Welcome

Please organize into teams
2 or 3 People/computer

Printed Slides are available
Slides are also available online

The NetFPGA machines
Username: root  Password: on whiteboard

NetFPGA homepage
http://NetFPGA.org
Tutorial Outline

- **Introduction**
  - Motivation
- **Exercise 1: Exploring the Reference Router**
  - Network Review: Basics of an IP Router
  - Demo 1: Reference Router running on the NetFPGA
- **Exercise 2: Enhancing the Reference Router**
  - Hardware: NetFPGA Platforms: 1G and 10G
  - Problem: Understanding buffer size requirements in a router
- **Exercise 3: Drop 1 in N Packets**
- **Concluding Remarks**
  - What next for you?
  - Group Discussion

Motivation
NetFPGA = Networked FPGA

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

Four elements:

• NetFPGA board
• Tools + reference designs
• Contributed projects
• Community
## NetFPGA Board Comparison

<table>
<thead>
<tr>
<th></th>
<th>NetFPGA 1G</th>
<th>NetFPGA 10G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet Ports</td>
<td>4 x 1Gbps</td>
<td>4 x 10Gbps SFP+</td>
</tr>
<tr>
<td>ZBT SRAM</td>
<td>4.5 MB</td>
<td>27 MB QDRII-SRAM</td>
</tr>
<tr>
<td>DDR2 SDRAM</td>
<td>64 MB</td>
<td>288 MB RLDRAM-II</td>
</tr>
<tr>
<td>PCI</td>
<td>Virtex II-Pro 50</td>
<td>PCI Express x8</td>
</tr>
<tr>
<td>Virtex</td>
<td>Virtex 5 TX240T</td>
<td></td>
</tr>
</tbody>
</table>

## NetFPGA board

- Networking Software running on a standard PC
- A hardware accelerator built with Field Programmable Gate Array driving Gigabit network links
Running the Router Kit

User-space development, 4x1GE line-rate forwarding

- OSPF
- BGP
- My Protocol
- Routing Table

IPv4 Router

Usage #1

Enhancing Modular Reference Designs

Verilog, System Verilog, VHDL, Bluespec,…

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

Verilog modules interconnected by FIFO interfaces
Creating new systems

- FPGA
- Memory
- PCI

EDA Tools (Xilinx, Mentor, etc.)

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

Verilog, System Verilog, VHDL, Bluespec...

My Design
(1GE MAC is soft/replaceable)

Tools + Reference Designs 1G

Tools:
- Compile designs
- Verify designs
- Interact with hardware

Reference designs:
- Router (HW)
- Switch (HW)
- Network Interface Card (HW)
- Router Kit (SW)
- SCONE (SW)
Contributed Projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Contributor</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFlow switch</td>
<td>Stanford University</td>
</tr>
<tr>
<td>Packet generator</td>
<td>Stanford University</td>
</tr>
<tr>
<td>NetFlow Probe</td>
<td>Brno University</td>
</tr>
<tr>
<td>NetThreads</td>
<td>University of Toronto</td>
</tr>
<tr>
<td>zFilter (Sp)router</td>
<td>Ericsson</td>
</tr>
<tr>
<td>Traffic Monitor</td>
<td>University of Catania</td>
</tr>
<tr>
<td>DFA</td>
<td>UMass Lowell</td>
</tr>
</tbody>
</table>

More projects: [http://netfpga.org/foswiki/NetFPGA/OneGig/ProjectTable](http://netfpga.org/foswiki/NetFPGA/OneGig/ProjectTable)
(pages will be refactored as we integrate NetFPGA10G)

Community

**Wiki**
- Documentation
  - User’s Guide
  - Developer’s Guide
- Encourage users to contribute

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

Over 1,000 users, using 1,900 cards at 150 universities in 32 countries

NetFPGA’s Defining Characteristics

- **Line-Rate**
  - Processes back-to-back packets
    - Without dropping packets
    - At full rate of Gigabit Ethernet Links
  - 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
  - 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 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?**
- 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
Let's have an example, but first....

