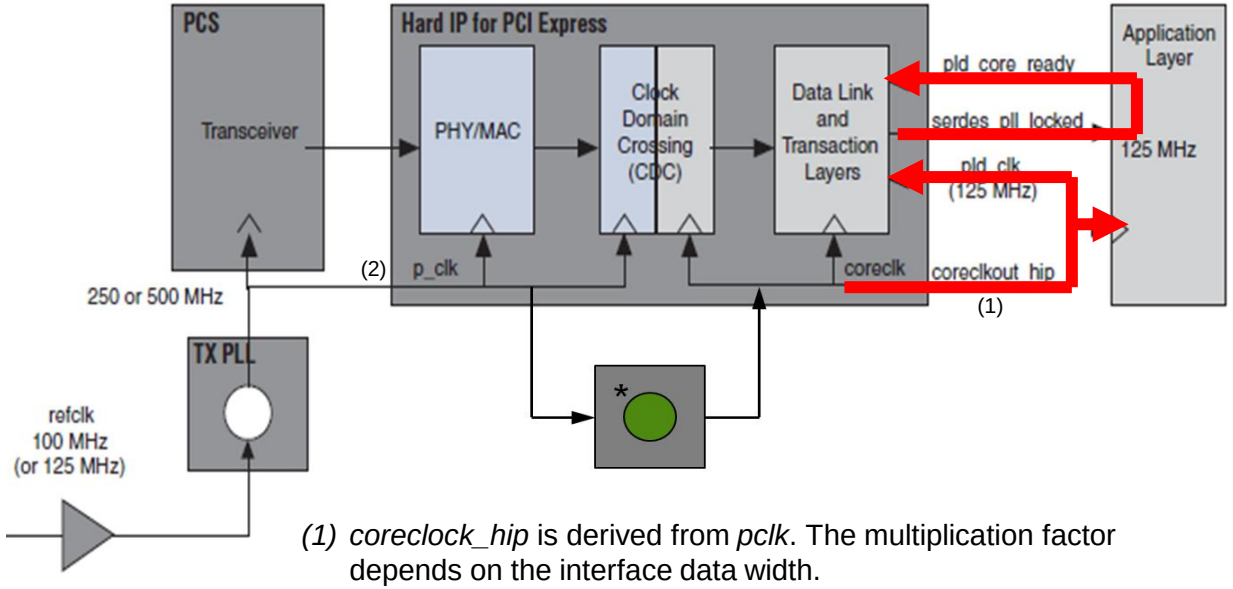
Connecting the HIP - Avalon-ST IP Core Clocks

Name	Frequency	Description
refclk(1)	100 or 125 MHz	• Dedicated free running Xcvr clock • ± 300 PPM input reference clock (Gen1, Gen2, Gen3) • Typically from PCI Express slot for or on-board oscillator
p_clk(2)	100 or 125 MHz	• Derived from refclk • User may disable for glitchless clock rate transitions (e.g. Gen1 → Gen2)
coreclkout_hip	62.5, 125, or 250 MHz	• Output clock generated by TX PLL • Provides Hard IP clock to Application Layer (FPGA core) • Derived by the transceiver from p_clk
pld_clk	62.5, 125, or 250 MHz	• Used by TL and application logic • Use by Application Layer and Transaction Layer • Same frequency as coreclkout_hip • May be derived by from coreclkout_hip
reconfig_xcvr_clk	100 - 125 MHz	• Required for Transceiver Reconfiguration Controller • Free running 100MHz-125MHz range • For Configuration via Protocol (CvP), *refclk* is used as *reconfig_xcvr_clk*
hip_reconfig_clk	50 - 125 MHz	• For the Hard IP reconfiguration interface

1. Arria 10 devices only support 100 MHz for refclk
2. For Information only. No user connection to p_clk.

Avalon-ST Cores - pld_clk generation

- *pld_clk* may be derived from *coreclock_hip* or generated independently by the Application Layer
  - *pld_clk* must be the same frequency as *coreclock_hip*(1)
  - *pld_clk* and core_clock_hip do not have to be in phase
- Application Layer should assert *pld_core_ready* when the logic is ready and *serdes_pll_locked* is asserted



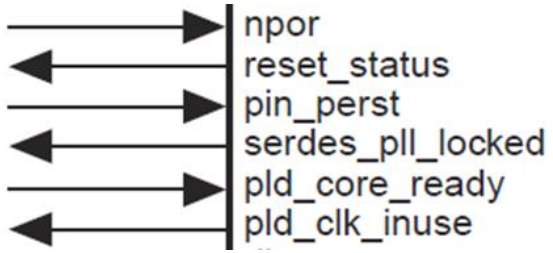
(1) *coreclock_hip* is derived from *pclk*. The multiplication factor depends on the interface data width.
(2) For Information only. No user connection to *p_clk*.

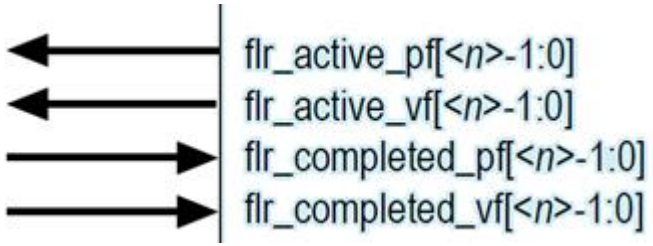
Avalon-ST Cores - Data Widths and Frequencies

PCIe Version	Link Width	Application Clock Frequency (MHz)	Avalon-ST Datapath Width
Gen1	x1	62.5	64
	x1	125	64
	x2	125	64
	x4	125	64
	x8	250	64
	x8	125	128
Gen2	x1	125	64
	x2	125	64
	x4	250	64
	x4	125	128
	x8	250	128
	x8	125	256
Gen3	x1	125	64
	x2	125	64
	x2	250	128
	x4	250	128
	x4	125	256
	x8	250	256

Connecting the HIP – Reset Control & Status

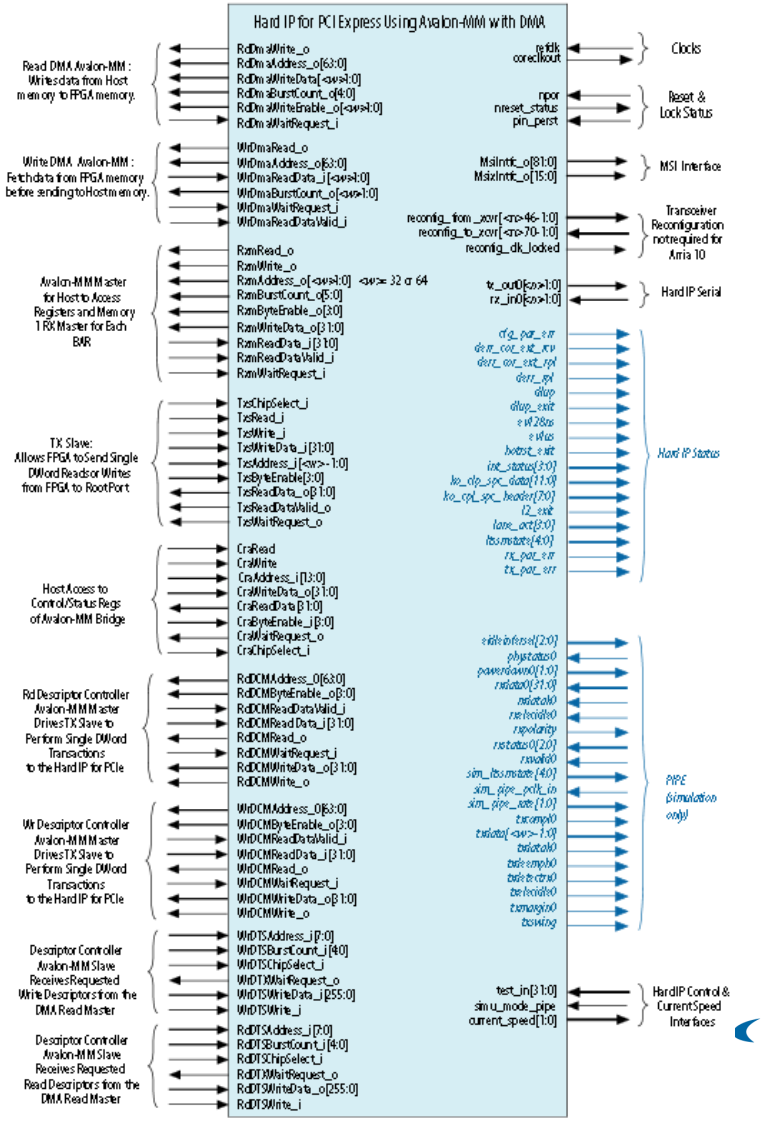
- pin_perstn
  - Reset from device I/O pin
  - One for each Hard IP block
  - Required for CvP
- npor
  - Reset from internal logic
  - Drive using logic OR of pin_perstn and internal reset signal
- reset_status
  - Indicates that the Hard IP clock is in reset
- Function Level reset
  - SR-IOV IP only
  - Allows individual resets per function



