NetFPGA : Tutorial in Seoul

Presented by:

Andrew W. Moore
(Cambridge University)
Martin Žádník
(Brno University of Technology)

Seoul, South Korea
February, 2009

http://NetFPGA.org

Welcome

Please organize into teams
2 People/computer

Wireless network
SSID : *Password on whiteboard*

Assume public IP assigned to eth0 for ssh login
root *: Password on whiteboard*

NetFPGA homepage
http://NetFPGA.org

NetFPGA Tutorial Program

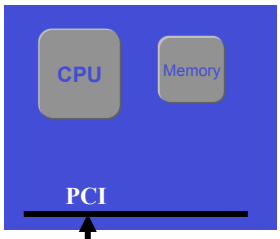
9:00 – 10:30 Session I
10:30 – 11:00 Coffee break
11:00 – 12:30 Session II
12:30 – 13:30 Lunch
13:30 – 15:00 Session III
15:00 – 15:30 Coffee break
15:30 – 17:00 Session IV

Tutorial Outline

- **Background**
  - Introduction
  - Basics of an IP Router
  - The NetFPGA Platform
- **The Stanford Base Reference Router**
  - Demo1: Reference Router running on the NetFPGA
  - Inside the NetFPGA hardware
  - Breakneck introduction to FPGAs and Verilog
  - Exercise 1: Build your own Reference Router
- **The Enhanced Reference Router**
  - Motivation: Understanding buffer size requirements in a router
  - Demo 2: Observing and controlling the queue size
  - Exercise 2: Enhancing the Reference Router
- **The Life of a Packet Through the NetFPGA**
  - Hardware Datapath
  - Interface to software: Exceptions and Host I/O
  - Exercise 3: Drop 1 in N Packets
- **Concluding Remarks**
  - Additional Hardware Platforms
  - Using NetFPGA for research and teaching
  - Group Discussion (All)

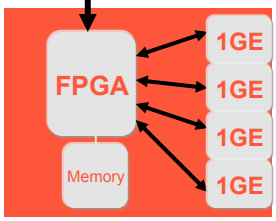
What is the NetFPGA?

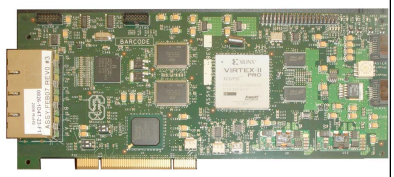
Networking Software running on a standard PC



PC with NetFPGA

A hardware accelerator built with Field Programmable Gate Array driving Gigabit network links





NetFPGA Board

Who, How, Why

Who uses the NetFPGA?

- Teachers
- Students
- Researchers

How do they use the NetFPGA?

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

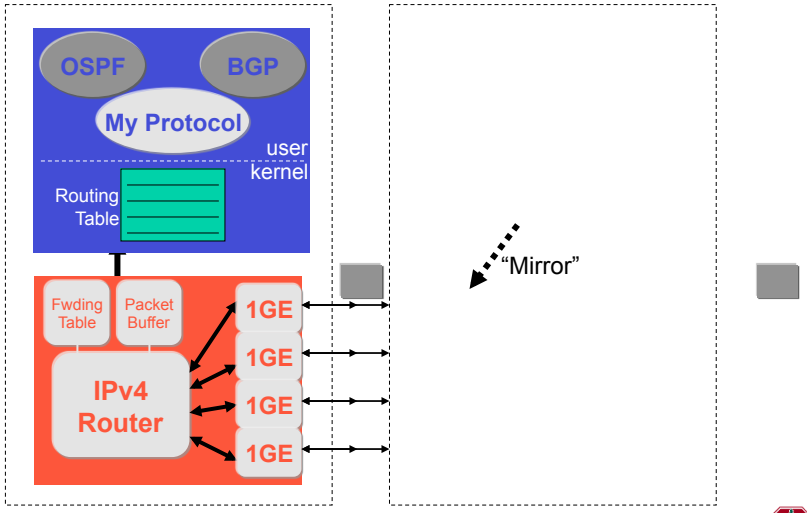
Why do they use the NetFPGA?

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

Usage #1

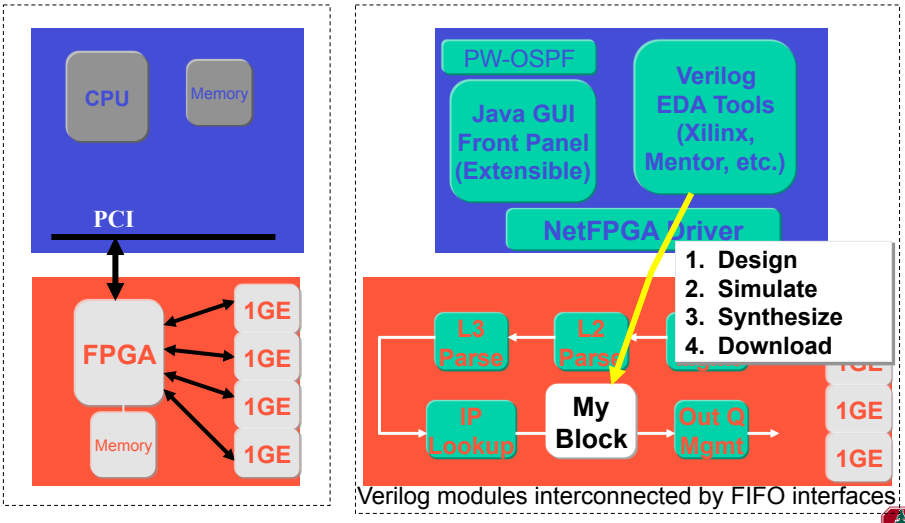
Running the Router Kit

User-space development, 4x1GE line-rate forwarding



Usage #2

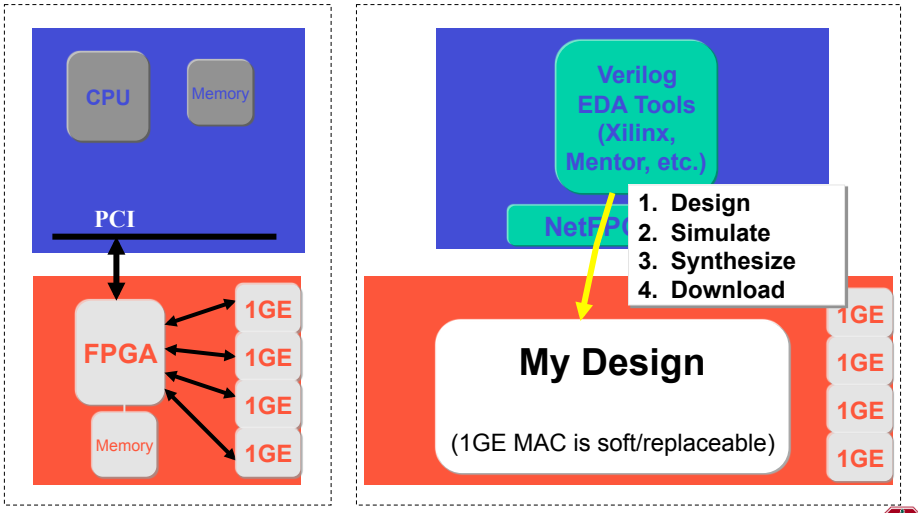
Enhancing Modular Reference Designs



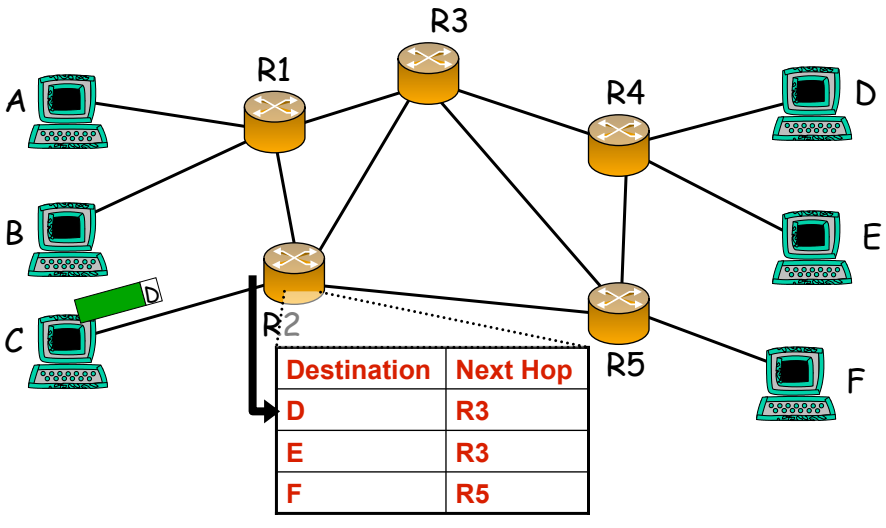
Verilog modules interconnected by FIFO interfaces

Usage #3

Creating new systems

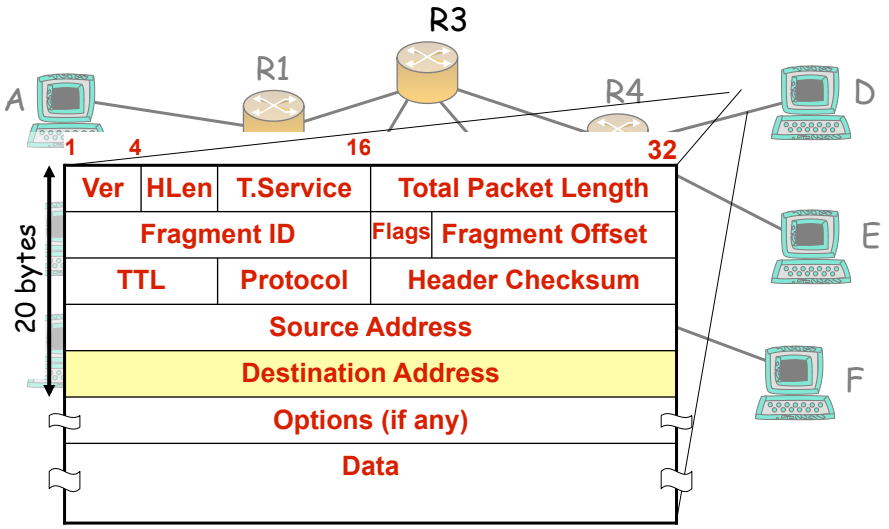


Basic Operation of an IP Router

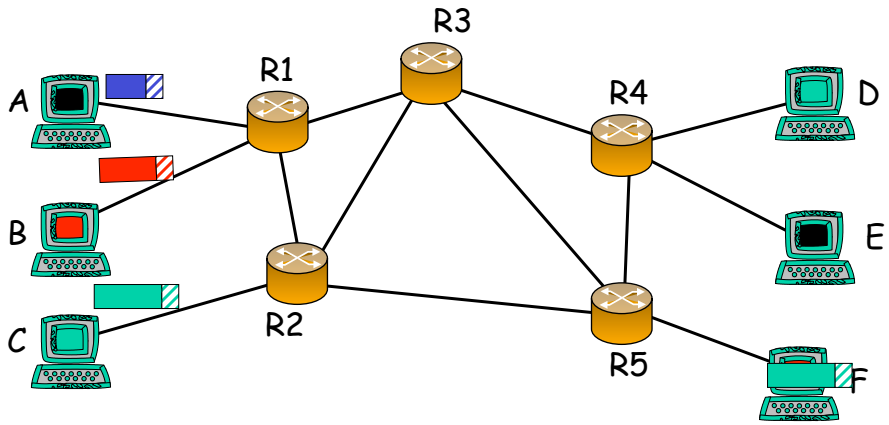


Destination	Next Hop
D	R3
E	R3
F	R5

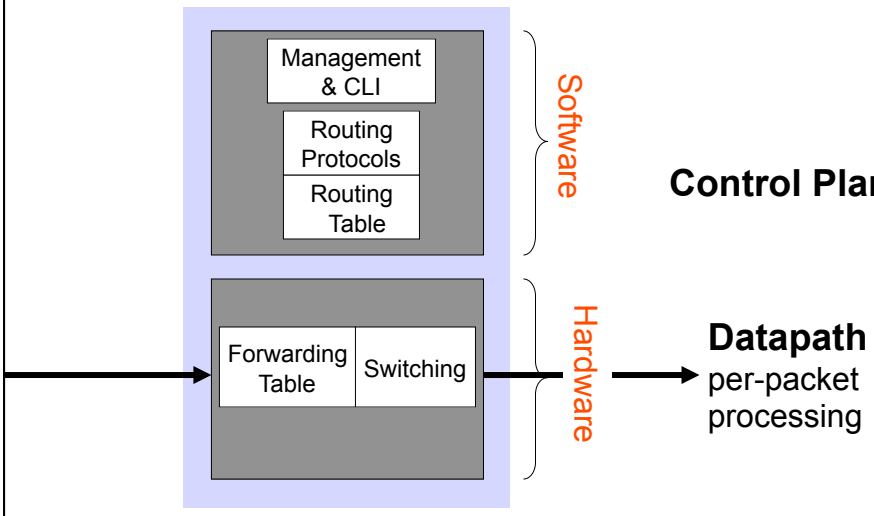
What does a router do?



What does a router do?



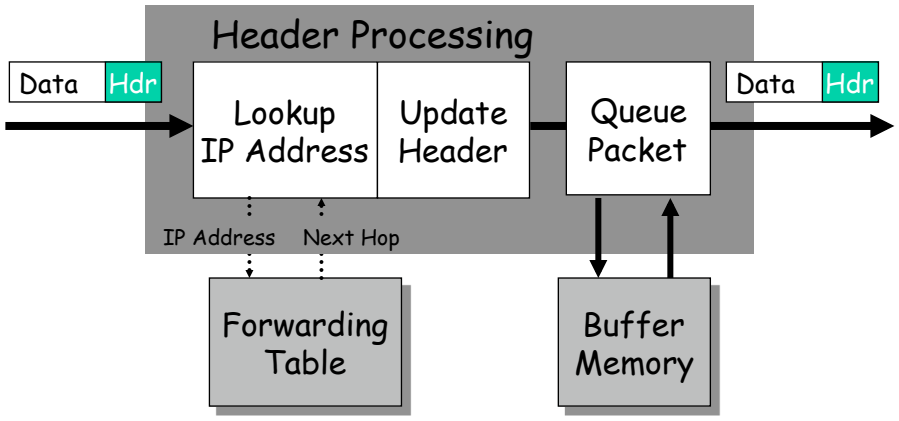
Basic Components of an IP Router



Per-packet processing in an IP Router

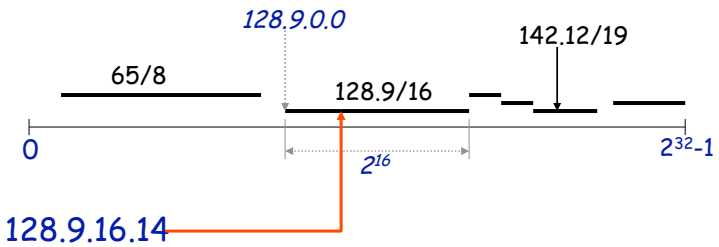
1. Accept packet arriving on an incoming link.
2. Lookup packet destination address in the forwarding table to identify outgoing port(s).
3. Manipulate IP header: e.g., decrement TTL, update header checksum.
5. Buffer packet in the output queue.
6. Transmit packet onto outgoing link.

