Tutorial

STEPHEN IBANEZ, NOA ZILBERMAN, ROBERT HALSTEAD,
SEAN CHOI, GIANNI ANTICHI

Outline

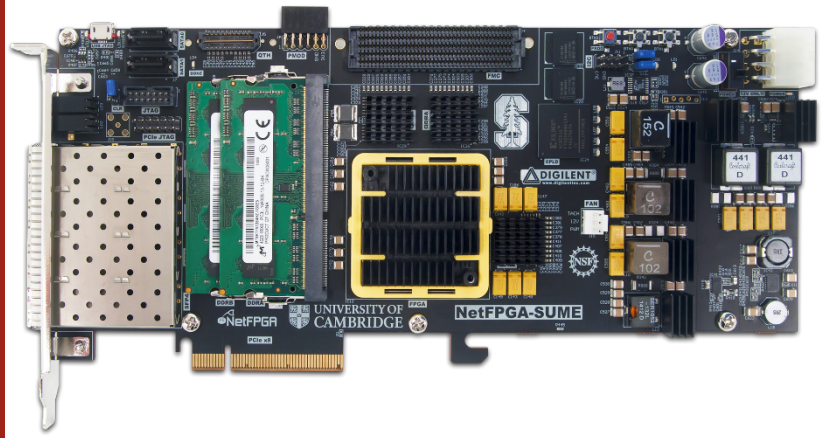
- NetFPGA Overview
- P4->NetFPGA Workflow Overview
- Tutorial Assignments

Overview

Section I: The NetFPGA platform

NetFPGA = Networked FPGA

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



=

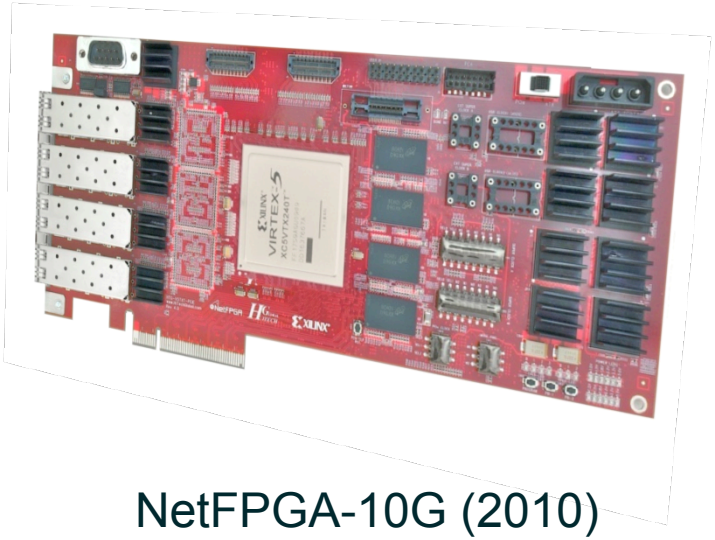
- Network Interface Card
- Hardware Accelerated Linux Router
- IPv4 Reference Router
- Traffic Generator
- Openflow Switch
- More Projects
- Add Your Project

NetFPGA XILINX

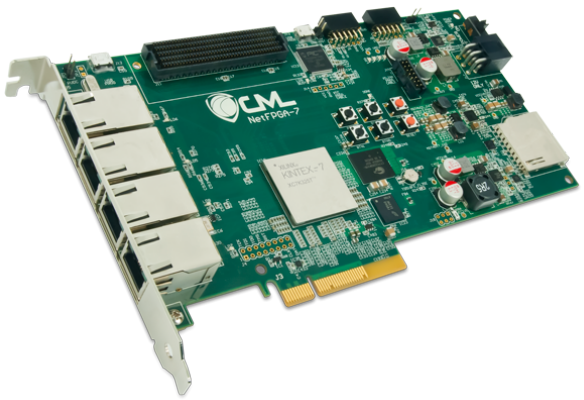
UNIVERSITY OF CAMBRIDGE Stanford University

NetFPGA Family of Boards

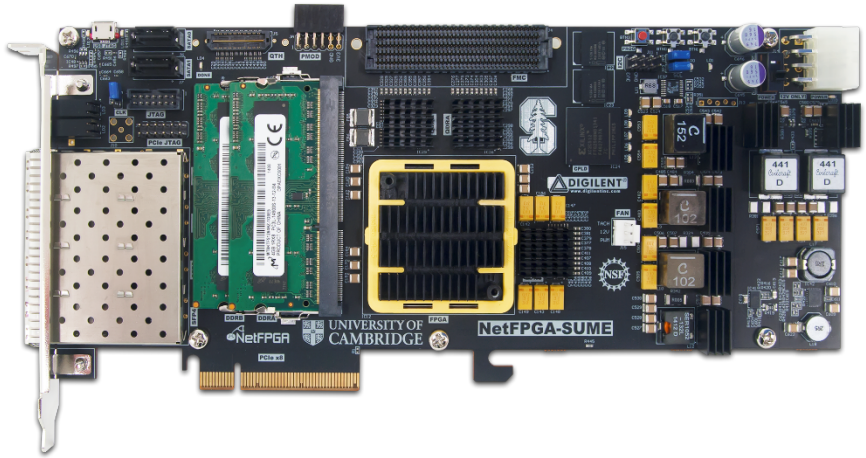
NetFPGA-1G (2006)



NetFPGA-10G (2010)



NetFPGA-1G-CML (2014)



NetFPGA SUME (2014)

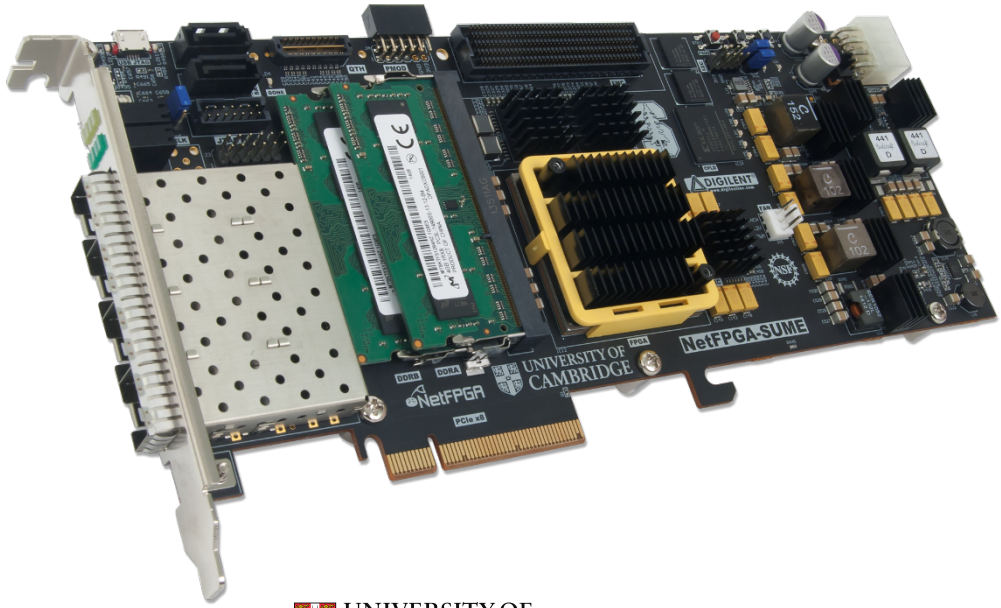
NetFPGA XILINX

UNIVERSITY OF CAMBRIDGE Stanford University

NetFPGA consists of…

Four elements:

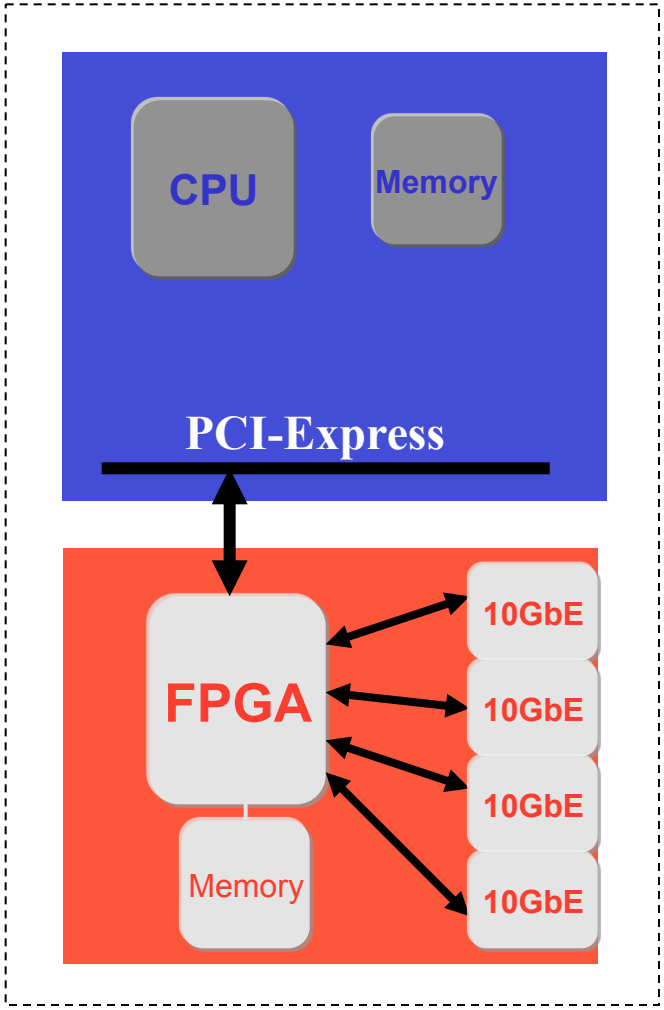
- NetFPGA board
- Tools + reference designs
- Contributed projects
- Community



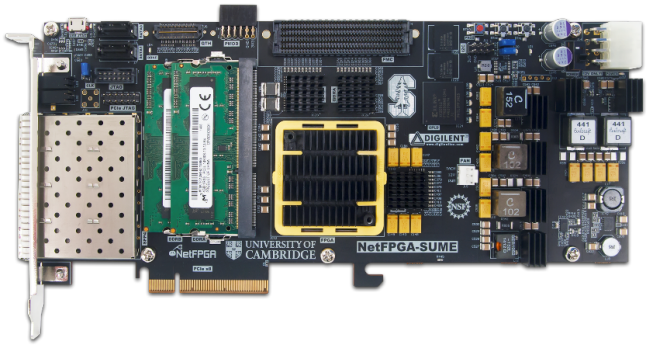
NetFPGA board

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 200 universities in over 47 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?

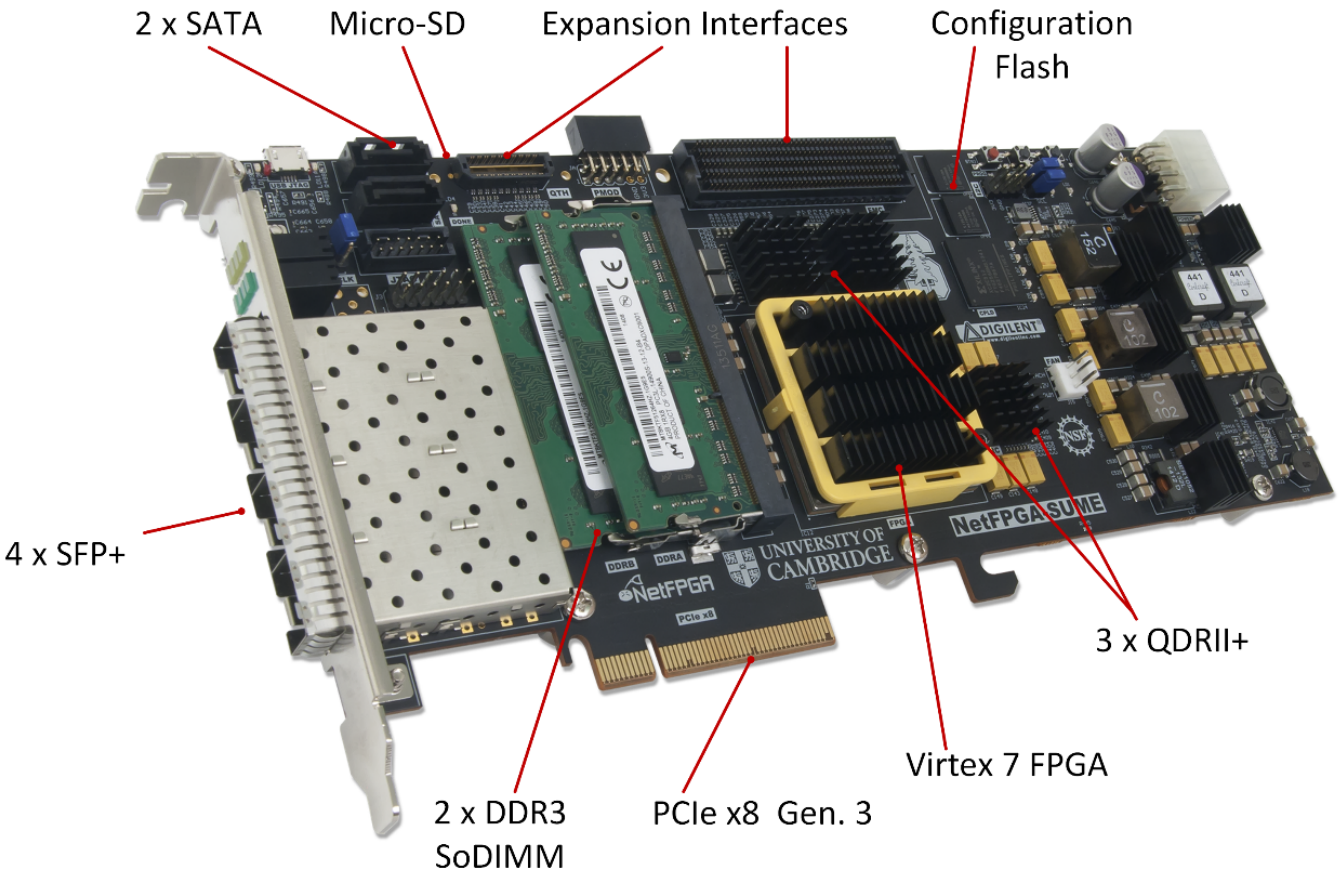
- To run the Router Kit
- To build modular reference designs
  - IPv4 router
  - 4-port NIC
  - Ethernet switch, …

Why do they use the NetFPGA?

