Building an Internet Router (P33)

Handout 1: What's a router?
Class project and logistics

Dr Andrew W. Moore
andrew.moore@cl.cam.ac.uk

Thank you supporters: redgate software, CISCO, Xilinx University Program, Ensoft, zeus

Some logistics

Web page: http://www.cl.cam.ac.uk/teaching/0910/P33/

TAs:
David Miller (Hardware) david.miller@cl.cam.ac.uk
Stephen Kell (Software) stephen.kell@cl.cam.ac.uk

Hot-spare TAs:
Phil Watts (Hardware) philip.watts @cl.cam.ac.uk
Chris Smowton (Software) chris.smowton@cl.cam.ac.uk

Grades:
Mphil (ACS) – Pass / Fail - based on a mark out of 100
All others (DTC) – Mark out of 100

Some more logistics

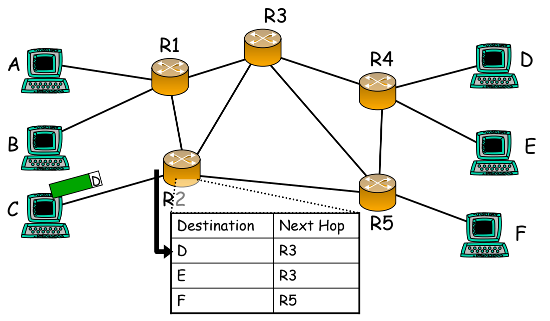
Class Mailing list: *bir-list@cl.cam.ac.uk*
think of this as a self-help mailing list for all of you

Staff Mailing list: *bir-staff@cl.cam.ac.uk*

Submission mail: *bir-tick@cl.cam.ac.uk*

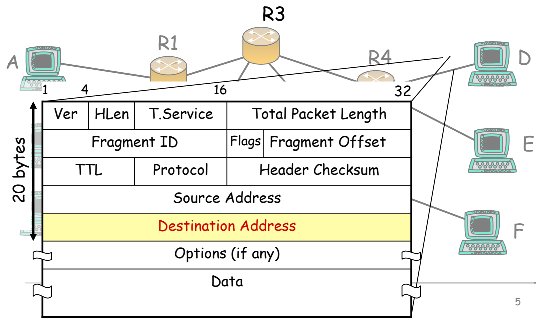
Group aliases can appear if you require them.

What is Routing?



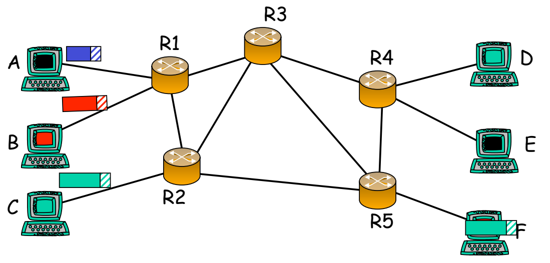
Destination	Next Hop
D	R3
E	R3
F	R5

What is Routing?

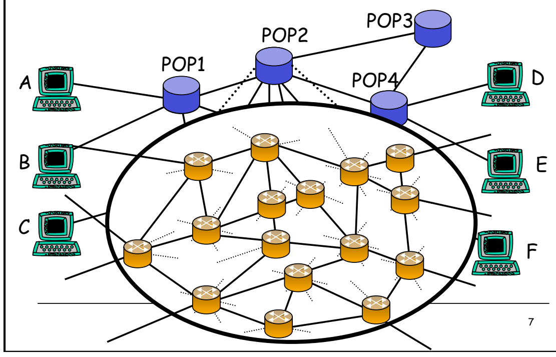


Ver	HLen	T.Service	Total Packet Length
Fragment ID		Flags	Fragment Offset
TTL	Protocol	Header Checksum	
Source Address			
Destination Address			
Options (if any)			
Data			

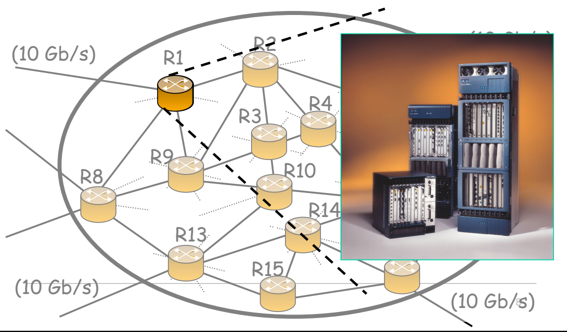
What is Routing?



