NetFPGA Summer Course

Presented by:

Noa Zilberman

Yury Audzevich

Technion

August 2 – August 6, 2015

http://NetFPGA.org

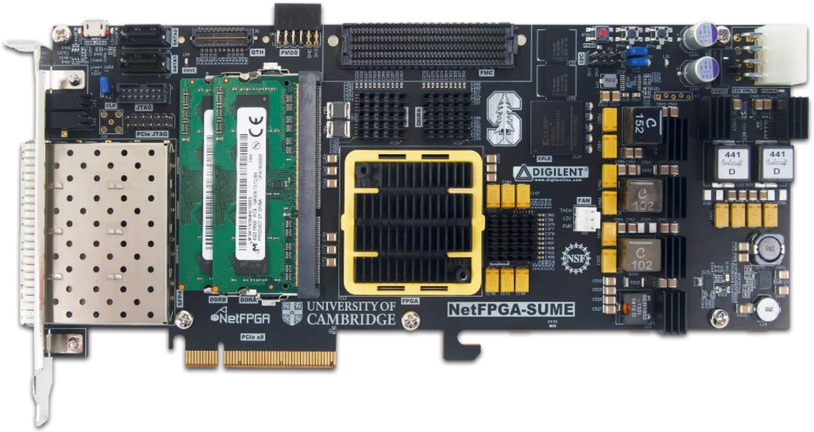
Day 1 Outline

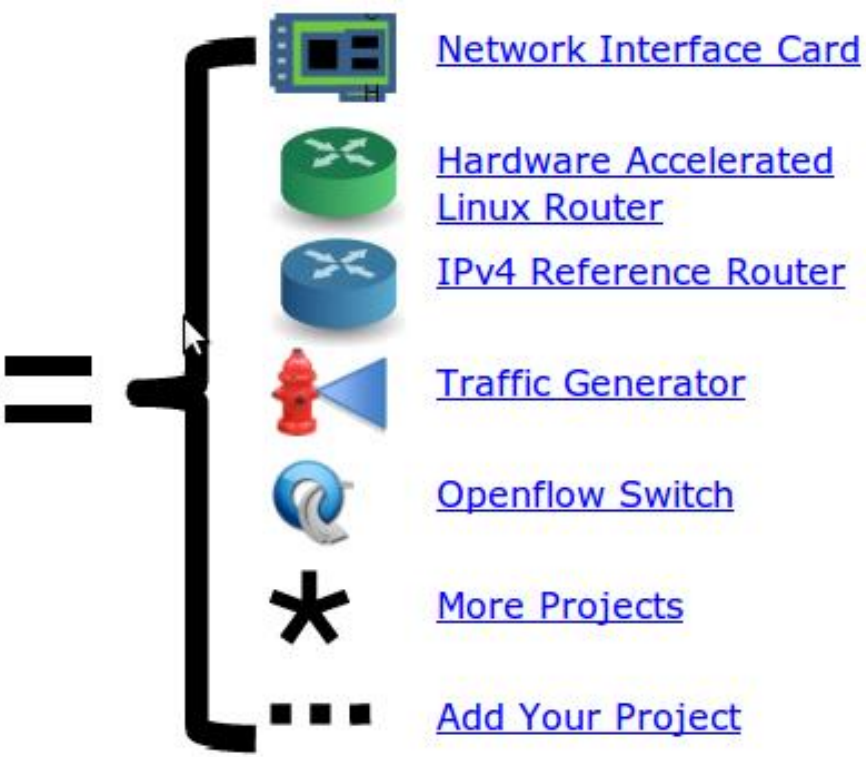
- **The NetFPGA platform**
  - Introduction
  - Overview of the NetFPGA Platform
- **NetFPGA SUME**
  - Hardware overview
- **Network Review**
  - Basic IP review
- **The Base Reference Switch**
  - Example I: Reference switch running on the NetFPGA
- **The Life of a Packet Through the NetFPGA**
  - Hardware Datapath
  - Interface to software: Exceptions and Host I/O
- **Infrastructure**
  - Tree
  - Verification Infrastructure
- **Examples of Using NetFPGA**
- **Example Project: Crypto Switch**
  - Introduction to a Crypto Switch
  - What is an IP core?
  - Getting started with a new project.
  - Crypto FSM
- **Simulation and Debug**
  - Write and Run Simulations for Crypto Switch
- **Concluding Remarks**

Section I: The NetFPGA platform

NetFPGA = Networked FPGA

A line-rate, flexible, open networking platform for teaching and research

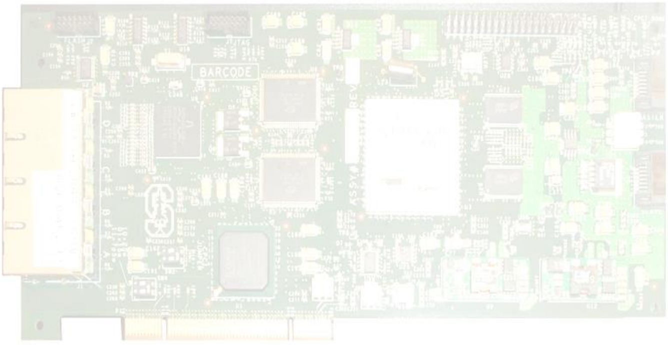




=

- Network Interface Card
- Hardware Accelerated Linux Router
- IPv4 Reference Router
- Traffic Generator
- Openflow Switch
- More Projects
- Add Your Project

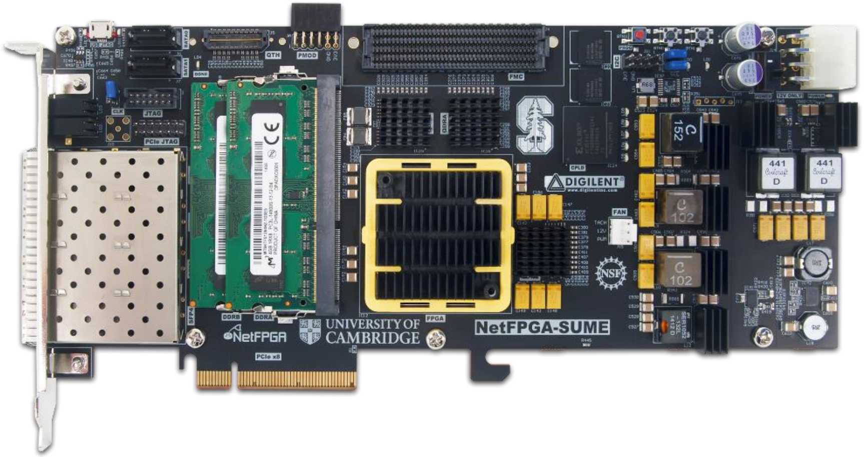
NetFPGA Family of Boards



NetFPGA-1G (2006)

NetFPGA-10G (2010)

NetFPGA-1G-CML (2014)



NetFPGA SUME (2014)

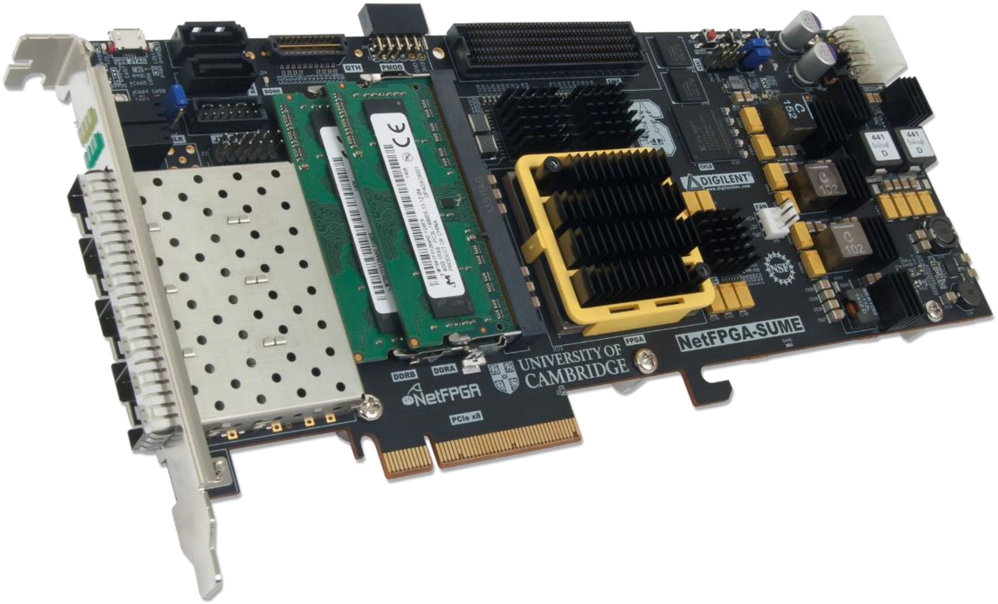
NetFPGA consists of…

Four elements:

NetFPGA GitHub Organization

The Interwebs http://www.netfpga.org

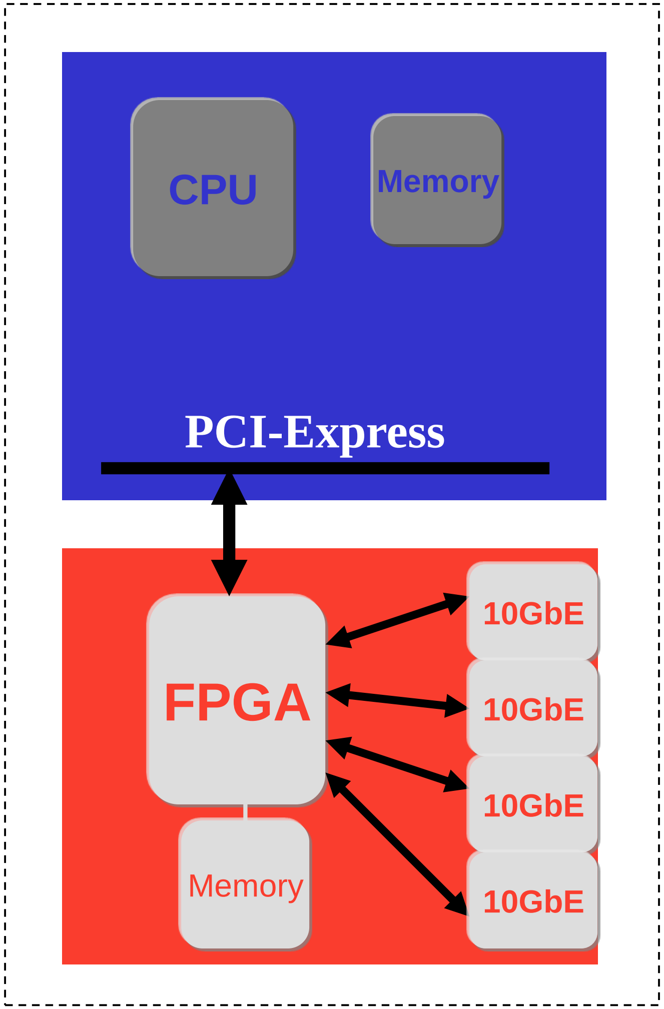
- NetFPGA board
- Tools + reference designs
- Contributed projects
- Community



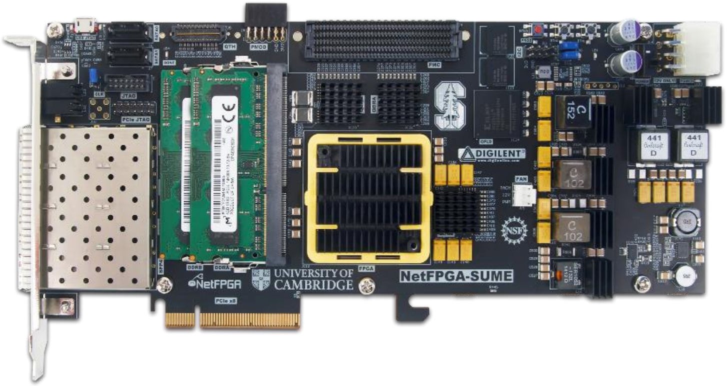
NetFPGA board

Networking Software running on a standard PC

A hardware accelerator built with Field Programmable Gate Array driving 1/10/ 100Gb/s network links



PC with NetFPGA



Tools + Reference Designs

Tools:

- Compile designs
- Verify designs
- Interact with hardware

Reference designs:

- Router (HW)
- Switch (HW)
- Network Interface Card (HW)
- Router Kit (SW)
- SCONE (SW)

Community

Wiki

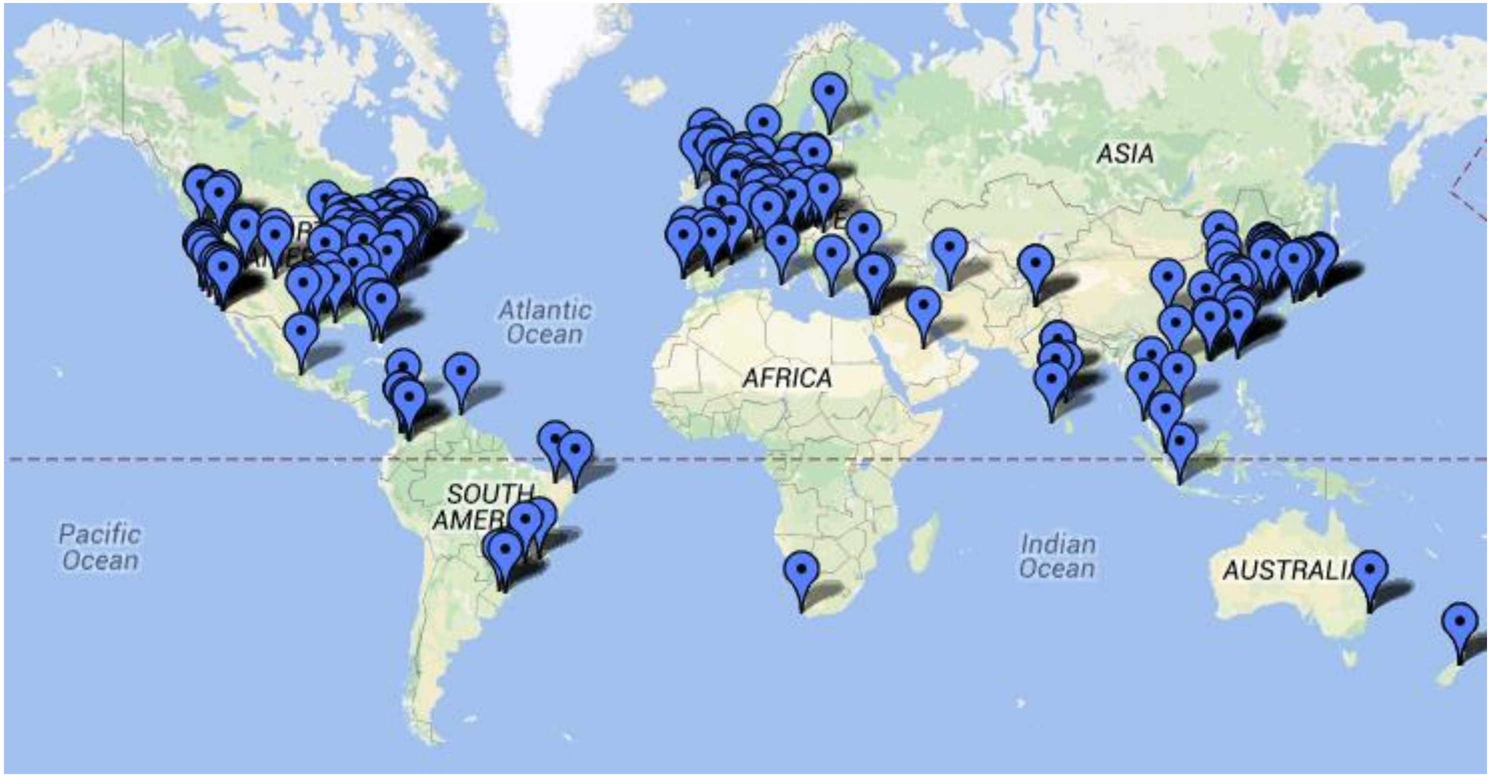
- **Documentation**
  - – User's Guide *"so you just got your first NetFPGA"*
  - – Developer's Guide *"so you want to build a …"*
- **Encourage users to contribute**

Forums

- **Support by users for users**
- **Active community - 10s-100s of posts/week**

International Community

Over 1,200 users, using over 3500 cards at 150 universities in 40 countries



NetFPGA's Defining Characteristics

- **Line-Rate**
  - Processes back-to-back packets
    - Without dropping packets
    - At full rate
  - Operating on packet headers
    - For switching, routing, and firewall rules
  - And packet payloads
    - For content processing and intrusion prevention

- **Open-source Hardware**
  - Similar to open-source software
    - Full source code available
    - BSD-Style License for SUME, LGPL 2.1 for 10G
  - But harder, because
    - Hardware modules must meet timing
    - Verilog & VHDL Components have more complex interfaces
    - Hardware designers need high confidence in specification of modules

Test-Driven Design

- **Regression tests**
  - Have repeatable results
  - Define the supported features
  - Provide clear expectation on functionality

- ***Example:* Internet Router**
  - Drops packets with bad IP checksum
  - Performs Longest Prefix Matching on destination address
  - Forwards IPv4 packets of length 64-1500 bytes
  - Generates ICMP message for packets with TTL <= 1
  - Defines how to handle packets with IP options or non IPv4

… and dozens more …

Every feature is defined by a regression test

Who, How, Why

Who uses the NetFPGA?

- Researchers
- Teachers
- Students

How do they use the NetFPGA?

- To run the Router Kit
- To build modular reference designs
  - IPv4 router
  - 4-port NIC
  - Ethernet switch, …

Why do they use the NetFPGA?

- To measure performance of Internet systems
- To prototype new networking systems

Summer Course Objectives

- **Overall picture of NetFPGA**
- **How reference designs work**
- **How you can work on a project**
  - – NetFPGA Design Flow
  - – Directory Structure, library modules and projects
  - – How to utilize contributed projects
    - Interface/Registers
  - – How to verify a design (Simulation and Hardware Tests)
  - – Things to do when you get stuck

AND… You build your own projects!

Section II: Hardware Overview

NetFPGA-1G-CML

- FPGA Xilinx Kintex7
- 4x 10/100/1000 Ports
- PCIe Gen.2 x4
- QDRII+-SRAM, 4.5MB
- DDR3, 512MB
- SD Card
- Expansion Slot

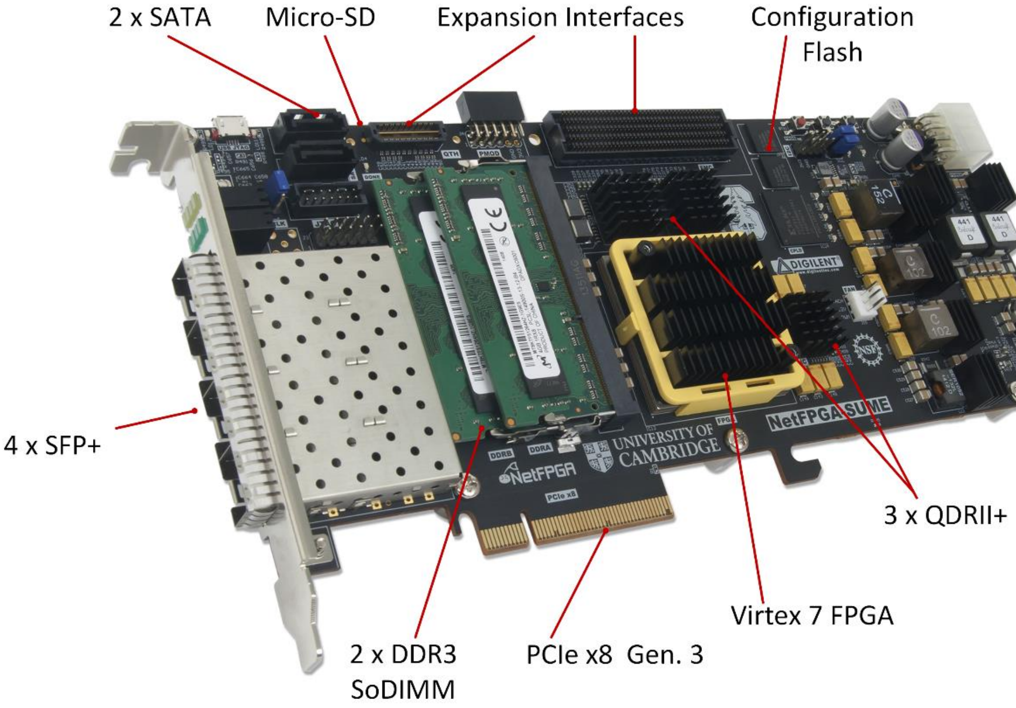
NetFPGA-10G

- **FPGA Xilinx Virtex5**
- **4 SFP+ Cages**
  - 10G Support
  - 1G Support
- **PCIe Gen.1 x8**
- **QDRII-SRAM, 27MB**
- **RLDRAM-II, 288MB**
- **Expansion Slot**

Time for a catch-up…

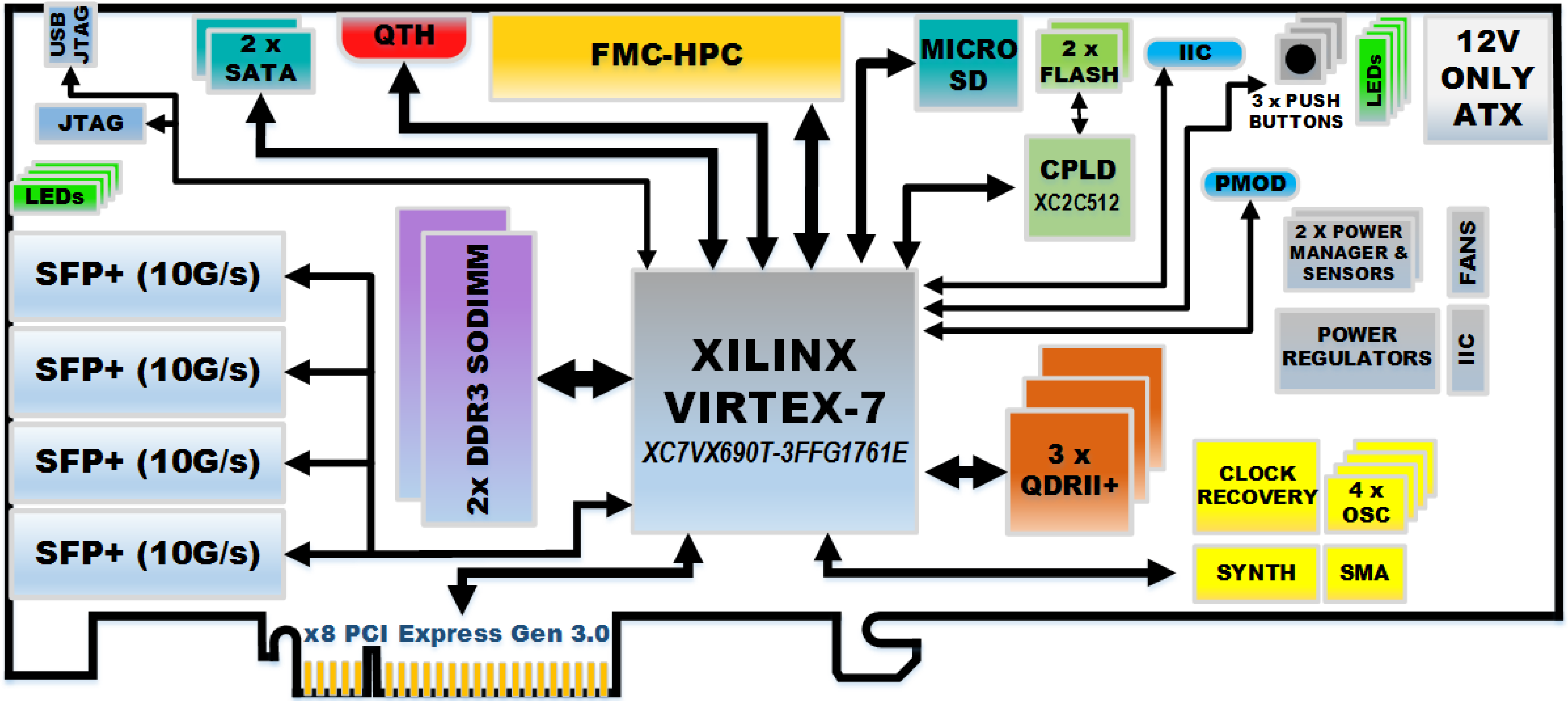
NetFPGA-SUME

- **A major upgrade over the NetFPGA-10G predecessor**
- **State-of-the-art technology**



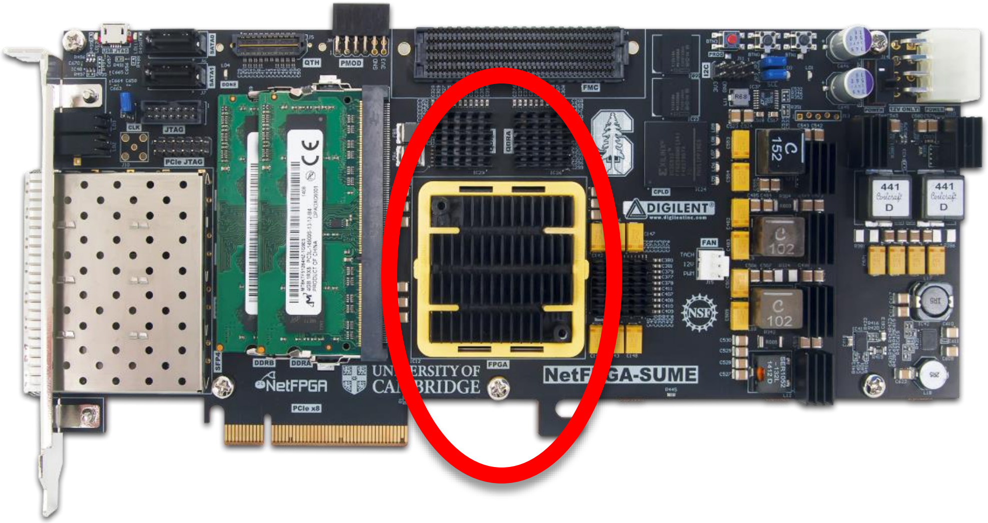
NetFPGA-SUME

- **High Level Block Diagram**



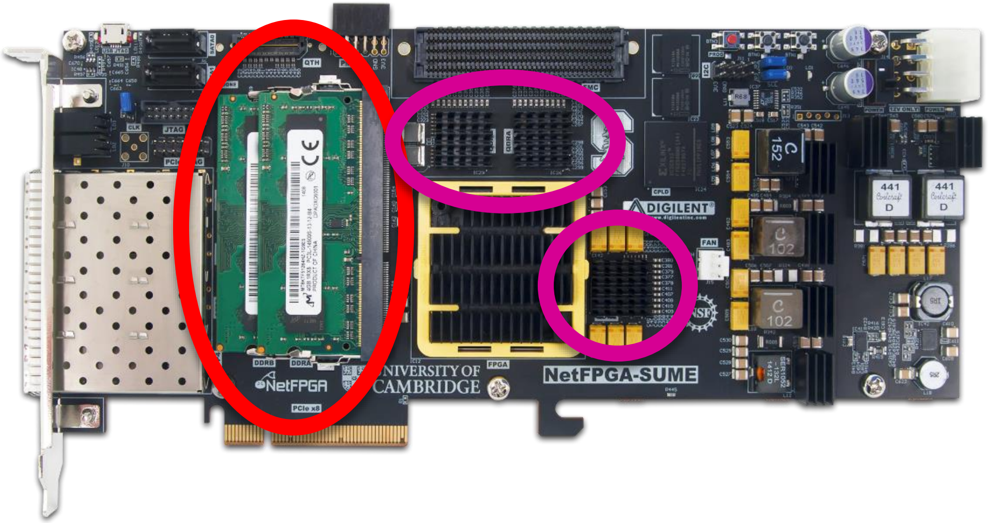
Xilinx Virtex 7 690T

- Optimized for high-performance applications
- 690K Logic Cells
- 52Mb RAM
- 3 PCIe Gen. 3 Hard cores



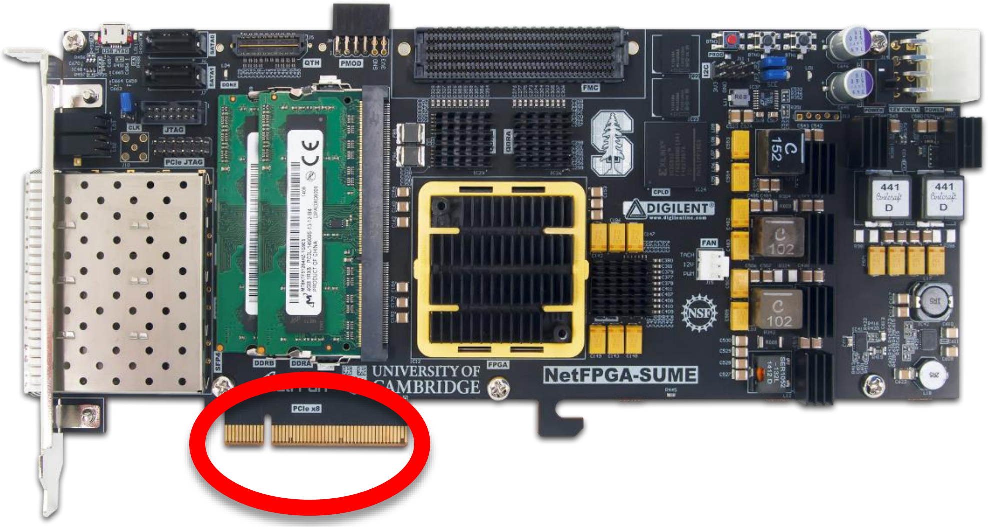
Memory Interfaces

- **DRAM:**
  2 x DDR3 SoDIMM
  1866MT/s, 4GB
- **SRAM:**
  3 x 9MB QDRII+,
  500MHz



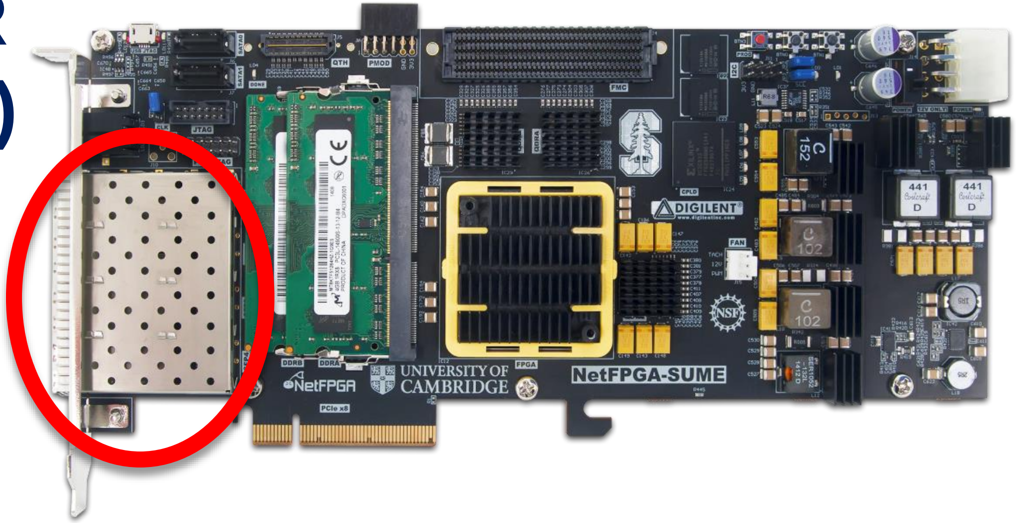
Host Interface

- PCIe Gen. 3
- x8 (only)
- Hardcore IP



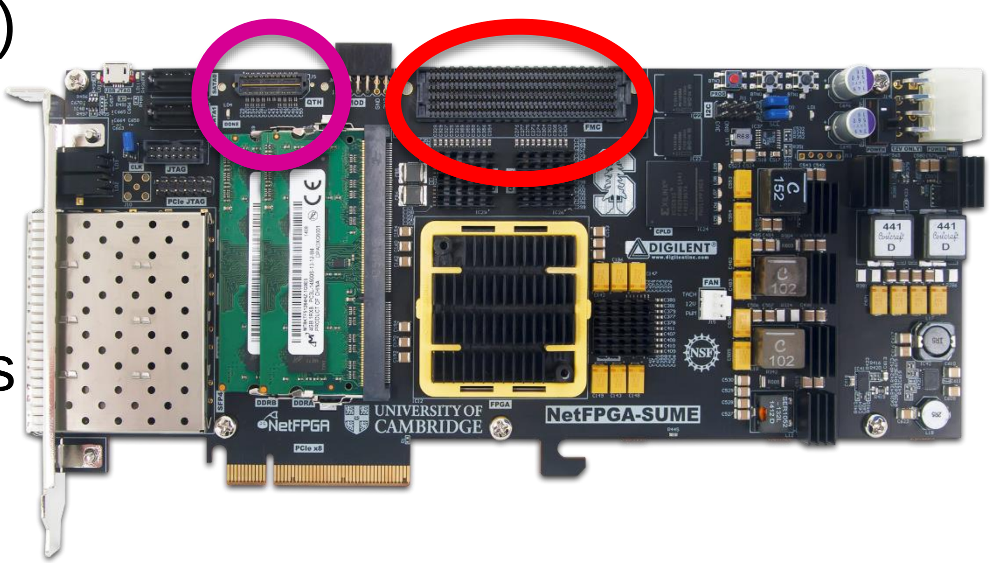
Front Panel Ports

- 4 SFP+ Cages
- Directly connected to the FPGA
- Supports 10GBase-R transceivers (default)
- Also Supports 1000Base-X transceivers and direct attach cables



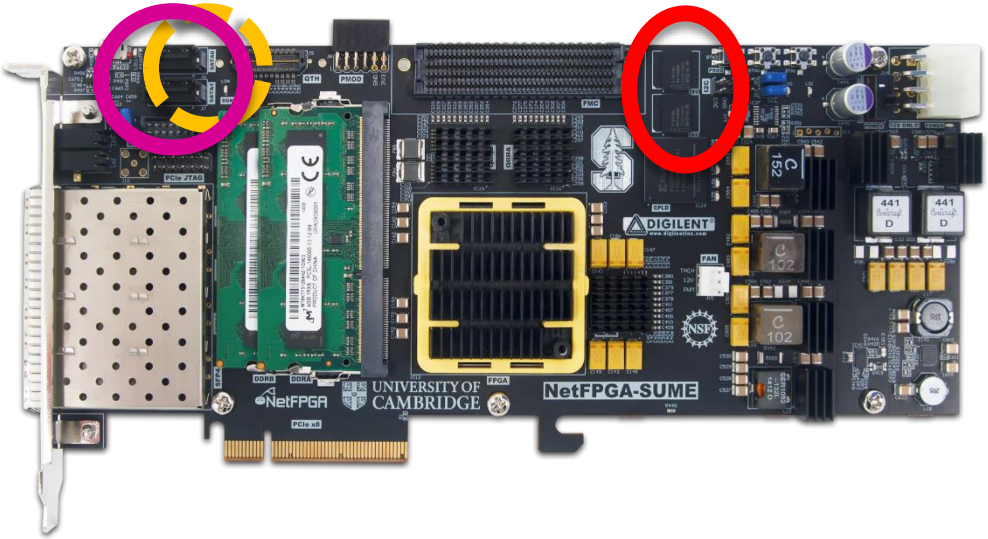
Expansion Interfaces

- **FMC HPC connector**
  - VITA-57 Standard
  - Supports Fabric Mezzanine Cards (FMC)
  - 10 x 12.5Gbps serial links
- **QTH-DP**
  - 8 x 12.5Gbps serial links