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

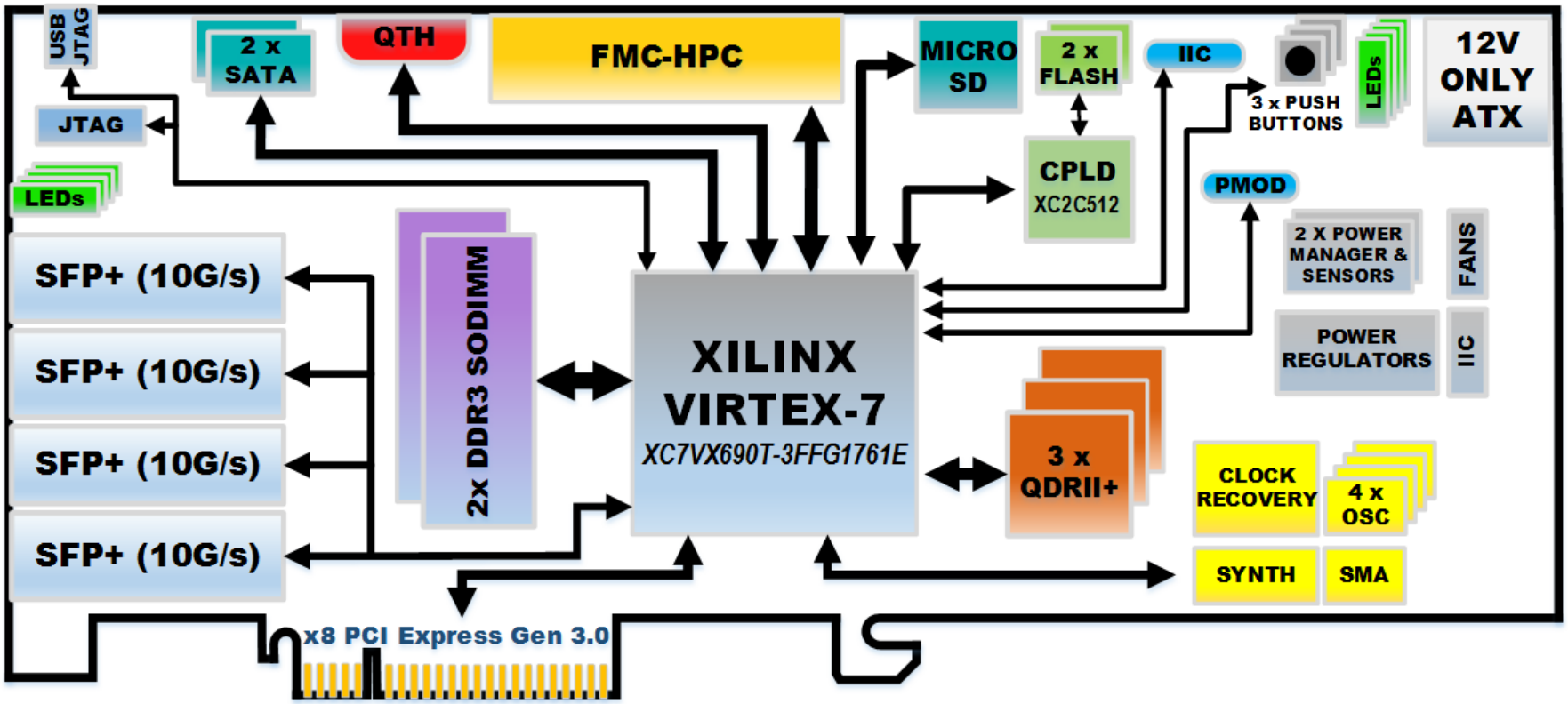
Section II: Hardware Overview

NetFPGA-SUME



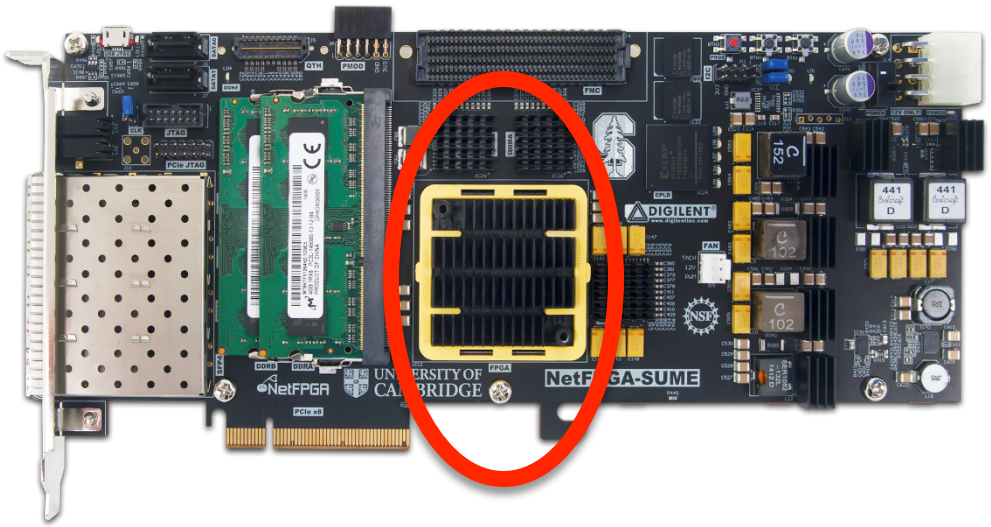
NetFPGA-SUME

High Level Block Diagram



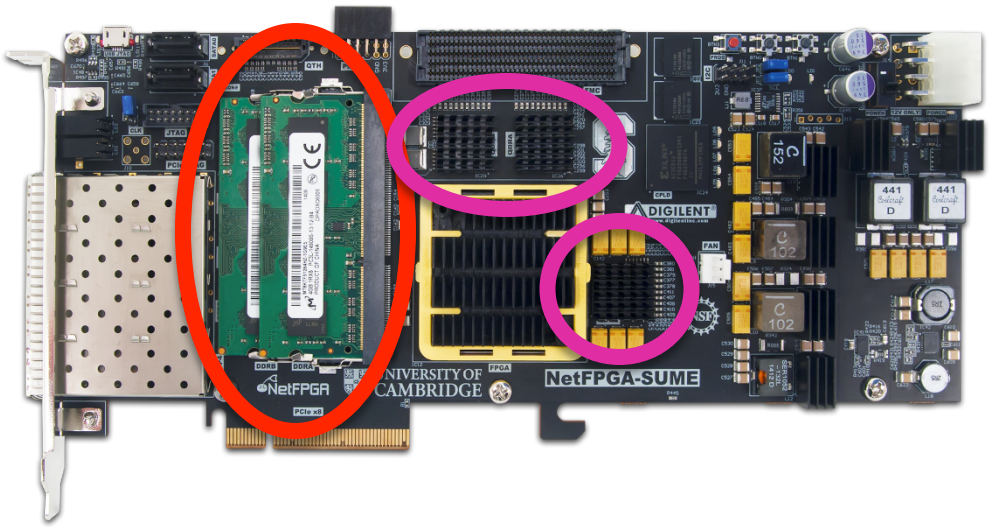
Xilinx Virtex 7 690T

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



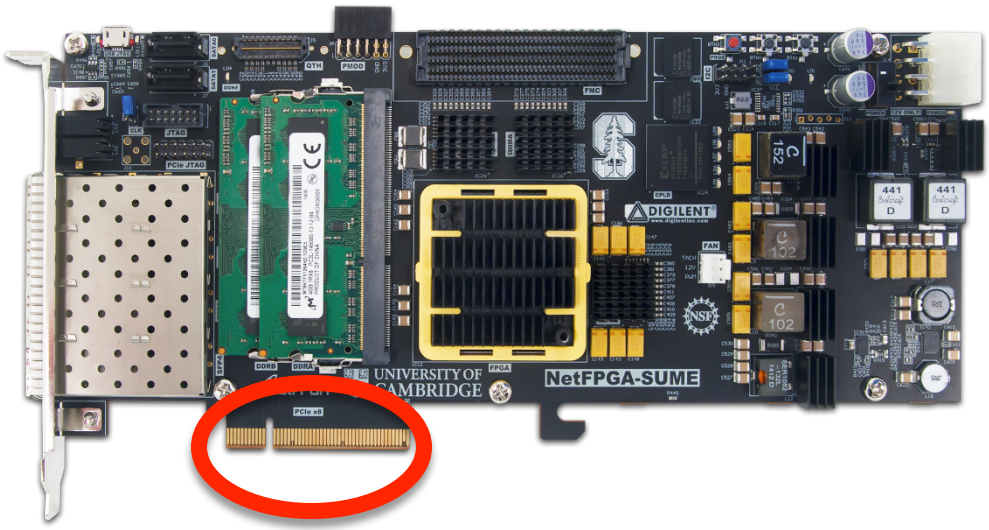
Memory Interfaces

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



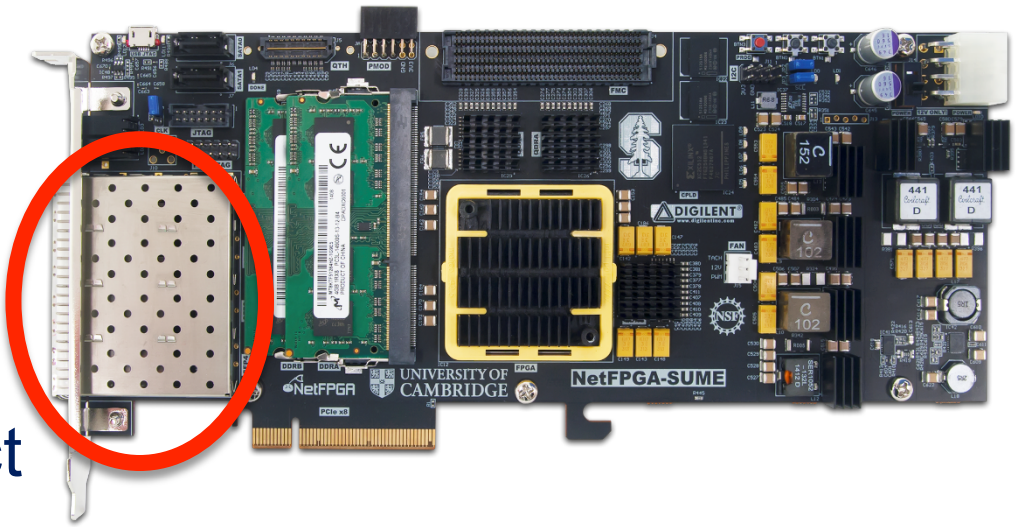
Host Interface

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



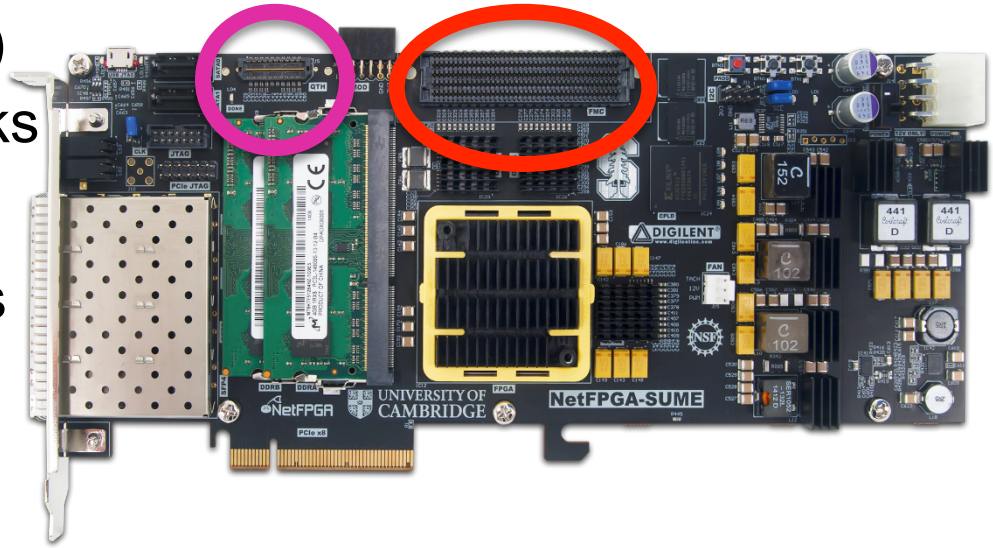
Front Panel Ports

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



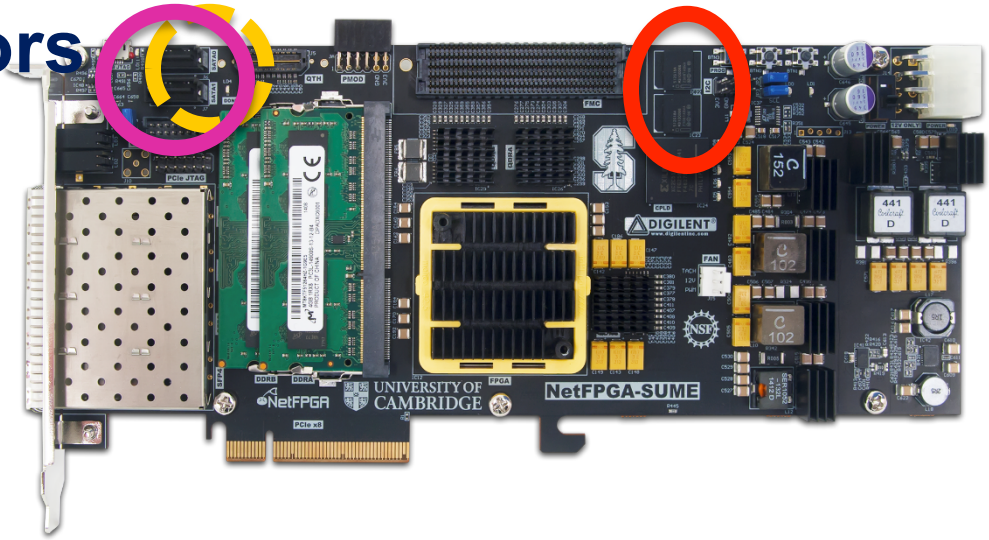
Expansion Interfaces

- **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

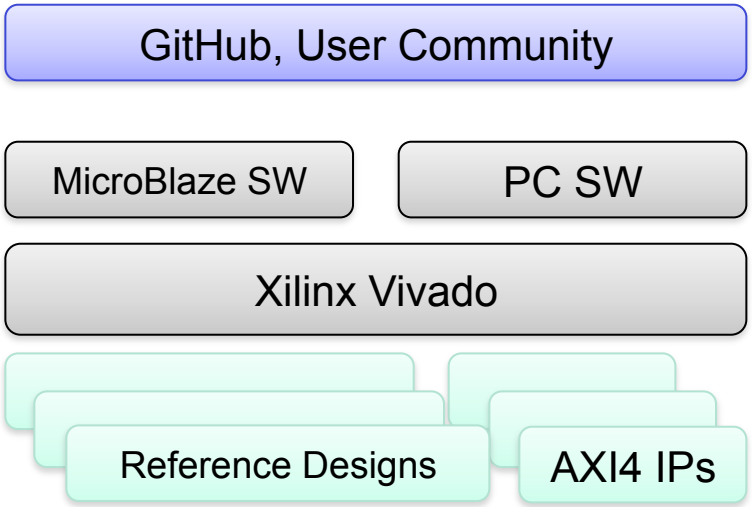


Storage

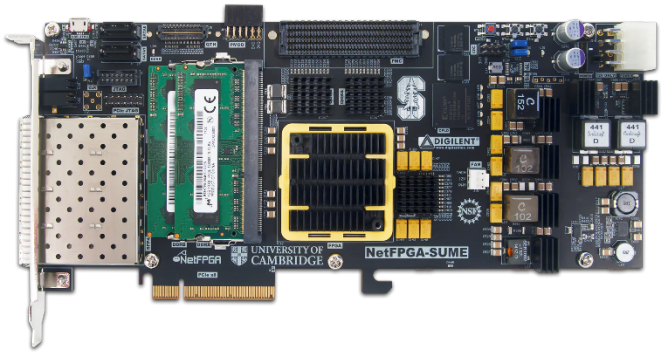
- 128MB FLASH
- 2 x SATA connectors
- Micro-SD slot
- Enable standalone operation