Generic Datapath Architecture

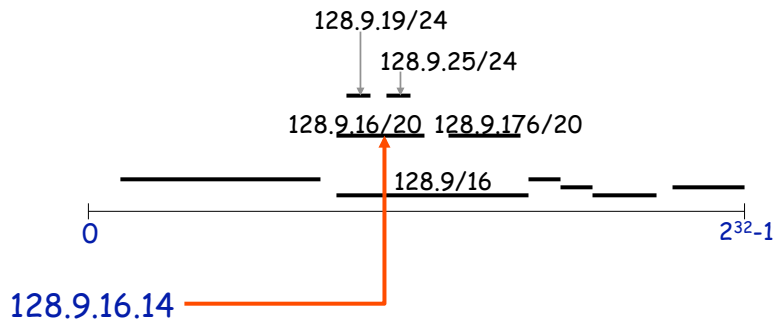


CIDR and Longest Prefix Matches

- The IP address space is broken into line segments.
- Each line segment is described by a ***prefix***.
- A prefix is of the form **x/y** where **x** indicates the prefix of all addresses in the line segment, and **y** indicates the length of the segment.
- e.g. The prefix 128.9/16 represents the line segment containing addresses in the range: 128.9.0.0 … 128.9.255.255.



Classless Interdomain Routing (CIDR)

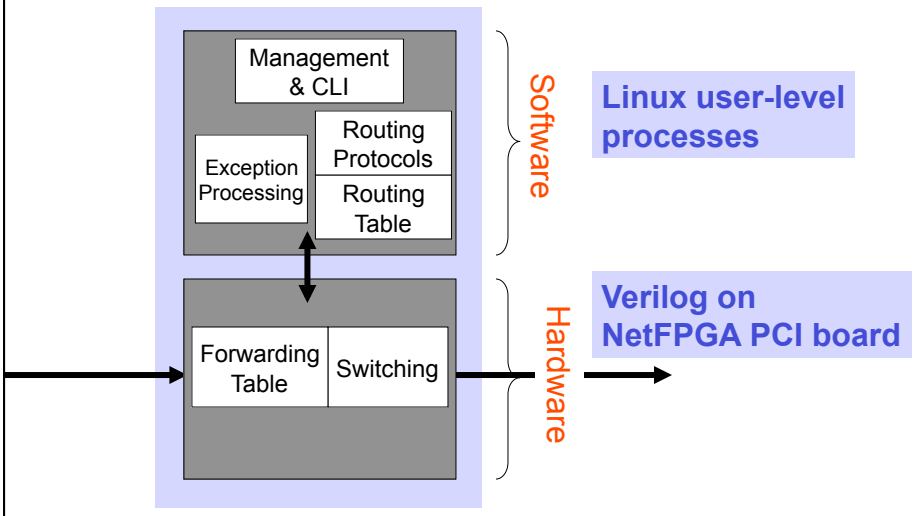


Most specific route = "longest matching prefix"

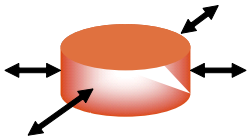
Techniques for LPM in hardware

- **Linear search**
  - Slow
- **Direct lookup**
  - Currently requires too much memory
  - Updating a prefix leads to many changes
- **Tries**
  - Deterministic lookup time
  - Easily pipelined but require multiple memories/ references
- **TCAM (Ternary CAM)**
  - Simple and widely used but have lower density than RAM and need more power
  - Gradually being replaced by algorithmic methods

An IP Router on NetFPGA



NetFPGA Router



Function
- 4 Gigabit Ethernet ports

Fully programmable
- FPGA hardware

Low cost

Open-source FPGA hardware
- Verilog base design

Open-souce Software
- Drivers in C and C++

NetFPGA Platform

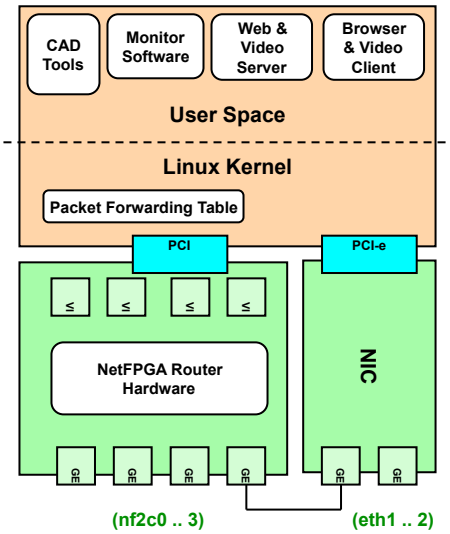
Major Components

– Interfaces
- 4 Gigabit Ethernet Ports
- PCI Host Interface

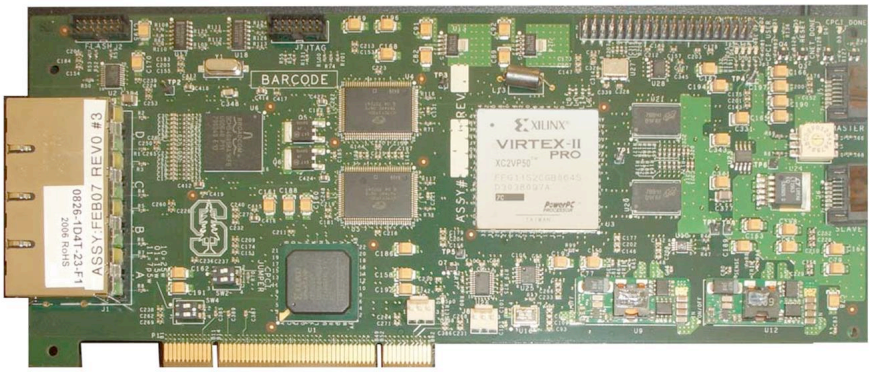
– Memories
- 36Mbits Static RAM
- 512Mbits DDR2 Dynamic RAM

– FPGA Resources
- Block RAMs
- Configurable Logic Block (CLBs)
- Memory Mapped Registers

NetFPGA System



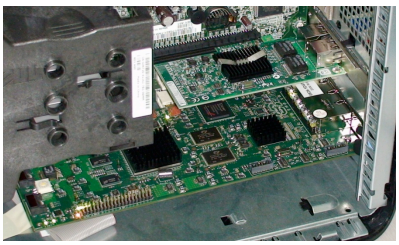
NetFPGA's Hardware Components

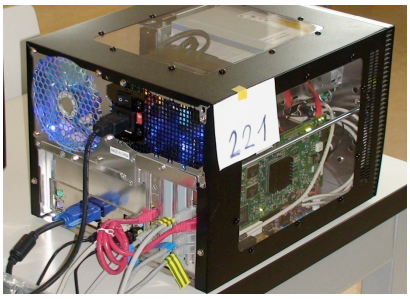


- **Xilinx Virtex-2 Pro FPGA for User Logic**
- **Xilinx Spartan for PCI Host Interface**
- **Cypress: 2 * 2.25 MB ZBT SRAM**
- **Micron: 64MB DDR2 DRAM**
- **Broadcom: PHY for 4 Gigabit Ethernet ports**

NetFPGA System Components

- **Network Ports**
  - Host PCI-express NIC
    - Dual Gigabit Ethernet ports on PCI-express card
  - NetFPGA
    - Quad Gigabit Ethernet ports on NetFPGA PCI card
- **Motherboard**
  - Standard AMD or Intel-based x86 computer with PCI and PCI-express slots
- **Processor**
  - Dual or Quad-Core CPU
- **Operating System**
  - Linux CentOS 5.2

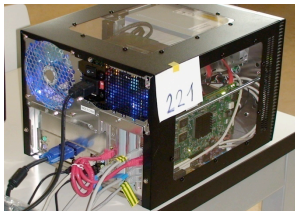




NetFPGA Cube Systems

- **PCs assembled from parts**
  - Stanford University
  - Cambridge University
- **Pre-built systems available**
  - Accent Technology Inc.
- **Details are in the Guide**

http://netfpga.org/static/guide.html



Rackmount NetFPGA Servers

2U Server
(Dell 2950)

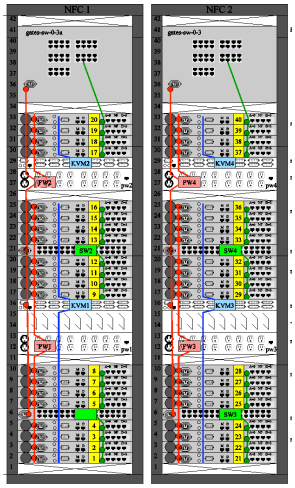
NetFPGA inserts in
PCI or PCI-X slot

1U Server
(Accent Technology Inc.)

Thanks: Brian Cashman for providing machine

Stanford NetFPGA Cluster

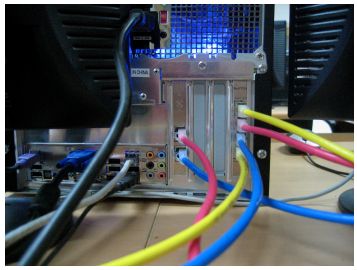
Stanford NetFPGA Cluster (NFC)
Internetconnect-side View

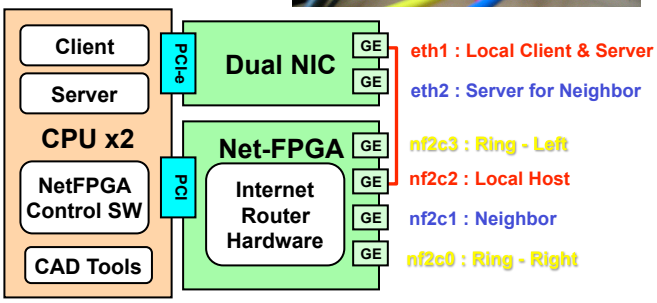


Statistics

- Rack of 40
  - 1U PCs with NetFPGAs
- Manged
  - Power
  - Console
  - LANs
- Provides 4*40=160 Gbps of full line-rate processing bandwidth

NetFPGA Lab Setup



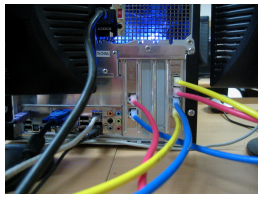


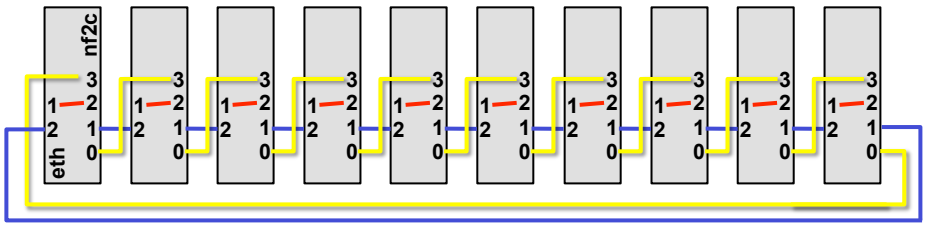
NetFPGA Hardware Set for Demo #1

Server delivers streaming HD video through a chain of NetFPGA Routers

Cable Configuration in the Lab

- **NetFPGA Gigabit Ethernet Interfaces**
  - nf2c3 : Left neighbor in network (yellow)
  - nf2c2 : Local host interface (red)
  - nf2c1 : Routes for adjacent server (blue)
  - nf2c0 : Right neighbor in network (yellow)
- **Host Ethernet Interfaces**
  - eth1 : Local host interface (red)
  - eth2 : Server for neighbor (blue)





Demo 1

Reference Router running on the NetFPGA

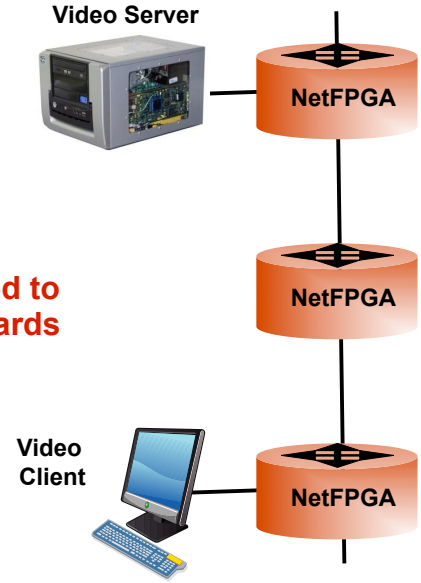
Demo 1

Setup for the Reference Router

Each NetFPGA card has four ports

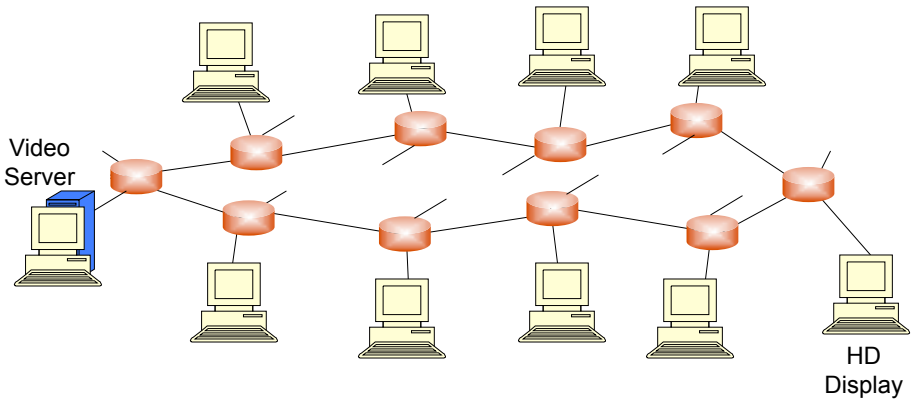
Port 2 connected to Client / Server

Ports 0 and 3 connected to adjacent NetFPGA cards



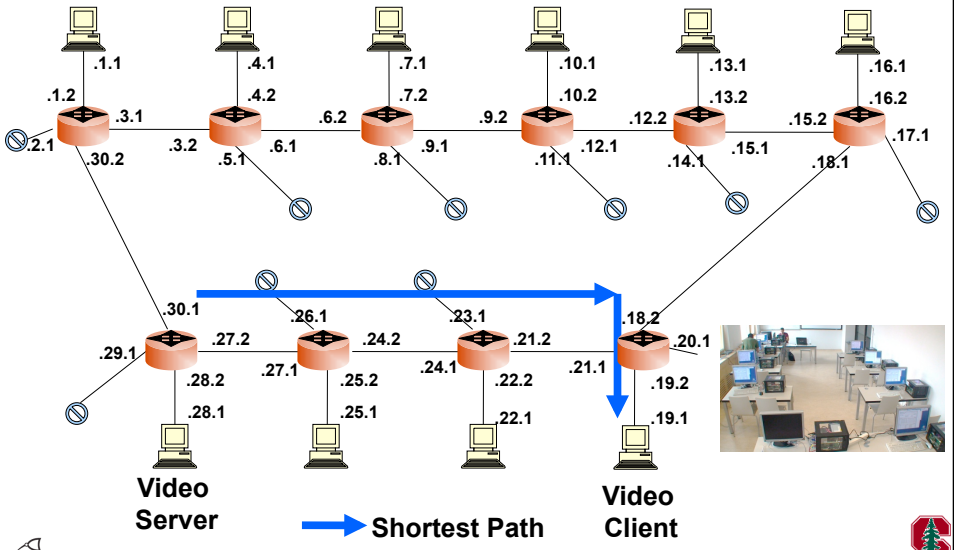
Demo 1

Topology of NetFPGA Routers



Demo 1

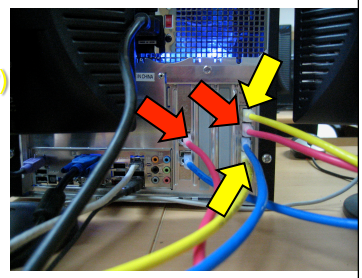
Subnet Configuration

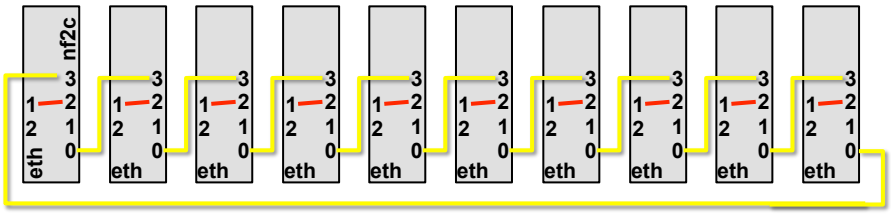


Demo 1

Cable Configuration for Demo 1

- **NetFPGA Gigabit Ethernet Interfaces**
  - nf2c3 : Left neighbor in network (yellow)
  - nf2c2 : Local host interface (red)
  - nf2c0 : Right neighbor in network (yellow)
- **Host Ethernet Interfaces**
  - eth1 : Local host interface (red)