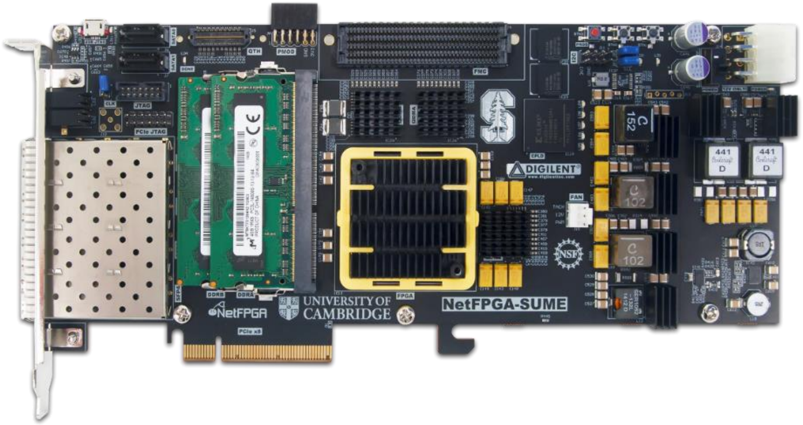


Storage

- **128MB FLASH**
- **2 x SATA connectors**
- **Micro-SD slot**
- **Enable standalone operation**

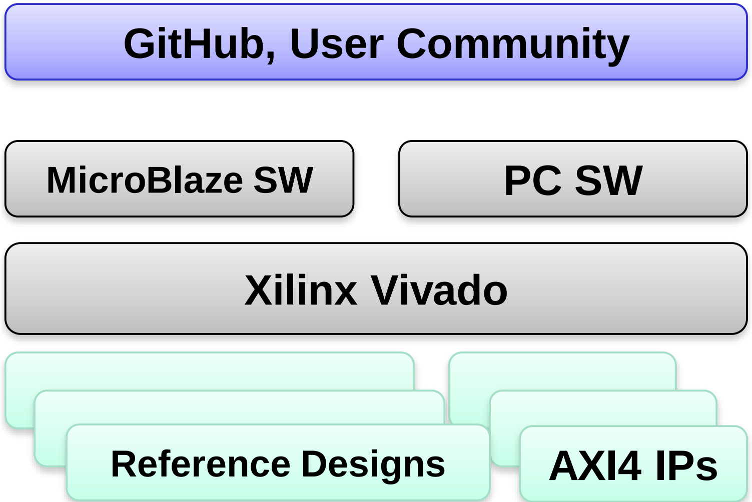


NetFPGA Board Comparison



NetFPGA SUME	NetFPGA 10G
Virtex 7 690T -3	Virtex 5 TX240T
8 GB DDR3 SoDIMM 1800MT/s	288 MB RLDRAM-II 800MT/s
27 MB QDRII+ SRAM, 500MHz	27 MB QDRII-SRAM, 300MHz
x8 PCI Express Gen. 3	x8 PCI Express Gen. 1
4 x 10Gbps Ethernet Ports	4 x 10Gbps Ethernet Ports
18 x 13.1Gb/s additional serial links	20 x 6.25Gb/s additional serial links

Beyond Hardware


GitHub, User Community

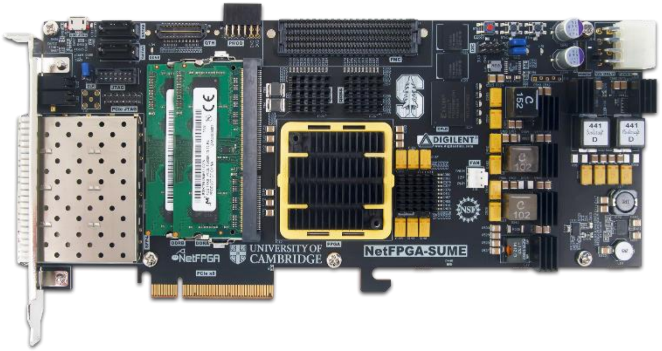
MicroBlaze SW

PC SW

Xilinx Vivado

Reference Designs

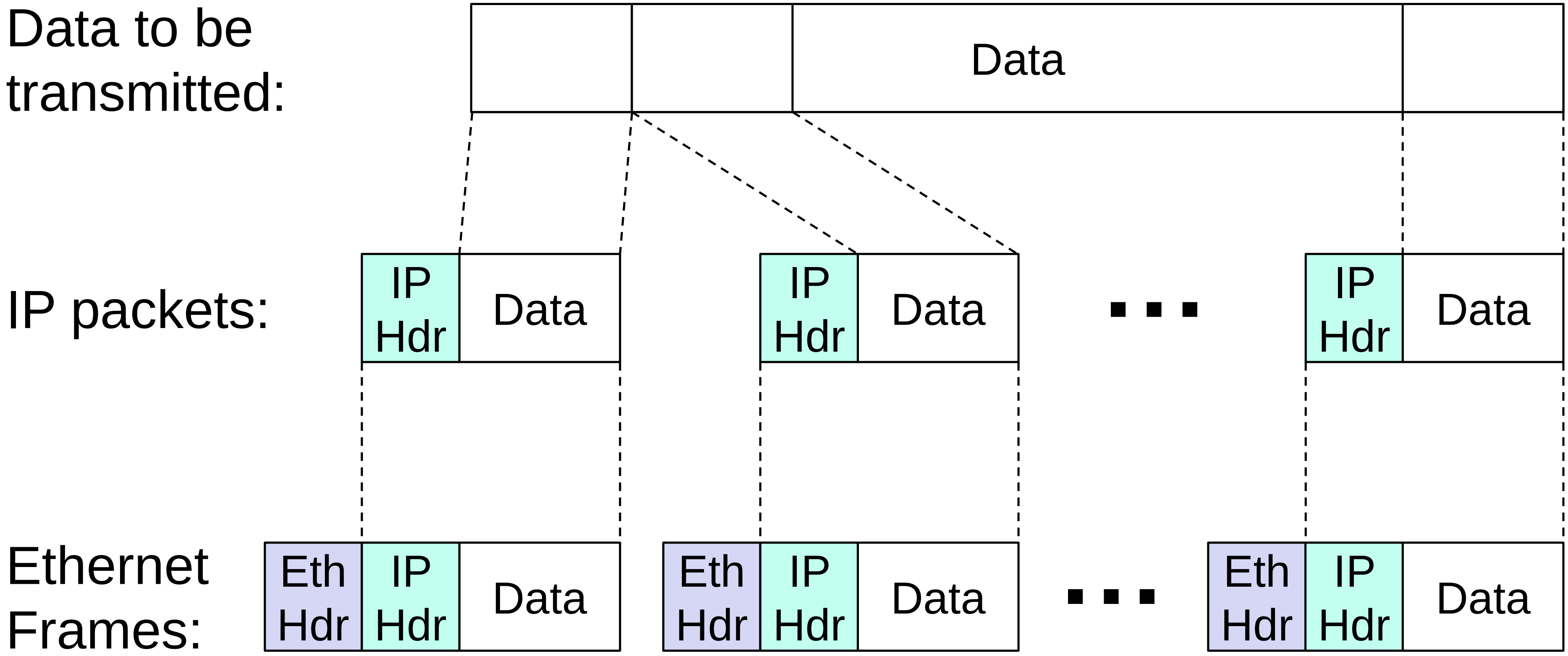
AXI4 IPs



- **NetFPGA Board**
- **Xilinx Vivado based IDE**
- **Reference designs using AXI4**
- **Software (embedded and PC)**
- **Public Repository**
- **Public Wiki**

Section II: Network review

Internet Protocol (IP)



Data to be transmitted:

Data

IP packets:

IP Hdr | Data

IP Hdr | Data

. . .

IP Hdr | Data

Ethernet Frames:

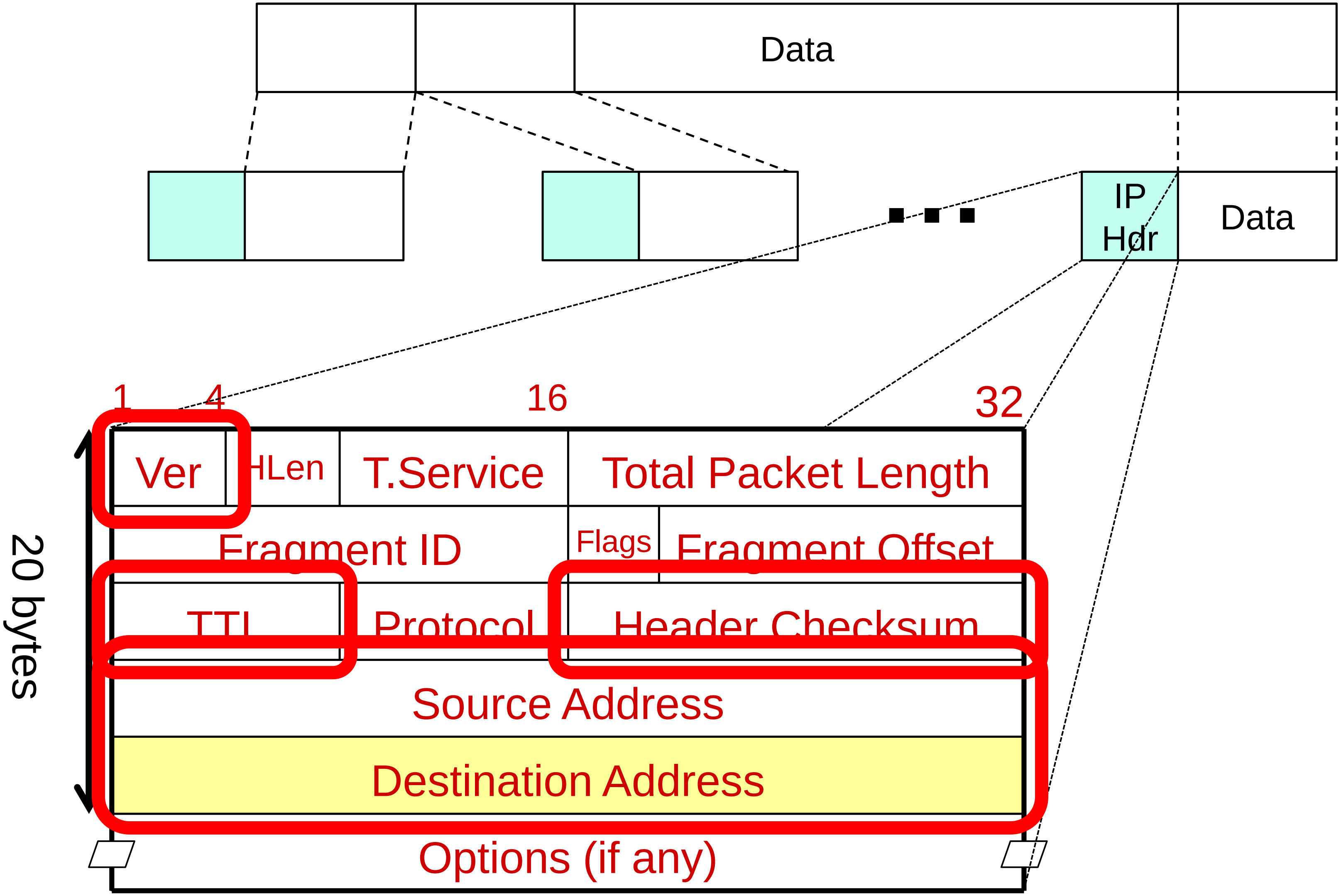
Eth Hdr | IP Hdr | Data

Eth Hdr | IP Hdr | Data

. . .

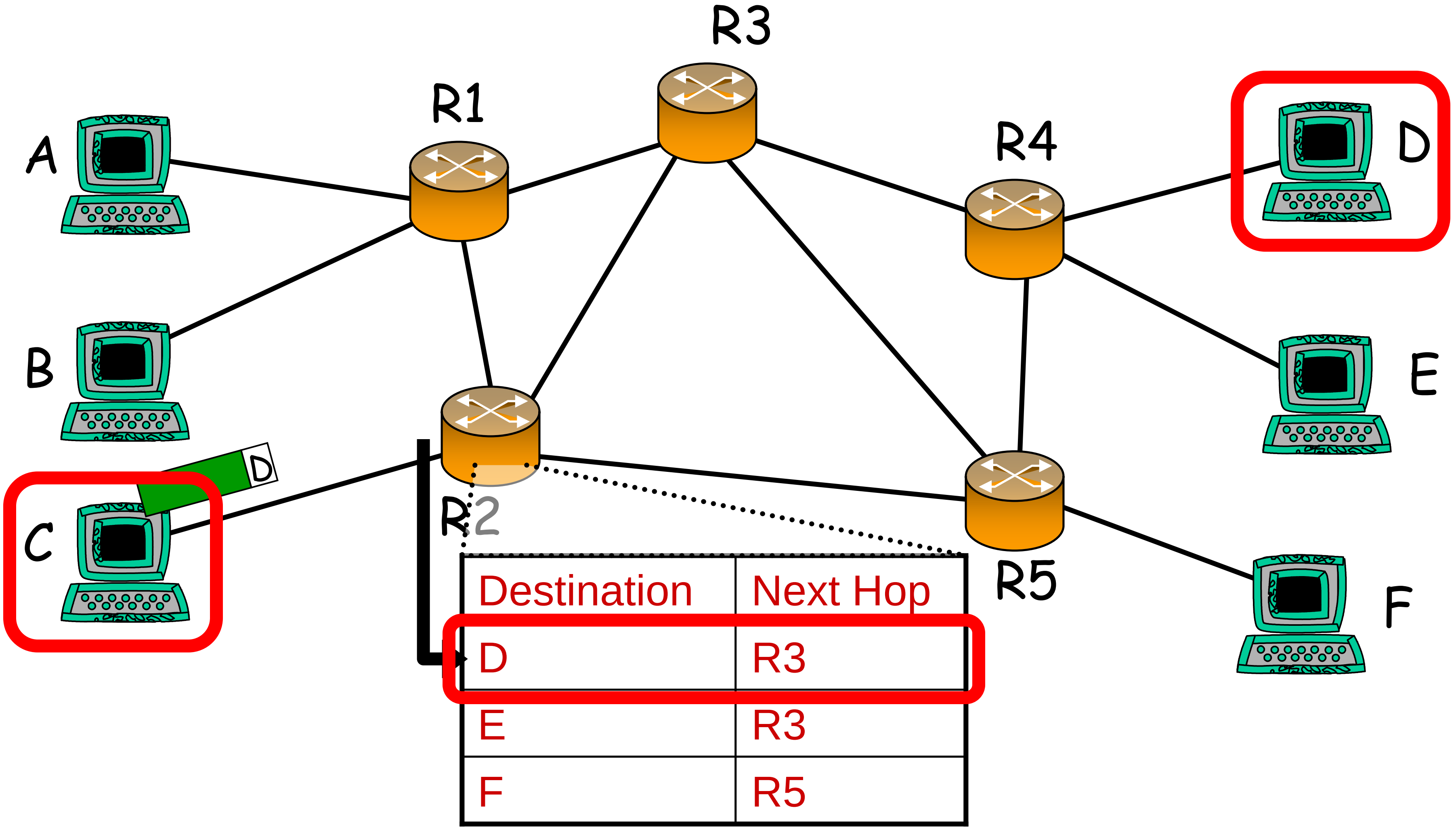
Eth Hdr | IP Hdr | Data

Internet Protocol (IP)



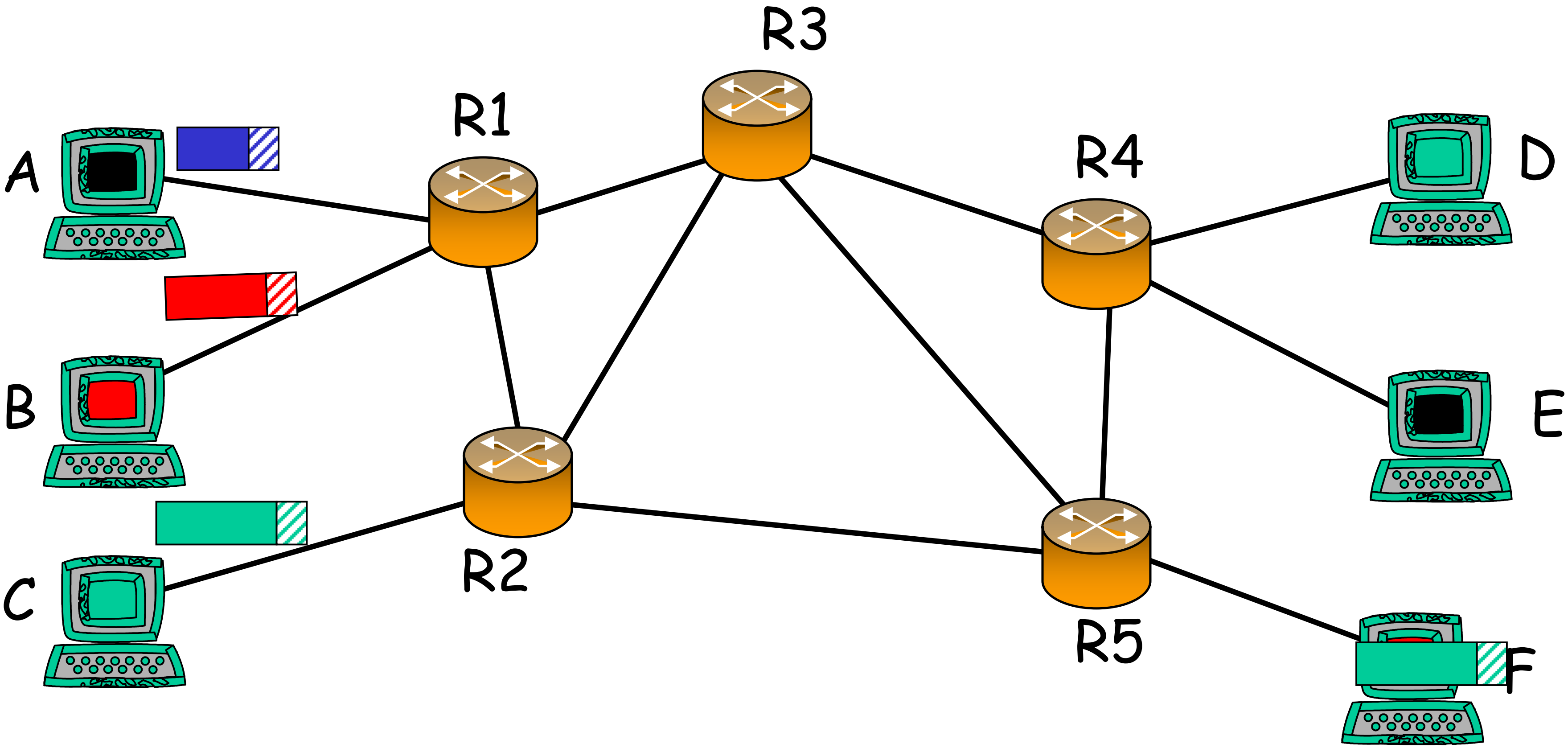
Ver	HLen	T.Service	Total Packet Length	
Fragment ID			Flags	Fragment Offset
TTL		Protocol	Header Checksum	
Source Address				
Destination Address				
Options (if any)				

Basic operation of an IP router



Destination	Next Hop
D	R3
E	R3
F	R5

Basic operation of an IP router



Forwarding tables

IP address } 32 bits wide → ~ 4 billion unique address

Naïve approach:

One entry per address

Entry	Destination	Port
1	0.0.0.0	1
2	0.0.0.1	2
⋮	⋮	⋮
2^{32}	255.255.255.255	12

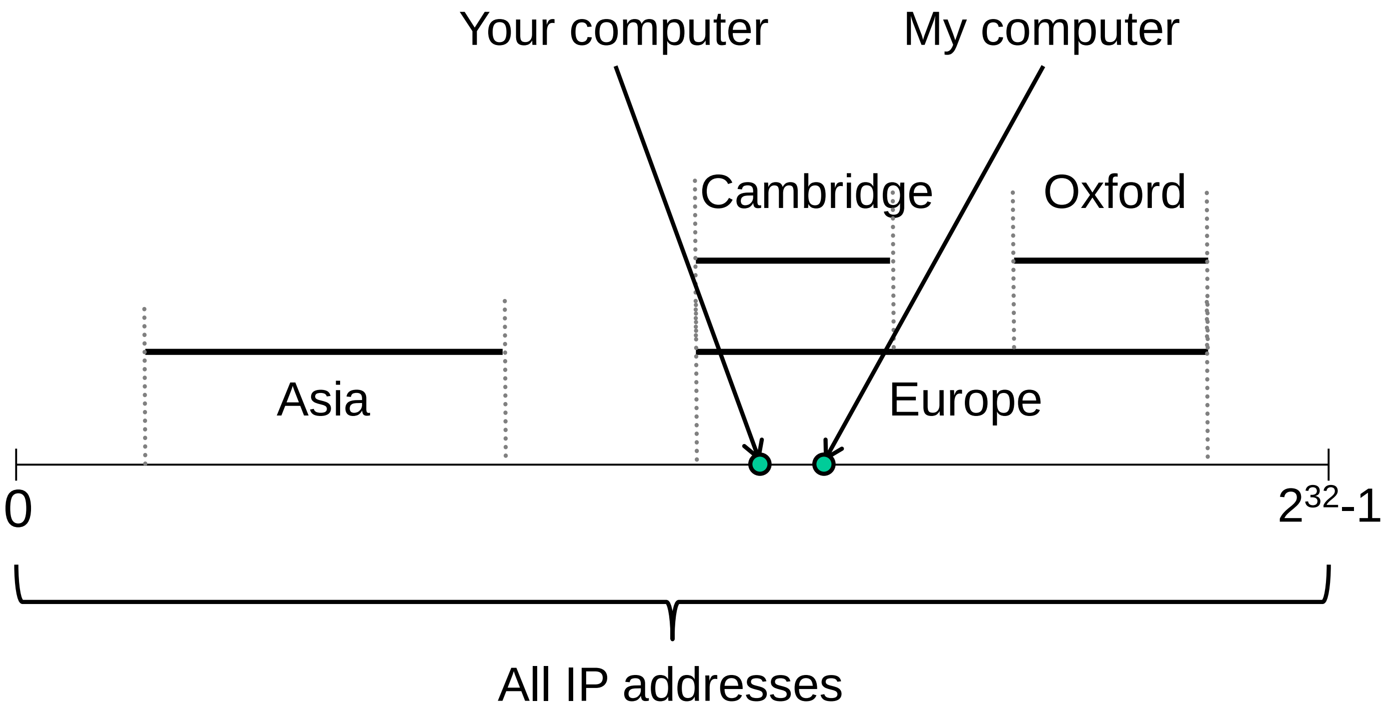
~ 4 billion entries

Improved approach:

Group entries to reduce table size

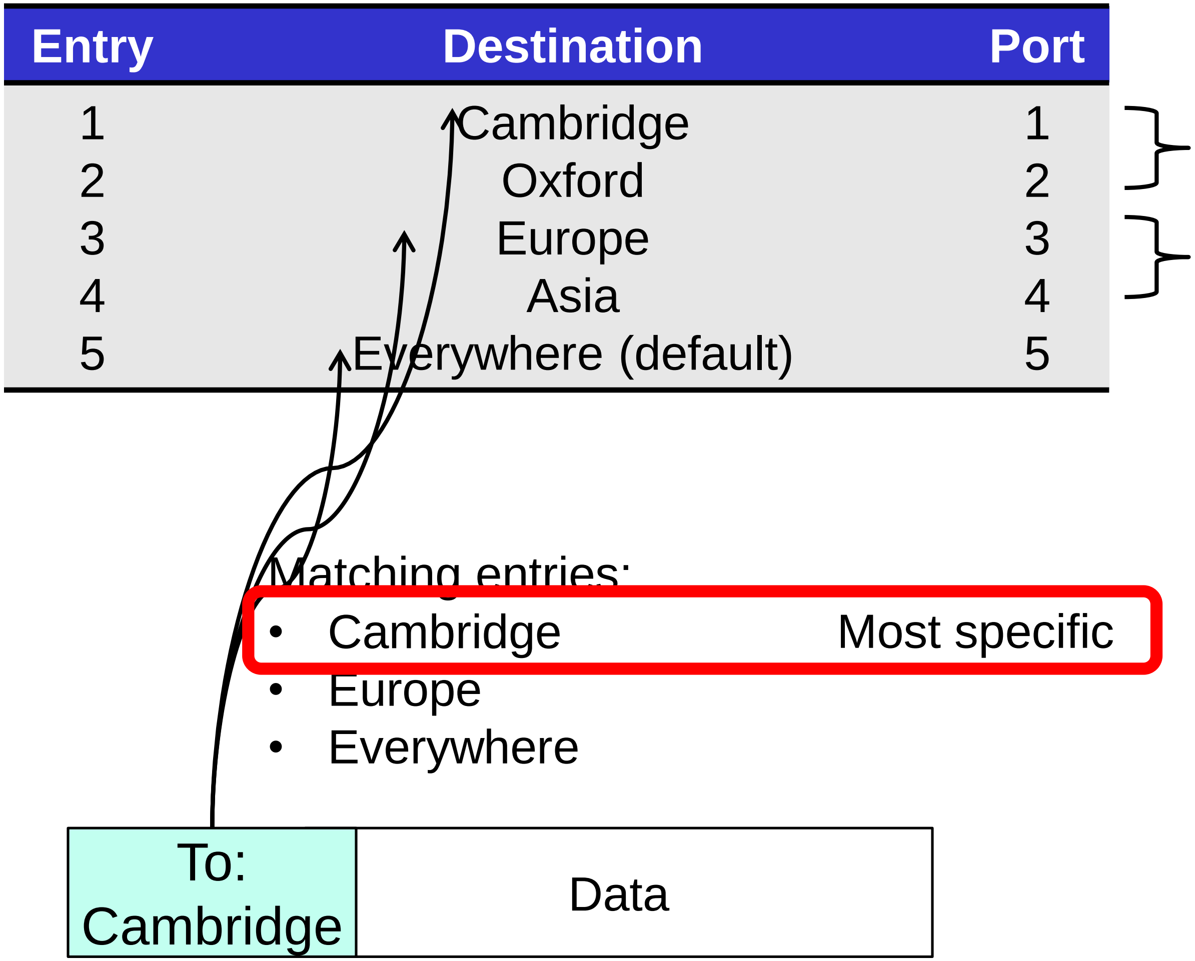
Entry	Destination	Port
1	0.0.0.0 – 127.255.255.255	1
2	128.0.0.1 – 128.255.255.255	2
⋮	⋮	⋮
50	248.0.0.0 – 255.255.255.255	12

IP addresses as a line



Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	Asia	4
5	Everywhere (default)	5

Longest Prefix Match (LPM)



Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	Asia	4
5	Everywhere (default)	5

Universities

Continents

Planet

Matching entries:

- Cambridge Most specific
- Europe
- Everywhere

Longest Prefix Match (LPM)

Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	Asia	4
5	Everywhere (default)	5

Universities

Continents

Planet

Matching entries:

- Europe Most specific
- Everywhere

To: Germany	Data

Implementing Longest Prefix Match

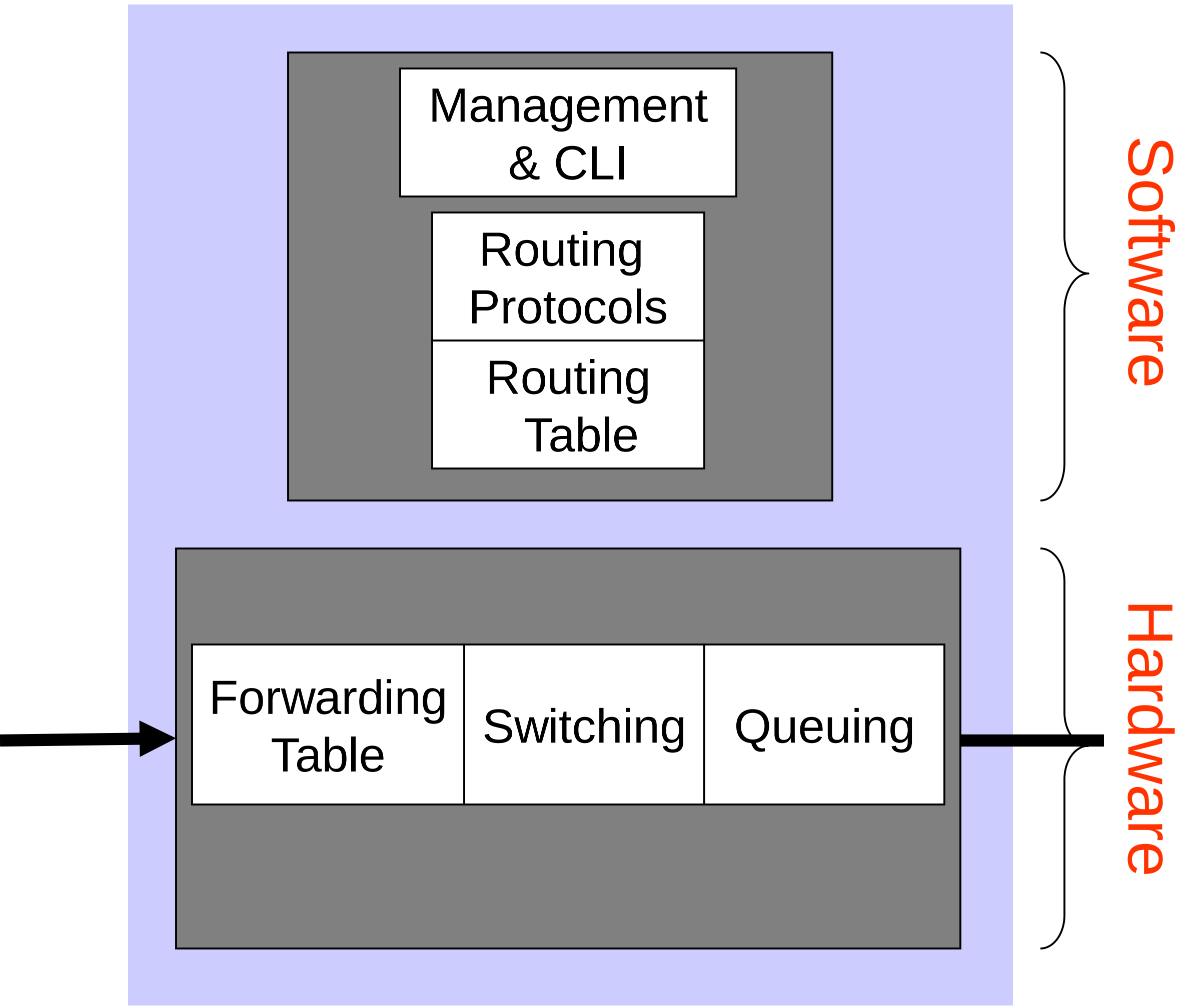
Entry	Destination	Port	
1	Cambridge	1	Searching
2	Oxford	2	
3	Europe	3	
4	Asia	4	FOUND
5	Everywhere (default)	5	