Network review

Internet Protocol (IP)

Data to be transmitted:

IP packets:

Ethernet Frames:
## Internet Protocol (IP)

### Header Structure

- **Ver**: Version (4 bits)
- **Len**: Header Length (4 bits)
- **T.Service**: Type of Service (8 bits)
- **Total Packet Length**: Total Packet Length (16 bits)
- **Fragment ID**: Fragment ID (16 bits)
- **Flags**: Flags (3 bits)
- **Fragment Offset**: Fragment Offset (13 bits)
- **Protocol**: Protocol (8 bits)
- **TTL**: Time to Live (8 bits)
- **Header Checksum**: Header Checksum (16 bits)
- **Source Address**: Source Address (32 bits)
- **Destination Address**: Destination Address (32 bits)
- **Options (if any)**: Options (variable length)

### Basic Operation of an IP Router

- **Nodes**: A, B, C, D, E, F
- **Routers**: R1, R2, R3, R4, R5
- **Paths**:
  - A -> R1 -> R2 -> R3 -> D
  - B -> R1 -> R2 -> R3 -> D
  - C -> R2 -> R3
  - D -> R2
  - E -> R4
  - F -> R5

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>R3</td>
</tr>
<tr>
<td>E</td>
<td>R3</td>
</tr>
<tr>
<td>F</td>
<td>R5</td>
</tr>
</tbody>
</table>
LONDON – June 15th, 2012

Basic operation of an IP router

Forwarding tables

IP address 32 bits wide \(\rightarrow\) \(\sim\) 4 billion unique address

Naïve approach:
One entry per address

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0.0.0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.0.0.1</td>
<td>2</td>
</tr>
<tr>
<td>(2^{32})</td>
<td>255.255.255.255</td>
<td>12</td>
</tr>
</tbody>
</table>

\(\sim\) 4 billion entries

Improved approach:
Group entries to reduce table size

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0.0.0 – 127.255.255.255</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>128.0.0.1 – 128.255.255.255</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>248.0.0.0 – 255.255.255.255</td>
<td>12</td>
</tr>
</tbody>
</table>
IP addresses as a line

Your computer  
Cambridge  
USA

My computer  
Oxford  
Europe

2^{32} - 1

Entry  | Destination  | Port
--- | --- | ---
1  | Cambridge  | 1
2  | Oxford  | 2
3  | Europe  | 3
4  | USA  | 4
5  | Everywhere (default)  | 5

Longest Prefix Match (LPM)

Entry  | Destination  | Port
--- | --- | ---
1  | Cambridge  | 1
2  | Oxford  | 2
3  | Europe  | 3
4  | USA  | 4
5  | Everywhere (default)  | 5

Matching entries:
- Cambridge  Most specific
- Europe
- Everywhere

To: Cambridge  Data
Longest Prefix Match (LPM)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cambridge</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Oxford</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Europe</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>USA</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Everywhere (default)</td>
<td>5</td>
</tr>
</tbody>
</table>

Matching entries:
- Europe
- Everywhere

To:
- France

Data

Implementing Longest Prefix Match

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cambridge</td>
<td>1</td>
<td>Searching</td>
</tr>
<tr>
<td>2</td>
<td>Oxford</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Europe</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>USA</td>
<td>4</td>
<td>FOUND</td>
</tr>
<tr>
<td>5</td>
<td>Everywhere (default)</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Most specific: Europe

Least specific: Everywhere
## Forwarding tables

32 bits wide → ~4 billion unique addresses

**Naive approach:**
One entry per address

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0.0.0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.0.0.1</td>
<td>2</td>
</tr>
<tr>
<td>2^32</td>
<td>255.255.255</td>
<td>12</td>
</tr>
</tbody>
</table>

~4 billion entries

**Improved approach:**
Group entries to reduce table size

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0.0.0 – 127.255.255</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>128.0.0.0 – 128.255.255</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>248.0.0.0 – 255.255.255</td>
<td>12</td>
</tr>
</tbody>
</table>