Beyond Hardware



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



Integrated Circuit Technology And Field Programmable Gate Arrays (FPGAs)

Integrated Circuit Technology

Full-custom Design

- Complementary Metal Oxide Semiconductor (CMOS)

Semi-custom ASIC Design

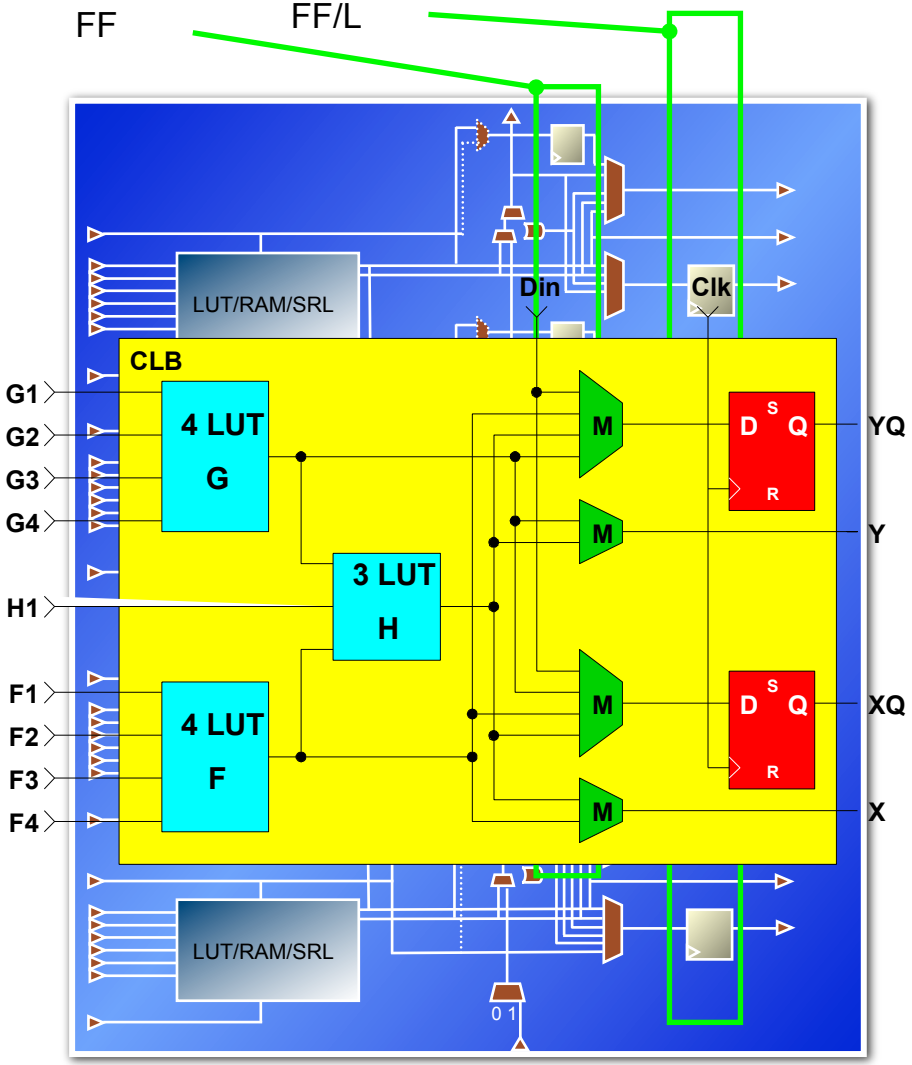
- Gate array
- Standard cell

Programmable Logic Device

- Programmable Array Logic
- Field Programmable Gate Arrays

Processors

Xilinx Virtex-7



- Logic Resources
- Memory resources
- Macro blocks
- Clocking resources
- Routing resources
- I/O resource
- ….

- A program is limited by the available resources

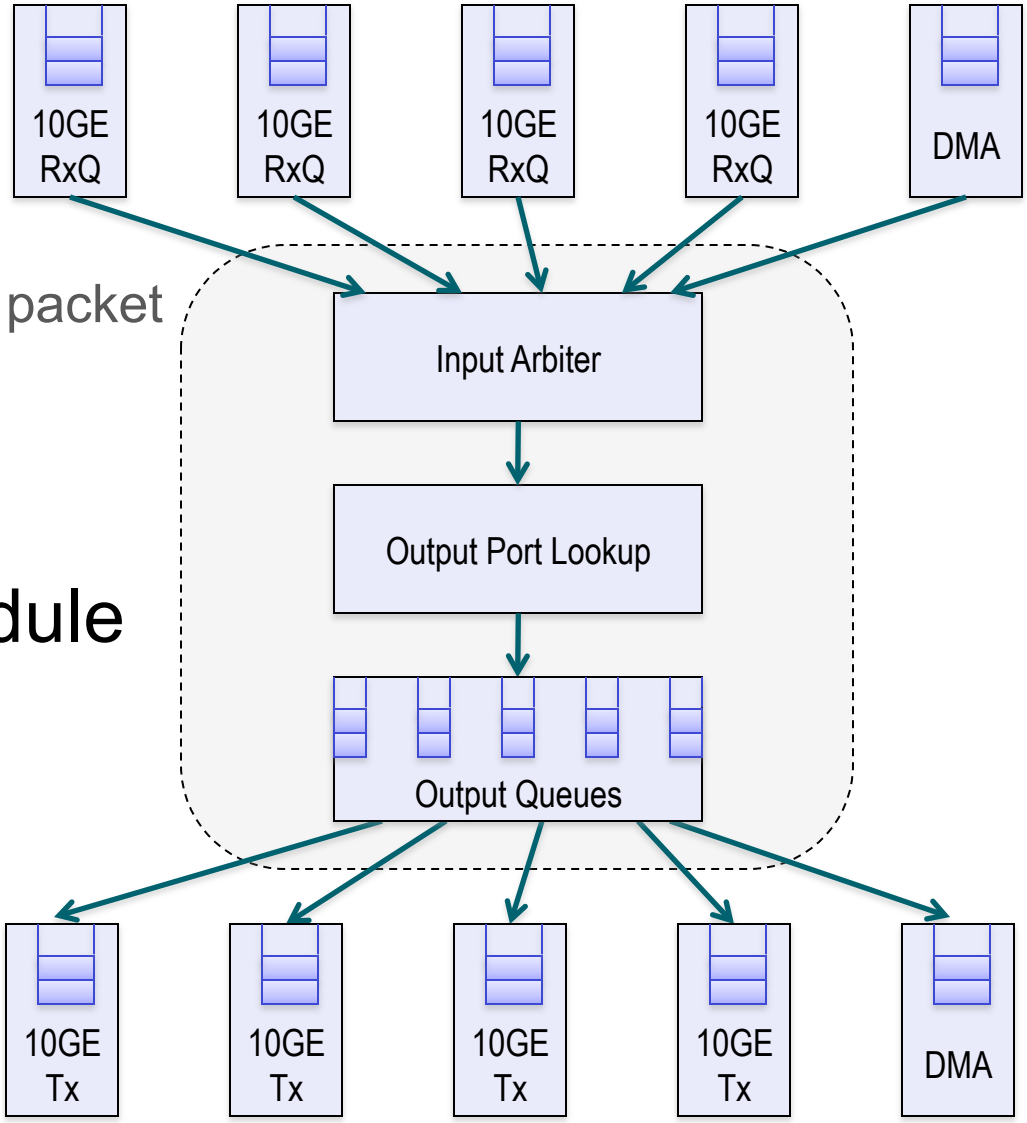
- A program keeps running:
  - Until reprogrammed
  - Until a power cycle

Diagram From: Xilinx, Inc.

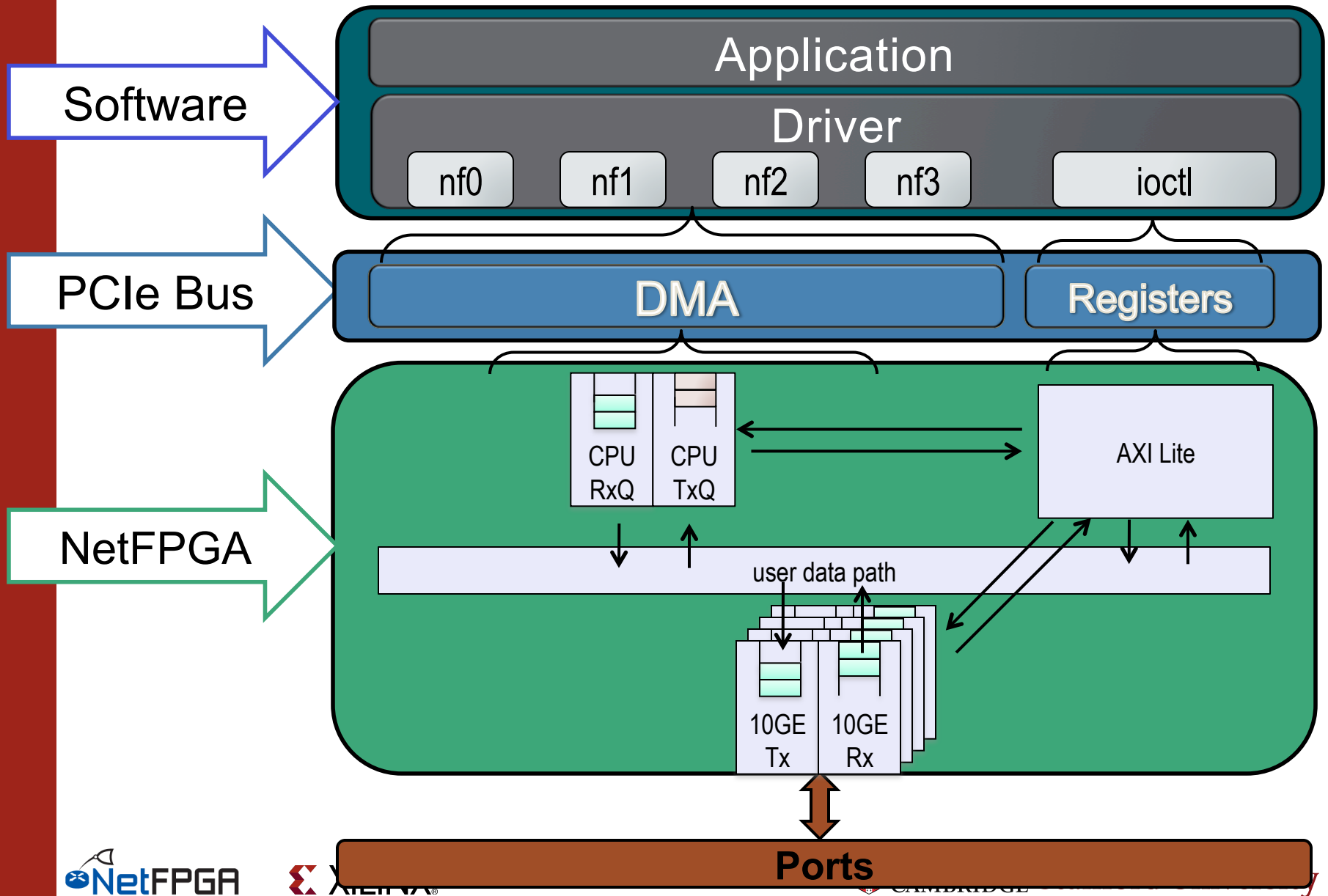
Section III: Life of a Packet

Reference Switch Pipeline

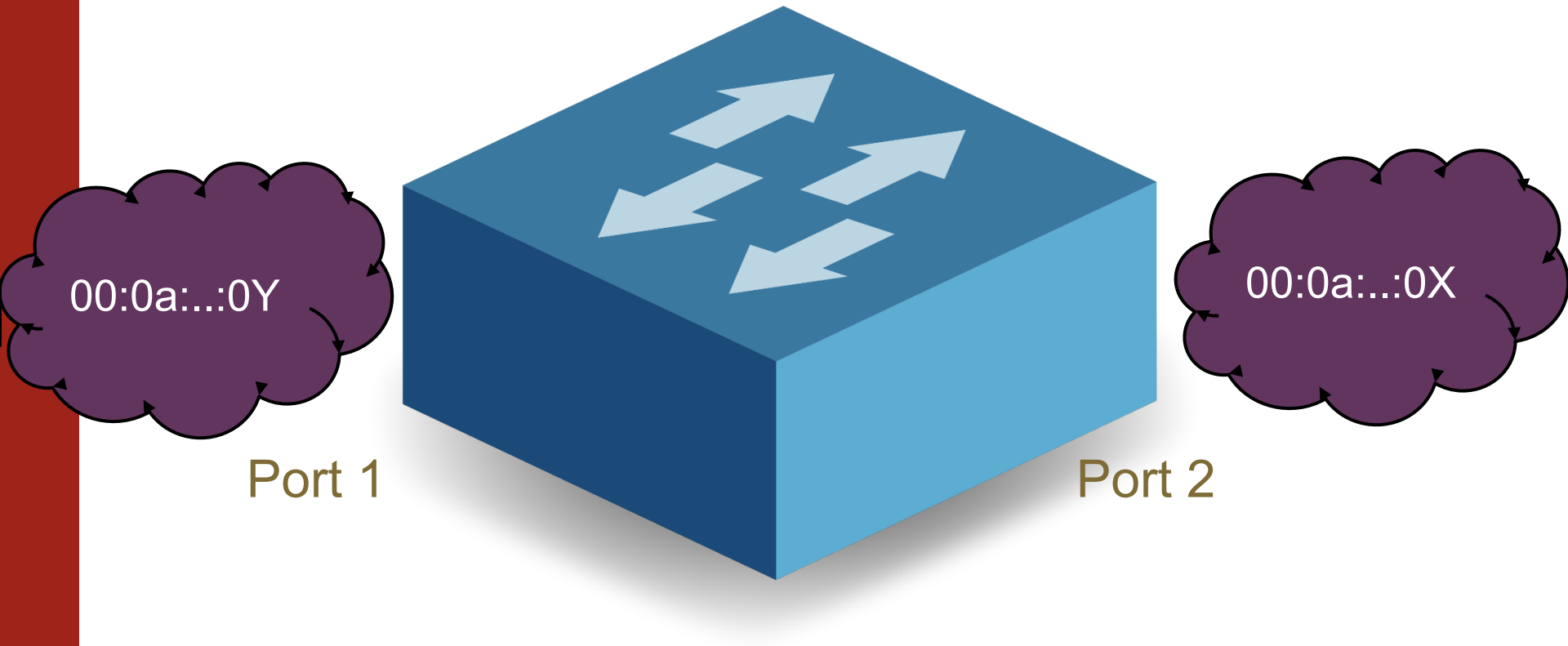
- Five stages
  - Input port
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output port
- Packet-based module interface
- Pluggable design