Most specific

↓

Least specific

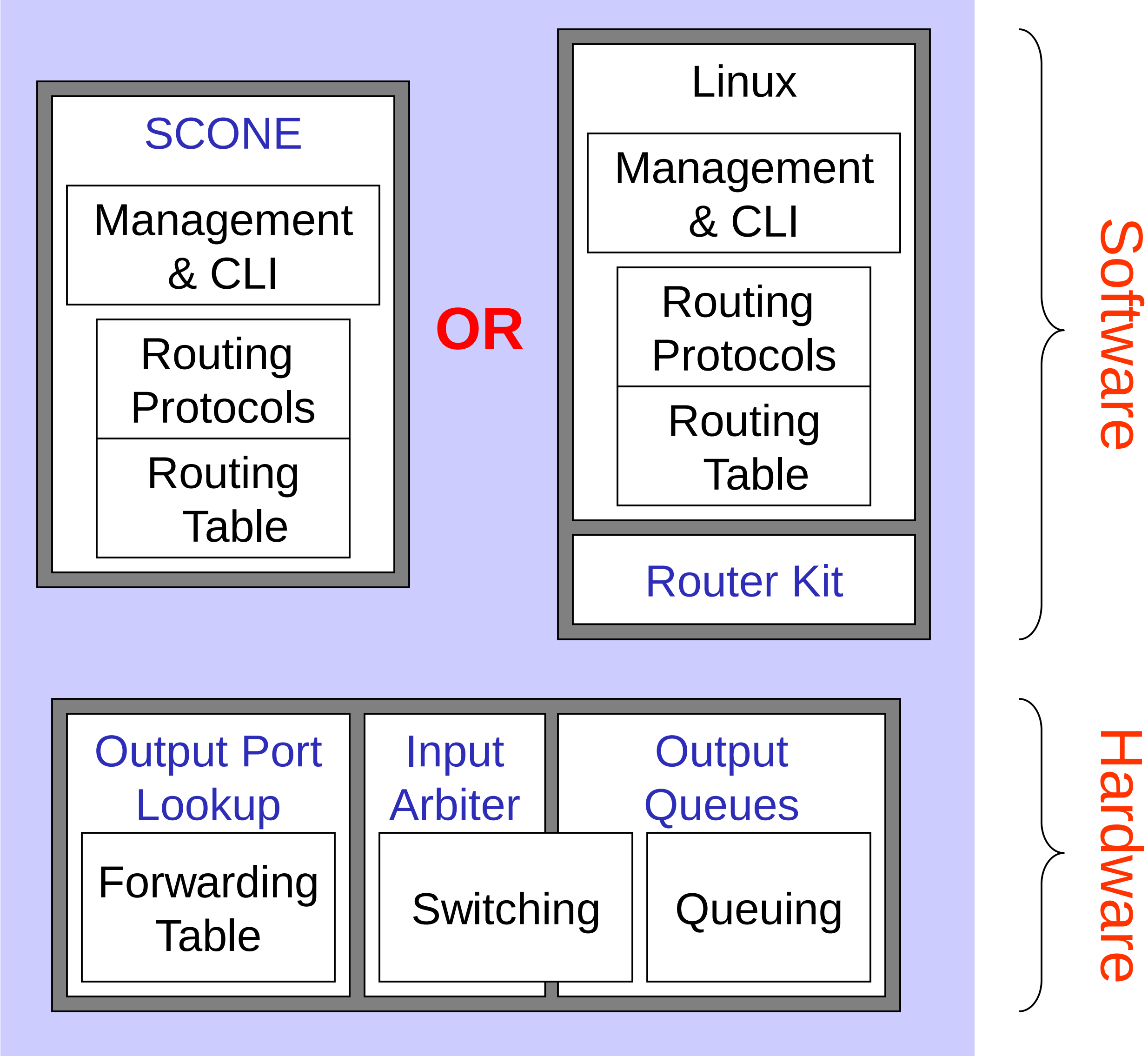
Basic components of an IP router



Control Plane

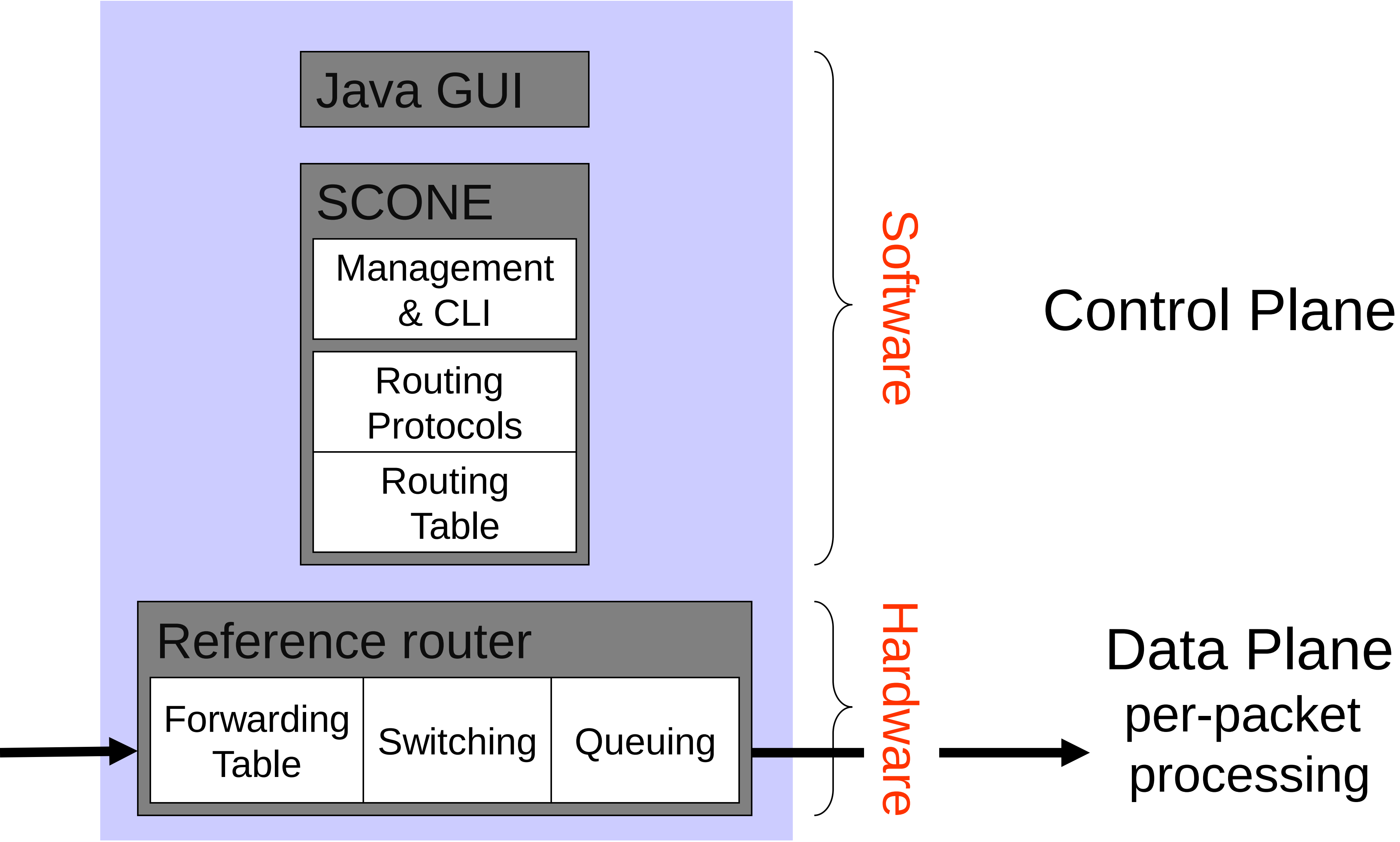
Data Plane
per-packet processing

IP router components in NetFPGA

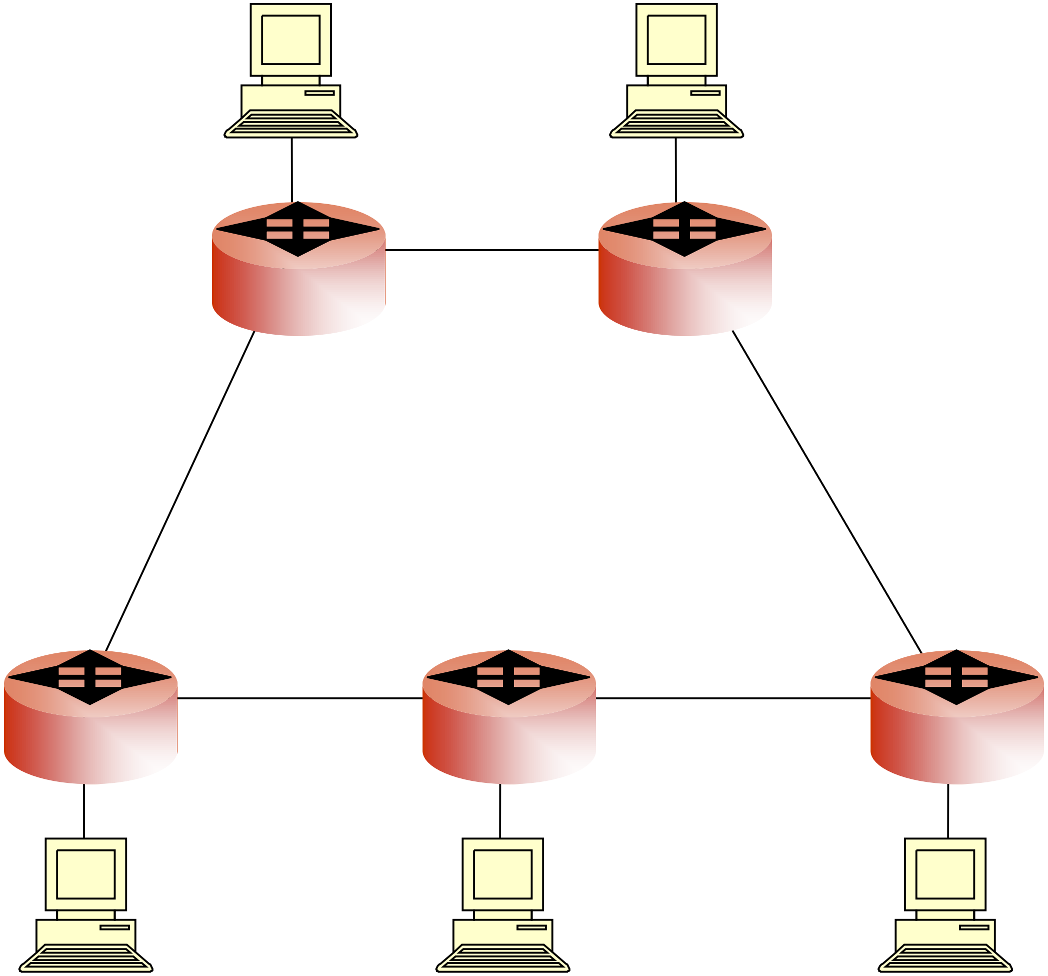


Section III: Example I

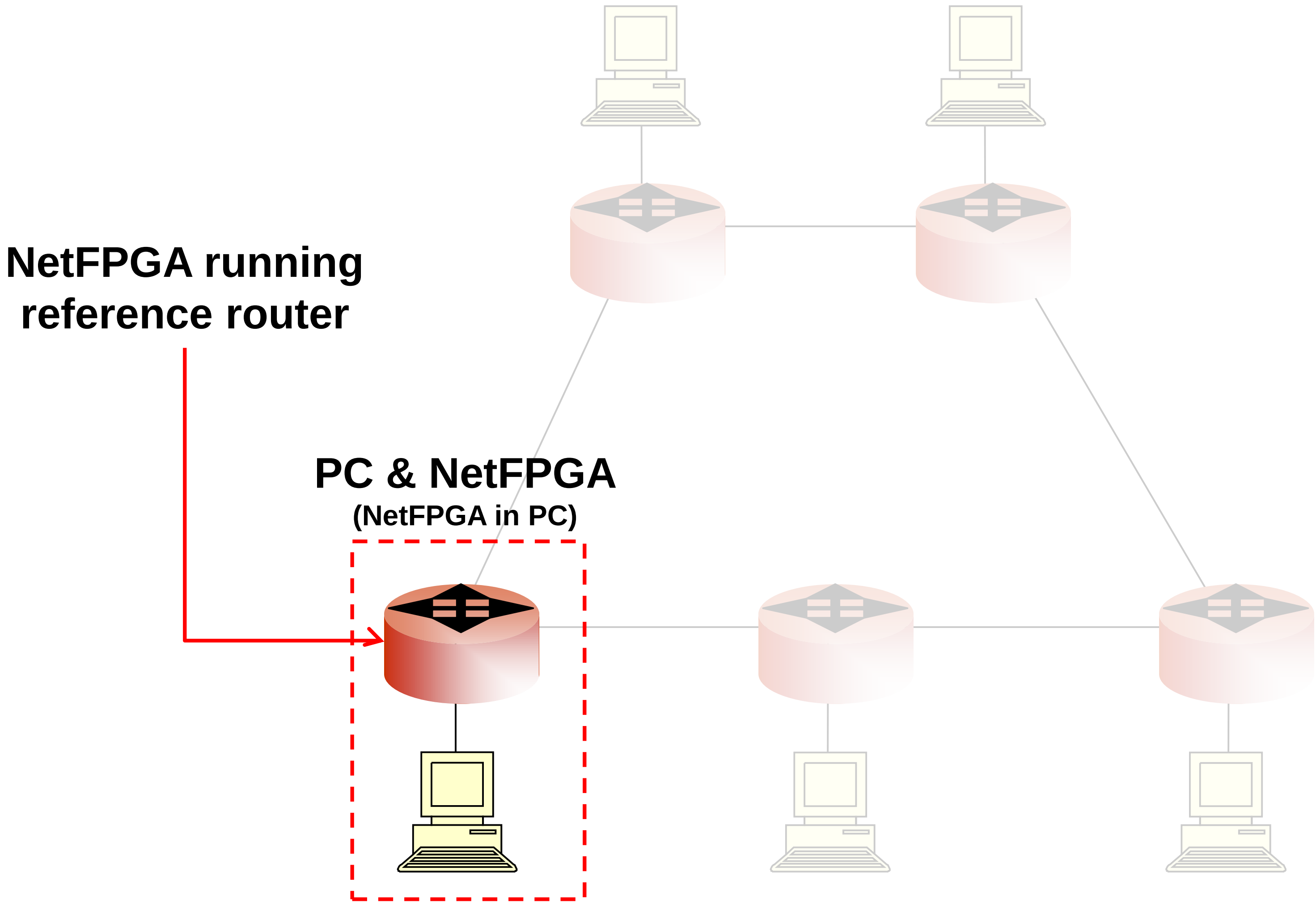
Operational IPv4 router



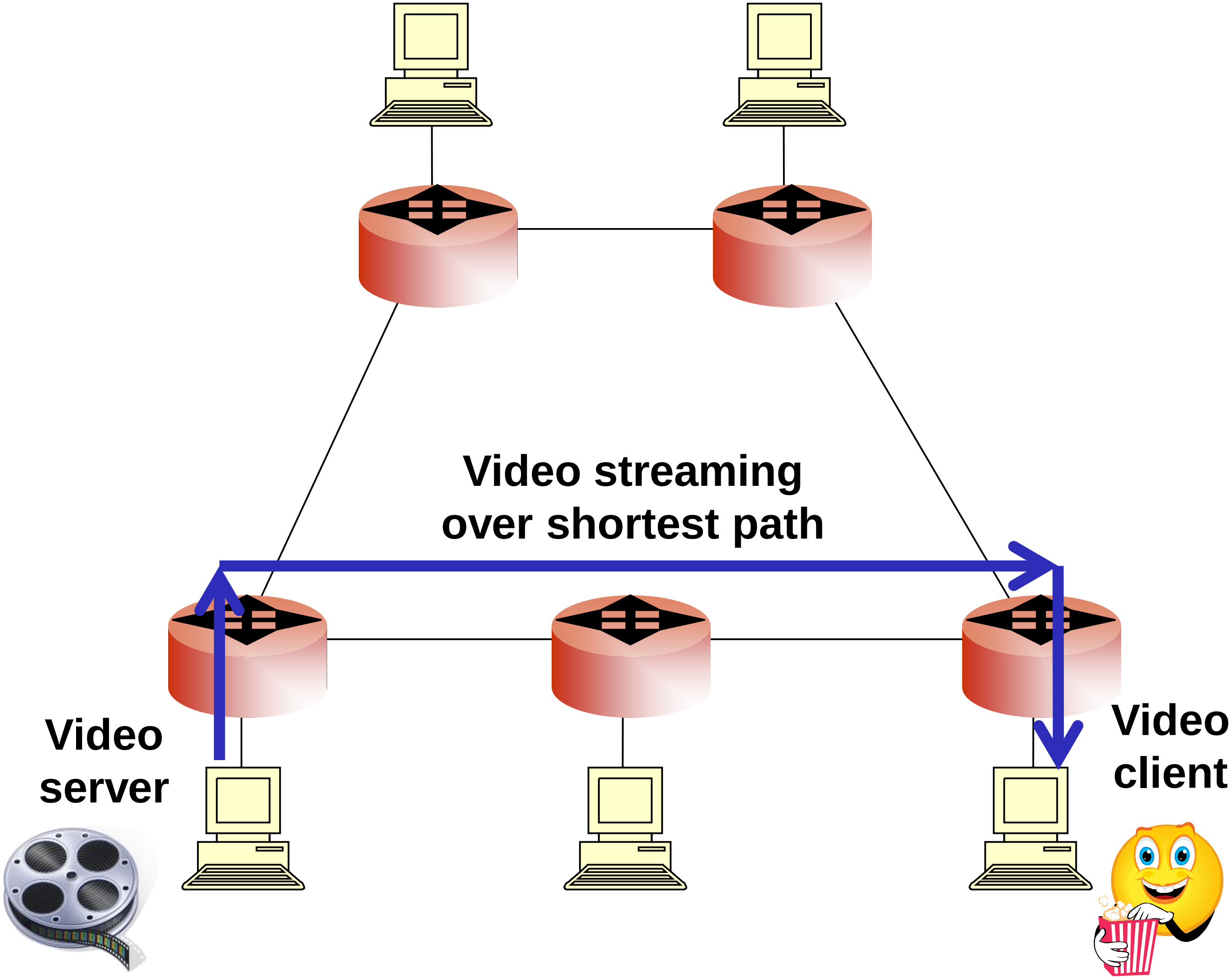
Streaming video



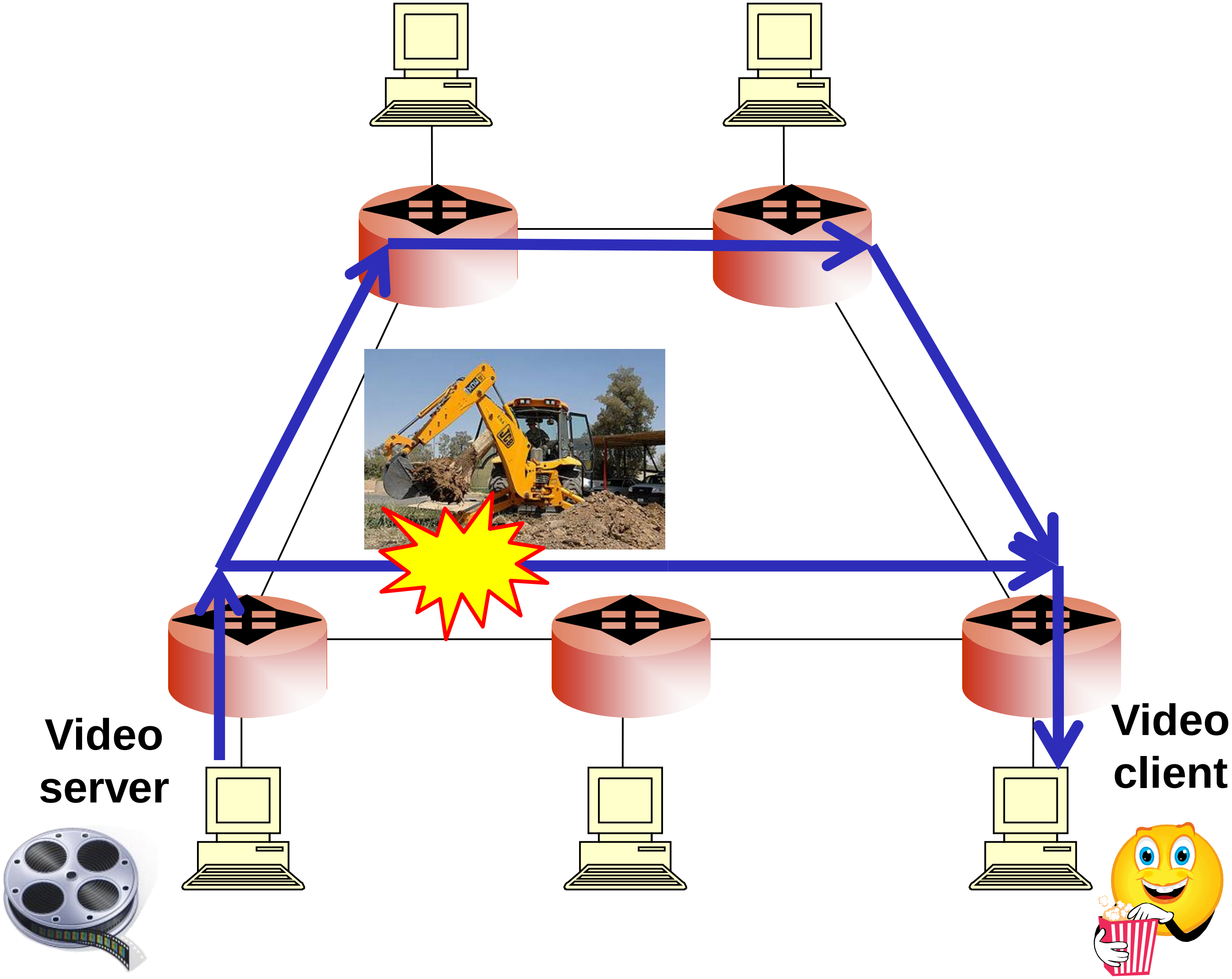
Streaming video



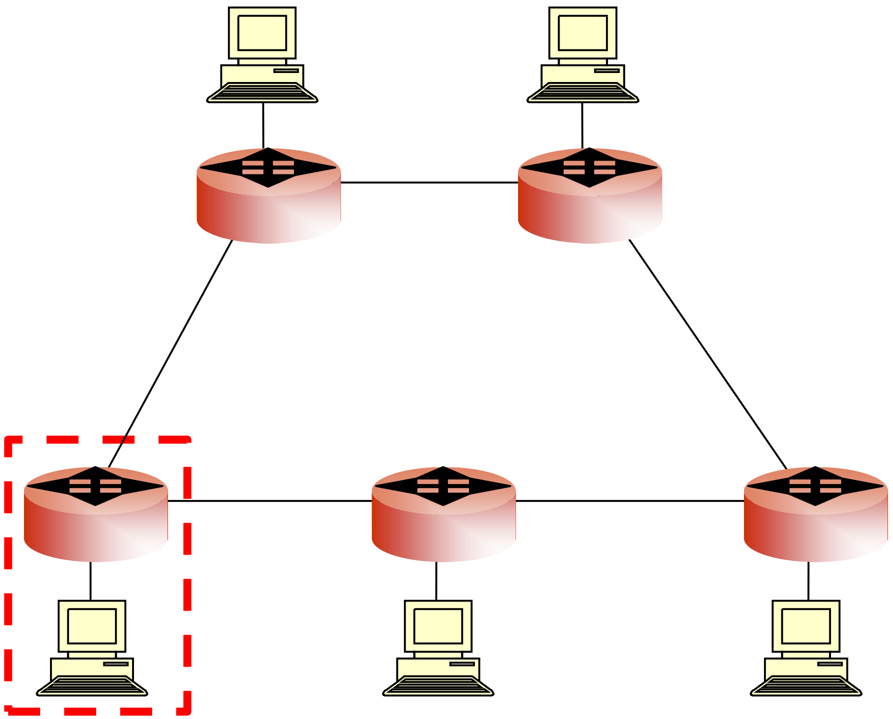
Streaming video

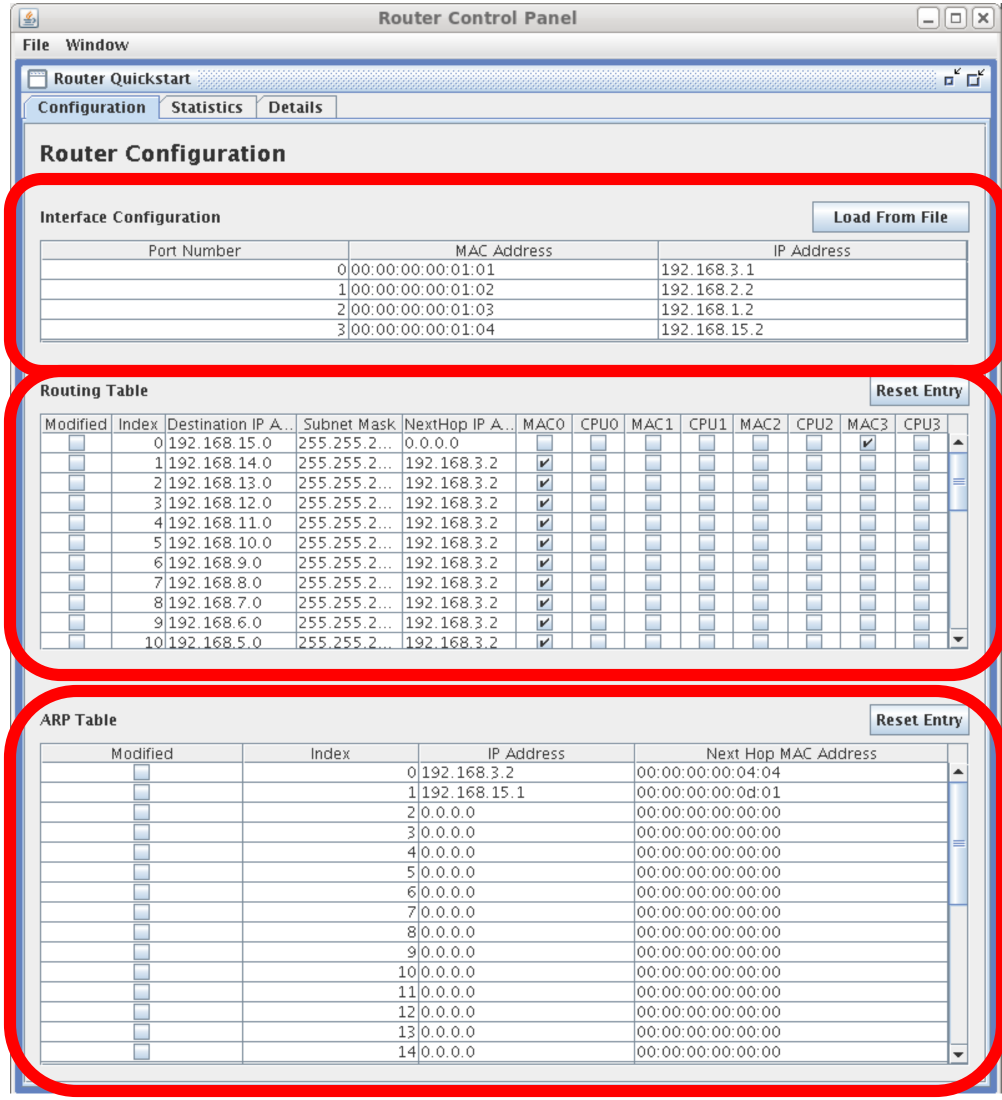


Streaming video



Observing the routing tables





Router Configuration

Interface Configuration

Port Number	MAC Address	IP Address
0	00:00:00:00:01:01	192.168.3.1
1	00:00:00:00:01:02	192.168.2.2
2	00:00:00:00:01:03	192.168.1.2
3	00:00:00:00:01:04	192.168.15.2

Routing Table

Modified	Index	Destination IP A...	Subnet Mask	NextHop IP A...	MAC0	CPU0	MAC1	CPU1	MAC2	CPU2	MAC3	CPU3
□	0	192.168.15.0	255.255.2...	0.0.0.0	□	□	□	□	□	□	☑	□
□	1	192.168.14.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	2	192.168.13.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	3	192.168.12.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	4	192.168.11.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	5	192.168.10.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	6	192.168.9.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	7	192.168.8.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	8	192.168.7.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	9	192.168.6.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□
□	10	192.168.5.0	255.255.2...	192.168.3.2	☑	□	□	□	□	□	□	□

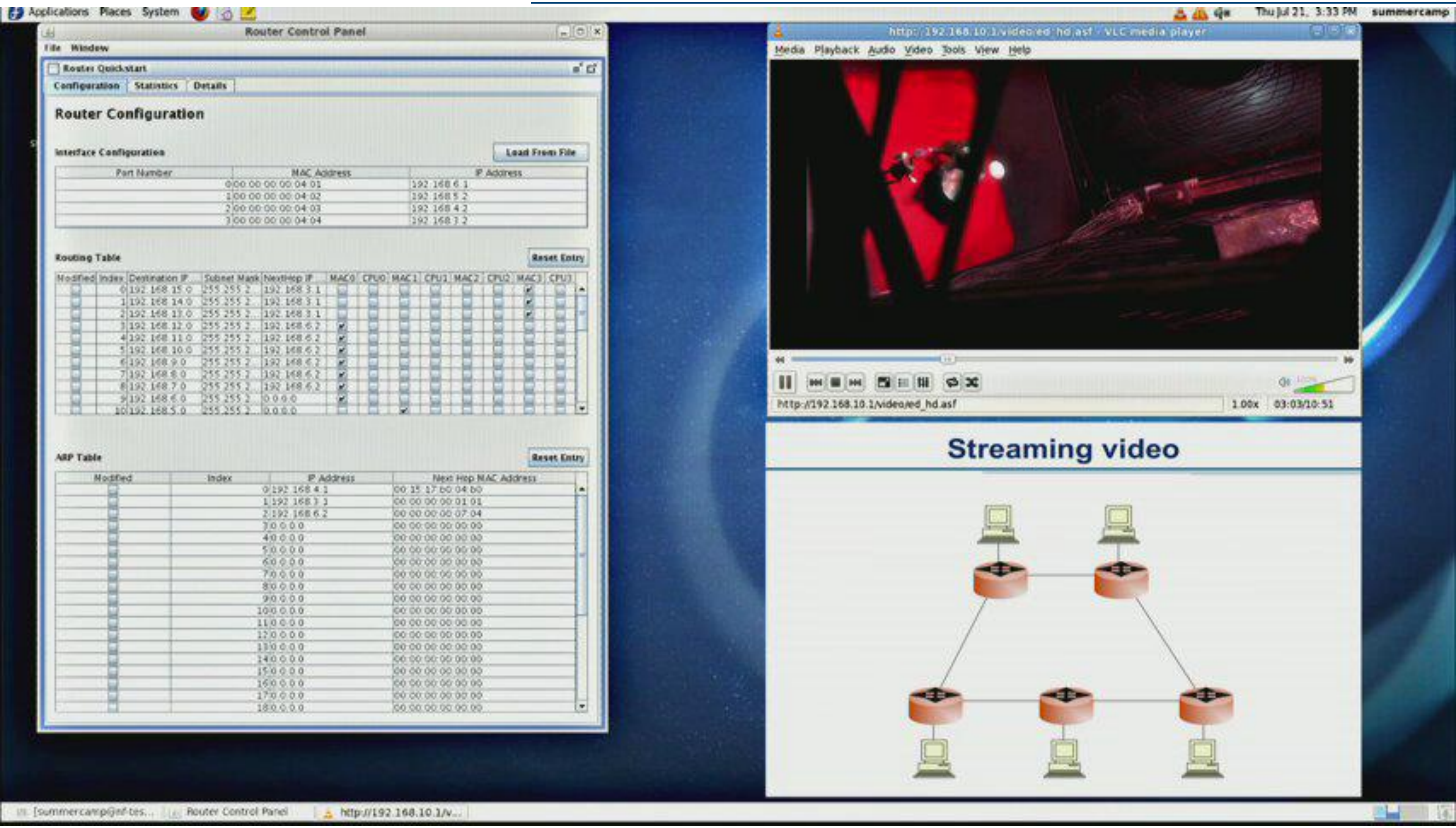
ARP Table

Modified	Index	IP Address	Next Hop MAC Address
□	0	192.168.3.2	00:00:00:00:04:04
□	1	192.168.15.1	00:00:00:00:0d:01
□	2	0.0.0.0	00:00:00:00:00:00
□	3	0.0.0.0	00:00:00:00:00:00
□	4	0.0.0.0	00:00:00:00:00:00
□	5	0.0.0.0	00:00:00:00:00:00
□	6	0.0.0.0	00:00:00:00:00:00
□	7	0.0.0.0	00:00:00:00:00:00
□	8	0.0.0.0	00:00:00:00:00:00
□	9	0.0.0.0	00:00:00:00:00:00
□	10	0.0.0.0	00:00:00:00:00:00
□	11	0.0.0.0	00:00:00:00:00:00
□	12	0.0.0.0	00:00:00:00:00:00
□	13	0.0.0.0	00:00:00:00:00:00
□	14	0.0.0.0	00:00:00:00:00:00

Columns:

- **Subnet address**
- **Subnet mask**
- **Next hop IP**
- **Output ports**

Example 1



Router Configuration

Interface Configuration

Port Number	MAC Address	IP Address
0	00:00:00:00:04:01	192.168.6.1
1	00:00:00:00:04:02	192.168.5.2
2	00:00:00:00:04:03	192.168.4.2
3	00:00:00:00:04:04	192.168.3.2

Routing Table

Modified	Index	Destination IP	Subnet Mask	NextHop IP	MAC0	CPU0	MAC1	CPU1	MAC2	CPU2	MAC3	CPU3
☐	0	192.168.15.0	255.255.2...	192.168.3.1	☐	☐	☐	☐	☐	☐	☑	☐
☐	1	192.168.14.0	255.255.2...	192.168.3.1	☐	☐	☐	☐	☐	☐	☑	☐
☐	2	192.168.13.0	255.255.2...	192.168.3.1	☐	☐	☐	☐	☐	☐	☑	☐
☐	3	192.168.12.0	255.255.2...	192.168.6.2	☑	☐	☐	☐	☐	☐	☐	☐
☐	4	192.168.11.0	255.255.2...	192.168.6.2	☑	☐	☐	☐	☐	☐	☐	☐
☐	5	192.168.10.0	255.255.2...	192.168.6.2	☑	☐	☐	☐	☐	☐	☐	☐
☐	6	192.168.9.0	255.255.2...	192.168.6.2	☑	☐	☐	☐	☐	☐	☐	☐
☐	7	192.168.8.0	255.255.2...	192.168.6.2	☑	☐	☐	☐	☐	☐	☐	☐
☐	8	192.168.7.0	255.255.2...	192.168.6.2	☑	☐	☐	☐	☐	☐	☐	☐
☐	9	192.168.6.0	255.255.2...	0.0.0.0	☑	☐	☐	☐	☐	☐	☐	☐
☐	10	192.168.5.0	255.255.2...	0.0.0.0	☐	☐	☑	☐	☐	☐	☐	☐

ARP Table

Modified	Index	IP Address	Next Hop MAC Address
☐	0	192.168.4.1	00:15:17:b0:04:b0
☐	1	192.168.3.1	00:00:00:00:01:01
☐	2	192.168.6.2	00:00:00:00:07:04
☐	3	0.0.0.0	00:00:00:00:00:00
☐	4	0.0.0.0	00:00:00:00:00:00
☐	5	0.0.0.0	00:00:00:00:00:00
☐	6	0.0.0.0	00:00:00:00:00:00
☐	7	0.0.0.0	00:00:00:00:00:00
☐	8	0.0.0.0	00:00:00:00:00:00
☐	9	0.0.0.0	00:00:00:00:00:00
☐	10	0.0.0.0	00:00:00:00:00:00
☐	11	0.0.0.0	00:00:00:00:00:00
☐	12	0.0.0.0	00:00:00:00:00:00
☐	13	0.0.0.0	00:00:00:00:00:00
☐	14	0.0.0.0	00:00:00:00:00:00
☐	15	0.0.0.0	00:00:00:00:00:00
☐	16	0.0.0.0	00:00:00:00:00:00
☐	17	0.0.0.0	00:00:00:00:00:00
☐	18	0.0.0.0	00:00:00:00:00:00

Streaming video

Review

NetFPGA as IPv4 router:

- •Reference hardware + SCONE software
- •Routing protocol discovers topology

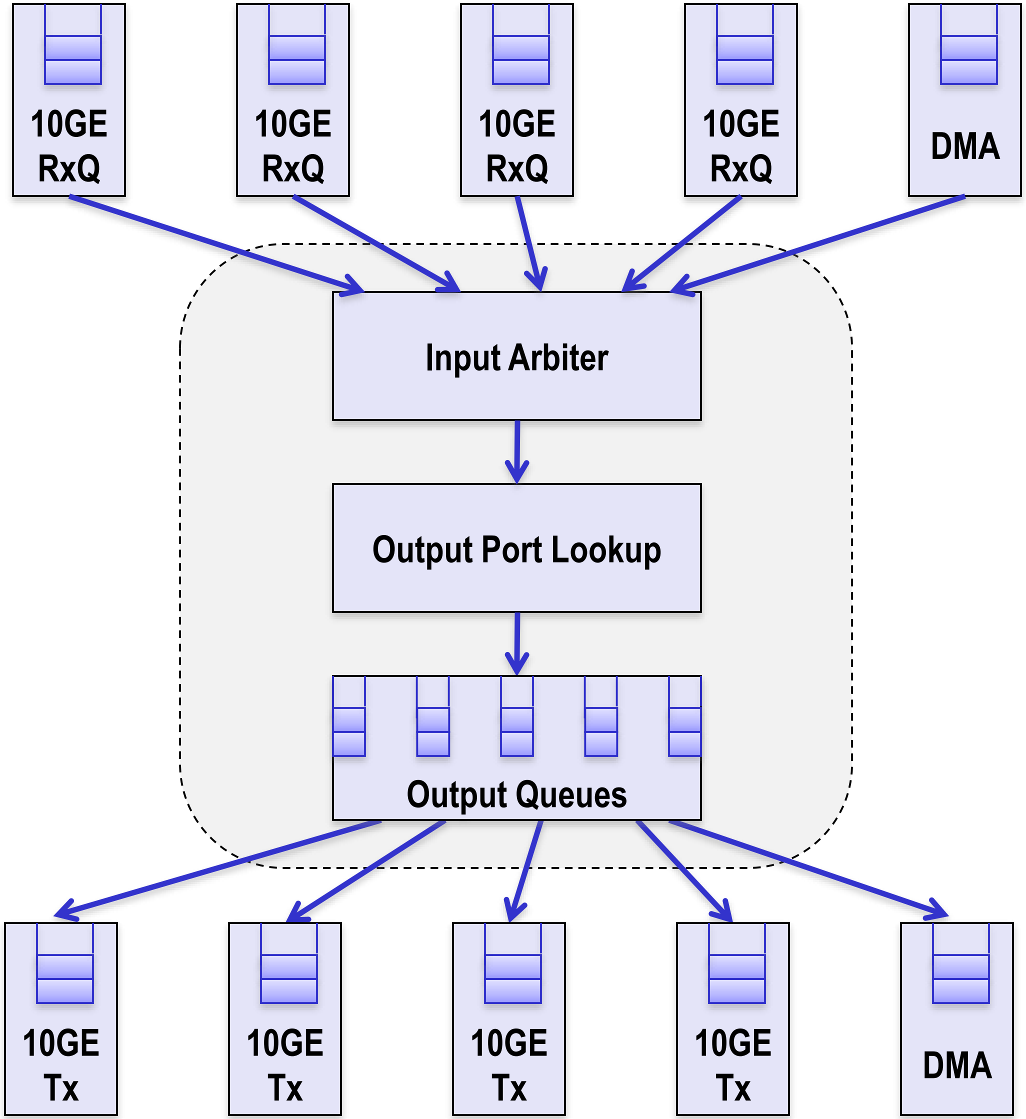
Demo:

- •Ring topology
- •Traffic flows over shortest path
- •Broken link: automatically route around failure

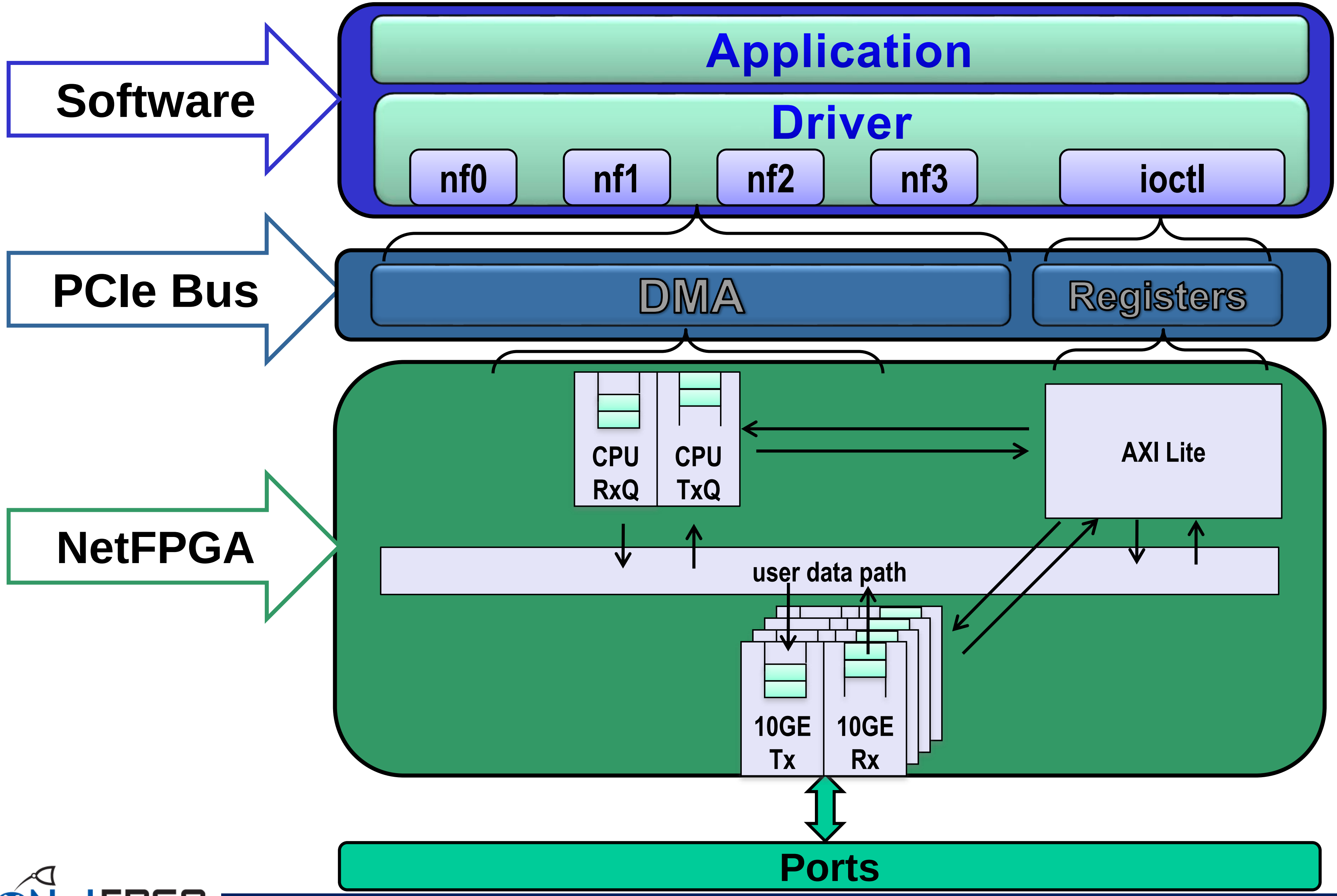
Section III: Life of a Packet

Reference Switch Pipeline

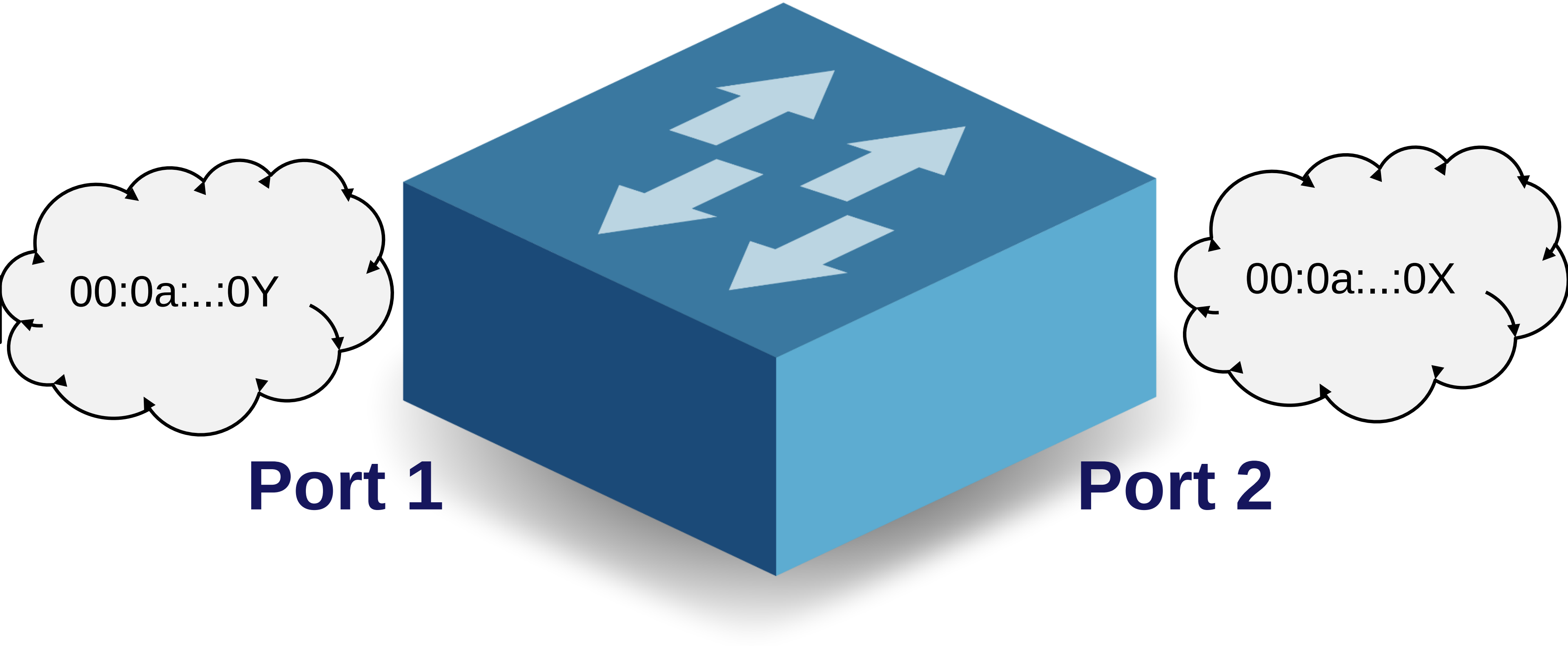
- **Five stages**
  - Input port
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output port
- **Packet-based module interface**
- **Pluggable design**



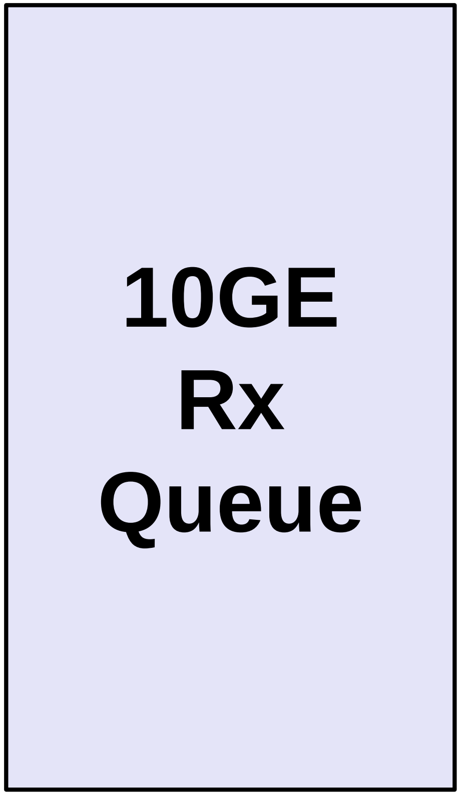
Full System Components



Life of a Packet through the Hardware

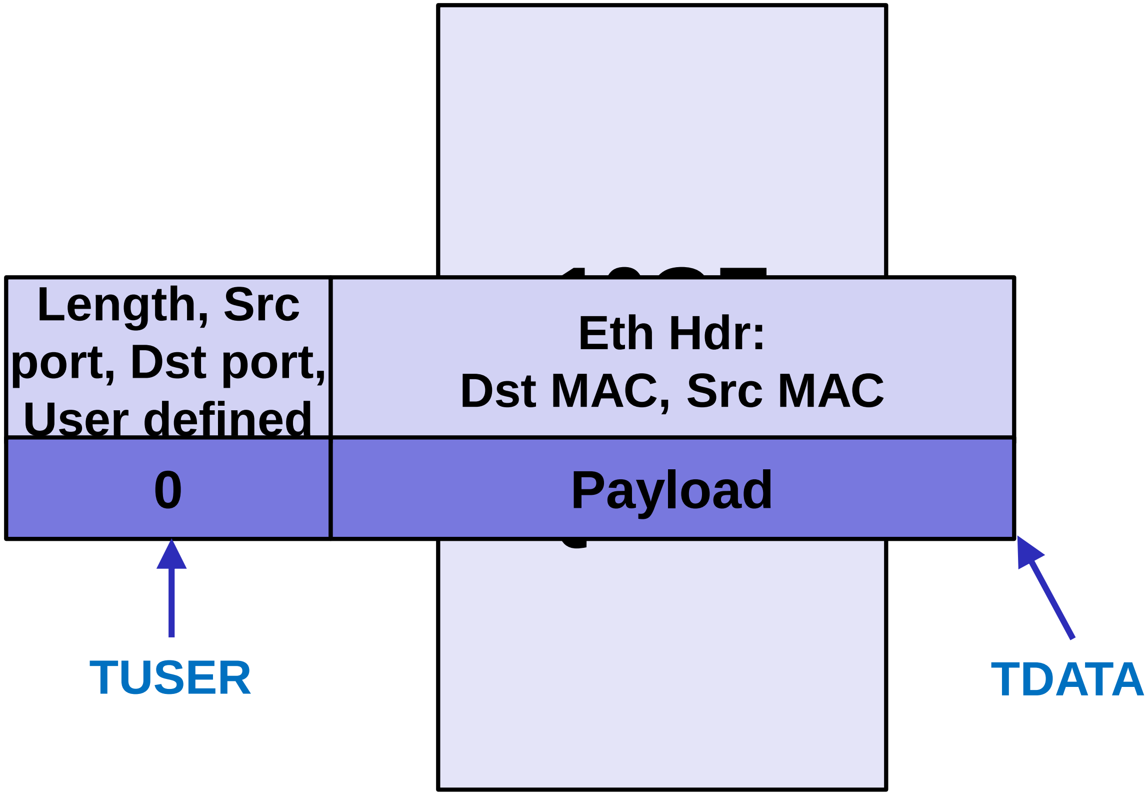


10GE Rx Queue

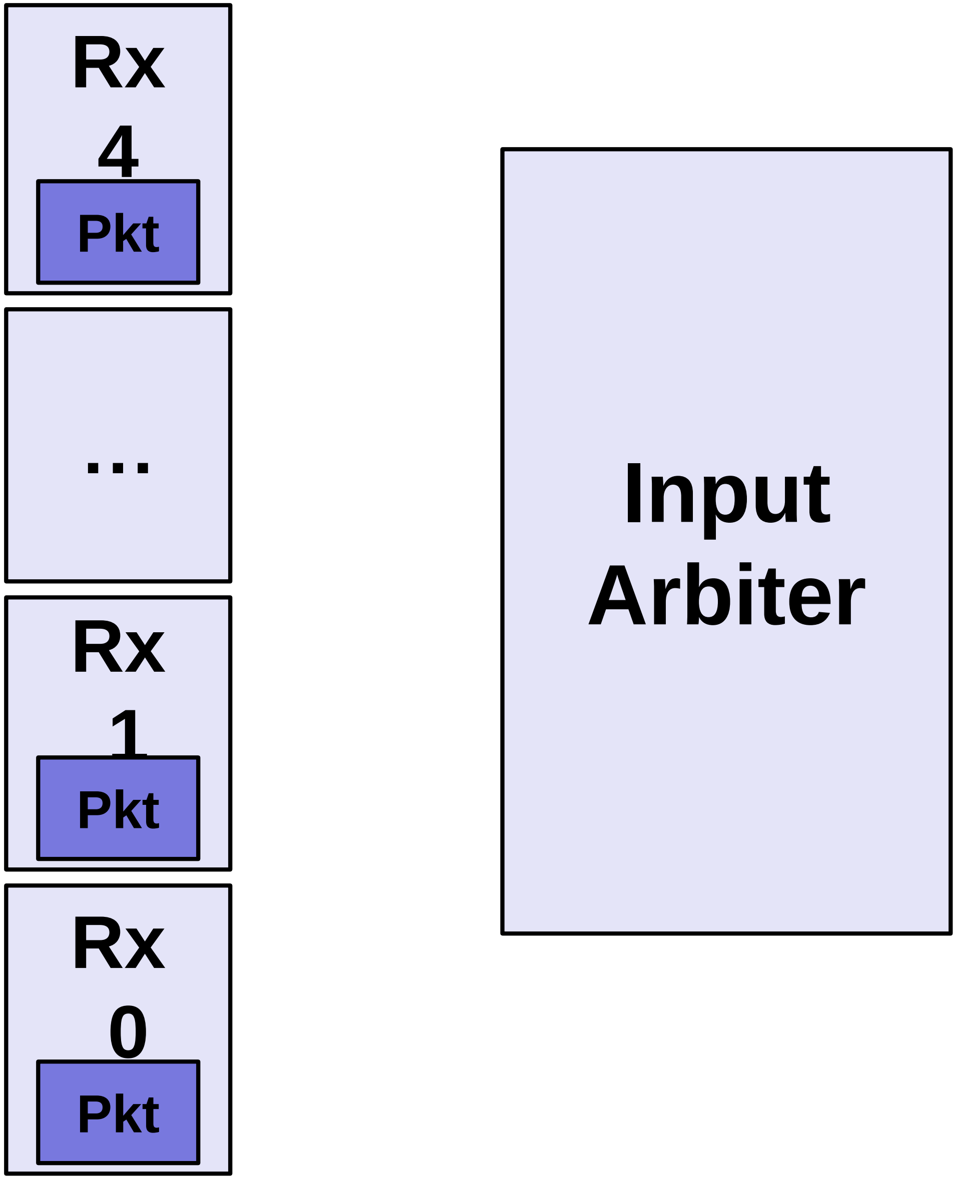


10GE Rx Queue

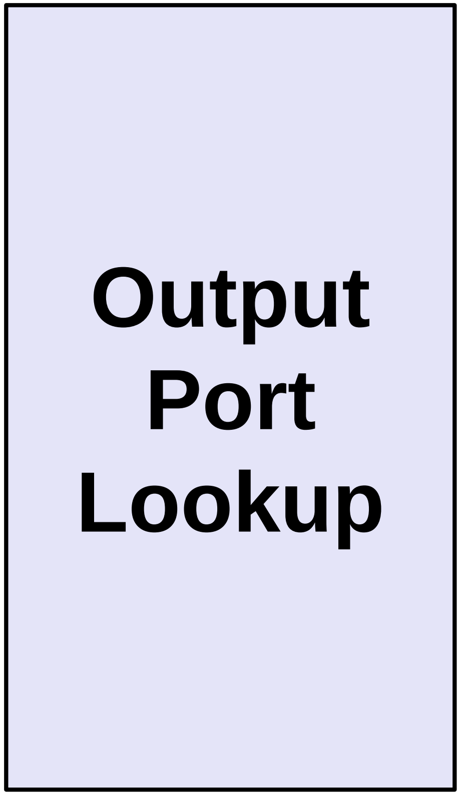
10GE Rx Queue



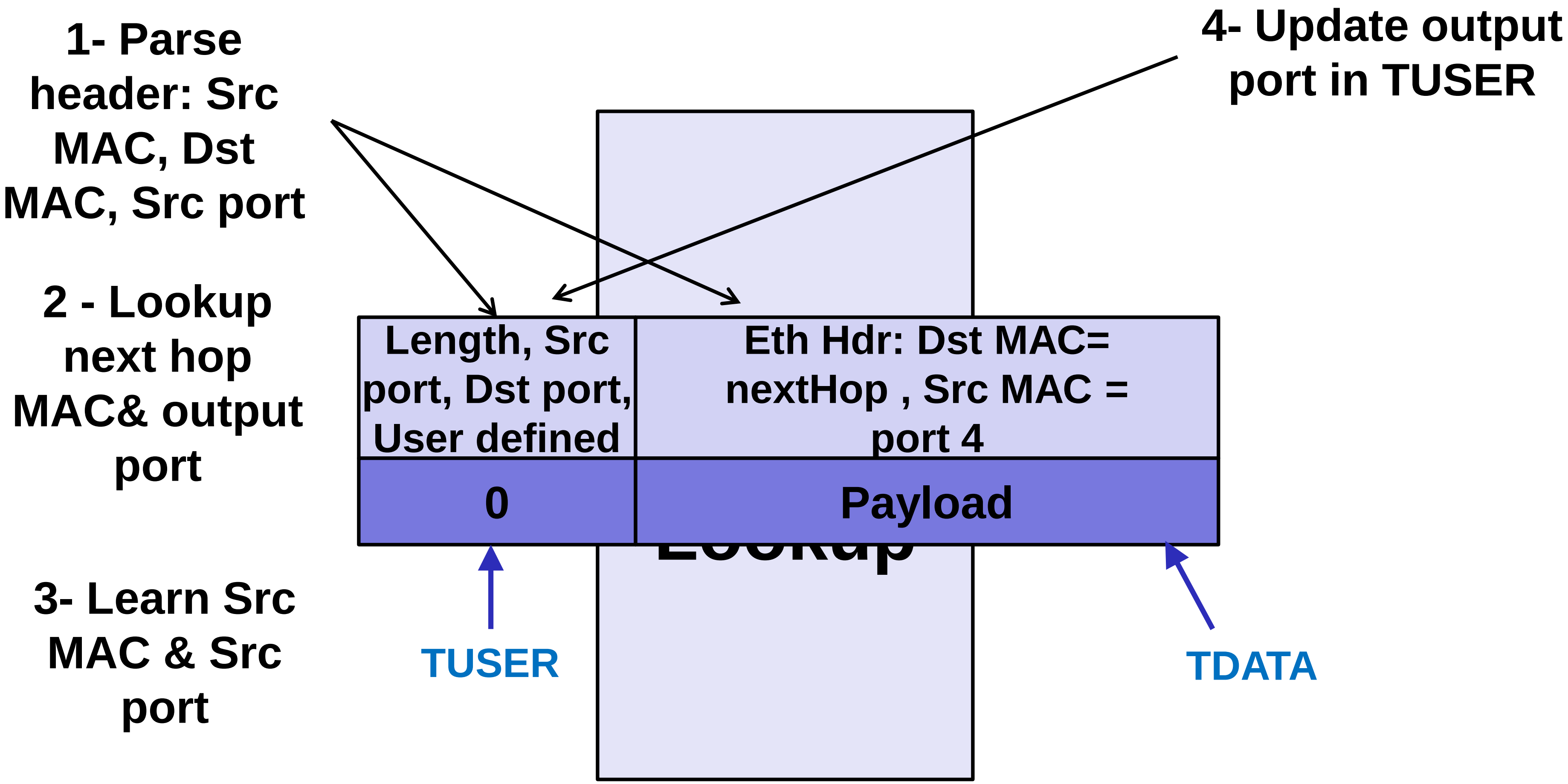
Input Arbiter



Output Port Lookup


Output Port Lookup

Output Port Lookup



1- Parse header: Src MAC, Dst MAC, Src port

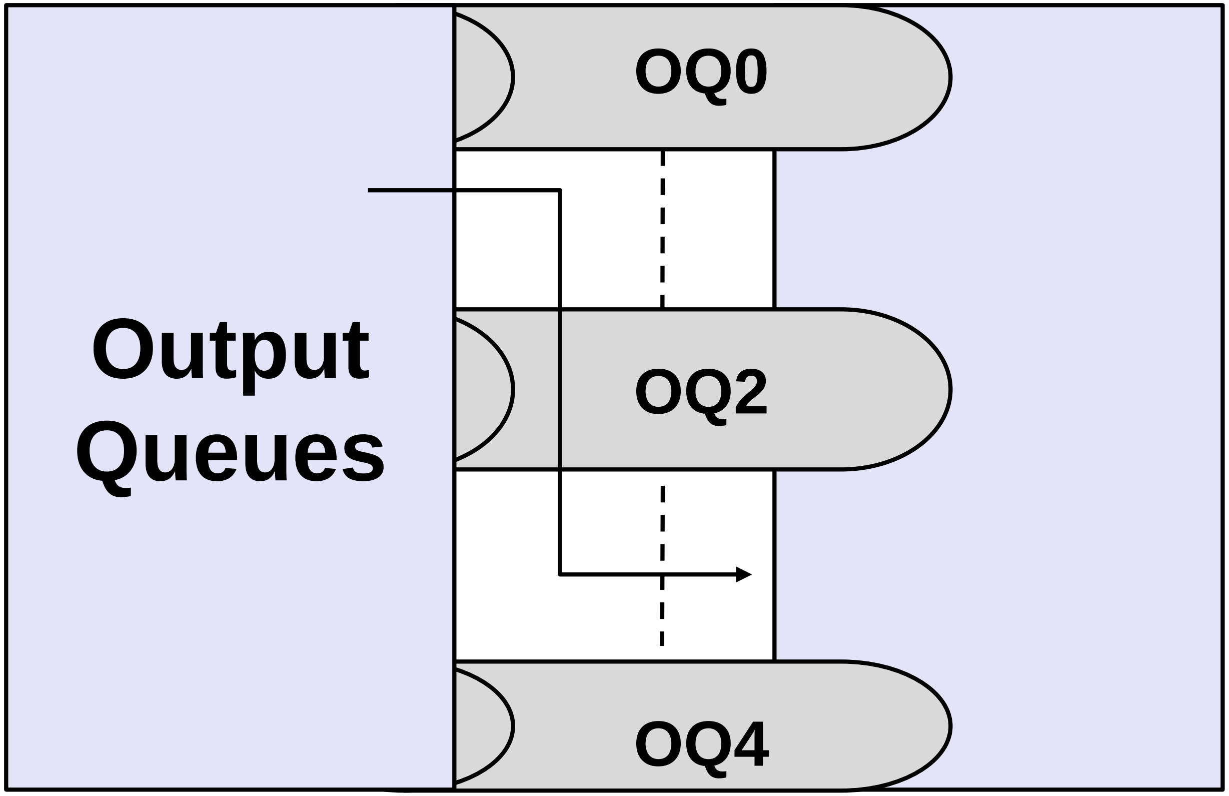
2 - Lookup next hop MAC& output port

3- Learn Src MAC & Src port

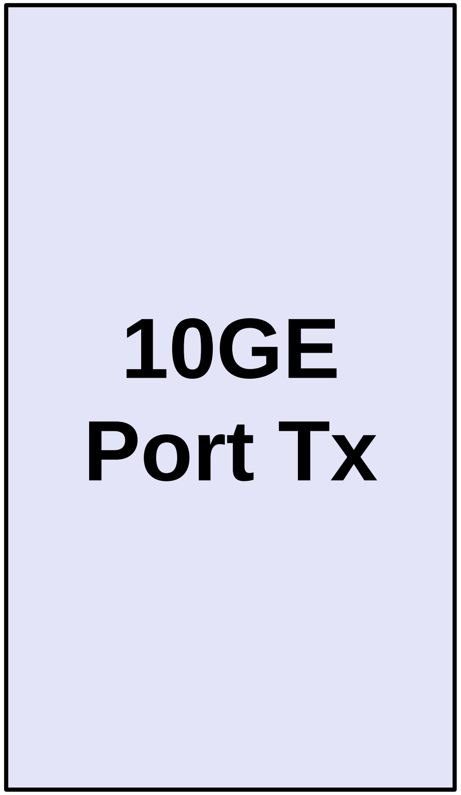
4- Update output port in TUSER

Length, Src port, Dst port, User defined	Eth Hdr: Dst MAC= nextHop , Src MAC = port 4
0	Payload

Output Queues

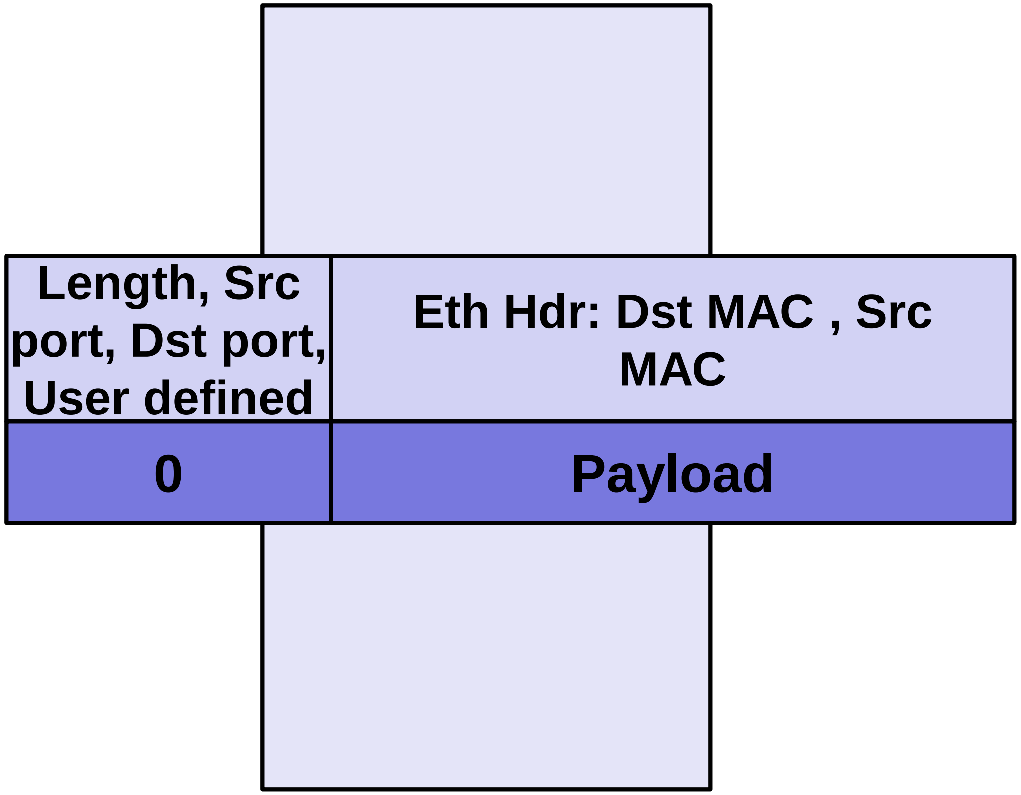


10GE Port Tx



10GE
Port Tx

MAC Tx Queue



Length, Src port, Dst port, User defined	Eth Hdr: Dst MAC , Src MAC
0	Payload

NetFPGA-Host Interaction

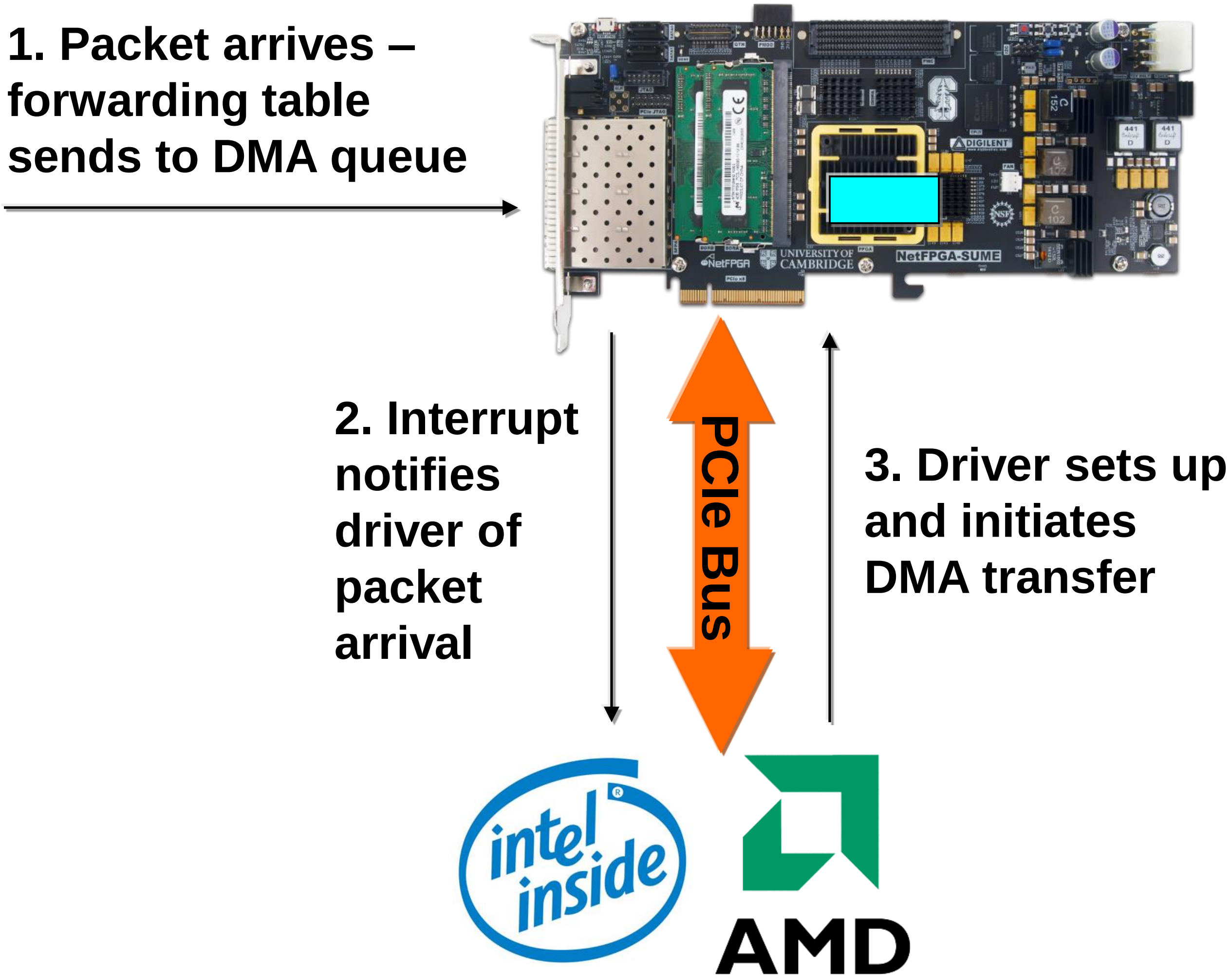
- **Linux driver interfaces with hardware**
  - Packet interface via standard Linux network stack

  - Register reads/writes via ioctl system call with wrapper functions:
    - rwaxi(int address, unsigned *data);

      eg:
      rwaxi(**0x7d4000000**, &val);

NetFPGA-Host Interaction

NetFPGA to host packet transfer



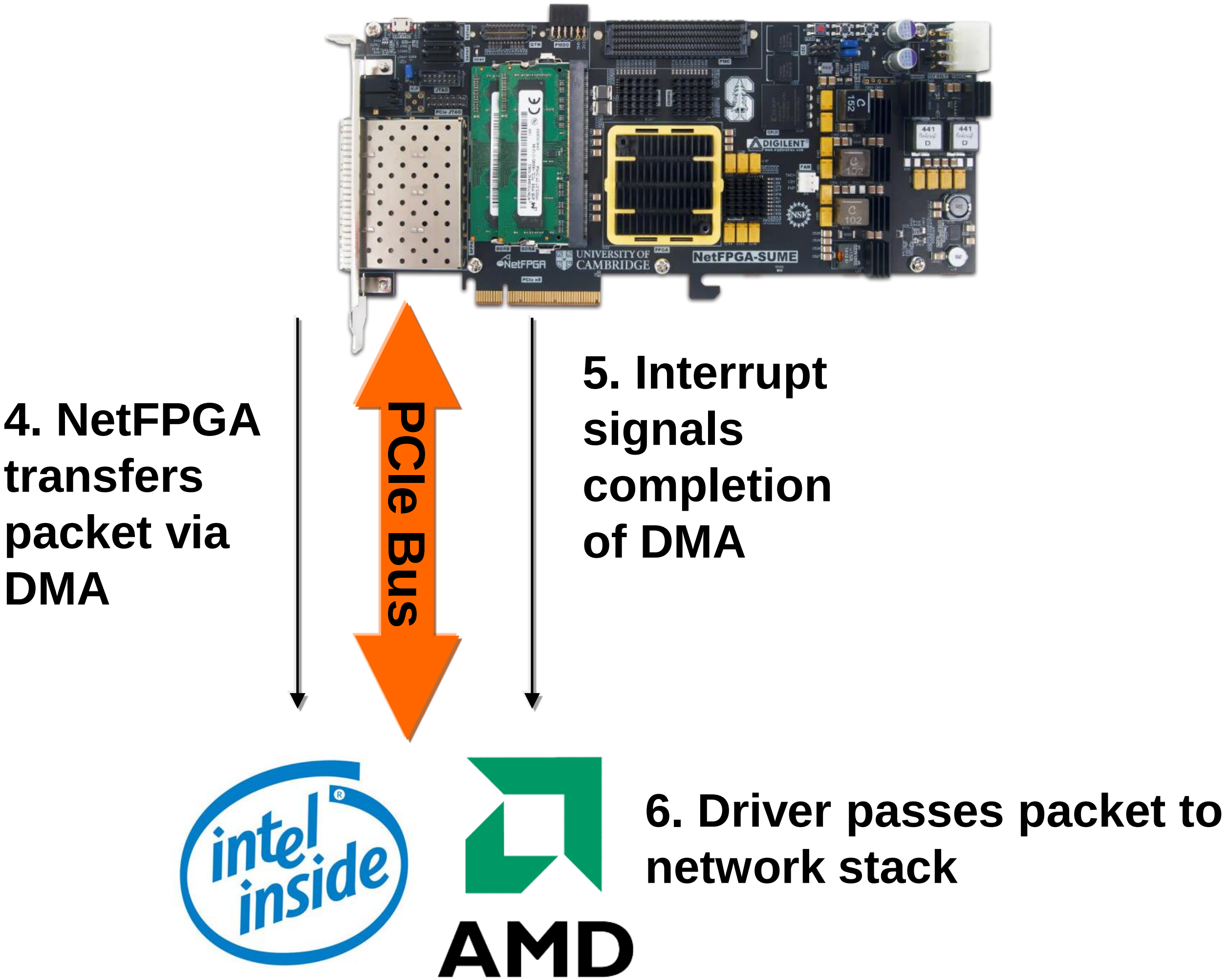
1. **Packet arrives – forwarding table sends to DMA queue**

2. **Interrupt notifies driver of packet arrival**

3. **Driver sets up and initiates DMA transfer**

NetFPGA-Host Interaction

NetFPGA to host packet transfer (cont.)



