P51: High Performance Networking

Introduction to NetFPGA Infrastructure

Noa Zilberman
noa.zilberman@cl.cam.ac.uk

Lent 2019/20
Infrastructure
Infrastructure

• Tree structure

• NetFPGA package contents
  • Reusable Verilog modules
  • Verification infrastructure
  • Build infrastructure
  • Utilities
  • Software libraries
Tree Structure (1)

NetFPGA-SUME-live

- projects (including reference designs)
- contrib-projects (contributed user projects)
- lib (custom and reference IP Cores and software libraries)
- tools (scripts for running simulations etc.)
- docs (design documentations and user-guides)
Tree Structure (2)

lib

  hw (hardware logic as IP cores)
    std (reference cores)
    contrib (contributed cores)
    xilinx (Xilinx based cores)

  sw (core specific software drivers/libraries)
    std (reference libraries)
    contrib (contributed libraries)
projects/reference_switch

- bitfiles (FPGA executables)
  - hw (Vivado based project)
    - constraints (contains user constraint files)
    - create_ip (contains files used to configure IP cores)
    - hdl (contains project-specific hdl code)
    - tcl (contains scripts used to run various tools)
  
- sw
  
- embedded (contains code for microblaze)
  
- host (contains code for host communication etc.)

- test (contains code for project verification)
## Reusable logic (IP cores)

<table>
<thead>
<tr>
<th>Category</th>
<th>IP Core(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O interfaces</td>
<td>Ethernet 10G Port&lt;br&gt; PCI Express&lt;br&gt; UART&lt;br&gt; GPIO</td>
</tr>
<tr>
<td>Output queues</td>
<td>BRAM based</td>
</tr>
<tr>
<td>Output port lookup</td>
<td>NIC&lt;br&gt; CAM based Learning switch</td>
</tr>
<tr>
<td>Memory interfaces</td>
<td>SRAM&lt;br&gt; DRAM&lt;br&gt; FLASH</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>FIFOs&lt;br&gt; AXIS width converter</td>
</tr>
</tbody>
</table>
Verification Infrastructure (1)

- Simulation and Debugging
  - built on industry standard Xilinx “xSim” simulator and “Scapy”
  - Python scripts for stimuli construction and verification
Verification Infrastructure (2)

- **xSim**
  - a High Level Description (HDL) simulator
  - performs functional and timing simulations for embedded, VHDL, Verilog and mixed designs

- **Scapy**
  - a powerful interactive packet manipulation library for creating “test data”
  - provides primitives for many standard packet formats
  - allows addition of custom formats
Build Infrastructure (2)

- Build/Synthesis (using Xilinx Vivado)
  - collection of shared hardware peripherals cores stitched together with AXI4: Lite and Stream buses
  - bitfile generation and verification using Xilinx synthesis and implementation tools
Build Infrastructure (3)

- Register system
  - collects and generates addresses for all the registers and memories in a project
  - uses integrated python and tcl scripts to generate HDL code (for hw) and header files (for sw)
implementation goes wild…
What’s a core?

• “IP Core” in Vivado
  - Standalone Module
  - Configurable and reuseable
• HDL (Verilog/VHDL) + TCL files
• Examples:
  - 10G Port
  - SRAM Controller
  - NIC Output port lookup
HDL (Verilog)

- NetFPGA cores
  - AXI-compliant
- AXI = Advanced eXtensible Interface
  - Used in ARM-based embedded systems
  - Standard interface
  - **AXI4/AXI4-Lite**: Control and status interface
  - **AXI4-Stream**: Data path interface
- Xilinx IPs and tool chains
  - Mostly AXI-compliant
Scripts (TCL)

- integrated into Vivado toolchain
  - Supports Vivado-specific commands
  - Allows to interactively query Vivado
- Has a large number of uses:
  - Create projects
  - Set properties
  - Generate cores
  - Define connectivity
  - Etc.
Inter-Module Communication
Using AXI-4 Stream (*Packets are moved as Stream*)
### AXI4-Stream

<table>
<thead>
<tr>
<th>AXI4-Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATA</td>
<td>Data Stream</td>
</tr>
<tr>
<td>TKEEP</td>
<td>Marks qualified bytes (i.e. byte enable)</td>
</tr>
<tr>
<td>TVALID</td>
<td>Valid Indication</td>
</tr>
<tr>
<td>TREADY</td>
<td>Flow control indication</td>
</tr>
<tr>
<td>TLAST</td>
<td>End of packet/burst indication</td>
</tr>
<tr>
<td>TUSER</td>
<td>Out of band metadata</td>
</tr>
</tbody>
</table>
## Packet Format