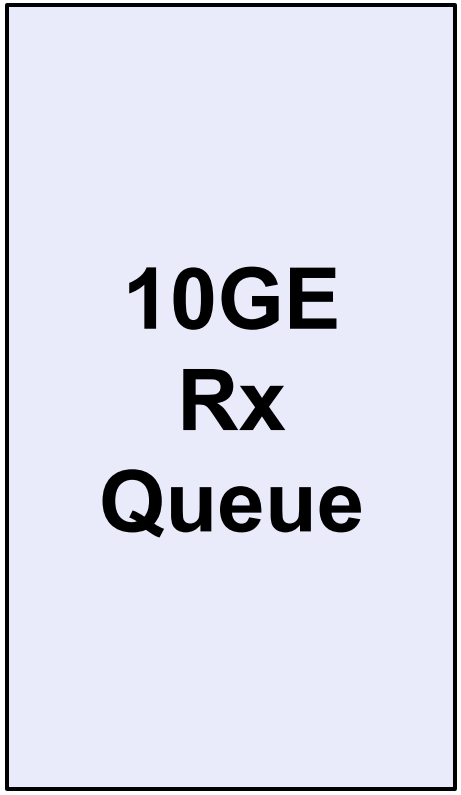
Full System Components



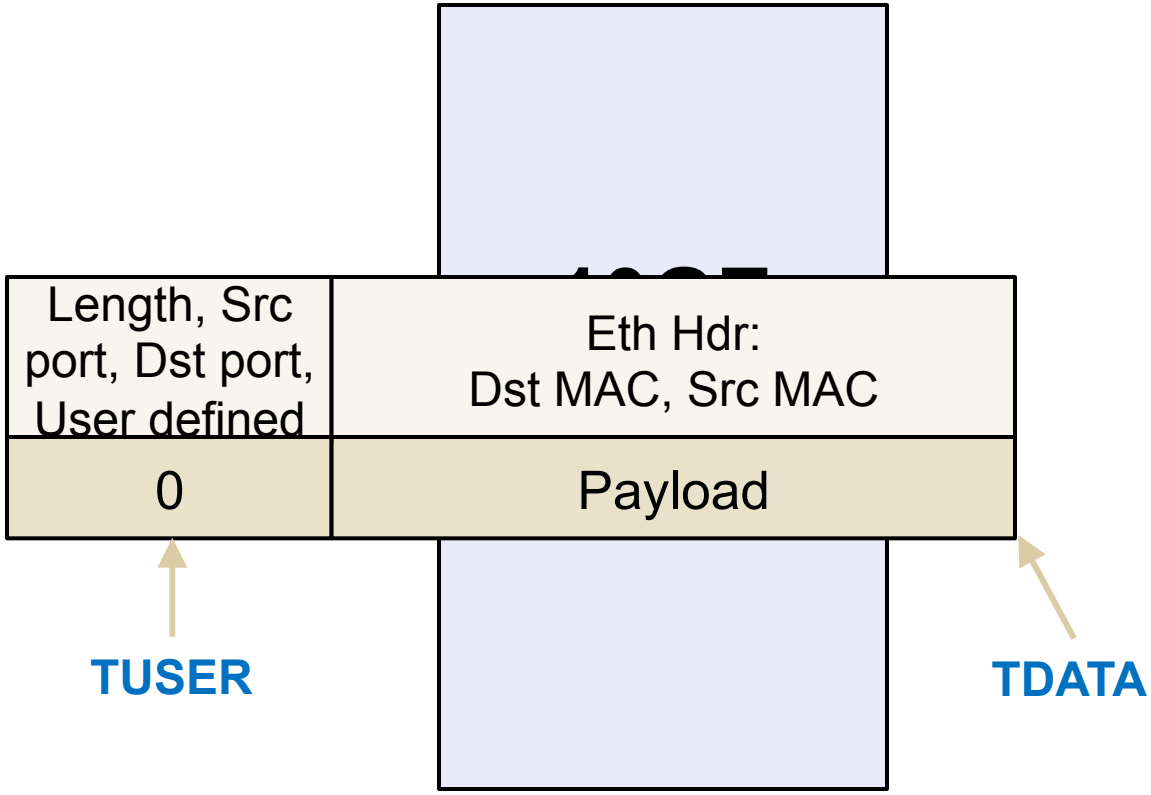
Life of a Packet through the Hardware



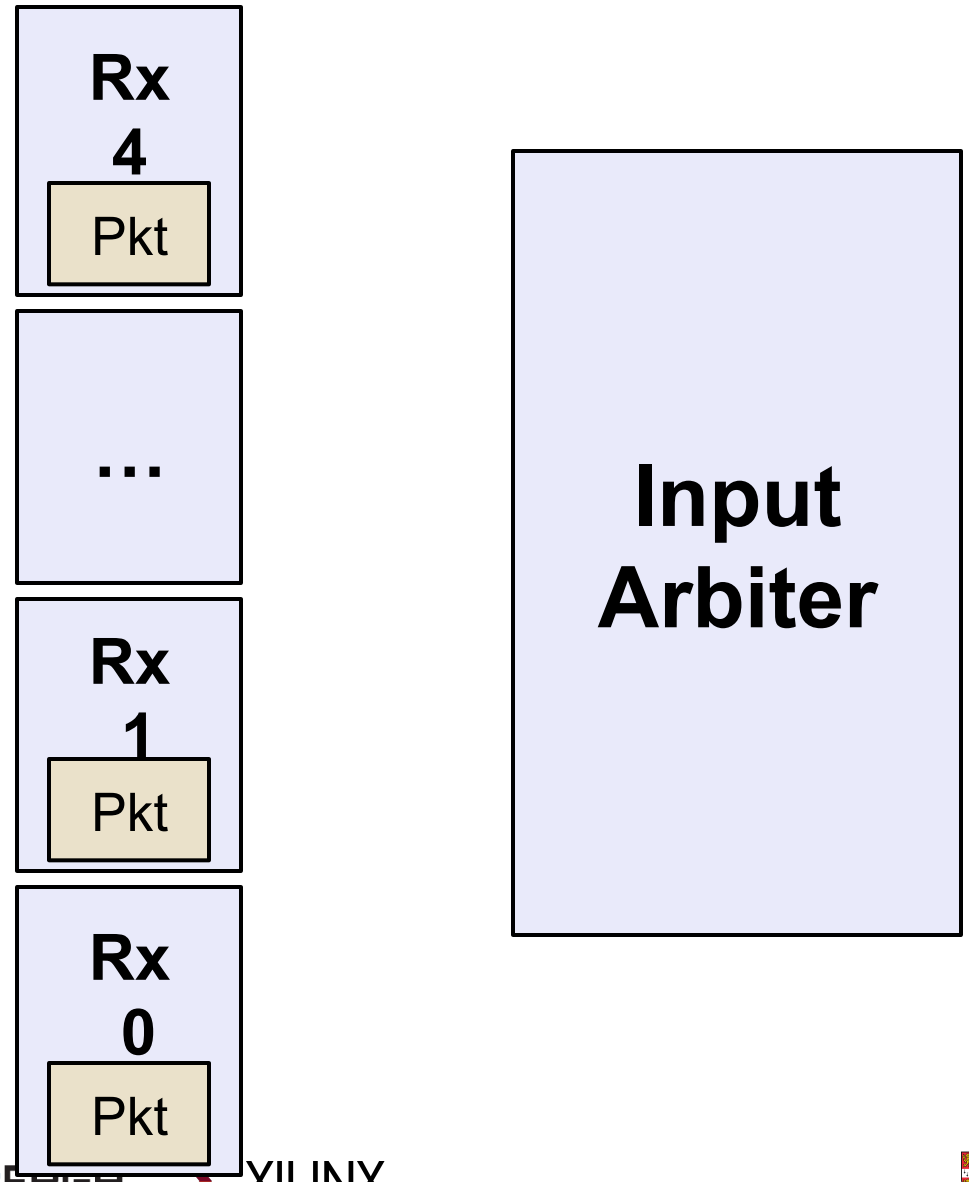
10GE Rx Queue



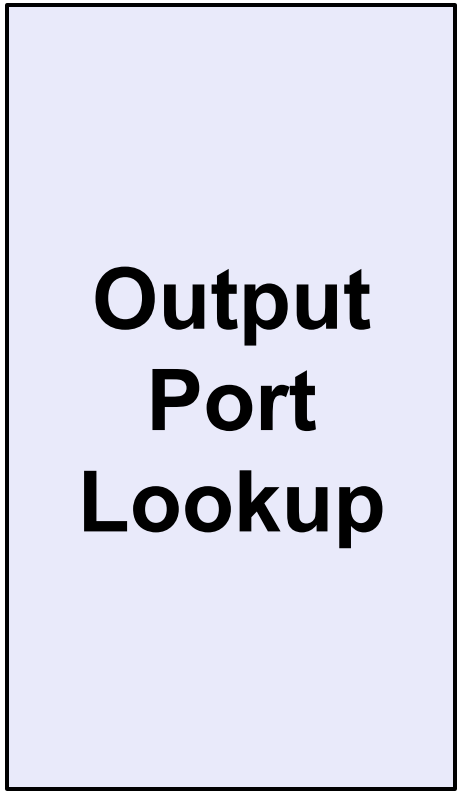
10GE Rx Queue



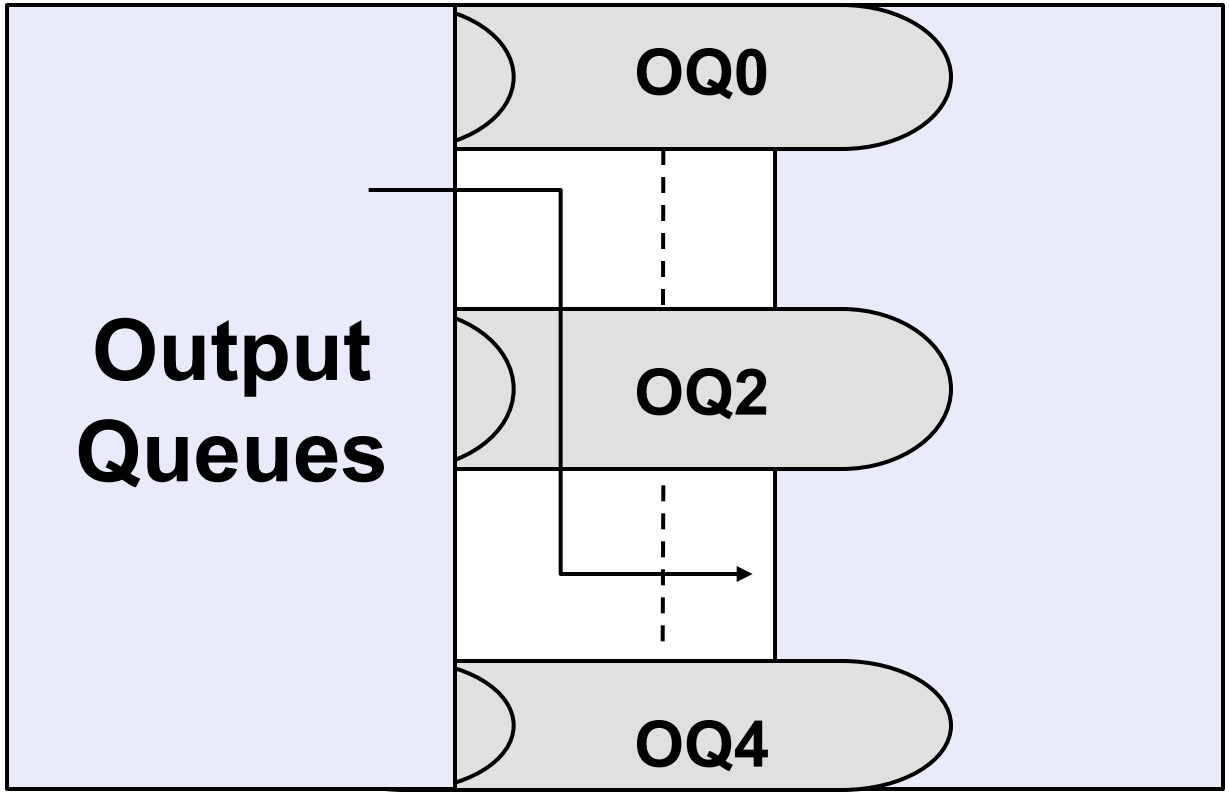
Input Arbiter



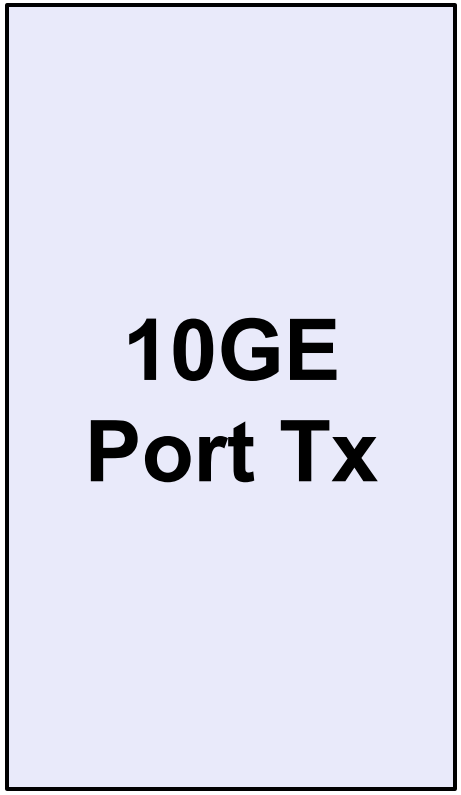
Output Port Lookup



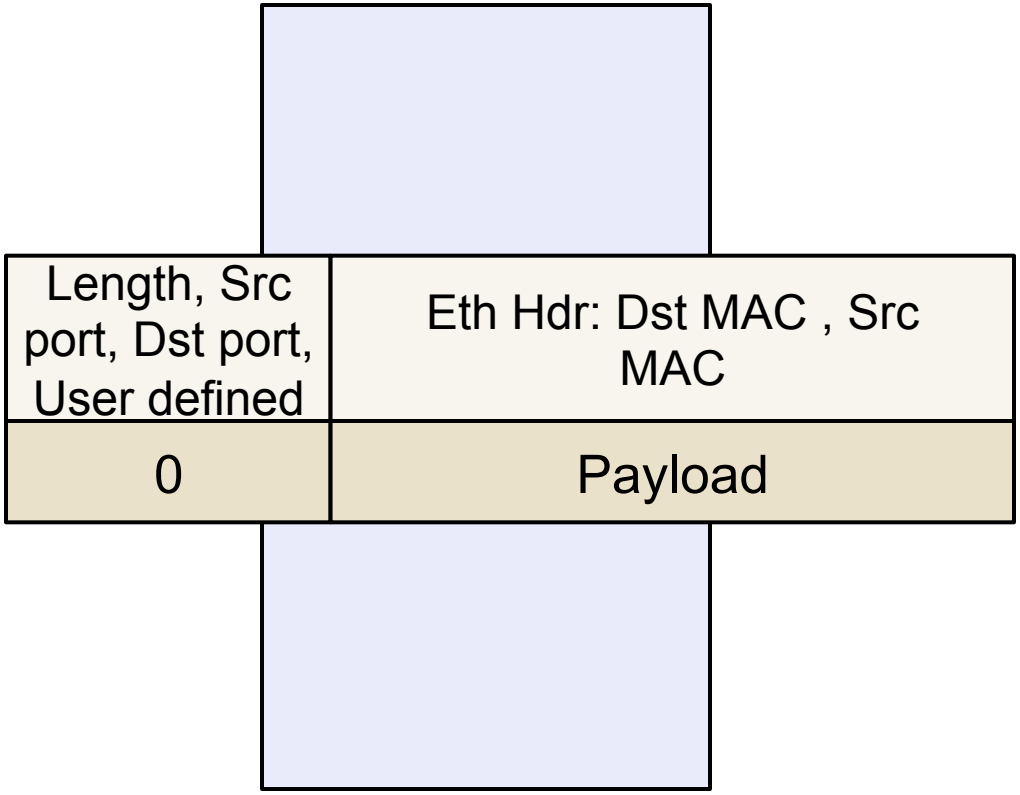
Output Queues



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);

