NetFPGA Summer Course

<u>Presented by:</u>

Noa Zilberman

Yury Audzevich

Technion

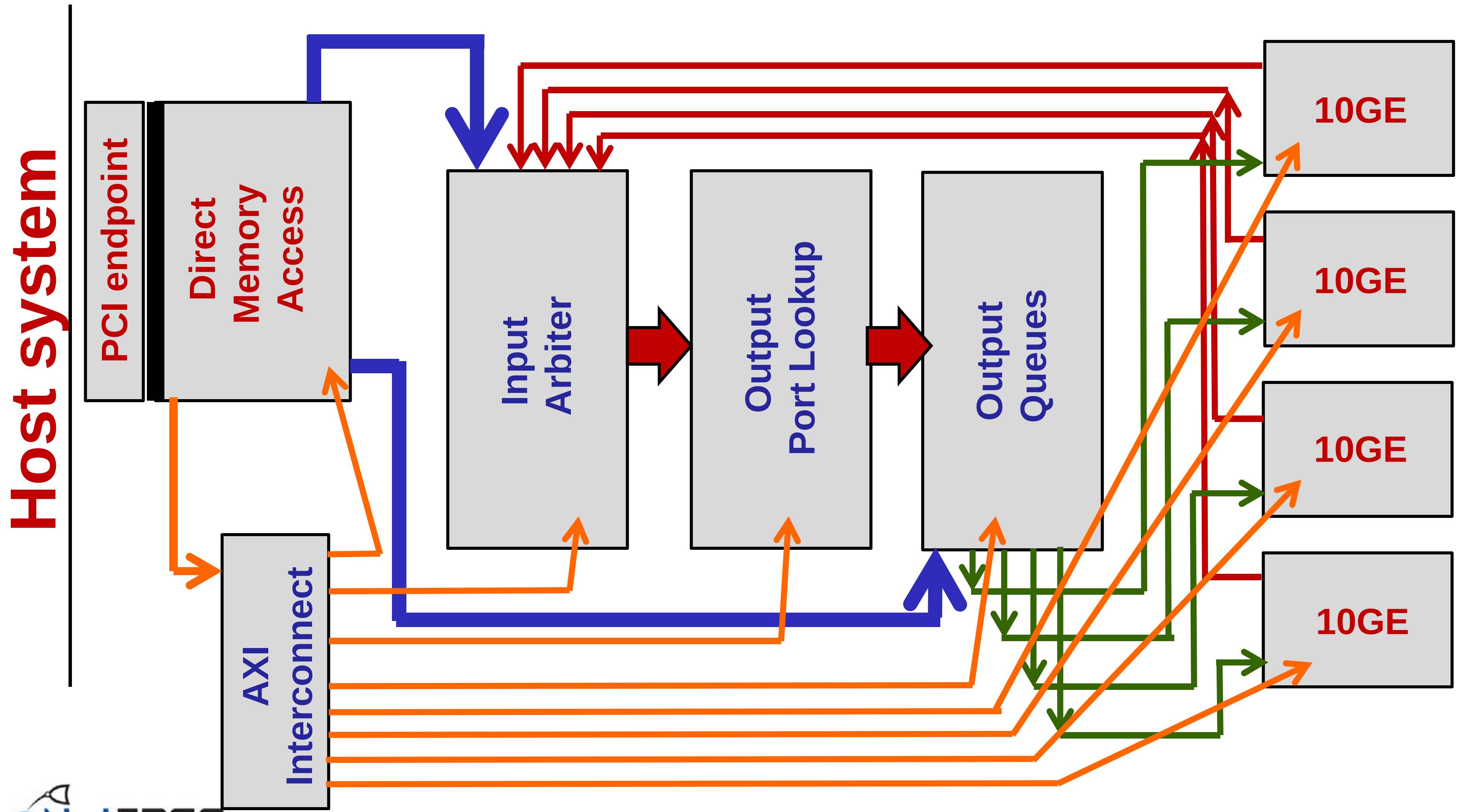
August 2 – August 6, 2015

http://NetFPGA.org

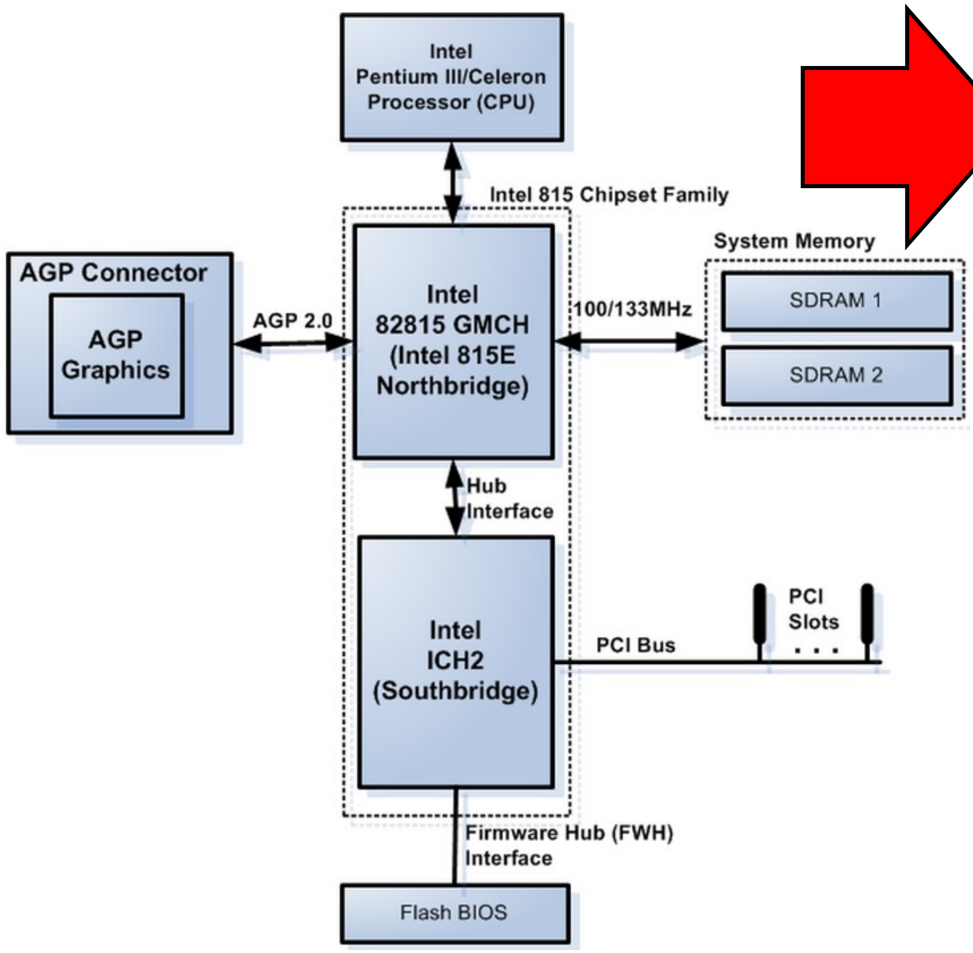
Section I: I/O Architectures

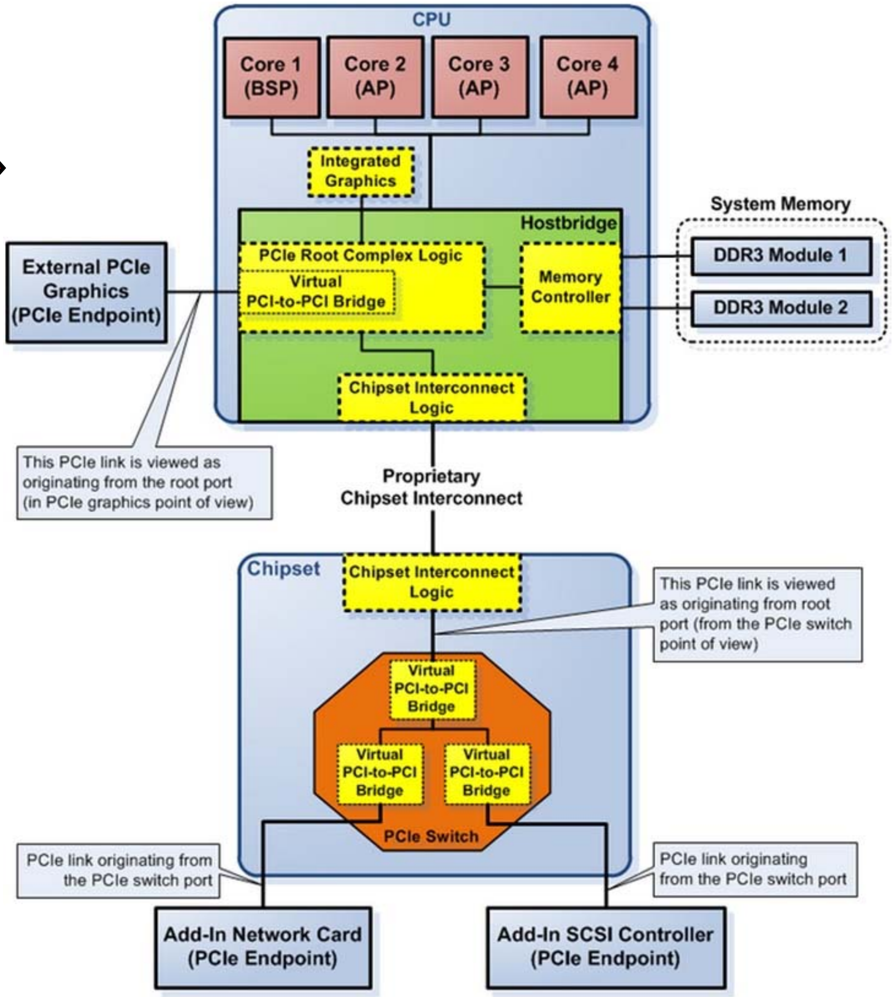
Reference NIC project

4x port NIC architecture:



Host architecture

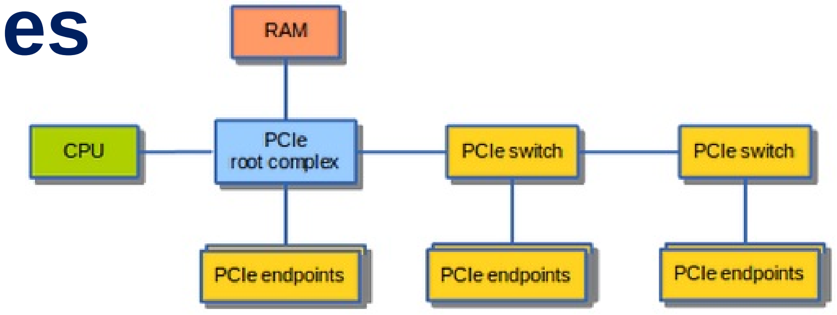



CPU
Core 1 (BSP)
Core 2 (AP)
Core 3 (AP)
Core 4 (AP)
Integrated Graphics
Hostbridge
PCIe Root Complex Logic
Virtual PCI-to-PCI Bridge
Memory Controller
System Memory
DDR3 Module 1
DDR3 Module 2
External PCIe Graphics (PCIe Endpoint)
Chipset Interconnect Logic
This PCIe link is viewed as originating from the root port (in PCIe graphics point of view)
Proprietary Chipset Interconnect
Chipset
Chipset Interconnect Logic
This PCIe link is viewed as originating from root port (from the PCIe switch point of view)
Virtual PCI-to-PCI Bridge
Virtual PCI-to-PCI Bridge
Virtual PCI-to-PCI Bridge
PCIe Switch
PCIe link originating from the PCIe switch port
PCIe link originating from the PCIe switch port
Add-In Network Card (PCIe Endpoint)
Add-In SCSI Controller (PCIe Endpoint)

Legacy vs. Recent (courtesy of Intel)

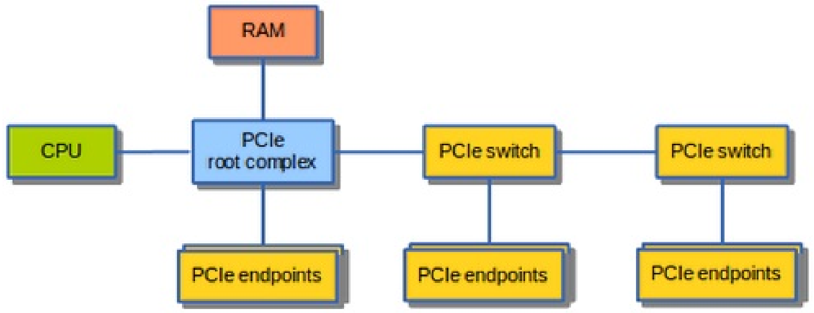
Interconnecting components

- **Need interconnections between**
  - CPU, memory, I/O controllers
- **Bus: shared communication channel**
  - Parallel set of wires for data and synchronization of data transfer
  - Can become a bottleneck
- **Performance limited by physical factors**
  - Wire length, number of connections
- **More recent alternative: high-speed serial connections with switches**
  - Like networks



Bus Types

- **Processor-Memory buses**
  - Short, high speed
  - Design is matched to memory organization
- **I/O buses**
  - Longer, allowing multiple connections
  - Specified by standards for interoperability
  - Connect to processor-memory bus through a bridge



I/O System Characteristics

- **Performance measures**
  - Latency (response time)
  - Throughput (bandwidth)
  - Desktops & embedded systems
    - Mainly interested in response time & diversity of devices
  - Servers
    - Mainly interested in throughput & expandability of devices
- **Dependability**
  - Particularly for storage devices (fault avoidance, fault tolerance, fault forecasting)

I/O Management and strategies

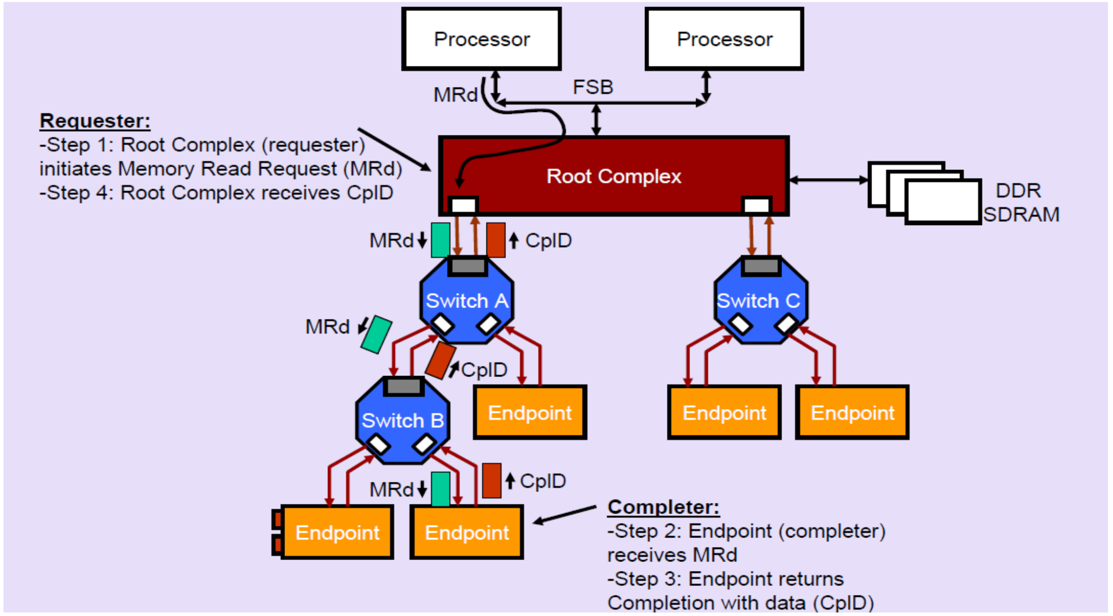
- **I/O is mediated by the OS**
  - Multiple programs share I/O resources
    - Need protection and scheduling
  - I/O causes asynchronous interrupts
    - Same mechanism as exceptions
  - I/O programming is fiddly
    - OS provides abstractions to programs

Strategies characterize the *amount of work* done by the CPU in the I/O operation:

- **Polling**
- **Interrupt Driven**
- **Direct Memory Access**

Programmed I/O

- **Periodically check I/O status register**
  - If device ready, do operation
  - If error, take action
- **Common in small or low-performance real-time embedded systems**
  - Predictable timing
  - Low hardware cost
- **Wastes CPU time**


Processor
Processor
MRd
FSB
Requester:
-Step 1: Root Complex (requester) initiates Memory Read Request (MRd)
-Step 4: Root Complex receives CplD
Root Complex
DDR SDRAM
MRd
CplD
Switch A
Switch C
MRd
CplD
Switch B
Endpoint
Endpoint
Endpoint
MRd
CplD
Endpoint
Endpoint
Completer:
-Step 2: Endpoint (completer) receives MRd
-Step 3: Endpoint returns Completion with data (CplD)

Interrupts

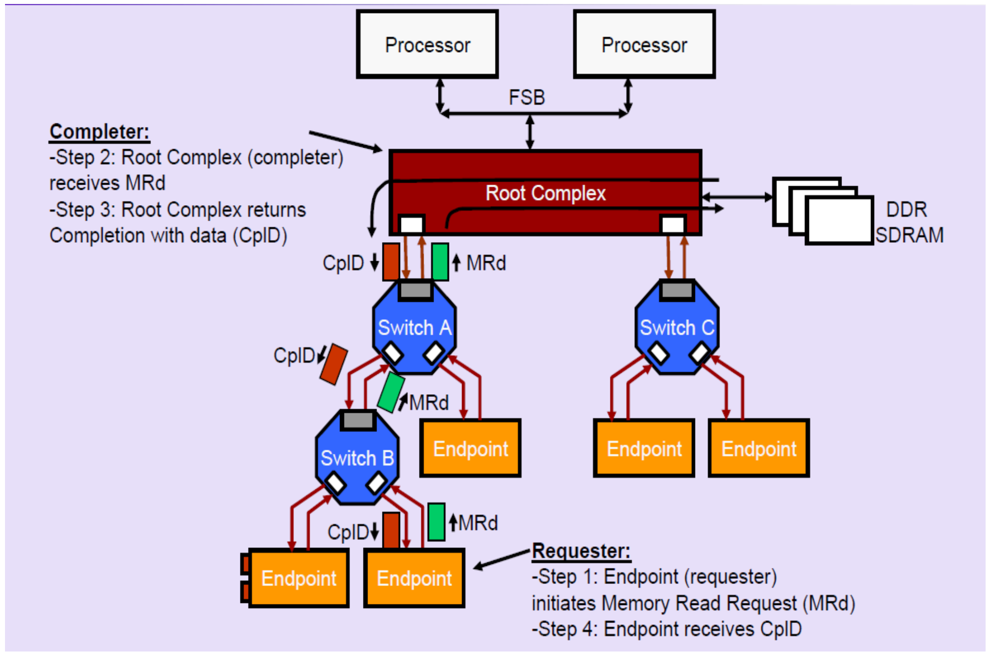
- **When a device is ready or error occurs**
  - Controller interrupts CPU
- **Interrupt is like an exception**
  - But not synchronized to instruction execution
  - Can invoke handler between instructions
  - Cause information often identifies the interrupting device
- **Priority interrupts**
  - Devices needing more urgent attention get higher priority
  - Can interrupt handler for a lower priority interrupt

Direct memory access

DMA is the hardware mechanism that allows peripheral components to transfer their I/O data directly to and from main memory (usually bounded) ***without the need*** to involve the system processor of individual transfers.