<table>
<thead>
<tr>
<th>TLAST</th>
<th>TUSER</th>
<th>TKEEP</th>
<th>TDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>V</td>
<td>0xFF…F</td>
<td>Eth Hdr</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0xFF…F</td>
<td>IP Hdr</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0xFF…F</td>
<td>…</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0x0…1F</td>
<td>Last word</td>
</tr>
<tr>
<td>Position</td>
<td>Content</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[15:0]</td>
<td>length of the packet in bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[23:16]</td>
<td>source port: one-hot encoded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[31:24]</td>
<td>destination port: one-hot encoded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[127:32]</td>
<td>6 user defined slots, 16bit each</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TVALID/TREADY Signal timing

- No waiting!
- Assert TREADY/TVALID whenever appropriate
- TVALID should not depend on TREADY
Byte ordering

- In compliance to AXI, NetFPGA has a specific byte ordering
  - 1st byte of the packet @ TDATA[7:0]
  - 2nd byte of the packet @ TDATA[15:8]
Developing a project
Embedded Development Kit

- Xilinx integrated design environment contains:
  - **Vivado**, a top level integrated design tool for “hardware” synthesis, implementation and bitstream generation
  - **Software Development Kit (SDK)**, a development environment for “software application” running on embedded processors like Microblaze
  - **Additional tools** (e.g. Vivado HLS)
Xilinx Vivado

A Vivado project consists of following:

- `<project_name>.xpr`
  - top level Vivado project file
- `tcl and HDL files that define the project`
- `system.xdc`
  - user constraint file
  - defines constraints such as timing, area, IO placement etc.
Xilinx Vivado (2)

- To invoke Vivado design tool, run:
  
  ```
  # vivado <project_root>/hw/project/<project_name>.xpr
  ```

- This will open the project in the Vivado graphical user interface
  
  - open a new terminal
  
  - cd <project_root>/projects/ <project_name>/
  
  - source /opt/Xilinux/Vivado/2016.4/settings64.sh
  
  - vivado hw/project/<project name>.xpr
Vivado Design Tool (1)

Flow Navigation

Design

Project Summary
Vivado Design Tool (2)

- IP Catalog: contains categorized list of all available peripheral cores
- IP Integrator: shows connectivity of various modules over AXI bus
- Project manager: provides a complete view of instantiated cores
Vivado Design Tool (3)

- **Address Editor:**
  - Under IP Integrator
  - Defines base and high address value for peripherals connected to AXI4 or AXI-LITE bus
  - Not AXI-Stream!
- These values can be controlled manually, using tcl
Getting started with a project (1)

- Each design is represented by a project
  - Location: NetFPGA-SUME-live/projects/<proj_name>
- Create a new project:
  - Normally:
    - Copy an existing project as the starting point
- Consists of:
  - Verilog source
  - Simulation tests
  - Hardware tests
  - Optional software
Getting started with a project (2)

- Shared modules included from netfpga/lib/hw
  - Generic modules that are re-used in multiple projects
  - Specify shared modules in project’s tcl file
Getting started with a project (3)

Preparing a module:

1. cd $IP_FOLDER/<ip name>
2. Write and edit files under <ip_name>/hdl Folder
3. make

Notes:
1. review ~/NetFPGA-SUME-live/tools/settings.sh
2. If you make changes: source ~/NetFPGA-SUME-live/tools/settings.sh
3. Check that make passes without errors
Register Infrastructure
Registers

- Registers system:
  - Automatically generated
  - Implementing registers in a module
    - Automatically generated cpu_regs module
  - Need to implement the registers’ functional logic
Registers bus

- Register communication follows the AXI4-Lite paradigm
- The AXI4-Lite interface provides a point-to-point bidirectional interface between a user Intellectual Property (IP) core and the AXI Interconnect
Register bus (AXI4-Lite interface)
Register bus

AXI LITE INTERCONNECT

AXI4-Lite Interface

<module>_cpu_regs

{registers signals}

user-defined module
## Registers – Module generation

- Spreadsheet based (xls / csv)
- Defines all the registers you intend to support and their properties
- Generates a python script (regs_gen.py), which generates the outputs
- Location: $SUME_FOLDER/tools/infrastructure

