NetFPGA Summer Course

Presented by:

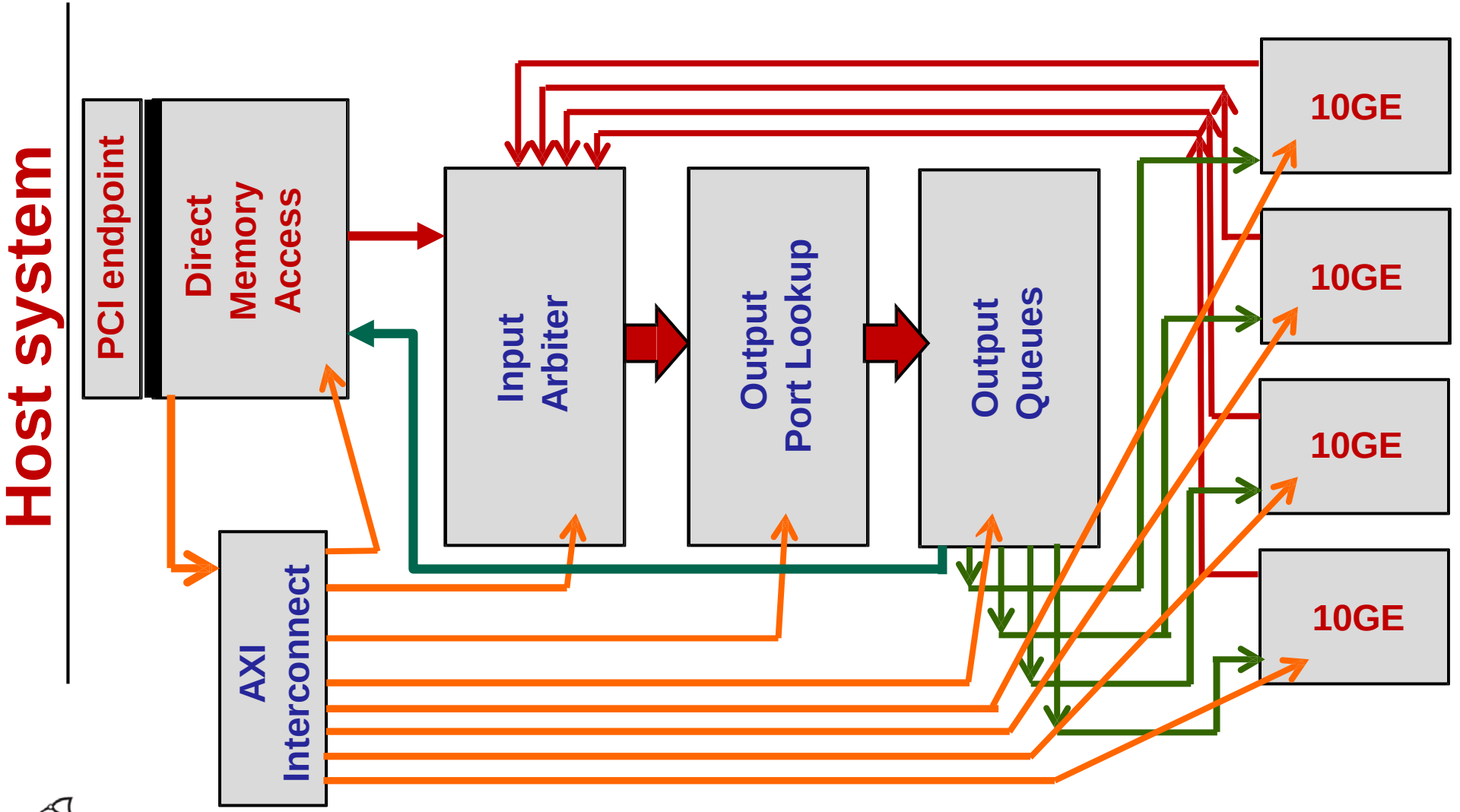
Andrew W Moore, Noa Zilberman, Gianni Antichi
Stephen Ibanez, Marcin Wojcik, Jong Hun Han,
Salvator Galea, Murali Ramanujam, Jingyun Zhang,
Yuta Tokusashi

University of Cambridge
July 24 – July 28, 2017

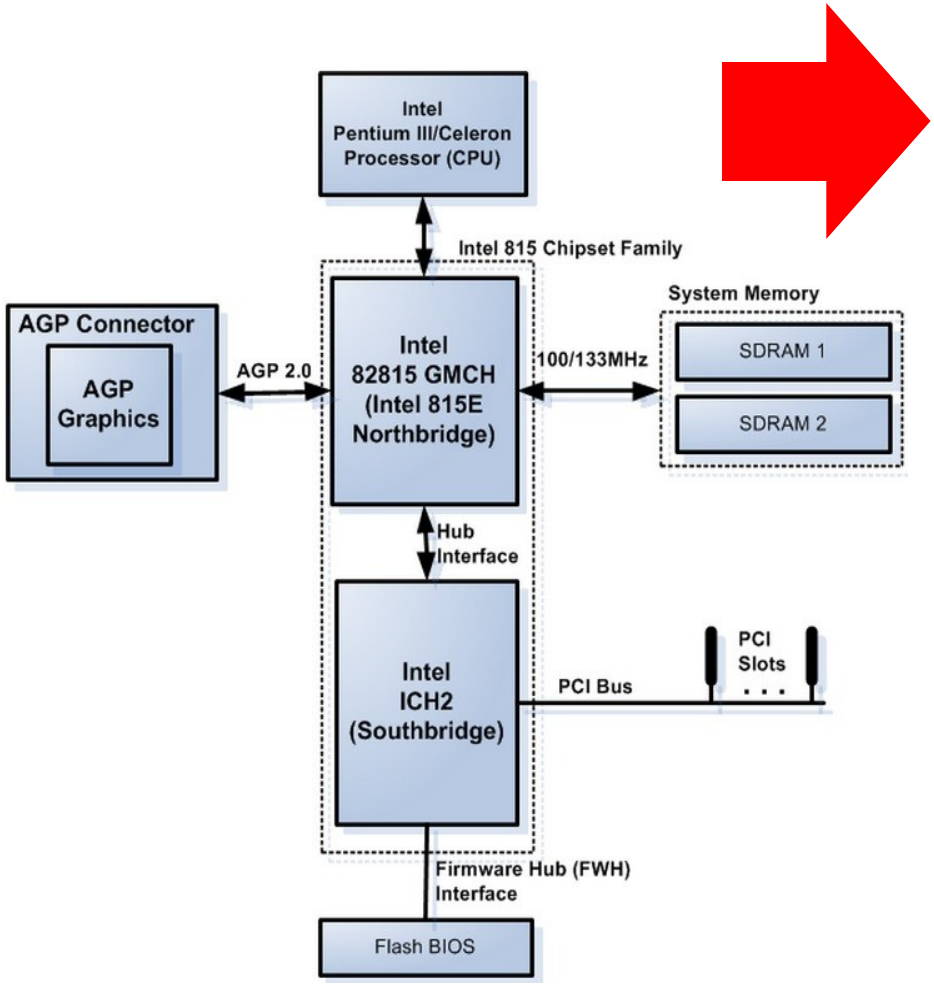
http://NetFPGA.org

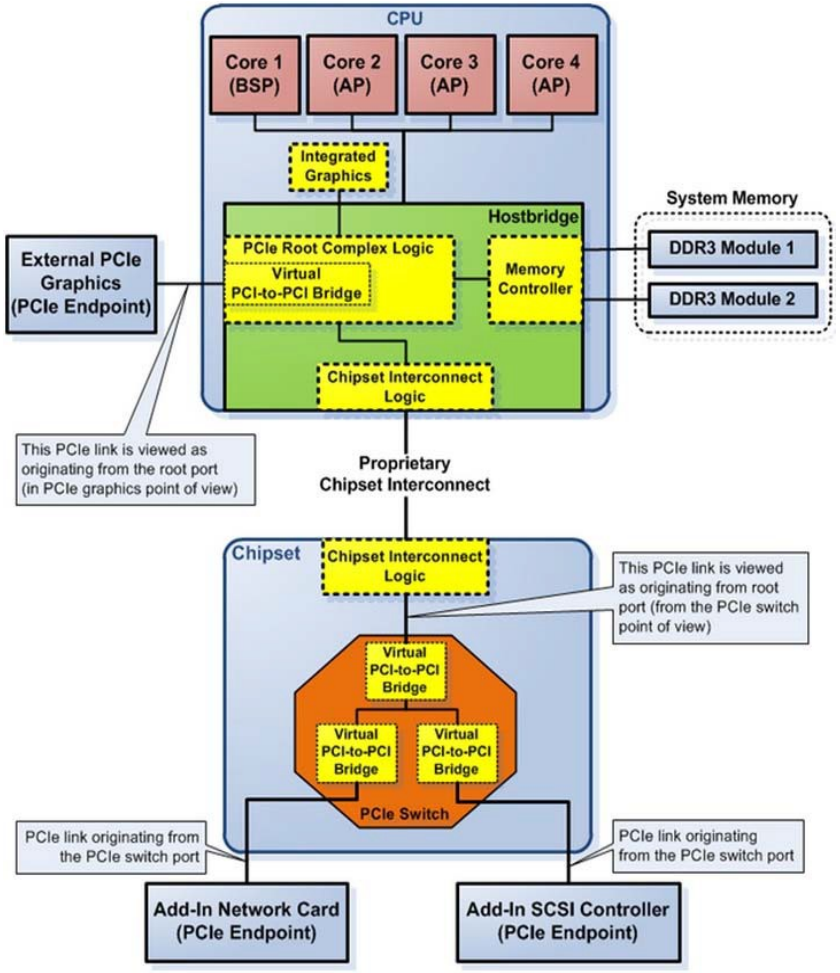
Reference NIC project

4x port NIC architecture:



Host architecture

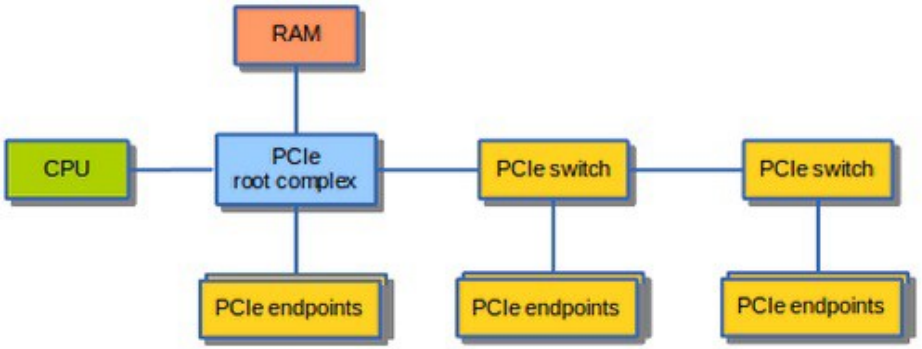




Legacy vs. Recent (courtesy of Intel)

Interconnecting components

- **Need interconnections between**
  - CPU, memory, storage, network, I/O controllers
- **Shared Bus: shared communication channel**
  - A set of parallel 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



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
- **Reliability**
  - 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**

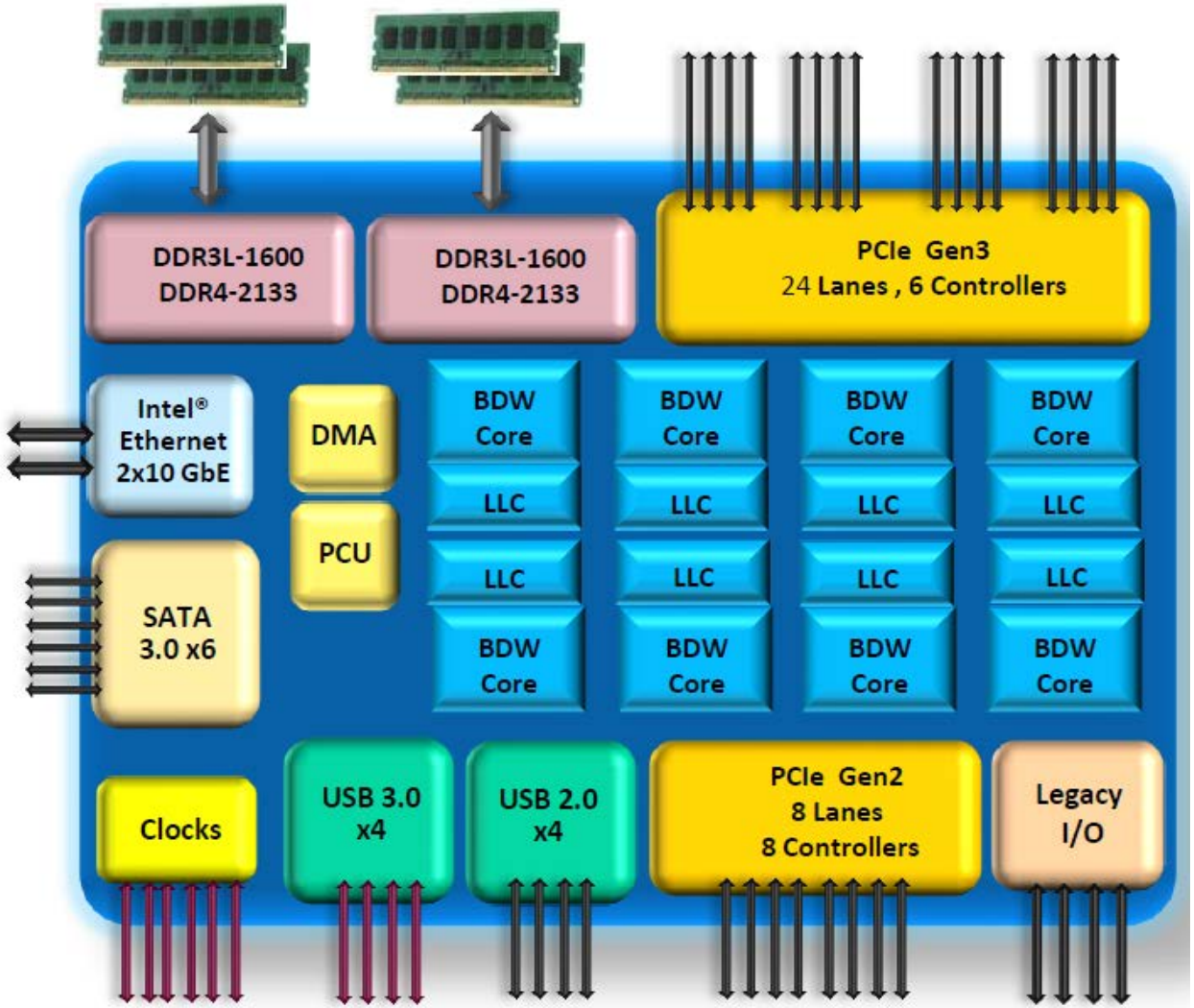
The I/O Access Problem

- **Question: how to transfer data from I/O devices to memory (RAM)?**
- **Trivial solution:**
  - Processor individually reads or writes every word
  - Transferred to/from I/O through an internal register to memory
- **Problems:**
  - Extremely inefficient – can occupy a processor for 1000's of cycles
  - Pollute cache

DMA

- **DMA – Direct Memory Access**
- **A modern solution to the I/O access problem**
- **The peripheral I/O can issue read/write commands directly to the memory**
  - Through the main memory controller
  - The processor does not need to execute any operation
- **Write: The processor is notified when a transaction is completed (interrupt)**
- **Read: The processor issues a signal to the I/O when the data is ready in memory**

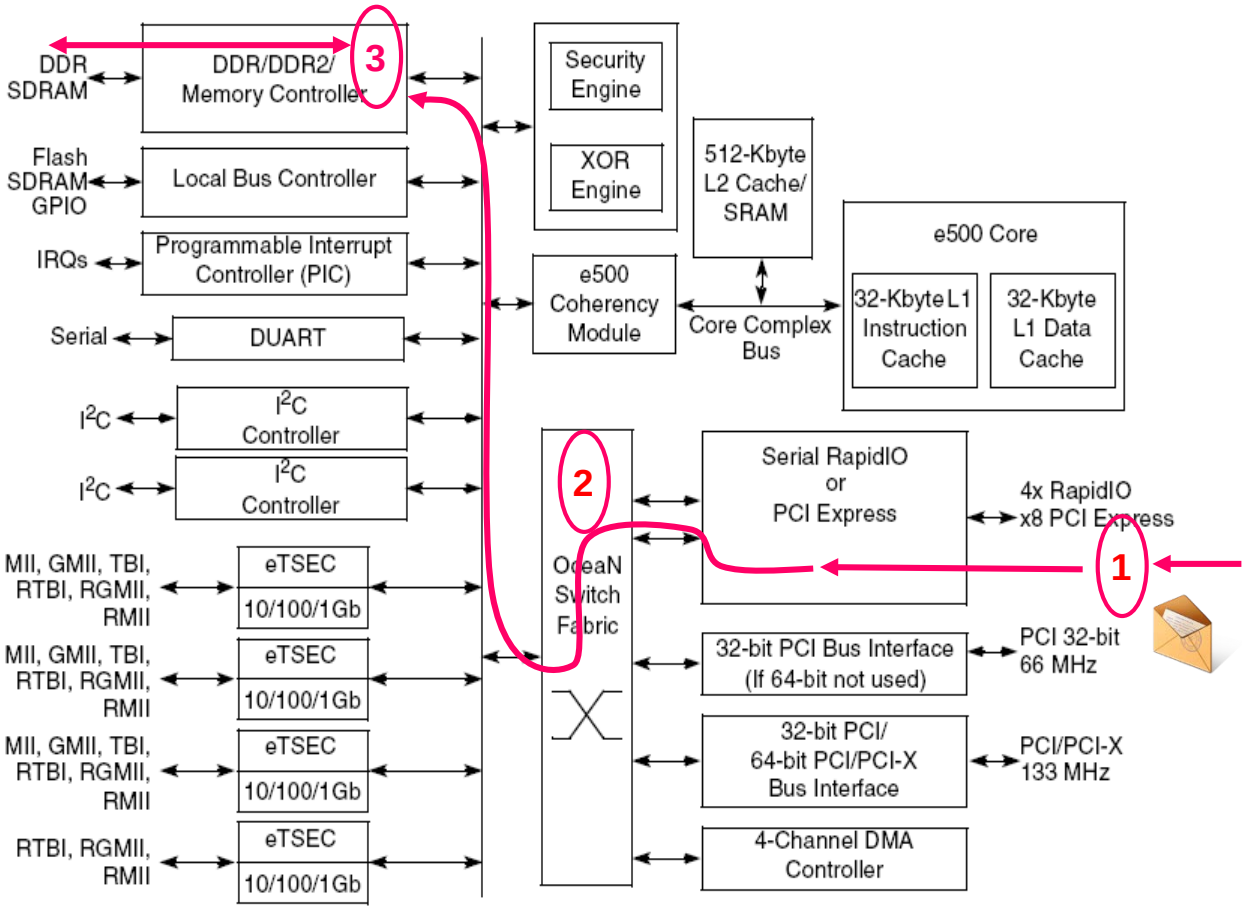
Example – Intel Xeon D



Example (Embedded Processor)