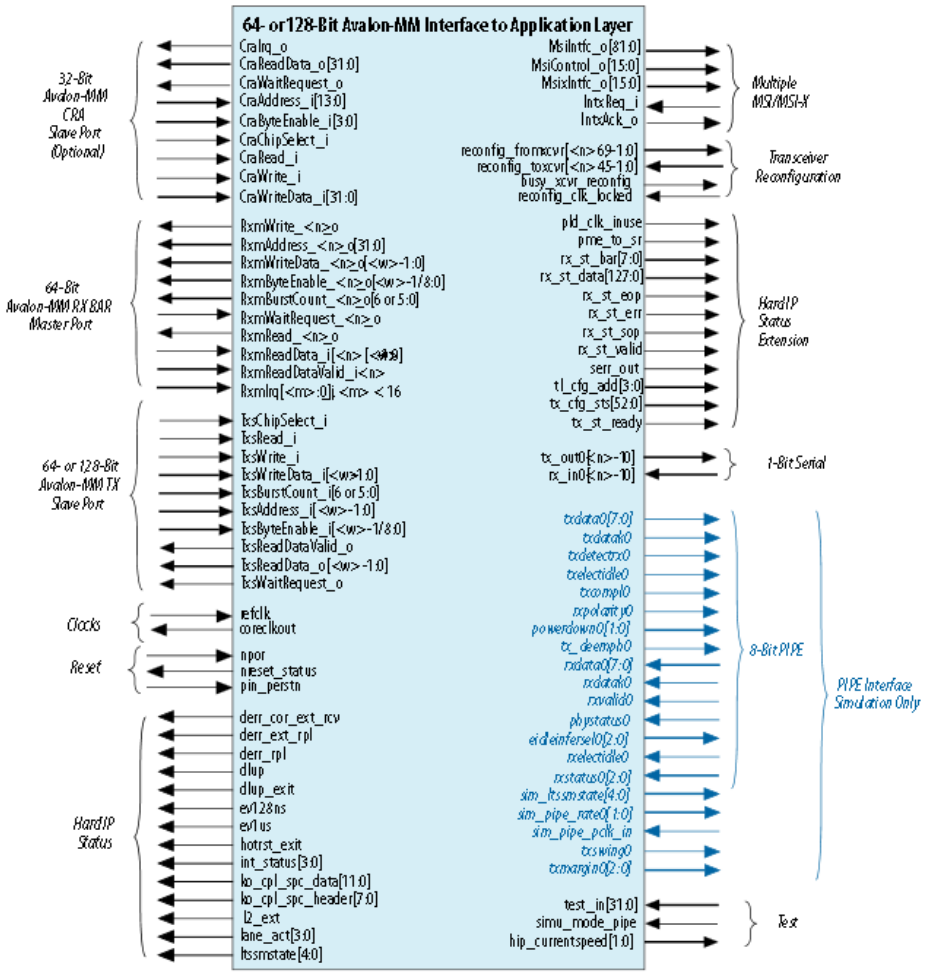


Connecting the HIP – Avalon-MM Component Symbols

Avalon-MM with DMA



Avalon-MM



Slight variations exist between these symbols and the symbols for other devices

Avalon-MM Component Interfaces

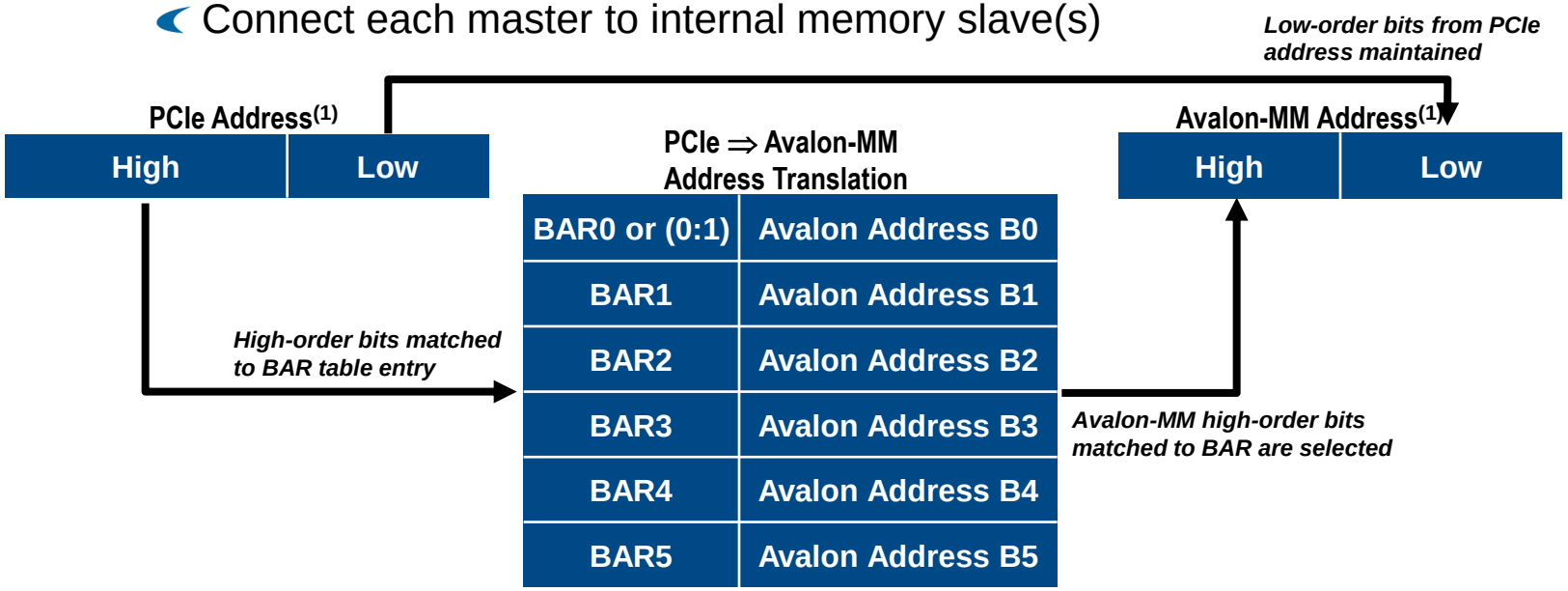
- Avalon-MM RX master
- Avalon-MM TX slave
- Control Register Access (CRA) Interface
- Clocks
  - Subset of Avalon-ST Component signals
- Reset & Lock Status
  - Subset of Avalon-ST Component signals
- Transceiver Reconfiguration
  - Same as Avalon-ST Component

Avalon-MM Component – Transaction Interfaces

- Avalon-MM master (Hard IP RX data)
  - 64 or 128-bit data bus
  - 32 or 64-bit address bus
    - Address translation required if 32-bit addressing selected
  - One complete interface per BAR (up to 6)
  - Requests from PCI Express fabric to application logic
    - PCI Express packets received are converted into Avalon-MM master read/write transactions
    - Avalon-MM read responses are converted into read completions
- Avalon-MM slave (Hard IP TX data)
  - 64 or 128-bit data bus
  - 32 or 64-bit address bus
    - Address translation required if 32-bit addressing selected
  - Requests from application logic to PCI Express fabric
    - Avalon-MM read/write transactions converted into PCI Express packets
    - Read completion packets are converted into Avalon-MM read responses
  - Not available in Completer-only implementation

PCI Express ⇒ Avalon-MM Address Translation (RX)

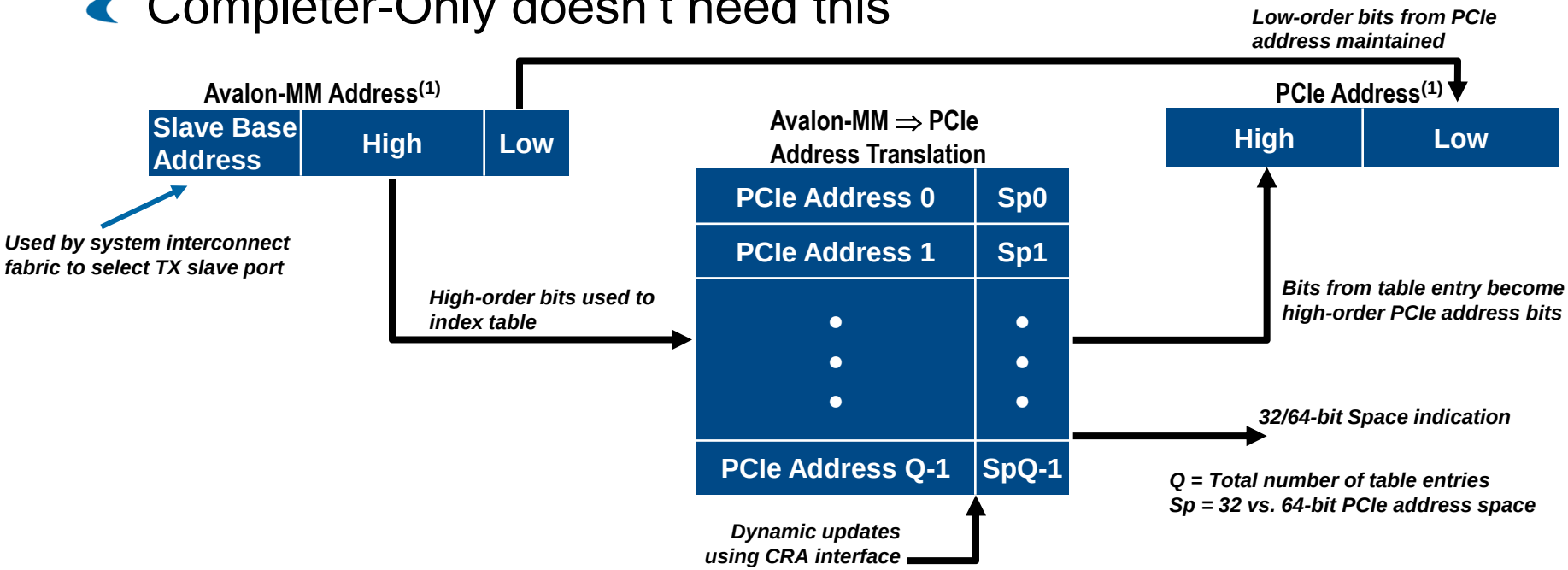
- Maps 64-bit PCIe addresses to local Avalon-MM 32-bit addresses
- For each bar enabled (up to 6) during IP customization
  - Separate translation table is created
  - Separate RX Avalon-MM master interface is created
    - Connect each master to internal memory slave(s)



1. Number of bits used for high/low portion of PCIe & Avalon-MM addressing determined by total memory

Avalon-MM ⇒ PCI Express Address Translation (TX)

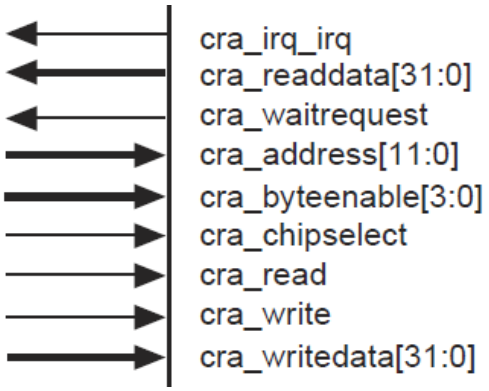
- Maps Avalon-MM addresses to PCI Express addresses
- Setup during IP customization
- Required only if PCIe request packets originate in Qsys subsystem (TX Slave Module)
- Completer-Only doesn't need this

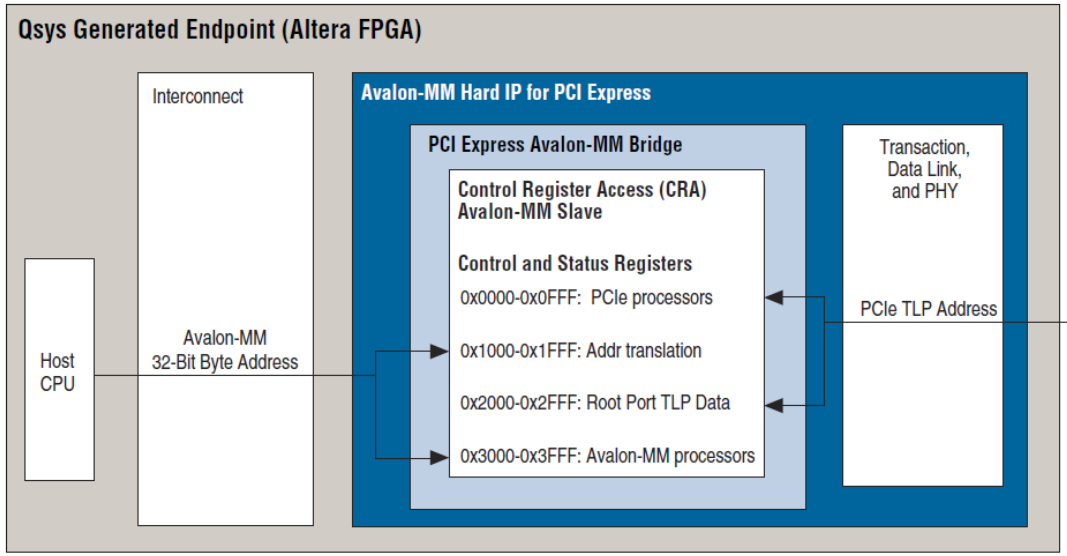


1. Number of bits used for high/low portion of PCIe & Avalon-MM addressing determined by total accessible PCIe address space

Avalon-MM Component – CRA Interface

- Access to the Avalon-MM bridge registers from
  - Upstream PCI Express devices
  - Avalon-MM masters
- Required for MSI for dynamic address translation
- Not available in Completer-only Single DWORD implementation
- Avalon-MM slave
  - 32-bit data
  - 16k Address space
  - Non-bursting