### OS: Windows

<table>
<thead>
<tr>
<th>Block</th>
<th>Register Name</th>
<th>Address</th>
<th>Description</th>
<th>Type</th>
<th>Bits</th>
<th>Endian Type</th>
<th>Access Mode</th>
<th>Valid for sub-modules</th>
<th>Default</th>
<th>Constraints, Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_name</td>
<td>Init</td>
<td>NA</td>
<td>When triggered, the module will perform SW reset</td>
<td>Global</td>
<td>0</td>
<td>Little</td>
<td></td>
<td>sub_ip_name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>ID</td>
<td>0</td>
<td>The ID of the module, to make sure that one accesses the right module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h0000DA03</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Version</td>
<td>4</td>
<td>Version of the module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h1</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Flip</td>
<td>8</td>
<td>The register returns the opposite value of what was written to it</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td>Returned value is at reset 32'hFFFFFFF</td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterIn</td>
<td>C</td>
<td>Number of incoming packets through the Field</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOvf</td>
<td>C</td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opf</td>
<td>1'b0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOut</td>
<td>10</td>
<td>Number of outgoing packets through the Field</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOvf</td>
<td>C</td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opf</td>
<td>1'b0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Debug</td>
<td>14</td>
<td>Debug Register, for simulation and debug</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>EndianEg</td>
<td>18</td>
<td>Example big endian register</td>
<td>Reg</td>
<td>31:0</td>
<td>Big</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
</tbody>
</table>
### Registers – Module generation

<table>
<thead>
<tr>
<th>Block</th>
<th>Register Name</th>
<th>Address</th>
<th>Description</th>
<th>Type</th>
<th>Bits</th>
<th>Endian Type</th>
<th>Access Mode</th>
<th>Valid for sub-modules</th>
<th>Default</th>
<th>Constraints, Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_name</td>
<td>Init</td>
<td>NA</td>
<td>When triggered, the module will perform SW reset</td>
<td>Global</td>
<td>0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h0000DA03</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>ID</td>
<td>0</td>
<td>The ID of the module, to make sure that one accesses the right module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h1</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Version</td>
<td>4</td>
<td>Version of the module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td>Returned value is at reset 32'hFFFFFFFF</td>
</tr>
<tr>
<td>IP_name</td>
<td>Flip</td>
<td>8</td>
<td>The register returns the opposite value of what was written to it</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterIn</td>
<td>C</td>
<td>Incoming Packets Counter</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterInOvf</td>
<td></td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opl</td>
<td>31'h0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOut</td>
<td>10</td>
<td>Outgoing Outgoing Packets Counter</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOutOvf</td>
<td></td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opl</td>
<td>1'b0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Debug</td>
<td>14</td>
<td>Debug Register, for simulation and debug</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>EndianEg</td>
<td>18</td>
<td>Example big endian register</td>
<td>Reg</td>
<td>31:0</td>
<td>Big</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
</tbody>
</table>

**OS:** Windows
Access Modes:

- RO - Read Only (by SW)
- ROC - Read Only Clear (by SW)
- WO - Write Only (by SW)
- WOE - Write Only Event (by SW)
- RWS - Read/Write by SW
- RWA - Read/Write by HW and SW
- RWCR - Read/Write clear on read (by SW)
- RW CW - Read/Write clear on write (by SW)
Registers – Module generation

Endian Mode:
• Little Endian – Most significant byte is stored at the highest address
  - Mostly used by CPUs
• Big Endian - Most significant byte is stored at the lowest address
  - Mostly used in networking
  - e.g. IPv4 address
Registers – Generated Modules

- `<module>_cpu_regs.v` – Interfaces AXI-Lite to dedicated registers signals
  To be placed under under `<core name>/hdl`

- `<module>_cpu_regs_defines.v` – Defines per register: width, address offset, default value
  To be placed under under `<core name>/hdl`

- `<module>_cpu_template.v` – Includes template code to be included in the top core Verilog.
  This file can be discarded after updating the top core verilog file.
Registers – Generated Modules

Same contents as `<module>_cpu_regs_defines.v`, but in different formats, used by software, build and test harness:

- `<module>_regs_defines.h`
  To be placed under `<core name>/data`

- `<module>_regs_defines.tcl`
  To be placed under `<core name>/data`

- `<module>_regs_defines.txt` – used by test harness
  To be placed under `<core name>/data`
Adding Registers Logic - Example

• Usage examples:

```verilog
always @(posedge axi_aclk)
  begin
    if (~resetn_sync)
    begin
      id_reg <= #1 `REG_ID_DEFAULT;
      ip2cpu_flip_reg <= #1 `REG_FLIP_DEFAULT;
      pktin_reg <= #1 `REG_PKTIN_DEFAULT;
    end
  end
else begin
  id_reg <= #1 `REG_ID_DEFAULT;
  ip2cpu_flip_reg <= #1 ~cpu2ip_flip_reg;
  pktin_reg <= #1 pktin_reg_clear ? 'h0 :
                                pkt_in ? pktin_reg + 1: pktin_reg;
end
```
NetFPGA-Host Interaction