Memory Mapped Access

1. Message arrives on I/O interface.
   Message is decoded to Mem read/write.
   Address is converted to internal address.
2. Mem Read/Write command goes through the switch to the internal bus and memory controller.
3. Memory controller executes the command to the DRAM.
   Returns data if required in the same manner.



DMA

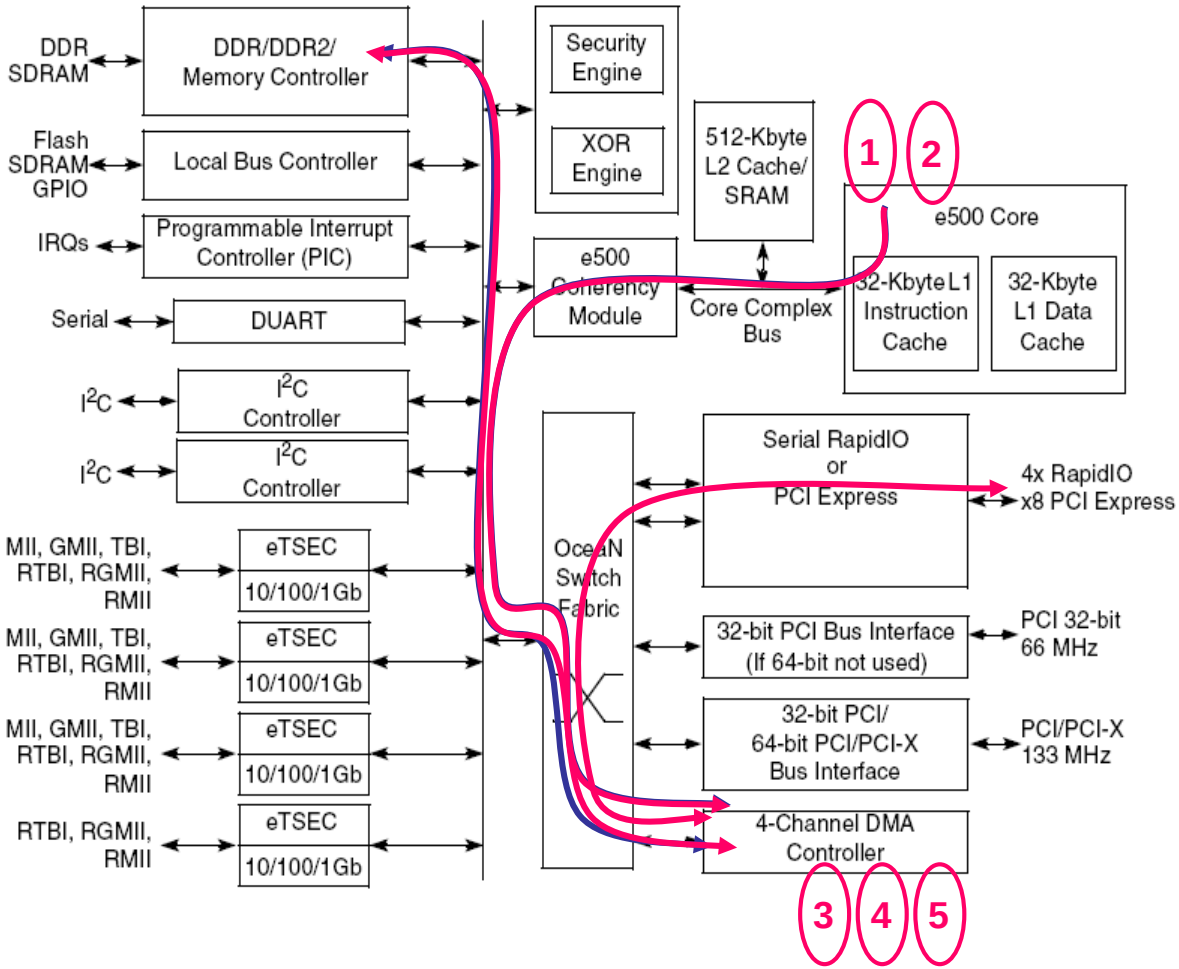
- **DMA accesses are usually handled in *buffers***
  - Single word/block is typically inefficient
- **The processors assigns the peripheral unit the buffers in advance**
- **The buffers are typically handled by *buffer descriptors***
  - Pointer to the buffer in the memory
  - May point to the next buffer as well
  - Indicates buffer status: Owner, valid etc.
  - May include additional buffer properties as well

Example (Embedded Processor)

DMA Access

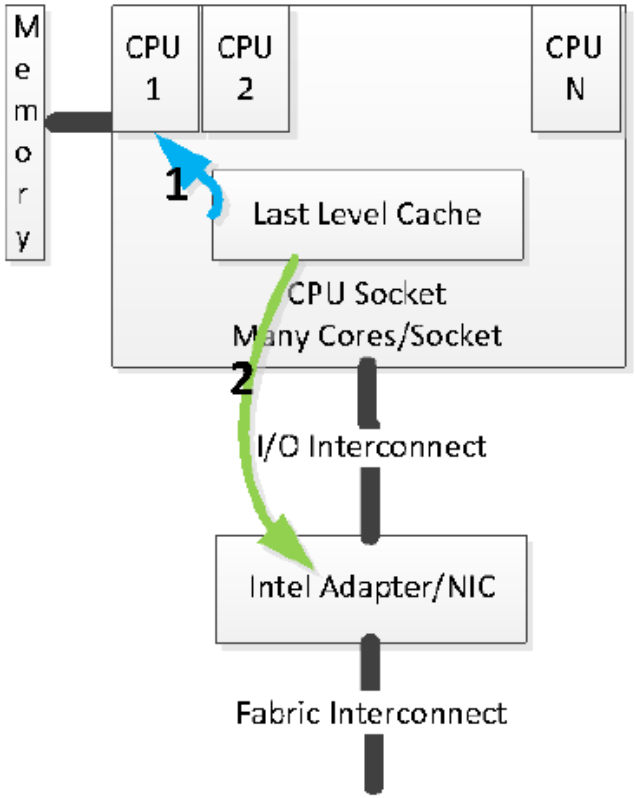
Transfers blocks of data between external interfaces and local address space

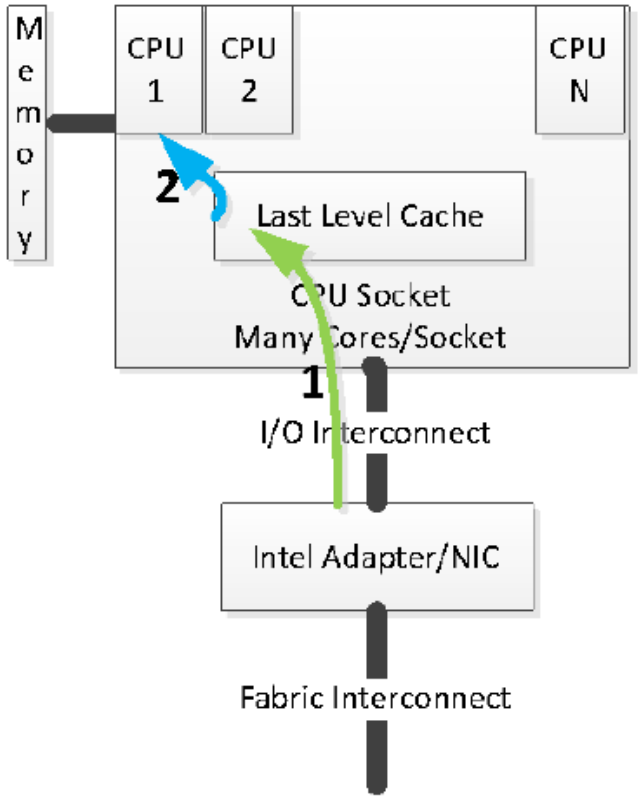
1. **A transfer is started by SW writing to DMA engine configuration registers**
2. **SW Polls DMA channel state to idle and sets trigger**
3. **DMA engine fetches a descriptor from memory**
4. **DMA engine reads block of data from source**
5. **DMA engine writes data to destination**