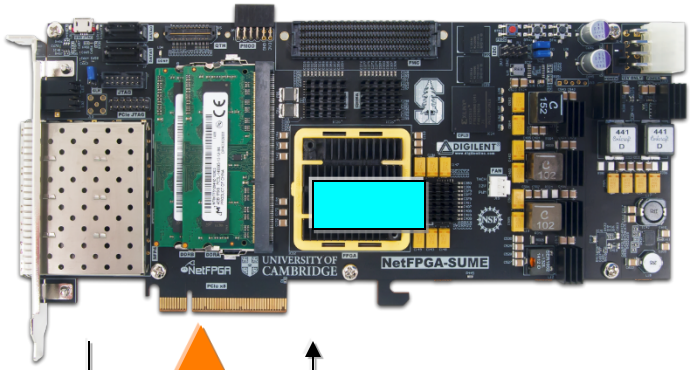
  eg:

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

NetFPGA-Host Interaction

NetFPGA to host packet transfer

1. Packet arrives – forwarding table sends to DMA queue



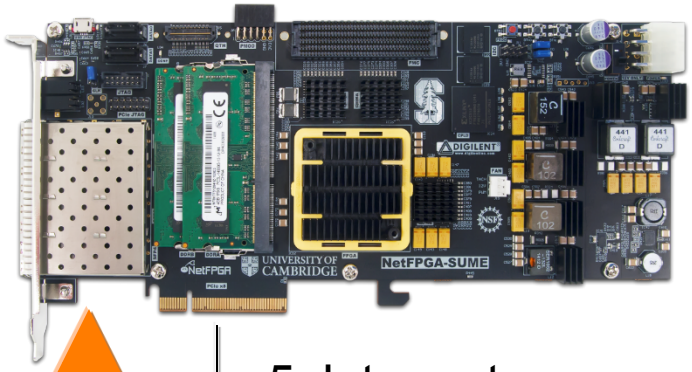
2. Interrupt notifies driver of packet arrival

PCIe Bus

3. Driver sets up and initiates DMA transfer

NetFPGA-Host Interaction

NetFPGA to host packet transfer (cont.)



4. NetFPGA transfers packet via DMA

PCIe Bus

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

PCIe Bus

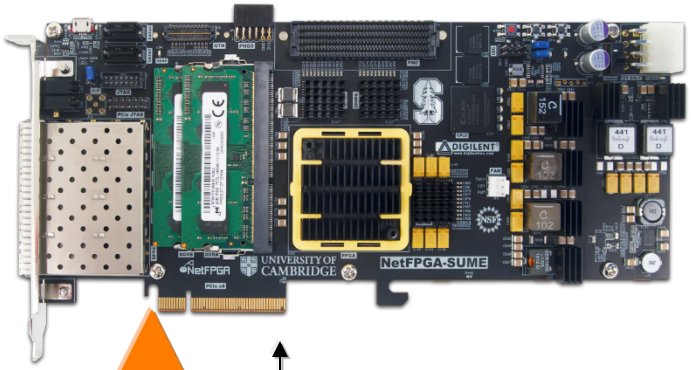
3. Interrupt signals completion of DMA

1. Software sends packet via network sockets

Packet delivered to driver

NetFPGA-Host Interaction

Register access



PCIe Bus

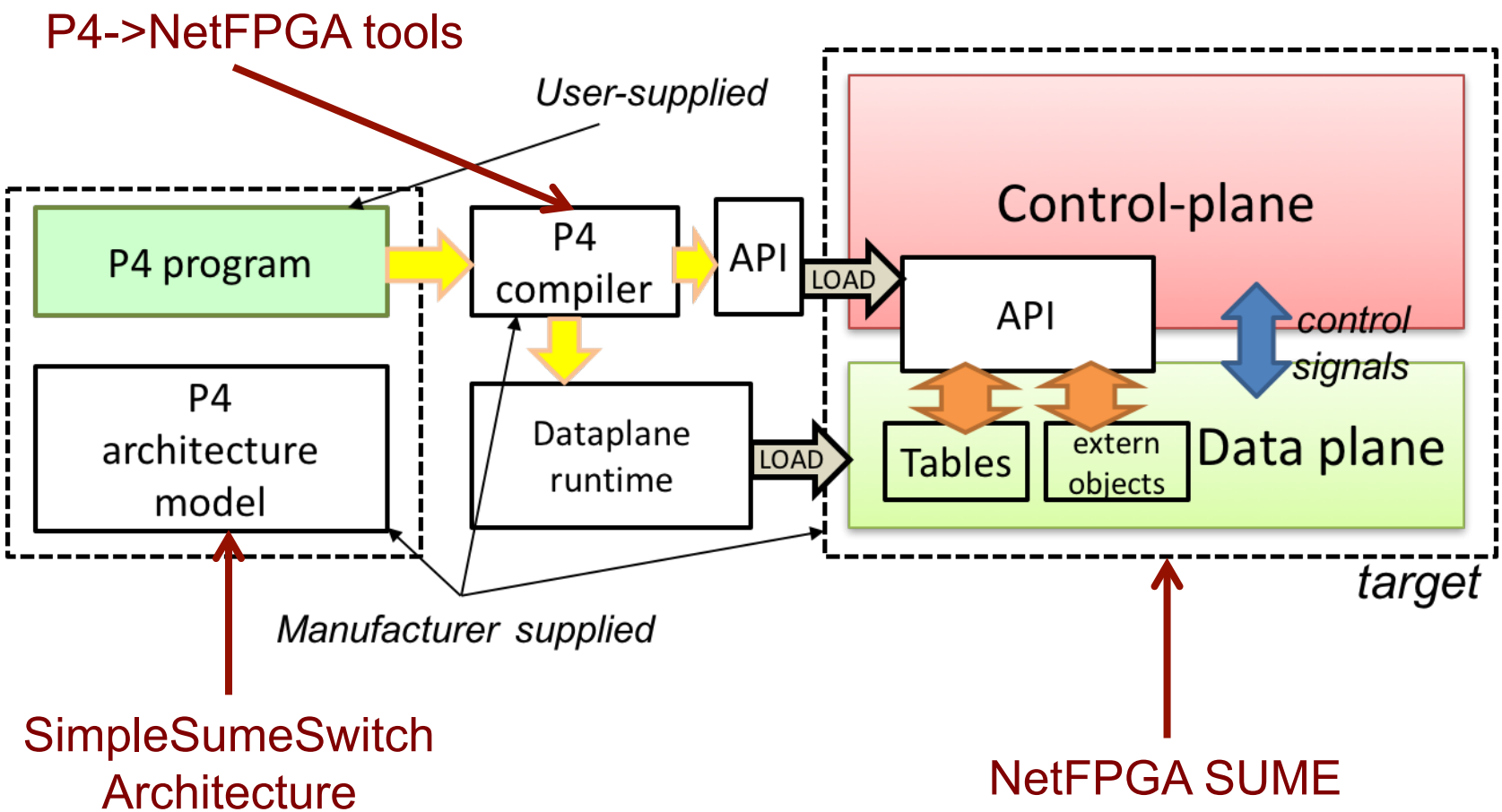
2. Driver performs PCIe memory read/write

1. Software makes ioctl call on network socket

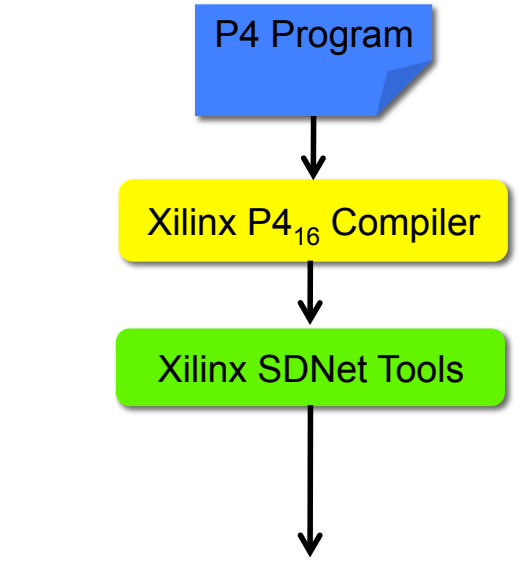
ioctl passed to driver

P4->NetFPGA Workflow Overview

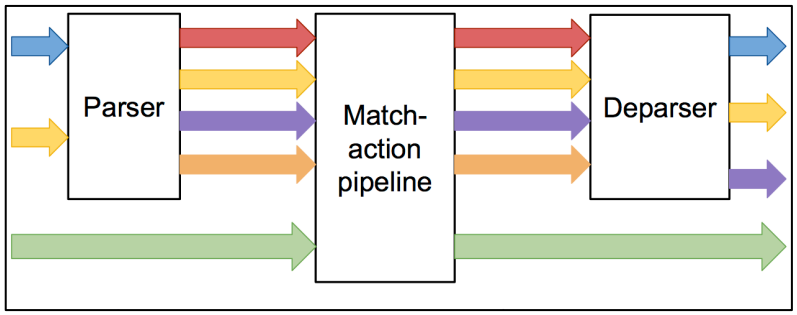
General Process for Programming a P4 Target



