Rapid Prototyping of High Bandwidth Devices in Open Source

Presented by:

Noa Zilberman, Yury Audzevich

University of Cambridge

August 31st, 2015

http://NetFPGA.org

Tutorial Outline

- **Open Source Hardware**
  - Introduction
  - Challenges
- **The NetFPGA platform**
  - Introduction
- **NetFPGA Hardware Overview**
  - Overview of NetFPGA Platforms
  - NetFPGA SUME
- **Life of a Packet**
- **Examples of Using NetFPGA**
- **Infrastructure**
  - Tree
  - Verification Infrastructure
- **Example Project**
  - Introduction
  - What is an IP core?
  - Getting started with a new project.
- **Simulation and Debug**
  - Write and Run Simulations
- **What to do next**
  - Available Resources
  - Getting Started
- **Concluding Remarks**

Section I: Open Source Hardware

Open? What is it anyway?

- **“I can see inside”**
- **Open Source Software**
  - Source code is available to the public
- **Open Standards**
  - A standard that is publicly available
  - Does not mean open access…
  - IETF, IEEE, ITU-T, …

Open Source Hardware

- **Can mean so many things…**
  - Firmware code
  - SoC design
  - Programmable logic design
  - Board design schematics
  - Board design layout
  - Board design *gerbers*
  - FPGA design
    - HDL code
    - Compiled outputs
    - Generated projects
  - ….

Open Source License

- **License ≠ Copyrights**
  - Copyright – the rights that you get in your work.
    - You have the exclusive right to *copy, distribute, display, perform, and make derivative works* based on your original work.
    - Copyright assignment – you give someone your copyrights.
  - License – Gives the other party permission to use some or all of your copyright rights.
    - You retain ownership of your copyrights.

Open Source License

- **Open source license:**
  - What you can do;
  - How you can redistribute the software/hardware.

- **The are many types of open source license**
  - Apache, BSD, GPL, MIT, Mozilla,…

- **Check carefully which one is right for you!**

- **… Not necessarily adequate for *Hardware***

Change the world? *Sure…..*

Open Source Networking

open-standard, vendor-independent APIs & designs

Core idea: SDN (Software Defined Networking)

SDN arose on the back of OpenFlow

Open Flow: a NetFPGA project

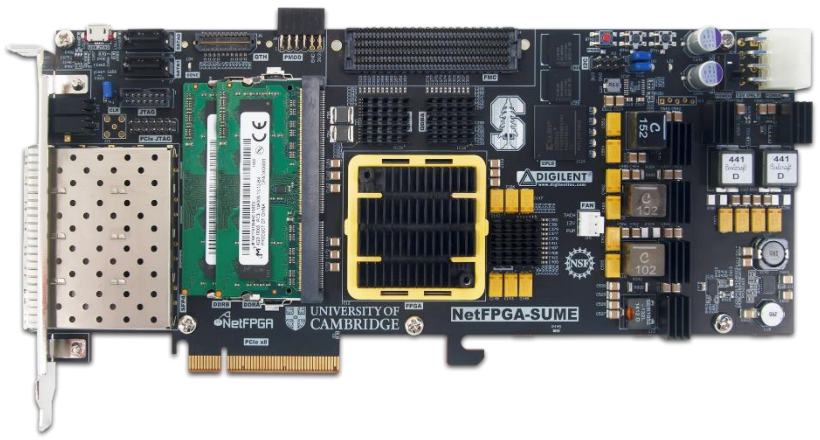
NetFPGA as a complete example

- Software, hardware, toolchain, platform
- Documentation
  - How to contribute?
- Super supporters
- Community & volunteers
- Planning for longevity
- ~~Hard~~

Section II: The NetFPGA platform

NetFPGA = Networked FPGA

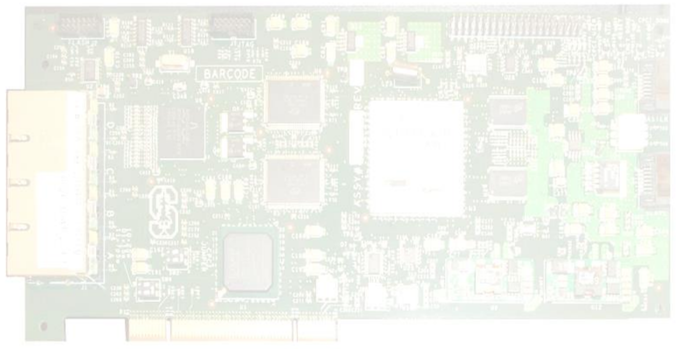
A line-rate, flexible, open networking platform for teaching and research