- **CPU "programs" DMA with range of block and memory location**

- **CPU when interrupted, checks errors & programs the new operation**


Processor
Processor
FSB
Completer:
-Step 2: Root Complex (completer) receives MRd
-Step 3: Root Complex returns Completion with data (CplD)
Root Complex
DDR SDRAM
CplD
MRd
Switch A
Switch C
CplD
MRd
Switch B
Endpoint
Endpoint
Endpoint
CplD
MRd
Endpoint
Endpoint
Requester:
-Step 1: Endpoint (requester) initiates Memory Read Request (MRd)
-Step 4: Endpoint receives CplD

Direct memory access (cont.)

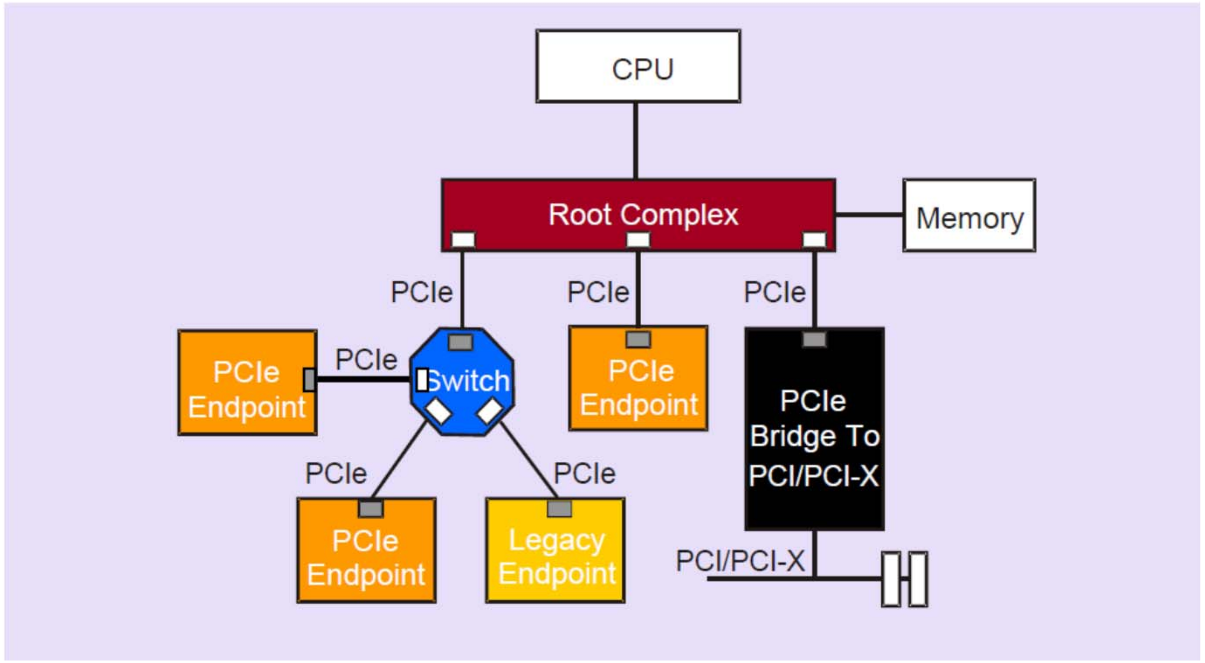
Scatter/gather DMAs are a special type of streaming DMAs:

- Handle cases when there are several discontinuous buffers, all of which need to be transferred to or from the device
- Devices accept a *scatterlist* of array pointers and lengths, and transfer them all in one DMA operation
- Good for "zero-copy" networking since packets can be built in multiple pieces

Section II: PCI Express

PCIe introduction

- PCIe is a ***serial point-to-point interconnect*** between two devices
- Implements ***packet based protocol (TLPs)*** for information transfer
- ***Scalable performance*** based on # of signal Lanes implemented on the PCIe interconnect
- Supports ***credit-based*** point-to-point flow control (not end-to-end)



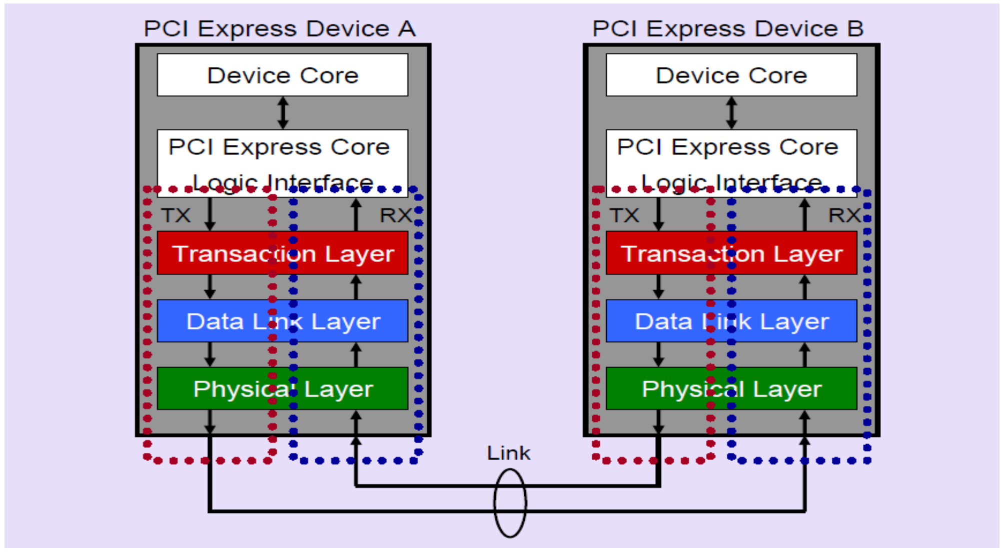
Provides:

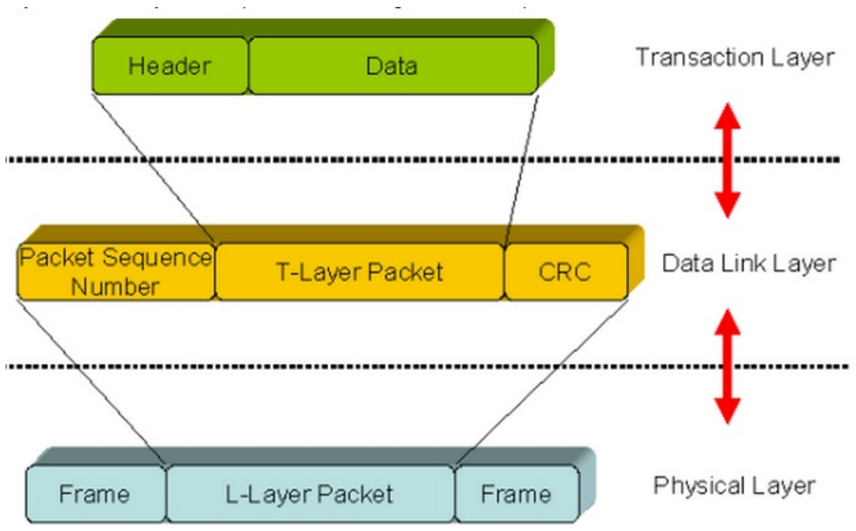
- processor independence & buffered isolation
- Bus mastering
- Plug and Play operation

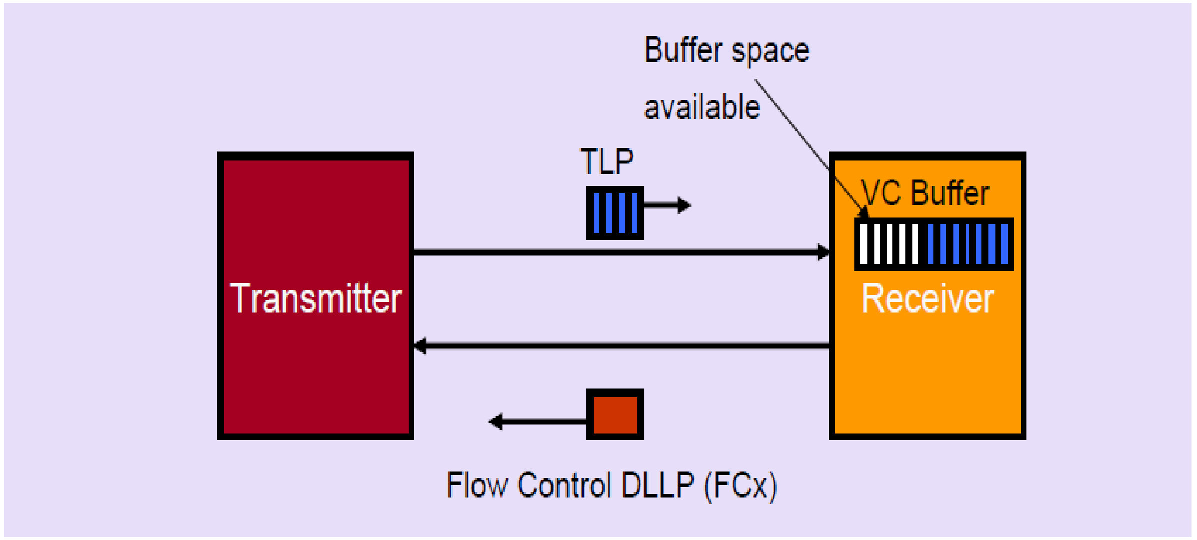
PCIe transaction types

- **Memory Read or Memory Write.** Used to transfer data from or to a memory mapped location
- **I/O Read or I/O Write.** Used to transfer data from or to an I/O location
- **Configuration Read or Configuration Write.** Used to discover device capabilities, program features, and check status in the 4KB PCI Express configuration space.
- **Messages. Handled like posted writes.** Used for event signaling and general purpose messaging.