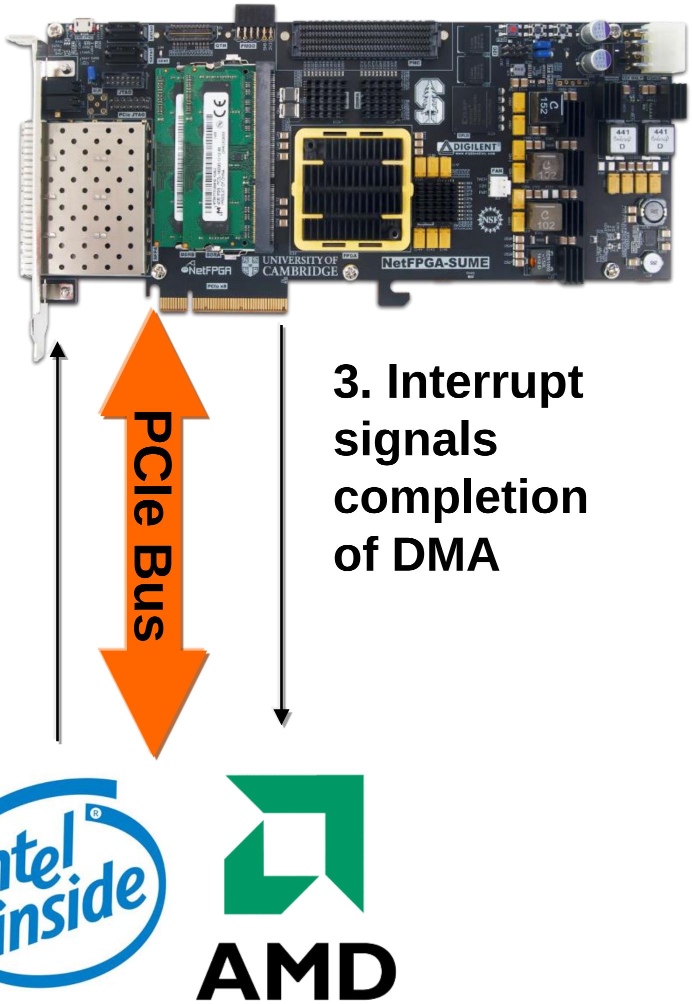
4. NetFPGA transfers packet via DMA

5. Interrupt signals completion of DMA

6. Driver passes packet to network stack

NetFPGA-Host Interaction

Host to NetFPGA packet transfers



2. Driver sets up and initiates DMA transfer

3. Interrupt signals completion of DMA

1. Software sends packet via network sockets

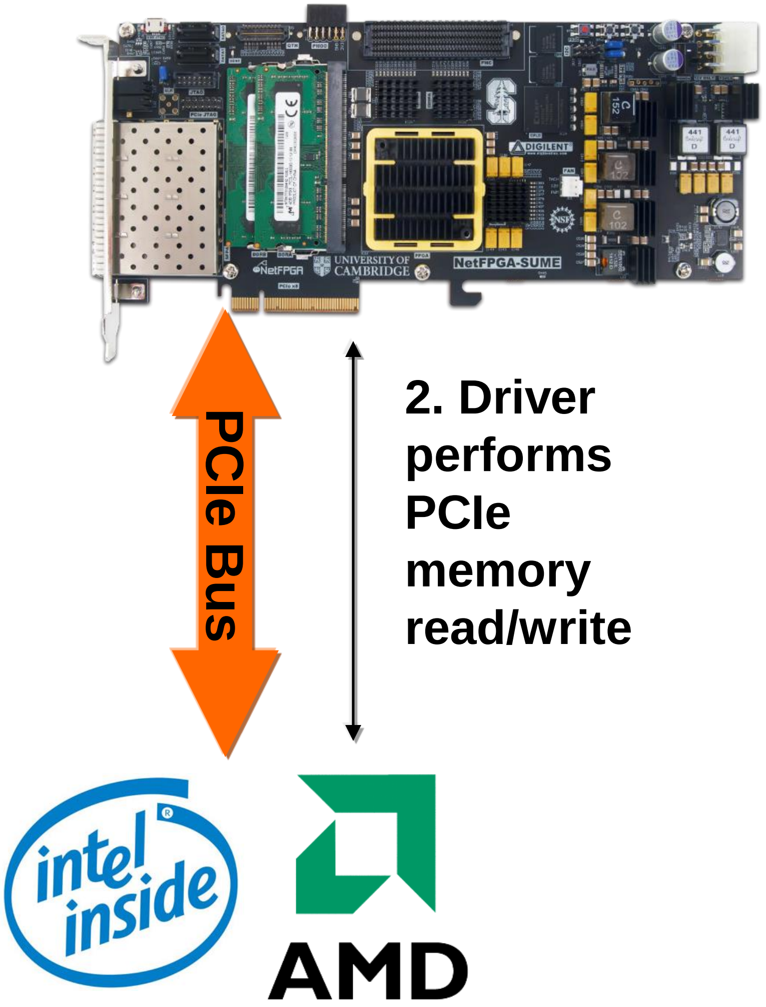
Packet delivered to driver

NetFPGA-Host Interaction

Register access

1. Software makes ioctl call on network socket

   ioctl passed to driver



PCIe Bus

2. Driver performs PCIe memory read/write

Section V: Infrastructure

Infrastructure

- **Tree structure**

- **NetFPGA package contents**
  - Reusable Verilog modules
  - Verification infrastructure
  - Build infrastructure
  - Utilities
  - Software libraries

NetFPGA package contents

- **Projects:**
  - HW: router, switch, NIC
  - SW: router kit, SCONE
- **Reusable Verilog modules**
- **Verification infrastructure:**
  - simulate designs (from AXI interface)
  - run tests against hardware
  - test data generation libraries (eg. packets)
- **Build infrastructure**
- **Utilities:**
  - register I/O
- **Software libraries**

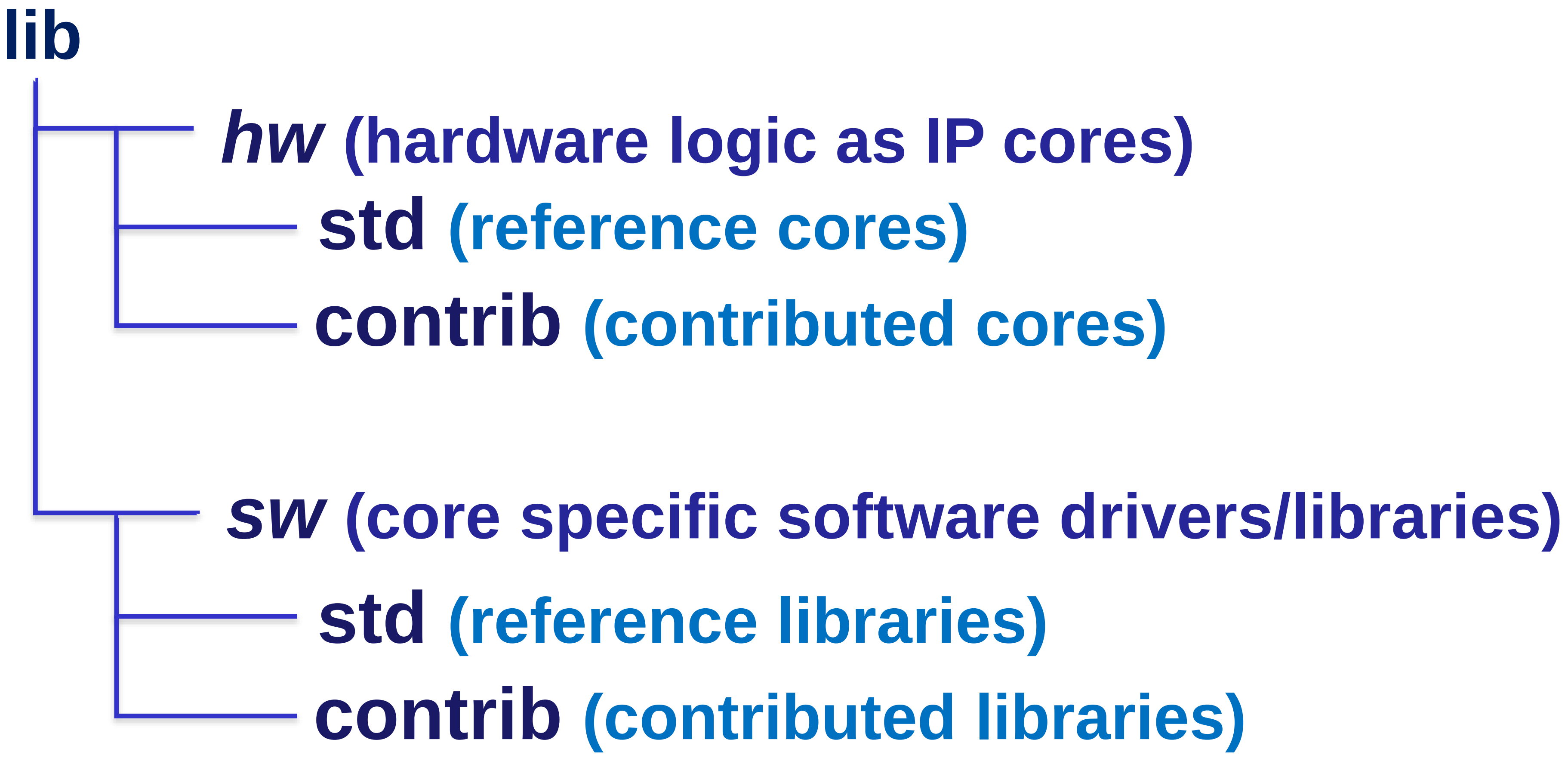
Tree Structure (1)

NetFPGA-SUME

- **projects** (including reference designs)
- **contrib-projects** (contributed user projects)
- **lib** (custom and reference IP Cores and software libraries)
- **tools** (scripts for running simulations etc.)
- **docs** (design documentations and user-guides)

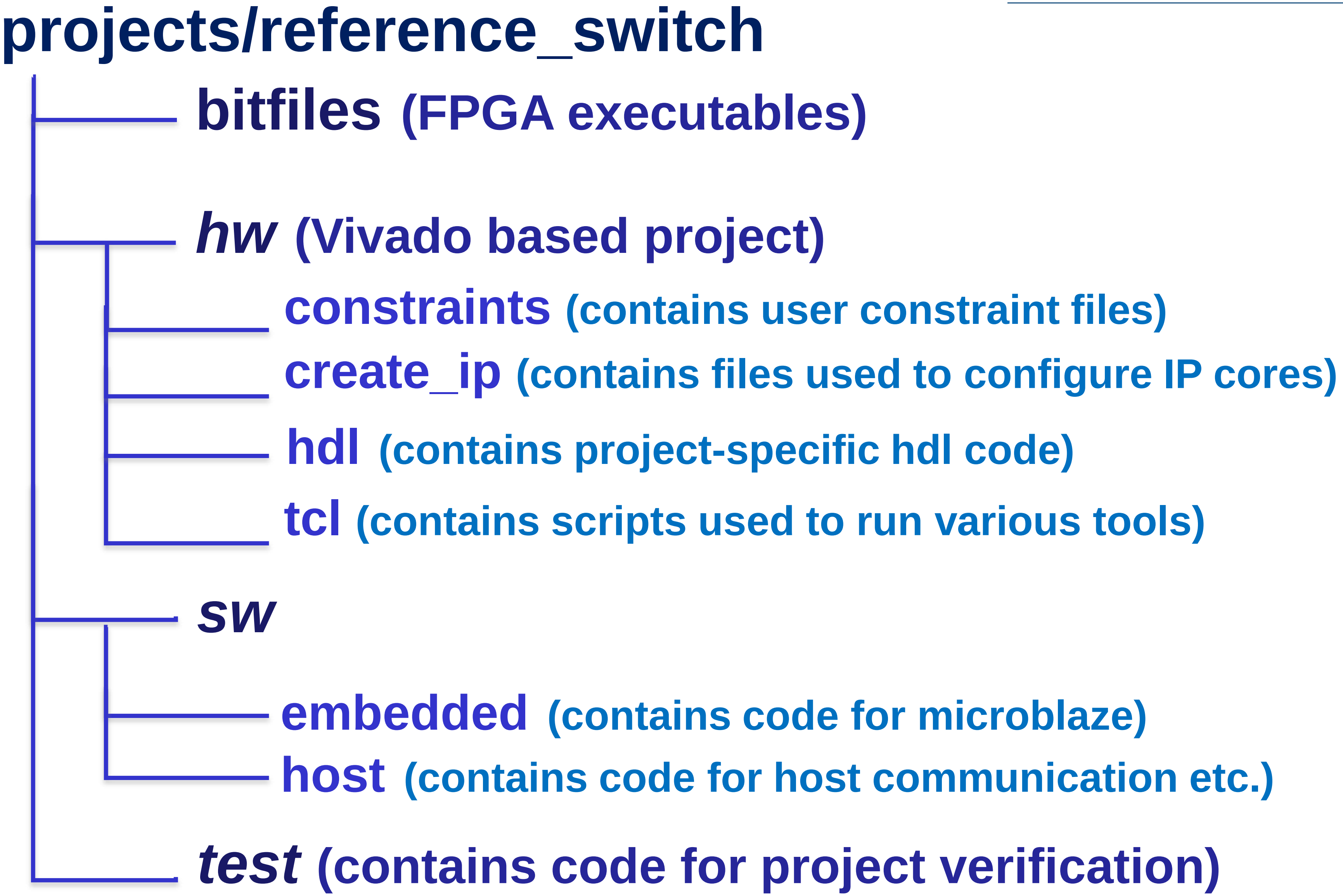
https://github.com/NetFPGA/NetFPGA-SUME-alpha

Tree Structure (2)



- **lib**
  - ***hw*** **(hardware logic as IP cores)**
    - **std** **(reference cores)**
    - **contrib** **(contributed cores)**
  - ***sw*** **(core specific software drivers/libraries)**
    - **std** **(reference libraries)**
    - **contrib** **(contributed libraries)**

Tree Structure (3)



projects/reference_switch

- **bitfiles** (FPGA executables)
- ***hw*** (Vivado based project)
  - **constraints** (contains user constraint files)
  - **create_ip** (contains files used to configure IP cores)
  - **hdl** (contains project-specific hdl code)
  - **tcl** (contains scripts used to run various tools)
- ***sw***
  - **embedded** (contains code for microblaze)
  - **host** (contains code for host communication etc.)
- ***test*** (contains code for project verification)

Reusable logic (IP cores)

Category	IP Core(s)
I/O interfaces	Ethernet 10G Port PCI Express UART GPIO
Output queues	BRAM based
Output port lookup	NIC CAM based Learning switch
Memory interfaces	SRAM DRAM FLASH
Miscellaneous	FIFOs AXIS width converter

Verification Infrastructure (1)

- **Simulation and Debugging**
  - built on industry standard Xilinx “xSim” simulator and “Scapy”
  - Python scripts for stimuli construction and verification

Verification Infrastructure (2)

- **xSim**
  - a High Level Description (HDL) simulator
  - performs functional and timing simulations for embedded, VHDL, Verilog and mixed designs
- **Scapy**
  - a powerful interactive packet manipulation library for creating “test data”
  - provides primitives for many standard packet formats
  - allows addition of custom formats

Build Infrastructure (2)

- **Build/Synthesis (using Xilinx Vivado)**
  - collection of shared hardware peripherals cores stitched together with *AXI4: Lite* and *Stream* buses
  - bitfile generation and verification using Xilinx synthesis and implementation tools

Build Infrastructure (3)

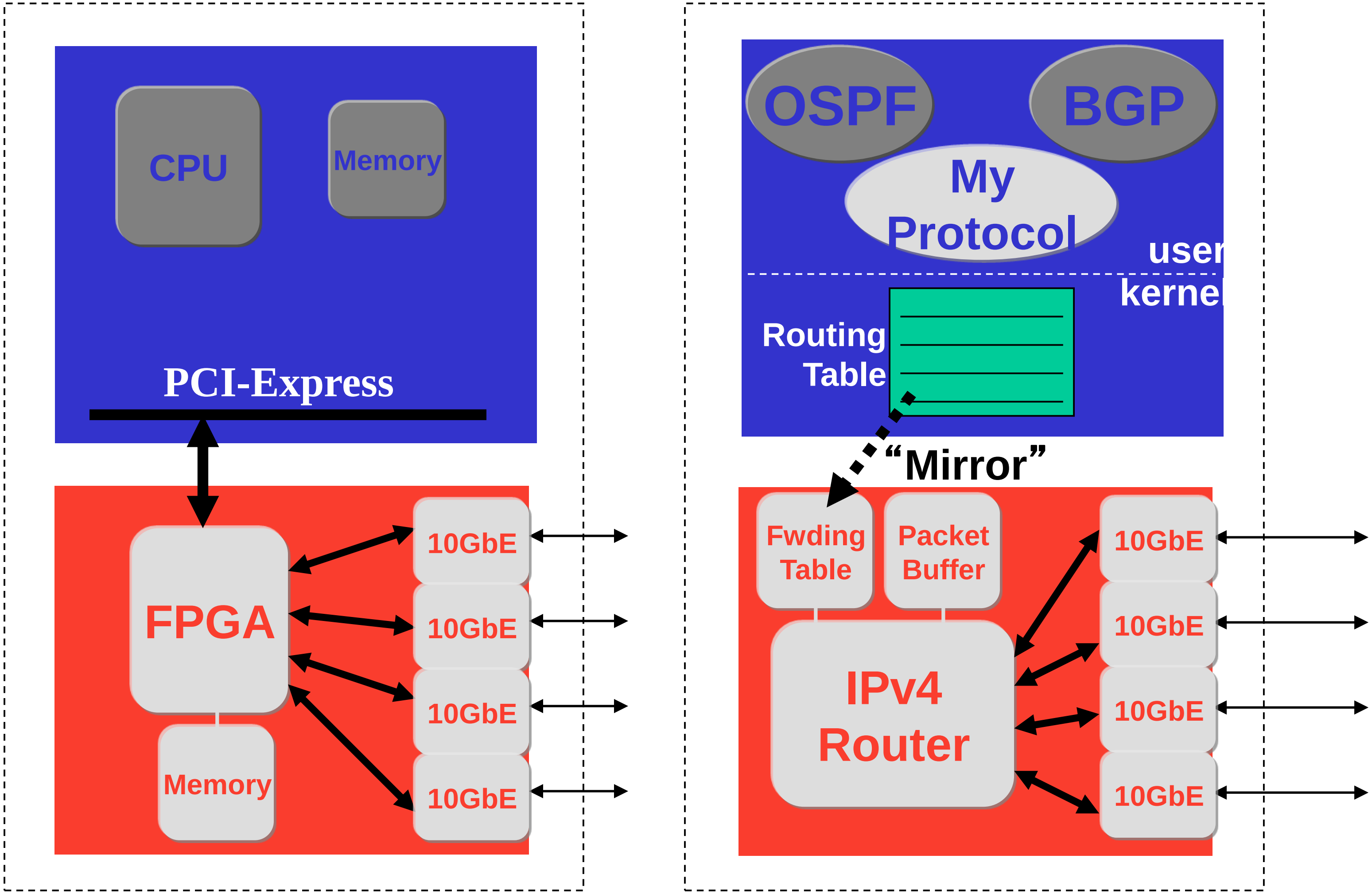
- **Register system**
  - collates and generates addresses for all the registers and memories in a project
  - uses integrated python and tcl scripts to generate HDL code (for hw) and header files (for sw)

Section VI: Examples of using NetFPGA

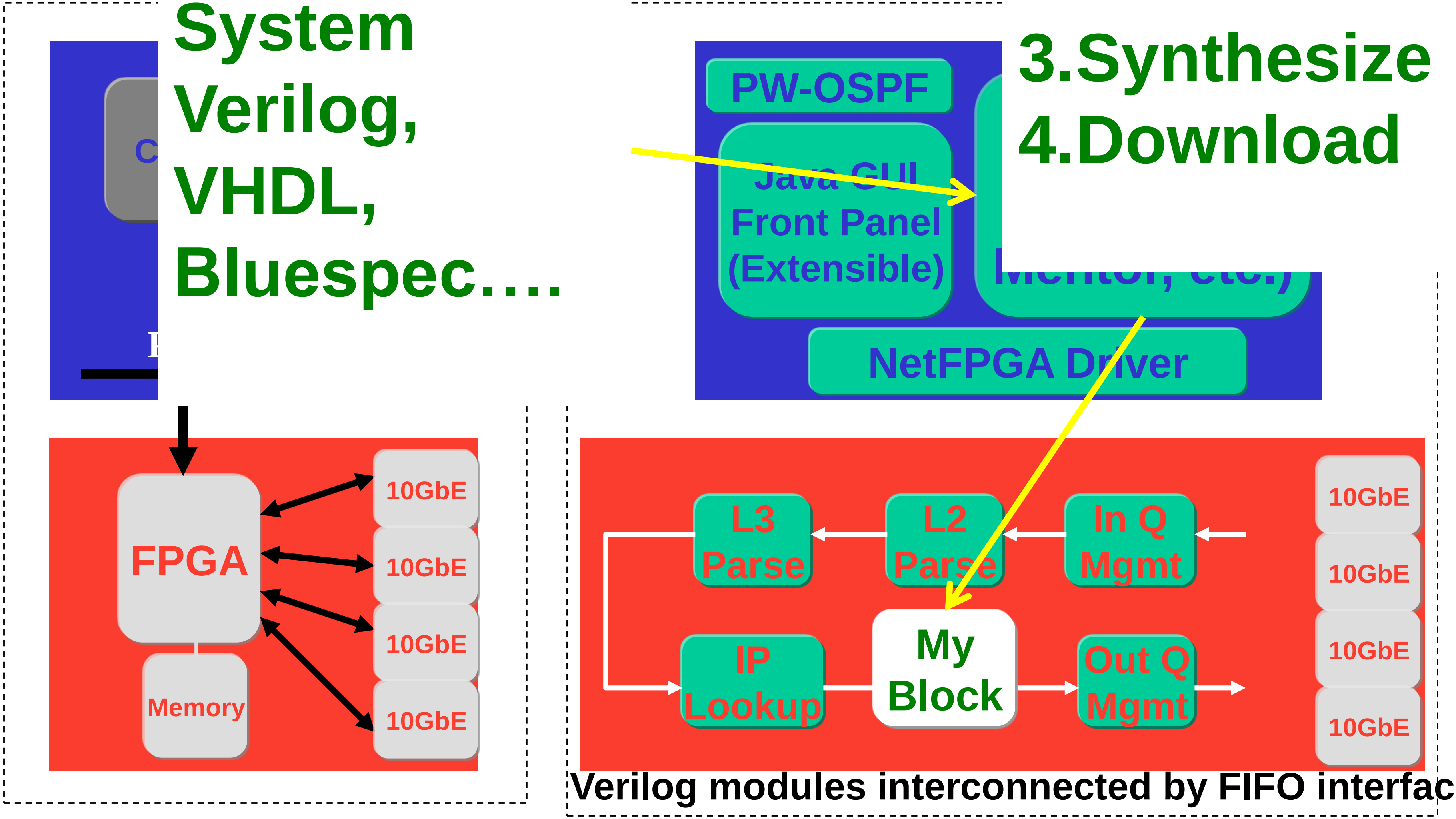
sage #1

Running the Reference Router

User-space development, 4x10GE line-rate forwarding



Enhancing Modular Reference Designs



Verilog, System Verilog, VHDL, Bluespec….

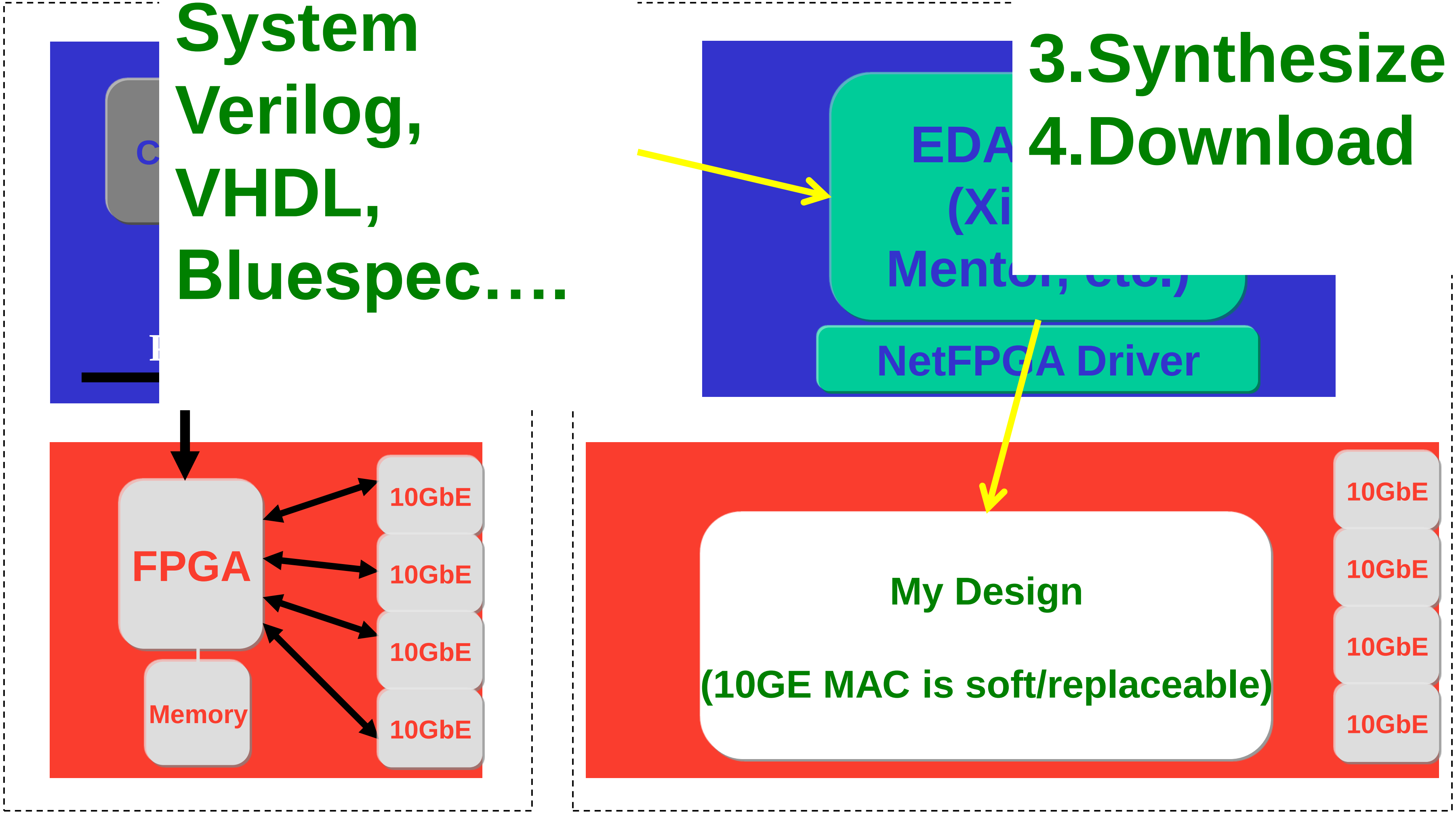
1. Design
2. Simulate
3. Synthesize
4. Download

Verilog modules interconnected by FIFO interface sage #3

Creating new systems

Verilog,
System
Verilog,
VHDL,
Bluespec….

1. Design
2. Simulate
3. Synthesize
4. Download



C

EDA
(Xi
Mento…, etc.)