Intel Data Direct I/O (DDIO)

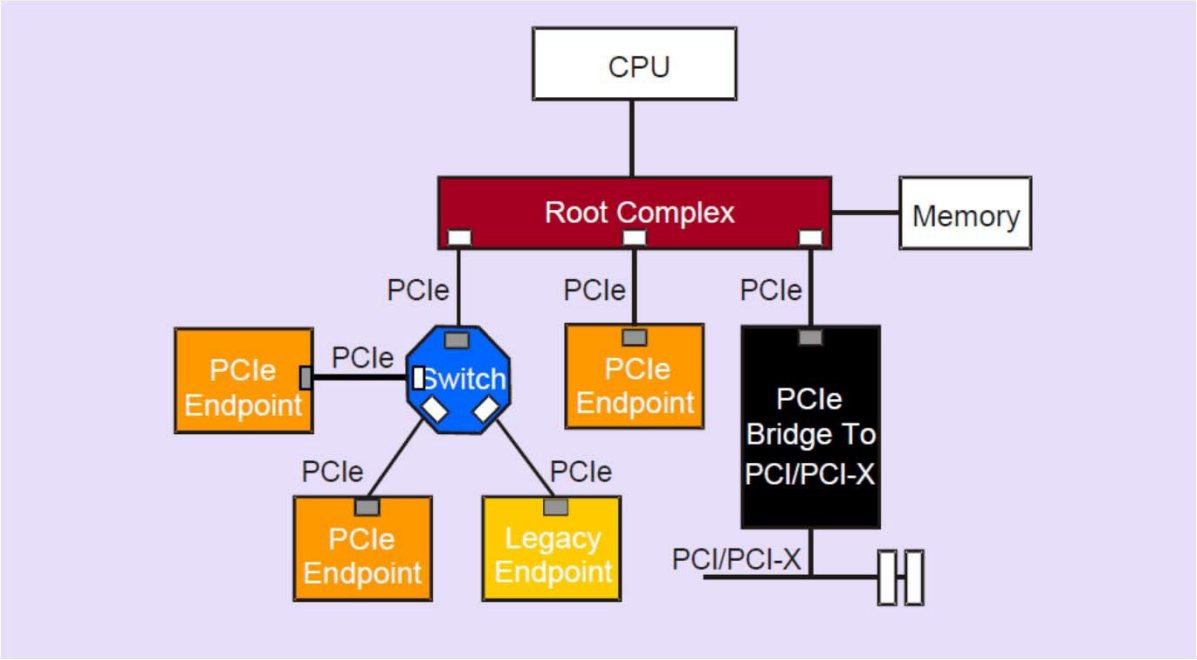
- **Data is written and read directly to/from the last level cache (LLC)**





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)



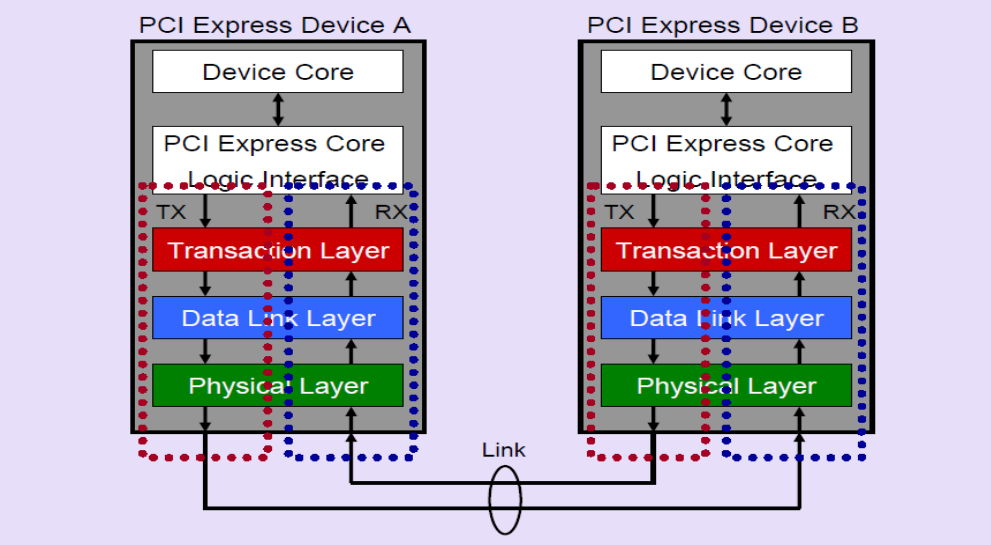
Provides:

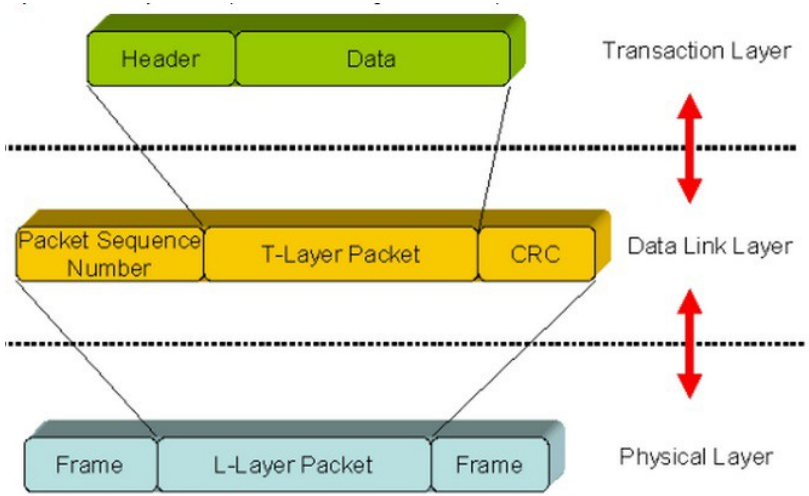
- Processor independence & buffered isolation
- Bus mastering
- Plug and Play operation

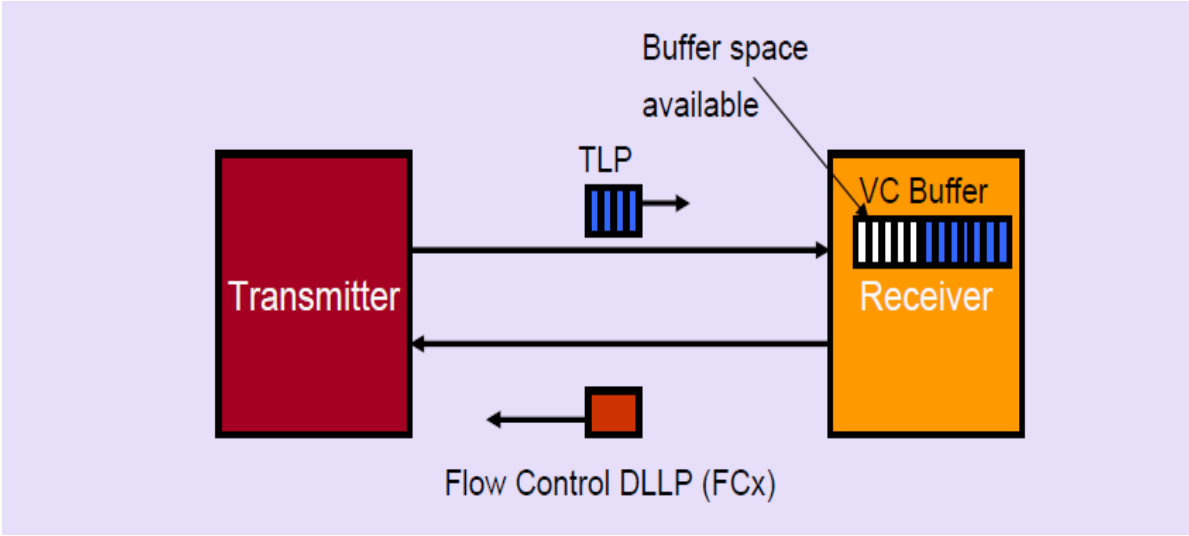
PCIe transaction types

- **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:

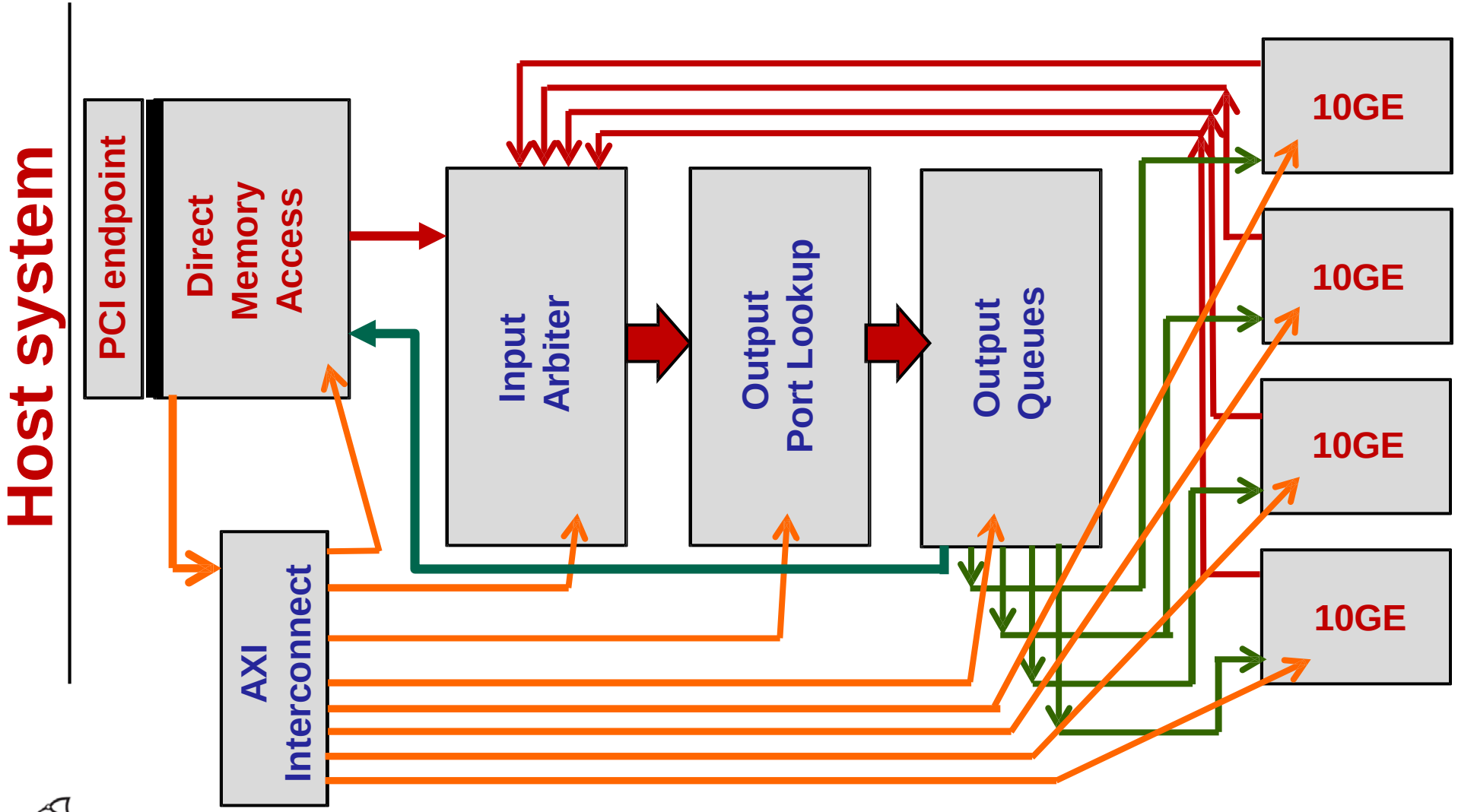
1. **Message Signaled Interrupts (MSI)**
   - interrupt the CPU by writing to a specific address in memory with a payload of 1 DW

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

Reference NIC project

4x port NIC architecture:

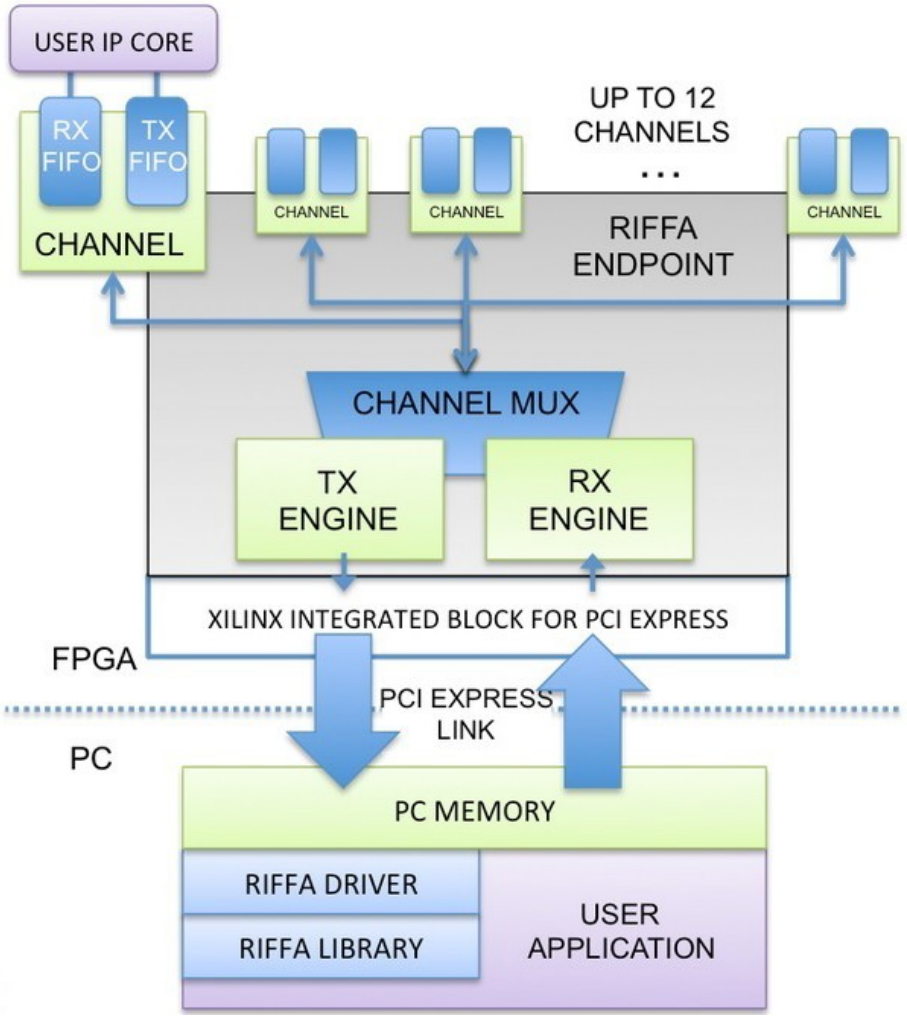


RIFFA

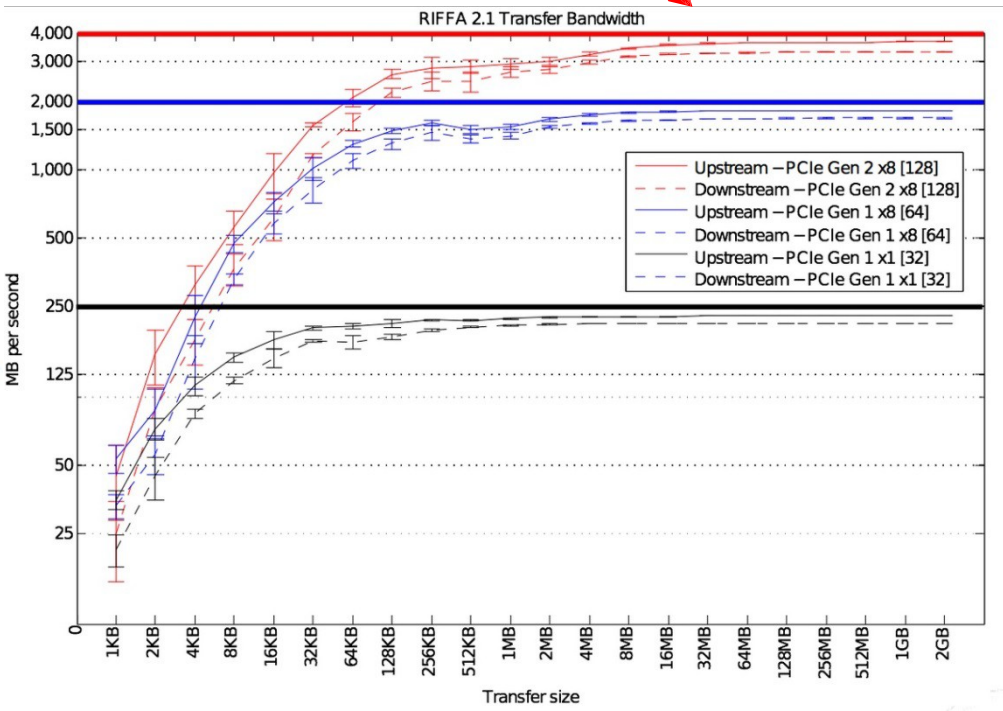
RIFFA (Reusable Integration Framework for FPGA Accelerators)