## IP addresses as a line

All IP addresses

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stanford</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Berkeley</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>North America</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Asia</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Everywhere (default)</td>
<td>5</td>
</tr>
</tbody>
</table>
### Longest Prefix Match (LPM)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stanford</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Berkeley</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>North America</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Asia</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Everywhere (default)</td>
<td>5</td>
</tr>
</tbody>
</table>

Matching entries:
- Stanford (Most specific)
- North America
- Everywhere

**To:** Stanford

### Longest Prefix Match (LPM)

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stanford</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Berkeley</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>North America</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Asia</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Everywhere (default)</td>
<td>5</td>
</tr>
</tbody>
</table>

Matching entries:
- North America (Most specific)
- Everywhere

**To:** Canada

**Data**
Implementing Longest Prefix Match

<table>
<thead>
<tr>
<th>Entry</th>
<th>Destination</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stanford</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Berkeley</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>North America</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Asia</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Everywhere (default)</td>
<td>5</td>
</tr>
</tbody>
</table>

Most specific → Least specific

Basic components of an IP router

- **Control Plane**
  - Management & CLI
  - Routing Protocols
  - Routing Table

- **Data Plane**
  - Forwarding Table
  - Switching
  - Queuing

- **Software**

- **Hardware**
  - per-packet processing
Example I

Reference Router running on the NetFPGA
Operational IPv4 router

- **Control Plane**
  - SCONE
    - Management & CLI
    - Routing Protocols
    - Routing Table
  - Reference router
    - Forwarding Table
    - Switching
    - Queuing

- **Data Plane**
  - Hardware
  - Software
  - per-packet processing

NetFPGA Lab Setup

- **Hardware**
  - Client
  - Server
  - CPU x2
  - NetFPGA Control SW
  - CAD Tools

- **Software**
  - Dual NIC
  - Internet Router
  - Hardware

- **Network Connections**
  - eth1: Local Client & Server
  - eth2: Server for Neighbor
  - nf2c3: Ring - Left
  - nf2c2: Local Host
  - nf2c1: Neighbor
  - nf2c0: Ring - Right
Server delivers streaming HD video through a chain of NetFPGA Routers.

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.
Toplogy of NetFPGA Routers

Video Server

HD Display

Subnet Configuration

Video Server

Shortest Path

Video Client
Cable Configuration for Demo 1

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

- **Host Ethernet Interfaces**
  - eth1: Local host interface (red)

Review

NetFPGA as IPv4 router:
- Reference hardware + SCONE software
- Routing protocol discovers topology

Demo:
- Ring topology
- Traffic flows over shortest path
- Broken link: automatically route around failure
Working IP Router

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

Streaming Video through the NetFPGA

- **Video server**
  - Source files
    - `/var/www/html/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)
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 …
**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

**Exercise 1**

Explore the Reference Router
LONDON – June 15th, 2012

**Reference Router Pipeline**

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

---

**Exploring the Reference Router**

**Objectives:**
- Run the software
- Explore router architecture

**Execution**
- Run the GUI
- Test connectivity and statistics with pings
- Explore pipeline in the details page
- Explore detailed statistics in the details page
Step 1 - Build the Hardware

Close all windows

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

Run “make clean”

Start synthesis with “make”

Step 2 - Run Reference Router

In a new terminal window

```
  cd ~/NF2/projects/tutorial_router/sw
```

To use the router hardware, type:

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

To stream video, run (in a new terminal)

```
  cd ~/NF2/projects/tutorial_router/sw
  ./mp 192.168.X.Y where X.Y = 25.1 or 19.1 or 7.1
```

Open a browser for the full list of host IP address
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.

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.
Step 6 - Explore Output Queues

Click on the Output Queues module in the Details tab

The page gives configuration details

...and statistics

First Break

(examine the running router, watch *Wallace and Gromit*, or get tea/coffee)
Hardware Overview

