NetFPGA Tutorial

Junho Suh
(jhsuh@mmlab.snu.ac.kr)
Content

- NetFPGA
- Basic IP Router
- A Basic IP Router on NetFPGA
- Exercises
Prerequisites

• Logic Design (4190.201)
• Computer Network (4190.411)
• Computer language
  • C, C++
• Hardware description language
  • Verilog, VDHL
What is NetFPGA?

- A **line-rate**, flexible, open networking platform for teaching and research
- **Per-packet processing**
  - 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
NetFPGA Elements

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

Networking Software running on a standard PC

A hardware accelerator built with Field Programmable Gate Array driving Gigabit network links
Tools + Reference Designs

- Tools
  - Compile designs
  - Verify designs
  - Interact with hardware
- Reference designs
  - Router (HW)
  - Switch (HW)
  - Network Interface Card (HW)
  - SCON (SW)
# Contributed Designs

<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/bin/view/NetFPGA/OneGig/ProjectTable](http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ProjectTable)
Community

- Wiki
  - Documentation
  - Encourage users to contribute
- Forums
  - Support by users for users
  - Active community – 10s ~ 100s of posts per week
Basic IP Router

- Basic Operation of an IP Router
Basic IP Router

- Basic Operation of an IP Router
Basic IP Router

- Basic Operation of an IP Router
Basic IP Router

• Basic Operation of an IP Router

<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>
Basic IP Router

- Basic Operation of an IP Router

<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>
Basic IP Router

- Basic Operation of an IP Router

<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>
Basic IP Router

- Basic Operation of an IP Router
Basic IP Router

• Basic Operation of an IP Router
Basic IP Router Components

Control Plane

Datapath
per-packet processing

Software

Management & CLI
Routing Protocols
Routing Table

Hardware

Forwarding Table
Switching
Basic IP Router Operations

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

4. Buffer packet in the output queue

5. Transmit packet onto outgoing link
Generic Datapath Architecture

Header Processing

- Lookup IP Address
- Update Header
- Queue Packet

Forwarding Table

Buffer Memory
CIDR and Longest Prefix Matches (LPM)

- 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
A Basic IP Router on NetFPGA

- Management & CLI
- Exception Processing
- Routing Protocols
- Routing Table
- Forwarding Table
- Switching

Software
- Linux user-level processes

Hardware
- Verilog on NetFPGA PCI board
Router Stages
Inter-Module Communication

• Using “Module Headers”:

<table>
<thead>
<tr>
<th>Ctrl Word (8 bits)</th>
<th>Data Word (64 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Module Hdr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>y</td>
<td>Last Module Hdr</td>
</tr>
<tr>
<td>0</td>
<td>Eth Hdr</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr</td>
</tr>
<tr>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>0x10</td>
<td>Last word of packet</td>
</tr>
</tbody>
</table>

Contain information such as packet length, input port, output port, ...
Inter-Module Communication
MAC Rx Queue

Eth Hdr:
Dst MAC = port 0,
Ethertype = IP

IP Hdr:
IP Dst: 192.168.2.3,
TTL: 64, Csum:0x3ab4

Data
MAC Rx Queue

- EthHdr:
  - Dst MAC = port 0,
  - Ethertype = IP

- IPHdr:
  - IP Dst: 192.168.2.3,
  - TTL: 64, Csum:0x3ab4

- Data
### MAC Rx Queue

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eth Hdr:</td>
</tr>
<tr>
<td>0</td>
<td>Dst MAC = port 0, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td>0</td>
<td>IP Dst: 192.168.2.3, TTL: 64, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Input Arbiter

packet scheduling algorithms
Input Arbiter

packet scheduling algorithms
Input Arbiter

packet scheduling algorithms
Input Arbiter

packet scheduling algorithms
Input Arbiter

packet scheduling algorithms
Input Arbiter

packet scheduling algorithms
Input Arbiter

packet scheduling algorithms
Output Port Lookup

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0</td>
</tr>
<tr>
<td></td>
<td>Src MAC = x,</td>
</tr>
<tr>
<td></td>
<td>Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3,</td>
</tr>
<tr>
<td></td>
<td>TTL: 64, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
# Output Port Lookup