- 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

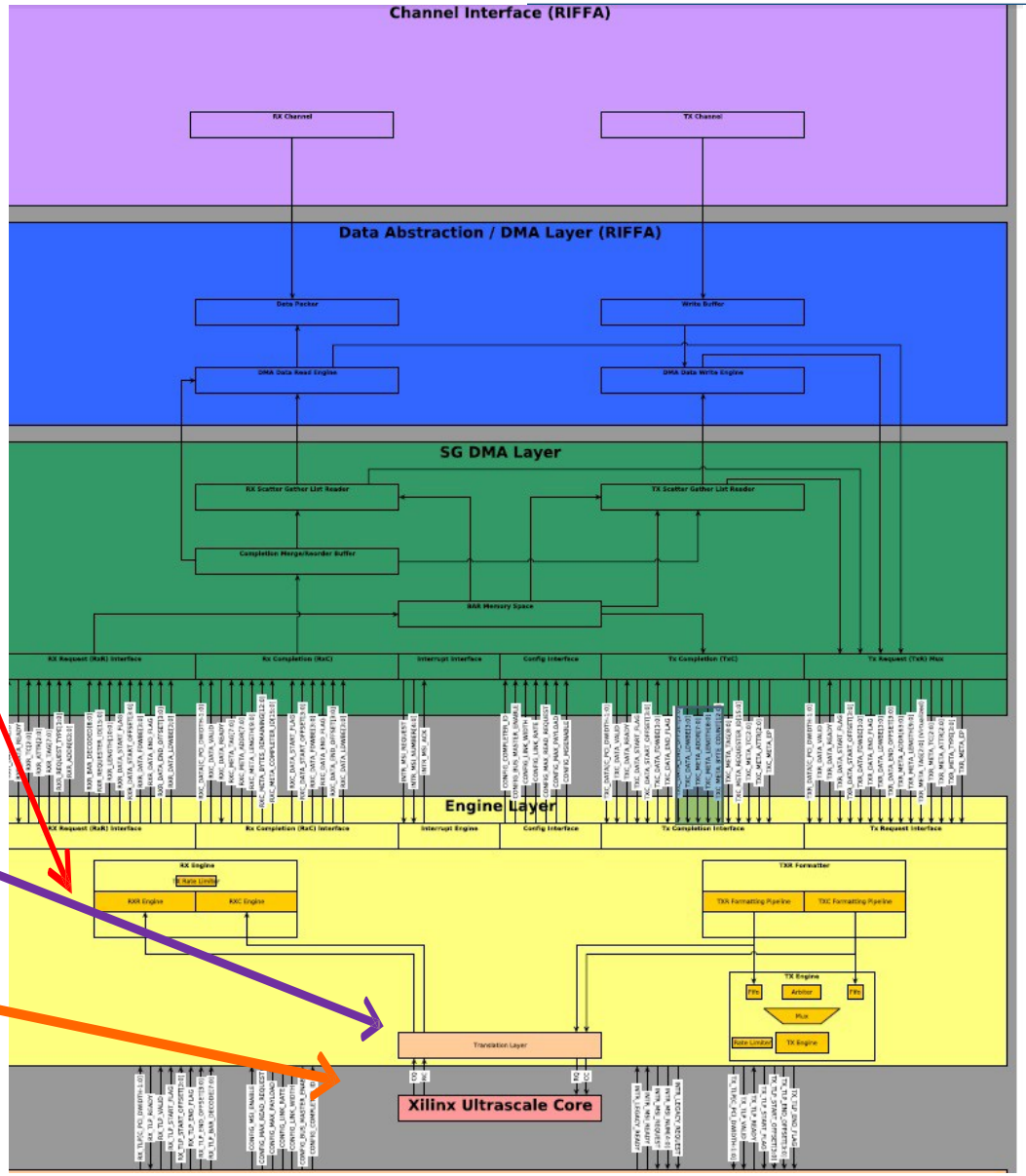


achieves 76% of the theoretical max



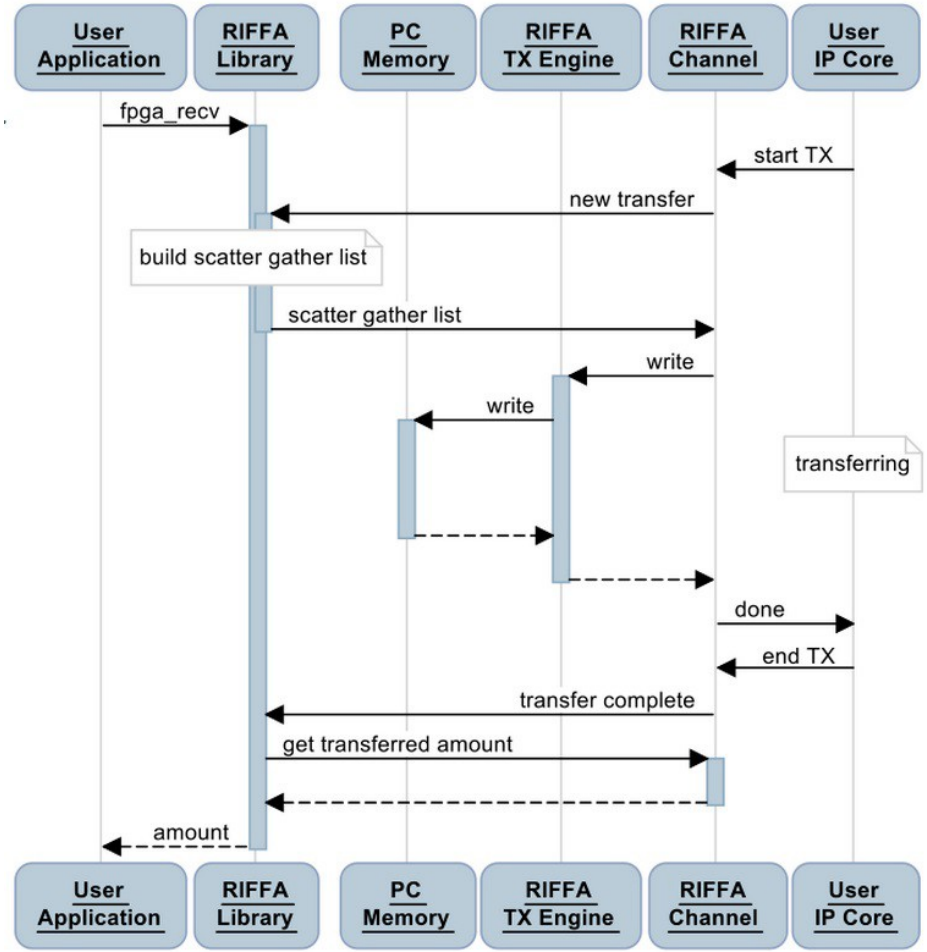
RIFFA architecture

- 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

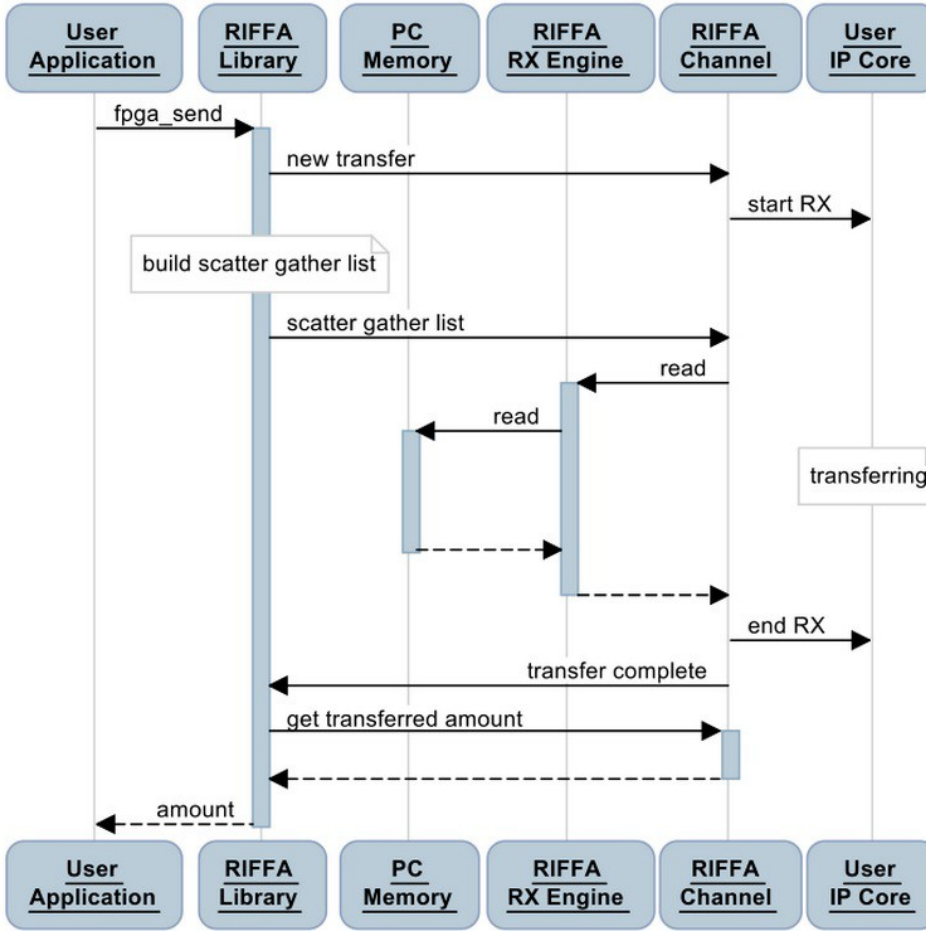


RIFFA Data transfer example

FPGA → Host



Host → FPGA



RIFFA Data transfer example (cont.)