Points of Presence (POPs)

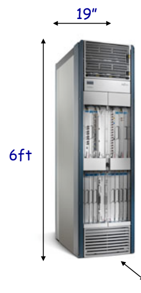


Where High Performance Routers are Used



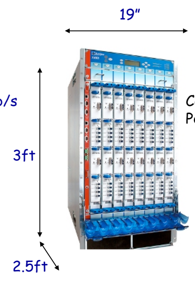
What a Router Looks Like

Cisco CSR-1



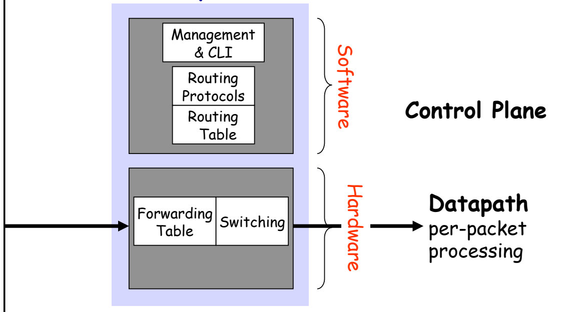
Capacity: 92Tb/s
Power: 4.7kW

Juniper T320



Capacity: 320Gb/s
Power: 2.9kW

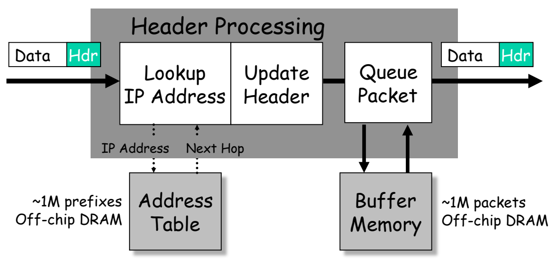
Basic Architectural Components of an IP Router



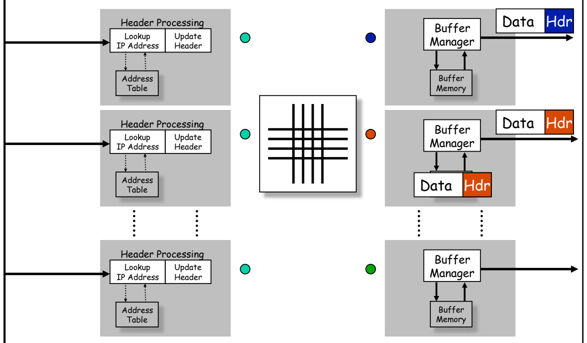
Per-packet processing in an IP Router

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

Generic Router Architecture

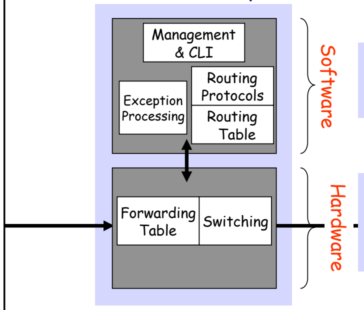


Generic Router Architecture



The P33 Project:

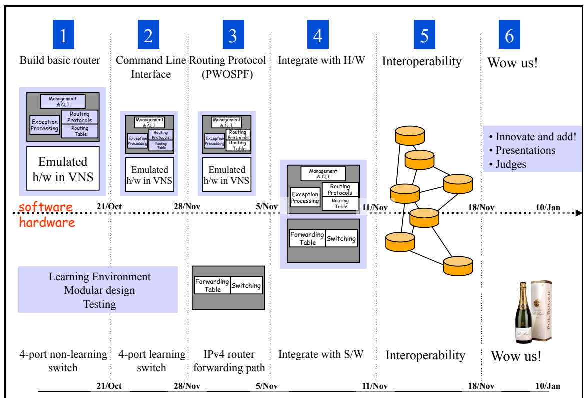
Build a 4-port x 1GE Router



Software Team
Develop s/w in Linux on PC in lab

Hardware Team
Develop h/w in Verilog on NetFPGA board in lab

14

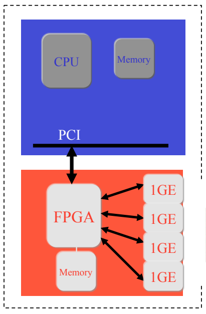


15

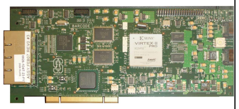
What is the NetFPGA?

Networking Software running on a standard PC

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

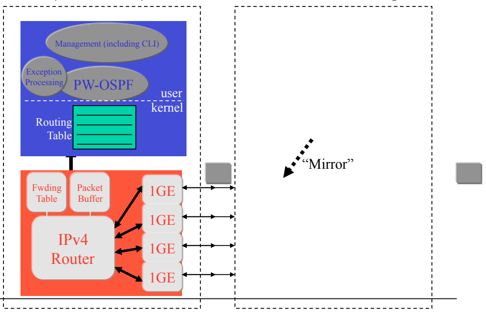


PC with NetFPGA


NetFPGA Board

Running the Router Kit

User-space development, 4x1GE line-rate forwarding



Next steps

- Explore the web page (http://www.cl.cam.ac.uk/teaching/0910/P33/)
- Decide if you still want to take the class
- Build a team of 3 people: 2 s/w and 1 h/w
- Come to SW02 on Tuesday 10am…

<u>We will meet with you every week during class time.</u>

18

Using the nf-test machines

- Each group is allocated an nf-test machine
  `{nf-test1,nf-test2,nf-test3,...}.nf.cl.cam.ac.uk`
- These machines are headless (no KVM) and located in SW02 (alcove)
- Login via ssh; set your own password and go
  We suggest running "`vncserver`" and then using "`vncviewer`"; as a useful way to leave jobs/desktop running and reconnect if you need to move location.
- SW02 is a busy classroom – please respect others
  If you need to gain physical access to an nf-test machine
  Do it quietly! You maybe refused entry in the afternoons – don't take it personally.

nf-test machine *rules*

- Abuse the machines – that's the end of this module for you – zero tolerance policy.
- only connect eth0 to the cisco (access) switch
- these machines run a firewall for a reason
  - if it doesn't work for you, lets fix the rules
- Play nicely with each other; your interoperability marks depend on it.

nf-test machine *information*

- **Warning:** Files stored on nf-test machines are *not* backed up - and may be lost at any time.
- You must make sure that you regularly copy your files into your REAL (computer laboratory) home directory as a back-up
- **Machines fail** – they usually fail 24 hours **before** the big deadline!

Tree Structure

NF2
- **bin** (scripts for running simulations and setting up the environment)
- **bitfiles** (contains the bitfiles for all projects that have been synthesized)
- **lib** (stable common modules and common parts needed for simulation/synthesis/design)
- **projects** (user projects, including reference designs)

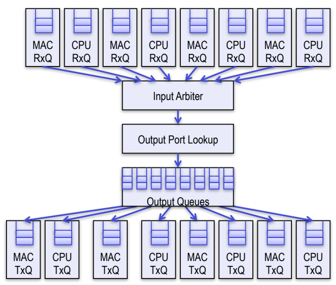
Tree Structure (2)

lib
- **C** (common software and code for reference designs)
- **java** (contains software for the graphical user interface)
- **Makefiles** (makefiles for simulation and synthesis)
- **Perl5** (common libraries to interact with reference designs and aid in simulation)
- **python** (common libraries to aid in regression tests)
- **scripts** (scripts for common functions)
- **verilog** (modules and files that can be reused for design)