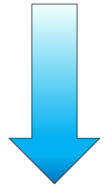
=

- Network Interface Card
- Hardware Accelerated Linux Router
- IPv4 Reference Router
- Traffic Generator
- Openflow Switch
- More Projects
- Add Your Project

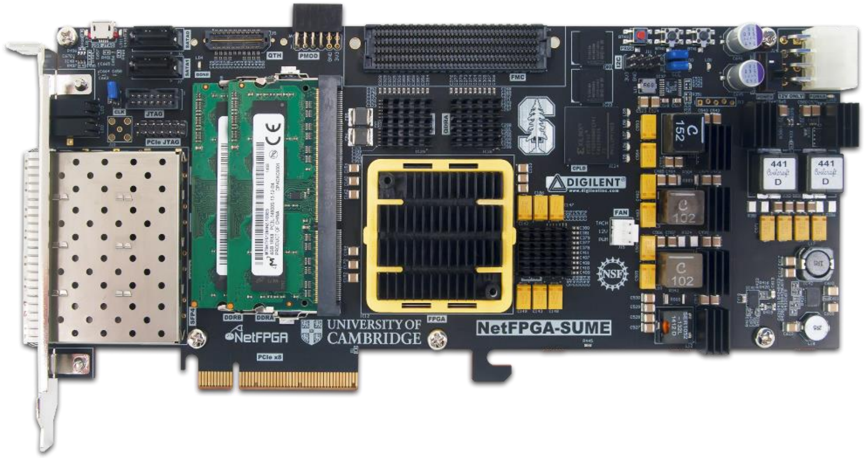
NetFPGA Family of Boards


NetFPGA-1G (2006)

NetFPGA-10G (2010)



NetFPGA-1G-CML (2014)


NetFPGA SUME (2014)

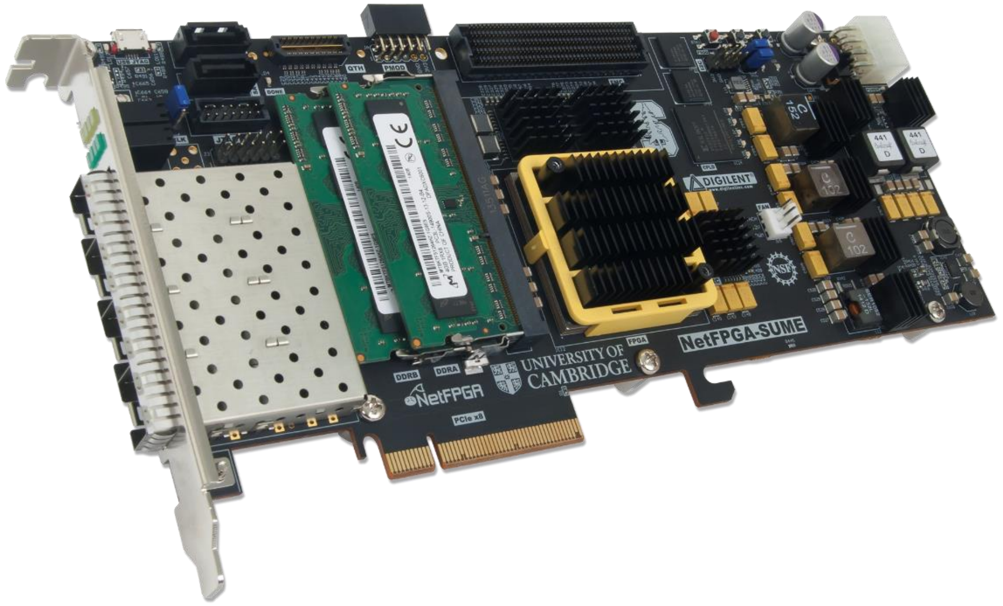
NetFPGA consists of…

Four elements:

NetFPGA GitHub Organization

The Interwebs http://www.netfpga.org

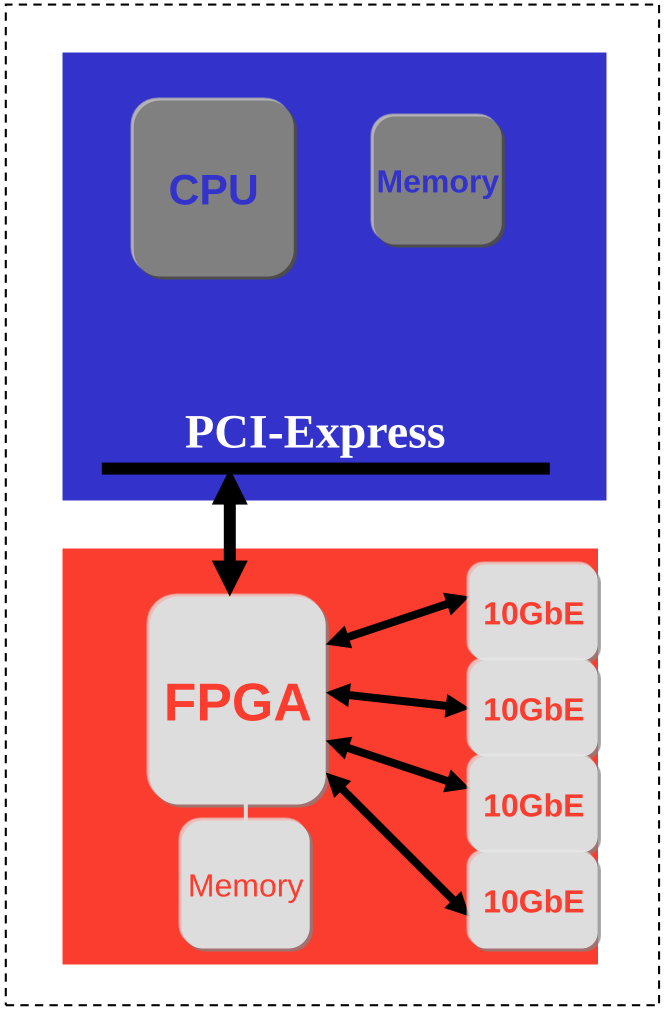
- NetFPGA board
- Tools + reference designs
- Contributed projects
- Community



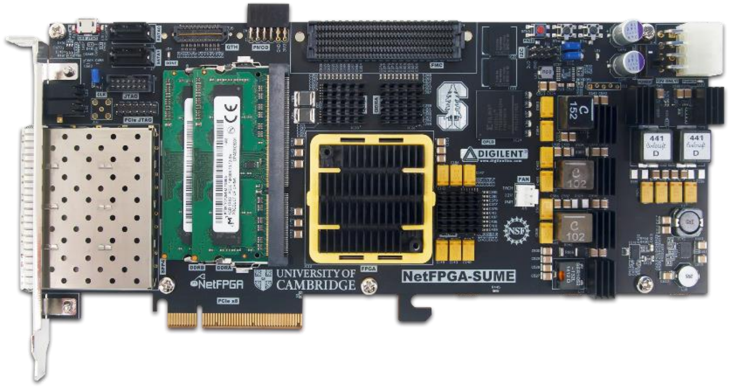
NetFPGA board

Networking Software running on a standard PC

A hardware accelerator built with Field Programmable Gate Array driving 1/10/ 100Gb/s network links



PC with NetFPGA



Tools + Reference Designs

Tools:

- Compile designs
- Verify designs
- Interact with hardware

Reference designs:

- Router (HW)
- Switch (HW)
- Network Interface Card (HW)
- Router Kit (SW)
- SCONE (SW)