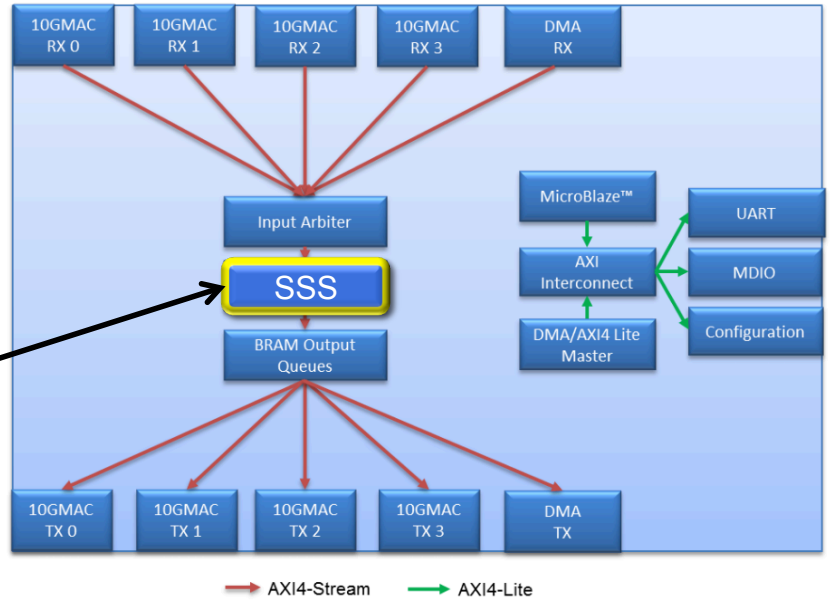
P4->NetFPGA Compilation Overview



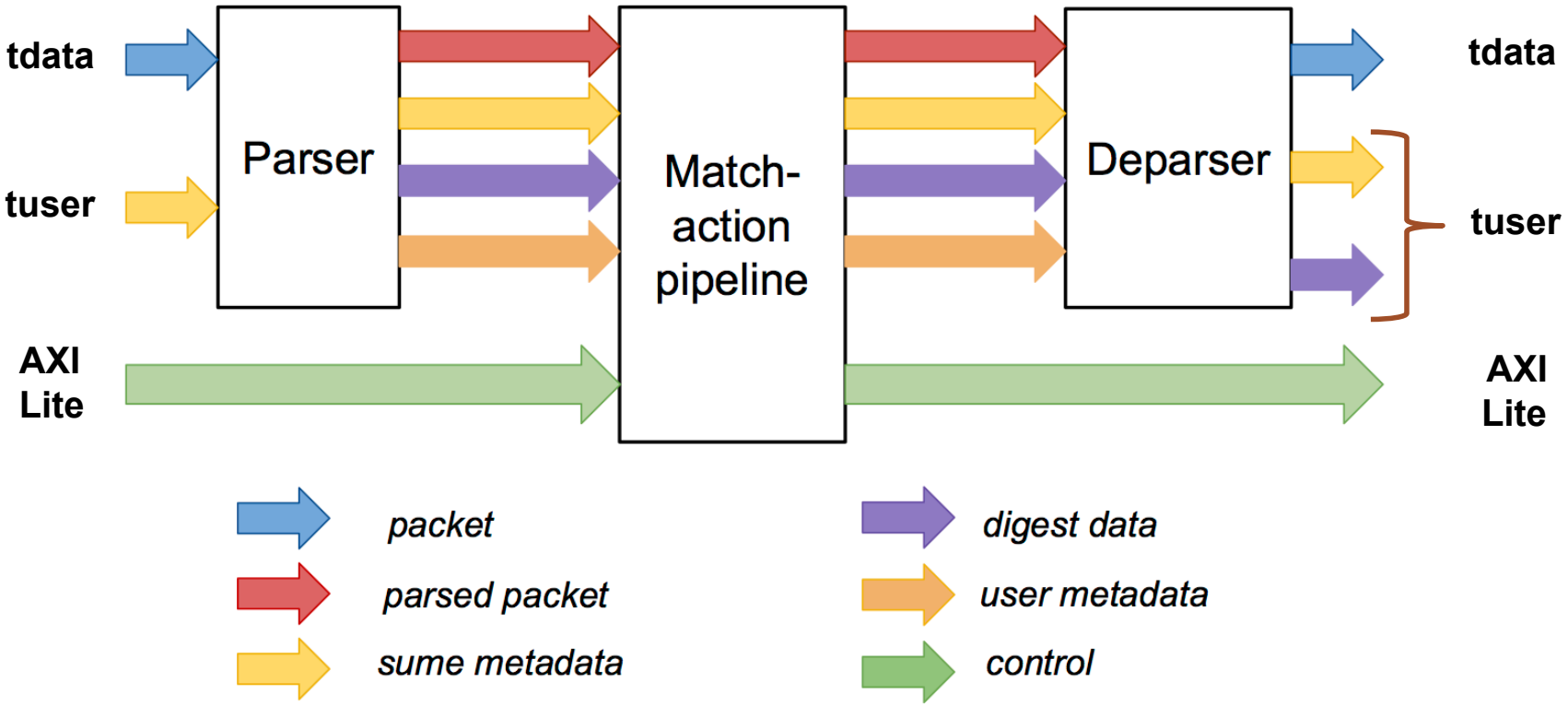
SimpleSumeSwitch Architecture



NetFPGA Reference Switch



SimpleSumeSwitch Architecture Model for SUME Target



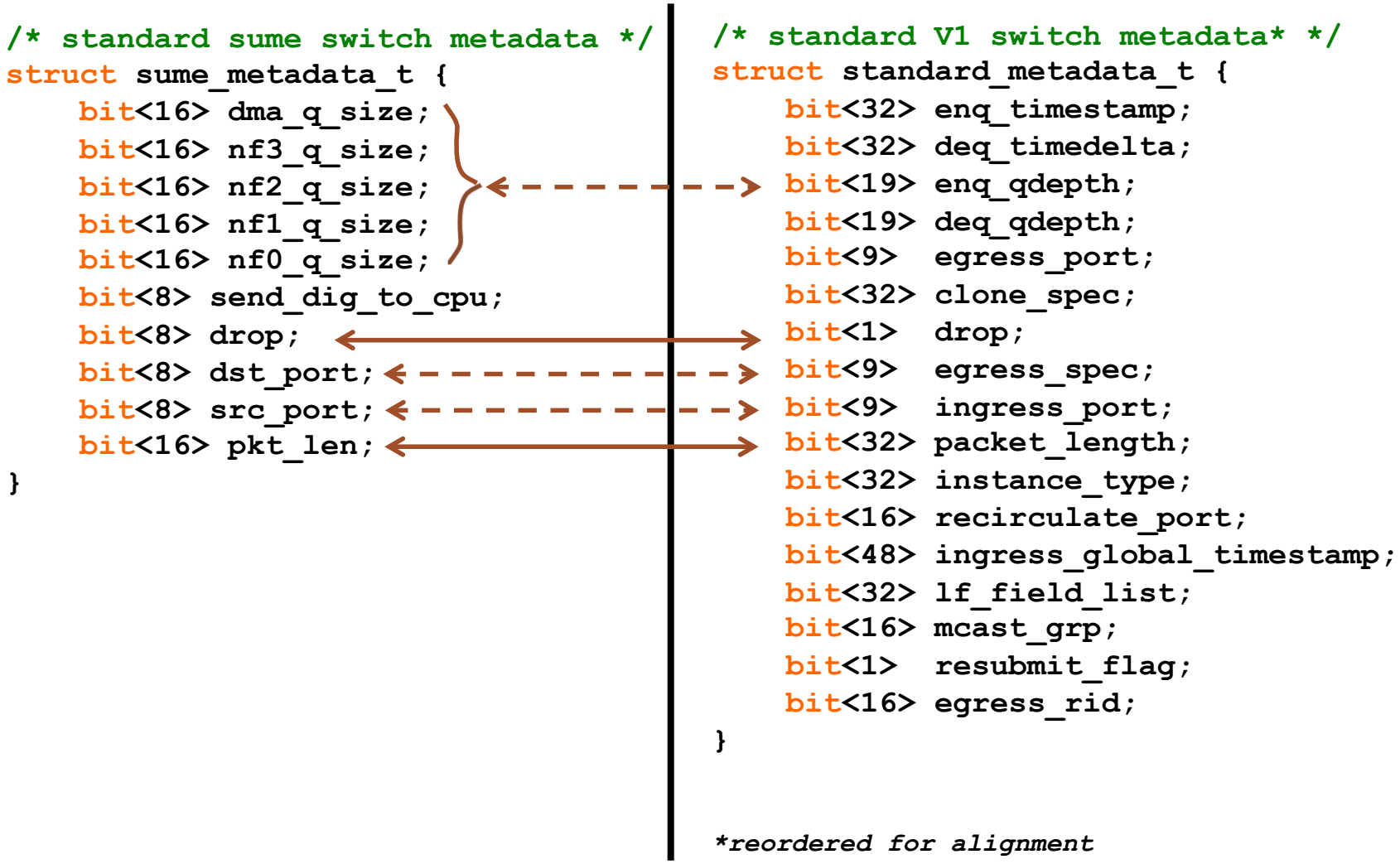
P4 used to describe parser, match-action pipeline, and deparser

Metadata in SimpleSumeSwitch Architecture

```
/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    bit<8> drop;
    bit<8> dst_port; // one-hot encoded
    bit<8> src_port; // one-hot encoded
    bit<16> pkt_len; // unsigned int
}
```

- `*_q_size` – size of each output queue, measured in terms of 32-byte words, when packet starts being processed by the P4 program
- `src_port`/`dst_port` – one-hot encoded, easy to do multicast
- `user_metadata`/`digest_data` – structs defined by the user

SimpleSumeSwitch vs. V1 Model



Overall P4 Program Structure

```
#include <core.p4>
#include <sume_switch.p4>

/******** CONSTANTS ********/
#define IPV4_TYPE   0x0800

/******** TYPES ********/
typedef bit<48> EthAddr_t;
header Ethernet_h {...}
struct Parsed_packet {...}
struct user_metadata_t {...}
struct digest_data_t {...}

/******** EXTERN FUNCTIONS ********/
extern void const_reg_rw(...);

/******** PARSERS and CONTROLS ********/
parser TopParser(...) {...}
control TopPipe(...) {...}
control TopDeparser(...) {...}

/******** FULL PACKAGE ********/
SimpleSumeSwitch(TopParser(), TopPipe(), TopDeparser()) main;
```

Externs

- Implement platform specific functions
  - Black box to P4 program
- Stateless – reinitialized for each packet
- Stateful – keep state between packets
- Xilinx Annotations
  - `@Xilinx_MaxLatency()` – maximum number of clock cycles an extern function needs to complete
  - `@Xilinx_ControlWidth()` – size in bits of the address space to allocate to an extern function

P4->NetFPGA Extern Function Library

- HDL modules invoked from within P4 programs
- Stateful Atoms

Atom	Description
R/W	Read or write state
RAW	Read, add to, or overwrite state
PRAW	Predicated version of RAW
ifElseRAW	Two RAWs, one each for when predicate is true or false
Sub	IfElseRAW with stateful subtraction capability

- Stateless Externs

Atom	Description
IP Checksum	Given an IP header, compute IP checksum
LRC	Longitudinal redundancy check, simple hash function
timestamp	Generate timestamp (granularity of 5 ns)

- Add your own!

Using Atom Externs in P4 – Resetting Counter

Packet processing pseudo code:

```
count[NUM_ENTRIES];

if (pkt.hdr.reset == 1):
    count[pkt.hdr.index] = 0
else:
    count[pkt.hdr.index]++
```

Using Atom Externs in P4 – Resetting Counter

```
#define REG_READ    0
#define REG_WRITE   1
#define REG_ADD     2

// count register
@Xilinx_MaxLatency(1)
@Xilinx_ControlWidth(3)
extern void count(in bit<3> index,
                  in bit<32> newVal,
                  in bit<32> incVal,
                  in bit<8> opCode,
                  out bit<32> result);

bit<16> index = pkt.hdr.index;
bit<32> newVal;
bit<32> incVal;
bit<8> opCode;

if(pkt.hdr.reset == 1) {
    newVal = 0;
    incVal = 0; // not used
    opCode = REG_WRITE;
} else {
    newVal = 0; // not used
    incVal = 1;
    opCode = REG_ADD;
}

bit<32> result; // the new value stored in count
count_reg_raw(index, newVal, incVal, opCode, result);
```

- State can be accessed exactly ***1 time***
- Using RAW atom here

- Instantiate atom

- Set metadata for state access

- **Single** state access!

P4-NetFPGA Workflow

1. Write P4 program
2. Write python gen_testdata.py script
3. Compile to verilog / generate API & CLI tools

   $ make
4. Run initial SDNet simulation

   $ ./vivado_sim.bash
5. Install SDNet output as SUME library core

   $ make install_sdnet
6. Run NetFPGA simulation

   $ ./nf_test sim –major switch –minor default
7. Build bitstream

   $ make
8. Test the hardware

All of your effort will go here

Directory Structure of $SUME_FOLDER

```
P4-NetFPGA-SIGCOMM/
|
|- contrib-projects/
|   |- sume-sdnet-switch/ → the main directory for P4 dev
|
|- lib/ → contains all of the SUME IP cores
|
|- tools/ → various NetFPGA scripts for test infra.
|
|- Makefile → builds all of the SUME IP cores
```

Directory Structure of $SUME_SDNET

```
sume-sdnet-switch/
|
|- bin/ → scripts used to automate workflow
|
|- templates/ → templates for externs, wrapper module,
|                CLI tools, new projects
|
|- projects/ → all of the P4 project directories
|   |- switch_calc/
```

Directory Structure of $P4_PROJECT_DIR

```
$P4_PROJECT_DIR/
|
|- src/ → P4 source files and commands.txt
|
|- testdata/ → scripts to generate testdata used for
|              verifying functionality of P4 program
|
|- simple_sume_switch/ → main SUME project directory,
|              top level HDL files and SUME sim scripts
|
|- sw/ → populated with API files and CLI tools and any
|        user software for the project
|
|- nf_sume_sdnet_ip/ → SDNet output directory
```

API & Interactive CLI Tool Generation

- Both Python API and C API
  - Manipulate tables and stateful elements in P4 switch
  - Used by control-plane program
- CLI tool
  - Useful debugging feature
  - Query various compile-time information
  - Interact directly with tables and stateful elements in real time

Debugging P4 Programs

- SDNet HDL simulation
- SDNet C++ simulation
  - Verbose packet processing info
  - Output PCAP file
- Full SUME design HDL simulation

What can you do in 2.5 days?

- Examples from the recent NetFPGA Summer Camp:
  - Heavy hitter detection
  - Event triggered network monitor
  - Packet Fuzzer
  - Tic-Tac-Toe

Get the Tools!

- Fill out P4->NetFPGA github registration form: here
- Request P4->NetFPGA tools and licenses from Xilinx: here
  - P4-SDNet License
  - Vivado license
  - 10G MAC Core license
- (For academic users) Request special price NetFPGA SUME: here
- (For industry users) Purchase NetFPGA SUME from Digilent: here
- Read the documentation: here

Today's Development Process

1. **Google Compute Instance:**
   - P4 development
   - P4 -> HDL compilation
   - Simulation + debugging
   - Bitstream generation
2. **Hardware Test Machines:**
   - Testing on SUME boards

- **Connecting to machines:**
  - Use "p4-netfpga-ssh-key"
  - Command line development OR VNC client

Tutorial Assignments

Tutorial Assignments

- Assignments described at this link:
  - https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Tutorial-Assignments
- Short version:
  - https://tinyurl.com/P4-NetFPGA-SIGCOMM-17

Assignment 1: Switch as a Calculator

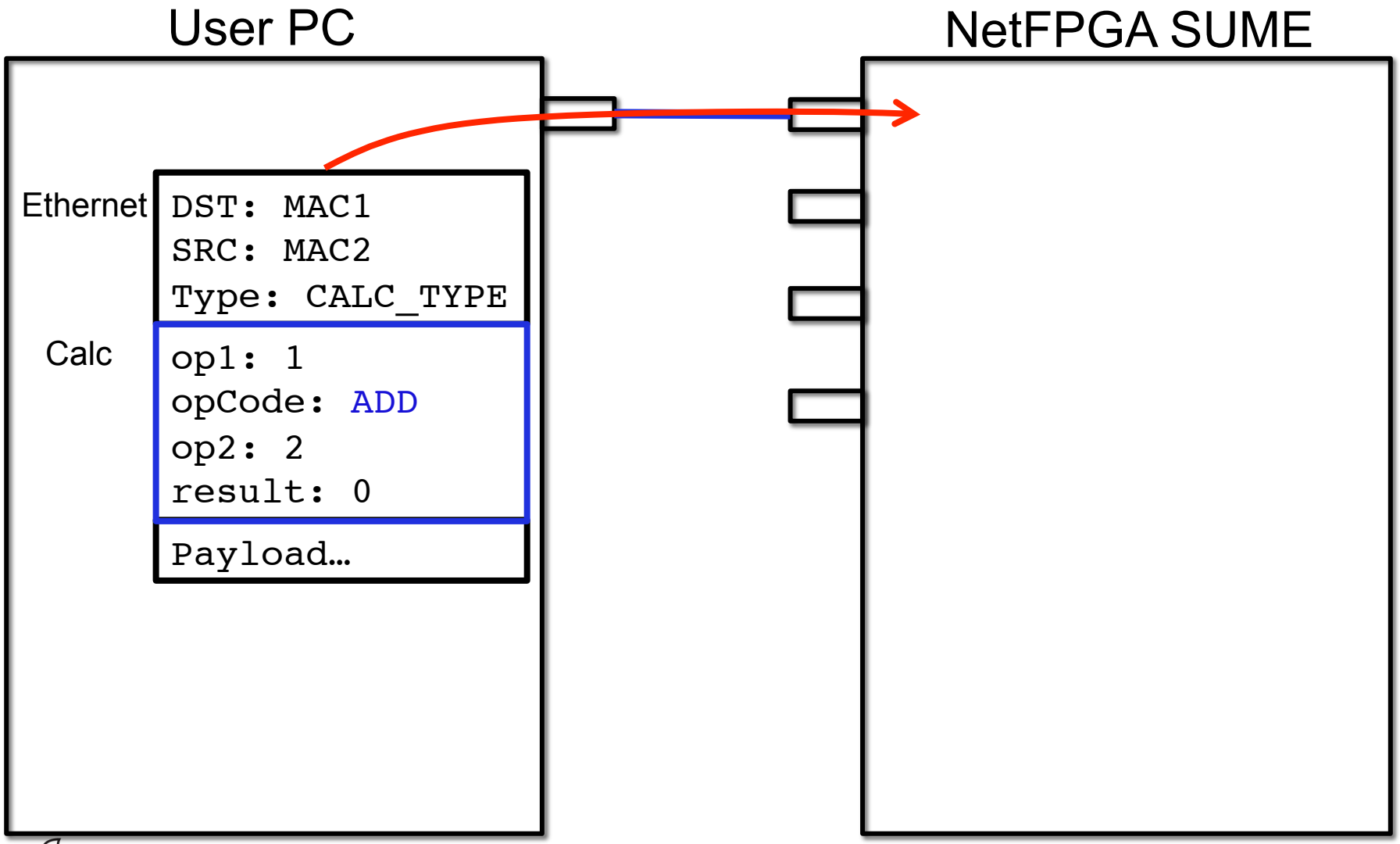
Switch as Calculator

Supported Operations:

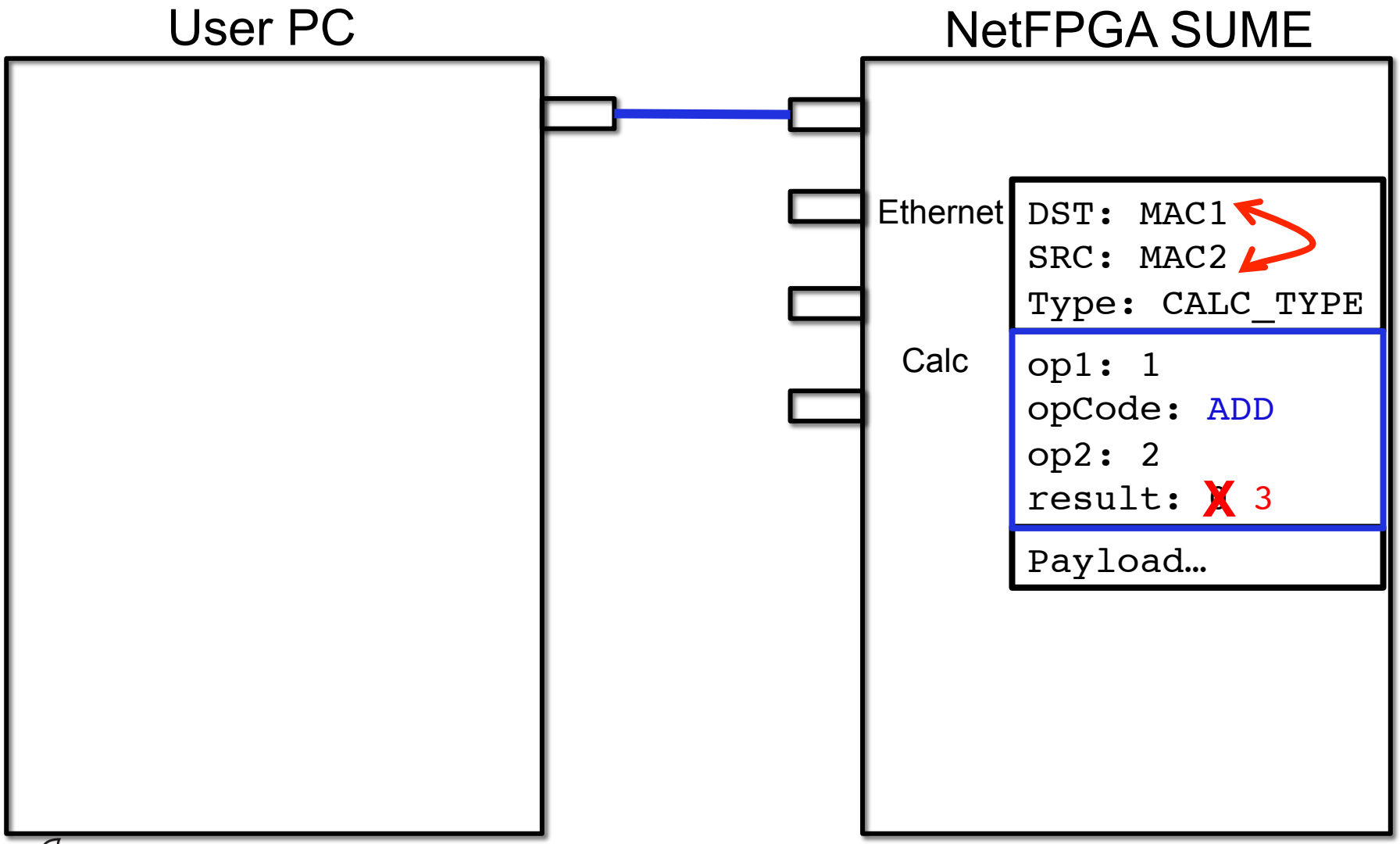
- ADD – add two operands
- SUBTRACT – subtract two operands
- ADD_REG – add operand to current value in register
- SET_REG – overwrite the current value of the register
- LOOKUP – Lookup the given key in the table

```
header Calc_h {
    bit<32> op1;
    bit<8> opCode;
    bit<32> op2;
    bit<32> result;
}
```