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0</td>
</tr>
<tr>
<td></td>
<td>Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3, TTL: 64, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
# Output Port Lookup

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop, Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

Monday, May 13, 13
# Output Port Lookup

1. Check input port matches Dst MAC

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop</td>
</tr>
<tr>
<td></td>
<td>Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1- Check input port matches Dst MAC

2- Check TTL, checksum

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop</td>
</tr>
<tr>
<td></td>
<td>Src MAC = port 4,</td>
</tr>
<tr>
<td></td>
<td>Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3,</td>
</tr>
<tr>
<td></td>
<td>TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
# Output Port Lookup

1. Check input port matches Dst MAC
2. Check TTL, checksum
3. Lookup next hop IP & output port (LPM)

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop</td>
</tr>
<tr>
<td></td>
<td>Src MAC = port 4,</td>
</tr>
<tr>
<td></td>
<td>Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3,</td>
</tr>
<tr>
<td></td>
<td>TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1. Check input port matches Dst MAC
2. Check TTL, checksum
3. Lookup next hop IP & output port (LPM)
4. Lookup next hop MAC address (ARP)

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop</td>
</tr>
<tr>
<td></td>
<td>Src MAC = port 4,</td>
</tr>
<tr>
<td></td>
<td>Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3,</td>
</tr>
<tr>
<td></td>
<td>TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1- Check input port matches Dst MAC
2- Check TTL, checksum
3- Lookup next hop IP & output port (LPM)
4- Lookup next hop MAC address (ARP)
5- Add output port header

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1. Check input port matches Dst MAC
2. Check TTL, checksum
3. Lookup next hop IP & output port (LPM)
4. Lookup next hop MAC address (ARP)

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop, Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

5. Add output port header
Output Port Lookup

1. Check input port matches Dst MAC
2. Check TTL, checksum
3. Lookup next hop IP & output port (LPM)
4. Lookup next hop MAC address (ARP)

5. Add output port header
6. Modify MAC Dst and Src addresses

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop, Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1- Check input port matches Dst MAC
2- Check TTL, checksum
3- Lookup next hop IP & output port (LPM)
4- Lookup next hop MAC address (ARP)

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0 output port = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop, Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

5- Add output port header
6- Modify MAC Dst and Src addresses
7- Decrement TTL and update checksum
Output Queues

[Diagram showing output queues OQ0, OQ4, and OQ7]
Output Queues
### MAC Tx Queue

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop, Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 63, Csum:0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

Monday, May 13, 13
## MAC Tx Queue

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = nextHop, Src MAC = port 4, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IPHdr: IP Dst: 192.168.2.3, TTL: 63, Csum: 0x3ac2</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
MAC Tx Queue
MAC Tx Queue

| 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  

Monday, May 13, 13
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

Software

PCI Bus

NetFPGA

Ethernet
Exception Packet

Software

PCI Bus

NetFPGA

Ethernet
Exception Packet

Software

PCI Bus

NetFPGA

DMA

Registers
## Output Port Lookup

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
## Output Port Lookup

1. Check input port matches Dst MAC

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1- Check input port matches Dst MAC

2- Check TTL, checksum – EXCEPTION!

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IPHdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Port Lookup

1- Check input port matches Dst MAC

2- Check TTL, checksum – EXCEPTION!

3- Add output port module

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

Monday, May 13, 13
Output Port Lookup

1- Check input port matches Dst MAC

2- Check TTL, checksum – EXCEPTION!

3- Add output port module

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>
Output Queues
Output Queues

![Diagram of Output Queues]

- OQ0
- OQ1
- Pkt
- OQ7
## CPU Tx Queue

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

Monday, May 13, 13
## CPU Tx Queue

<table>
<thead>
<tr>
<th>0xff</th>
<th>Pkt length, input port = 0, output port = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</td>
</tr>
<tr>
<td>0</td>
<td>IP Hdr: IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td>0</td>
<td>Data</td>
</tr>
</tbody>
</table>