Demo 1

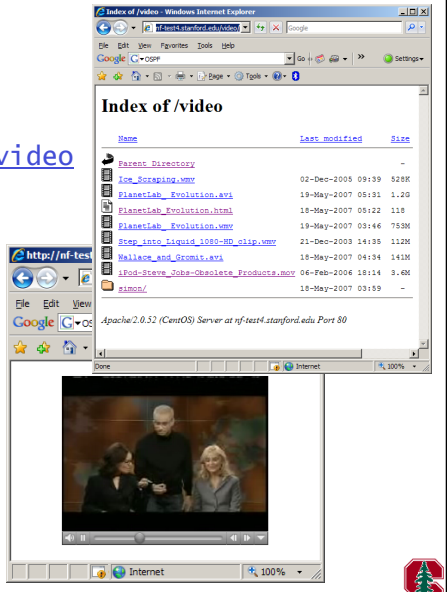
Working IP Router

- **Objectives**
  - Become familiar with Stanford Reference Router
  - Observe PW-OSPF re-routing traffic around a failure

Demo 1

Streaming Video through the NetFPGA

- **Video server**
  - Source files
    `/var/www/html/video`
  - Network URL :
    `http://192.168.Net.Host/video`
- **Video client**
  - Windows Media Player
  - Linux `mplayer`
- **Video traffic**
  - MPEG2 HDTV (35 Mbps)
  - MPEG2 TV (9 Mbps)
  - DVI (3 Mbps)
  - WMF (1.7 Mbps)



Demo 1 Physical Configuration

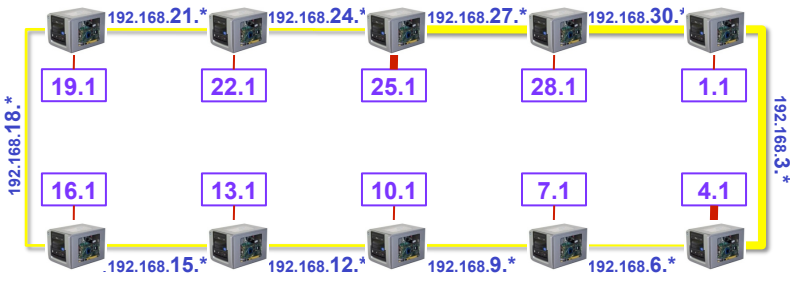
Key:

eth1 of Host PC
192.168.X.Y

NetFPGA Router #

To stream mplayer video from server 4.1, type:
`mp 192.168.4.1`

Any PC can stream traffic through multiple NetFPGA routers in the ring topology to any other PC



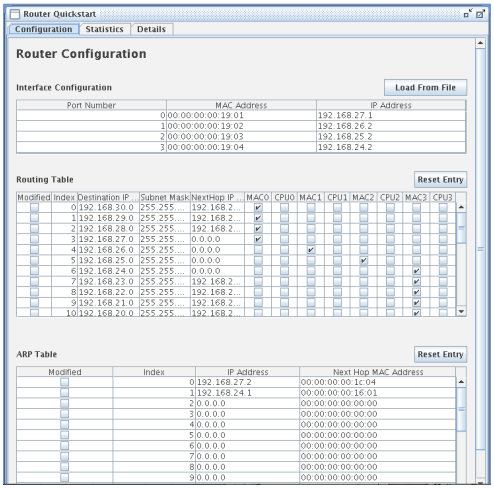
Demo 1

Step 1 – Observe the Routing Tables

The router is already configured and running on your machines

The routing table has converged to the routing decisions with minimum number of hops

Next, break a link …

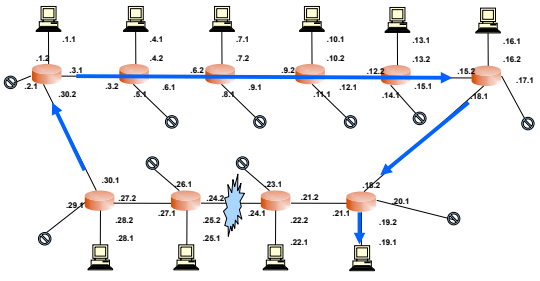


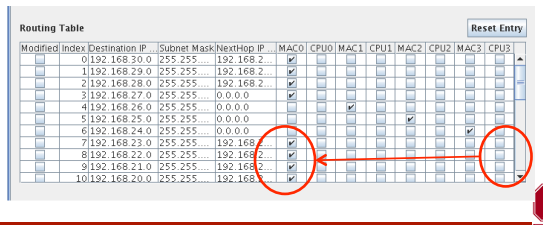
Demo 1

Step 2 - Dynamic Re-routing

Break the link between video server and video client

Routers re-route traffic around the broken link and video continues playing





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

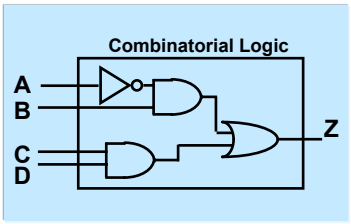
Programmable Logic Device
- Programmable Array Logic
- Field Programmable Gate Arrays

Processors

Look-Up Tables

Combinatorial logic is stored in Look-Up Tables (LUTs)

- Also called Function Generators (FGs)
- Capacity is limited only by number of inputs, not complexity
- Delay through the LUT is constant



A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Diagram From: Xilinx, Inc

Xilinx CLB Structure

Each slice has four outputs

- Two registered outputs, two non-registered outputs
- Two BUFTs associated with each CLB, accessible by all 16 CLB outputs

Carry logic run vertically

- Signals run upward
- Two independent carry chains per CLB

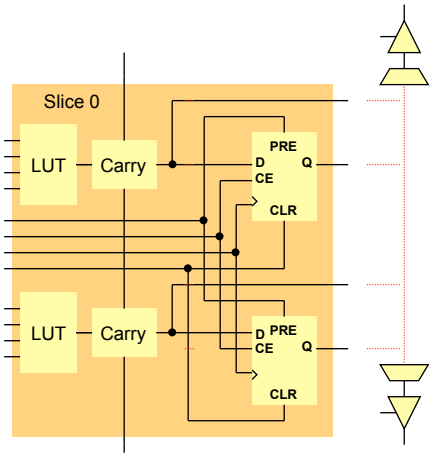
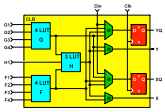


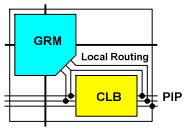
Diagram From: Xilinx, Inc.

Field Programmable Gate Arrays

CLB (4 slices)
- Primitive element of FPGA

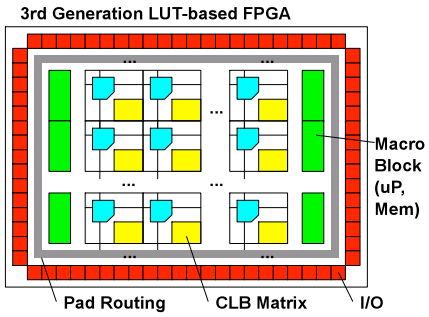


Routing Module
- Global routing
- Local interconnect

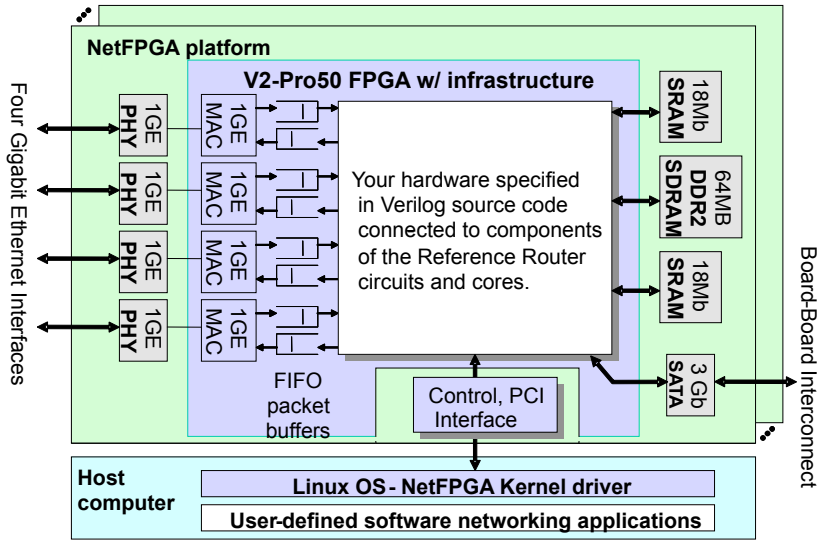


Macro Blocks
- Block Memories
- Microprocessor

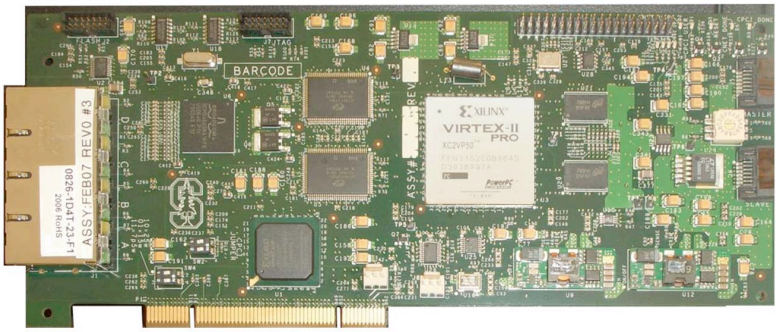
I/O Block



NetFPGA Block Diagram



Details of the NetFPGA



- **Fits into standard PCI slot**
  - Standard Bus: 32 bits, 33 MHz
- **Provides interfaces for processing network packets**
  - 4 Gigabit Ethernet Ports
- **Allows hardware-accelerated processing**
  - Implemented with Field Programmable Gate Array (FPGA) Logic

Introduction to the Verilog Hardware Description Language

Hardware Description Languages

- **Concurrent**
  - By default, Verilog statements evaluated concurrently
- **Express *fine grain* parallelism**
  - Allows *gate-level* parallelism
- **Provides Precise Description**
  - Eliminates ambiguity about operation
- **Synthesizable**
  - Generates hardware from description

Verilog Data Types

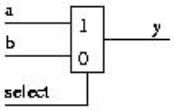
```
reg [7:0] A; // 8-bit register, MSB to LSB
             // (Preferred bit order for NetFPGA)
reg [0:15] B; // 16-bit register, LSB to MSB

B = {A[7:0],A[0:7]}; // Assignment of bits

reg [31:0] Mem [0:1023]; // 1K Word Memory

integer Count;   // simple signed 32-bit integer
integer K[1:64]; // an array of 64 integers
time Start, Stop; // Two 64-bit time variables
```

From: CSCI 320 Computer Architecture
Handbook on Verilog HDL, by Dr. Daniel C. Hyde :
http://eesun.free.fr/DOC/VERILOG/verilog-manual.html

Signal Multiplexers

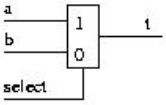
Two input multiplexer (using if / else)

```
reg y;
always @*
      if (select)
         y = a;
      else
         y = b;
```



Two input multiplexer (using ternary operator ?:)

```
wire t = (select ? a : b);
```

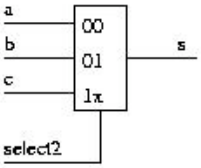


From: http://eesun.free.fr/DOC/VERILOG/synvlg.html

Larger Multiplexers

Three input multiplexer

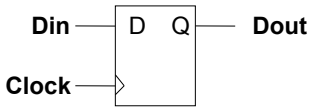
```
reg s;
always @*
   begin
   case (select2)
      2'b00: s = a;
      2'b01: s = b;
      default: s = c;
   endcase
   end
```



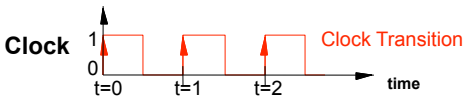
From: http://eesun.free.fr/DOC/VERILOG/synvlg.html

Synchronous Storage Elements

- **Values change at times governed by clock**

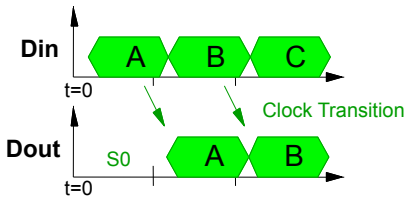


– Clock
  - Input to circuit

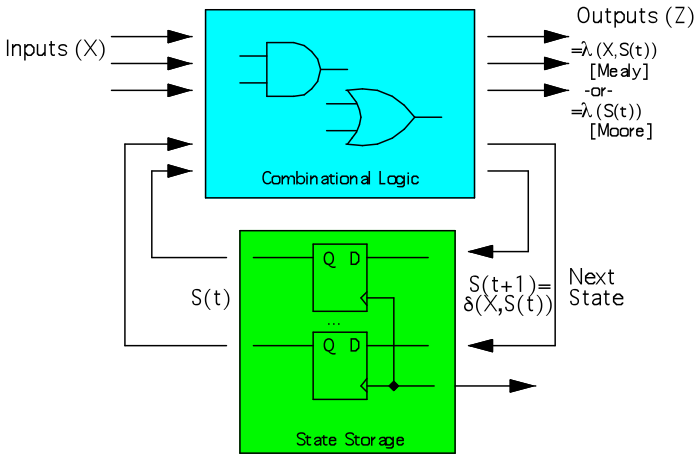


– Clock Event
  - Example: Rising edge

– Flip/Flop
  - Transfers value from D_{in} to D_{out} on clock event



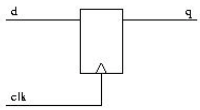
Finite State Machines

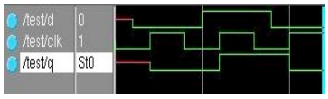


Synthesizable Verilog: Delay Flip/Flops

D-type flip flop

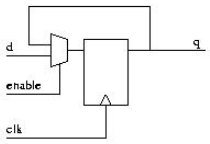
```
reg q;
always @ (posedge clk)
     q <= d;
```

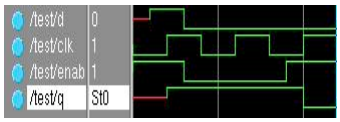




D type flip flop with *data enable*

```
reg q;
always @ (posedge clk)
     if (enable)
       q <= d;
```