PCIe architecture







Interrupt Model

PCI Express supports three interrupt reporting mechanisms:

1. Message Signaled Interrupts (MSI)

- interrupt the CPU by writing to a specific address in memory with a payload of 1 DW

2. Message Signaled Interrupts - X (MSI-X)

- MSI-X is an extension to MSI, allows targeting individual interrupts to different processors

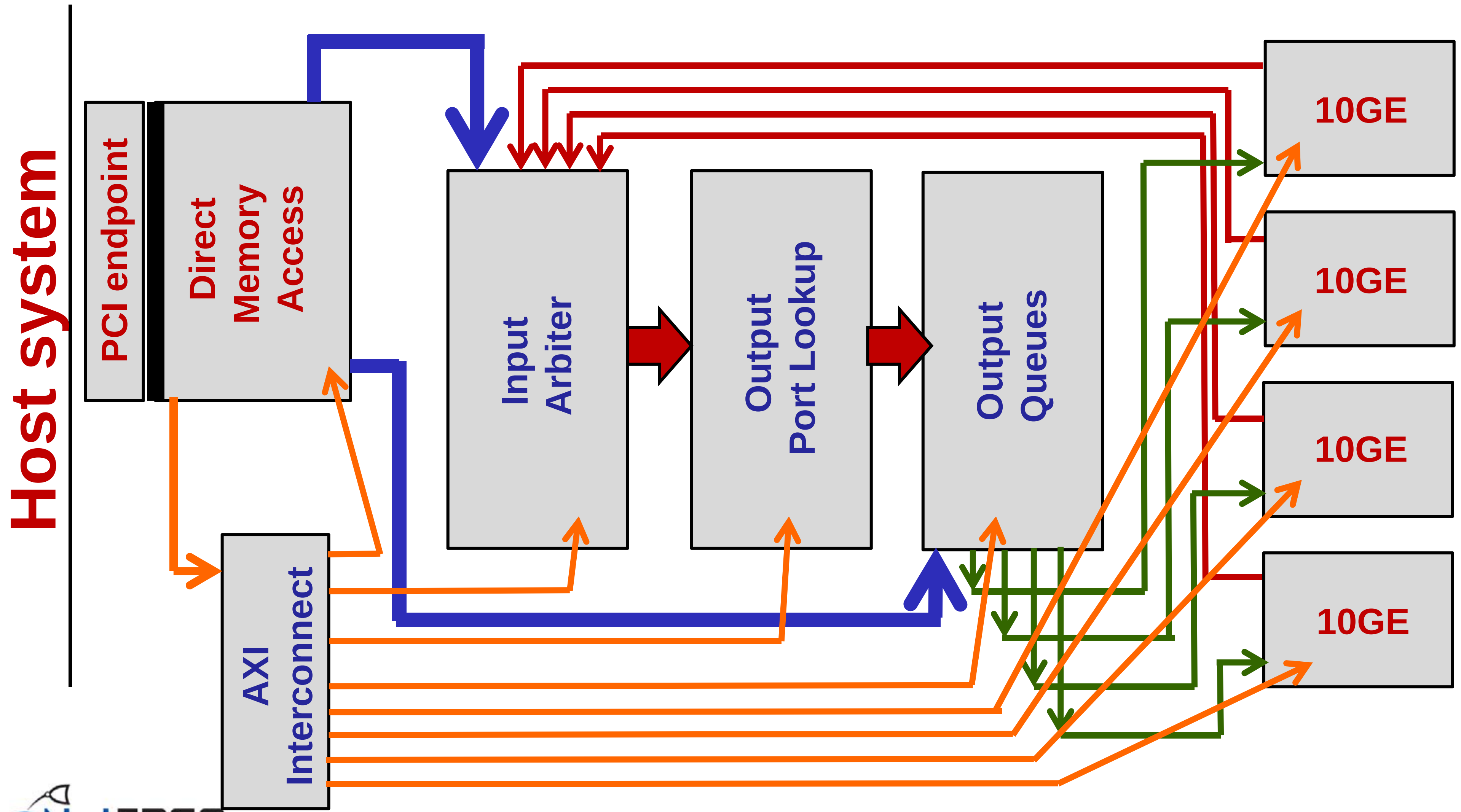
3. INTx Emulation

- **four physical interrupt signals INTA-INTD are messages upstream**
- **ultimately be routed to the system interrupt controller**

Section III: RIFFA DMA

Reference NIC project

4x port NIC architecture:

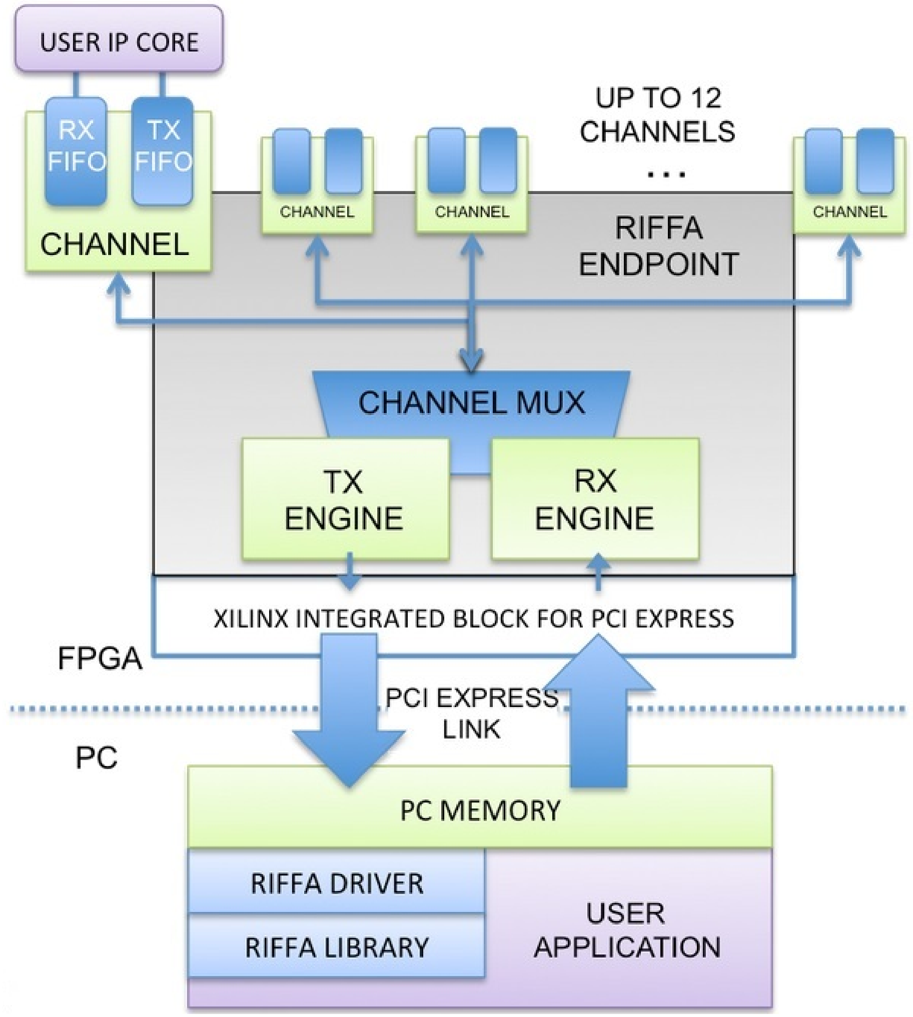


RIFFA

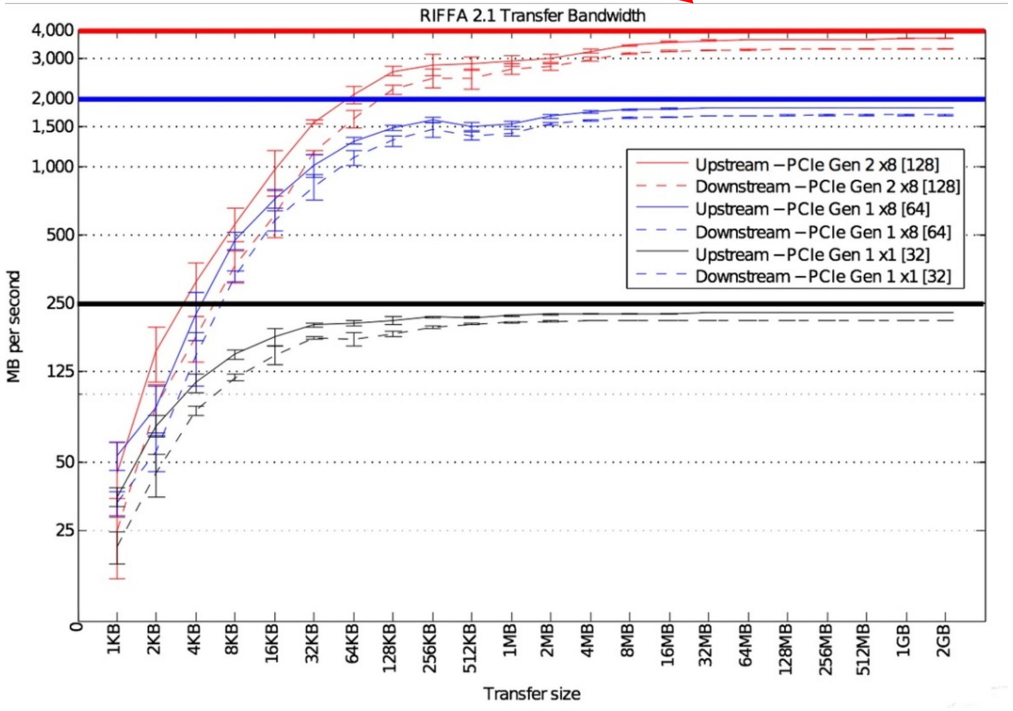
RIFFA (Reusable Integration Framework for FPGA Accelerators)

- Developed by UCSD
- RIFFA has been tested with both Altera and Xilinx devices
- Driver supports Windows and Linux OSes
- Provide bindings for C/C++, Python, MATLAB and Java
- Latest generation of the original engine
- At the moment supports only Gen 2.0 PCIe
- Github: https://github.com/drichmond/riffa