Monday, May 13, 13
CPU Tx Queue
CPU Tx Queue

<table>
<thead>
<tr>
<th></th>
<th>EthHdr: Dst MAC = 0, Src MAC = x, Ethertype = IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IPHdr:</td>
</tr>
<tr>
<td></td>
<td>IP Dst: 192.168.2.3, TTL: 1, Csum:0x3ab4</td>
</tr>
<tr>
<td></td>
<td>Data</td>
</tr>
</tbody>
</table>
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
Exception Packet

Software

PCI Bus

NetFPGA

Ethernet
Exception Packet

Software

PCI Bus

NetFPGA

Ethernet
Exception Packet

Software

PCI Bus

NetFPGA

Ethernet
Exception Packet

Software

PCI Bus

NetFPGA

Ethernet

SCONE

PW-OSPF

Java GUI

Driver

nf2c0

nf2c1

nf2c2

nf2c3

ioctl

DMA

Registers

nf2_reg_grp

CPU RxQ

CPU TxQ

MAC RxQ

MAC TxQ

user data path
NetFPGA-Host Interaction

- Linux driver interfaces with hardware
- Packet interface via standard Linux network stack
- (Alternative) Register reads/writes via ioctl system call with wrapper functions:
  - slower but eliminates the need to deal with network sockets
    - readReg(nf2device *dev, int address, unsigned *rd_data);
    - writeReg(nf2device *dev, int address, unsigned *wr_data);
    - ex) readReg(&nf2, OQ_NUM_PKTS_STORED_0, &val);
NetFPGA-Host Interaction

NetFPGA to host packet transfer

1. Packet arrives – forwarding table sends to CPU queue

2. Interrupt notifies driver of packet arrival

3. Driver sets up and initiates DMA transfer
NetFPGA-Host Interaction

NetFPGA to host packet transfer (cont.)

4. NetFPGA transfers packet via DMA

5. Interrupt signals completion of DMA

6. Driver passes packet to network stack
NetFPGA-Host Interaction
Host to NetFPGA packet transfers

1. Software sends packet via network sockets
Packet delivered to driver

2. Driver sets up and initiates DMA transfer

3. Interrupt signals completion of DMA
NetFPGA-Host Interaction

Register access

1. Software makes ioctl call on network socket
   ioctl passed to driver

2. Driver performs PCI memory read/write
More Details...

• For NetFPGA

• Visit http://netfpga.org
More Details...

- For Verilog

Exercise 2
Enhancing the Reference Router
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
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 netfpga/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
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 “netfpga/projects/tutorial_router/synth”

Run “make clean”

Start synthesis with “make”
Go back to “Demo 2: Step 1” after synthesis completes and redo the steps with your own router.

To run your router:
1. cd netfpga/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.
Excercise 3
Drop 1 in N Packets
Objective

- 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.
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 “xemacs netfpga/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 “netfpga/projects/tutorial_router/synth”

Run “make clean”

Start synthesis with “make”
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 netfpga/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`
Step 5 – Measurements

- Determine iperf TCP throughput to neighbor’s server for each of several values of N
  - Similar to Demo 2, Step 8
    - cd netfpga/projects/tutorial_router/sw
    - ./iperf.sh
    - 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
Excercise 4
Cryptography
Goal: Implement a NIC that encrypts upon transmission and decrypts upon reception.
Outline

• Tree Structure

• Develop a cryptography module
  – Quick overview of XOR “cryptography”
  – Implement crypto module
  – Write software simulations
  – Synthesize
  – Write hardware tests
Tree Structure

netfpga

- **bin** (scripts for running simulations and setting up the environment)
- **bitfiles** (contains the bitfiles for all projects that have been synthesized)
- **lib** (shared Verilog modules, libraries needed for simulation/synthesis/design)
- **projects** (user projects, including reference designs)
lib

C  (common software and code for reference designs)

Java  (contains software for the graphical user interface)

Makefiles  (makefiles for simulation and synthesis)

Perl5  (libraries to interact with reference designs, create test data, and manage simulations/regression tests)

Python  (common libraries to aid in regression tests)

Scripts  (utility scripts – less commonly used than those in the bin directory)