NetFPGA Driver

FPGA

Memory

10GbE

10GbE

10GbE

10GbE

My Design

(10GE MAC is soft/replaceable)

10GbE

10GbE

10GbE

10GbE

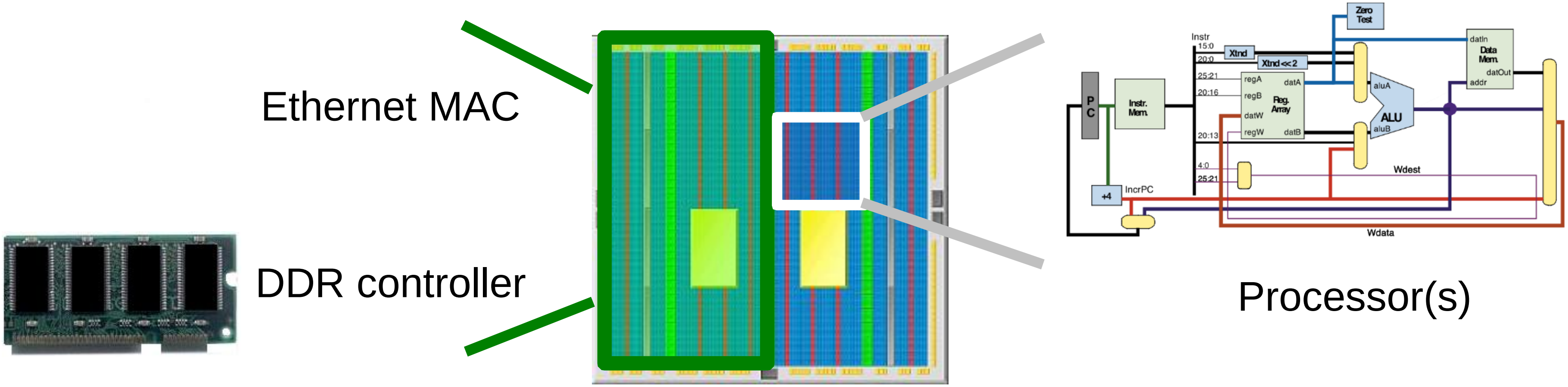
Contributed Projects

Platform	Project	Contributor
1G	OpenFlow switch	Stanford University
	Packet generator	Stanford University
	NetFlow Probe	Brno University
	NetThreads	University of Toronto
	zFilter (Sp)router	Ericsson
	Traffic Monitor	University of Catania
	DFA	UMass Lowell
10G	Bluespec switch	UCAM/SRI International
	Traffic Monitor	University of Pisa
	NF1G legacy on NF10G	Uni Pisa & Uni Cambridge
	High perf. DMA core	University of Cambridge
	BERI/CHERI	UCAM/SRI International
	OSNT	UCAM/Stanford/GTech/CNRS

OpenFlow

- **The most prominent NetFPGA success**
- **Has reignited the Software Defined Networking movement**
- **NetFPGA enabled OpenFlow**
  - A widely available open-source development platform
  - Capable of line-rate and
- **was, until its commercial uptake, the reference platform for OpenFlow.**

Soft Processors in FPGAs



Ethernet MAC

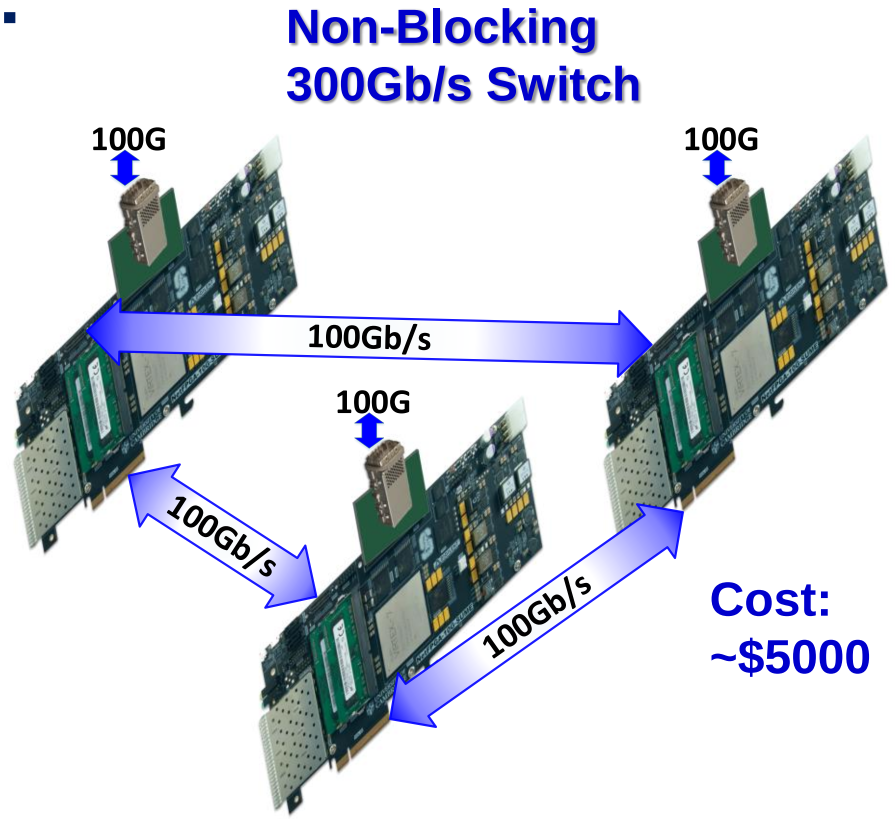
DDR controller

Processor(s)

- Soft processors: processors in the FPGA fabric
- User uploads program to soft processor
- Easier to program software than hardware in the FPGA
- Could be customized at the instruction level
- CHERI – 64bit MIPS soft processor, BSD OS

100Gb/s Aggregation

- **A development platform that can aggregate 100Gb/s for:**
  - – Operating systems
  - – Protocols Testing
  - – Measurements



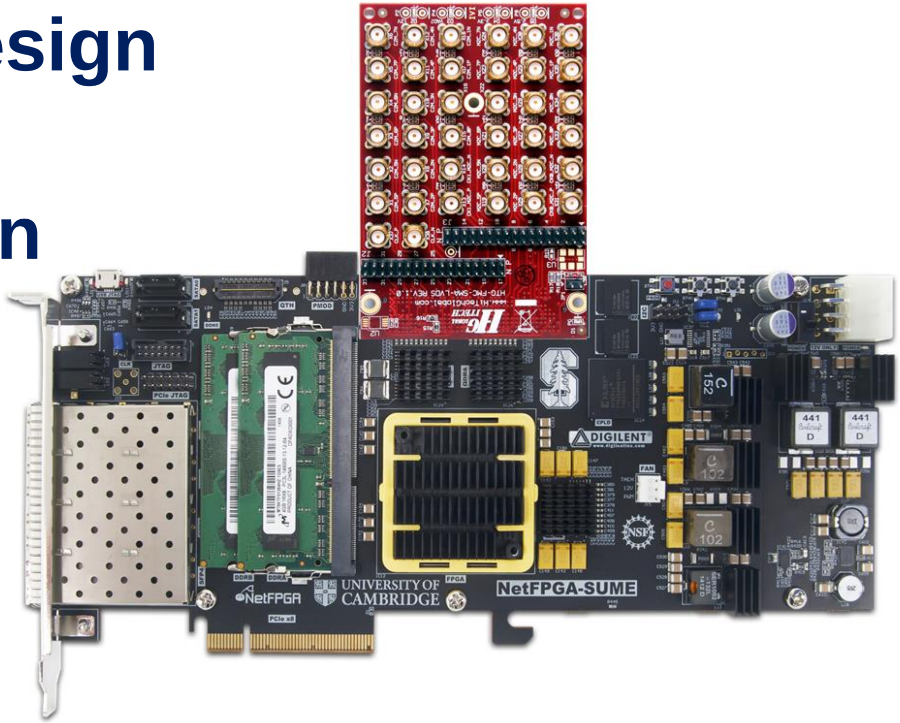
Non-Blocking 300Gb/s Switch

Cost: ~$5000

- **NetFPGA SUME can:**
  - – Aggregate 100Gb/s as Host Bus Adapter
  - – Be used to create large scale switches

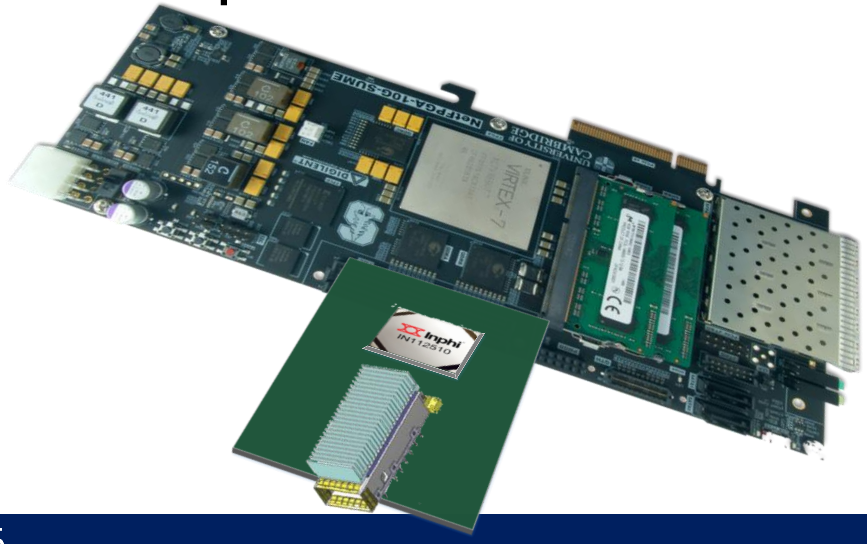
Physical Interface Design

- A deployment and interoperability test platform
  - Permits replacement of physical-layer
  - Provides high-speed expansion interfaces with standardised interfaces
- Allows researchers to design custom daughterboards
- Permits closer integration



Power Efficient MAC

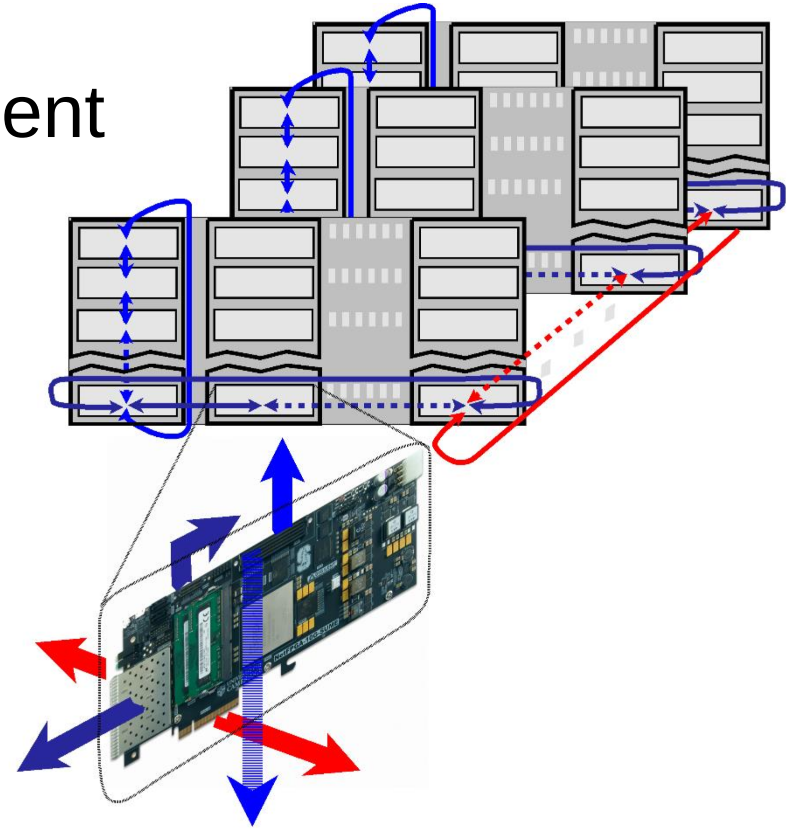
- **A Platform for 100Gb/s power-saving MAC design (e.g. lights-out MAC)**
- **Porting MAC design to SUME permits:**
  - Power measurements
  - Testing protocol’s response
  - Reconsideration of power-saving mechanisms
  - Evaluating suitability for complex architectures and systems



Interconnect

- **Novel Architectures with line-rate performance**
  - A lot of networking equipment
  - Extremely complex

- **NetFPGA SUME allows prototyping a *complete* solution**



N x N xN Hyper-cube

How might we use NetFPGA?

Well I'm not sure about you but here is a list I created:

- Build an accurate, fast, line-rate NetDummy/nistnet element
- A flexible home-grown monitoring card
- Evaluate new packet classifiers
  - (and application classifiers, and other neat network apps….)
- Prototype a full line-rate next-generation Ethernet-type
- Trying any of Jon Crowcrofts' ideas (Sourceless IP routing for example)
- Demonstrate the wonders of Metarouting in a different implementation (dedicated hardware)
- Provable hardware (using a C# implementation and kiwi with NetFPGA as target h/w)
- Hardware supporting Virtual Routers
- Check that some brave new idea actually works
  - e.g. Rate Control Protocol (RCP), Multipath TCP,
- toolkit for hardware hashing
- MOOSE implementation
- IP address anonymization
- SSL decoding "bump in the wire"
- Xen specialist nic
- computational co-processor
- Distributed computational co-processor
- IPv6 anything
- IPv6 – IPv4 gateway (6in4, 4in6, 6over4, 4over6, ….)
- Netflow v9 reference
- PSAMP reference
- IPFIX reference
- Different driver/buffer interfaces (e.g. PFRING)
- or "escalators" (from gridprobe) for faster network monitors
- Firewall reference
- GPS packet-timestamp things
- High-Speed Host Bus Adapter reference implementations
  - Infiniband
  - iSCSI
  - Myranet
  - Fiber Channel
- Smart Disk adapter (presuming a direct-disk interface)
- Software Defined Radio (SDR) directly on the FPGA (probably UWB only)
- Routing accelerator
  - Hardware route-reflector
  - Internet exchange route accelerator

- Hardware channel bonding reference implementation
- TCP sanitizer
- Other protocol sanitizer (applications… UDP DCCP, etc.)
- Full and complete Crypto NIC
- IPSec endpoint/ VPN appliance
- VLAN reference implementation
- metarouting implementation
- virtual <pick-something>
- intelligent proxy
- application embargo-er
- Layer-4 gateway
- h/w gateway for VoIP/SIP/skype
- h/w gateway for video conference spaces
- security pattern/rules matching
- Anti-spoof traceback implementations (e.g. BBN stuff)
- IPtv multicast controller
- Intelligent IP-enabled device controller (e.g. IP cameras or IP powerr
- DES breaker
- platform for flexible NIC API evaluations
- snmp statistics reference implementation
- sflow (hp) reference implementation
- trajectory sampling (reference implementation)
- implementation of zeroconf/netconf configuration language for rou
- h/w openflow and (simple) NOX controller in one…
- Network RAID (multicast TCP with redundancy)
- inline compression
- hardware accelorator for TOR
- load-balancer
- openflow with (netflow, ACL, ….)
- reference NAT device
- active measurement kit
- network discovery tool
- passive performance measurement
- active sender control (e.g. performance feedback fed to endpoints fo
- Prototype platform for NON-Ethernet or near-Ethernet MACs
  - Optical LAN (no buffers)

- Build an accurate, fast, line-rate NetDummy/nistnet element
- A flexible home-grown monitoring card
- Evaluate new packet classifiers
  - (and application classifiers, and other neat network apps….)
- Prototype a full line-rate next-generation Ethernet-type
- Trying any of Jon Crowcrofts' ideas (Sourceless IP routing for example)
- Demonstrate the wonders of Metarouting in a different implementation (dedicated hardware)
- Provable hardware (using a C# implementation and kiwi with NetFPGA as target h/w)
- Hardware supporting Virtual Routers
- Check that some brave new idea actually works
  - e.g. Rate Control Protocol (RCP), Multipath TCP,

How might YOU use NetFPGA?

- Build an accurate, fast, line-rate NetDummy/nistnet element
- A flexible home-grown monitoring card
- Evaluate new packet classifiers
  - (and application classifiers, and other neat network apps….)
- Prototype a full line-rate next-generation Ethernet-type
- Trying any of Jon Crowcrofts' ideas (Sourceless IP routing for example)
- Demonstrate the wonders of Metarouting in a different implementation (dedicated hardware)
- Provable hardware (using a C# implementation and kiwi with NetFPGA as target h/w)
- Hardware supporting Virtual Routers
- Check that some brave new idea actually works

  e.g. Rate Control Protocol (RCP), Multipath TCP,
- toolkit for hardware hashing
- MOOSE implementation
- IP address anonymization
- SSL decoding "bump in the wire"
- Xen specialist nic
- computational co-processor
- Distributed computational co-processor
- IPv6 anything
- IPv6 – IPv4 gateway (6in4, 4in6, 6over4, 4over6, ….)
- Netflow v9 reference
- PSAMP reference
- IPFIX reference
- Different driver/buffer interfaces (e.g. PFRING)
- or "escalators" (from gridprobe) for faster network monitors
- Firewall reference
- GPS packet-timestamp things
- High-Speed Host Bus Adapter reference implementations
  - Infiniband
  - iSCSI
  - Myranet
  - Fiber Channel
- Smart Disk adapter (presuming a direct-disk interface)
- Software Defined Radio (SDR) directly on the FPGA (probably UWB only)
- Routing accelerator
  - Hardware route-reflector
  - Internet exchange route accelerator

- Hardware channel bonding reference implementation
- TCP sanitizer
- Other protocol sanitizer (applications… UDP DCCP, etc.)
- Full and complete Crypto NIC
- IPSec endpoint/ VPN appliance
- VLAN reference implementation
- metarouting implementation
- virtual <pick-something>
- intelligent proxy
- application embargo-er
- Layer-4 gateway
- h/w gateway for VoIP/SIP/skype
- h/w gateway for video conference spaces
- security pattern/rules matching
- Anti-spoof traceback implementations (e.g. BBN stuff)
- IPtv multicast controller
- Intelligent IP-enabled device controller (e.g. IP cameras or IP powerr
- DES breaker
- platform for flexible NIC API evaluations
- snmp statistics reference implementation
- sflow (hp) reference implementation
- trajectory sampling (reference implementation)
- implementation of zeroconf/netconf configuration language for rou
- h/w openflow and (simple) NOX controller in one…
- Network RAID (multicast TCP with redundancy)
- inline compression
- hardware accelorator for TOR
- load-balancer
- openflow with (netflow, ACL, ….)
- reference NAT device
- active measurement kit
- network discovery tool
- passive performance measurement
- active sender control (e.g. performance feedback fed to endpoints fo
- Prototype platform for NON-Ethernet or near-Ethernet MACs
  - Optical LAN (no buffers)

Section VII: Example Project: Crypto Switch

Project: Cryptographic Switch

Implement a learning switch that encrypts upon transmission and decrypts upon reception

Cryptography

XOR function

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

XORing a value with itself *always* yields 0

XOR written as: ^ ⊻ ⊕

XOR is *commutative:* (A ^ B) ^ C = A ^ (B ^ C)

Cryptography (cont.)

Simple cryptography:

- Generate a secret key
- Encrypt the message by XORing the message and key
- Decrypt the ciphertext by XORing with the key

Explanation:

$$\begin{aligned}(M \;\hat{}\; K) \;\hat{}\; K &= M \;\hat{}\; (K \;\hat{}\; K) && \longleftarrow \boxed{\text{Commutativity}} \\ &= M \;\hat{}\; 0 && \longleftarrow \boxed{A \;\hat{}\; A = 0} \\ &= M\end{aligned}$$

Cryptography (cont.)

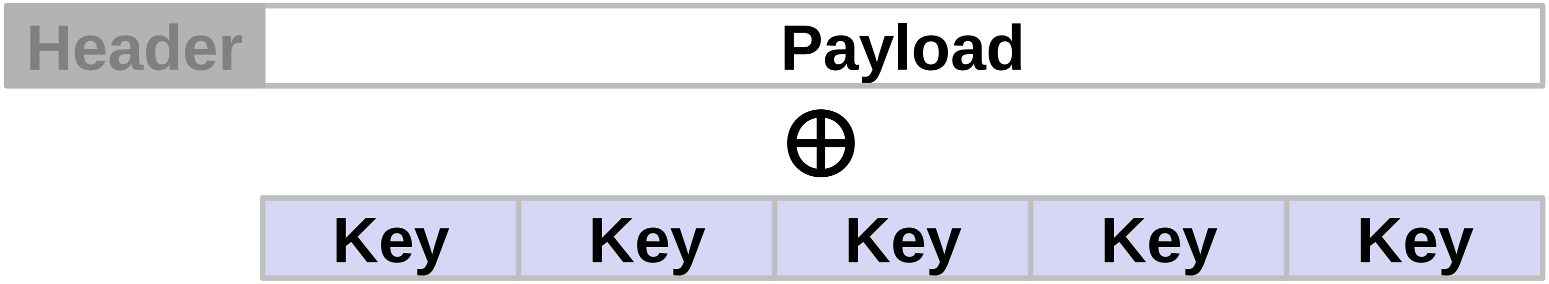
Example:

Message:	00111011
Key:	10110001
Message ^ Key:	10001010
Key:	10110001
Message ^ Key ^ Key:	00111011

Cryptography (cont.)

Idea: Implement simple cryptography using XOR

- 32-bit key
- Encrypt every word in payload with key



Note: XORing with a one-time pad of the same length of the message is secure/uncrackable. See: http://en.wikipedia.org/wiki/One-time_pad

implementation goes wild…

What’s a core?

- “IP Core” in Vivado
  - Standalone Module
  - Configurable and reuseable

- HDL (Verilog/VHDL) + TCL files

- Examples:
  - 10G Port
  - SRAM Controller
  - NIC Output port lookup

HDL (Verilog)

- **NetFPGA cores**
  - AXI-compliant

- **AXI = Advanced eXtensible Interface**
  - Used in ARM-based embedded systems
  - Standard interface
  - **AXI4/AXI4-Lite**: Control and status interface
  - **AXI4-Stream**: Data path interface

- **Xilinx IPs and tool chains**
  - Mostly AXI-compliant

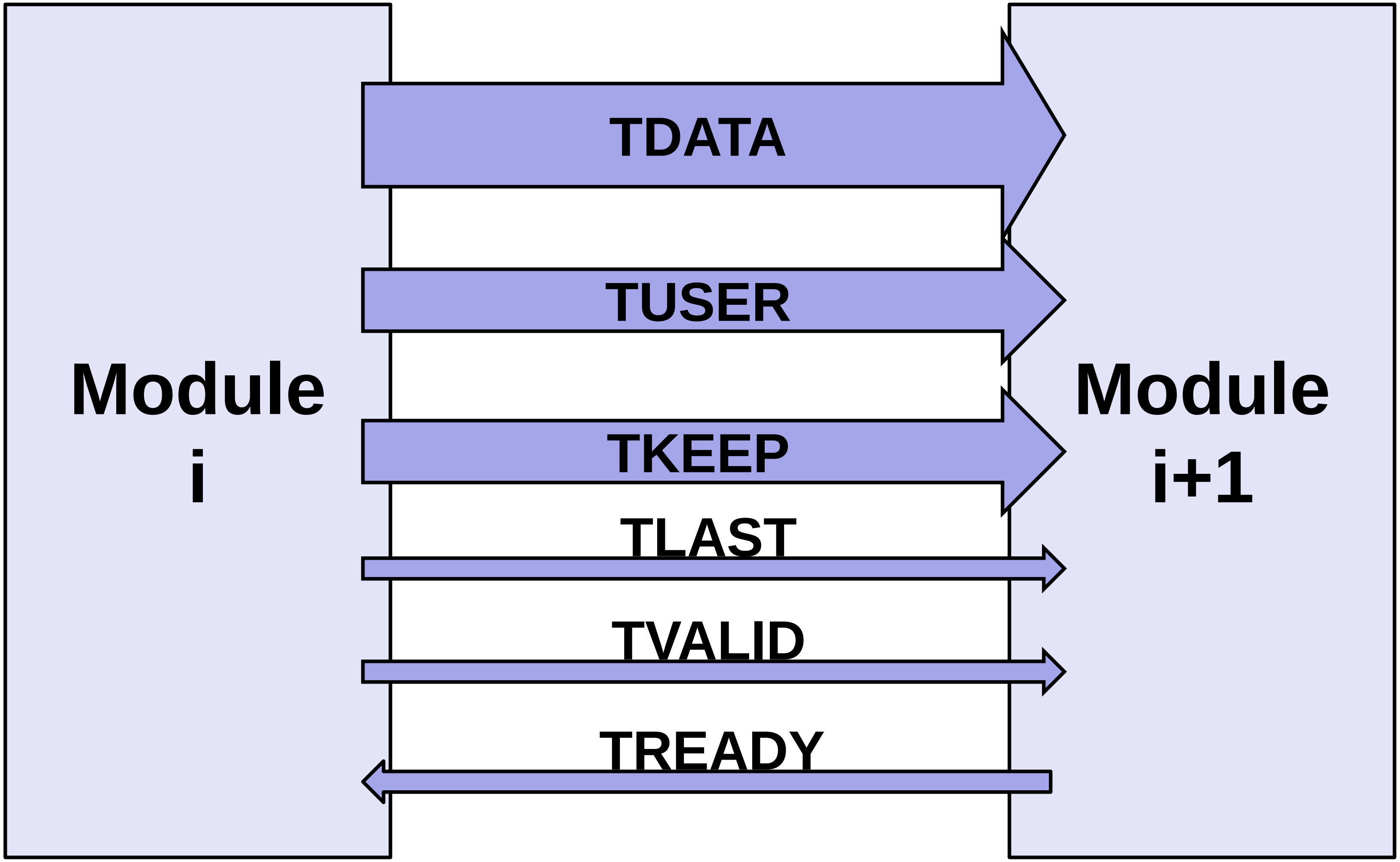
Scripts (TCL)

- **Integrated into Vivado toolchain**
  - Supports Vivado-specific commands
  - Allows to interactively query Vivado

- **Has a large number of uses:**
  - Create projects
  - Set properties
  - Generate cores
  - Define connectivity
  - Etc.

Inter-Module Communication

– Using AXI-4 Stream (*Packets are moved as Stream)*



AXI4-Stream

AXI4-Stream	Description
TDATA	Data Stream
TKEEP	Marks NULL bytes (i.e. byte enable)
TVALID	Valid Indication
TREADY	Flow control indication
TLAST	End of packet/burst indication
TUSER	Out of band metadata

Packet Format

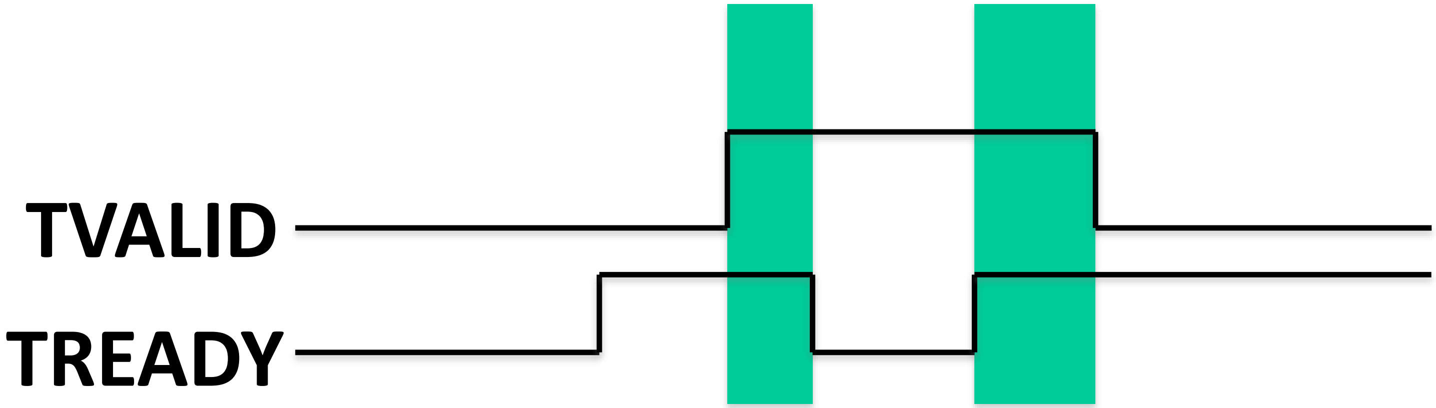
TLAST	TUSER	TKEEP	TDATA
0	X	0xFF…F	Eth Hdr
0	X	0xFF…F	IP Hdr
0	X	0xFF…F	…
1	X	0x0…1F	Last word

TUSER

Position	Content
[15:0]	length of the packet in bytes
[23:16]	source port: one-hot encoded
[31:24]	destination port: one-hot encoded
[127:32]	6 user defined slots, 16bit each

TVALID/TREADY Signal timing

- No waiting!
- Assert TREADY/TVALID whenever appropriate
- TVALID should ***<u>not</u>*** depend on TREADY



Byte ordering

- **In compliance to AXI, NetFPGA has a specific byte ordering**
  - 1st byte of the packet @ TDATA[7:0]
  - 2nd byte of the packet @ TDATA[15:8]

Getting started with a new project:

Embedded Development Kit

- **Xilinx integrated design environment contains:**
  - **Vivado**, a top level integrated design tool for “hardware” synthesis , implementation and bitstream generation
  - **Software Development Kit (SDK)**, a development environment for “software application” running on embedded processors like Microblaze
  - **Additional tools** (e.g. Vivado HLS)

Xilinx Vivado

- **A Vivado project consists of following:**
  - **<project_name>.xpr**
    - top level Vivado project file
  - **Tcl and HDL files that define the project**
  - **system.xdc**
    - user constraint file
    - defines constraints such as timing, area, IO placement etc.

Xilinx Vivado (2)

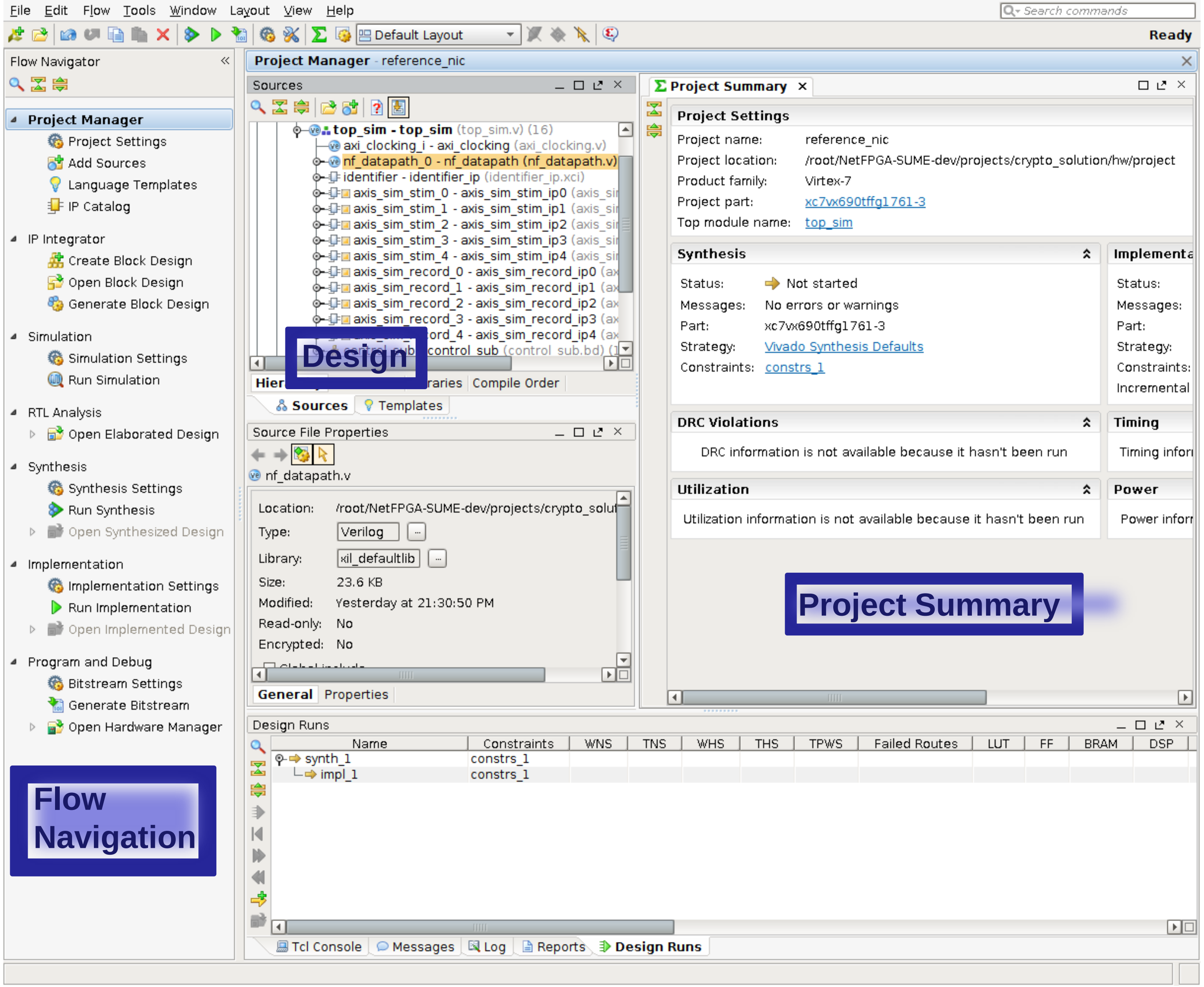
- **To invoke Vivado design tool, run:**

```
# vivado <project_root>/hw/project/<project_name>.xpr
```

- **This will open the project in the Vivado graphical user interface**

  - `open a new terminal`
  - `cd <project_root>/projects/ <project_name>/`
  - `source /opt/Xilinx/Vivado/2014.4/settings64.sh`
  - `vivado hw/project/<project name>.xpr`

Vivado Design Tool (1)


File Edit Flow Tools Window Layout View Help
Search commands
Default Layout
Ready
Flow Navigator
Project Manager
Project Settings
Add Sources
Language Templates
IP Catalog
IP Integrator
Create Block Design
Open Block Design
Generate Block Design
Simulation
Simulation Settings
Run Simulation
RTL Analysis
Open Elaborated Design
Synthesis
Synthesis Settings
Run Synthesis
Open Synthesized Design
Implementation
Implementation Settings
Run Implementation
Open Implemented Design
Program and Debug
Bitstream Settings
Generate Bitstream
Open Hardware Manager
Project Manager - reference_nic
Sources
top_sim - top_sim (top_sim.v) (16)
axi_clocking_i - axi_clocking (axi_clocking.v)
nf_datapath_0 - nf_datapath (nf_datapath.v)
identifier - identifier_ip (identifier_ip.xci)
axis_sim_stim_0 - axis_sim_stim_ip0
axis_sim_stim_1 - axis_sim_stim_ip1
axis_sim_stim_2 - axis_sim_stim_ip2
axis_sim_stim_3 - axis_sim_stim_ip3
axis_sim_stim_4 - axis_sim_stim_ip4
axis_sim_record_0 - axis_sim_record_ip0
axis_sim_record_1 - axis_sim_record_ip1
axis_sim_record_2 - axis_sim_record_ip2
axis_sim_record_3 - axis_sim_record_ip3
axis_sim_record_4 - axis_sim_record_ip4
control_sub (control_sub.bd)
Hierarchy Libraries Compile Order
Sources Templates
Source File Properties
nf_datapath.v
General Properties
Project Summary
Tcl Console Messages Log Reports Design Runs

Design

Project Summary

Flow Navigation

Source File Properties – nf_datapath.v

Location: /root/NetFPGA-SUME-dev/projects/crypto_solut
Type: Verilog
Library: xil_defaultlib
Size: 23.6 KB
Modified: Yesterday at 21:30:50 PM
Read-only: No
Encrypted: No

Project Settings

Project name: reference_nic
Project location: /root/NetFPGA-SUME-dev/projects/crypto_solution/hw/project
Product family: Virtex-7
Project part: xc7vx690tffg1761-3
Top module name: top_sim

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7vx690tffg1761-3
Strategy: Vivado Synthesis Defaults
Constraints: constrs_1

Implementation

Status:
Messages:
Part:
Strategy:
Constraints:
Incremental

DRC Violations

DRC information is not available because it hasn't been run

Timing

Timing infor

Utilization

Utilization information is not available because it hasn't been run

Power

Power infor

Design Runs

Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP
synth_1	constrs_1										
impl_1	constrs_1										

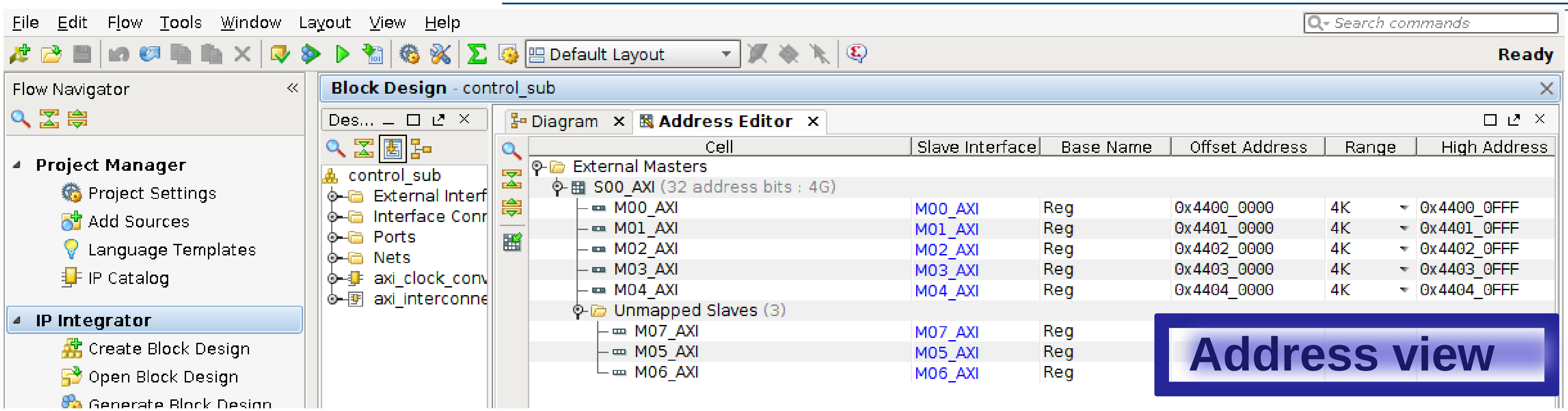
Vivado Design Tool (2)

- IP Catalog: contains categorized list of all available peripheral cores

- IP Integrator: shows connectivity of various modules over AXI bus

- Project manager: provides a complete view of instantiated cores

Vivado Design Tool (3)