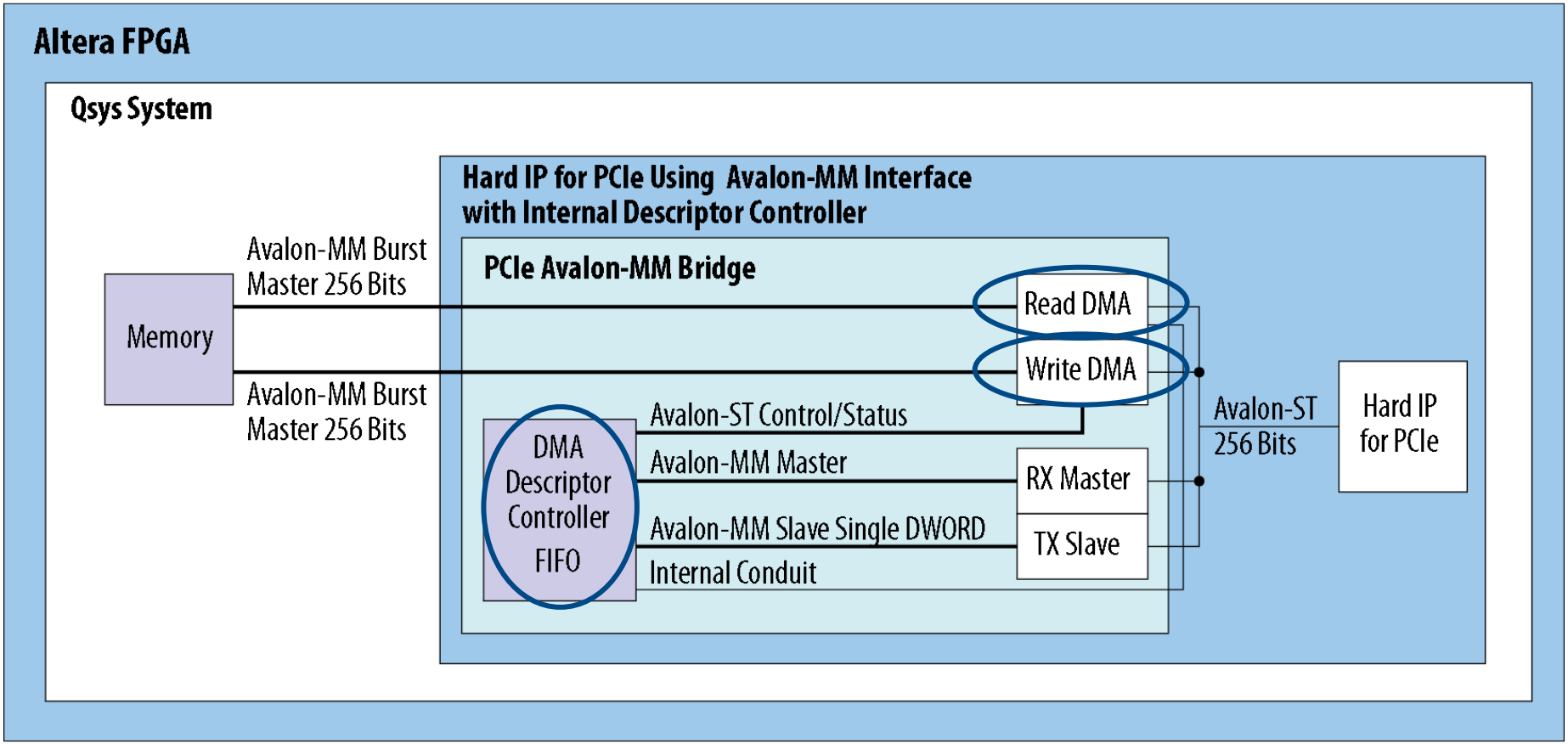




Avalon-MM Component - Bridge Register Space

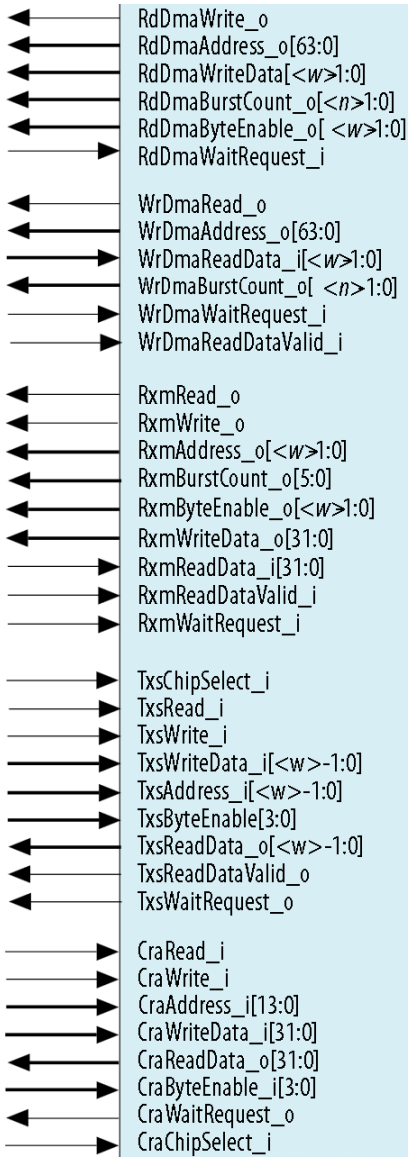
- PCI Express Processors Only (0x0000 to 0x0FFF)
  - PCI Express Interrupt enable controls
  - Read / Write access to bridge mailbox registers
- Translation Tables (0x1000 to 0x1FFF)
  - Avalon-MM to PCI Express translation tables
  - PCI Express processor or Avalon-MM processor
- Root Port request registers (0x2000 to 0x2FFF)
  - Embedded processor, programs these registers to send the data to send Configuration TLPs, I/O TLPs, single dword Memory Reads and Write request, and receive interrupts from an Endpoint.
- Avalon-MM processors only (0x3000 to 0x3FFF)
  - Avalon-MM interrupt enable controls, write access to the Avalon-MM-to-PCI Express mailbox registers, and read access to PCI Express Avalon-MM bridge mailbox registers.

Avalon-MM with DMA Component Interfaces



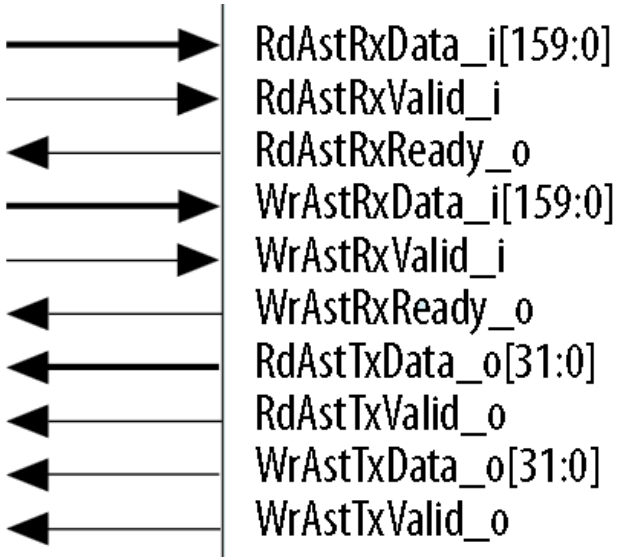
Avalon-MM DMA Component – Standard Interfaces

- Read DMA master (*RdDma**)
  - Connect to Avalon-MM slaves transfer to them contents of PCIe packets received from upstream
  - Connect to Avalon-MM slave descriptor table retrieved from host
- Write DMA master (*WrDma**)
  - Connect to Avalon-MM slaves to their transfer contents upstream as PCIe TLPs
- Avalon-MM RX master (*Rxm**)
  - Translates single dword read and write TLPs received from the host and translates them to Avalon-MM requests
- Avalon-MM TX slave (*Txs**)
  - Avalon-MM masters can use this slave port for non-DMA access to PCI Express memory space
- CRA Avalon-MM slave interface (*Cra**)
  - Avalon-MM and PCIe masters use this interface to access bridge registers



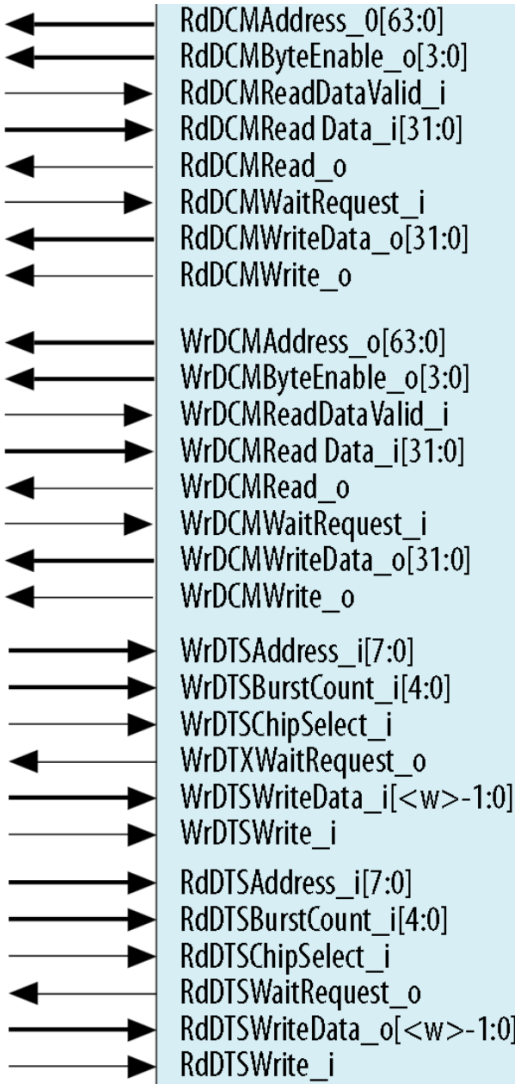
Avalon-MM DMA Component - External DMA Controller Interfaces

- Read DMA Avalon-ST Sink (RdAstRx*)
  - DMA engine receives descriptor instructions from descriptor controller
- Read DMA Avalon-ST Source (RdAstTx*)
  - DMA engine sends status to descriptor controller
- Write DMA Avalon-ST Sink (WrAstRx*)
  - DMA engine receives descriptor instructions from descriptor controller
- Write DMA Avalon-ST Source (WrAstTx*)
  - DMA engine sends status to descriptor controller



Avalon-MM DMA Component - Embedded DMA Controller Interfaces

- Read Descriptor Controller master (*RdDCM**)
  - Provides status information from Read Descriptor Controller
  - Connect to the Avalon-MM TX slave interface to send status information upstream
- Write descriptor controller master (*WrDCM**)
  - Provides status information from Write Descriptor Controller
  - Connect to the Avalon-MM TX slave interface to send status information upstream
- Read descriptor table slave (*RdDTS**)
  - Stores descriptors for the Read Descriptor Controller
  - Connect to the Read DMA master interface
- Write descriptor controller slave (*WrDTS**)
  - Stores descriptors for the Write Descriptor Controller
  - Connect to the Read DMA master interface



Building a Hard IP for PCI Express Design Section Agenda

- Hard IP for PCI Express Features
- Customizing the Hard IP for PCI Express Settings
- Connecting the Hard IP for PCI Express Interfaces
- Design Implementation

.QSYS Source File

- Each .QSYS file represents a single Qsys system (components, connections and parameterizations)
- All other files for Qsys system created during system generation
- Can use multiple Qsys files in a single Quartus Prime project (hierarchy)

Qsys Output Files (“Legacy”)

- Top files
  - <system_name>**.sopcinfo**
    - XML file describing Qsys system used for software development tools
  - <system_name>**.bsf**
    - Symbol file for Quartus II schematic editor *(optional)*
  - <system_name>/<system_name>**.html**
    - Generation report including output files generated, component list, memory map, etc.
- Files for synthesis
  - Located in **<system_name>/synthesis** folder
    - <system_name>**.qip**
      - Script file that adds all files needed for synthesis to Quartus II project
    - <system_name>**.[v|vhd]**
      - System top-level file connecting all components together
    - Submodule files for synthesis
      - Located in **<system_name>/synthesis/submodules** folder
      - Combination of Verilog, SystemVerilog, and/or VHDL files representing system
      - **.sdc** timing constraint files may also be generated

Qsys Output Files ("Standard," Arria® 10 Devices & Later)