RIFFA Overview

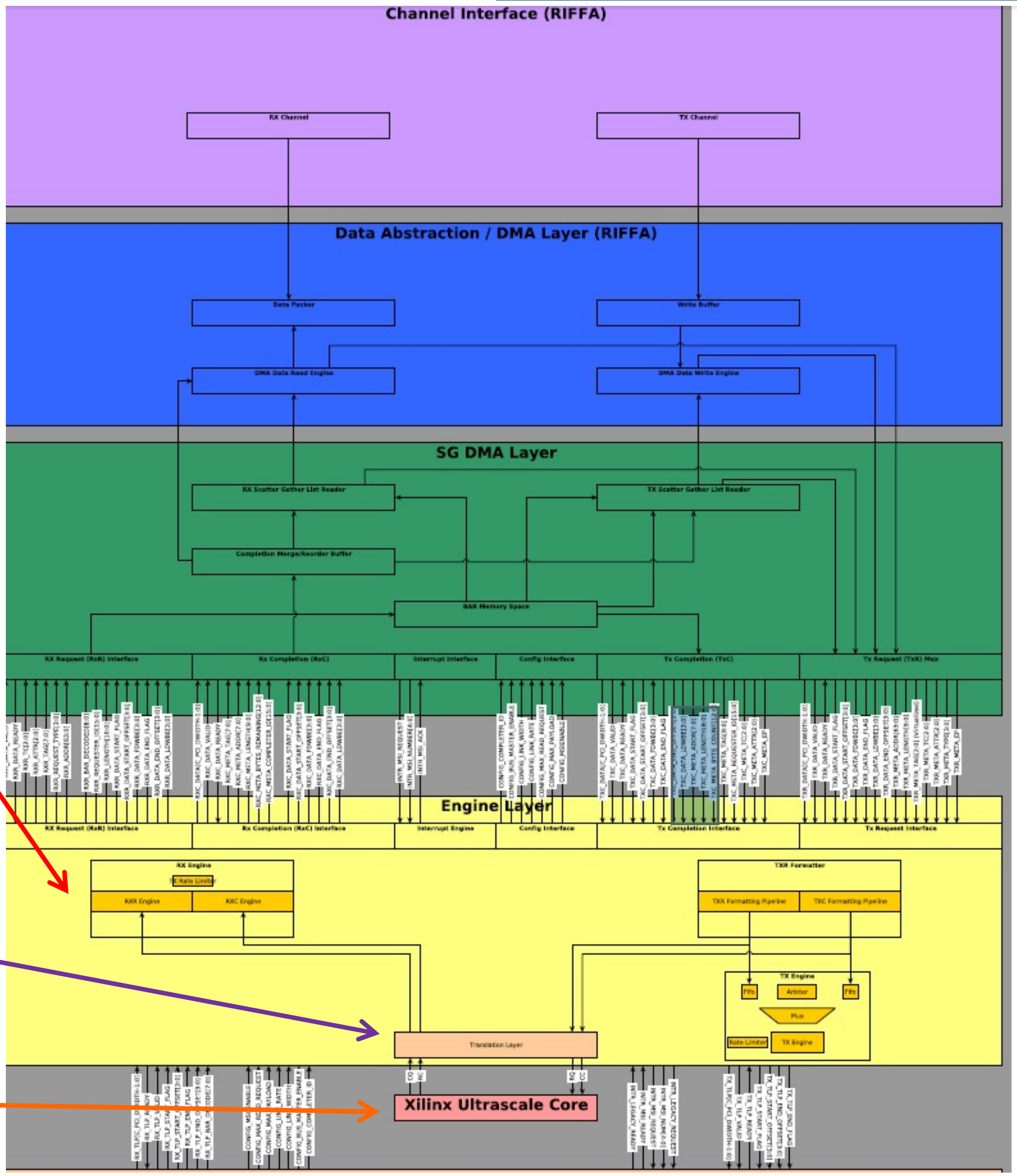


achieves 76% of the theoretical max



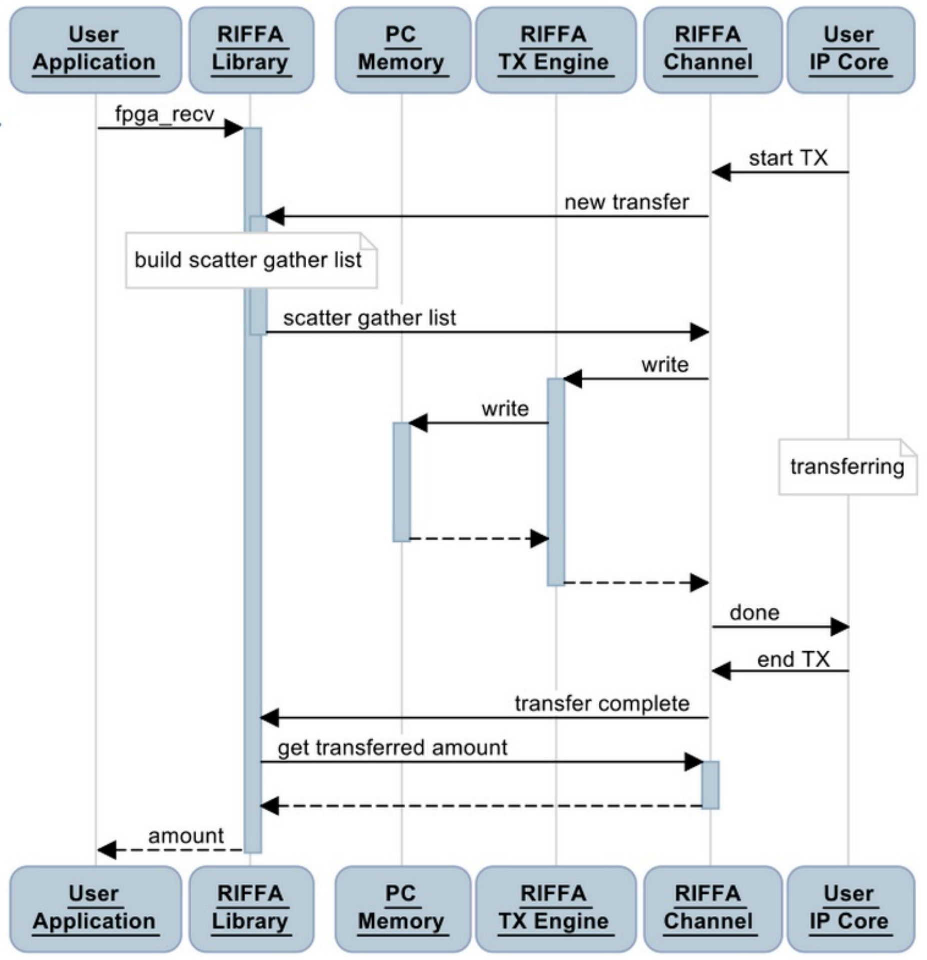
RIFFA architecture

- **Data Abstraction / DMA Layer is responsible for making requests to read data from, or write data to host memory**
- **SG DMA Layer: reading from and writing to scatter gather lists; supplying addresses to data-request logic**
- **Formatting Engine Layer is responsible for formatting requests and completions into packets.**
- **Translation Layer provides a set of vendor-independent interfaces and signal names**
- **Vendor IP interfaces provide low-level access to the PCIe bus**

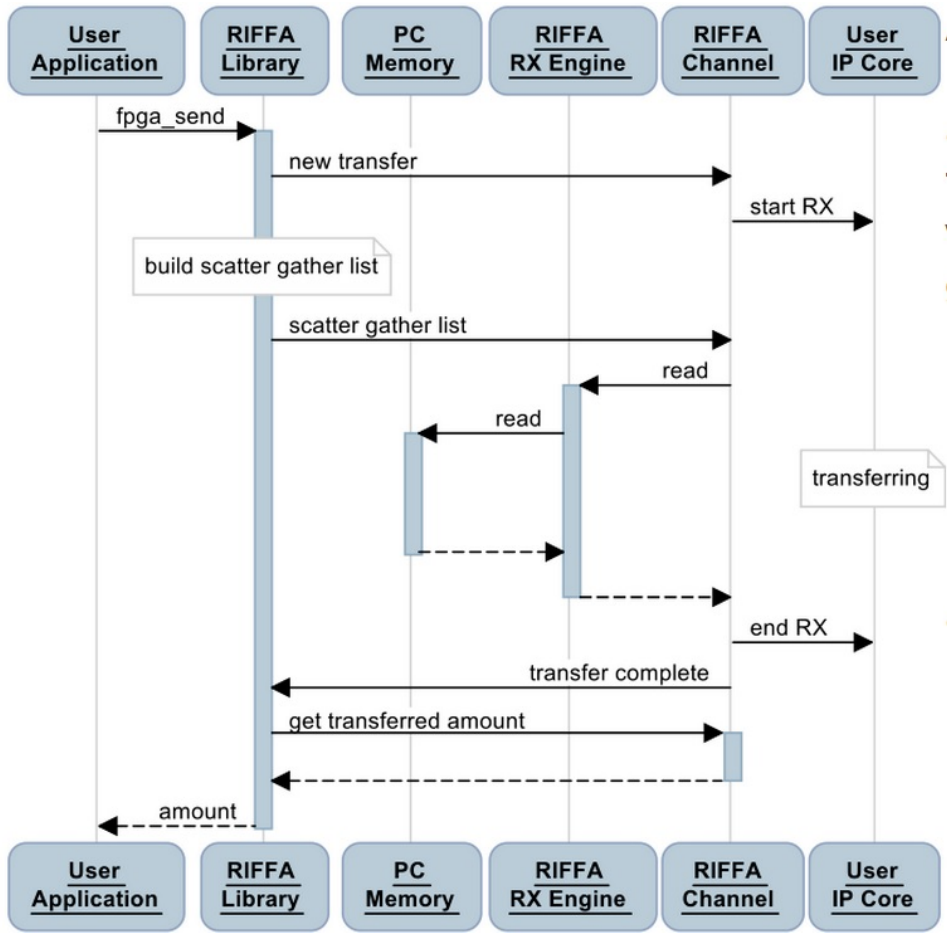


RIFFA Data transfer example

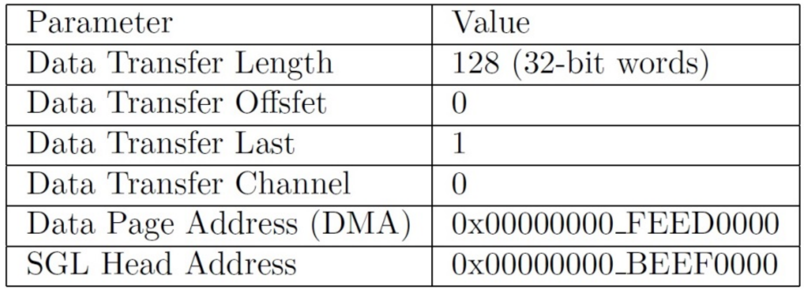
FPGA -> Host



Host-> FPGA



RIFFA Data transfer example (cont.)