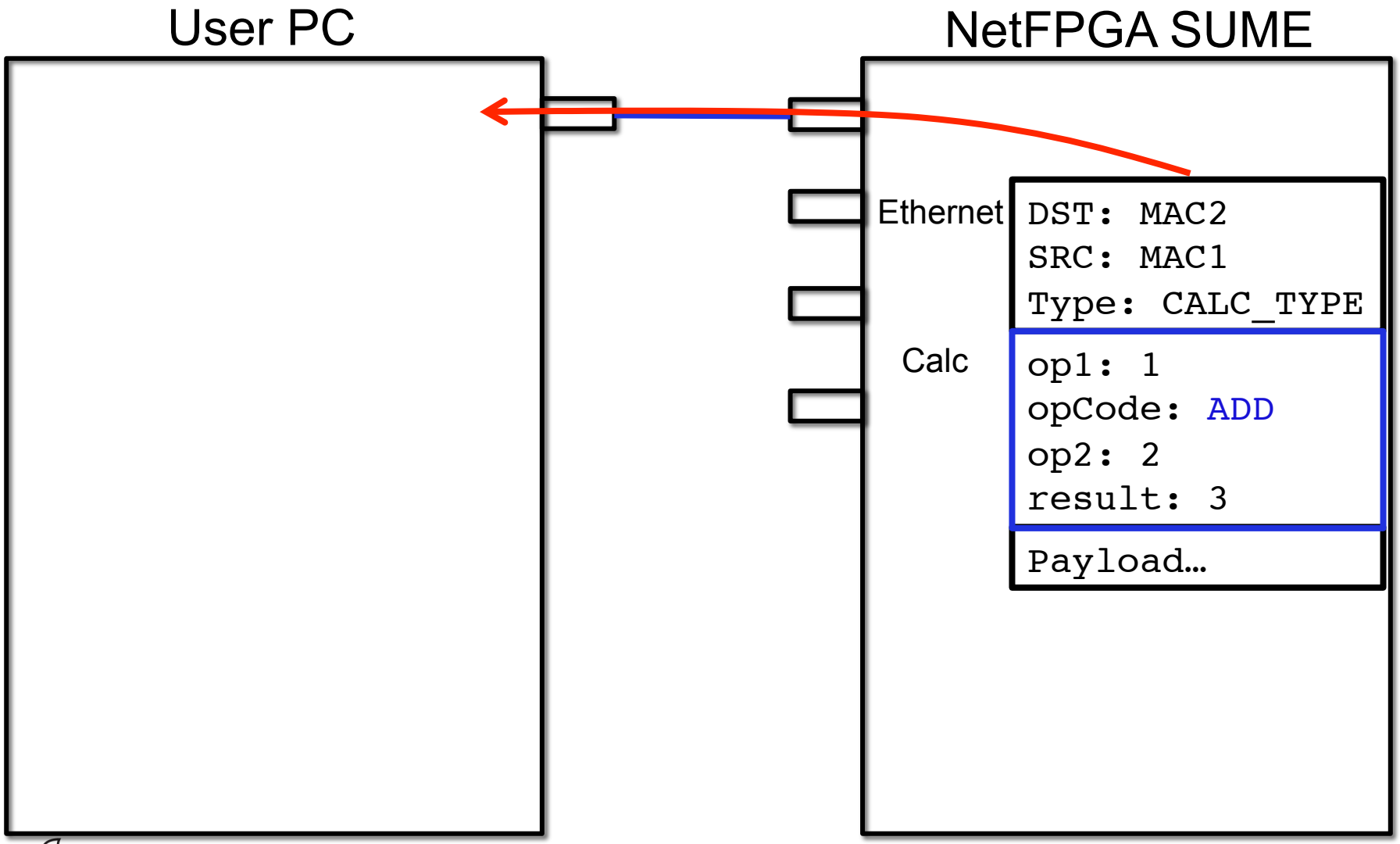
Switch as Calculator



Switch as Calculator

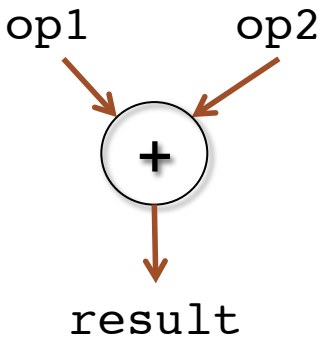


Switch as Calculator

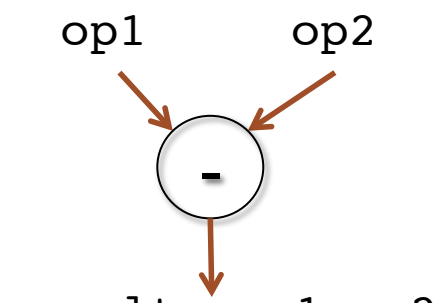


Switch Calc Operations

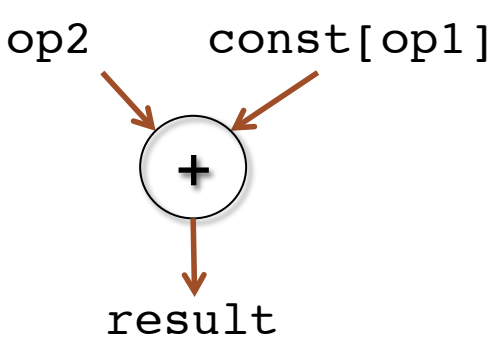
ADD



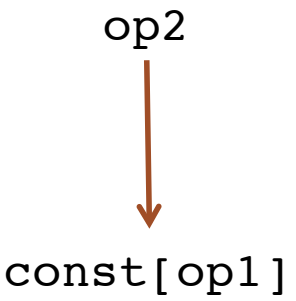
SUB



ADD_REG



SET_REG



LOOKUP

key: op1 →

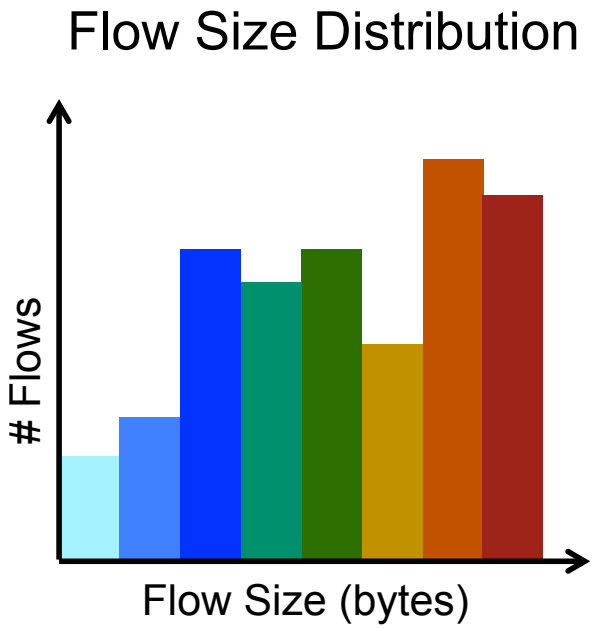
key	val
0	1
1	16
2	16^2
3	16^3

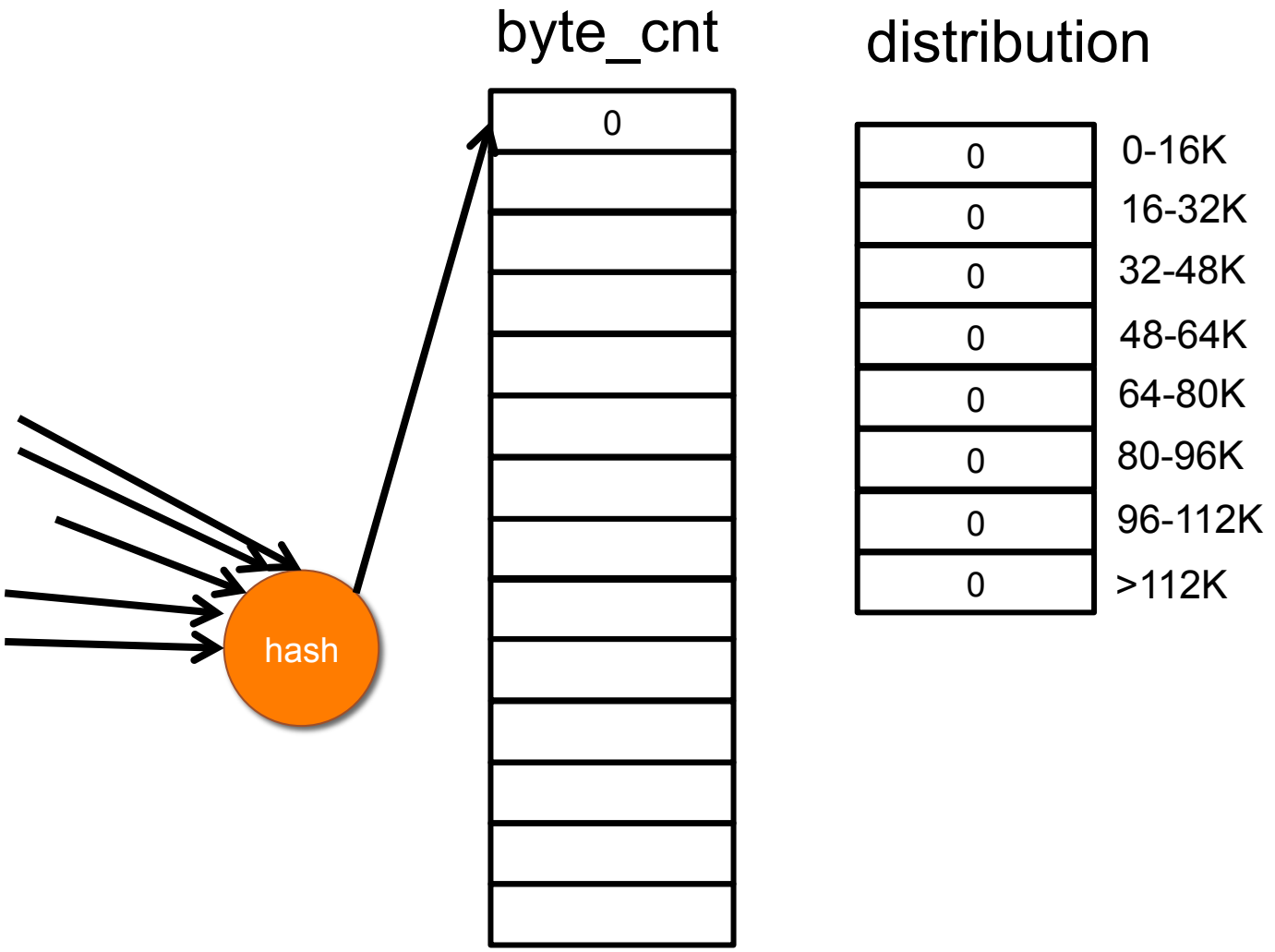
→ result: val

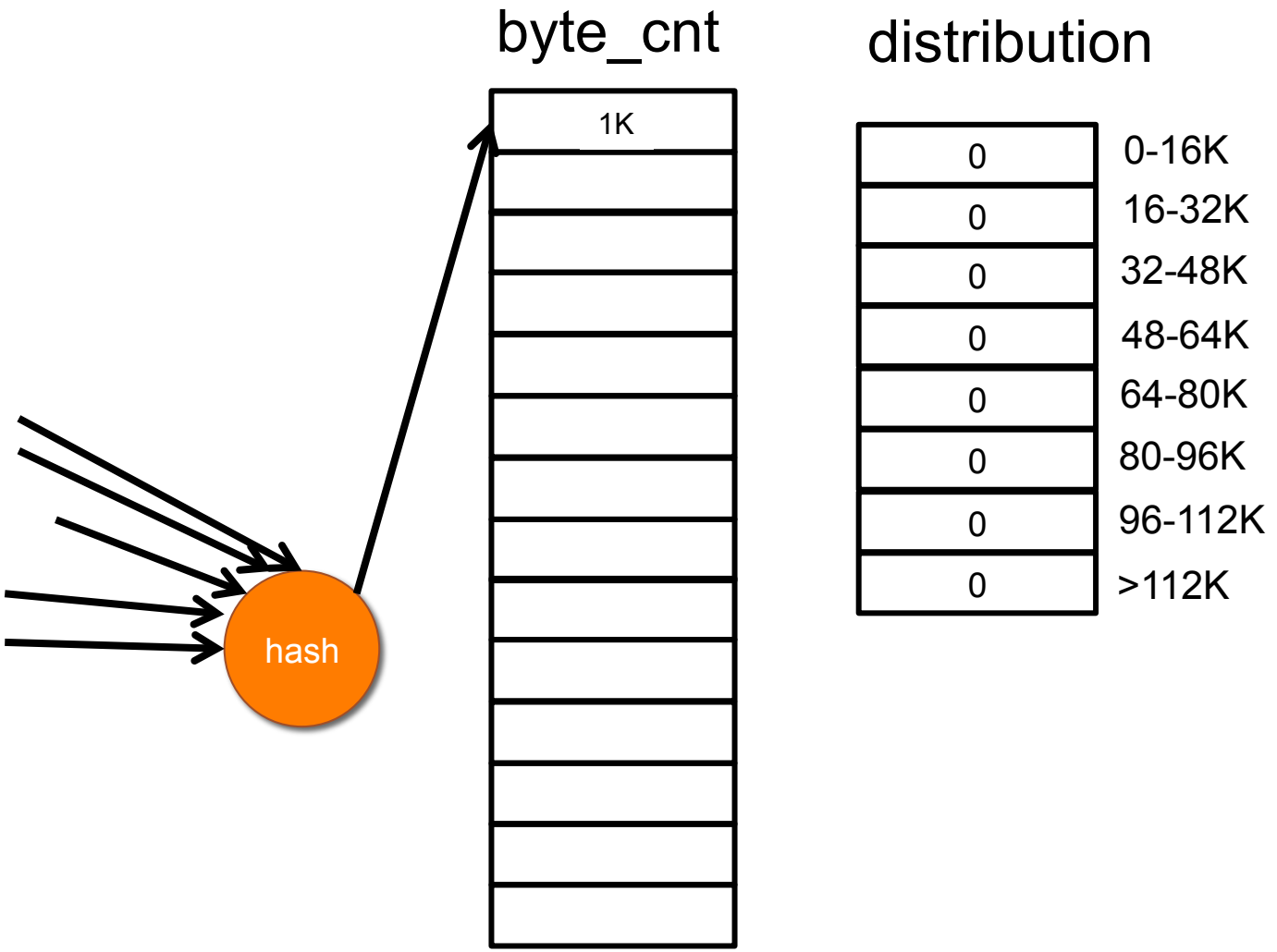
Assignment 2: TCP Monitor

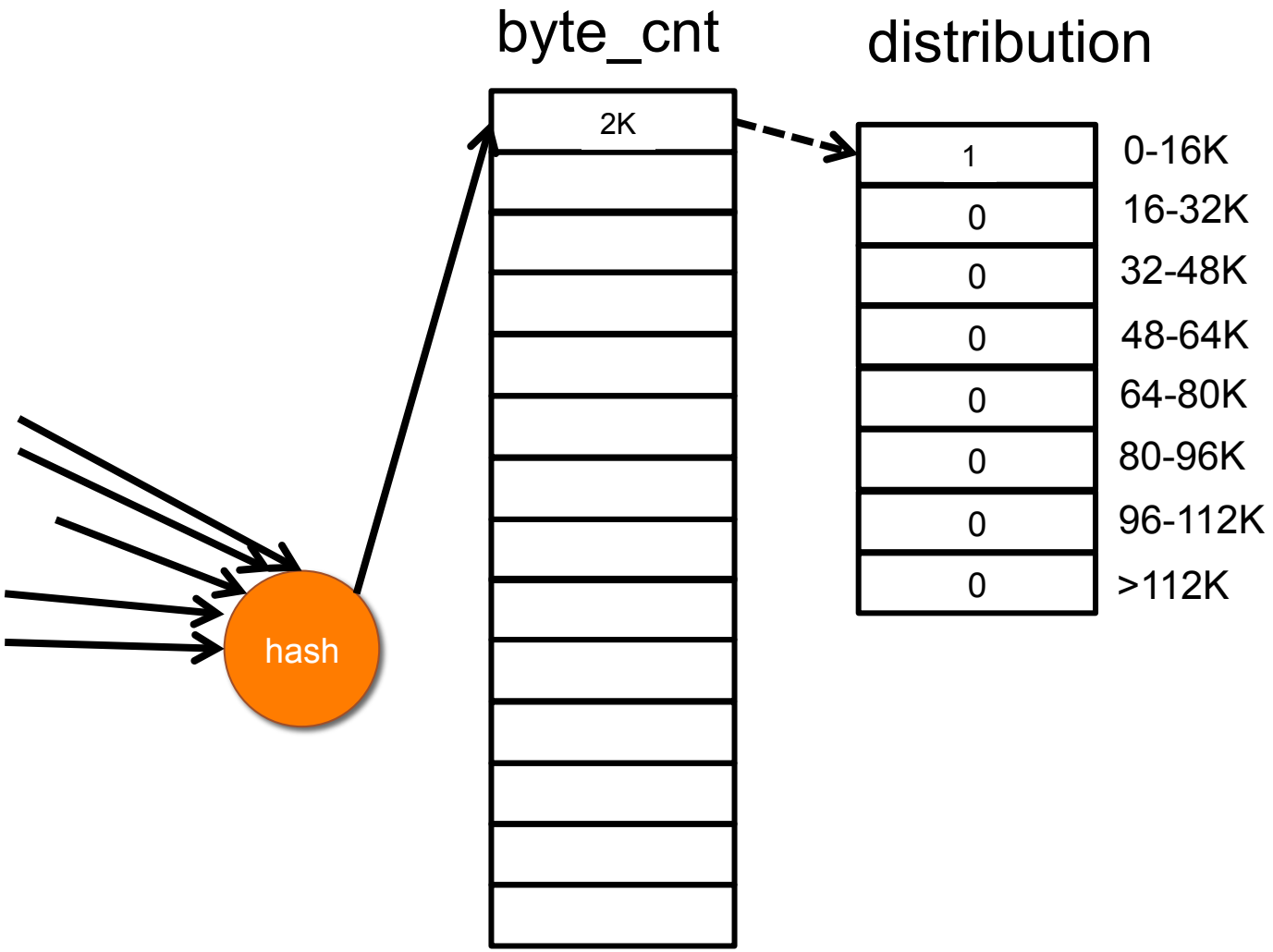
TCP Monitor

- Practice writing stateful P4 programs for NetFPGA SUME
- Compute TCP flow size distribution in the data-plane
- Flow is determined by 5-tuple and delimited by SYN/FIN
- Fine grained flow monitoring capabilities with P4




byte_cnt
distribution
0
hash
0
0
0
0
0
0
0
0-16K
16-32K
32-48K
48-64K
64-80K
80-96K
96-112K
>112K


byte_cnt
distribution
1K
hash
0
0
0
0
0
0
0
0
0-16K
16-32K
32-48K
48-64K
64-80K
80-96K
96-112K
>112K


byte_cnt
distribution
2K
1
0
0
0
0
0
0
0
0-16K
16-32K
32-48K
48-64K
64-80K
80-96K
96-112K
>112K
hash

Assignment 3: In-band Network Telemetry (INT)

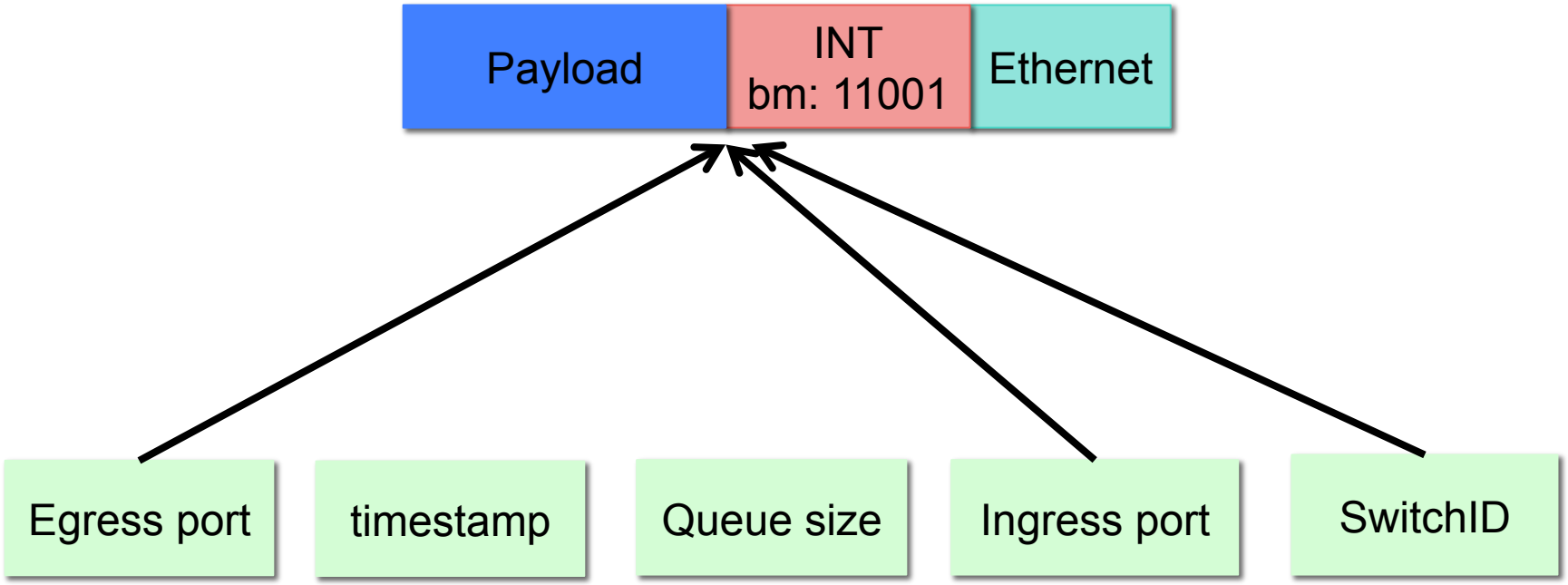
In-band Network Telemetry (INT)

- One of the most popular applications for programmable data-planes
- All about gaining more visibility into network
- Basic idea:
  - Source requests each switch along path to insert some desired metadata into packet (using a bitmask)
- Example metadata:
  - Switch ID
  - Ingress Port
  - Egress Port
  - Timestamp
  - Queue Occupancy

In-band Network Telemetry (INT)

- Bitmask format (5 bits):

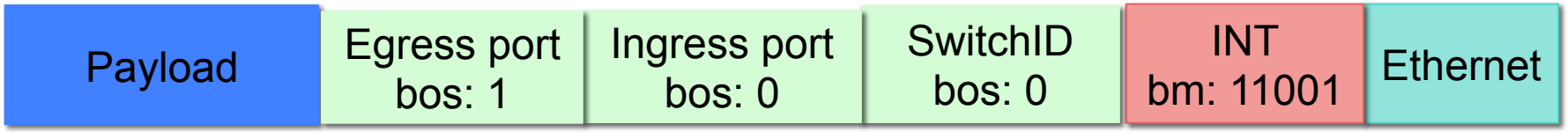
<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>

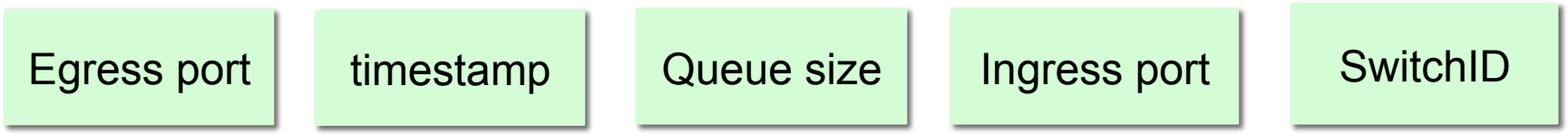


In-band Network Telemetry (INT)

- Bitmask format (5 bits):

<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>





Additional Details

API

- Auto generated Python API
- Exact match table API functions:
  - table_cam_add_entry()
  - table_cam_delete_entry()
  - table_cam_get_size()
  - table_cam_read_entry()
- Ternary match table API functions:
  - table_tcam_add_entry()
  - table_tcam_clean()
  - table_tcam_erase_entry()
  - table_tcam_verify_entry()
- LPM table API functions:
  - table_lpm_load_dataset()
  - table_lpm_set_active_lookup_bank()
  - table_lpm_verify_dataset()
- Register API functions:
  - reg_read()
  - reg_write()
- C API also available

Implementing Extern Functions

1. Implement verilog extern module in $SUME_SDNET/templates/externs/
2. Add entry to $SUME_SDNET/bin/extern_data.py

- No Need to modify any existing code
- If the extern has a control interface a cpu_regs module will be generated automatically to easily expose control_registers using the AXI Lite interface

Debugging SDNet Simulation

- HDLSimulation
  - Xsim or Questa sim
  - Error:

```
expected < tuple_out_sume_metadata > = < 00000000000000000000000100040041 >
actual   < tuple_out_sume_metadata > = < 00000000000000000000000100040040 >

$ $P4_PROJECT_DIR/testdata/sss_sdnet_tuples.py --parse
   00000000000000000000000100040041 sume
Parsed Tuple:
-----------------------
dma_q_size  =  0
nf3_q_size  =  0
nf2_q_size  =  0
nf1_q_size  =  0
nf0_q_size  =  0
send_dig_to_cpu  =  0
drop  =  1
dst_port  = 00000000
src_port  = 00000100
pkt_len  =  65
```

Debugging SDNet Simulation

- C++ model
  - Prints debug info
    - Hit or miss in tables
    - Header/Metadata field modification along the way
  - Generates Packet_expect.pcap to compare to dst.pcap
- -skipEval option
  - Compare C++ model output to HDL simulation – make sure SDNet compiler is working properly

Debugging SUME Simulation

- What to do if SDNet simulation passes but SUME simulation fails?
  - Make sure you've:
    - $ cd $P4_PROJECT_DIR && make config_writes
    - $ cd $NF_DESIGN_DIR/test/sim_major_minor && make
    - $ cd $P4_PROJECT_DIR && make install_sdnet
  - Run SUME simulation longer?
  - Make sure all configuration writes have completed before applying the test packets
  - Make sure SDNet module has finished reset before applying configuration writes
  - Check nf_*_expected.axi and nf_*_logged.axi
  - Add signals to simulation waveform
  - Perhaps issue with wrapper module?

SDNet Module HDL Wrapper

- Located in: $SUME_SDNET/templates/sss_wrapper/
- Tasks:
  - Generate tuple_in_VALID signal for s_axis_tuser
  - Reverse reset polarity
  - Reverses endianness of digest_data and inserts into last 80 bits of m_axis_tuser

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

NetFPGA XILINX UNIVERSITY OF CAMBRIDGE Stanford University

Acknowledgements (II)