• Register reads/writes via ioctl system call

• Useful command line utilities

```bash
cd $APPS_FOLDER/sume_riffa_v1_0_0/
./rwaxi -a 0x44010000
./rwaxi -a 0x44010000 -w 0x1234
```

You must program the FPGA and load the driver before using these commands!
Can I collect the registers addresses in a unique .h file?
NetFPGA-Host Interaction

• Need to create the sume_register_defines.h file
  
  • cd $NF_DESIGN_DIR/hw
  
  • make reg

• The sume_register_defines.h file will be placed under $NF_DESIGN_DIR/sw/embedded/src
NetFPGA-Host Interaction

Required steps:

• Generate .h file per core
  • Automatically generated by the python script

• Edit $NF_DESIGN_DIR/hw/tcl/
  $NF_PROJECT_NAME_defines.tcl
  • Indicate the address mapping you use

• Edit $NF_DESIGN_DIR/hw/tcl/
  export_regiters.tcl
  • Indicate the location of all IP cores used
    • Default path assumed is under \lib\hw\cores
NetFPGA-Host Interaction

- `sume_register_defines.h` is automatically generated when creating a project
  - Using NetFPGA TCL scripts, the `.h` file will match the hardware
  - Note that changes in the GUI will not be reflected!

- Post implementation, for the SDK, use
  ```
  $NF_DESIGN_DIR/hw/tcl/export_hardware.tcl
  ```
  - Uses vivado’s export
  - Does not include the registers list, only memory map
Step by step

1. In Libreoffice set security to medium
2. Open tools/infrastructure/module_generation.xls
   or $IP_FOLDER/module_name/data/module_generation.xls
3. Change block name to match your module name
   (for sub-module this is optional)
4. Delete all indirect registers (and others you don’t want)
   (note potential issues in some releases)
5. Change OS to Linux
6. Press “Generate Registers”
7. From console, run python regs_gen.py
8. cp *.v $IP_FOLDER/module_name/hdl
9. cp <*.tcl,*.h,*.txt> $IP_FOLDER/module_name/data
10. Copy lines from template file to module_name.v
11. Add in module_name.v support for functionality
Testing Registers with Simulation

• `nftest_regwrite(address, value)`
  • `nftest_regwrite(0x44010008, 0xABCD)`
 • `nftest_regread(address)`
  • `nftest_regread(0x44010000)`
 • `nftest_regread_expect(address, expected_value)`
  • `nftest_regread_expect(0x44010000, 0xDA01)`
 • Can use registers names
  • `nftest_regread(SUME_INPUT_ARBITER_0_ID)`
 • Used within run.py
 • You don’t need to edit any other file
Simulating Register Access

1. Define register stimulus

2. The testbench executes the stimulus

3. Simulation accesses are written to a log file

4. A script can compare expected and actual values
   And declare success or failure

Legend:
- DUT: Design Under Test
- stim: stimulus
- tb: testbench
- sim: simulation
cd $NF_DESIGN_DIR/test/
less reg_stim.axi

- An example of write format:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Byte Enable</th>
<th>strobe</th>
</tr>
</thead>
<tbody>
<tr>
<td>77000000</td>
<td>0eadc0de</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000004</td>
<td>acce55ed</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000008</td>
<td>add1c7ed</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>7700000c</td>
<td>ca0ebabe</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000010</td>
<td>c0dedead</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000014</td>
<td>55edacce</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000018</td>
<td>babeca1e</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>7700001c</td>
<td>abcd9ab</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000020</td>
<td>cde2abcd</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>77000024</td>
<td>e4abcde3</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

with other useful information like, time, barriers etc..
cd $NF_DESIGN_DIR/test/both_learning_sw

less reg_stim.axi

- An example read format:

```
# Ten DWORD quick reads from the nic_output_port_loopup interface (without waits.)

77000000
77000004
77000008
7700000c
77000010
77000014
77000018
7700001c
77000020
77000024

# Never wrap addresses until after WAIT flag!
```

with other useful information like, time, barriers etc..
cd $NF_DESIGN_DIR/test/both_learning_sw
less reg_stim.log