Verilog  (modules that can be reused in designs)
projects/crypto_nic

- **doc** (project specific documentation)
- **include** (XML files defining project and any local modules, auto-generated Verilog register defines)
- **lib** (C/Perl defines for registers)
- **regress** (regression tests to test generated bitfiles)
- **src** (non-library Verilog code used for synthesis and simulation)
- **sw** (software elements of the project)
- **synth** (project-specific .xco files to generate cores, Makefile to implement the design)
- **verif** (simulation tests)
Cryptography

• Simple cryptography: XOR function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ^ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• XOR written as: ^ ⊻ ⊕
• XOR is commutative
Cryptography

- Simple cryptography: XOR function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ^ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- XOR written as: ^ ⊻
- XOR is commutative

XORing a value with itself always yields 0
• Example:

Message: 00111011
Key: 10110001

Message ^ Key: 10001010

Message ^ Key ^ Key: 00111011

• Explanation:
  – A ^ A = 0
  – So, M ^ K ^ K = M ^ 0 = M
Idea:
Implement simple cryptography using XOR
Implementing a Crypto Module (1)

• What do we want to encrypt?
  – IP payload only
    • Plaintext IP header allows routing
    • Content is hidden
  – Encrypt bytes 35 onward
    • Bytes 1-14 – Ethernet header
    • Bytes 15-34 – IPv4 header (assume no options)
  – Assume all packets are IPv4 for simplicity
Implementing a Crypto Module (2)

- State machine (draw on next page):
  - Module headers on each packet
  - Datapath 64-bits wide
    - \( 34 / 8 \) is not an integer!

- Inside the crypto module
Crypto Module State Diagram

Hint: We suggest 4 states (or 3 if you’re feeling adventurous)
State Diagram to Verilog (1)

Module location

1. **Crypto module** to encrypt and decrypt packets
Inter-module Communication

Module i

Module i+1

data

ctrl

wr

rdy

CLK

RDY

WR

DATA

CTRL
State Diagram to Verilog (2)

• Projects:
  – Each design represented by a project
    Format: netfpga/projects/<proj_name>
    • netfpga/projects/crypto_nic
  – Consists of:
    • Verilog source
    • Simulation tests
    • Hardware tests
  – Missing:
    • State diagram implementation
    • Simulation tests
    • Regression tests
• Projects (cont):
  – Shared modules included from netfpga/lib/verilog
    • Generic modules that are re-used in multiple projects
    • Specify shared modules in project’s include/project.xml
  – Local src modules override shared modules

  – crypto_nic:
    • Local: user_data_path.v, crypto.v
    • Everything else: shared modules
Exploring project.xml (1)