NetFPGA-1G
Xilinx Virtex II Pro 50

- 53,000 Logic Cells
- Block RAMs
- Embedded PowerPC

Network and Memory

- Gigabit Ethernet
  - 4 RJ45 Ports
  - Broadcom PHY

- Memories
  - 4.5MB Static RAM
  - 64MB DDR2 Dynamic RAM
Other IO

• **PCI**
  - Memory Mapped Registers
  - DMA Packet Transferring

• **SATA**
  - Board to Board communication

NetFPGA-10G

• A major upgrade
• State-of-the-art technology
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>NetFPGA 1G</th>
<th>NetFPGA 10G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>4 x 1Gbps Ethernet Ports</td>
<td>4 x 10Gbps SFP+</td>
</tr>
<tr>
<td></td>
<td>4.5 MB ZBT SRAM</td>
<td>27 MB QDRII-SRAM</td>
</tr>
<tr>
<td></td>
<td>64 MB DDR2 SDRAM</td>
<td>288 MB RLDRAM-II</td>
</tr>
<tr>
<td>Memories</td>
<td>27 MB QDRII-SRAM</td>
<td>PCI Express x8</td>
</tr>
<tr>
<td></td>
<td>288 MB RLDRAM-II</td>
<td>Virtex 5 TX240T</td>
</tr>
<tr>
<td>I/O Interfaces</td>
<td>PCI</td>
<td>PCI Express x8</td>
</tr>
<tr>
<td>ICs</td>
<td>Virtex II-Pro 50</td>
<td>Virtex 5 TX240T</td>
</tr>
</tbody>
</table>

### 10 Gigabit Ethernet

- **4 SFP+ Cages**
- **AEL2005 PHY**
- **10G Support**
  - Direct Attach Copper
  - 10GBASE-R Optical Fiber
- **1G Support**
  - 1000BASE-T Copper
  - 1000BASE-X Optical Fiber
### Others

- **QDRII-SRAM**
  - 27MB
  - Storing routing tables, counters and statistics
- **RLDRAM-II**
  - 288MB
  - Packet Buffering
- **PCI Express x8**
  - PC Interface
- **Expansion Slot**

### Xilinx Virtex 5 TX240T

- Optimized for ultra high-bandwidth applications
- 48 GTX Transceivers
- 4 hard Tri-mode Ethernet MACs
- 1 hard PCI Express Endpoint
Beyond Hardware

- NetFPGA-10G Board
- Xilinx EDK based IDE
- Reference designs with ARM AXI4
- Software (embedded and PC)
- Public Repository (GitHub)
- Public Wiki (GitHub)

Registration for access is required – but available without limit
https://github.com/NetFPGA/NetFPGA-10G-empty/wiki/Going-Beta

NetFPGA-1G 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-1G Servers

2U Server
(Dell 2950)

NetFPGA inserts in PCI or PCI-X slot

1U Server
(Accent Technology Inc.)

Stanford NetFPGA-1G Cluster

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

<table>
<thead>
<tr>
<th># packets at 10Gb/s</th>
<th>1,000,000</th>
<th>10,000</th>
<th>20</th>
</tr>
</thead>
</table>

\[ 2T \times C \quad (1) \rightarrow \quad \frac{2T \times C}{\sqrt{n}} \quad (2) \rightarrow \quad O(\log W) \]

(1) Assume: Large number of desynchronized flows; 100% utilization
(2) Assume: Large number of desynchronized flows; <100% utilization
Why 2TxC for a single TCP Flow?

Continuous ARQ (TCP) adapting to congestion

Rule for adjusting $W$
- If an ACK is received: $W \leftarrow W + 1/W$
- If a packet is lost: $W \leftarrow W/2$

Only $W$ packets may be outstanding

$W = 1$

util = 0%

time

W
LONDON – June 15th, 2012

Time Evolution of a Single TCP Flow

- Time evolution of a single TCP flow through a router. Buffer is 2T*C

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.
Exercise 2: Enhancing the Reference 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
Reference Router Pipeline

We need to add two modules

1. **Event Capture** to capture output queue events (writes, reads, drops)

2. **Rate Limiter** to create a bottleneck

Enhanced Router Pipeline

We need to add two modules

1. **Event Capture** to capture output queue events (writes, reads, drops)

2. **Rate Limiter** to create a bottleneck
An aside: emacs Tips

We will modify Verilog source code with emacs

- 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

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

Alt-X vhdl-mode<CR>
Type
emacs NF2/projects/tutorial_router/src/user_data_path.v
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).