From: http://eesun.free.fr/DOC/VERILOG/synvlg.html

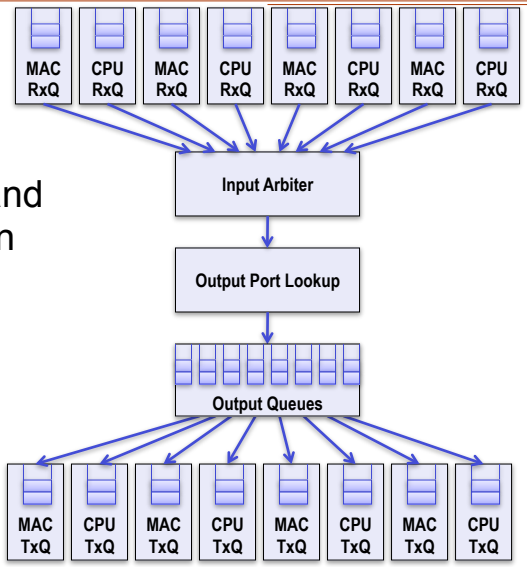
Exercise 1

Build the Reference Router

Exercise 1

Reference Router Pipeline

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



Exercise 1

Make your own router

Objectives:

- Learn how to build hardware
- Run the software
- Explore router architecture

Execution

- Start synthesis
- Rerun the GUI with the new hardware
- Test connectivity and statistics with pings
- Explore pipeline in the details page
- Explore detailed statistics in the details page

Exercise 1

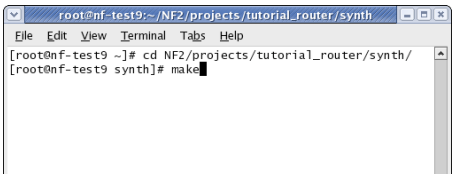
Step 1 - Build the Hardware

Close all windows

Start terminal, cd to “NF2/projects/tutorial_router/synth”

Run “make clean”

Start synthesis with “make”



First Break

(while hardware compiles)

Exercise 1

Step 2 - Run Homemade Router

cd to “NF2/projects/tutorial_router/sw”

To use the just-built router hardware, type:

./tut_router_gui.pl --use_bin ../../../bitfiles/tutorial_router.bit

To stream video, run:

./mp 192.168.X.Y where X.Y = 25.1 or 19.1 or 7.1

(or other server as listed on Demo 1 handout)

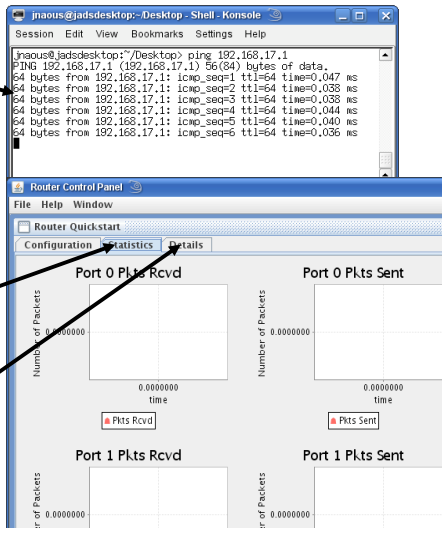
Exercise 1

Step 4 - Connectivity and Statistics

Ping any addresses 192.168.x.y where x is from 1-20 and y is 1 or 2

Open the statistics tab in the Quickstart window to see some statistics

Explore more statistics in modules under the details tab

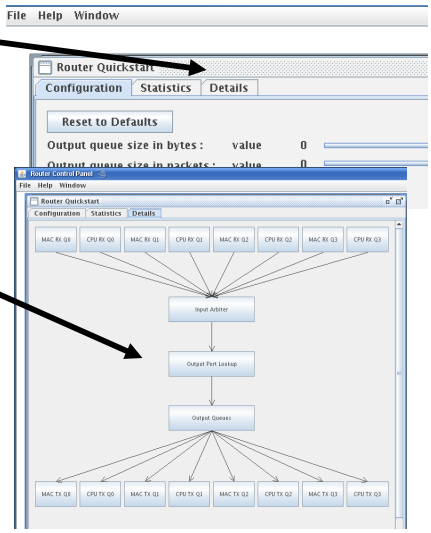


Exercise 1

Step 5 - Explore Router Architecture

Click the Details tab of the Quickstart window

This is the reference router pipeline – a canonical, simple-to-understand, modular router pipeline



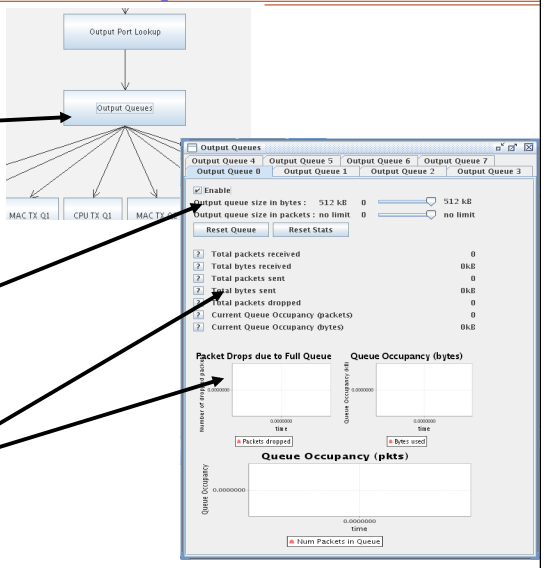
Exercise 1

Step 6 - Explore Output Queues

Click on the Output Queues module in the Details tab

The page gives configuration details

…and statistics



Understanding Buffer Size Requirements in a Router

Buffer Requirements in a Router

Buffer size matters:

- Small queues reduce delay
- Large buffers are expensive

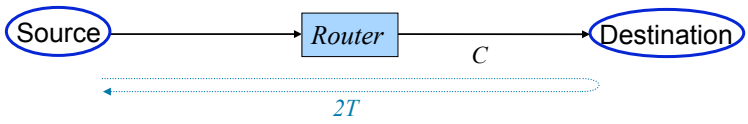
Theoretical tools predict requirements

- Queuing theory
- Large deviation theory
- Mean field theory

Yet, there is no direct answer

- Flows have a closed-loop nature
- Question arises on whether focus should be on equilibrium state or transient state

Rule-of-thumb



- **Universally applied rule-of-thumb:**
  - A router needs a buffer size: $B = 2T \times C$
  - 2T is the two-way propagation delay (or just 250ms)
  - C is capacity of bottleneck link
- **Context**
  - Mandated in backbone and edge routers
  - Appears in RFPs and IETF architectural guidelines
  - Already known by inventors of TCP
    - [Van Jacobson, 1988]
  - Has major consequences for router design

The Story So Far

# packets at 10Gb/s	1,000,000	10,000	20

$$2T \times C \xrightarrow{(1)} \frac{2T \times C}{\sqrt{n}} \xrightarrow{(2)} O(\log W)$$

(1) Assume: Large number of desynchronized flows; 100% utilization
(2) Assume: Large number of desynchronized flows; <100% utilization

Using NetFPGA to explore buffer size

- **Need to reduce buffer size and measure occupancy**
- **Alas, not possible in commercial routers**
- **So, we will use the NetFPGA instead**

Objective:

– Use the NetFPGA to understand how large a buffer we need for a **single** TCP flow.

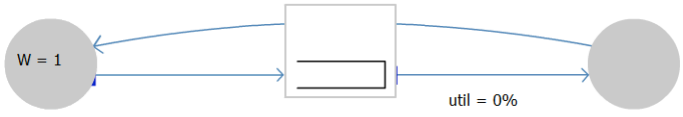
Why 2TxC for a single TCP Flow?

Only W packets may be outstanding

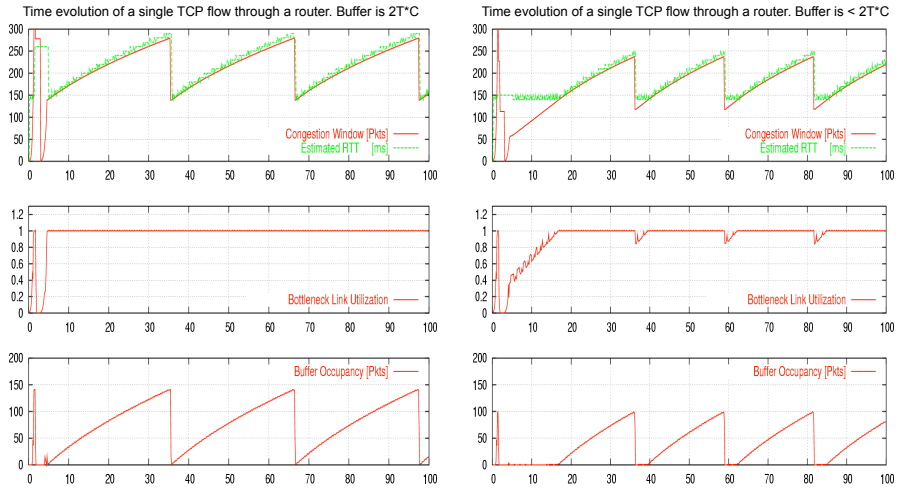
Rule for adjusting W

– If an ACK is received: W ← W+1/W

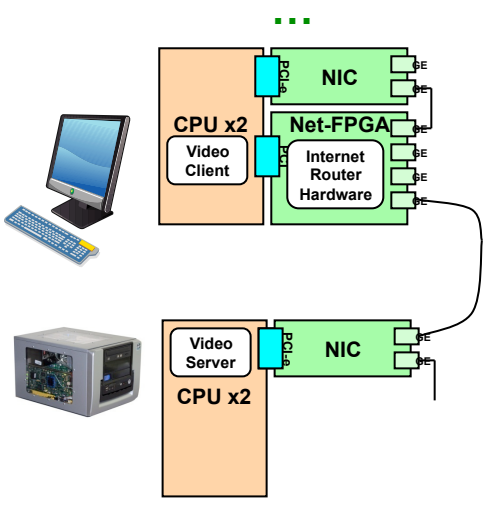
– If a packet is lost: W ← W/2



Time Evolution of a Single TCP Flow



NetFPGA Hardware Set for Demo #2



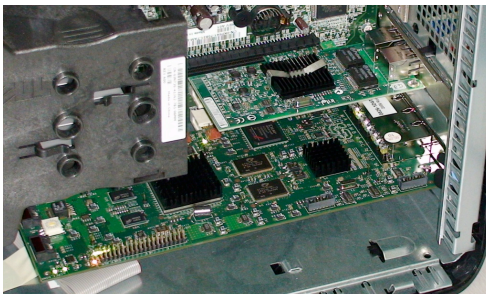
Server delivers streaming HD video to adjacent client

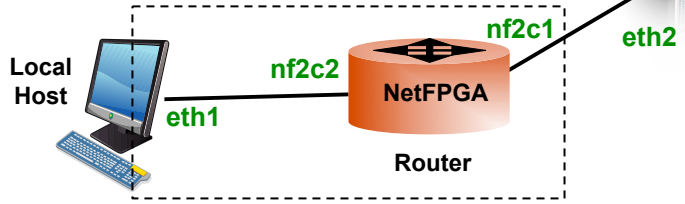
Demo 2

Observing and Controlling the Queue Size

Demo 2

Setup for the Demo 2

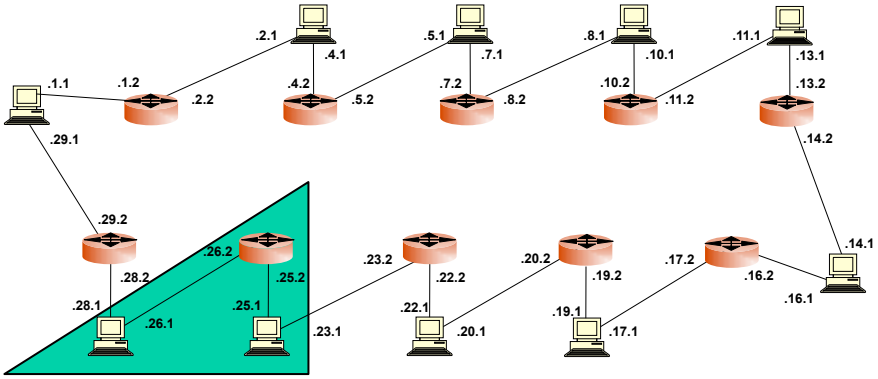




Demo 2

Interfaces and Subnets

- eth1 connects your host to your NetFPGA Router
- nf2c2 routes to nf2c1 (your adjacent server)
- eth2 serves web and video traffic to your neighbor
- nf2c0 & nf2c3 (the network ring) are unused

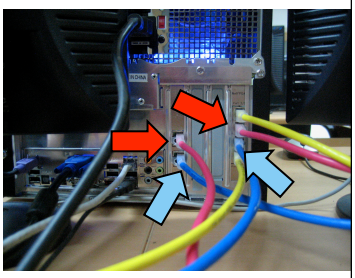


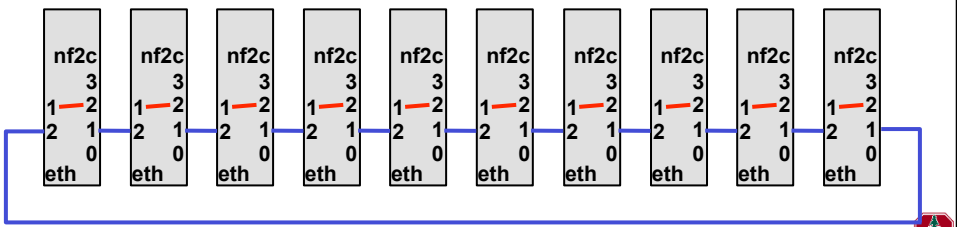
This configuration allows you to modify and test your router without affecting others

Demo 2

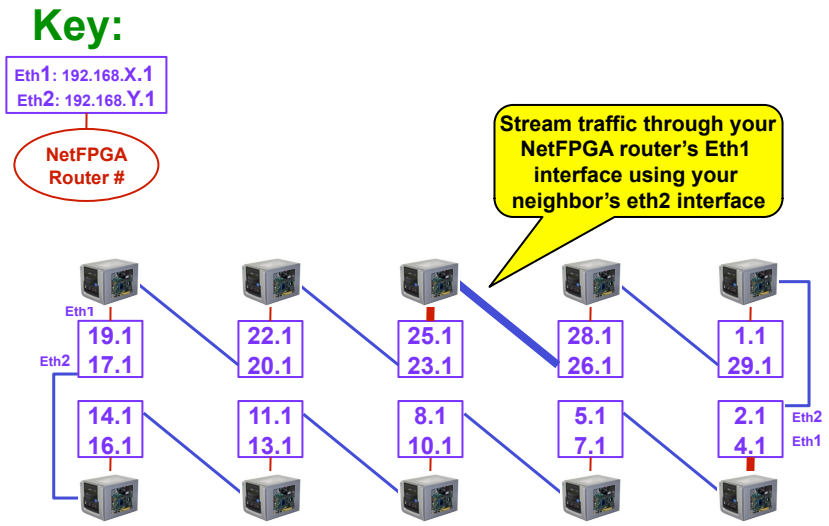
Cable Configuration for Demo 2

- NetFPGA Gigabit Ethernet Interfaces
  - nf2c2 : Local host interface (red)
  - nf2c1 : Router for adjacent server (blue)
- Host Ethernet Interfaces
  - eth1 : Local host interface (red)
  - eth2 : Server for neighbor (blue)