Parameter	Value
Data Transfer Length	128 (32-bit words)
Data Transfer Offsfet	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

RIFFA Data transfer example (cont.)

Parameter	Value
Data Transfer Length	128 (32-bit words)
Data Transfer Offsfet	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

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

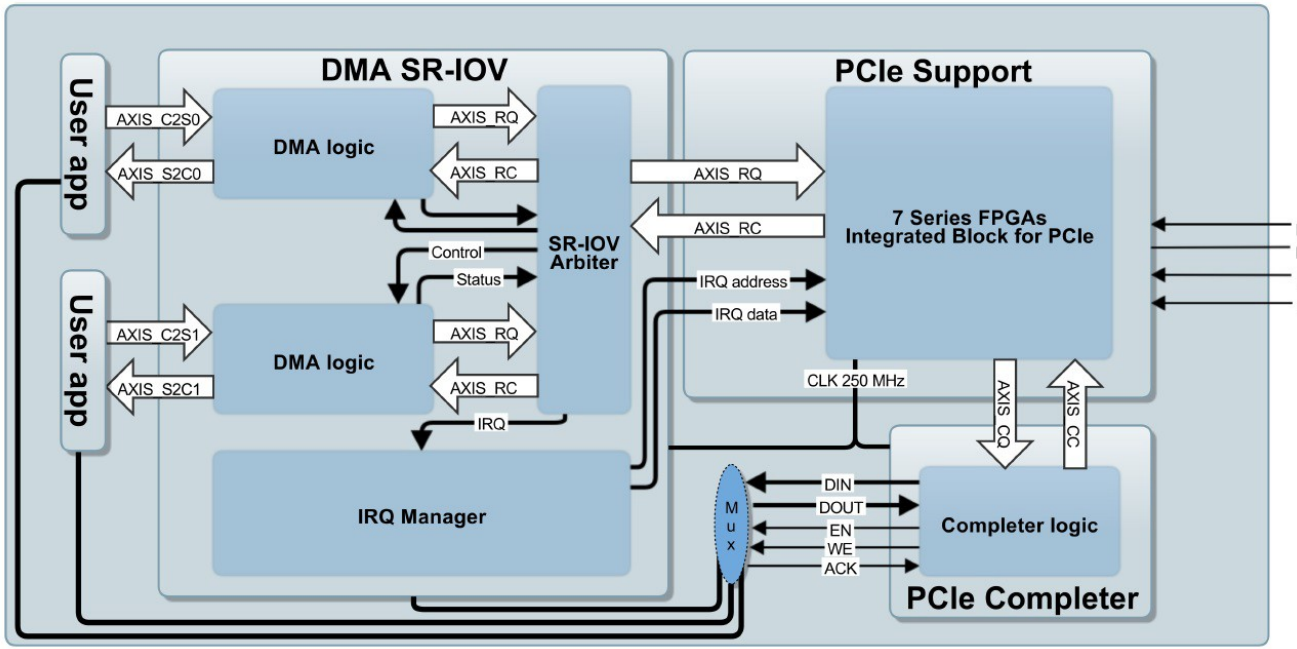
- Based on an interface of the card (can have multiple cards)
- Uses standard struct ifreq with struct sume_ifreq data pointer
- Supports read write operations on registers (see: nf_sume.h, rwaxi tool)
- Second PCIe channel
- Only one outstanding register r/w possible at a time
- Writing initiates full 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

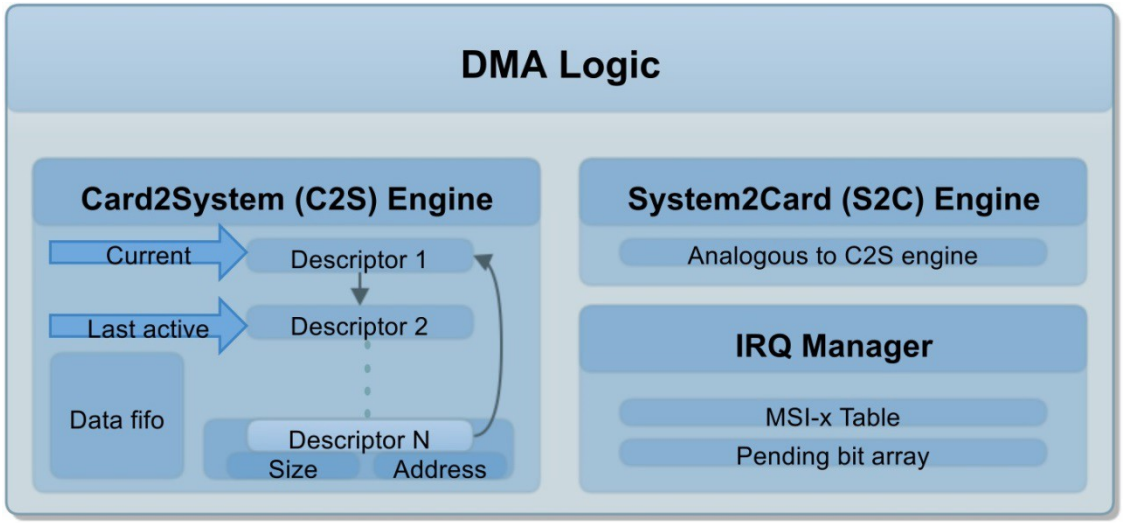
UAM DMA

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

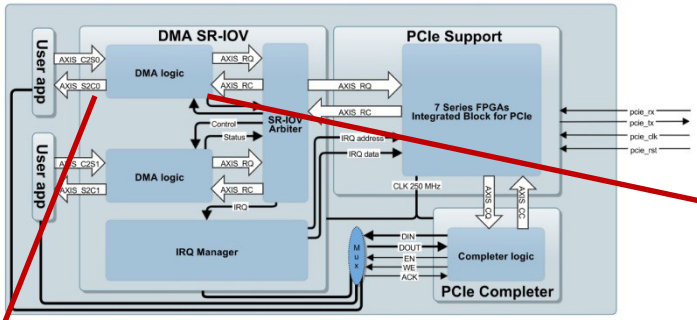
- Supports PCIe Gen 3.0 x8 speeds
- Designed to be lightweight and easy to understand
- Tailored for Xilinx platform only
- Designed for virtualized environments (SR-IOV)
- Has been tested on Linux platform