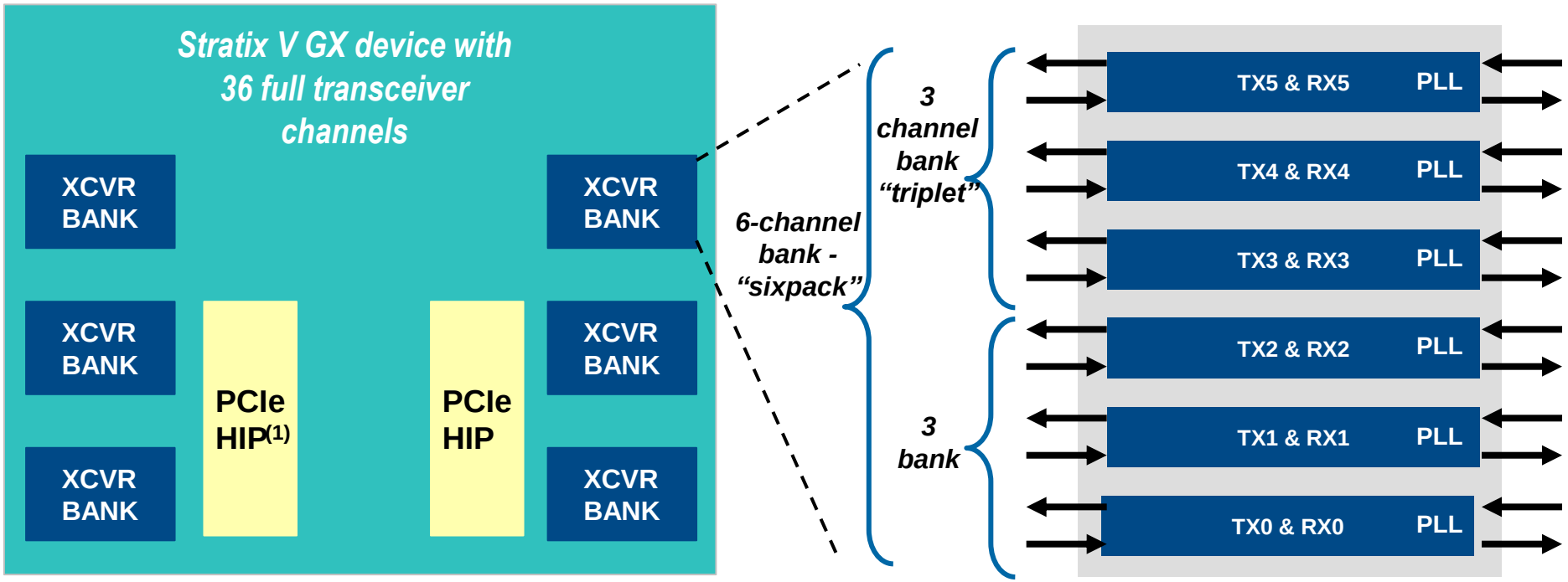
- Top files
  - <system_name>**.sopcinfo**
    - XML file describing Qsys system used for software development tools
  - <system_name>/<system_name>**.bsf**
    - Symbol file for Quartus II schematic editor *(optional)*
  - <system_name>/<system_name>**.html**
    - Generation report including output files generated, component list, memory map, etc.
  - <system_name>/<system_name>**.qip**
    - Script file that adds all files needed for synthesis to Quartus II project
- Files for synthesis
  - Located in **<system_name>/synth** folder
    - <system_name>**.[v|vhd]**
      - System top-level file connecting all components together
  - Component subfolders contain submodule files for synthesis
    - Unique **synth** folder in each component folder
    - Combination of Verilog, SystemVerilog, and/or VHDL files representing each component
    - **.sdc** timing constraint files may also be generated

Assigning PCI Express Transceivers

- Place PCI Express lanes carefully
  - Transceiver channels are the physical implementation of PCI Express lanes
  - Clocking resources, bonding channels, and fitting restrict transceiver channel placement
  - Channels are placed by assigning the locations of the serial input (*rx_in*) and output (*tx_out*) pins
    - Use any supported I/O placement methodology (see I/O System Design online training)
  - The next few slides describe channel placement requirements
- Make I/O standard assignments
  - Assign Current Mode Logic (CML) standard to the serial input pins
  - Assign High-Speed Differential IO standard to serial output pins
  - Assign CML or HCSL standard to reference clock input pin
  - Other required PMA analog settings handled automatically

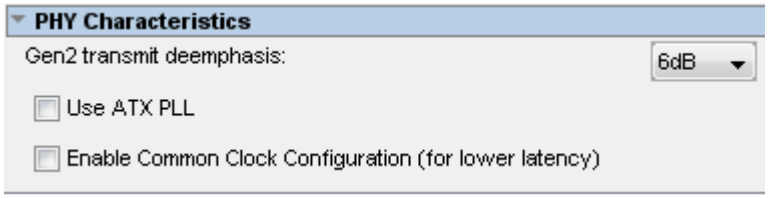
General Transceiver Arrangement

- Each Hard IP for PCI Express can connect to up to 2 transceiver banks
- Designers must ensure they choose lane locations (transceivers) that connect to the same Hard IP block



(1) HIP = Hard IP

Choose the TX PLL

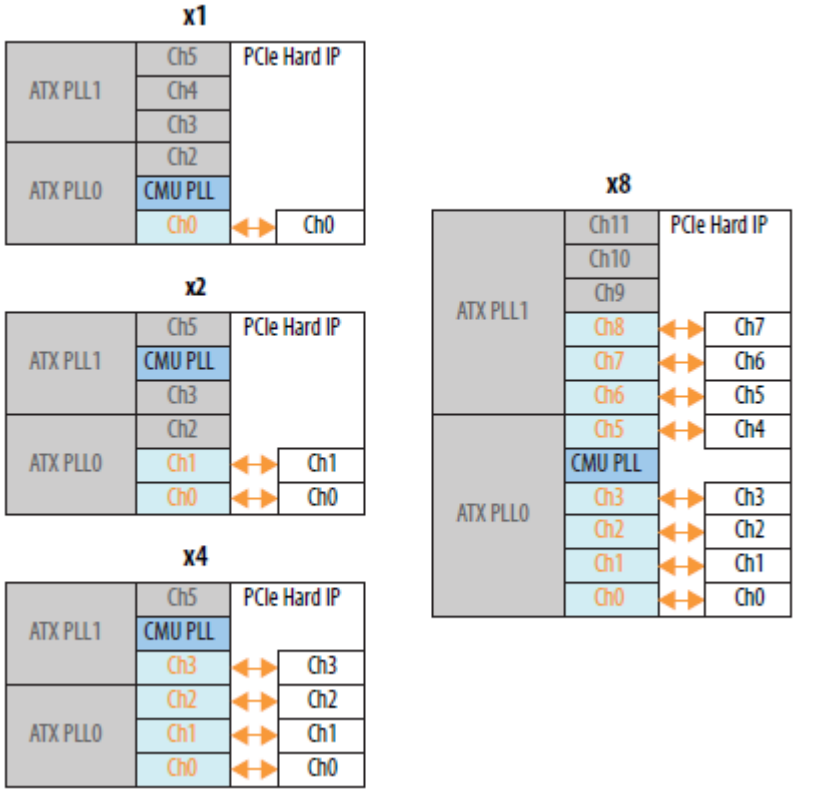


- Cyclone V and Arria V devices
  - Only clock management unit (CMU) PLL available
- Stratix V and Arria V GZ devices
  - Gen1 or Gen 2
    - Choose auxiliary transmit (ATX) PLL if CvP is disabled
    - Choose CMU PLL if CvP is enabled
  - Gen3
    - Requires use of both an ATX (Gen3 rate) and a CMU PLL (Gen1/Gen2 rates)
    - Link established at Gen 1 rate then negotiated up to Gen2 and Gen3
- Arria 10 devices
  - Gen1 and/or Gen2
    - Choose fractional PLL (fPLL)
  - Gen3
    - Requires use of both an ATX PLL (Gen3 rate) and an fPLL (Gen1/Gen2 rates)
    - Link established at Gen 1 rate then negotiated up to Gen2 and Gen3

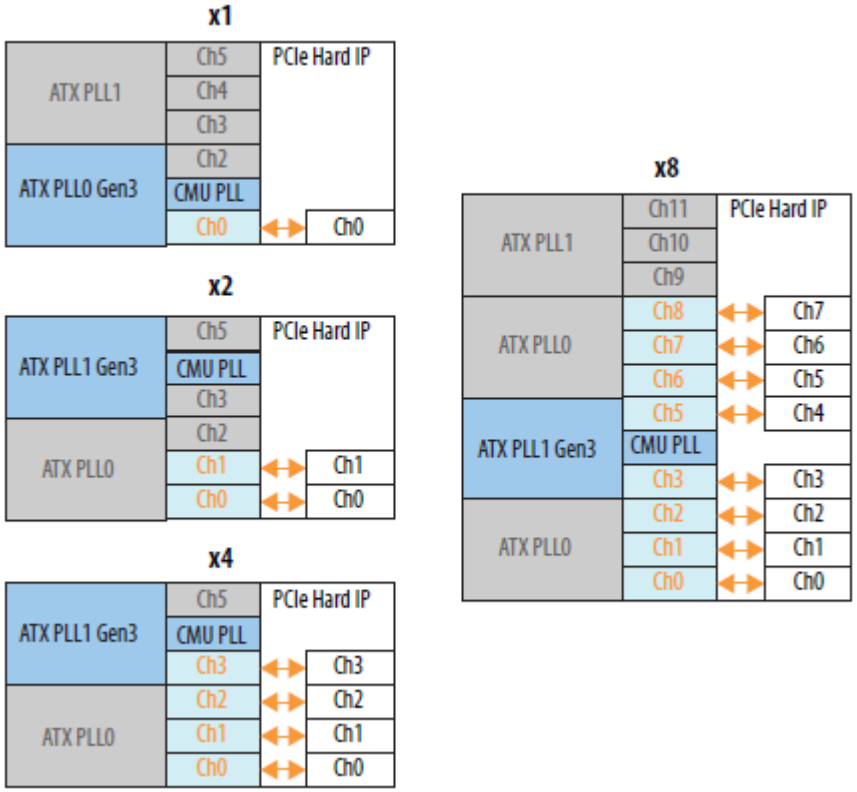
See transceiver section of device handbook or transceiver user guide for detailed description of PLLs and their differences.

28-nm Channel Placement Guidelines

Gen1/Gen2(1)