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).
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

---

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)
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)

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
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)

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

Step 8 - Add Rate Limiter

Scroll down until you reach the next “excluded” block

Uncomment the block containing the rate limiter instantiations.

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

Save (ctrl+x+s)
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)

Lunch is downstairs
Exercise 2

Observing and Controlling the Queue Size
With YOUR enhanced router!

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.
NetFPGA Hardware Set for Exercise #2

Server delivers streaming HD video to adjacent client

Setup for the Exercise 2

Adjacent Web & Video Server

Local Host

NetFPGA

Router

eth1

nf2c2

nf2c1

eth2
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.

Cable Configuration for Exercise 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)
Exercise 2 Configuration

Key:
- Eth1: 192.168.X.1
- Eth2: 192.168.Y.1
- NetFPGA Router 

Stream traffic through your NetFPGA router’s Eth1 interface using your neighbor’s eth2 interface

Exercise 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
Step 1 - Run Your Enhanced Router

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

Type
ADI...
./tut_adv_router_gui.pl -use_bin\..
../../../bitfiles/tutorial_router.bit

A familiar GUI should start

Step 2 - Explore Your Enhanced Router

Click on the Details tab

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

Two modules added

1. **Event Capture** to capture output queue events (writes, reads, drops)

2. **Rate Limiter** to create a bottleneck

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
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

Step 5 - Start Event Capture

Click on the Event Capture module under the Details tab

This should start the configuration page
Step 6 - Configure Event Capture

Please do these in the ORDER below...

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.

Step 7 - Start TCP Transfer

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

Start a new terminal and cd to “NF2/projects/tutorial_router/sw”

Type “./iperf.sh”
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 – observe the BUFFER occupancy

Leave the control windows open
Running the Router Kit

User-space development, 4x1GE line-rate forwarding

CPU  Memory
PCI

OSPF  My Protocol  BGP
Routing Table

IPv4 Router
Fwding Table
Packet Buffer

FPGA  Memory

Usage #1

NetFPGA is a Community
Altera-DE4 NetFPGA Reference Router

UMassAmherst

- Migration of NetFPGA infrastructure to DE4 Stratix IV – 4X logic vs. Virtex 2
- PCI Express Gen2 – 5.0Gbps/lane data
- External DDR2 RAM – 8-Gbyte capacity.
- Status: Functional – basic router performance matches current NetFPGA
- Lots of logic for additional functions
- Russ Tessier (tessier@ecs.umass.edu)

Free repository available from UMass in September 2011

Usage #2

Enhancing Modular Reference Designs

Verilog, System Verilog, VHDL, Bluespec….

Verilog modules interconnected by FIFO interfaces.
Creating new systems

Verilog, System Verilog, VHDL, Bluespec….

EDA Tools (Xilinx, Mentor, etc.)

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

Usage #3

My Design

(1GE MAC is soft/replaceable)

NetThreads, NetThreads-RE, NetTM

Martin Labrecque
Gregory Steffan
ECE Dept.

Geoff Salmon
Monia Ghobadi
Yashar Ganjali
CS Dept.

U. of Toronto
Soft Processors in FPGAs

- Soft processors: processors in the FPGA fabric
- User uploads program to soft processor
- Easier to program software than hardware in the FPGA
- Could be customized at the instruction level

Process packets in software!
Fast enough?

Performance In Packet Processing

- The application defines the requirements

Scientific instruments (< 100 Mbps/link)
Home networking (~100 Mbps/link)
Edge routing (≥ 1 Gbps/link)

Are soft processors fast enough?
Realistic Goals

- 1 gigabit stream
- 2 processors running at 125 MHz
- Cycle budget for back-to-back packets:
  - 152 cycles for minimally-sized 64B packets;
  - 3060 cycles for maximally-sized 1518B packets

Soft processors can perform non-trivial processing at 1gigE!