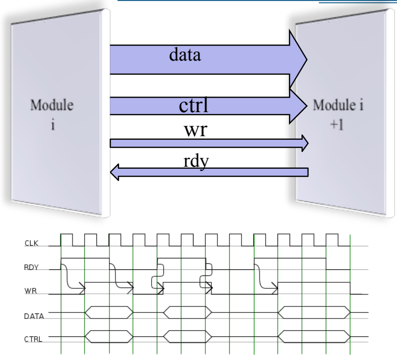
Tree Structure (3)

projects
- **doc** (project specific documentation)
- **include** (contains file to include verilog modules from lib, and creates project specific register defines files)
- **regress** (regression tests used to test generated bitfiles)
- **src** (contains non-library verilog code used for synthesis and simulation)
- **sw** (all software parts of the project)
- **synth** (contains user .xco files to generate cores and Makefile to implement the design)
- **verif** (simulation tests)

State Diagram to Verilog (1)



Inter-module Communication



State Diagram to Verilog (2)

- **Projects:**
  - Each design represented by a project
    Format: NF2/projects/<proj_name>

    e.g., NF2/projects/bir_starter
  - Consists of:
    - •Verilog source
    - •Simulation tests
    - •Hardware tests
    - •Libraries
    - •Optional software
  - What is Missing?
    - refer to the web page
      "Implement four port hard-wired switch" is the start.

State Diagram to Verilog (3)

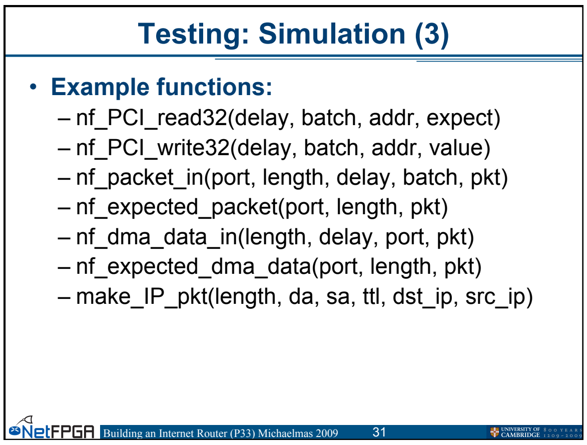
- **Projects (cont):**
  - Pull in modules from NF2/lib/verilog
    - Generic modules that are re-used in multiple projects
    - Specify shared modules in project's include/ lib_modules.txt
  - Local src modules override shared modules
  - bir_starter:
    - Local in_arb_regs.v input_arbiter.v output_port_lookup.v
    - Everything else: shared modules

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**

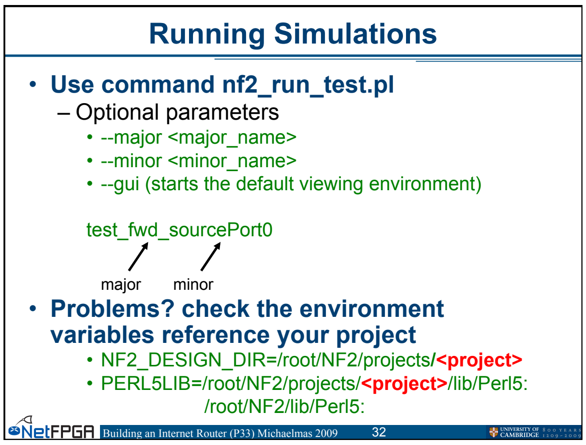
Testing: Simulation (2)

- **Simulations located in project/verif**
- **Multiple simulations per project**
  - Test different features
- **Example:**
  - bir_starter/verif/test_fwd_sourcePort0
    - Test forwarding of packets from port 0 to port 1



Testing: Simulation (3)

- **Example 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)