Demo 2 Configuration



Demo 2

Enhanced Router

Objectives

- Observe router with new modules
- New modules: rate limiting, event capture

Execution

- Run event capture router
- Look at routing tables
- Explore details pane
- Start tcp transfer, look at queue occupancy
- Change rate, look at queue occupancy

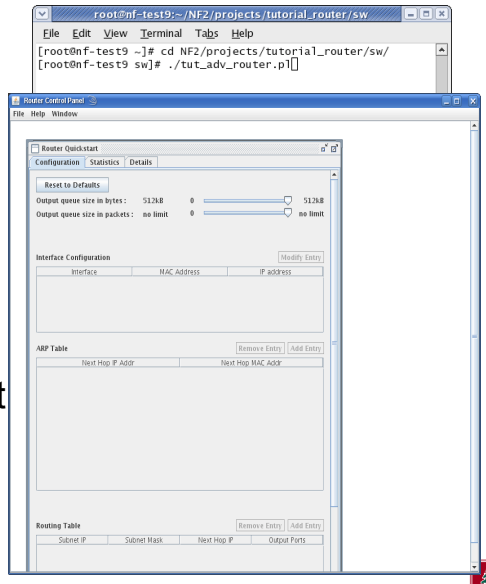
Demo 2

Step 1 - Run Pre-made Enhanced Router

Start terminal and cd to "NF2/projects/tutorial_router/sw/"

Type "./tut_adv_router_gui.pl"

A familiar GUI should start

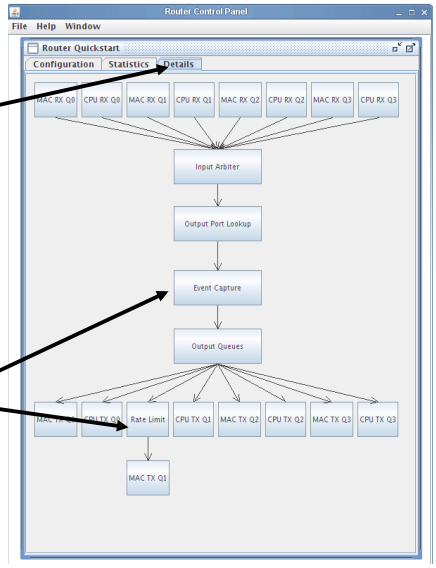


Demo 2

Step 2 - Explore Enhanced Router

Click on the Details tab

A similar pipeline to the one seen previously shown with some additions

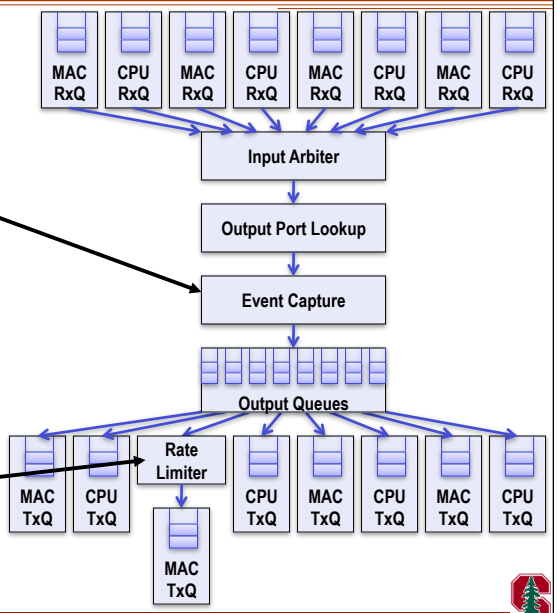


Demo 2

Enhanced Router Pipeline

Two modules added

1. **Event Capture** to capture output queue events (writes, reads, drops)
2. **Rate Limiter** to create a bottleneck

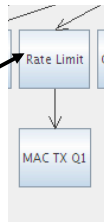


Demo 2

Step 3 - Decrease the Link Rate

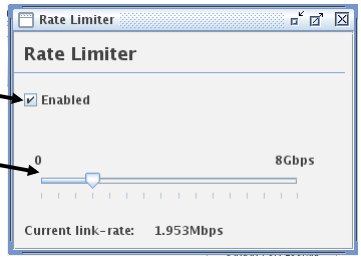
To create bottleneck and show the TCP "sawtooth," link-rate is decreased.

In the Details tab, click the "Rate Limit" module



Check Enabled

Set link rate to 1.953Mbps



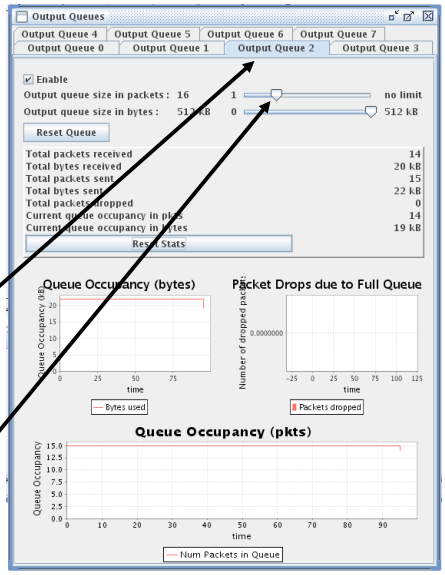
Demo 2

Step 4 – Decrease Queue Size

Go back to the Details panel and click on "Output Queues"

Select the "Output Queue 2" tab

Change the output queue size in packets slider to 16

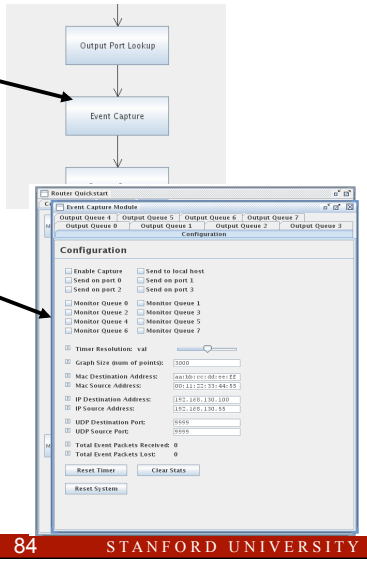


Demo 2

Step 5 - Start Event Capture

Click on the Event Capture module under the Details tab

This should start the configuration page



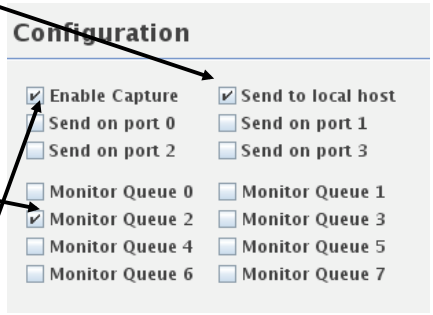
Demo 2

Step 6 - Configure Event Capture

Check **Send to local host** to receive events on the local host

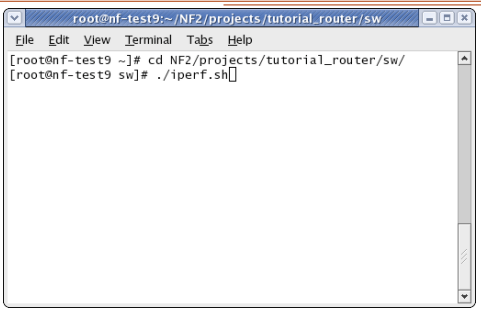
Check **Monitor Queue 2** to monitor output queue of MAC port1

Check **Enable Capture** to start event capture



Demo 2

Step 7 - Start TCP Transfer



We will use *iperf* to run a large TCP transfer and look at queue evolution

Start a terminal and cd to "NF2/projects/tutorial_router/sw"

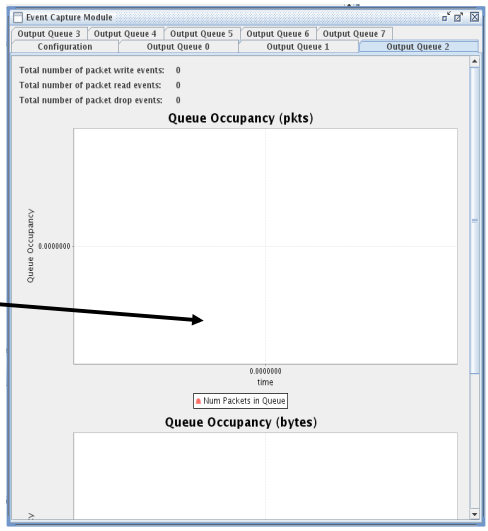
Type "./iperf.sh"

Demo 2

Step 8 - Look at Event Capture Results

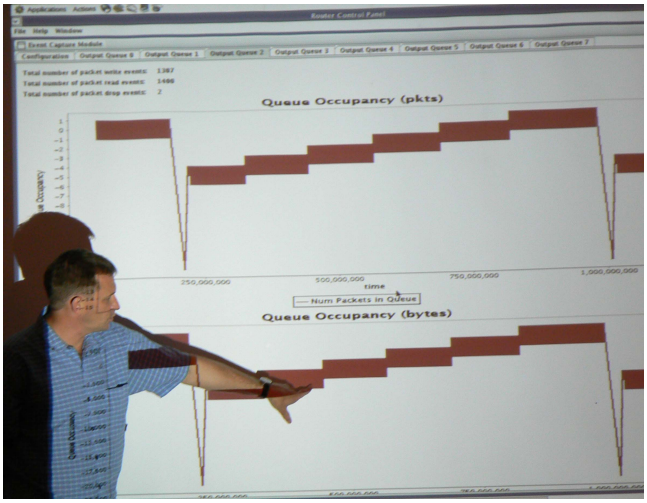
Click on the Event Capture module under the Details tab.

The sawtooth pattern should now be visible.



Queue Occupancy Charts

Observe the TCP/IP sawtooth



Leave the control windows open

Exercise 2: Enhancing the Reference Router

Exercise 2

Enhance Your Router

Objectives

- Add new modules to datapath
- Synthesize and test router

Execution

- Open user_datapath.v, uncomment delay/ rate/event capture modules
- Synthesize
- After synthesis, test the new system

An aside: xemacs Tips

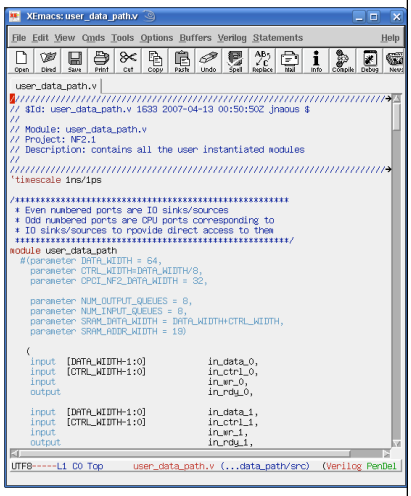
We will modify Verilog source code with xemacs

- To undo a command, type
  - ctrl+shift+'-'
- To cancel a multi-keystroke command, type
  - ctrl+g
- To select lines,
  - hold shift and press the arrow keys
- To comment (remove from compilation) selected lines, type
  - ctrl+c+c
- To uncomment a commented block,
  - move the cursor inside the commented block
  - type ctrl+c+u
- To save, type
  - ctrl+x+s
- To search for a term, type
  - ctrl+s search_pattern

Exercise 2

Step 1 - Open the Source

We will modify the Verilog source code to add event capture and rate limiter modules



We will simply comment and uncomment existing code

Open terminal

Type

xemacs NF2/projects/tutorial_router/src/user_data_path.v

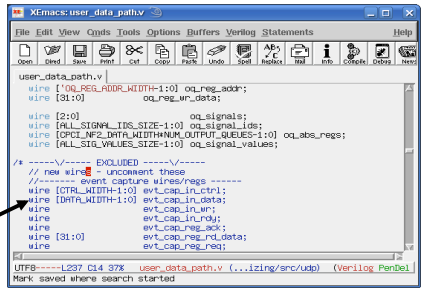
Exercise 2

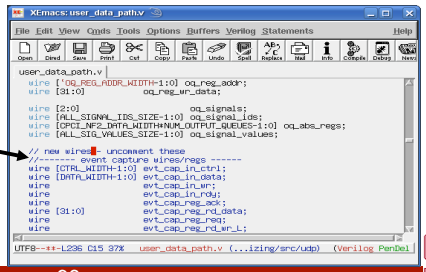
Step 2 - Add Wires

Now we need to add wires to connect the new modules

Search for “new wires” (ctrl +s new wires), then press Enter

Uncomment the wires (ctrl +c+u)





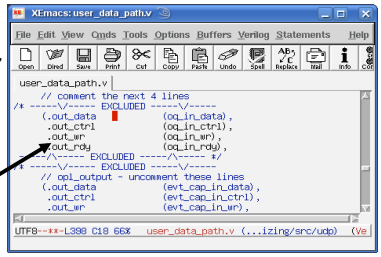
Exercise 2

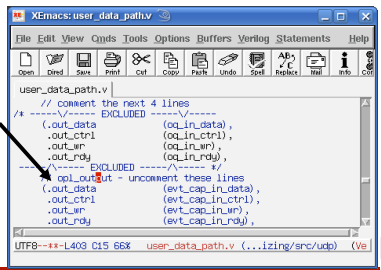
Step 3a - Connect Event Capture

Search for opl_output (ctrl+s opl_output), then press Enter

Comment the four lines above (up, shift + up + up + up + up, ctrl+c+c)

Uncomment the block below to connect the outputs (ctrl+s opl_out, ctrl+c+u)





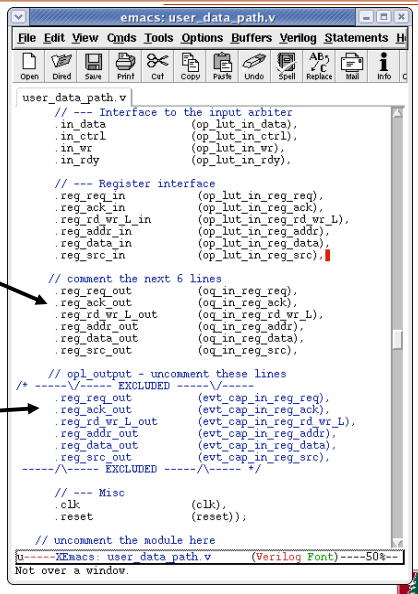
Exercise 2

Step 3b - Connect the Output Queue Registers

Search for opl_output (ctrl+s opl_output, Enter)

Comment the 6 lines (select the six lines by using shift+arrow keys, then type ctrl+c+c)

Uncomment the commented block by scrolling down into the block and typing ctrl+c +u

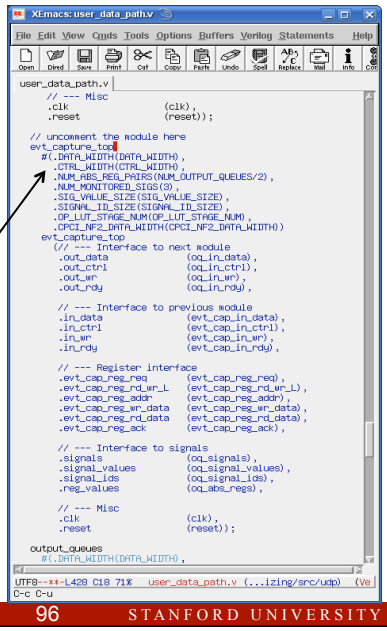


Exercise 2

Step 4 - Add the Event Capture Module

Search for evt_capture_top (ctrl+s evt_capture_top), then press Enter

Uncomment the block (ctrl +c+u)