Latency to answer a ping request:
  - Quad Xeon server → 48.9 us +/- 17.5 us
  - NetThreads in NetFPGA 1G → 5.1 us +/- 0.04 us

NetThread projects provide:

- Efficient multithreaded design
  - Parallel threads deliver performance
- System Features
  - System is easy to program in C
  - Time to results is very short

We hope to see many projects
We also need help:
  - e.g. software that could be ported: operating system, lwIP
NetThread followup Questions?

Ask: Martin Labrecque
martinL@eecg.utoronto.ca

• NetThreads, NetThreads-RE & NetTM available with supporting software at:
  http://www.netfpga.org/foswiki/bin/view/NetFPGA/OneGig/NetThreads
  http://www.netfpga.org/foswiki/bin/view/NetFPGA/OneGig/NetThreadsRE
  http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/NetTM

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" (since '05)
    Quarter-long course targeted at graduates
    http://cs344.stanford.edu/

- Rice University
  - Network Systems Architecture (since '08)

- Cambridge University
  - Build an Internet Router (since '09)
    Quarter-long course targeted at graduates
    http://www.cl.cam.ac.uk/teaching/current/P33/

- University of Wisconsin
  - CS838 "Rethinking the Internet Architecture"
    http://pages.cs.wisc.edu/~akella/CS838/F09/

- University of Bonn
  - "Building a Hardware Router"
    http://bit.ly/Kmo0rA

See: http://netfpga.org/teachers.html

Components of NetFPGA Course

- **Documentation**
  - System Design
  - Implementation Plan

- **Deliverables**
  - Hardware Circuits
  - System Software
  - Milestones

- **Testing**
  - Proof of Correctness
  - Integrated Testing
  - Interoperability

- **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

1. Build basic router
2. Command Line Interface
3. Routing Protocol (PW-OSPF)
4. Integrate with H/W
5. Interoperability
6. Final Project

- Innovate and add!
- Presentations
- Judges

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-port non-learning switch</td>
<td>4-port learning switch</td>
<td>IPv4 router forwarding path</td>
<td>Integrate with S/W</td>
<td>Interoperability</td>
<td>Final Project</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Learning Environment</td>
<td>Modular design</td>
<td>Testing</td>
<td>Interoperability</td>
<td>Final Project</td>
<td>Final Project</td>
</tr>
</tbody>
</table>

13/06/2012
## Typical NetFPGA Course Plan

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

## Presentations

Stanford CS344

[http://cs344.stanford.edu](http://cs344.stanford.edu)

Cambridge P33

[http://www.cl.cam.ac.uk/teaching/0910/P33/](http://www.cl.cam.ac.uk/teaching/0910/P33/)
Exercise 3: Controlled packet-loss

Beyond just observation, using NetFPGA for an experiment

Controlled packet-loss

- Packet networks have loss; evaluating loss we use modeling, simulation, emulation, real-world experiments

- NetFPGA can implement a controlled, packet loss mechanism with none of the disadvantages of emulation...

- Exercise 3: Drop 1 in N Packets....
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.

Our Enhanced Router Pipeline

One module added
1. Drop Nth Packet
to drop every Nth packet from the reference router pipeline
**New Even-More Enhanced Router Pipeline**

**One module added**

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

---

**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 `emacs NF2/projects/tutorial_router/src/drop_nth_packet.v`
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)

Step 3 - Build the Hardware

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

Run “make clean”

Start synthesis with “make”
Third Break

(while hardware compiles)

Step 4 – 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  (Open a new terminal first)
   – replace N with a number (such as 100)

To enable packet dropping, type:  
   regwrite 0x2000700 0x1

To disable packet dropping, type:  
   regwrite 0x2000700 0x0
Step 5a – Measurements (network)

- Explore loss across network
- Ping to neighbour’s server (and other servers)
  - Set a loss of 1 in 100 then, similar to Exercise 1
  - Ping 192.168.x.2 (where x is your immediate neighbour’s server)
    - What is the loss? 1 in 100?
  - Now, ping any addresses 192.168.x.y where x is from 1-20 and y is 1 or 2
  - Can you compute the loss-rate of a neighbour’s router?
  - Apart from ping packets, what other packets might be lost?
    (routing activities, control packets, ARP, ...)