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

System to card transfer, S2C

Transmit data from the FPGA to the host through Memory Write Request Operations

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 Assoaciated elements

Operation

Operation

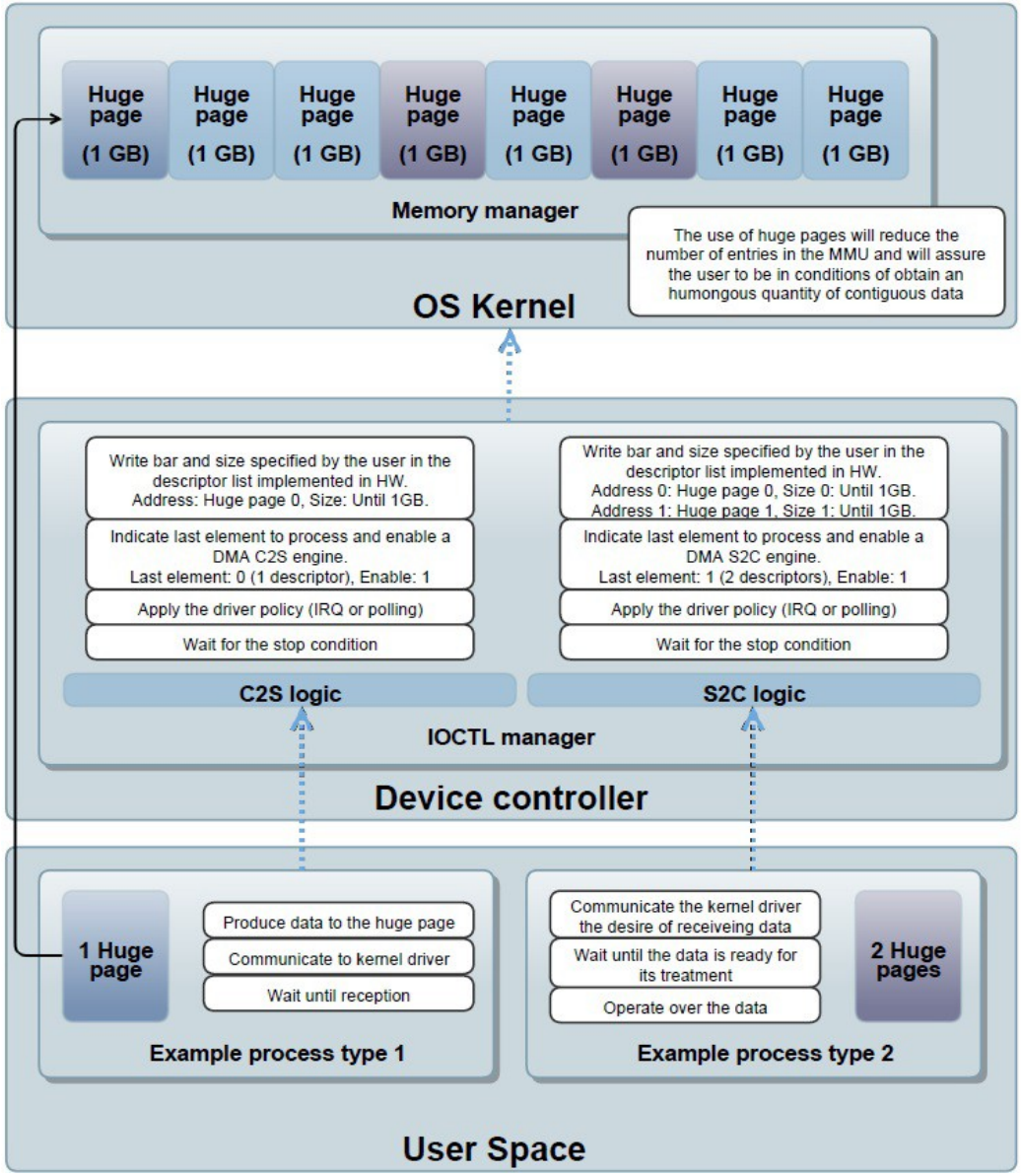
Core

Future inclusion

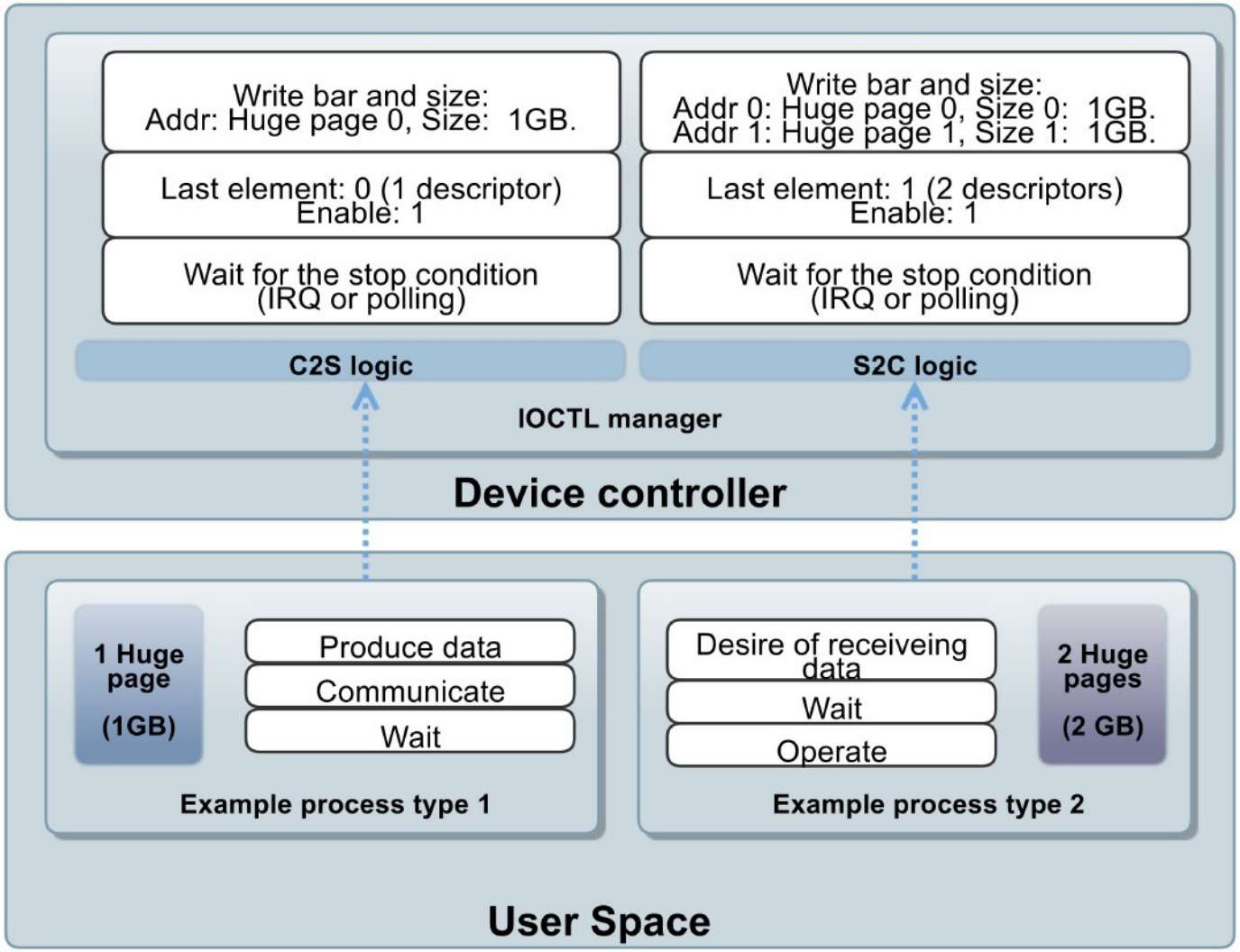
Currently provided

Component

SW/HW perspective



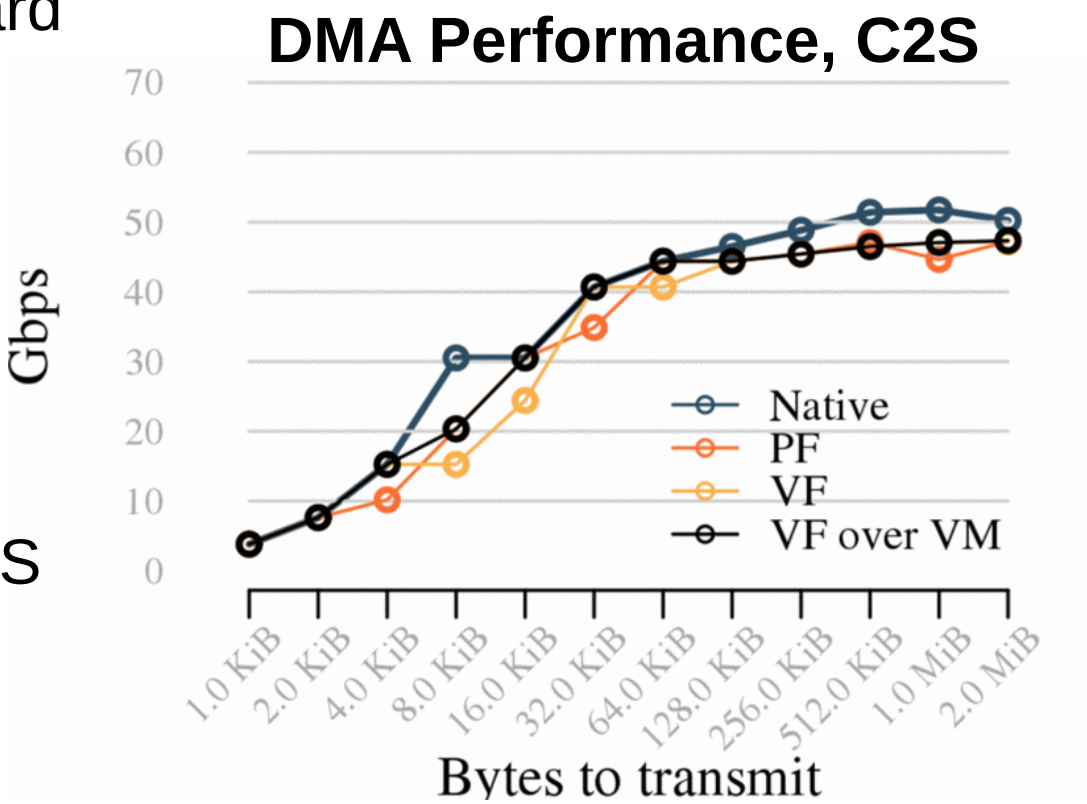
SW/HW perspective (cont.)



Evaluation

Setup:

- Supermicro X9DRD-iF motherboard
- Dual Intel Xeon CPU E5-2650 v2 @ 2.60GHz
- 64 GiB of DDR3 RAM clocked at 1600 MHz
- NetFPGA SUME board
- One VM and one VF
  - using KVM
  - 8GB of RAM
  - Up to 4 cores
- Enabled VT-x, VT-d and SR-IOV
  - Both BIOS and OS



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, Hwanju Kim, Marco Forconesi, Jinyun Zhang, Bjoern Zeeb, Robert Watson, Salvator Galea, Marcin Wojcik, Diana Andreea Popescu, Murali Ramanujam

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, Stephen Gabriel Ibanez

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 , Andreas Fiessler, Robert Soule, Pietro Bressana, Yuta Tokusashi

Steve Wang, Erik Cengar, Michael Alexander, Sam Bobrowicz, Garrett Aufdemberg, Patrick Kane, Tom Weldon

Patrick Lysaght, Kees Vissers, Michaela Blott, Shep Siegel, Cathal McCabe

Acknowledgements (II)