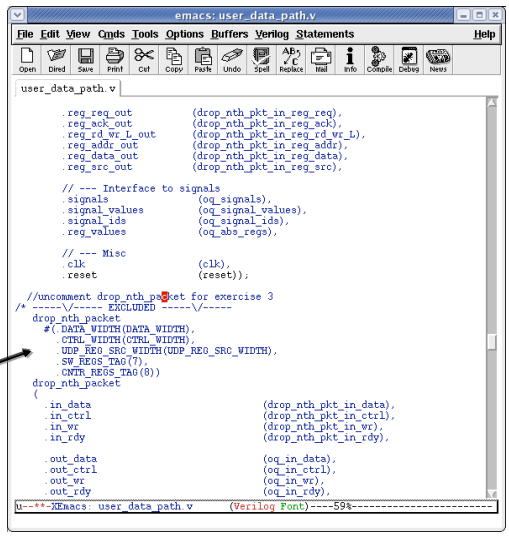


Exercise 2

Step 5 - Add the Drop Nth Module

Search for drop_nth_packet (ctrl+s drop_nth_packet), then press Enter

Uncomment the block (ctrl +c+u)



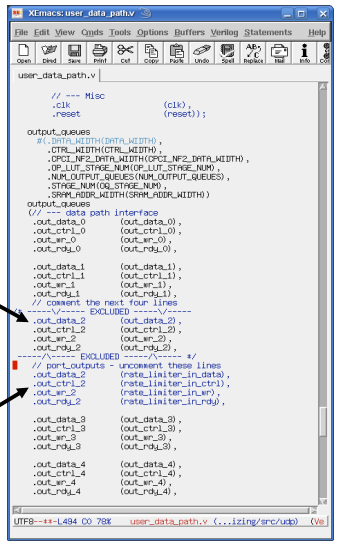
Exercise 2

Step 6 - Connect the Output Queue to the Rate Limiter

Search for port_outputs (ctrl +s port_outputs), then press (Enter)

Comment the 4 lines above (select the four lines by using shift+arrow keys), then type (ctrl+c+c)

Uncomment the commented block by scrolling down into the block and typing ctrl+c +u



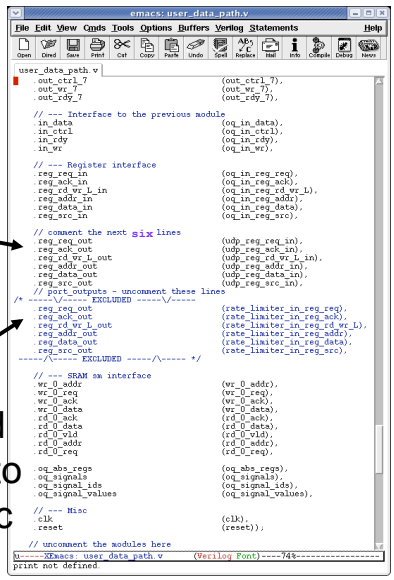
Exercise 2

Step 7 - Connect the Registers

Search for port_outputs (ctrl +s port_outputs), then press (Enter)

Comment the 6 lines (select the six lines by using shift+arrow keys), then type (ctrl+c+c)

Uncommon the commented block by scrolling down into the block and typing (ctrl+c +u)



Exercise 2

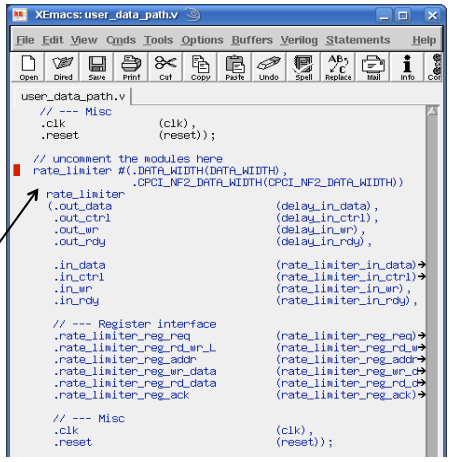
Step 8 - Add Rate Limiter

Scroll down until you reach the next “excluded” block

Uncommon the block containing the rate limiter instantiations.

Scroll into the block, type (ctrl+c+u)

Save (ctrl+x+s)



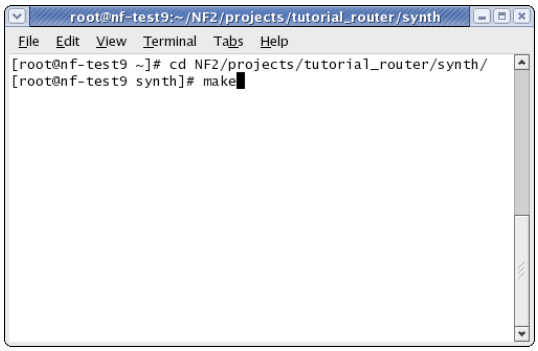
Exercise 2

Step 9 - Build the Hardware

Start terminal, cd to “NF2/projects/tutorial_router/synth”

Run “make clean”

Start synthesis with “make”

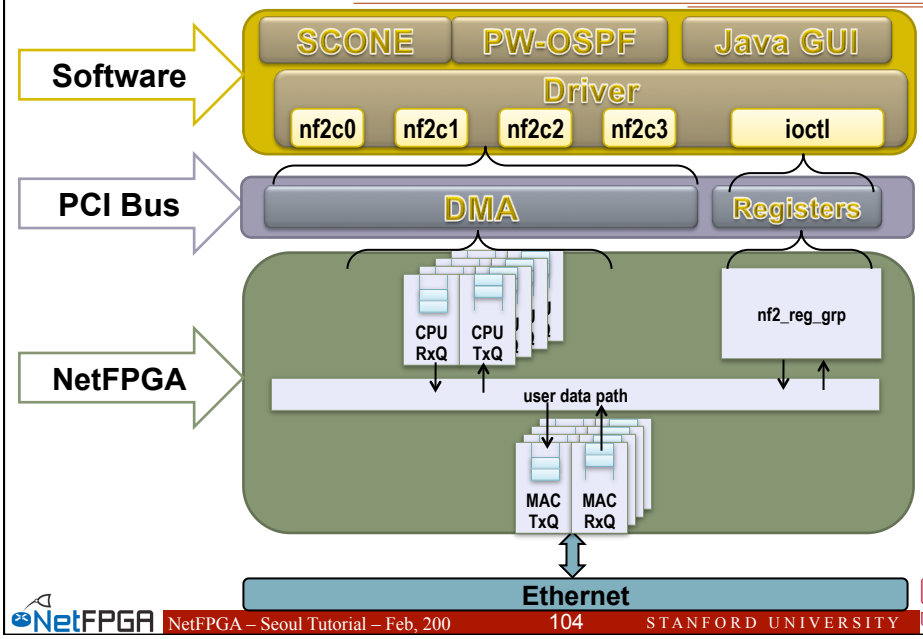


Second Break

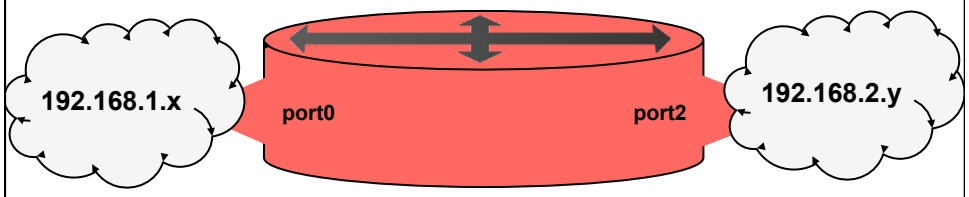
(while hardware compiles)

Hardware Datapath

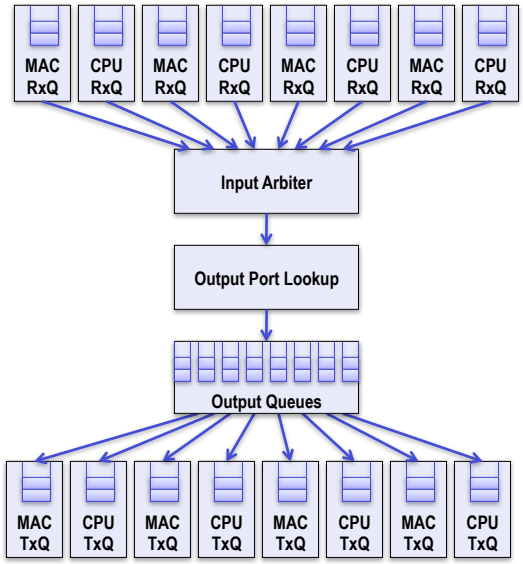
Full System Components



Life of a Packet through the Hardware



Router Stages Again



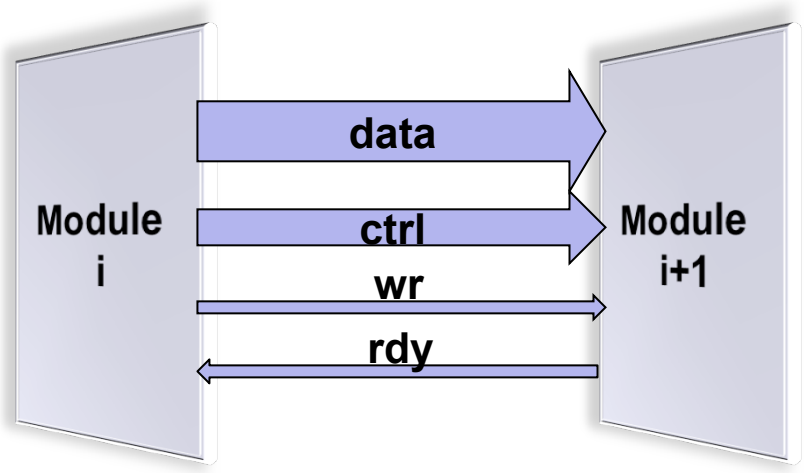
Inter-Module Communication

Using "Module Headers":

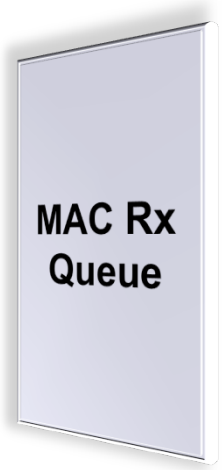
Ctrl Word (8 bits)	Data Word (64 bits)
x	Module Hdr
…	…
y	Last Module Hdr
0	Eth Hdr
0	IP Hdr
0	…
0x10	Last word of packet

Contain information such as packet length, input port, output port, …

Inter-Module Communication



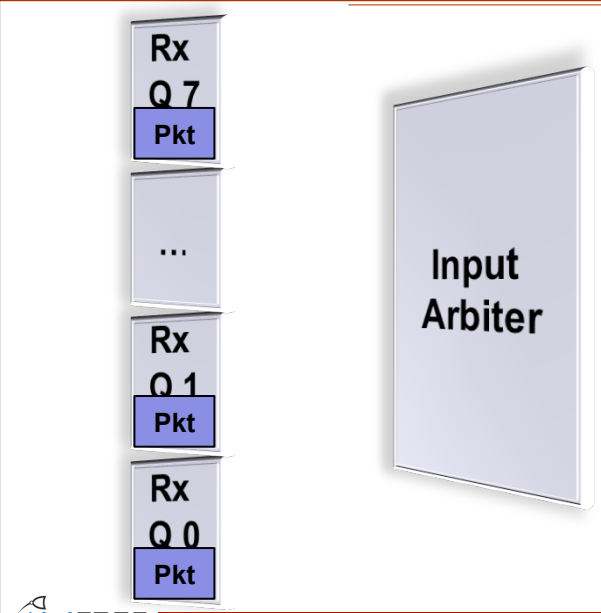
MAC Rx Queue



Rx Queue

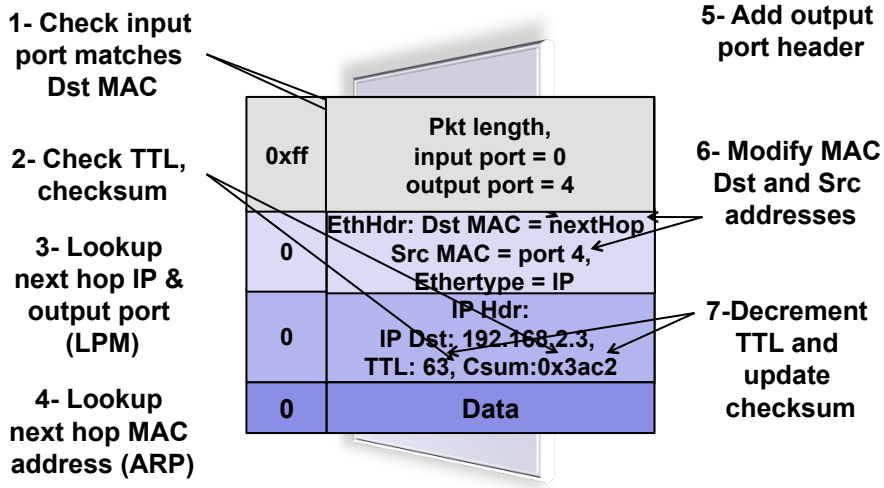
0xff	Pkt length, input port = 0
0	Eth Hdr: Dst MAC = port 0, Ethertype = IP
0	IP Hdr: IP Dst: 192.168.2.3, TTL: 64, Csum:0x3ab4
0	Data

Input Arbiter

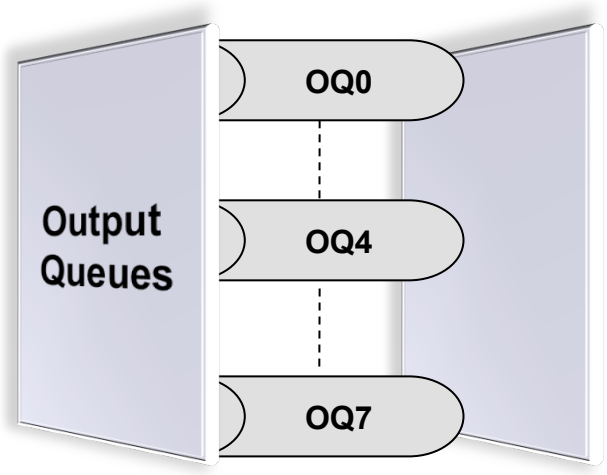


Output Port Lookup

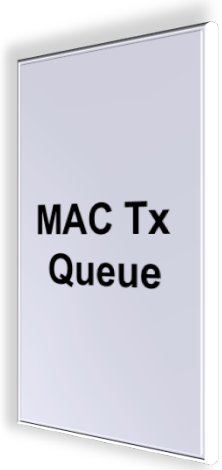
Output Port Lookup



Output Queues



MAC Tx Queue



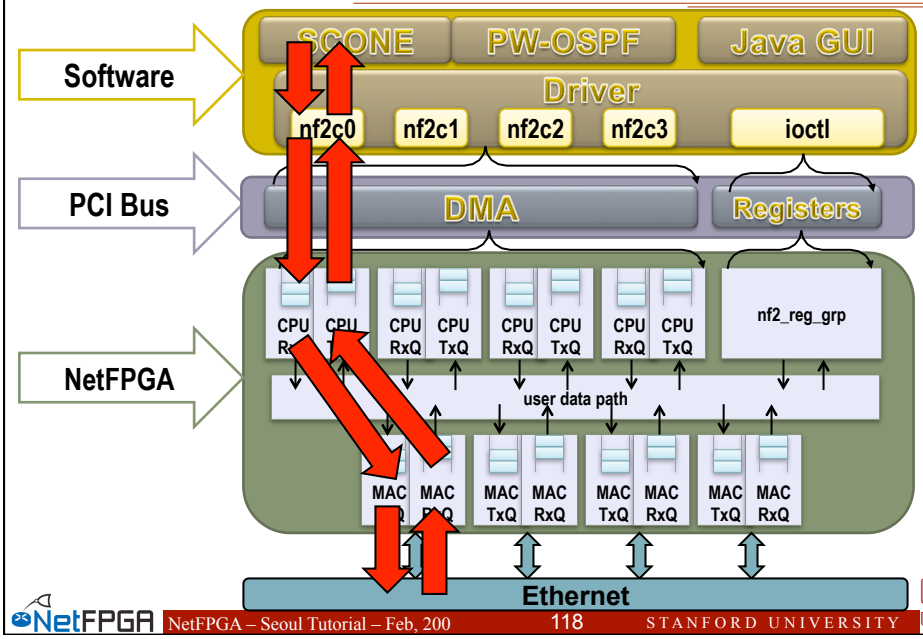
MAC Tx Queue

0xff	Pkt length, input port = 0 output port = 4
0	EthHdr: Dst MAC = nextHop Src MAC = port 4, Ethertype = IP
0	IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2
0	Data