Parameter	Value
Data Transfer Length	128 (32-bit words)
Data Transfer Offset	0
Data Transfer Last	1
Data Transfer Channel	0
Data Page Address (DMA)	0x00000000_FEED0000
SGL Head Address	0x00000000_BEEF0000

Note: each channel has its own SG DMA list logic

Host SEND case

1) User wants to make a of transfer **128 32-bit words**;

2) The RIFFA driver writes **{32'd128**} to **Channel 0's RX Length register**, and **{31'd0,1'b1}** to **Channel 0's RX OffLast** register

3) The RIFFA driver **allocates** an **SGL** with **1 element (4 32-bit words)** at address **{64'h0000_ 0000_ BEEF_ 0000}**

4) The driver **fills the list with the length and address** of the user data: **{32'd0,32'd128,64'h0000_ 0000_ FEED_ 0000}**

5) driver communicates the address and length of the SGL by writing **{32'hBEEF0000} to Channel 0's** RX SGL Address Low register, {32'd0} to Channel 0's RX SGL Address High register, and {32'd4} to Channel 0's RX SGL Length register

6) SG List Requester on the FPGA issues a read request for 4 32-bit starting at address **0xBEEF0000**

7) The FPGA receieves a completion with 4 32-bit words

8) RX Port Reader removes the SG element from the FIFO, and issues several read requests to receive all 128 32-bit words. Compl are reordered in reorder buffer.

9) RIFFA raises an interrupt with the last word of data put into main FIFO. driver reads the Interrupt Status Register of the FPGA and determines that Channel 0 has nished the RX Transaction

Networking with RIFFA

SUME RIFFA driver:

- RIFFA DMA engine design dominated
- Single BAR for info and transfer programming
- 2 channels: 1 for packets, 1 for registers
- Single interrupt
- Single global lock
- Supports 1..4 ports, Ethernet interfaces named nf<n>

Networking with RIFFA (cont.)

Packets – CHANNEL 0

- First PCIe channel (De)Multiplexes ports to interfaces and vice versa based on 128bit meta data
- Currently uses a 4k temporary buffer per direction currently (with 16bit offset for 32bit L3 alignment, will DMA directly to “skb” data area in the future)
- 1 packet per DMA transaction

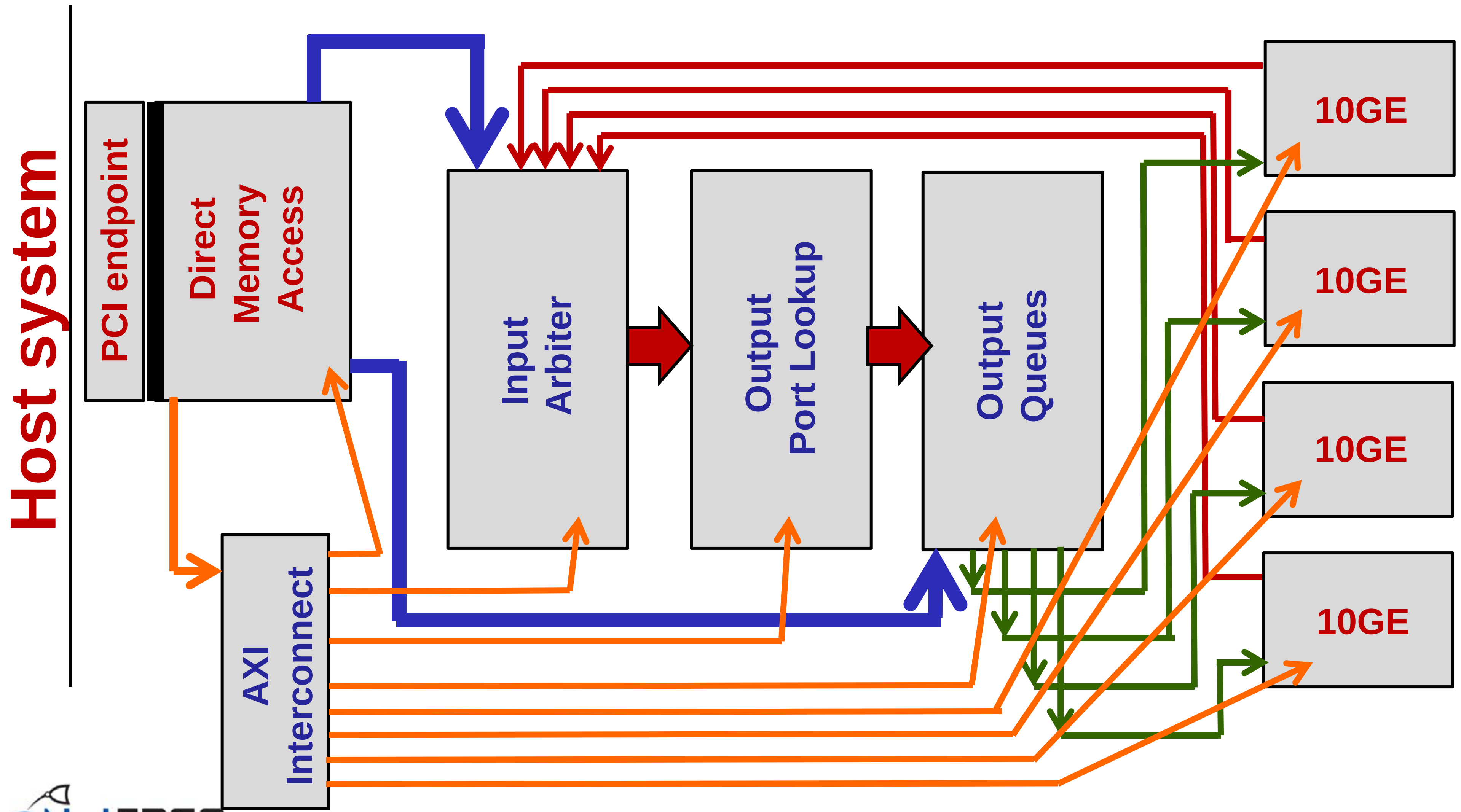
IOCTL (Register r/w) – CHANNEL 1

- Based on an interface of the card (can have multiple cards)
- Uses standard struct ifreq with struct sume_ifreq data pointer
- Supports read write operations on registers (see: nf_sume.h, rwaxi tool)
- Second PCIe channel
- Only one outstanding register r/w possible at a time
- Writing initiates full DMA transaction with address, value, and 0x1f STRB
- Read is like a write with 0x00 STRB, followed by a 2nd DMA transaction to read value back
- Each read/write goes through similar DMA transfer cycle packet data goes through

Section III: An alternative DMA design

Reference NIC project

4x port NIC architecture:

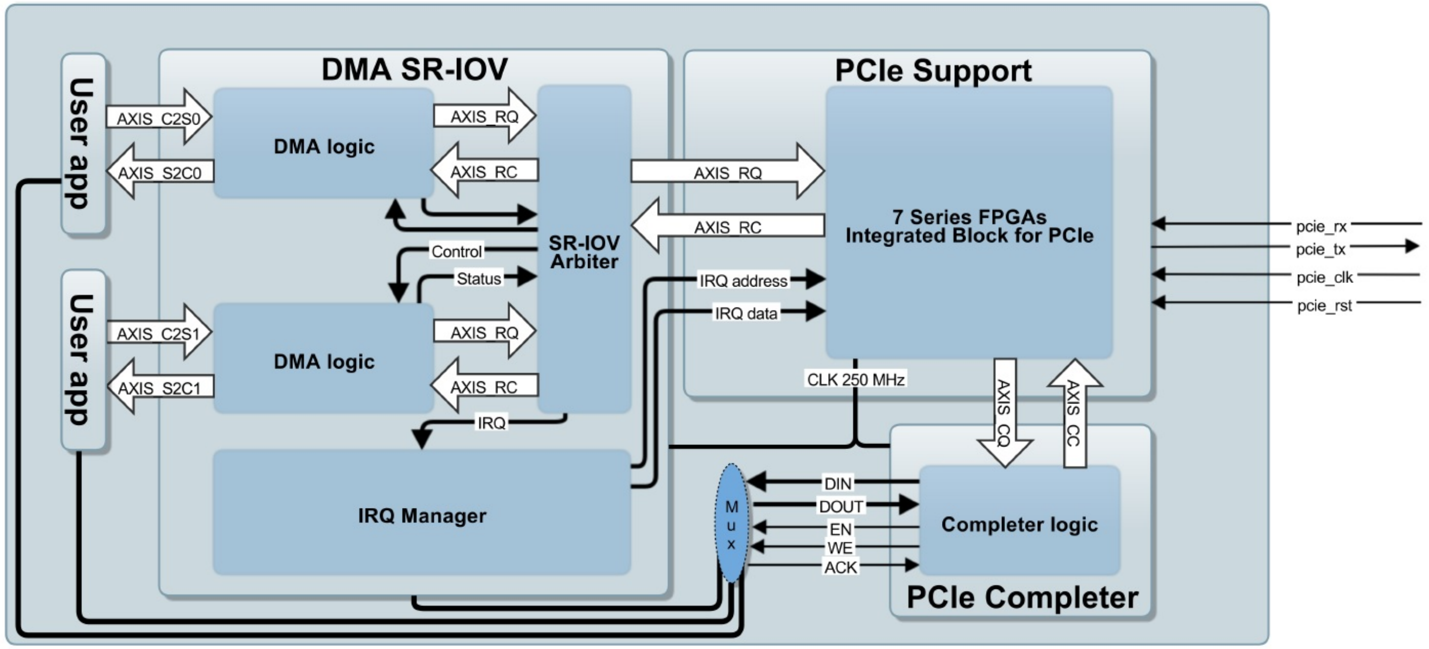


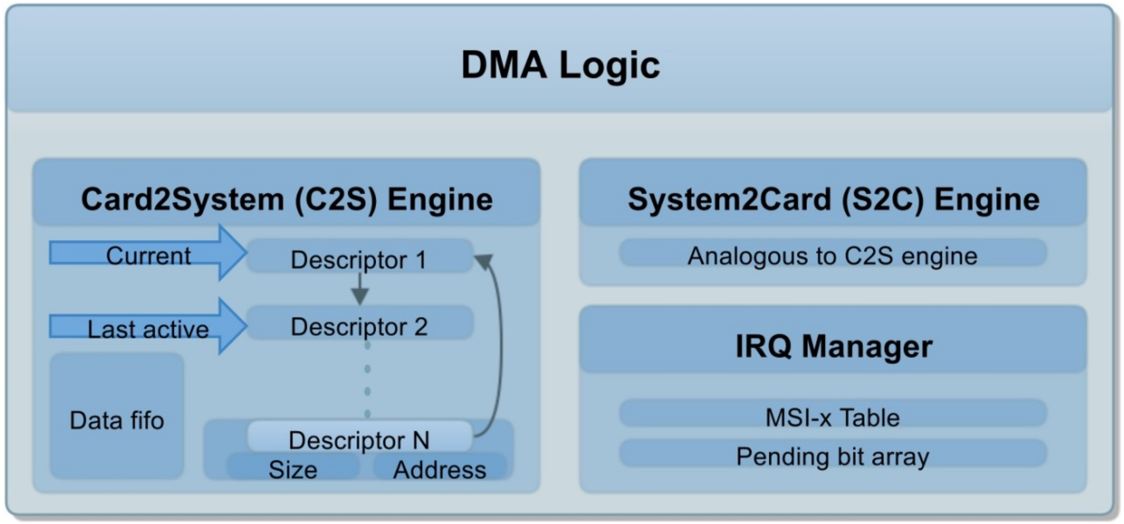
UAM DMA

Build by University Autonoma Madrid (UAM) in collaboration with NetFPGA's Cambridge team

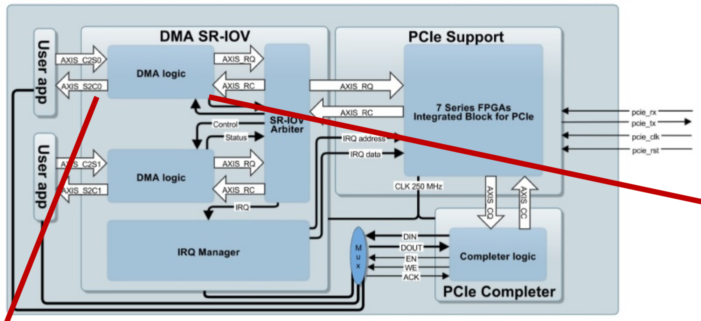
- **Supports PCIe Gen 3.0 x8 speeds**
- **Designed to be extremely lightweight and easy to understand**
- **Tailored for Xilinx platform only**
- **Designed for virtualized environments (SR-IOV)**
- **Has been tested on Linux platform**

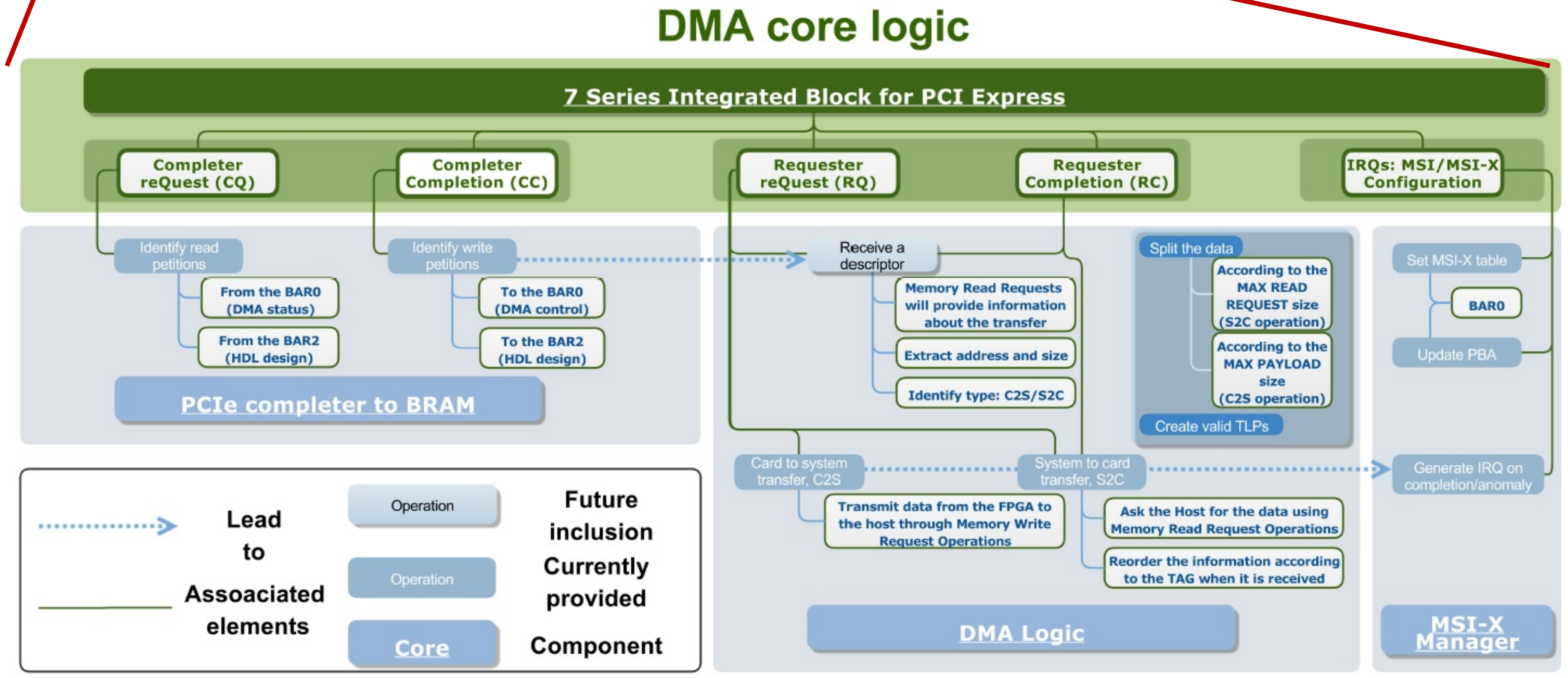
DMA Architecture





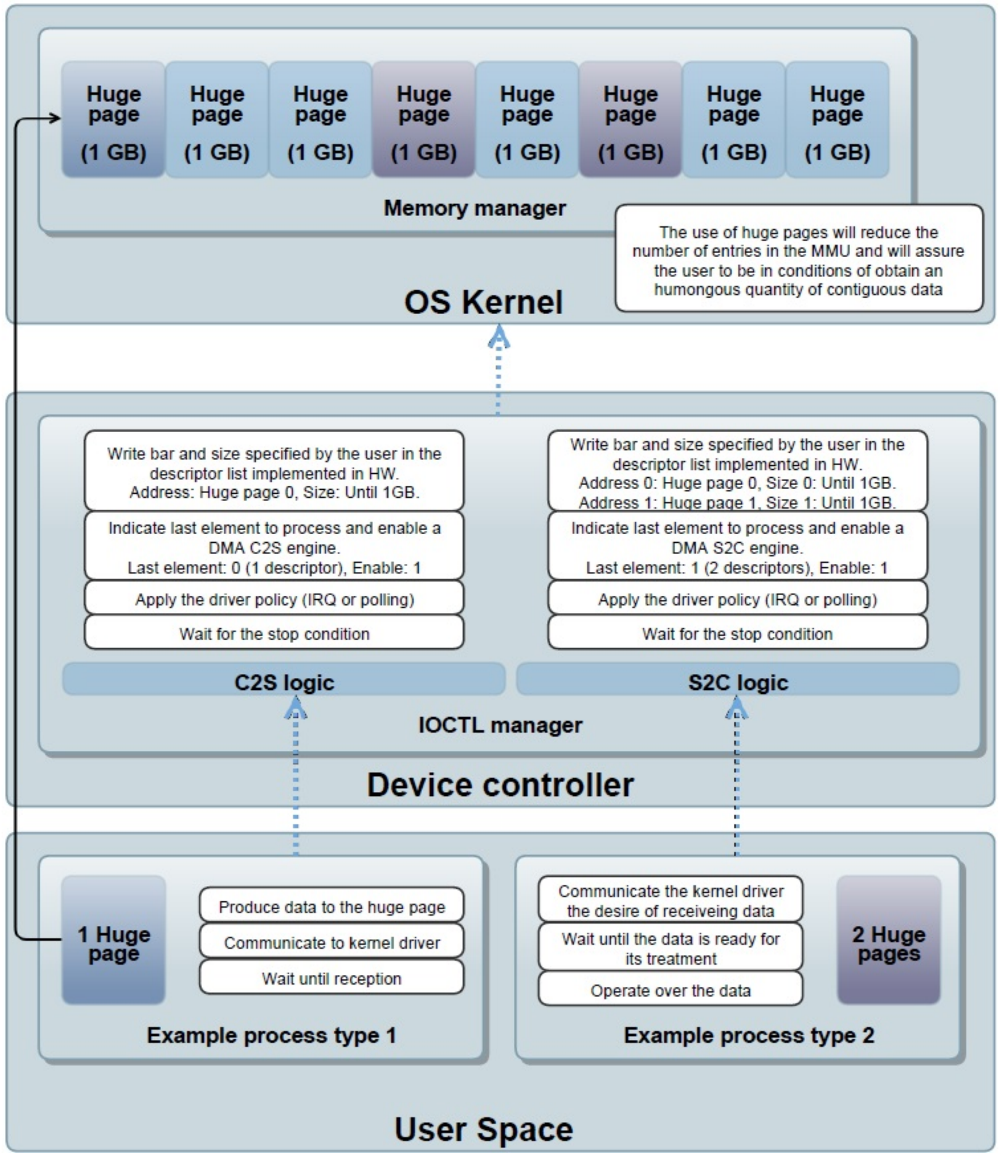
DMA Architecture (cont.)