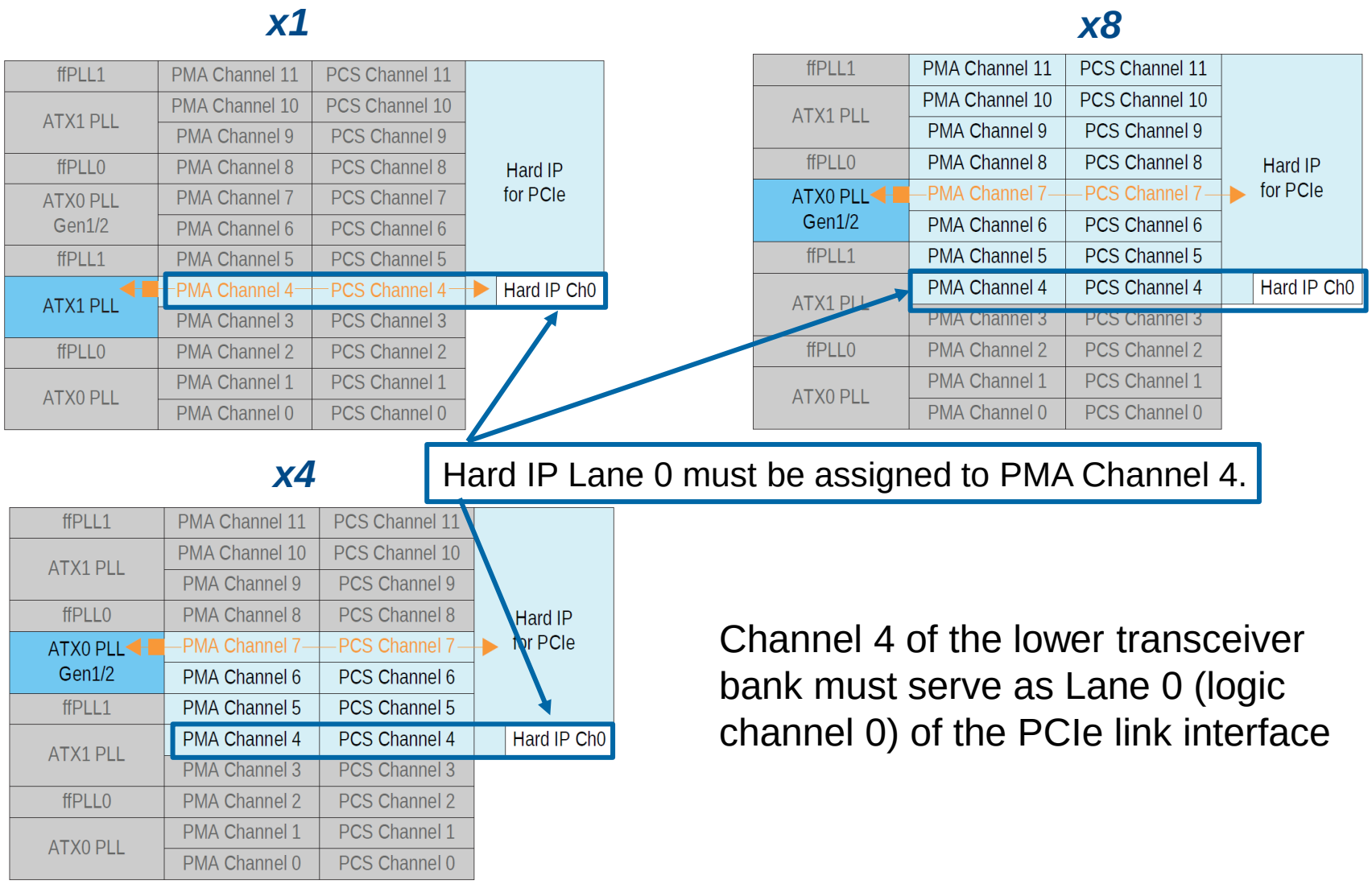


Gen3



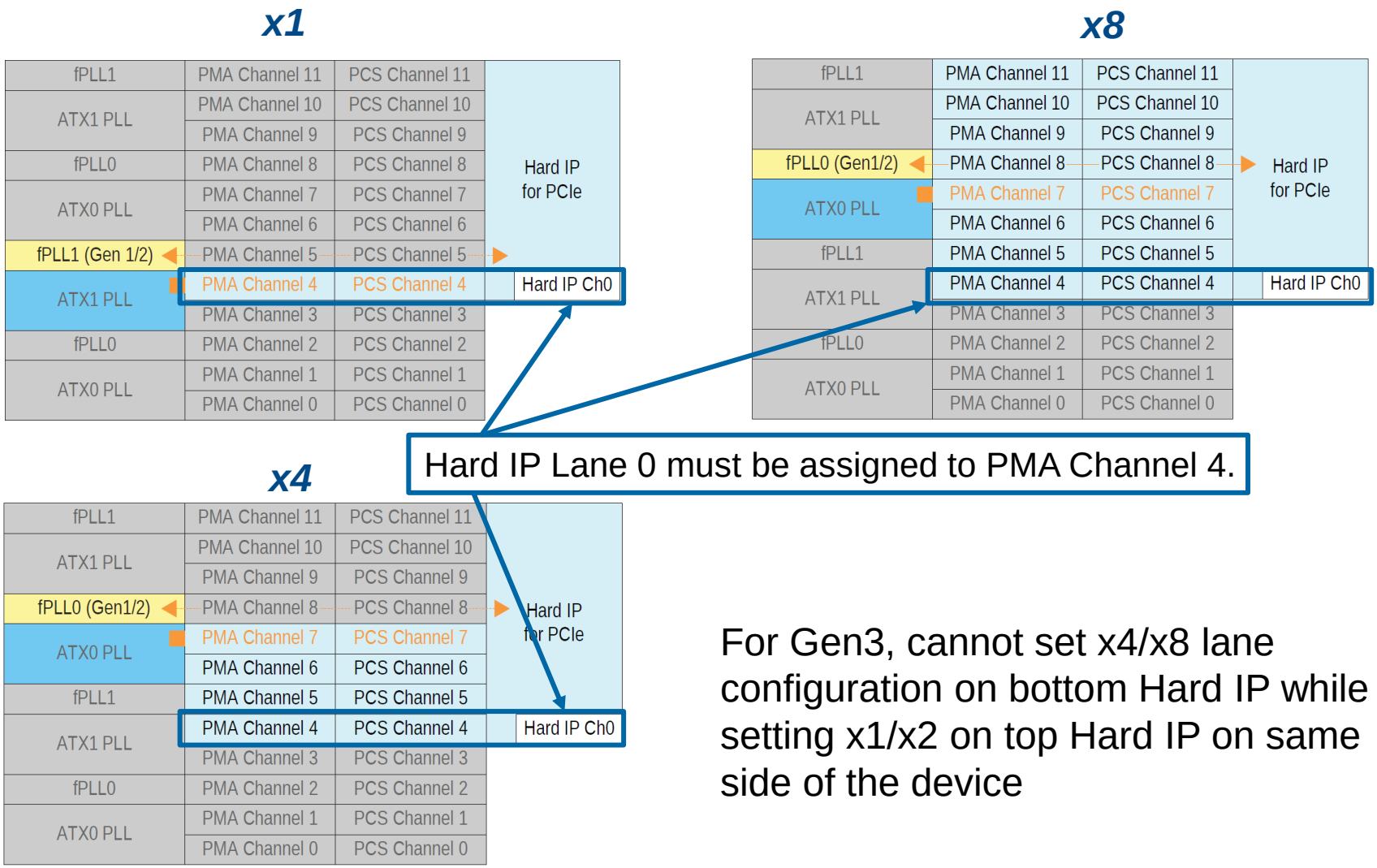
1. For Gen1/Gen2 ATX PLL usage, use the placements for Gen3 removing the CMU PLL.

Arria 10 Devices – Channel Placements Gen 1/2 – ATX PLL



Channel 4 of the lower transceiver bank must serve as Lane 0 (logic channel 0) of the PCIe link interface

Arria 10 Devices – Channel Placements Gen 3



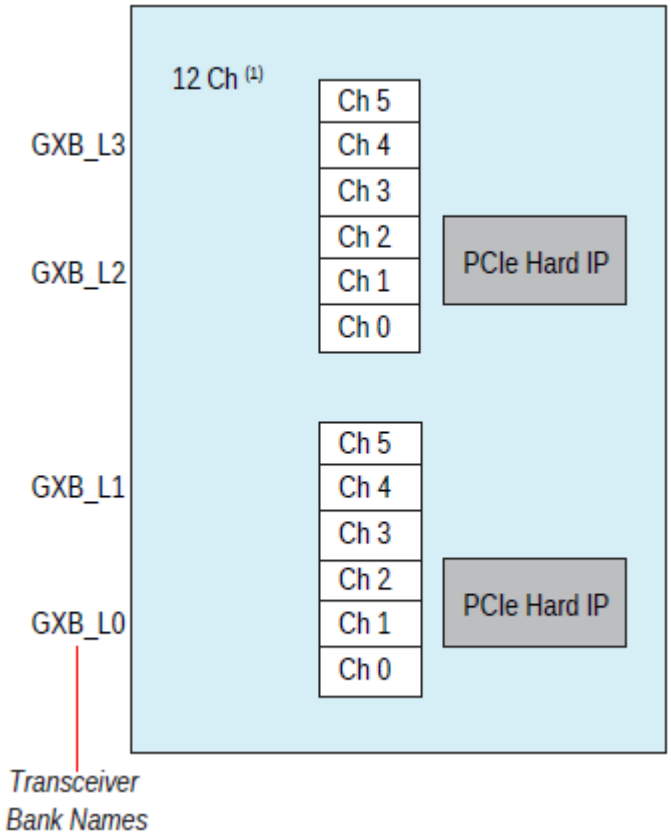
For Gen3, cannot set x4/x8 lane configuration on bottom Hard IP while setting x1/x2 on top Hard IP on same side of the device

Input Reference Clocks

- Provide 100 (or 125 MHz) reference clocks for training transceiver PLLs
  - Arria 10 devices support 100 MHz clock only
- Recommended clock sources (lowest to highest jitter)
  1. Dedicated reference clock pin closest to the TX PLL (direct path)
  2. Dedicated reference clock pin within transceiver bank
  3. Dedicated reference clock pin outside of transceiver bank on the same side of the device
- For more information about channel placement and clocking refer to “Transceiver Clocking and Channel Placement Guidelines” in the device handbook or user guide

Transceiver Banks – Device Handbook

- Transceiver bank information is located in transceiver section of the device handbook
  - Cyclone V Device Handbook
  - Arria V Device Handbook
  - Stratix V Device Handbook
  - Arria 10 Device Handbook
- Transceivers are named in banks of three channels
- In Cyclone V devices, bank names are numbered beginning with GXB_L0



Identifying Package Pins

- Transceiver and reference clock pin assignments are also shown in the device pin information document

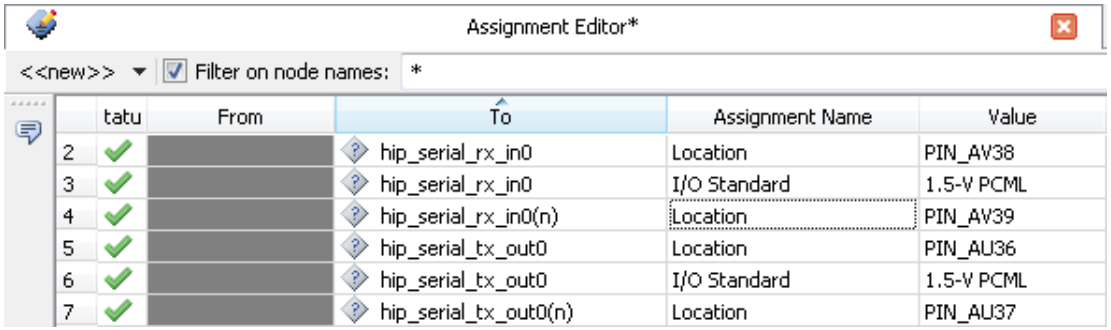
Bank Number	VREF	Pin Name/Function	U484
GXB_L1		REFCLK1Ln	G4
GXB_L1		REFCLK1Lp	F5
GXB_L1		GXB_TX_L4n	E1
GXB_L1		GXB_TX_L4p	E2
GXB_L1		GXB_RX_L4p,GXB_REFCLK_L4p	G2
GXB_L1		GXB_RX_L4n,GXB_REFCLK_L4n	G1
GXB_L1		GXB_TX_L3n	J1
GXB_L1		GXB_TX_L3p	J2
GXB_L1		GXB_RX_L3p,GXB_REFCLK_L3p	L2
GXB_L1		GXB_RX_L3n,GXB_REFCLK_L3n	L1
GXB_L0		GXB_TX_L2n	N1
GXB_L0		GXB_TX_L2p	N2
GXB_L0		GXB_RX_L2p,GXB_REFCLK_L2p	R2
GXB_L0		GXB_RX_L2n,GXB_REFCLK_L2n	R1
GXB_L0		GXB_TX_L1n	U1
GXB_L0		GXB_TX_L1p	U2
GXB_L0		GXB_RX_L1p,GXB_REFCLK_L1p	W2
GXB_L0		GXB_RX_L1n,GXB_REFCLK_L1n	W1
GXB_L0		GXB_TX_L0n	Y3
GXB_L0		GXB_TX_L0p	Y4
GXB_L0		GXB_RX_L0p,GXB_REFCLK_L0p	AA2
GXB_L0		GXB_RX_L0n,GXB_REFCLK_L0n	AA1
GXB_L0		REFCLK0Lp	V4
GXB_L0		REFCLK0Ln	U4