Community

Wiki

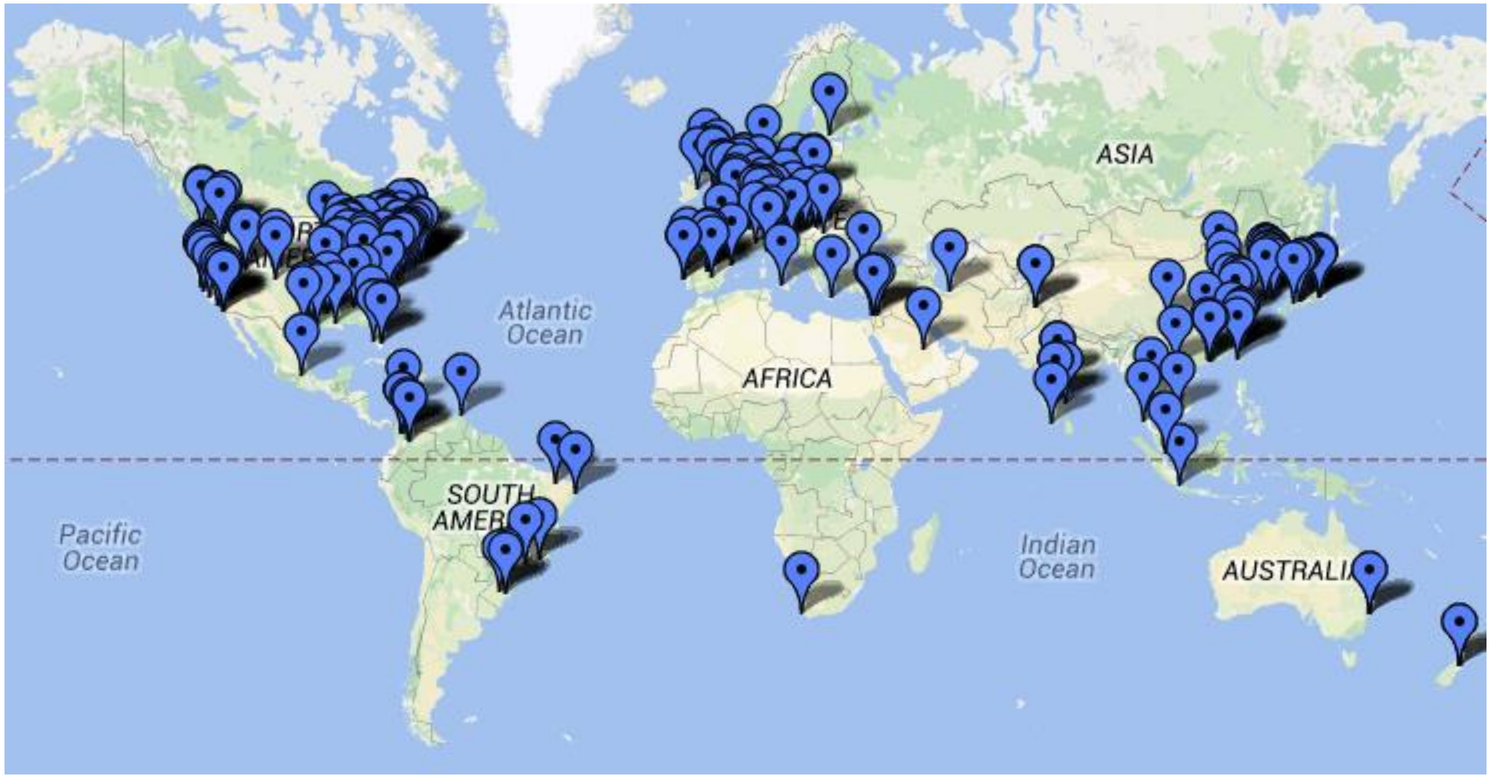
- **Documentation**
  - User's Guide *"so you just got your first NetFPGA"*
  - Developer's Guide *"so you want to build a …"*
- **Encourage users to contribute**

Forums

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

International Community

Over 1,200 users, using over 3500 cards at 150 universities in 40 countries



NetFPGA's Defining Characteristics

- **Line-Rate**
  - Processes back-to-back packets
    - Without dropping packets
    - At full rate
  - 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 for SUME, LGPL 2.1 for 10G
  - 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 to handle 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?