Step 5b – Measurements (transport)

- Determine iperf TCP throughput to neighbour’s server for each of several values of N
  - Similar to Exercise 2, Step 7
  - TCP throughput with:
    - Drop circuit disabled
      - TCP Throughput = ________ Mbps
    - Drop one in N = 10,000 packets
      - TCP Throughput = ________ Mbps
    - Drop one in N = 1,000 packets
      - TCP Throughput = ________ Mbps
    - Drop one in N = 100 packets
      - TCP Throughput = ________ Mbps
  - Explain why TCPs throughput is so low given that only a tiny fraction of packets are lost
Step 5c – Measurements (subjective)

- Consider video throughput to a neighbour’s server for each of several values of N

  To stream video, run (in a new terminal)
  ```
  cd ~/NF2/projects/tutorial_router/sw
  ./mp 192.168.X.Y where X,Y = 25.1 or 19.1 or 7.1
  ```
  *Open firefox for the full list of host IP address*

  - Similar to Exercise 1, Step 2
  - Subjective video quality...
    - Drop circuit disabled
      - Video Quality = Excellent / Good / Fair / Poor / Bad
    - Drop one in N = 10,000 packets
      - Video Quality = Excellent / Good / Fair / Poor / Bad
    - Drop one in N = 1,000 packets
      - Video Quality = Excellent / Good / Fair / Poor / Bad
    - Drop one in N = 100 packets
      - Video Quality = Excellent / Good / Fair / Poor / Bad

Conclusion
Conclusions

• **NetFPGA Provides**
  – Open-source, hardware-accelerated Packet Processing
  – Modular interfaces arranged in reference pipeline
  – Extensible platform for packet processing

• **NetFPGA Reference Code Provides**
  – Large library of core packet processing functions
  – Scripts and GUIs for simulation and system operation
  – Set of Projects for download from repository

• **The NetFPGA Base Code**
  – Well defined functionality defined by regression tests
  – Function of the projects documented in the Wiki Guide

What to do next?
Explore existing projects:

Networked FPGAs in Research

1. OpenFlow
   • 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
6. Precise Time Protocol (PTP)
   • Synchronization among Routers
To get started with your project

Prepare for your project

a) Learn NetFPGA by yourself

b) Encourage others to Complete a hands-on tutorial too

c) Consider attending (hosting) a summer school – doesn’t have to be summer!

Learn by Yourself

Users Guide - for those that have just got their first NetFPGA board

NetFPGA website (www.netfpga.org)
Learn by Yourself

NetFPGA website (www.netfpga.org)

Online tutor – coming soon!

Support for NetFPGA enhancements provided by redgate software
Photos from NetFPGA Tutorials

SIGCOMM - Seattle, Washington, USA

SIGMETRICS - San Diego, California, USA

EuroSys - Glasgow, Scotland, U.K.

Beijing, China

Bangalore, India


NetFPGA 2-Day workshop in Cambridge

Want a tutorial/workshop at your institution? talk to Andrew

- 20 attendees (full house)
- accommodation for non-locals
- 30% commercial attendees
Thoughts for (Prospective) Contributors

• **Build Modular components**
  – Describe shared registers (as per 2.0 release)
  – Consider how modules would be used in larger systems

• **Define functionality clearly**
  – Through regression tests
  – With repeatable results

• **Disseminate projects**
  – Post open-source code
  – Document projects on Web, Wiki, and Blog

• **Expand the community of developers**
  – Answer questions in the Discussion Forum
  – Collaborate with your peers to build new applications

Acknowledgments

*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

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

Andrew Moore, David Miller, Muhammad Shahbaz, Martin Zadnik

For help with todays tutorial

Yury Audzevich, Ed Cree, Andriy Gordiyuchuk, James Snee

*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

Kees Vissers, Michaela Blott, Shep Siegel
Special thanks to our Partners:

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

Other NetFPGA Tutorials Presented At:

See: http://NetFPGA.org/tutorials/

Thanks to our Sponsors:

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

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.
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

Feedback

• We thrive on feedback – please fill in the survey now.....


Thanks!

NetFPGA website (www.netfpga.org)