Assigning Pin Locations and IO Standard

- Pin Planner

Node Name	Location	I/O Bank	VREF Group	I/O Standard
hip_serial_rx_in0	PIN_AV38	B0L		1.5-V PCML
hip_serial_rx_in0(n)	PIN_AV39	B0L		1.5-V PCML
hip_serial_tx_out0	PIN_AU36	B0L		1.5-V PCML
hip_serial_tx_out0(n)	PIN_AU37	B0L		1.5-V PCML

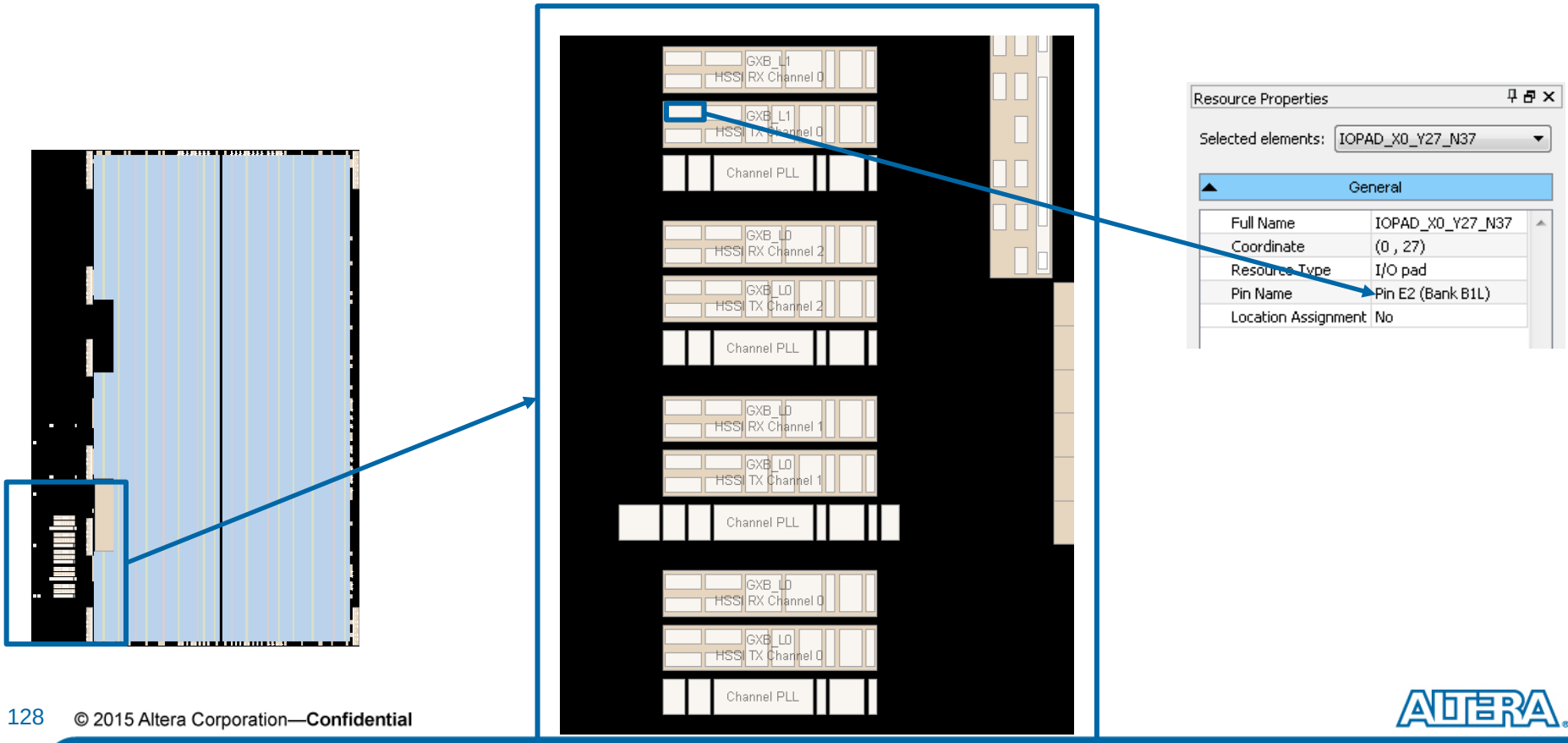
- Assignment Editor



- QSF file
  - set_location_assignment PIN_AV39 -to "hip_serial_rx_in0(n)"
  - set_location_assignment PIN_AV38 -to hip_serial_rx_in0
  - set_location_assignment PIN_AU37 -to "hip_serial_tx_out0(n)"
  - set_location_assignment PIN_AU36 -to hip_serial_tx_out0
  - set_instance_assignment -name IO_STANDARD "1.5-V PCML" -to hip_serial_rx_in0
  - set_instance_assignment -name IO_STANDARD "1.5-V PCML" -to hip_serial_tx_out0

Viewing Transceivers – Chip Planner

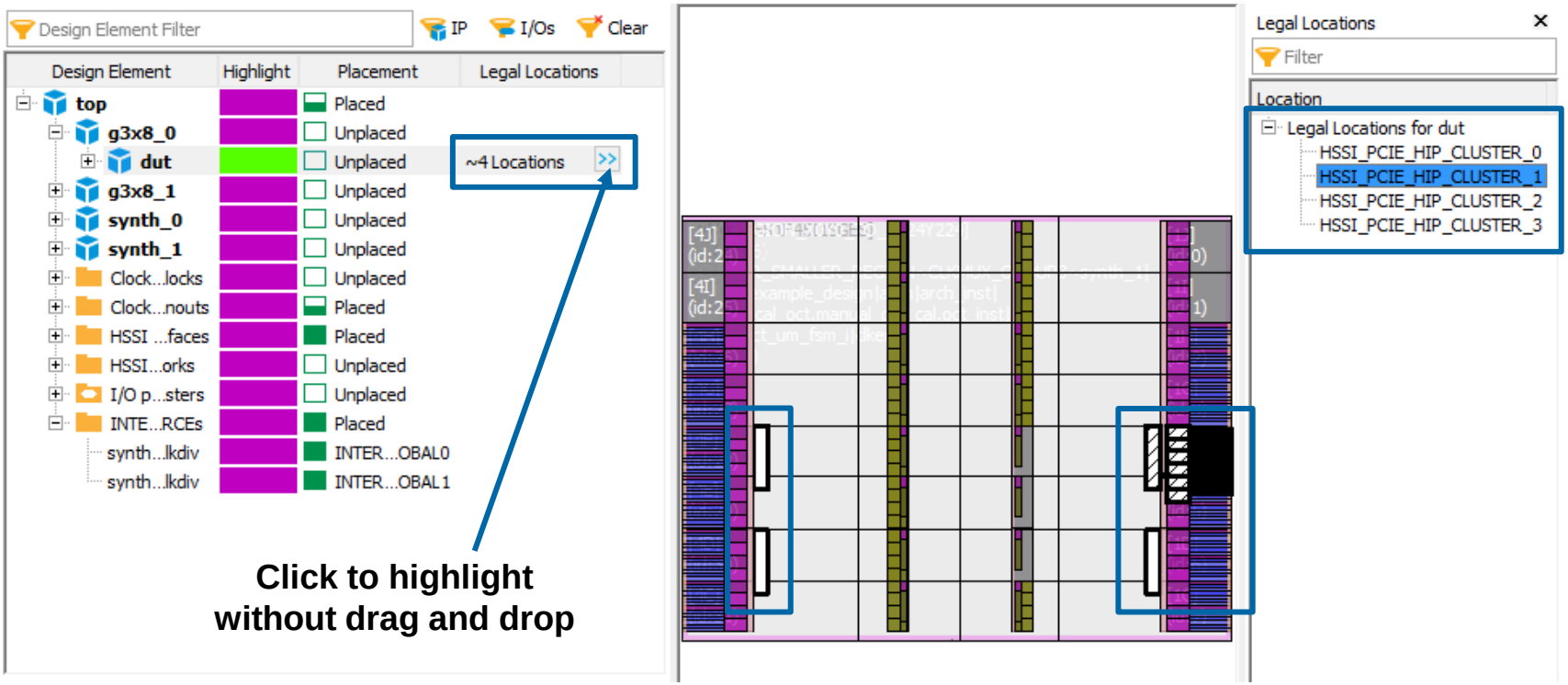
- Transceiver bank names can be found using Chip Planner by zooming into the transceiver section
- The transceiver pin location can be found in the Resource Properties window after selecting the IO pad



Viewing Transceivers – BluePrint(1)

- View list of legal locations for a design element without clicking and dragging

Legal locations listed and highlighted in both Chip and Package views



1. For more information about BluePrint , view the online training *Fast & Easy I/O System Design with BluePrint*

Creating PCI Express Links Using FPGAs

Simulating a Hard IP for PCI Express Design

Simulating a PCI Express Design Section Agenda

- PCIe simulation design example and testbench
- Simulation Driver
- Changes for PCIe simulation
- Simulation hierarchy
- Simulation with the ModelSim® tool
- PCI Express BFM Choices

Not Covered

- Root Port
  - Focus on Endpoint configuration
- Gen3 simulation
- Avalon-MM PCI Express component example
  - Depends on the design and its memory map

Hard IP for PCI Express Simulation

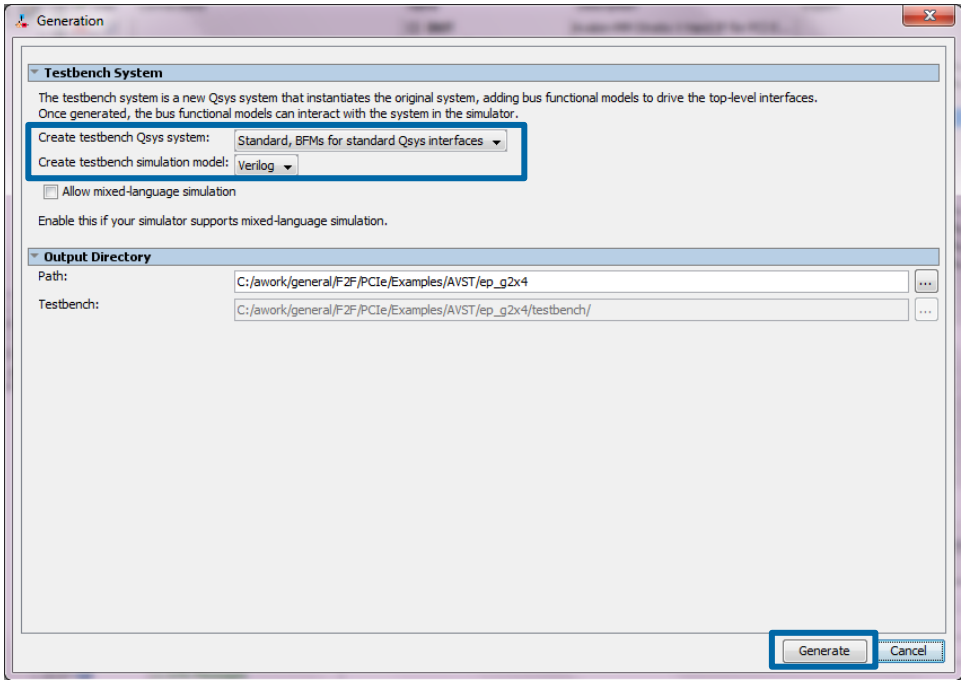
- Enable testbench generation in Qsys Generate dialog box
  - Simulation model generated for Hard IP (Root Port or Endpoint)
  - Testbench created to exercise Hard IP (Root Port or Endpoint)
  - Script files generated to automate running simulation tool
- Locate testbench design in *testbench* subdirectory
- Launch simulation tool using generated scripts
  - e.g. ModelSim tool:
    - `do msim_setup.tcl`
    - `ld_debug`
    - `run -all`
- Analyze design behavior using testbench
  - Uses a generated bus functional model (BFM) to emulate the other end of the PCIe link during simulation
  - Uses configuration routine to set up configuration registers
  - Uses Chaining DMA application module to read and write memory transactions between host and FPGA