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
External Masters					
S00_AXI (32 address bits : 4G)					
M00_AXI	M00_AXI	Reg	0x4400_0000	4K	0x4400_0FFF
M01_AXI	M01_AXI	Reg	0x4401_0000	4K	0x4401_0FFF
M02_AXI	M02_AXI	Reg	0x4402_0000	4K	0x4402_0FFF
M03_AXI	M03_AXI	Reg	0x4403_0000	4K	0x4403_0FFF
M04_AXI	M04_AXI	Reg	0x4404_0000	4K	0x4404_0FFF
Unmapped Slaves (3)					
M07_AXI	M07_AXI	Reg			
M05_AXI	M05_AXI	Reg			
M06_AXI	M06_AXI	Reg			

Address view

- **Address Editor:**
  - **- Under IP Integrator**
  - **- Defines base and high address value for peripherals connected to AXI4 or AXI-LITE bus**
    - **Not AXI-Stream!**
- **These values can be controlled manually, using tcl**

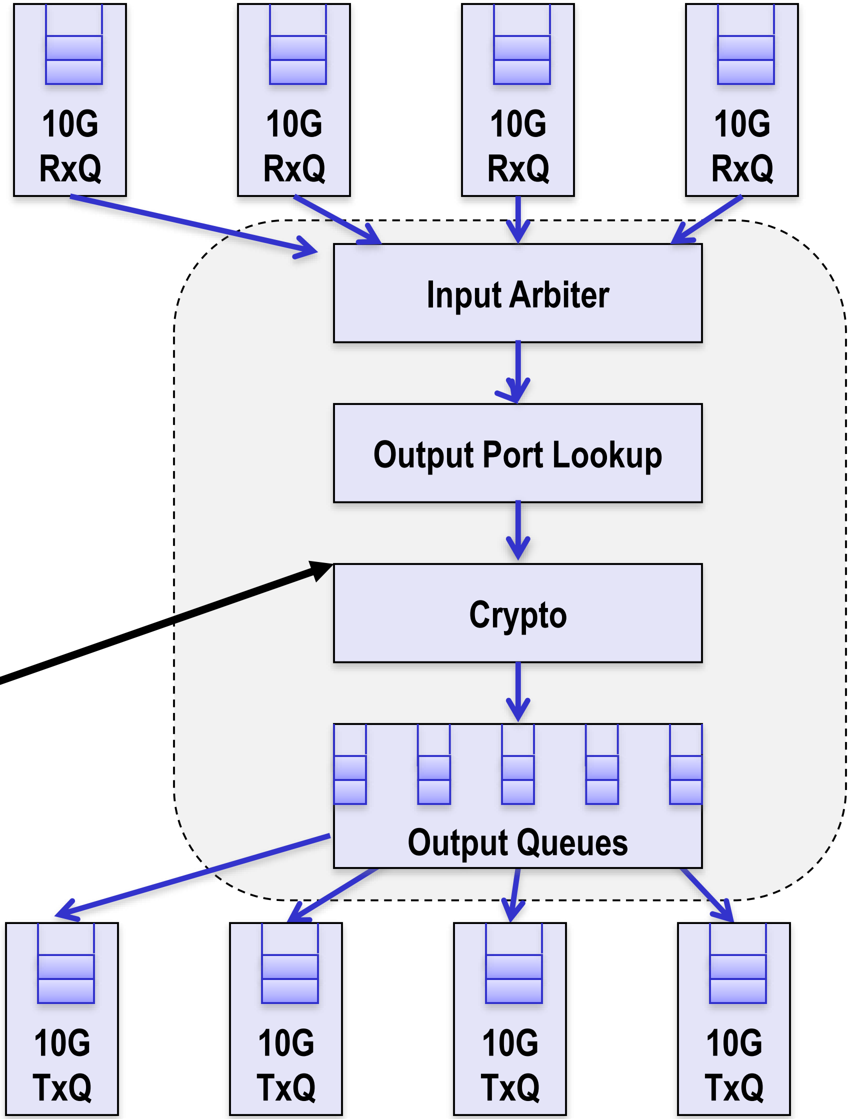
Getting started with a new project (1)

- **Projects:**
  - – Each design is represented by a project
  - – Location: NetFPGA-SUME-alpha/projects/<proj_name>
  - – Create a new project:
    - Normally:
      - – copy an existing project as the starting point
    - Today:
      - – pre-created project (crypto_switch)
  - – Consists of:
    - Verilog source
    - Simulation tests
    - Hardware tests
    - Optional software

Getting started with a new project (3)

Typically implement functionality in one or more modules under the top wrapper

Crypto module to encrypt and decrypt packets



Getting started with a new project (4)

- Shared modules included from netfpga/lib/hw
  - Generic modules that are re-used in multiple projects
  - Specify shared modules in project's tcl file

- crypto_switch:

Local	Shared
crypto	Everything else

Getting started with a new project (5)

Create crypto core using core template:

1. cd $NF_DESIGN_DIR/hw/local_ip
2. cp -r example_ip crypto
3. Write and edit files under *crypto* Folder
4. cd $NF_DESIGN_DIR/hw/
5. vi Makefile
   - Refer to Line 61
6. make core

Notes:

1. review ~/NetFPGA-SUME-alpha/tools/settings.sh
2. make sure NF_PROJECT_NAME=crypto_switch
3. If you make chages: source ~/NetFPGA-SUME-alpha/tools/settings.sh

crypto.v

```
Module crypto
   #(
      parameter C_M_AXIS_DATA_WIDTH = 256,
      parameter C_S_AXIS_DATA_WIDTH = 256,
       ...)
   (
      ...
   )

 //----------------------- regs/wires ---------------------------
   ...
   //----------------------- modules ------------------------------
   ...
   //----------------------- logic ------------------------------
   ...

endmodule
```

Module port declaration

crypto.v (2)

```
//------------------------- Modules------

  fallthrough_small_fifo #(
     .WIDTH(...),
     .MAX_DEPTH_BITS(2)
  ) input_fifo (
     .din        ({fifo_out_tlast, fifo_out_tuser,..}), // Data in
     .wr_en      (s_axis_tvalid & s_axis_tready),  // Write enable
     .rd_en      (in_fifo_rd_en),        // Read the next word
     .dout       ({s_axis_tlast, s_axis_tuser, ..}),
     .full       (),
     .nearly_full(in_fifo_nearly_full),
     .prog_full  (),
     .empty      (in_fifo_empty),
     .reset      (!axi_aresetn),
     .clk        (axi_aclk)
  );
```

Packet data dumped in a FIFO. Allows some “decoupling” between input and output.

crypto.v (3)

```
//------------------------- Logic-------------------------------

assign s_axis_tready = !in_fifo_nearly_full;
assign m_axis_tuser = fifo_out_tuser;
...

always @(*) begin
     // Default value
     in_fifo_rd_en = 0;

     if (m_axis_tready && !in_fifo_empty) begin
        in_fifo_rd_en = 1;
     end
  end
```

Combinational logic to read data from the FIFO. (Data is output to output ports.)

You'll want to add your state in this section.

Project Design Flow

- **There are several ways to design and integrate a project, e.g.**
  - Using Verilog files for connectivity and TCL scripts for project definition
  - Using Vivado's Block Design (IPI) flow

- **We will use the first, but introduce the second**

Project Integration

- **vi $NF_DESIGN_DIR/hw/nf_datapath.v**
- **Add the new module between the output port lookup and output queues**
- **Connect S3_AXI to the AXI_Lite interface of the block**
  - Not mandatory now, but will help for tomorrow

Project Integration

- **Edit the TCL file which generates the project:**
- **vi $NF_DESIGN_DIR/hw/tcl/<project_name>_sim.tcl**
- **Add the following lines:**

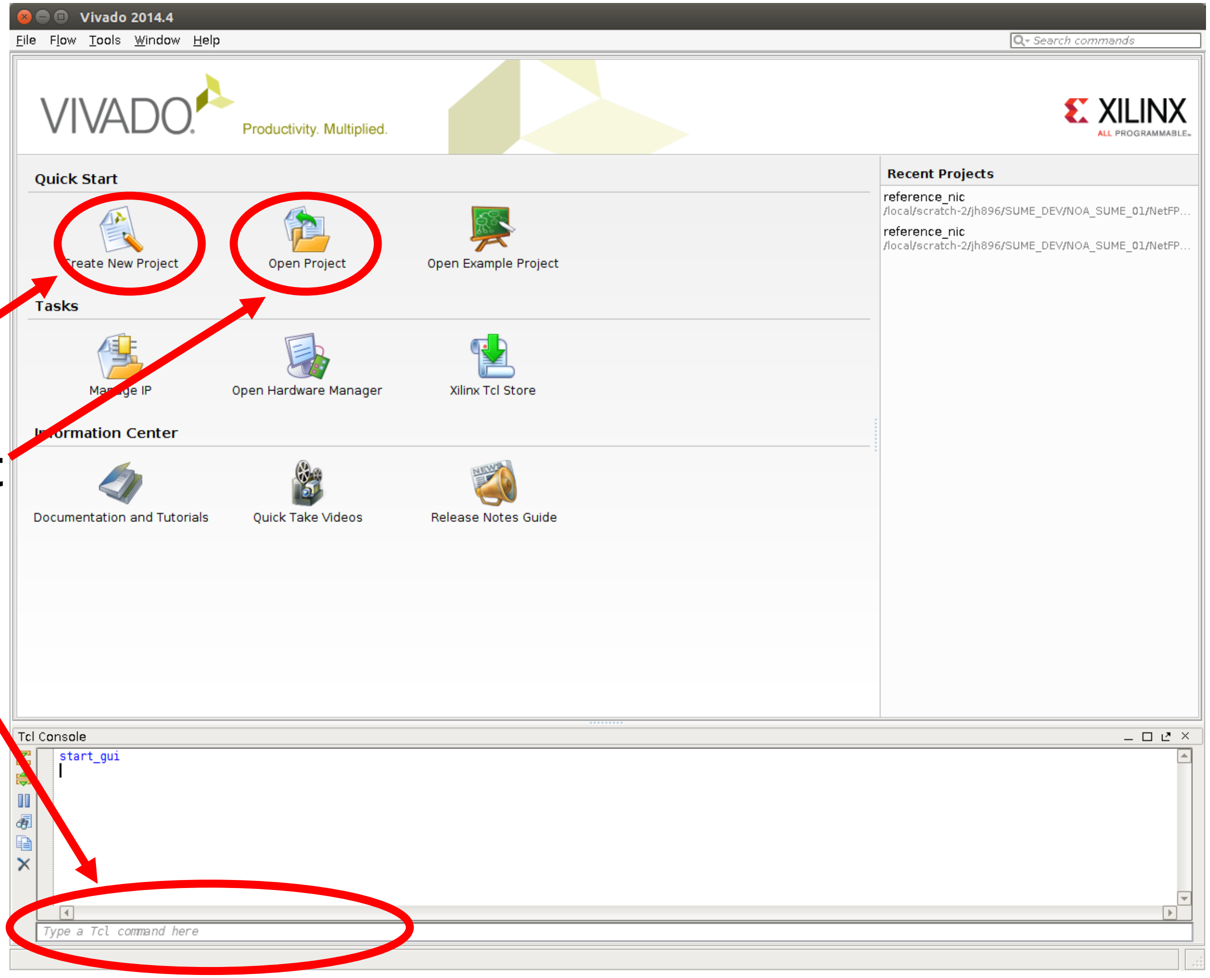
```
create_ip -name <core_name> -vendor NetFPGA -library NetFPGA -module_name <core>_ip

set_property generate_synth_checkpoint false [get_files <core>_ip.xci]

reset_target all [get_ips <core>_ip]

generate_target all [get_ips <core>_ip]
```

- **Save time for later, add the same text also in:**

$NF_DESIGN_DIR/tcl/<project_name>.tcl

Project Integration – Block Design

Create a new project
OR
Open an existing project
OR
run a TCL script
(also through tools)

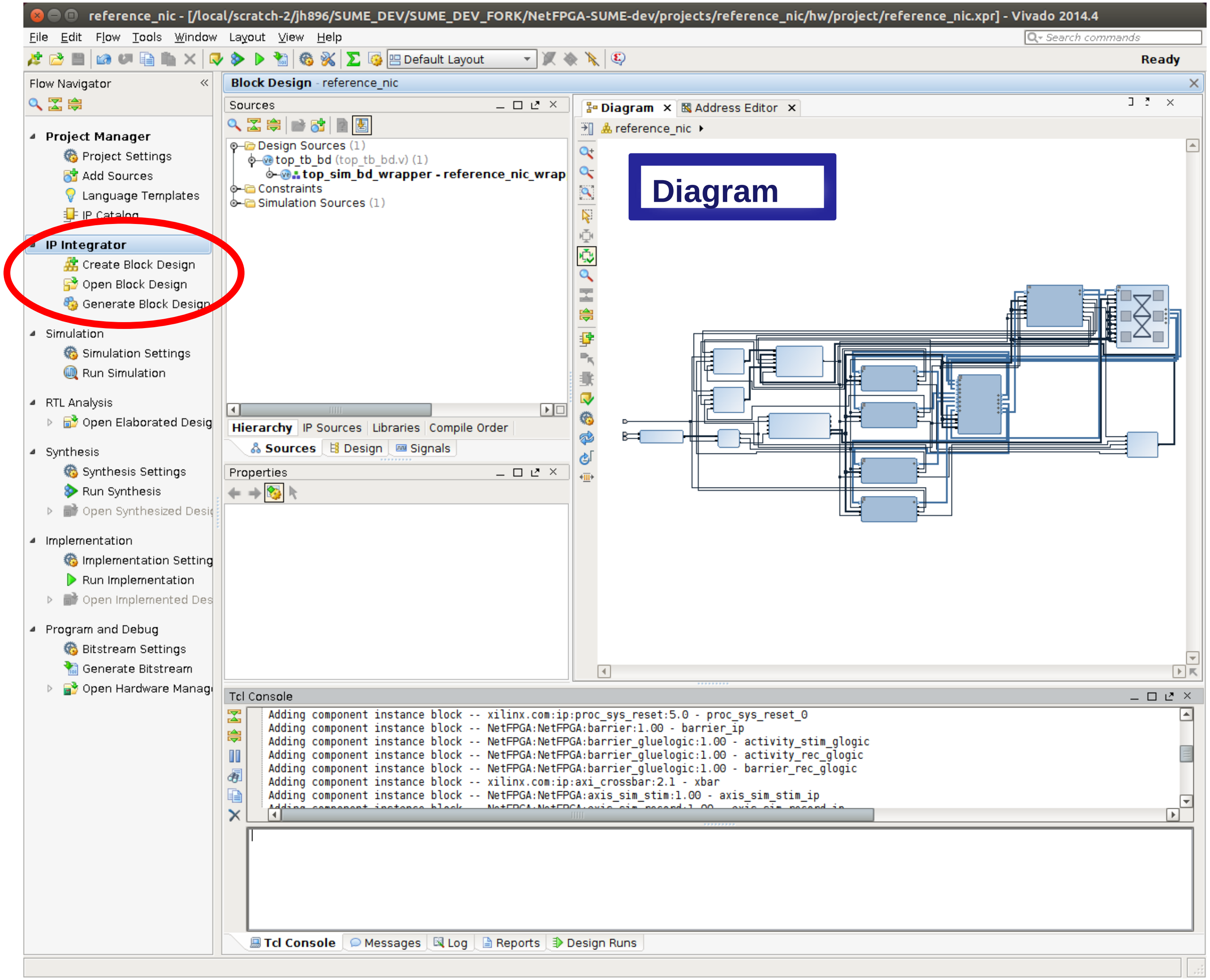


reference_nic
/local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFP...
reference_nic
/local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFP...

```
start_gui
```

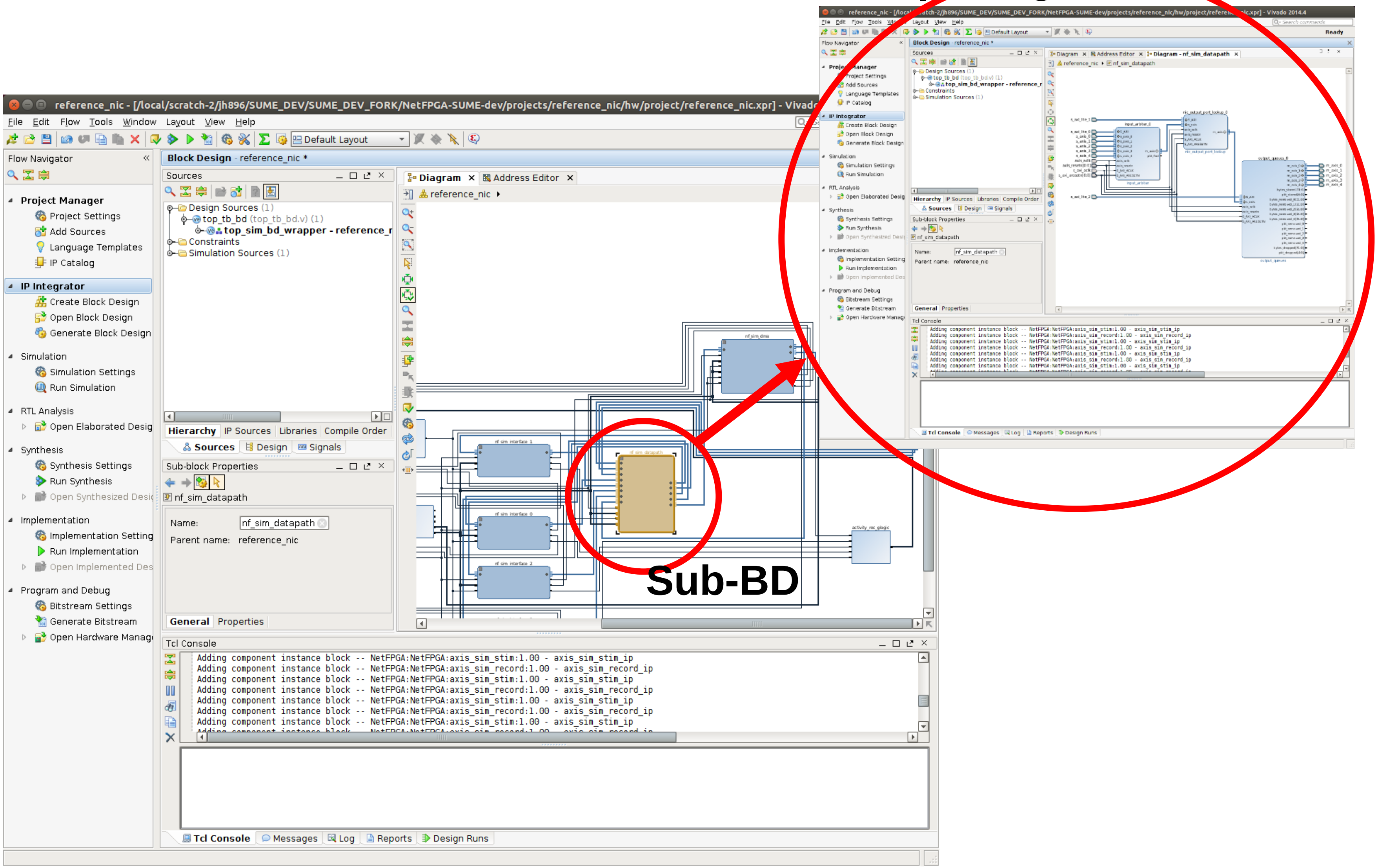
Project Integration – Block Design (2)

Open block design



```
Adding component instance block -- xilinx.com:ip:proc_sys_reset:5.0 - proc_sys_reset_0
Adding component instance block -- NetFPGA:NetFPGA:barrier:1.00 - barrier_ip
Adding component instance block -- NetFPGA:NetFPGA:barrier_gluelogic:1.00 - activity_stim_glogic
Adding component instance block -- NetFPGA:NetFPGA:barrier_gluelogic:1.00 - activity_rec_glogic
Adding component instance block -- NetFPGA:NetFPGA:barrier_gluelogic:1.00 - barrier_rec_glogic
Adding component instance block -- xilinx.com:ip:axi_crossbar:2.1 - xbar
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
```

Project Integration – Block Design (3)

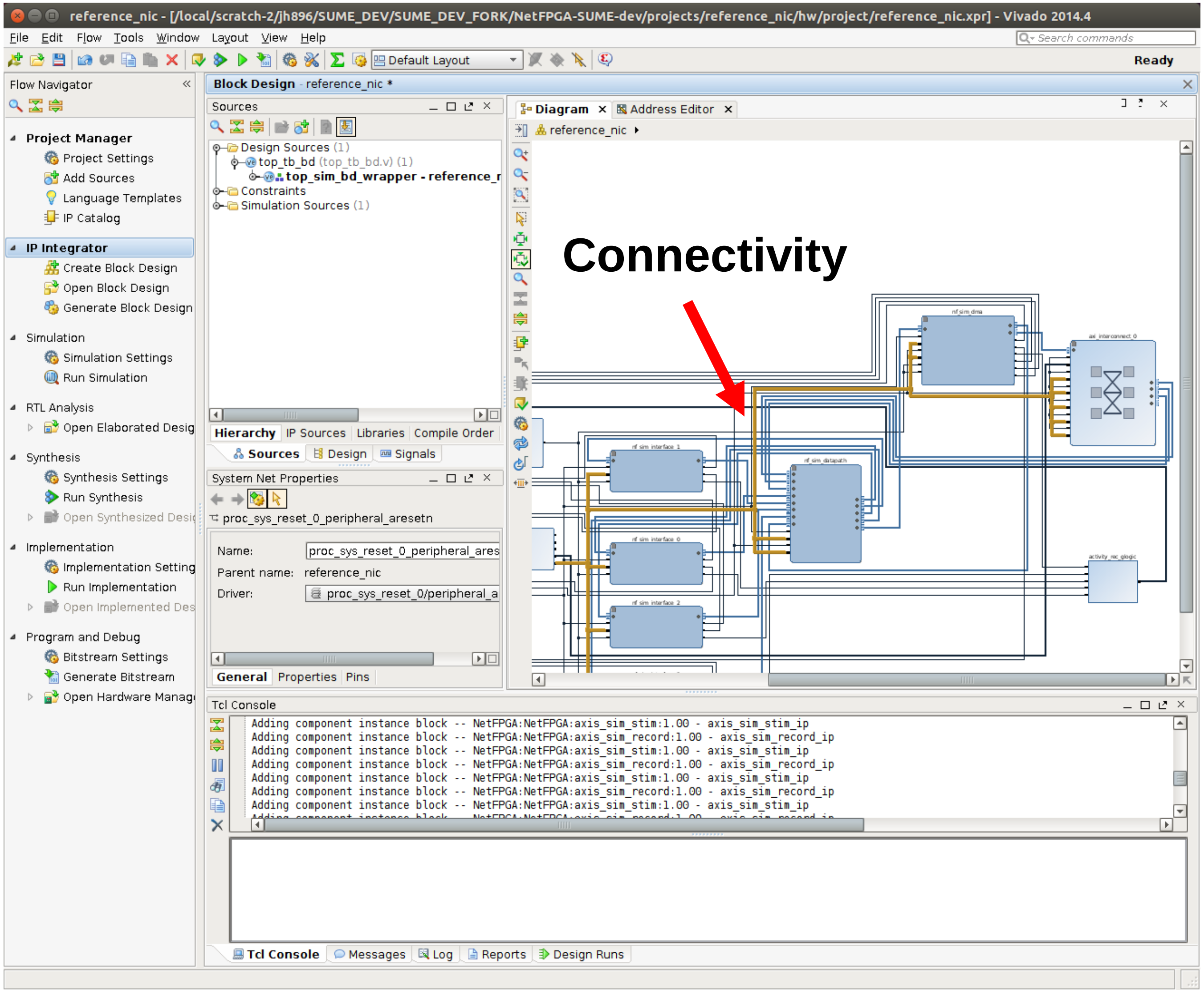


Opening Sub-BD

Sub-BD

```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
```

Project Integration – Block Design (4)



reference_nic - [/local/scratch-2/jh896/SUME_DEV/SUME_DEV_FORK/NetFPGA-SUME-dev/projects/reference_nic/hw/project/reference_nic.xpr] - Vivado 2014.4

Name: proc_sys_reset_0_peripheral_ares

Parent name: reference_nic

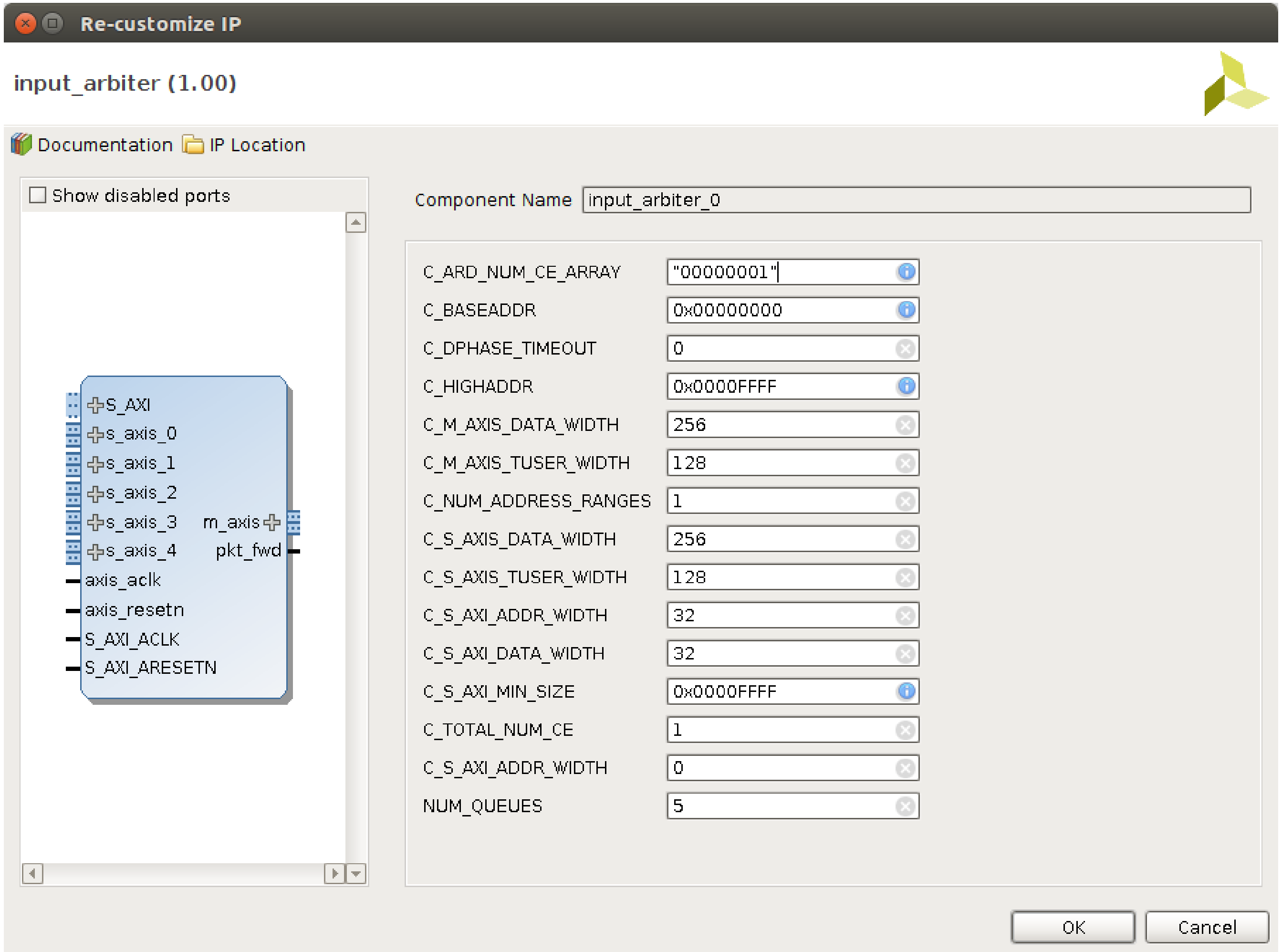
Driver: proc_sys_reset_0/peripheral_a

```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
```

Connectivity

Project Integration – Block Design (5)

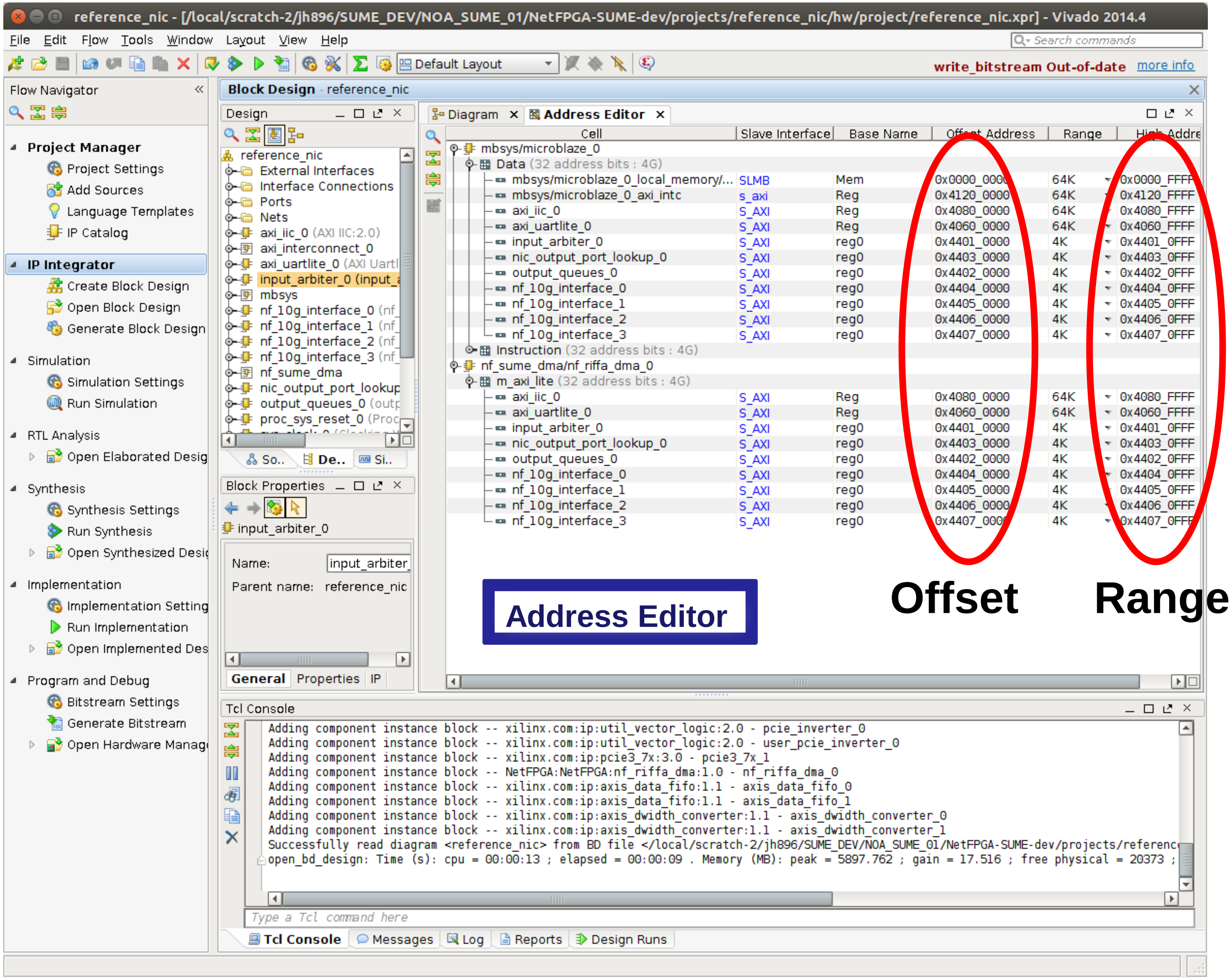
Setting module parameters



Component Name: input_arbiter_0

C_ARD_NUM_CE_ARRAY	"00000001"
C_BASEADDR	0x00000000
C_DPHASE_TIMEOUT	0
C_HIGHADDR	0x0000FFFF
C_M_AXIS_DATA_WIDTH	256
C_M_AXIS_TUSER_WIDTH	128
C_NUM_ADDRESS_RANGES	1
C_S_AXIS_DATA_WIDTH	256
C_S_AXIS_TUSER_WIDTH	128
C_S_AXI_ADDR_WIDTH	32
C_S_AXI_DATA_WIDTH	32
C_S_AXI_MIN_SIZE	0x0000FFFF
C_TOTAL_NUM_CE	1
C_S_AXI_ADDR_WIDTH	0
NUM_QUEUES	5

Project Integration – Block Design (6)



Name: input_arbiter_

Parent name: reference_nic

Cell	Slave Interface	Base Name	Offset Address	Range	High Addre
mbsys/microblaze_0					
Data (32 address bits : 4G)					
mbsys/microblaze_0_local_memory/...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
mbsys/microblaze_0_axi_intc	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
axi_iic_0	S_AXI	Reg	0x4080_0000	64K	0x4080_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
input_arbiter_0	S_AXI	reg0	0x4401_0000	4K	0x4401_0FFF
nic_output_port_lookup_0	S_AXI	reg0	0x4403_0000	4K	0x4403_0FFF
output_queues_0	S_AXI	reg0	0x4402_0000	4K	0x4402_0FFF
nf_10g_interface_0	S_AXI	reg0	0x4404_0000	4K	0x4404_0FFF
nf_10g_interface_1	S_AXI	reg0	0x4405_0000	4K	0x4405_0FFF
nf_10g_interface_2	S_AXI	reg0	0x4406_0000	4K	0x4406_0FFF
nf_10g_interface_3	S_AXI	reg0	0x4407_0000	4K	0x4407_0FFF
Instruction (32 address bits : 4G)					
nf_sume_dma/nf_riffa_dma_0					
m_axi_lite (32 address bits : 4G)					
axi_iic_0	S_AXI	Reg	0x4080_0000	64K	0x4080_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
input_arbiter_0	S_AXI	reg0	0x4401_0000	4K	0x4401_0FFF
nic_output_port_lookup_0	S_AXI	reg0	0x4403_0000	4K	0x4403_0FFF
output_queues_0	S_AXI	reg0	0x4402_0000	4K	0x4402_0FFF
nf_10g_interface_0	S_AXI	reg0	0x4404_0000	4K	0x4404_0FFF
nf_10g_interface_1	S_AXI	reg0	0x4405_0000	4K	0x4405_0FFF
nf_10g_interface_2	S_AXI	reg0	0x4406_0000	4K	0x4406_0FFF
nf_10g_interface_3	S_AXI	reg0	0x4407_0000	4K	0x4407_0FFF