• Location: project/<proj_name>/include

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nf:project ...>
    <nf:name>Crypto NIC</nf:name>
    <nf:description>NIC with basic crypto support</nf:description>
    <nf:version_major>0</nf:version_major>
    <nf:version_minor>1</nf:version_minor>
    <nf:version_revision>0</nf:version_revision>
    <nf:dev_id>0</nf:dev_id>
</nf:project>
```

Unique ID to identify project
See: http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/DeviceIDList
<nf:use_modules>
    core/io_queues/cpu_dma_queue
    core/io_queues/ethernet_mac
    core/input_arbiter/rr_input_arbiter
    core/nf2/generic_top
    core/nf2/reference_core
    core/output_port_lookup/nic
    core/output_queues/sram_rr_output_queues
    core/sram_arbiter/sram_weighted_rr
    core/user_data_path/reference_user_data_path
    core/io/mdio
    core/cpci_bus
    core/dma
    core/user_data_path/udp_reg_master
    core/io_queues/add_rm_hdr
    core/stripe_headers/keep_length
    core/utils/generic_regs
    core/utils
</nf:use_modules>
<nf:memalloc layout="reference">
  <nf:group name="core1">
    <nf:instance name="device_id" />
    <nf:instance name="dma" base="0x0500000" />
    <nf:instance name="mdio" />
    <nf:instance name="nf2_mac_grp" count="4" />
    <nf:instance name="cpu_dma_queue" count="4" />
  </nf:group>
  <nf:group name="udp">
    <nf:instance name="in_arb" />
    <nf:instance name="crypto" />
    <nf:instance name="strip_headers" />
    <nf:instance name="output_queues" />
  </nf:group>
</nf:memalloc>
</nf:project>

 Specify where to instantiate modules, the number of instances, and the memory addresses to use
State Diagram to Verilog (4)

Your task:

1. Copy netfpga/lib/verilog/core/module_template/src/module_template.v to netfpga/projects/crypto_nic/src/crypto.v

2. Implement your state diagram in src/crypto.v
   – Initially use a static 32-bit key
module module_template

#(
    parameter DATA_WIDTH = 64,
    parameter CTRL_WIDTH = DATA_WIDTH/8,
    parameter UDP_REG_SRC_WIDTH = 2
)

(...
...

//------------------------ Signals------------------------
...

//---------------------- Local assignments ------------------
...

Module port declaration
Packet data dumped in a FIFO. Allows some “decoupling” between input and output.
generic_regs
  #(
    .UDP_REG_SRC_WIDTH (UDP_REG_SRC_WIDTH),
    .TAG (0),
    .REG_ADDR_WIDTH (1),
    .NUM_COUNTERS (0),
    .NUM_SOFTWARE_REGS (0),
    .NUM_HARDWARE_REGS (0)
  )
module_regs ( }
  ...
);
always @(*) begin
    // Default values
    out_wr_int = 0;
    in_fifo_rd_en = 0;

    if (!in_fifo_empty && out_rdy) begin
        out_wr_int = 1;
        in_fifo_rd_en = 1;
    end
end
Suggest sequence of steps:

1. Rename module at top of file
2. Create a static key value
   - Constants can be declared in the module with localparam:
     
     \[
     \text{localparam MY EXAMPLE = 32'h01234567;} \\
     \]
3. Implement your state machine without modifying the packet
4. Update your state machine to modify the packet by XORing the key and the payload
   - Use two copies of the key to create a 64-bit value to XOR with data words
Testing: Simulation (1)

• Simulation allows testing without requiring lengthy synthesis process

• NetFPGA provides Perl simulation infrastructure to:
  – Send/receive packets
    • Physical ports and CPU
  – Read/write registers
  – Verify results

• Simulations run in ModelSim/VCS/ISim
Testing: Simulation (2)

- Simulations located in project/verif
- Multiple simulations per project
  - Test different features
- Example:
  - crypto_nic/verif/test_nic_short
    - Send one packet from CPU, expect packet out physical port
    - Send one packet in physical port, expect packet to CPU
Testing: Simulation (3)

• Useful functions:
  – nf_PCI_read32(delay, batch, addr, expect)
  – nf_PCI_write32(delay, batch, addr, value)
  – nf_packet_in(port, length, delay, batch, pkt)
  – nf_expected_packet(port, length, pkt)
  – nf_dma_data_in(length, delay, port, pkt)
  – nf_expected_dma_data(port, length, pkt)
  – make_IP_pkt(length, da, sa, ttl, dst_ip, src_ip)
  – encrypt_pkt(key, pkt)
  – decrypt_pkt(key, pkt)
• Your task:

1. Template files:
   - netfpga/projects/crypto_nic/verif/test_crypto_encrypt/make_pkts.pl
   - netfpga/projects/crypto_nic/verif/test_crypto_decrypt/make_pkts.pl

2. Implement your Perl verif tests
   - Use the example verif test (test_nic_short)
Running Simulations

• Use command `nf_run_test.pl`
  – Optional parameters
    • `--major <major_name>`
    • `--minor <minor_name>`
    • `--gui` (starts the default viewing environment)

```plaintext
test_crypto_encrypt
```

• Set env. variables to reference your project