PCIe Simulation Design Example

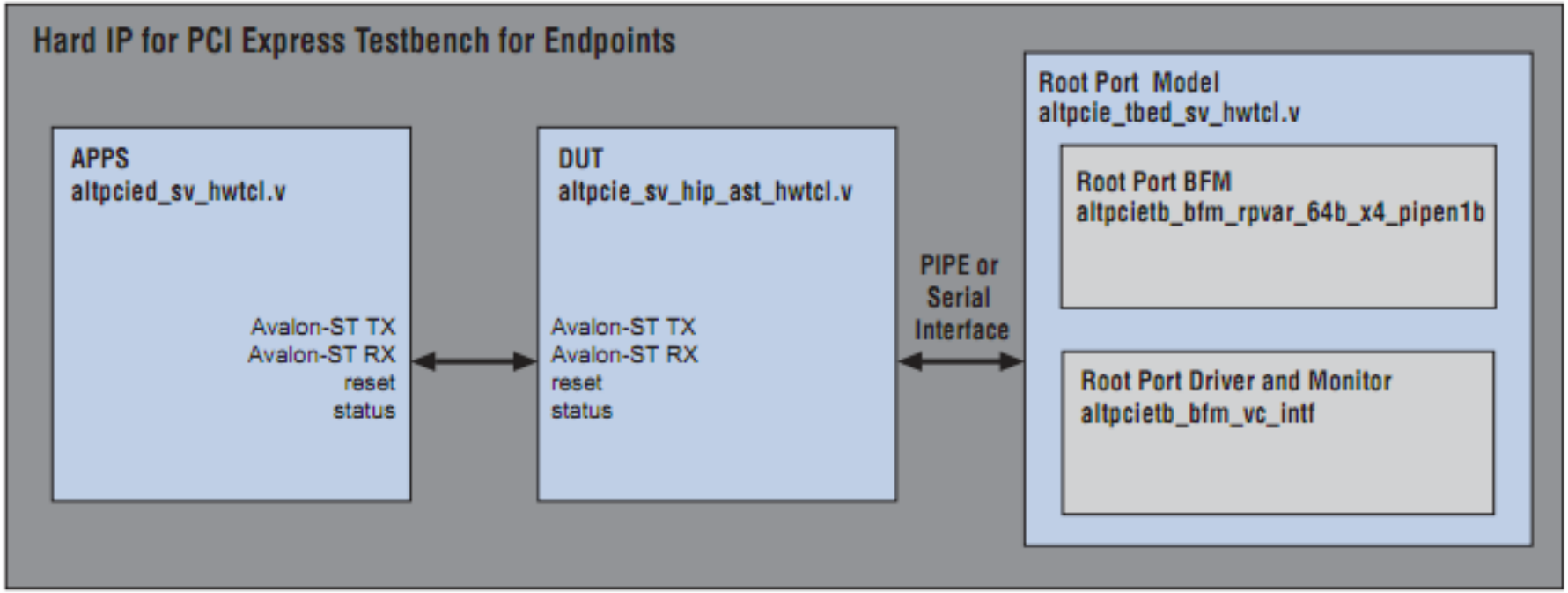
- Fully functioning application-layer design example built in Qsys
  - Endpoint design example & testbench
  - Root Port design example & testbench
- Generated by Qsys or IP Catalog
  - Using user values chosen PCIe IP parameter editor
- Located in
  - Includes simulation model for Qsys system including Hard IP instantiation
- May be used for initial hardware verification

Generate Testbench System

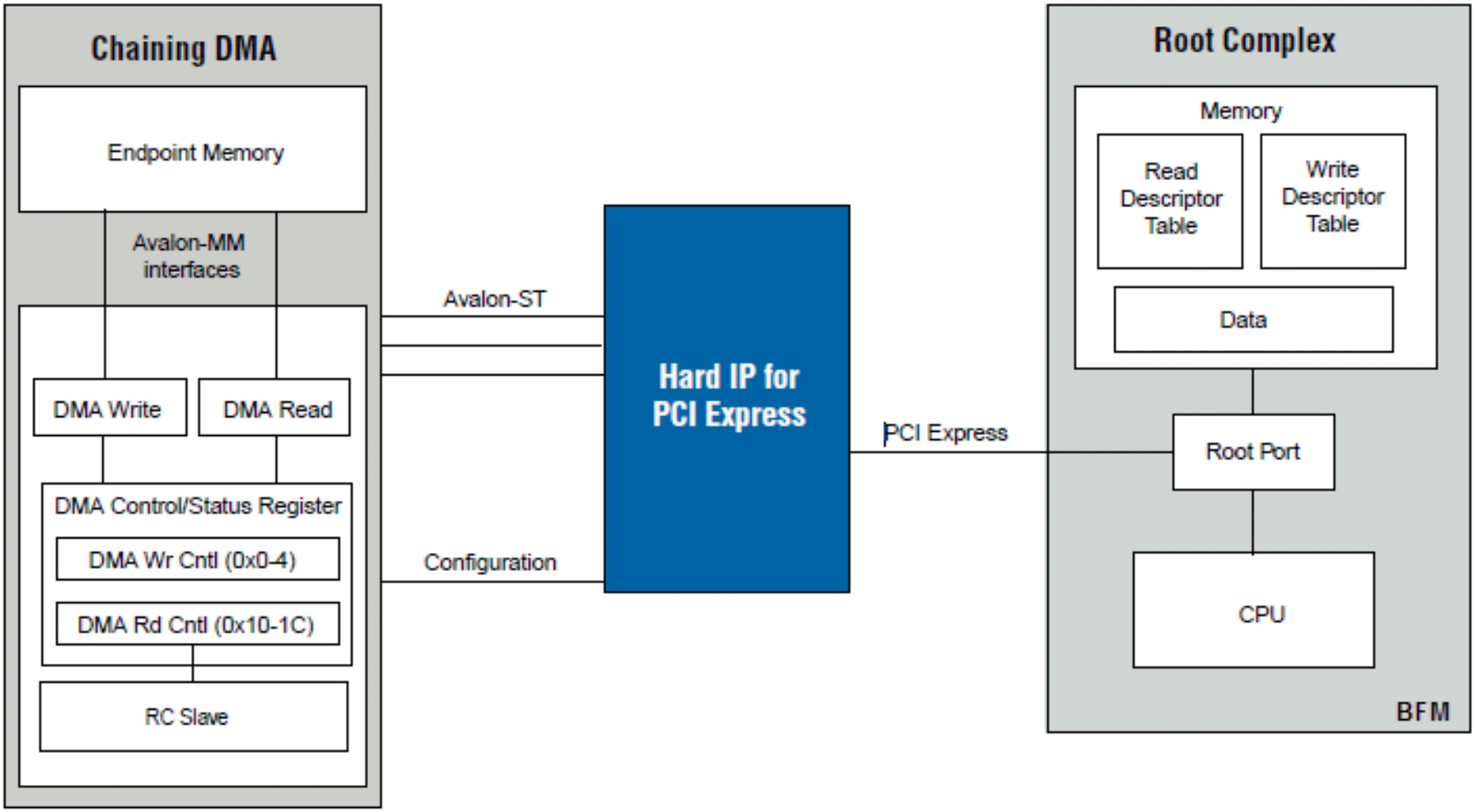
- Generate -> Generate Testbench System
- Set *Create testbench Qsys system* to
  - **Standard, BFMs for standard Avalon interfaces**
- Set *Create testbench simulation model* to
  - **Verilog/VHDL**
- Click Generate



PCIe Endpoint Testbench Example



PCIe Endpoint Testbench Example (2)



Simulation Testbench Behavior

- Uses configuration routine to set up configuration registers
- Uses a generated BFM to emulate the other end (Root Port) of the PCIe link during simulation
- Uses a Test Driver Module to initiate PCIe transactions
  - Made up of Verilog procedures
- Uses Chaining DMA application module to read and write memory transactions across PCIe link

Qsys Testbench Output Files (“Legacy”)

- Top files
  - <system_name>.sopcinfo
    - XML file describing Qsys system used for software development tools
  - <system_name>/<system_name>.html
    - Generation report for original system including output files generated, component list, memory map etc.
- Files for simulation
  - Located in <system_name>/testbench folder
    - <system_name>_tb.qsys
      - Generated Qsys system that includes BFMs connected to the DUT
    - <system_name>_tb.html
      - Generation report for testbench system
    - Third-party vendor script folders
      - Separate folders for most popular simulation tools
      - Scripts and libraries included to help automate setup and running of simulation
    - Submodule files for simulation
      - Located in <system_name>/testbench/<system_name>_tb/simulation/submodules folder
      - Combination of Verilog, SystemVerilog, and/or VHDL files representing simulation models of system components

Qsys Testbench Output Files ("Standard," Arria® 10 Devices & Later)

- Top file
  - <system_name>.sopcinfo
    - XML file describing Qsys system used for software development tools
- Files for simulation
  - Located in <system_name>_tb folder
    - <system_name>_tb.qsys
      - Generated Qsys system that includes BFMs connected to the DUT
    - <system_name>.html & <system_name>_tb.html
      - Generation report for original and testbench systems including output files generated, component list, memory map etc.
  - Located in <system_name>_tb/<system_name>_tb folder
    - Component subfolders contain submodule files for simulation
      - Unique sim folder in each component folder
      - Combination of Verilog, SystemVerilog, and/or VHDL files representing simulation models of system components
    - Third-party vendor script folders in sim folder
      - Separate folders for most popular simulation tools
      - Scripts and libraries included to help automate setup and running of simulation

The Simulation Driver

- To customize the testbench, modify the simulation driver file
- Locate the driver file
  - search for a file with “driver” in the name
  - In this example, the file is called altpcietb_bfm_driver_chaining.v
- Search for the last “main” in simulation driver file
- Pseudo code for “main”
  - Enumerate RP and EP Configuration Space (ebfm_cfg_rp_ep())
  - Check for valid DMA BARs (find_mem_bar())
  - If valid DMA BARs and USE_CDMA =1, execute DMA tests
    - chained_dma_test()
  - Check for valid target BARs (find_mem_bar())
  - If valid target BAR and USE_TARGET =1, execute target test
    - downstream_loop() => default
  - Display test status
  - Print error message if error occurs or data check fails

Locations of the Root Port Driver Tasks

- Avalon-ST Gen1 and Gen2 BFM
  - **RP driver:** **altpcietb_bfm_driver_chaining.v**
  - **Basic tasks**: **altpcietb_bfm_rdwr.v**
  - **Configuration tasks**: **altpcietb_bfm_configure.v**
- Avalon-ST Gen3 BFM
  - Supports serial mode by default (for PIPE mode refer to the User Guide)
  - **RP driver:** **altpcietb_bfm_driver_rp** module defined inside **altpcietb_bfm_rp_gen3_x8.v**
  - **Basic tasks**: **altpcietb_g3bfm_rdwr.v**
  - **Configuration tasks**: **altpcietb_g3bfm_configure.v**
- Avalon-MM Gen1 and Gen2 BFM
  - Same tasks but resides in difference files

Common Changes for PCIe Simulation

- PIPE mode vs. serial mode simulation
  - PIPE mode helps to speed up simulation
  - Search for string "**serial_sim_hwtcl**" in ***<Qsys_system_name>_tb.v***
    - 1 = Serial mode (default)
    - 0 = PIPE mode
- Disable the scrambler for debugging link
  - Search for 32bit string "test_in" in ***altpcietb_bfm_top_rp.v***
  - Assign test_in[2] = 1 to disable scrambler
- Accelerate simulation
  - Shorten time outs during link training
  - Search for 32bit string "test_in" in ***altpcietb_bfm_top_rp.v***
  - Assign test_in[0] = 1 to select a smaller counter values in the core