Exception Packet

- Example: TTL = 0 or TTL = 1
- Packet has to be sent to the CPU which will generate an ICMP packet as a response
- Difference starts at the Output Port lookup stage

Exception Packet Path

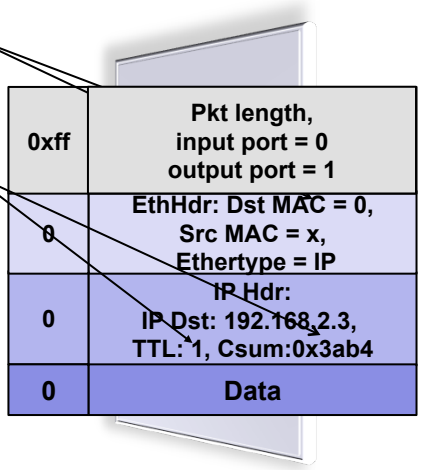


Output Port Lookup

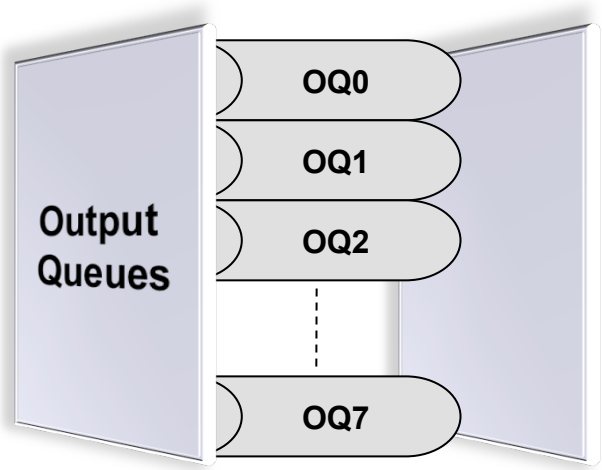
1- Check input port matches Dst MAC

2- Check TTL, checksum – EXCEPTION!

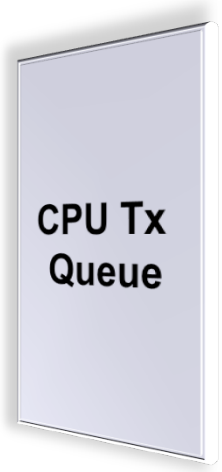
3- Add output port module



Output Queues



CPU Tx Queue



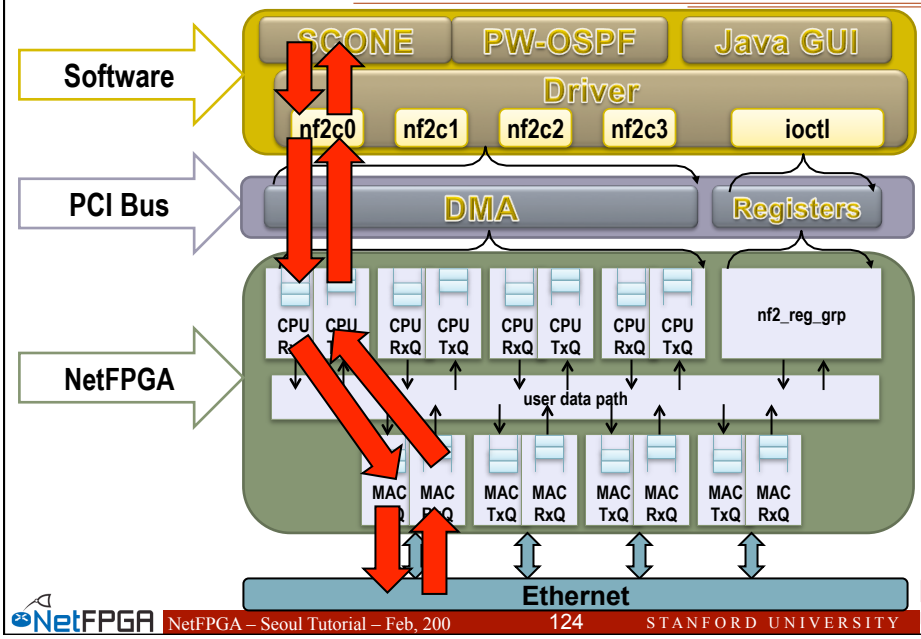
CPU Tx Queue

0xff	Pkt length, input port = 0 output port = 1
0	EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP
0	IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4
0	Data

ICMP Packet

- For the ICMP packet, the packet arrives at the CPU Rx Queue from the PCI Bus
- It follows the same path as a packet from the MAC until it reaches the Output Port Lookup
- The OPL module sees the packet is from the CPU Rx Queue 1 and sets the output port directly to 0
- The packet then continues on the same path as the non-exception packet to the Output Queues and then MAC Tx queue 0

ICMP Packet Path



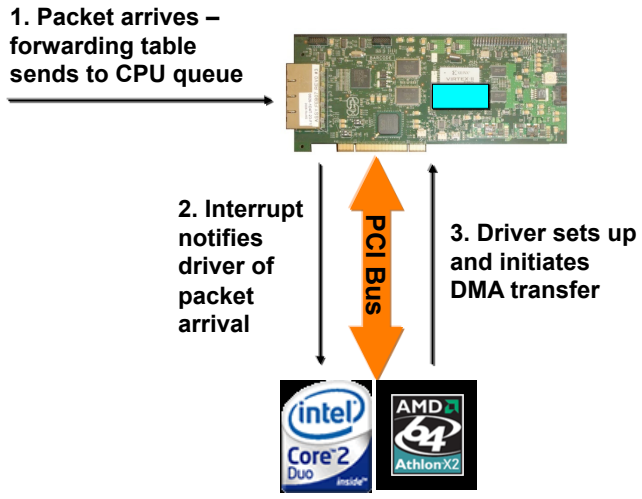
NetFPGA-Host Interaction

- **Linux driver interfaces with hardware**
  - Packet interface via standard Linux network stack
  - Register reads/writes via ioctl system call with wrapper functions:
    - readReg(nf2device *dev, int address, unsigned *rd_data);
    - writeReg(nf2device *dev, int address, unsigned *wr_data);

    eg:
    readReg(&nf2, OQ_NUM_PKTS_STORED_0, &val);

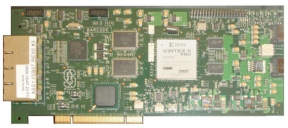
NetFPGA-Host Interaction

NetFPGA to host packet transfer



NetFPGA-Host Interaction

NetFPGA to host packet transfer (cont.)



4. NetFPGA transfers packet via DMA

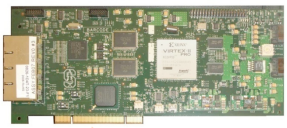
PCI Bus

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

PCI Bus

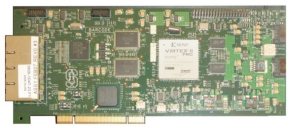
3. Interrupt signals completion of DMA

1. Software sends packet via network sockets

Packet delivered to driver

NetFPGA-Host Interaction

Register access



2. Driver performs PCI memory read/write

1. Software makes ioctl call on network socket

ioctl passed to driver

NetFPGA-Host Interaction

- **Packet transfers shown using DMA interface**
- **Alternative: use programmed IO to transfer packets via register reads/writes**
  - slower but eliminates the need to deal with network sockets

Exercise 2

Step 10 – Perfect the Router

Go back to "Demo 2: Step 1" after synthesis completes and redo the steps with your own router

To run your router:

1- cd NF2/projects/tutorial_router/sw

2- type "./tut_adv_router_gui.pl --use_bin ../../../bitfiles/tutorial_router.bit"

You can change the bandwidth and queue size settings to see how that affects the evolution of queue occupancy

Exercise 3

Drop 1 in N Packets

Objectives

- Add counter and FSM to the code
- Synthesize and test router

Execution

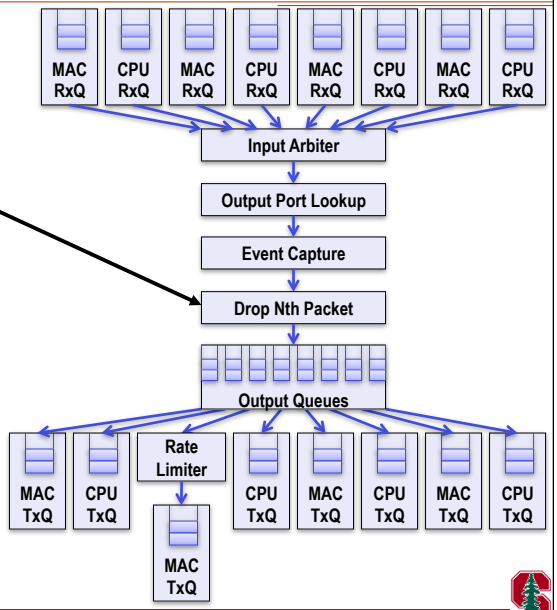
- Open drop_nth_packet.v
- Insert counter code
- Synthesize
- After synthesis, test the new system.

Exercise 3

New Reference Router Pipeline

One module added

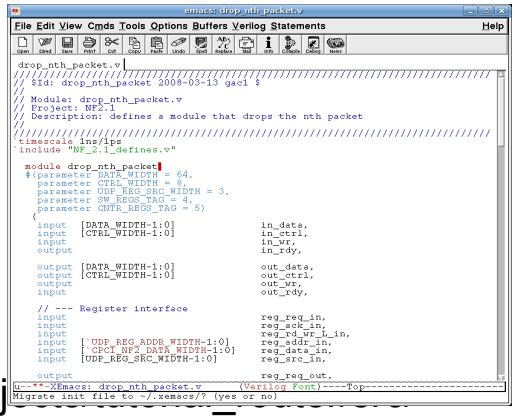
1. **Drop Nth Packet**
   to drop every Nth packet from the reference router pipeline



Exercise 3

Step 1 - Open the Source

We will modify the Verilog source code to add a counter to the drop_nth_packet module



Open terminal

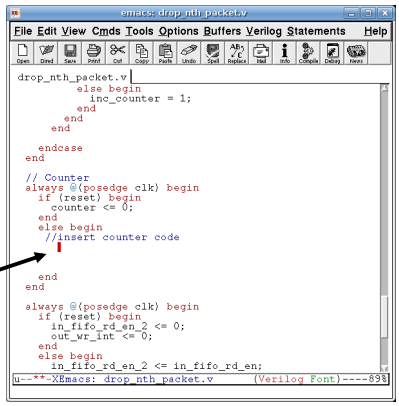
Type “xemacs NF2/proj drop_nth_packet.v

Exercise 3

Step 2 - Add Counter to Module

Add counter using the following signals:

- `counter`
  - –16 bit output signal that you should increment on each packet pulse
- `rst_counter`
  - – reset signal (a pulse input)
- `inc_counter`
  - – increment (a pulse input)



Search for insert counter

(ctrl+s insert counter, Enter)

Insert counter and save

(ctrl+x+s)

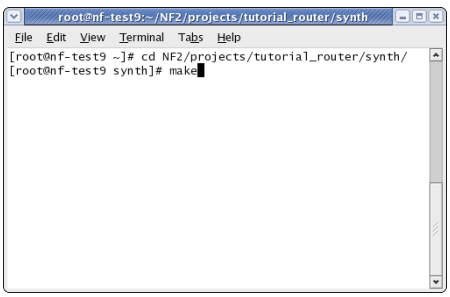
Exercise 3

Step 3 - Build the Hardware

Start terminal, cd to “NF2/projects/tutorial_router/synth”

Run “make clean”

Start synthesis with “make”



Using the NetFPGA in the Classroom

NetFPGA in the Classroom

- **Stanford University**
  - EE109 "Build an Ethernet Switch"
    - Undergraduate course for all EE students
    - http://www.stanford.edu/class/ee109/
  - CS344 "Building an Internet Router"
    - Quarter-long course
    - Taught Spr'05, Spr'07, Spr'08
    - http://cs344.stanford.edu
- **Rice University**
  - Network Systems Architecture
    - Spr'08
    - http://comp519.cs.rice.edu/
- **Cambridge University**
  - "Build an Internet Router"
    - Michaelmas'09
    - http://www.cl.cam.ac.uk/admissions/acs/modules/#bir

See: http://netfpga.org/netfpgawiki/index.php/Teachers

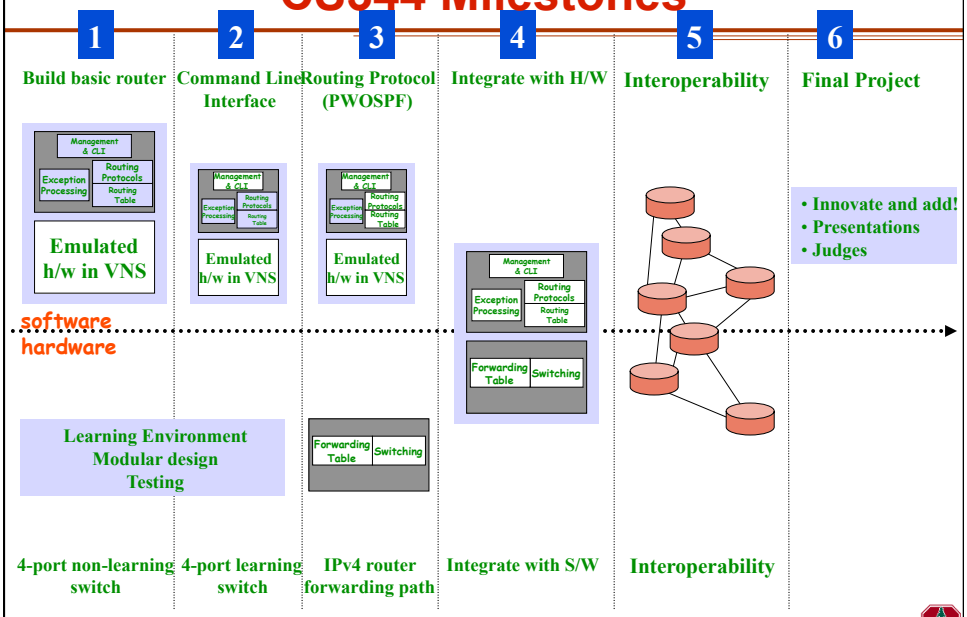
Components of NetFPGA Course

- **Documentation**
  - System Design
  - Implementation Plan
- **Deliverables**
  - Hardware Circuits
  - System Software
  - Milestones
- **Testing**
  - Proof of Correctness
  - Integrated Testing
  - Interoperabilty
- **Post Mortem**
  - Lessons Learned