```plaintext
export NF_DESIGN_DIR=/root/netfpga/projects/<project>
export PERL5LIB=/root/netfpga/projects/<project>/lib/Perl5:/root/netfpga/lib/Perl5:
```
Running Simulations

- When running modelsim interactively:
  - Click "no" when simulator prompts to finish
  - Changes to code can be recompiled without quitting

- ModelSim:
  - bash# cd /tmp/$(whoami)/verif/<projname>;
    make model_sim
  - VSIM 5> restart -f; run -a

- Do ensure $NF_DESIGN_DIR is correct
Replacing static key with register

- Can set the key via a register instead

- Need to understand the register system 😊

- Register system:
  - Specify registers provided by module in the module XML file
  - Implement registers in module
    - Can usually use generic_regs
Register bus

reg_req_in -> reg_req_out
reg_ack_in -> reg_ack_out
reg_rd_wr_L_in -> reg_rd_wr_L_out
reg_addr_in -> reg_addr_out
reg_data_in -> reg_data_out
reg_src_in -> reg_src_out
Module XML file (1)

- Each module (with registers) has an XML file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nf:module ...>
    <nf:name>crypto</nf:name>
    <nf:description>Registers for Crypto Module</nf:description>
    <nf:prefix>crypto</nf:prefix>
    <nf:location>udp</nf:location>
    <nf:blocksize>64</nf:blocksize>
</nf:module>
```

- Name/description
- Prefix appears before register names in source code
- Location: where in the design should this module be instantiated?
- Amount of memory to allocate to the block
<nf:registers>
  <nf:register>
    <nf:name>key</nf:name>
    <nf:description>The Key value used by the Crypto Module</nf:description>
    <nf:type>generic_software32</nf:type>
  </nf:register>
</nf:registers>

Can also declare constants and data types

Register declaration: need name, description, and width or type
Generic Registers Module

generic_regs # (
    .UDP_REG_SRC_WIDTH  (UDP_REG_SRC_WIDTH),
    .TAG                (`CRYPTO_BLOCK_TAG),
    .REG_ADDR_WIDTH     (`CRYPTO_REG_ADDR_WIDTH),
    .NUM_COUNTERS       (0),
    .NUMSOFTWARE_REGS   (1),
    .NUM_HARDWARE_REGS  (0))

crypto_regs (
    .reg_req_in       (reg_req_in),
    ...
    .reg_src_out      (reg_src_out),
    ...
    .software_regs    (key),
    .hardware_regs    (),
    ...

Make sure you declare key as a 32-bit wire
Replacing static key with register

• Replace the static key with the key from the registers

• Update your simulations to set the key
To synthesize your project

- Run make in the synth directory (netfpga/projects/crypto_nic/synth)
Regression Tests

- Test hardware module

- Perl Infrastructure provided to
  - Read/Write registers
  - Read/Write tables
  - Send Packets
  - Check Counters
Example Regression Tests

• Reference Router
  – Send Packets from CPU
  – Longest Prefix Matching
  – Longest Prefix Matching Misses
  – Packets dropped when queues overflow
  – Receiving Packets with IP TTL <= 1
  – Receiving Packets with IP options or non IPv4
  – Packet Forwarding
  – Dropping packets with bad IP Checksum
Perl Libraries

- Specify the Interfaces
  - eth1, eth2, nf2c0 … nf2c3

- Start packet capture on Interfaces

- Create Packets
  - MAC header
  - IP header
  - PDU

- Read/Write Registers

- Read/Write Reference Router tables
  - Longest Prefix Match
  - ARP
  - Destination IP Filter
Regression Test Examples

• Reference Router
  – Packet Forwarding
    • regress/test_packet_forwarding
  – Longest Prefix Match
    • regress/test_lpm
  – Send and Receive
    • regress/test_send_rec
Creating a Regression Test

• Useful functions:
  – `nftest_regwrite(interface, addr, value)`
  – `nftest_regread(interface, addr)`
  – `nftest_send(interface, frame)`
  – `nftest_expect(interface, frame)`
  – `encrypt_pkt(key, pkt)`
  – `decrypt_pkt(key, pkt)`

  – `$pkt = NF::IP_pkt->new(len => $length,
                     DA => $DA, SA => $SA,
                     ttl => $TTL, dst_ip => $dst_ip,
                     src_ip => $src_ip);`
Creating a Regression Test (2)

- Your task:

  1. Template files netfpga/projects/crypto_nic/regress/
     test_crypto_encrypt/run

  2. Implement your Perl verif tests
Running Regression Test

- Run the command
  
nf_regress_test.pl --project crypto_nic