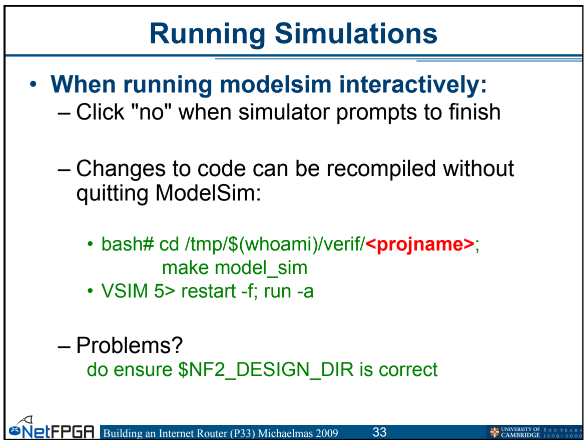
Running Simulations

- **Use command nf2_run_test.pl**
  - Optional parameters
    - --major <major_name>
    - --minor <minor_name>
    - --gui (starts the default viewing environment)

test_fwd_sourcePort0

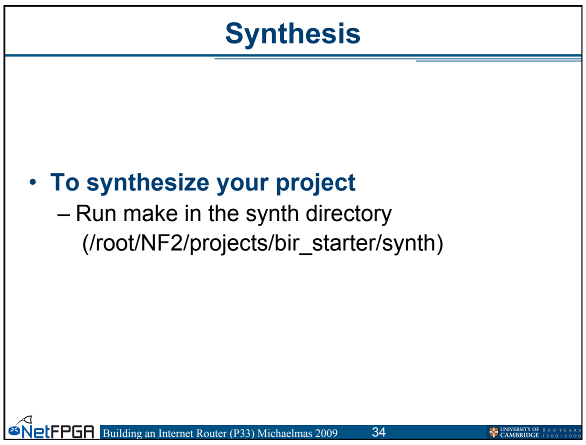
major minor

- **Problems? check the environment variables reference your project**
  - NF2_DESIGN_DIR=/root/NF2/projects/**<project>**
  - PERL5LIB=/root/NF2/projects/**<project>**/lib/Perl5:
    /root/NF2/lib/Perl5:



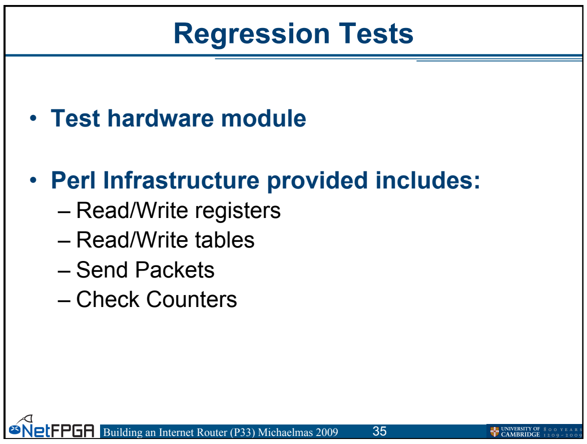
Running Simulations

- **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
  - Problems?
    do ensure $NF2_DESIGN_DIR is correct



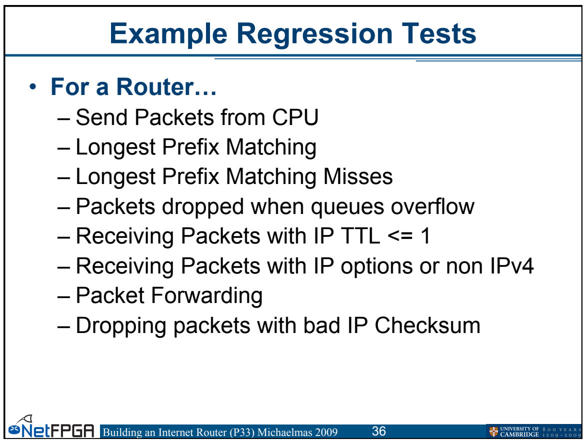
Synthesis

- **To synthesize your project**
  - Run make in the synth directory
    (/root/NF2/projects/bir_starter/synth)



Regression Tests

- **Test hardware module**
- **Perl Infrastructure provided includes:**
  - Read/Write registers
  - Read/Write tables
  - Send Packets
  - Check Counters



Example Regression Tests

- **For a 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)
  -
```
$pkt = NF2::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
   /root/NF2/projects/bir_starter/regress/test_fwd_sourcePort0/run.pl
2. Implement your Perl verif tests

Running Regression Test

- **Run the command**

```
nf2_regress_test.pl --project bir_starter
```