Important PCIe Testbench Features

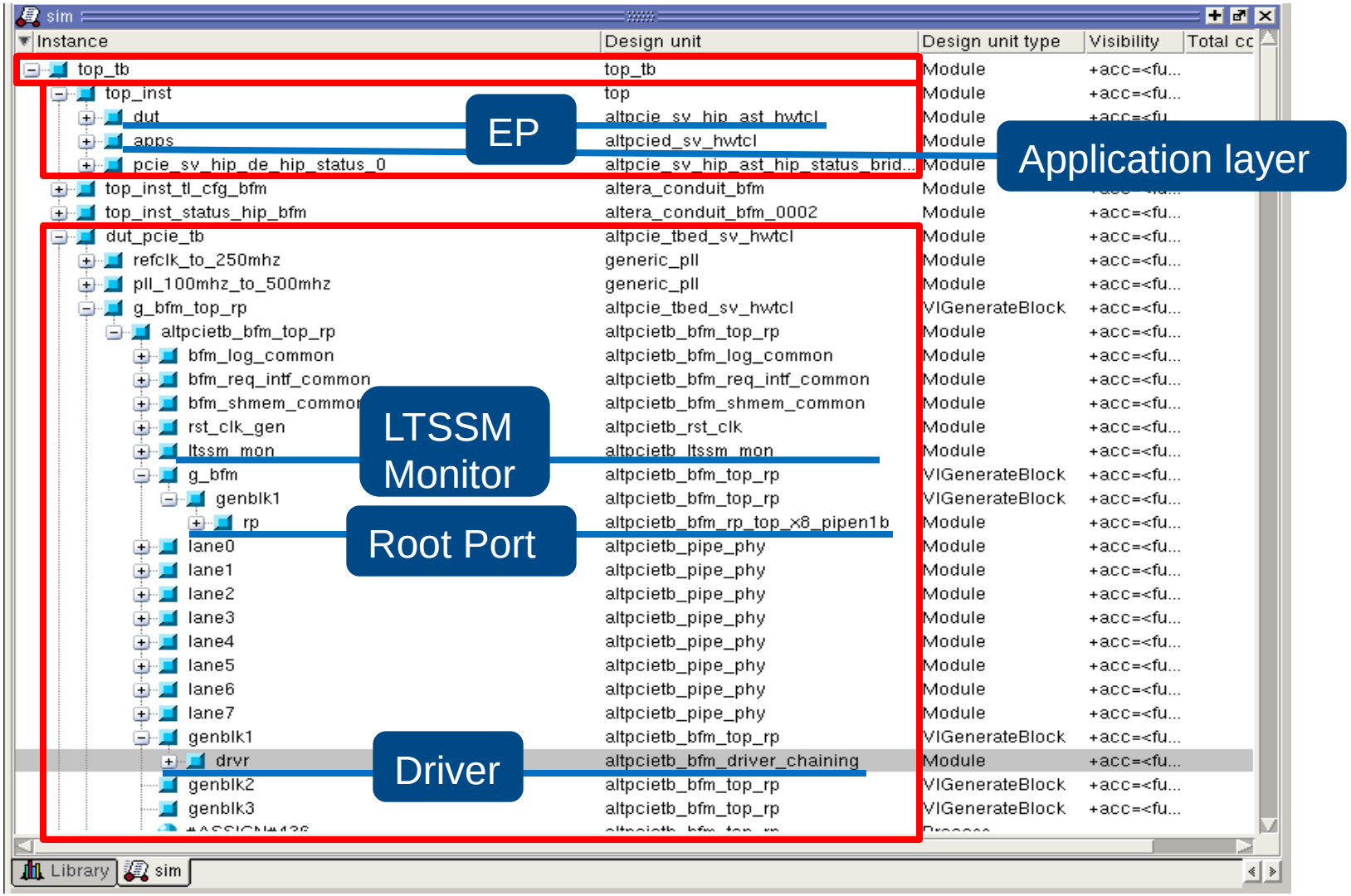
- Enable all tests
  - Search for "apps_type_hwtcl" in ***<system_name>_tb.v***
    - 1 = PCIe configuration only
    - 2 = PCIe configuration and chaining DMA tests (default)
- Disable/Enable DMA test
  - Search for "USE_CDMA" string in ***altpcietb_bfm_top_rp.v***
    - USE_CDMA = 0 => Disable CDMA
    - USE_CDMA = 1 => Enable CDMA (Default for Gen1/Gen2)
- Disable Target test
  - Search for USE_TARGET string in ***altpcietb_bfm_top_rp.v***
    - USE_TARGET = 0 => Disable
    - USE_TARGET = 1 => Enable

Running Simulation with the ModelSim(1) Tool

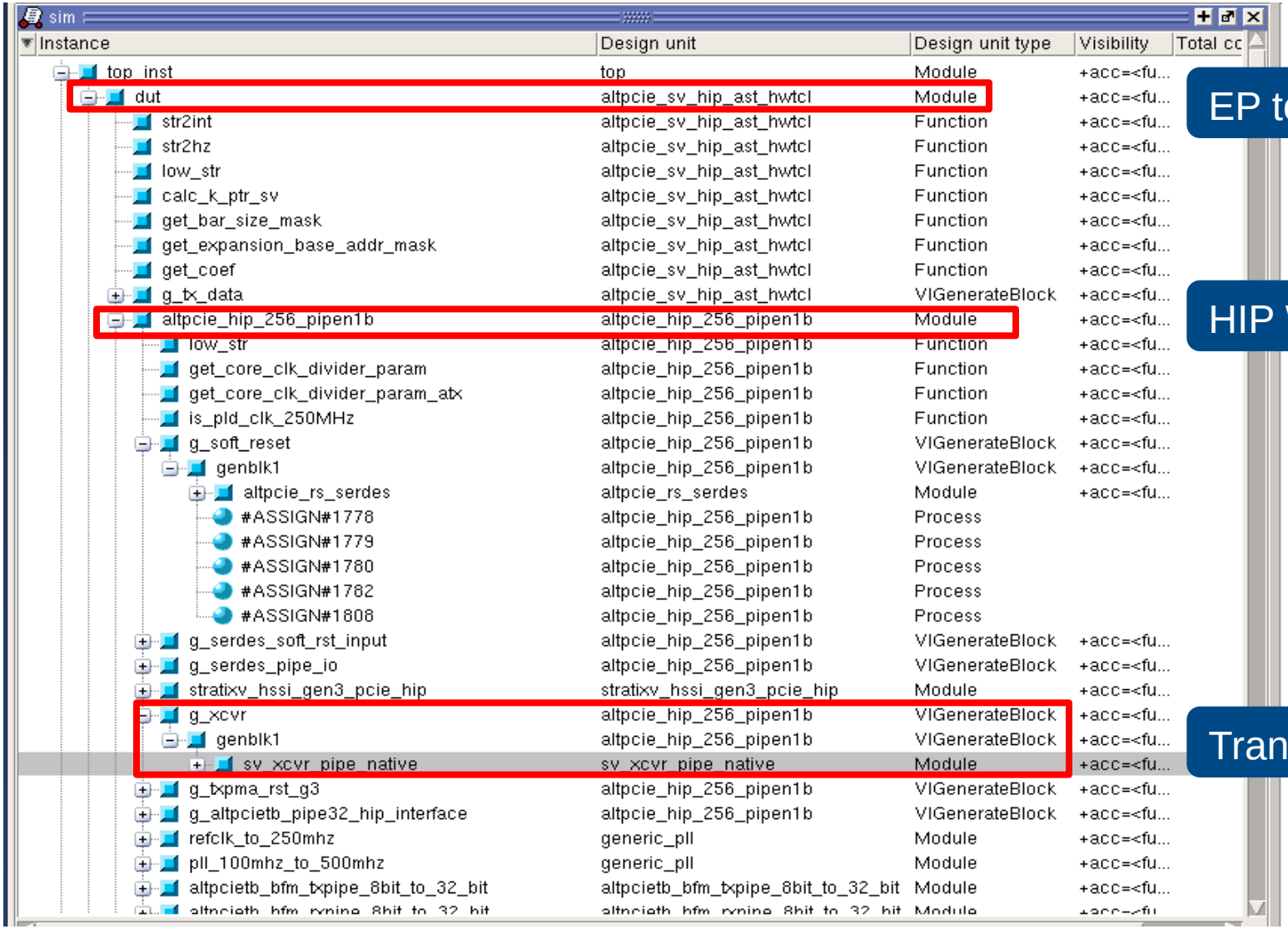
- Start ModelSim tool (vsim)
- Change directory to mentor directory
  - File → Change Directory
- Source the setup file
  - Tools menu → Tcl → Execute Macro
  - Type ***source msim_setup.tcl*** in Transcript window
- Compile and load the design
  - Type ld or ld_debug in Transcript window
- Add debugging signals to Wave window
- Run simulation until finish
  - Type run –all in Transcript window

(1) Running simulation with tools by Cadence, Synopsys and Aldec is also supported.

Gen1/2 Top Level ModelSim Hierarchy



Gen1/2 EP DUT ModelSim Hierarchy



Add Debugging Signals

- Signals to add to Wave window (all found on Device Under Test (DUT) module)
  - RX Avalon-ST interface (rx_st_*)
  - TX Avalon-ST interface (tx_st_*)
  - Clocks (coreclkout, pld_clk, refclk)
  - Reset signals (npor, pin_perstn)
  - LTSSM states (ltssmstate[4:0])
  - PIPE signals
- Save signals in *wave.do* file for future simulations

Tracing PCIe TLP in Wave Window

- Note when the request is executed
- Correlate the Packet in the simulation waveforms
  - Packets found on PIPE or Avalon interface do not line up exactly with the time stamp in the log file due to internal delay
- Requires PCIe knowledge to identify the transaction

Reverse Engineering the Testbench

1. Run the simulation using the existing design
2. Find out what file under submodules folder that contains messages
   - Use “grep” command in Linux to search for the string
3. Search for the task(s) that display them
4. Figure out from where the task is called
5. Continue steps 1 to step 4 until satisfied

Creating Your Own Test Case

- Modify the Root Port driver model (*altpcietb_bfm_driver_chaining.v*) to vary the transactions performed
- Insert the new test to the main routine
- All BFM function calls are defined in the User Guide

Example Testbench Procedures

- Chained DMA Test
  - Top-level procedure that runs the chaining DMA read and write
  - chained_dma_test (bar_table, bar_num, direction, use_msi, use_eplast)
- DMA Read Test
  - Use for DMA reads from the Endpoint memory to the BFM shared memory
  - dma_rd_test (bar_table, bar_num, use_msi, use_eplast)
- DMA Set Write Descriptor Data
  - Use to configure the BFM shared memory for the DMA write
  - dma_set_wr_desc_data (bar_table, bar_num)
- MSI Poll
  - Tracks MSI completion from the Endpoint
  - msi_poll(max_number_of_msi,msi_address,msi_expected_dmawr,msi_expected_dmard,dma_write,dma_read)

Simpler Chaining DMA Testbench Example

- Contains modified design files providing greater control and visibility
  - Simple tasks and examples for performing individual transactions
  - instead of the monolithic tasks we have in our delivered testbench
- Found on Altera Wiki site
- Gen1/Gen2/Gen3 support
- Targets Stratix V device
- Step by step instructions
- Multiple test types
  - DMA read/writes
  - Memory accesses
  - Configuration accesses

PCI Express BFM Choices

- Altera PCIe BFM
  - Pros
    - Free
    - Demonstrate basic Avalon-ST or Avalon-MM transactions
    - Sanity check in early state of user's design phase
    - Learning tools for PIPE interface and link training sequence
  - Cons
    - Do not reflect user's configuration
    - Hard to modify
    - No monitor or score-boarding to check for errors
    - Not recommended for system level regression tests
- Third-party BFM is
  - Example using third-party BFM by Denali http://www.alterawiki.com/wiki/Stratix_V_PCI_Express_Gen3_x8_PIPE_Simulation_with_Denali_Bus_Functional_Model_(BFM)
  - Preferred for system level regression tests

Thank You