- Researchers
- Teachers
- Students

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

Section III: NetFPGA Hardware Overview

NetFPGA-1G-CML

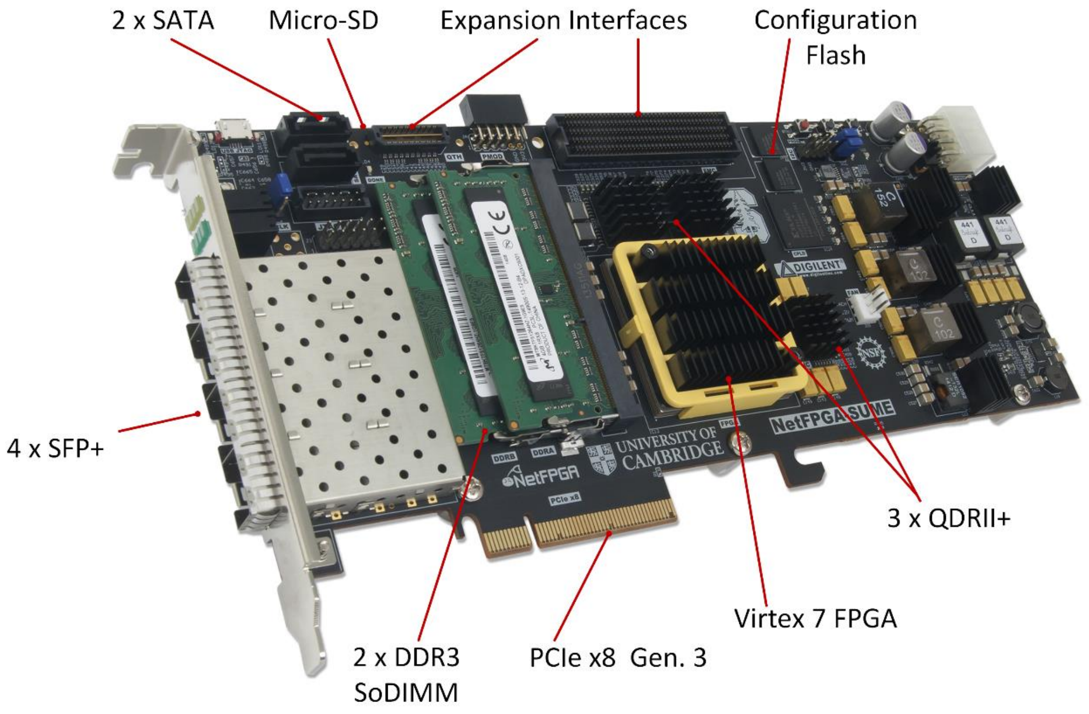
- FPGA Xilinx Kintex7
- 4x 10/100/1000 Ports
- PCIe Gen.2 x4
- QDRII+-SRAM, 4.5MB
- DDR3, 512MB
- SD Card
- Expansion Slot

NetFPGA-10G

- **FPGA Xilinx Virtex5**
- **4 SFP+ Cages**
  - 10G Support
  - 1G Support
- **PCIe Gen.1 x8**
- **QDRII-SRAM, 27MB**
- **RLDRAM-II, 288MB**
- **Expansion Slot**