NetFPGA in the Classroom

- **Stanford CS344: "Build an Internet Router"**
  - Courseware available on-line
  - Students work in teams of three
    - 1-2 software
    - 1-2 hardware
  - Design and implement router in 8 weeks
  - Write software for CLI and PW-OSPF
  - Show interoperability with other groups
  - Add new features in remaining two weeks
    - Firewall, NAT, DRR, Packet capture, Data generator, …

CS344 Milestones



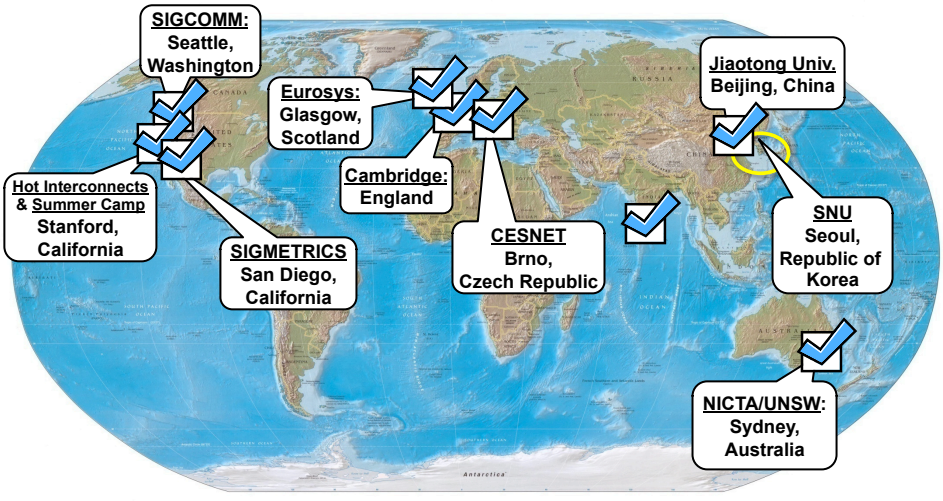
Typical NetFPGA Course Plan

Week	Software	Hardware	Deliver
1	Verify Software Tools	Verify CAD Tools	Write Design Document
2	Build Software Router	Build Non-Learning Switch	Run Software Router
3	Cmd. Line Interface	Build Learning Switch	Run Basic Switch
4	Router Protocols	Output Queues	Run Learning Switch
5	Implement Protocol	Forwarding Path	Interface SW & HW
6	Control Hardware	Hardware Registers	HW/SW Test
7	Interoperate Software & Hardware		Router Submission
8	Plan New Advanced Feature		Project Design Plan
9	Show new Advanced Feature		Demonstration

CS344 Project Presentations

http://cs344.stanford.edu

NetFPGA Worldwide Tutorial Series



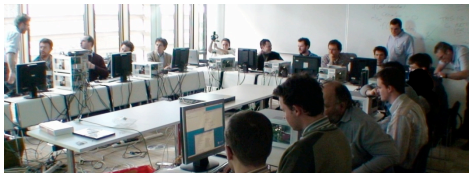
Photos from NetFPGA Tutorials

SIGCOMM - Seattle, Washington, USA

Beijing, China

SIGMETRICS - San Diego, California, USA

Bangalore, India



EuroSys - Glasgow, Scotland, U.K.

http://netfpga.org/php/events.php

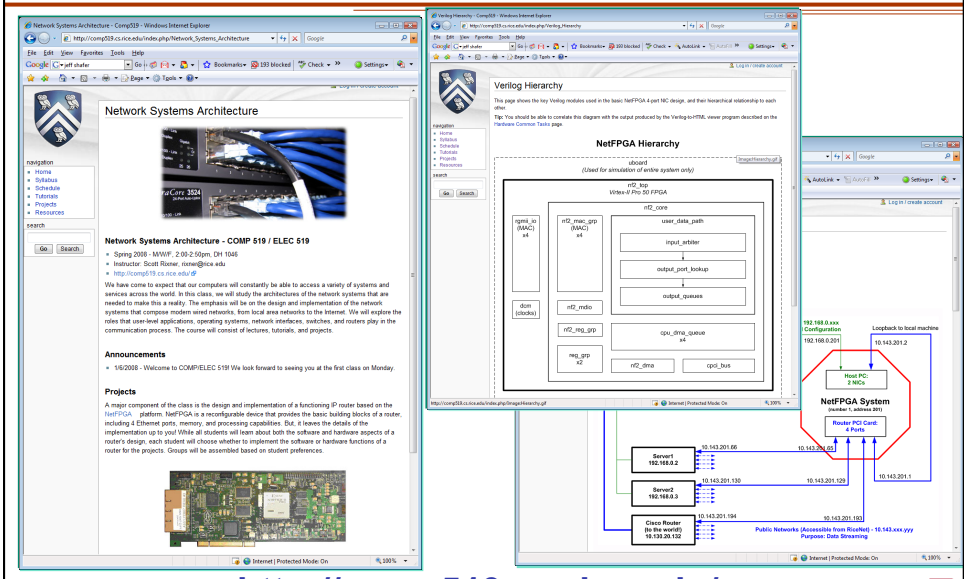
Deployed NetFPGA hardware

(July 2008)

- Princeton University
- Rice University
- Georgia Tech
- Washington University
- University of Utah
- University of Toronto
- University of Wisconsin
- University of Connecticut
- University of California, San Diego (UCSD)
- University of California, Los Angeles (UCLA)
- University of Idaho
- University of Massachusetts (UMass)
- University of Pennsylvania (UPenn)
- North Carolina State University
- Lehigh University
- State University of New York (SUNY), Buffalo
- State University of New York (SUNY), Binghamton
- University of Florida
- Rutgers
- Western New England College
- Emerson Network Power
- ICSI
- Agilent
- Cisco
- Quanta Computer, Inc.
- Zones Inc.
- Cambridge University
- India Institute of Science (IISc), Bangalore
- Ecole Polytechnique de Montreal
- Beijing Jaiotong University
- China Zhejiang University
- National Taiwan University
- University of New South Wales
- University of Hong Kong
- University of Sydney
- University of Bologna
- University of Naples
- University of Pisa, Italy
- University of Quebec
- University of Jinan
- University of Amsterdam
- University of Waterloo
- University of Victoria
- Chung Yuan Christan University, Taiwan (CYCU)
- Universite de Technologie de Compiegne (UTC)
- Catholic University of Rio De Janeiro
- University Leiden (The Netherlands)
- National United University
- Kookman University (South Korea)
- Kasetsart University (Thailand)
- Helsinki Institute for Information Technology (HIIT)
- CESNET

Map of Deployed Hardware
(July 2008)

Network Systems Architecture at Rice



http://comp519.cs.rice.edu/

Networked FPGAs in Research

1. **Managed flow-table switch**
   - http://OpenFlowSwitch.org/
2. **Buffer Sizing**
   - Reduce buffer size & measure buffer occupancy
3. **RCP: Congestion Control**
   - New module for parsing and overwriting new packet
   - New software to calculate explicit rates
4. **Deep Packet Inspection (FPX)**
   - TCP/IP Flow Reconstruction
   - Regular Expression Matching
   - Bloom Filters
5. **Packet Monitoring (ICSI)**
   - Network Shunt

11. **Precise Time Protocol (PTP)**
   - Synchronization among Routers

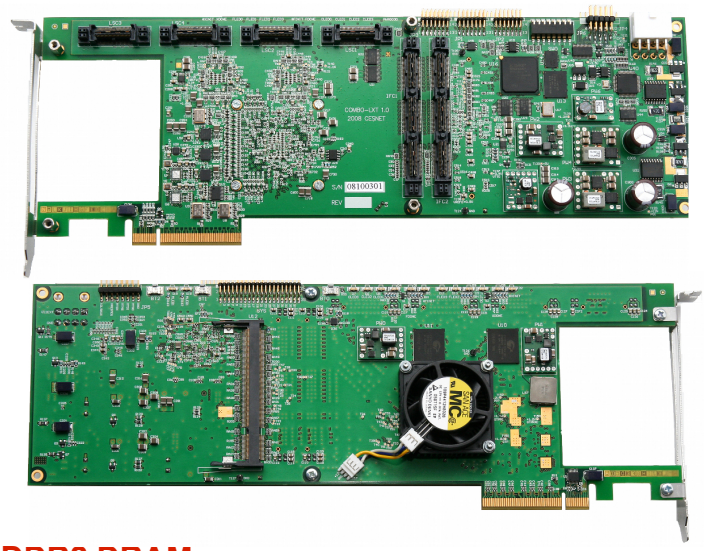
CESNET / Liberouter

- **FPGA cards and accelerated network solutions :** www.liberouter.org
  - CESNET
  - Masaryk University
  - Brno University of Technology
- **Hardware cards**
  - Existing 1/10 Gbps family of COMBO cards
  - Upcoming 10/100 Gbps family of COMBOv2 cards
- **NetCOPE platform for rapid development of network applications**
  - PCI Express support, fast DMA transfers, pre-processed input packets
  - Extensive tool set for data streams manipulation
  - Cooperation with NetFPGA to create compatible application interface
- **Network applications**
  - NetFlow monitoring – collecting network statistics about flows
  - Payload checking – scanning content of packet payload
  - Hardware filtration and forwarding
- **Commercial spin-off company INVEA-TECH :** www.invea-tech.com

CESNET UNIVERSITAS MASARYKIANA BRUNENSIS FIT invea tech

CESNET – COMBO-LXT

- Virtex 5 LXT-110 FPGA
- PCIe x 8 Host Interface
- Supports multiple 10 Gbps (and faster) interfaces
- QDR-II Memory
- SO-DIMM for DDR2 DRAM



Third Break

(while hardware compiles)

Exercise 3

Step 5 – Test your Router

You can watch the number of received and sent packets to watch the module drop every Nth packet. Ping a local machine (i.e. 192.168.7.1) and watch for missing pings

To run your router:

1- Enter the directory by typing:

cd NF2/projects/tutorial_router/sw

2- Run the router by typing:

./tut_adv_router_gui.pl --use_bin ../../../bitfiles/tutorial_router.bit

To set the value of N (which packet to drop)

type regwrite 0x2000704 N

– replace N with a number (such as 100)

To enable packet dropping, type:

regwrite 0x2000700 0x1

To disable packet dropping, type:

regwrite 0x2000700 0x0

Exercise 3

Step 5 – Measurements

- **Determine iperf TCP throughput to neighbor's server for each of several values of N**
  - Similar to Demo 2, Step 8
  - Ping 192.168.x.2 (where x is your neighbor's server)
  - TCP throughput with:
    - Drop circuit disabled
      - TCP Throughput = _____ Mbps
    - Drop one in N = 1,000 packets
      - TCP Throughput = _____ Mbps
    - Drop one in N = 100 packets
      - TCP Throughput = _____ Mbps
    - Drop one in N = 10 packets
      - TCP Throughput = _____ Mbps
- **Explain why TCPs throughput is so low given that only a tiny fraction of packets are lost**

Visit http://NetFPGA.org

Join the NetFPGA.org Community

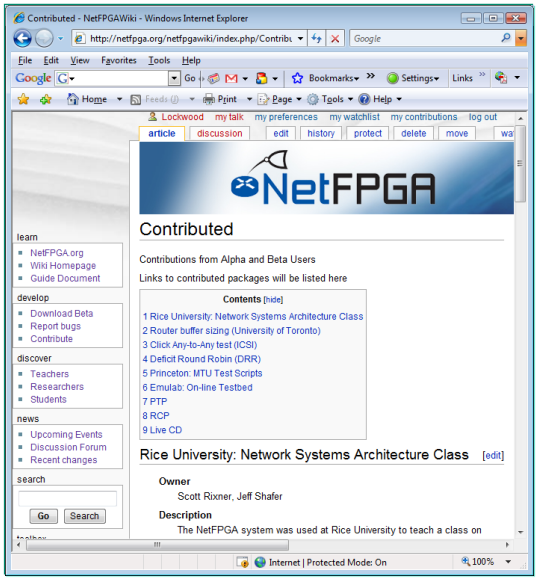
- Log into the Wiki
- Access the Beta code
- Join the netfpga-beta mailing list
- Join the discussion forum

Learn from the On-line Guide

- Obtain hardware, software, & gateware
- Install software, CAD tools, & simulation models
- Verify installation using regression self-tests
- Walk through the reference designs
- Learn about contributed packages

Contribute to the Project

- Search for related work
- List your project on the Wiki
- Link your project homepage



Project Ideas for the NetFPGA

- IPv6 Router (in high demand)
- TCP Traffic Generator
- Valiant Load Balancing
- Graphical User Interface (like CLACK)
- MAC-in-MAC Encapsulation
- Encryption / Decryption modules
- RCP Transport Protocol
- Packet Filtering (Firewall, IDS, IDP)
- TCP Offload Engine
- DRAM Packet Queues
- 8-Port Switch using SATA Bridge
- Build our own MAC (from source, rather than core)
- Use XML for Register Definitions

http://netfpga.org/netfpgawiki/index.php/Module_Wishlist

Group Discussion

- Your plans for using the NetFPGA
  - Teaching
  - Research
  - Other
- Resources needed for your class
  - Source code
  - Courseware
  - Examples
- Your plans to contribute
  - Expertise
  - Capabilities
  - Collaboration Opportunities

Survey

- **How did you like this this tutorial?**
  - What did you find useful?
  - What should be improved?
  - What should be removed?
  - What should be added?
- **Can we post the video from this event?**
  - If not, please let us know.
- **Complete On-line survey**
  - http://netfpga.org/tutorial_survey.html

Acknowledgments

NetFPGA Team at Stanford University: May 2008

Jianying Luo, Jad Naous, Nick McKeown, Glen Gibb, G. Adam Covington, David Erickson, John W. Lockwood, Brandon Heller

Not Shown in photo: Paul Hartke, Neda Beheshti, Sara Bolouki

Special thanks to:

Patrick Lysaght, Veena Kumar, Paul Hartke, Anna Acevedo
Xilinx University Program (XUP)

XILINX®

Other NetFPGA Tutorial Presented At:

HOT INTERCONNECTS
UNIVERSITY OF CAMBRIDGE
CESNET
NICTA
SIGMETRICS
acm
THE UNIVERSITY OF NEW SOUTH WALES
UNIVERSITY of GLASGOW
SIGCOMM 2008 SEATTLE
FIT
Indian Institute of Science Bangalore, India
BEIJING JIAOTONG UNIVERSITY

See: http://netfpga.org/php/events.php

Acknowledgments

- **Support for the NetFPGA project has been provided by the following companies and institutions**

CISCO
XILINX®
HUAWEI
Google
NATIONAL SCIENCE FOUNDATION
Agilent Technologies
DIGILENT
Juniper NETWORKS
CYPRESS PERFORM
BROADCOM.
Synplicity
Micron

***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.*