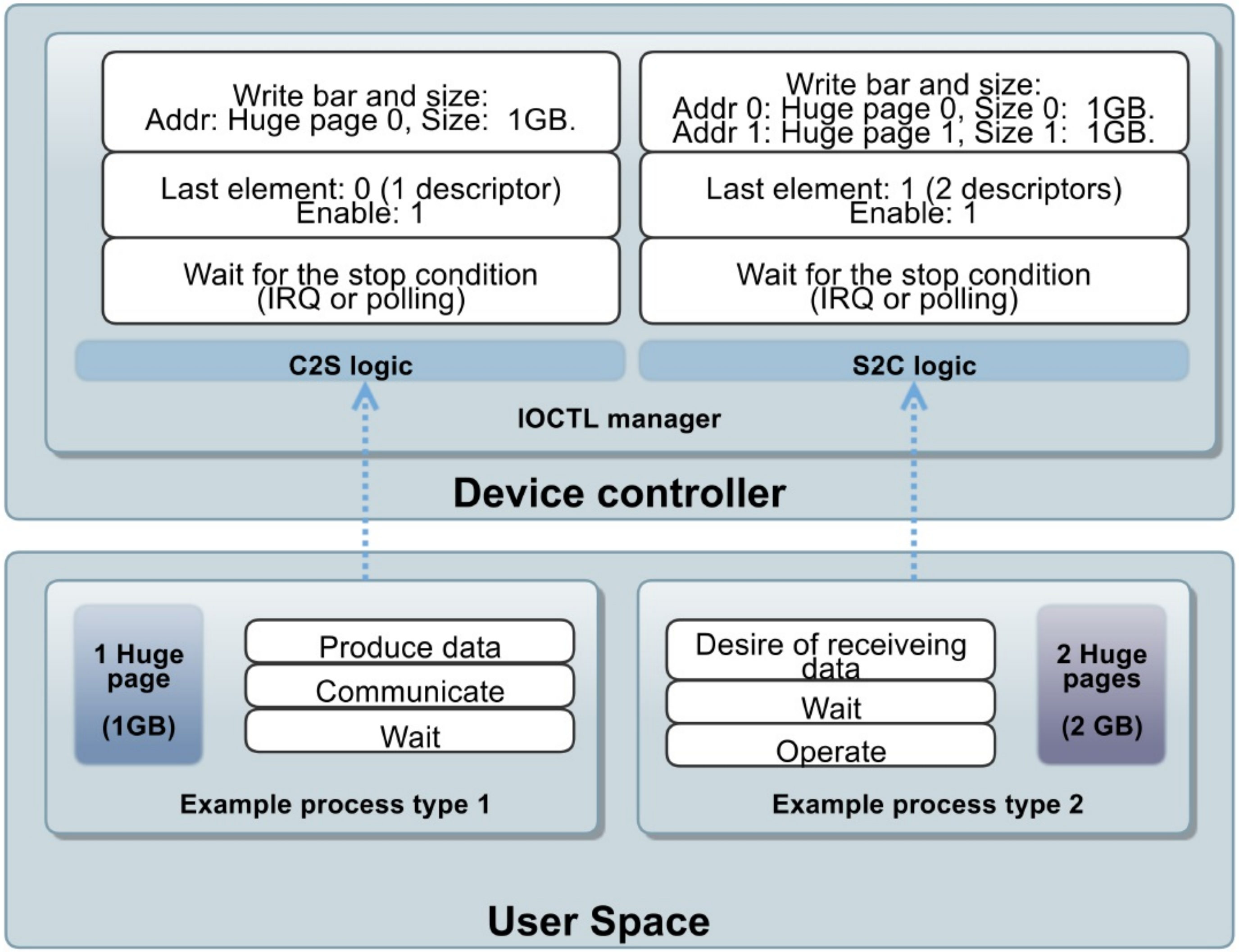


DMA core logic
7 Series Integrated Block for PCI Express
Completer reQuest (CQ)
Completer Completion (CC)
Requester reQuest (RQ)
Requester Completion (RC)
IRQs: MSI/MSI-X Configuration
Identify read petitions
From the BAR0 (DMA status)
From the BAR2 (HDL design)
Identify write petitions
To the BAR0 (DMA control)
To the BAR2 (HDL design)
PCIe completer to BRAM
Receive a descriptor
Memory Read Requests will provide information about the transfer
Extract address and size
Identify type: C2S/S2C
Split the data
According to the MAX READ REQUEST size (S2C operation)
According to the MAX PAYLOAD size (C2S operation)
Create valid TLPs
Card to system transfer, C2S
Transmit data from the FPGA to the host through Memory Write Request Operations
System to card transfer, S2C
Ask the Host for the data using Memory Read Request Operations
Reorder the information according to the TAG when it is received
DMA Logic
Set MSI-X table
BAR0
Update PBA
Generate IRQ on completion/anomaly
MSI-X Manager
Lead to
Associated elements
Operation
Future inclusion
Operation
Currently provided
Core
Component

SW/HW perspective


Huge page (1 GB)
Huge page (1 GB)
Huge page (1 GB)
Huge page (1 GB)
Huge page (1 GB)
Huge page (1 GB)
Huge page (1 GB)
Huge page (1 GB)
Memory manager
The use of huge pages will reduce the number of entries in the MMU and will assure the user to be in conditions of obtain an humongous quantity of contiguous data
OS Kernel
Write bar and size specified by the user in the descriptor list implemented in HW. Address: Huge page 0, Size: Until 1GB.
Indicate last element to process and enable a DMA C2S engine. Last element: 0 (1 descriptor), Enable: 1
Apply the driver policy (IRQ or polling)
Wait for the stop condition
Write bar and size specified by the user in the descriptor list implemented in HW. Address 0: Huge page 0, Size 0: Until 1GB. Address 1: Huge page 1, Size 1: Until 1GB.
Indicate last element to process and enable a DMA S2C engine. Last element: 1 (2 descriptors), Enable: 1
Apply the driver policy (IRQ or polling)
Wait for the stop condition
C2S logic
S2C logic
IOCTL manager
Device controller
1 Huge page
Produce data to the huge page
Communicate to kernel driver
Wait until reception
Example process type 1
Communicate the kernel driver the desire of receiveing data
Wait until the data is ready for its treatment
Operate over the data
2 Huge pages
Example process type 2
User Space

SW/HW perspective (cont.)



Future plans

- The initial tests show 40Gbps+ throughput achieved with one channel
- Network driver extensions
- Part of next release of NetFPGA platform

Section IX: Conclusion

Acknowledgments (I)

NetFPGA Team at University of Cambridge (Past and Present):

Andrew Moore, David Miller, Muhammad Shahbaz, Martin Zadnik
Matthew Grosvenor, Yury Audzevich, Neelakandan Manihatty-Bojan,
Georgina Kalogeridou, Jong Hun Han, Noa Zilberman, Gianni Antichi,
Charalampos Rotsos, Marco Forconesi, Jinyun Zhang, Bjoern Zeeb

NetFPGA Team at Stanford University (Past and Present):

Nick McKeown, Glen Gibb, Jad Naous, David Erickson,
G. Adam Covington, John W. Lockwood, Jianying Luo, Brandon Heller, Paul Hartke, Neda Beheshti, Sara Bolouki, James Zeng,
Jonathan Ellithorpe, Sachidanandan Sambandan, Eric Lo

All Community members (including but not limited to):

Paul Rodman, Kumar Sanghvi, Wojciech A. Koszek,
Yahsar Ganjali, Martin Labrecque, Jeff Shafer, Eric Keller ,
Tatsuya Yabe, Bilal Anwer, Yashar Ganjali, Martin Labrecque,
Lisa Donatini, Sergio Lopez-Buedo

Kees Vissers, Michaela Blott, Shep Siegel, Cathal McCabe

Acknowledgements (II)

UNIVERSITY OF CAMBRIDGE

LELAND STANFORD JUNIOR UNIVERSITY · DIE LUFT DER FREIHEIT WEHT · 1891

NSF

XILINX®

EPSRC
Pioneering research and skills

BROADCOM®

Micron™

CYPRESS
PERFORM

ALGO-LOGIC

DIGILENT®

Google

Atomic Rules

Global HiTech

__Disclaimer:__ ***Any opinions, findings, conclusions, or recommendations expressed in these materials do not necessarily reflect the views of the National Science Foundation or of any other sponsors supporting this project.***

This effort is also sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. This material is approved for public release, distribution unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.