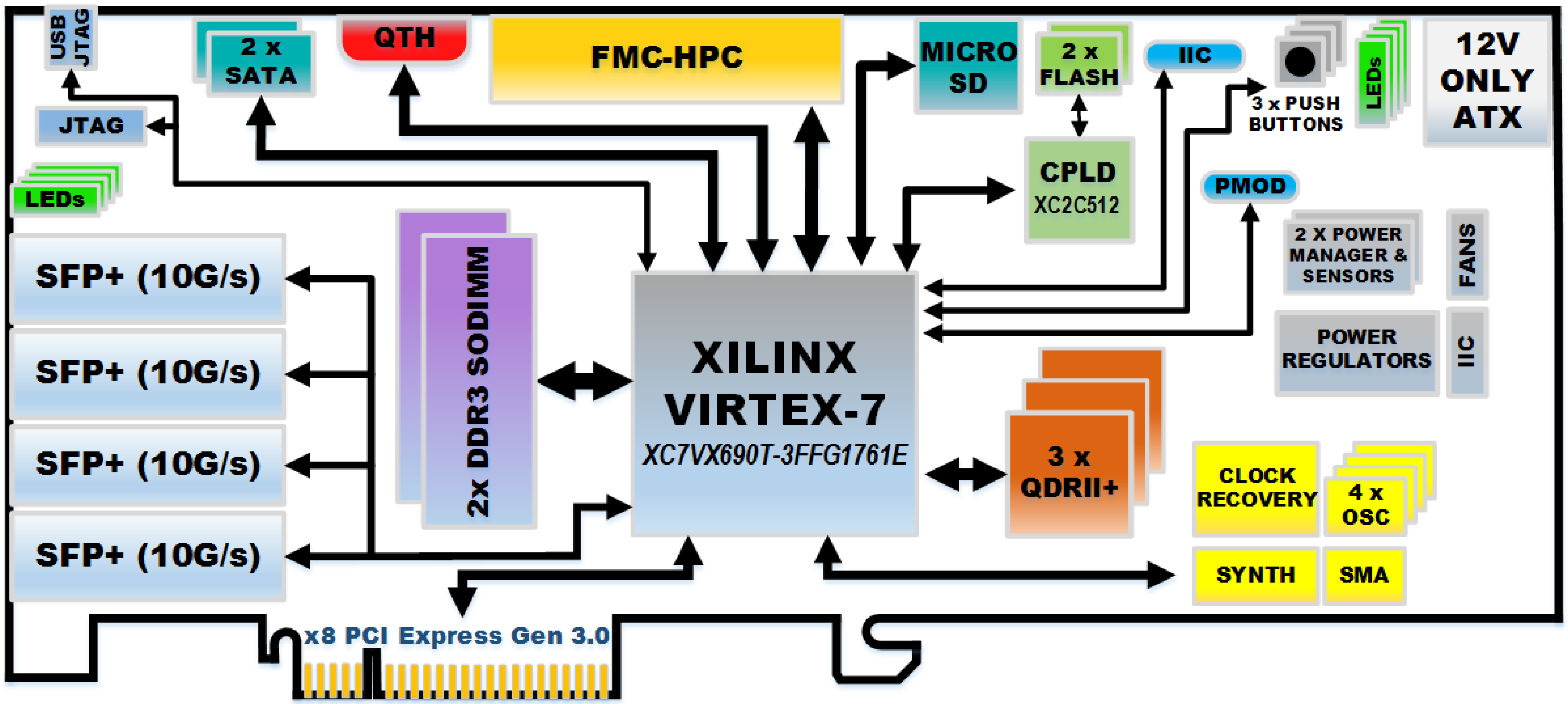
NetFPGA-SUME

- A major upgrade over the NetFPGA-10G predecessor
- State-of-the-art technology



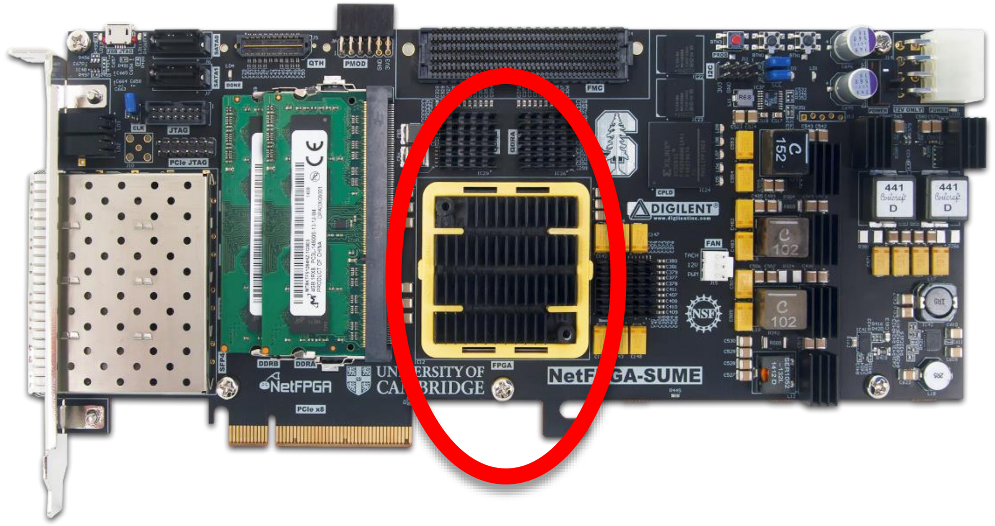
NetFPGA-SUME

- **High Level Block Diagram**



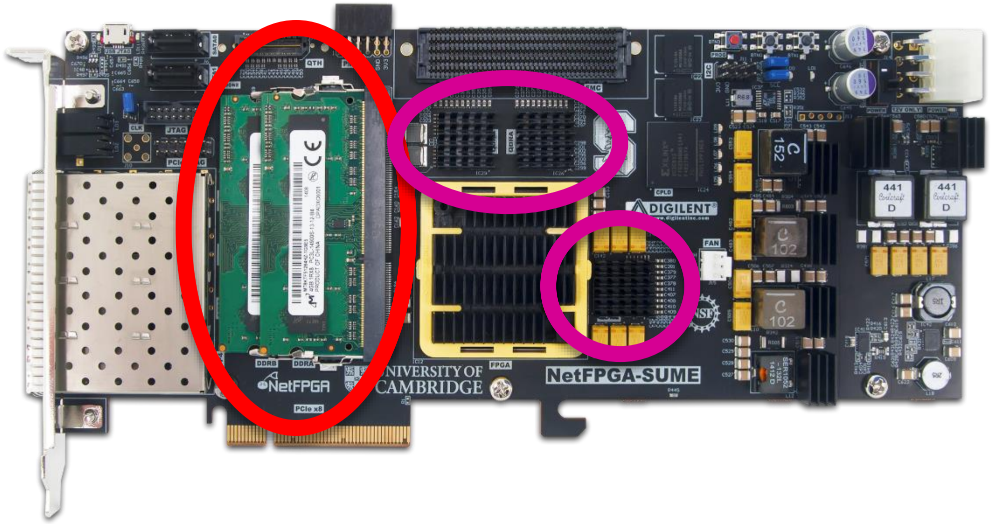