<table>
<thead>
<tr>
<th>Time</th>
<th>WRITE</th>
<th>READ</th>
</tr>
</thead>
<tbody>
<tr>
<td># 1335 ns</td>
<td>DEADC0DE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 1405 ns</td>
<td>ACCE55ED (OKAY)</td>
<td>BABECA1E (OKAY)</td>
</tr>
<tr>
<td># 1475 ns</td>
<td>ADD1C7ED (OKAY)</td>
<td>ABCDE9AB (OKAY)</td>
</tr>
<tr>
<td># 1545 ns</td>
<td>CA0EBABE (OKAY)</td>
<td>CDE2ABCD (OKAY)</td>
</tr>
<tr>
<td># 1615 ns</td>
<td>55EDACCE (OKAY)</td>
<td>E4ABCDE3 (OKAY)</td>
</tr>
<tr>
<td># 1685 ns</td>
<td>BABECA1E (OKAY)</td>
<td>55EDACCE (OKAY)</td>
</tr>
<tr>
<td># 1755 ns</td>
<td>DEADC0DE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 1825 ns</td>
<td>ACCE55ED (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 1895 ns</td>
<td>ADD1C7ED (OKAY)</td>
<td>55EDACCE (OKAY)</td>
</tr>
<tr>
<td># 1965 ns</td>
<td>CA0EBABE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2035 ns</td>
<td>55EDACCE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2095 ns</td>
<td>BABECA1E (OKAY)</td>
<td>55EDACCE (OKAY)</td>
</tr>
<tr>
<td># 2155 ns</td>
<td>DEADC0DE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2215 ns</td>
<td>ACCE55ED (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2275 ns</td>
<td>ADD1C7ED (OKAY)</td>
<td>55EDACCE (OKAY)</td>
</tr>
<tr>
<td># 2335 ns</td>
<td>CA0EBABE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2395 ns</td>
<td>55EDACCE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2455 ns</td>
<td>BABECA1E (OKAY)</td>
<td>55EDACCE (OKAY)</td>
</tr>
<tr>
<td># 2515 ns</td>
<td>DEADC0DE (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
<tr>
<td># 2575 ns</td>
<td>ACCE55ED (OKAY)</td>
<td>C0DEDEAD (OKAY)</td>
</tr>
</tbody>
</table>
P4-NetFPGA: Adding table entries

```
cd $P4_PROJECT_DIR
Vim commands.txt

table_cam_add_entry lookup_table set_result 0x00000000 => 0x00000001
table_cam_add_entry lookup_table set_result 0x00000001 => 0x000000010
table_cam_add_entry lookup_table set_result 0x00000002 => 0x000000100
```
P4-NetFPGA: Adding table entries

- Supported table types: cam / tcam / lpm (direct currently not supported by api)

- Common commands:
  read_entry, delete_entry, add_entry
  - Commands vary between different table types

- Full list of api:
  /contrib-projects/sume-sdnet-switch/templates/CLI_template/p4_tables_api.py
P4-NetFPGA: Register Access

- Supported commands: reg_read, reg_write
- Accesses an array of registers
- Source code:
  /contrib-projects/sume-sdnet-switch/templates/CLI_template/p4_regs_api.py
  - reg_read(reg_name, index)
  - reg_write(reg_name, index, val)
Section II: Testing Hardware
• To synthesize your project:

```
  cd $NF_DESIGN_DIR

  make
```
Hardware Tests

• Test compiled hardware

• Test infrastructure provided to
  • Send Packets
  • Check Counters
  • Read/Write registers
  • Read/Write tables
Python Libraries

- Start packet capture on interfaces
- Clear all tables in hardware
- Create packets
  - MAC header
  - IP header
  - PDU
Creating a Hardware Test

Useful functions:

Packet generation:
  make_IP_pkt(...) – see wiki
  encrypt_pkt(key, pkt)
  decrypt_pkt(key, pkt)

Packet transmission/reception:
  nftest_send_phy(interface, pkt)
  nftest_expect_phy(interface, pkt)
  nftest_send_dma(interface, pkt)
  nftest_expect_dma(interface, pkt)

Register access:
  nftest_regwrite(addr, value)
  nftest_regread_expect(addr, expect)
Understanding Hardware Test

- cd $NF_DESIGN_DIR/test/both_learning_sw
- vim run.py
- “isHW” indicates HW test
- “connections/conn” file declares the physical connections
  
  nf0: eth1
  nf1: eth2
  nf2:
  nf3:

- “global/setup” file defines the interfaces

  proc = Popen(['ifconfig', 'eth2', '192.168.101.1'], stdout=PIPE)

Your task:
- Remember to source the settings.sh file
- Edit run.py to create your test
- Edit setup and conn files
Running Hardware Tests

• Use command nf_test.py

• Required Parameter
  • sim hw or both (right now only use hw)

• Optional parameters
  • --major <major_name>
  • --minor <minor_name>
    both_learning_sw
    major minor
Running Hardware Tests

• Having problems?

• Take advantage of the wiki!

• Detailed explanations

• Tips for debug