Address Editor

Offset

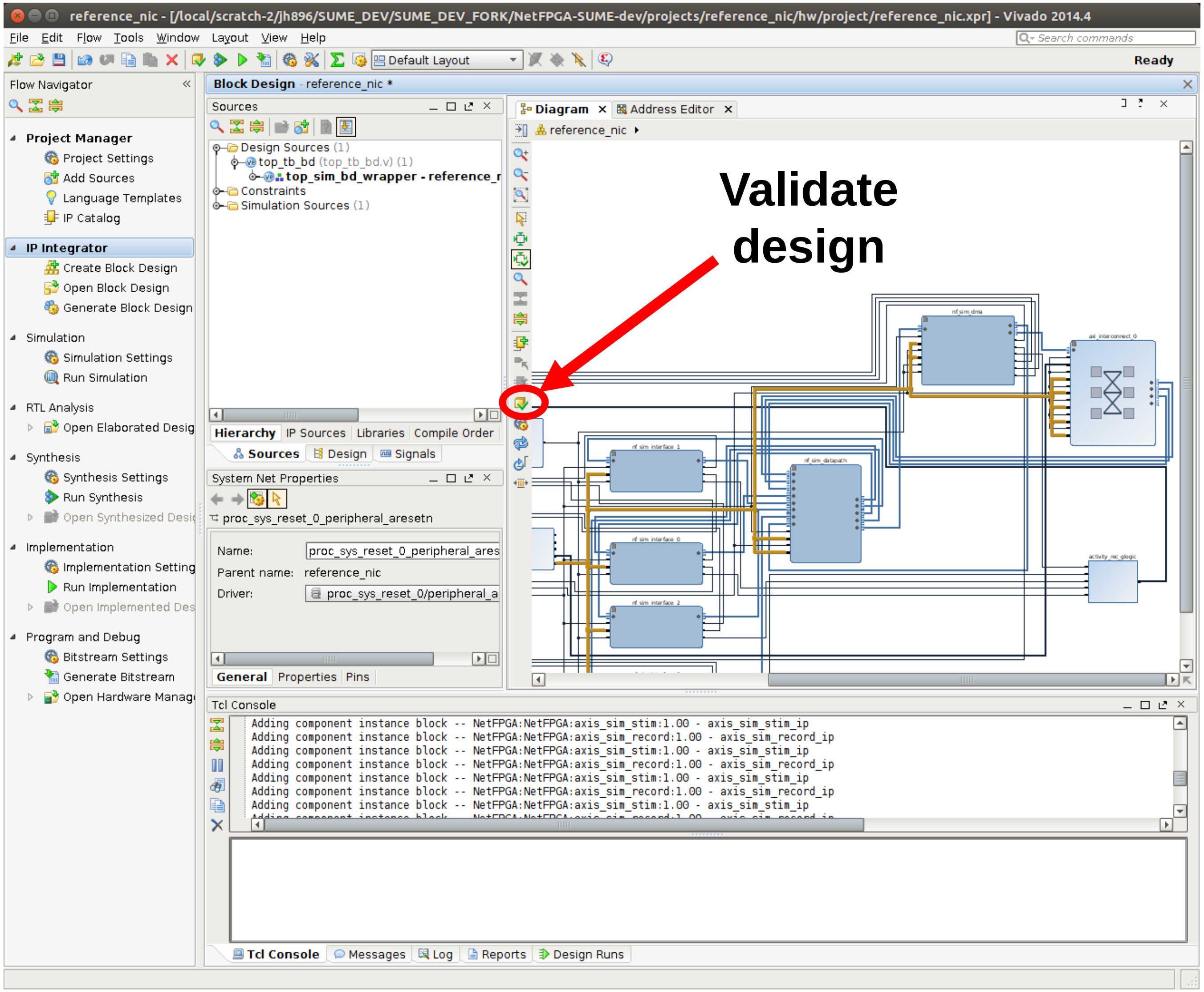
Range

Tcl Console

```
Adding component instance block -- xilinx.com:ip:util_vector_logic:2.0 - pcie_inverter_0
Adding component instance block -- xilinx.com:ip:util_vector_logic:2.0 - user_pcie_inverter_0
Adding component instance block -- xilinx.com:ip:pcie3_7x:3.0 - pcie3_7x_1
Adding component instance block -- NetFPGA:NetFPGA:nf_riffa_dma:1.0 - nf_riffa_dma_0
Adding component instance block -- xilinx.com:ip:axis_data_fifo:1.1 - axis_data_fifo_0
Adding component instance block -- xilinx.com:ip:axis_data_fifo:1.1 - axis_data_fifo_1
Adding component instance block -- xilinx.com:ip:axis_dwidth_converter:1.1 - axis_dwidth_converter_0
Adding component instance block -- xilinx.com:ip:axis_dwidth_converter:1.1 - axis_dwidth_converter_1
Successfully read diagram <reference_nic> from BD file </local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFPGA-SUME-dev/projects/referenc
open_bd_design: Time (s): cpu = 00:00:13 ; elapsed = 00:00:09 . Memory (MB): peak = 5897.762 ; gain = 17.516 ; free physical = 20373 ;
```

Type a Tcl command here

Project Integration – Block Design (7)



reference_nic - [/local/scratch-2/jh896/SUME_DEV/SUME_DEV_FORK/NetFPGA-SUME-dev/projects/reference_nic/hw/project/reference_nic.xpr] - Vivado 2014.4

Validate design

Name: proc_sys_reset_0_peripheral_ares

Parent name: reference_nic

Driver: proc_sys_reset_0/peripheral_a

```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
```

Summary to this Point

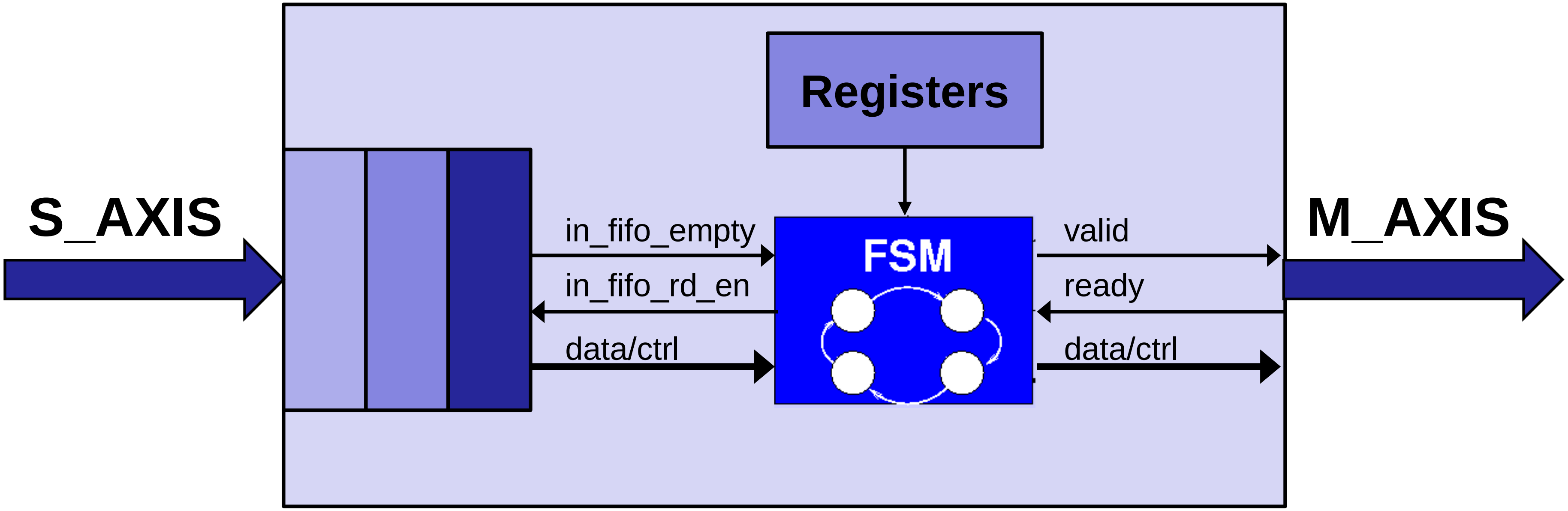
- **Created a new project**
- **Created a new core named crypto**
- **Wired the new core into the pipline**
  - After output_port_lookup
  - Before output_queues
- **Next we will write the Verilog code!**

Implementing the Crypto Module (1)

- **What do we want to encrypt?**
  - IP payload only
    - Plaintext IP header allows routing
    - Content is hidden
  - Encrypt bytes 35 onward
    - Bytes 1-14 – Ethernet header
    - Bytes 15-34 – IPv4 header (assume no options)
    - Remember AXI byte ordering
  - For simplicity, assume all packets are IPv4 without options

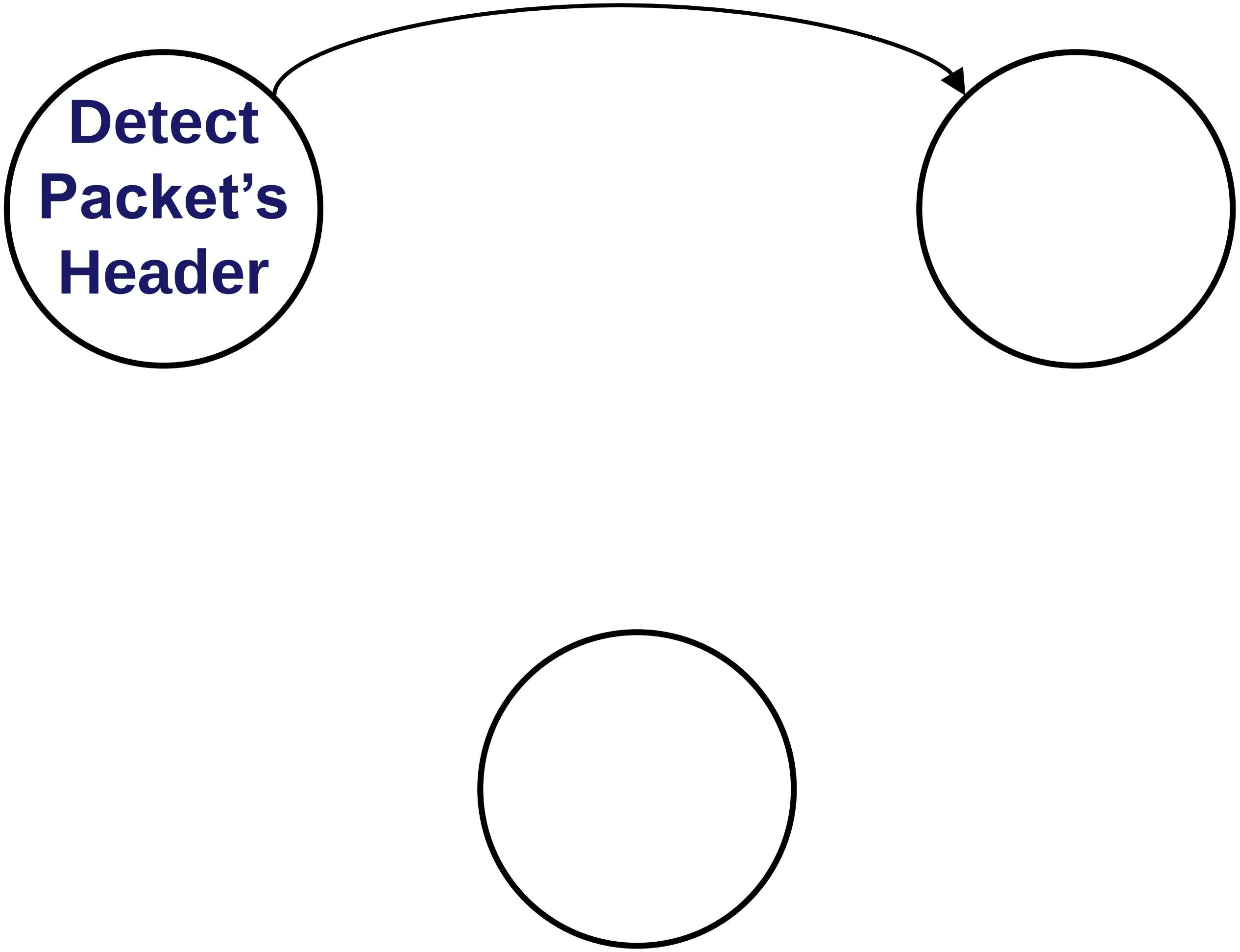
Implementing the Crypto Module (2)

- **State machine (shown next):**
  - – Module headers on each packet
  - – Datapath 256-bits wide
    - 34 / 32 is not an integer! ☹
- **Inside the crypto module**



Crypto Module State Diagram

Hint: We suggest 3 states



Implementing the Crypto Module (3)

Implement your state machine inside crypto.v

Suggested sequence of steps:

1. Set the key value
   - set the key = 32’hffffffff;
2. Write your state machine to modify the packet by XORing the key and the payload
   - Use eight copies of the key to create a 256-bit value to XOR with data words
3. Do not pay attention to the register infrastructure that will be explained later.

More Verilog: Assignments 1

- **Continuous assignments**
  - appear *outside* processes (`always @` blocks):

```
assign foo = baz & bar;
```

  - targets must be declared as `wire`s
  - always “happening” (*ie*, are concurrent)

More Verilog: Assignments 2

- **Non-blocking assignments**
  - appear *inside* processes (`always @` blocks)
  - use only in *sequential* (clocked) processes:

```
always @(posedge clk) begin
        a <= b;
        b <= a;
end
```

  - occur in next *delta* ('moment' in simulation time)
  - targets must be declared as `reg`s
  - never clock any process other than with a clock!

More Verilog: Assignments 3

- **Blocking assignments**
  - appear *inside* processes (`always @` blocks)
  - use only in *combinatorial* processes:
    - (combinatorial processes are much like continuous assignments)

```
always @(*) begin

        a = b;
        b = a;
end
```

  - occur one after the other (as in sequential langs like C)
  - targets must be declared as `reg`s – even though not a register
  - never use in sequential (clocked) processes!

More Verilog: Assignments 3

- Blocking assignments
  - appear *inside* processes (`always @` blocks)
  - use only in *combinatorial* processes:
    - (combinatorial processes are much like continuous assignments)

```
always @(*) begin
        tmp = a;
        a = b;
        b = tmp;
end
```

unlike non-blocking, have to use a temporary signal

  - occur one after the other (as in sequential langs like C)
  - targets must be declared as `reg`s – even though not a register
  - never use in sequential (clocked) processes!

(hints)

- **Never assign one signal from two processes:**

```
always @(posedge clk) begin
    foo <= bar;
end

always @(posedge clk) begin
    foo <= quux;
end
```

(hints)

- **In combinatorial processes:**
  - take great care to assign in all possible cases

```
always @(*) begin
    if (cond) begin
        foo = bar;
    end
end
```

  - (latches ‹as opposed to flip-flops› are bad for timing closure)

(hints)

- **In combinatorial processes:**
  - take great care to assign in all possible cases

```
always @(*) begin
     if (cond) begin
          foo = bar;
     else
          foo = quux;
     end
end
```

(hints)

- **In combinatorial processes:**
  - (or assign a default)

```
always @(*) begin
    foo = quux;

    if (cond) begin
        foo = bar;
    end
end
```

Section VIII: Simulation and Debug

Testing: Simulation

- **Simulation allows testing without requiring lengthy synthesis process**

- **NetFPGA simulation environment allows:**
  - Send/receive packets
    - Physical ports and CPU
  - Read/write registers
  - Verify results

- **Simulations run in xSim**

- **We provides an unified infrastructure for both HW and simulation tests**

Testing: Simulation

- **We will simulate the "crypto_switch" design under the "simulation framework"**
- **We will show you how to**
  - create simple packets using scapy
  - transmit and reconcile packets sent over 10G Ethernet and PCIe interfaces
  - the code can be found in the "test" directory inside the crypto_switch project

Testing: Simulation(2)

Run a simulation to verify changes:

1. make sure “NF_DESIGN_DIR” variable in the tools/settings.sh file located in ~/NetFPGA-SUME-alpha points to the crypto_switch project.

2. source ~/NetFPGA-SUME-alpha/tools/settings.sh (export NF_DESIGN_DIR=~/NetFPGA-SUME-alpha/projects/crypto_switch)

3. make –C $NF_DESIGN_DIR/hw reg

4. cd ~/NetFPGA-SUME-alpha/tools/scripts

5. ./nf_test.py sim --major crypto –minor test
   - Or ./nf_test.py sim --major crypto –major test --gui (if you want to run the gui

Now we can simulate the crypto functionality

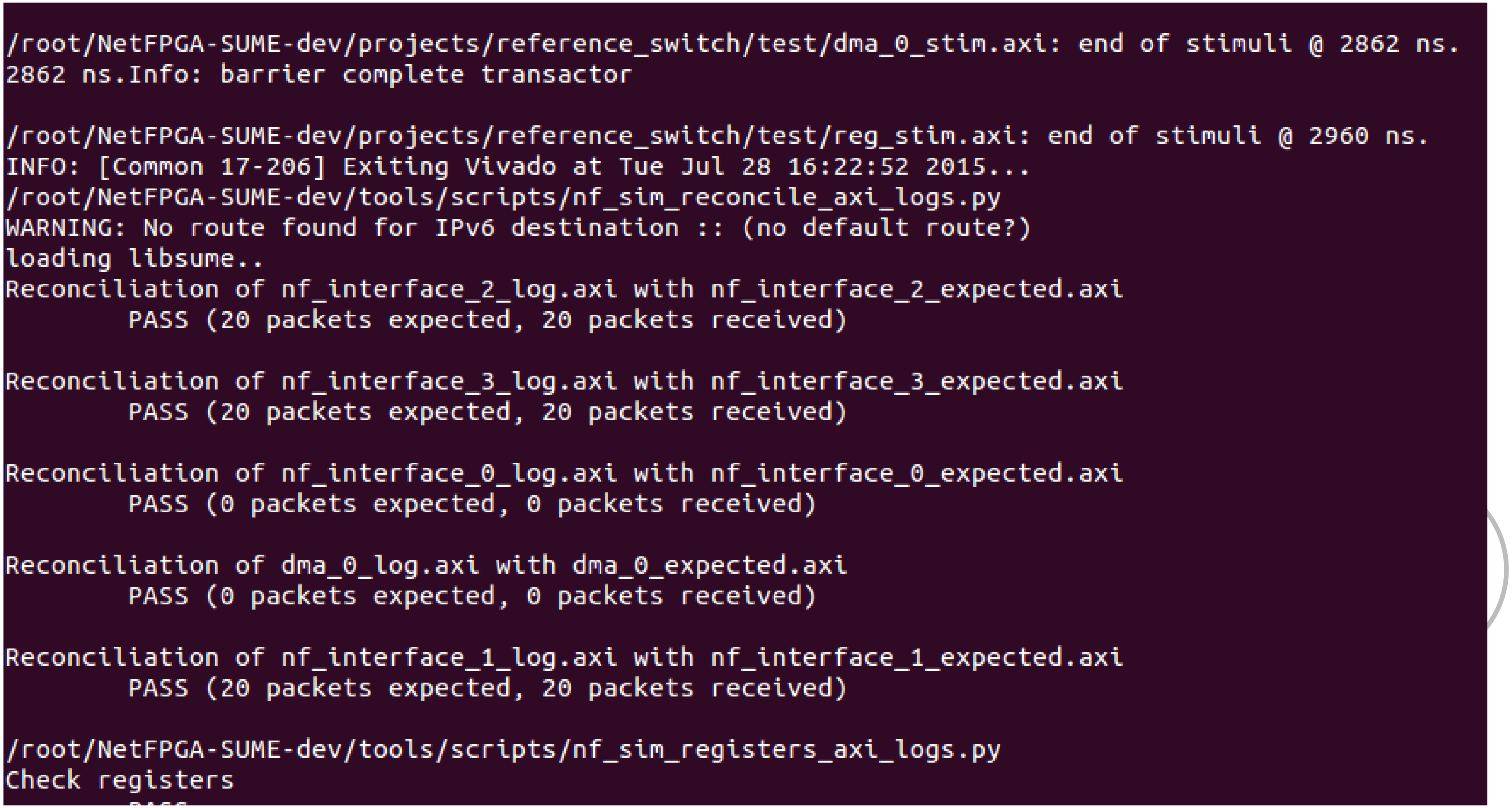
Crypto Switch simulation

```
cd $NF_DESIGN_DIR/test/both_crypto_test
vim run.py
```

- The “**isHW**” statement enables the HW test (we will look into it tomorrow)
- Let’s focus on the “**else**” part of the statement
- **make_IP_pkt** fuction creates the IP packet that will be used as stimuli
- **pkt.tuser_sport** is used to set up the correct source port of the packet
- **encrypt_pkt** encrypts the packet
- **pkt.time** selects the time the packet is supposed to be sent
- **nftest_send_phy/dma** are used to send a packet to a given interface
- **nftest_expected_phy/dma** are used to expect a packet in a given interface
- **nftest_barrier** is used to block the simulation till the previous statement has been completed (e.g., send_pkts -> barrier -> send_more_pkts)

4

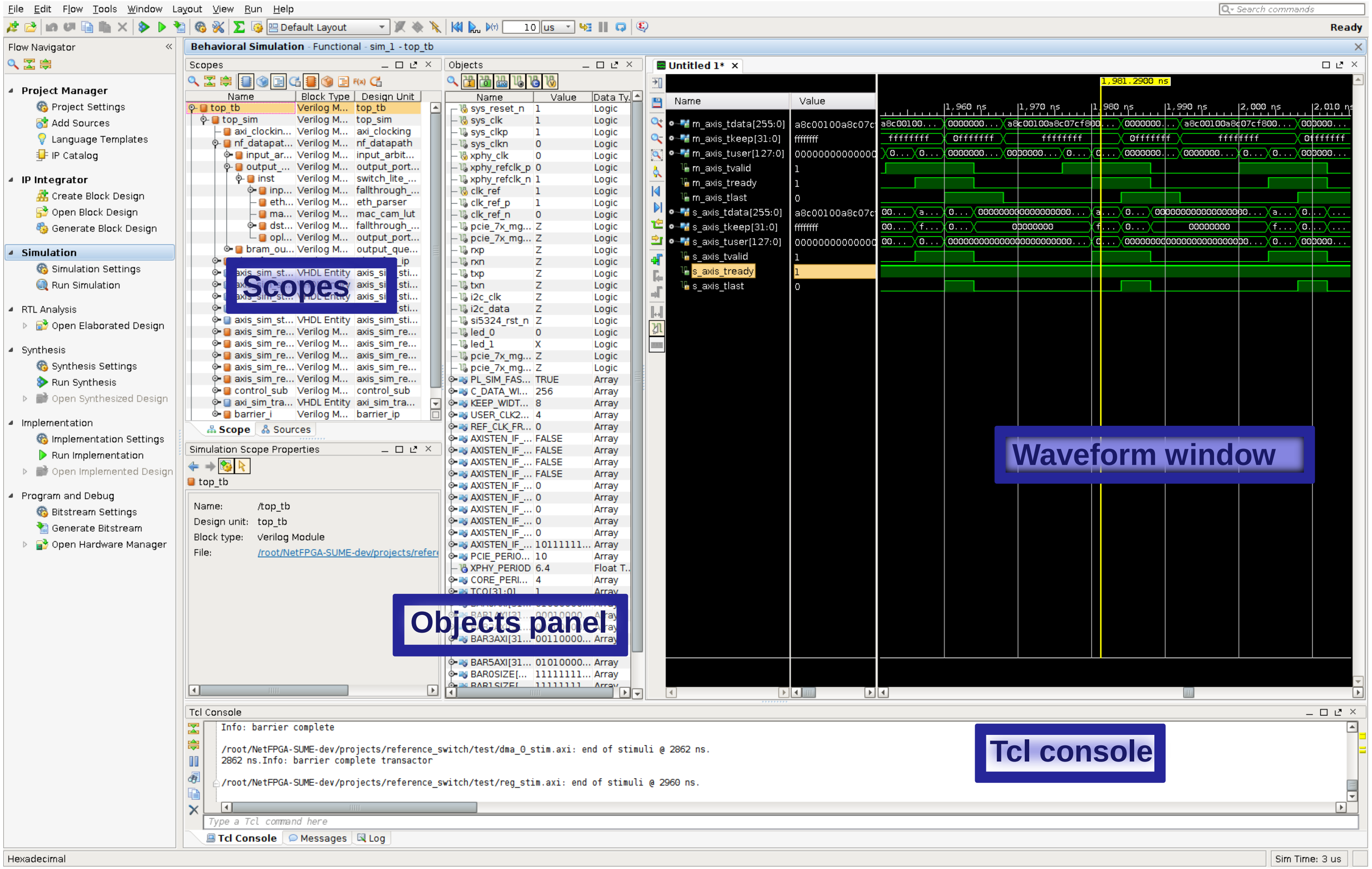
The results are in…



```
/root/NetFPGA-SUME-dev/projects/reference_switch/test/dma_0_stim.axi: end of stimuli @ 2862 ns.
2862 ns.Info: barrier complete transactor

/root/NetFPGA-SUME-dev/projects/reference_switch/test/reg_stim.axi: end of stimuli @ 2960 ns.
INFO: [Common 17-206] Exiting Vivado at Tue Jul 28 16:22:52 2015...
/root/NetFPGA-SUME-dev/tools/scripts/nf_sim_reconcile_axi_logs.py
WARNING: No route found for IPv6 destination :: (no default route?)
loading libsume..
Reconciliation of nf_interface_2_log.axi with nf_interface_2_expected.axi
        PASS (20 packets expected, 20 packets received)

Reconciliation of nf_interface_3_log.axi with nf_interface_3_expected.axi
        PASS (20 packets expected, 20 packets received)

Reconciliation of nf_interface_0_log.axi with nf_interface_0_expected.axi
        PASS (0 packets expected, 0 packets received)

Reconciliation of dma_0_log.axi with dma_0_expected.axi
        PASS (0 packets expected, 0 packets received)

Reconciliation of nf_interface_1_log.axi with nf_interface_1_expected.axi
        PASS (20 packets expected, 20 packets received)

/root/NetFPGA-SUME-dev/tools/scripts/nf_sim_registers_axi_logs.py
Check registers
        PASS
```

- **As expected, total of 20 packets are received on each interface**

Running simulation in xSim



Scopes

Name	Block Type	Design Unit
top_tb	Verilog M...	top_tb
top_sim	Verilog M...	top_sim
axi_clockin...	Verilog M...	axi_clocking
nf_datapat...	Verilog M...	nf_datapath
input_ar...	Verilog M...	input_arbit...
output_...	Verilog M...	output_port...
inst	Verilog M...	switch_lite_...
inp...	Verilog M...	fallthrough_...
eth...	Verilog M...	eth_parser
ma...	Verilog M...	mac_cam_lut
dst...	Verilog M...	fallthrough_...
opl...	Verilog M...	output_port...
bram_ou...	Verilog M...	output_que...
axis_sim_st...	VHDL Entity	axis_sim_sti...
axis_sim_st...	VHDL Entity	axis_sim_sti...
axis_sim_re...	Verilog M...	axis_sim_re...
axis_sim_re...	Verilog M...	axis_sim_re...
axis_sim_re...	Verilog M...	axis_sim_re...
axis_sim_re...	Verilog M...	axis_sim_re...
axis_sim_re...	Verilog M...	axis_sim_re...
control_sub	Verilog M...	control_sub
axi_sim_tra...	VHDL Entity	axi_sim_tra...
barrier_i	Verilog M...	barrier_ip

Simulation Scope Properties

top_tb

Name: /top_tb
Design unit: top_tb
Block type: Verilog Module
File: /root/NetFPGA-SUME-dev/projects/refere

Objects

Name	Value	Data Ty...
sys_reset_n	1	Logic
sys_clk	1	Logic
sys_clkp	1	Logic
sys_clkn	0	Logic
xphy_clk	0	Logic
xphy_refclk_p	0	Logic
xphy_refclk_n	1	Logic
clk_ref	1	Logic
clk_ref_p	1	Logic
clk_ref_n	0	Logic
pcie_7x_mg...	Z	Logic
pcie_7x_mg...	Z	Logic
rxp	Z	Logic
rxn	Z	Logic
txp	Z	Logic
txn	Z	Logic
i2c_clk	Z	Logic
i2c_data	Z	Logic
si5324_rst_n	Z	Logic
led_0	0	Logic
led_1	X	Logic
pcie_7x_mg...	Z	Logic
pcie_7x_mg...	Z	Logic
PL_SIM_FAS...	TRUE	Array
C_DATA_WI...	256	Array
KEEP_WIDT...	8	Array
USER_CLK2...	4	Array
REF_CLK_FR...	0	Array
AXISTEN_IF_...	FALSE	Array
AXISTEN_IF_...	FALSE	Array
AXISTEN_IF_...	FALSE	Array
AXISTEN_IF_...	FALSE	Array
AXISTEN_IF_...	0	Array
AXISTEN_IF_...	0	Array
AXISTEN_IF_...	0	Array
AXISTEN_IF_...	0	Array
AXISTEN_IF_...	0	Array
AXISTEN_IF_...	10111111...	Array
PCIE_PERIO...	10	Array
XPHY_PERIOD	6.4	Float T...
CORE_PERI...	4	Array
TCQ[31:0]	1	Array
BAR1AXI[31...	00010000...	Array
BAR3AXI[31...	00110000...	Array
BAR5AXI[31...	01010000...	Array
BAR0SIZE[...	11111111...	Array
BAR1SIZE[...	11111111...	Array

Waveform window

Name	Value
m_axis_tdata[255:0]	a8c00100a8c07c
m_axis_tkeep[31:0]	ffffffff
m_axis_tuser[127:0]	0000000000000
m_axis_tvalid	1
m_axis_tready	1
m_axis_tlast	0
s_axis_tdata[255:0]	a8c00100a8c07c
s_axis_tkeep[31:0]	ffffffff
s_axis_tuser[127:0]	0000000000000
s_axis_tvalid	1
s_axis_tready	1
s_axis_tlast	0

Cursor: 1,981.2900 ns; time scale 1,960 ns, 1,970 ns, 1,980 ns, 1,990 ns, 2,000 ns, 2,010 ns

Tcl Console

```
Info: barrier complete

/root/NetFPGA-SUME-dev/projects/reference_switch/test/dma_0_stim.axi: end of stimuli @ 2862 ns.
2862 ns.Info: barrier complete transactor

/root/NetFPGA-SUME-dev/projects/reference_switch/test/reg_stim.axi: end of stimuli @ 2960 ns.
```

Type a Tcl command here

Running simulation in xSim (2)

- Scopes panel: displays process and instance hierarchy
- Objects panel: displays simulation objects associated with the instance selected in the instance panel
- Waveform window: displays wave configuration consisting of signals and busses
- Tcl console: displays simulator generated messages and can executes Tcl commands

Simulation gone wild

When "./nf_test.py sim ….. "
1
source /opt/Xilinx/Vivado/2014.4/settings64.sh

2
Edit and source NetFPGA-SUME-alpha/tools/settings.sh

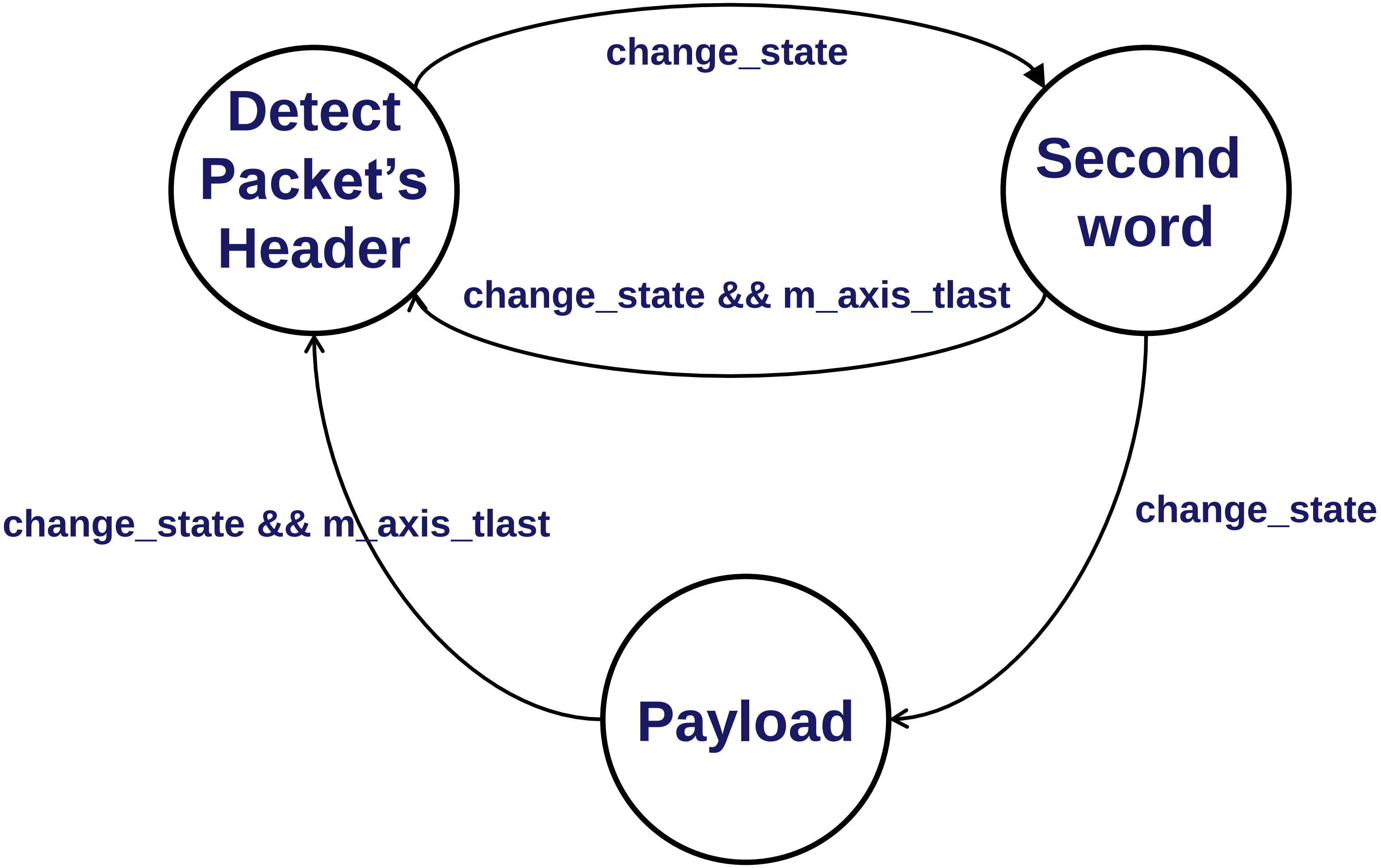
3
Run "make core" under projects/crypto_switch/hw/

4
Check that crypto_switch.tcl, crypto_switch_sim.tcl, export_registers.tcl are all up to date with your changes

5
if sim finishes but complains that each test passes 10 packets but all tests FAIL – this means your static key is different between your code and your run.py file
check the log

Crypto Module State Diagram: Solution

change_state = m_axis_tvalid && m_axis_tready



it is time for the first synthesis!!!

Synthesis

- **To synthesize your project:**

```
cd ~/$NF_DESIGN_DIR/
make clean; make
```

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.