Xilinx Virtex 7 690T

- Optimized for high-performance applications
- 690K Logic Cells
- 52Mb RAM
- 3 PCIe Gen. 3 Hard cores



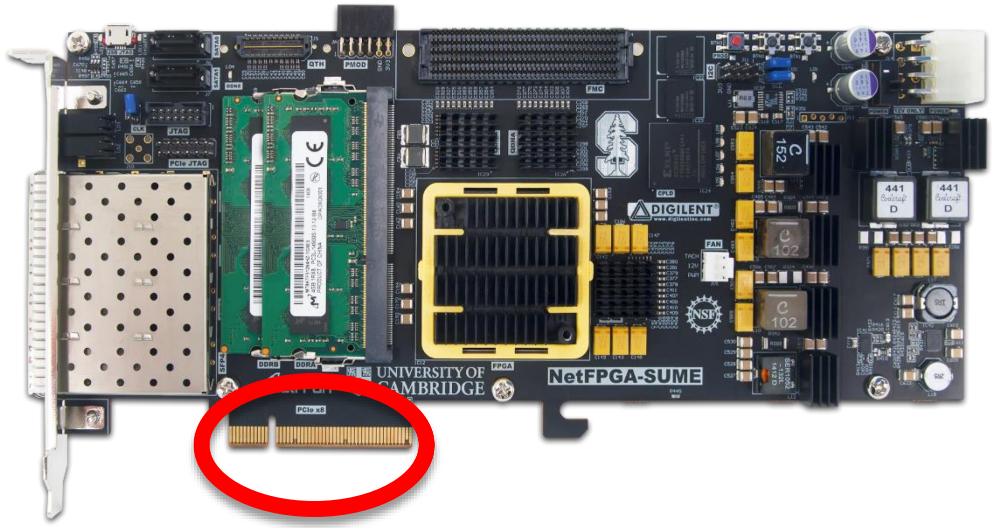
Memory Interfaces

- **DRAM:**
  2 x DDR3 SoDIMM
  1866MT/s, 4GB
- **SRAM:**
  3 x 9MB QDRII+,
  500MHz



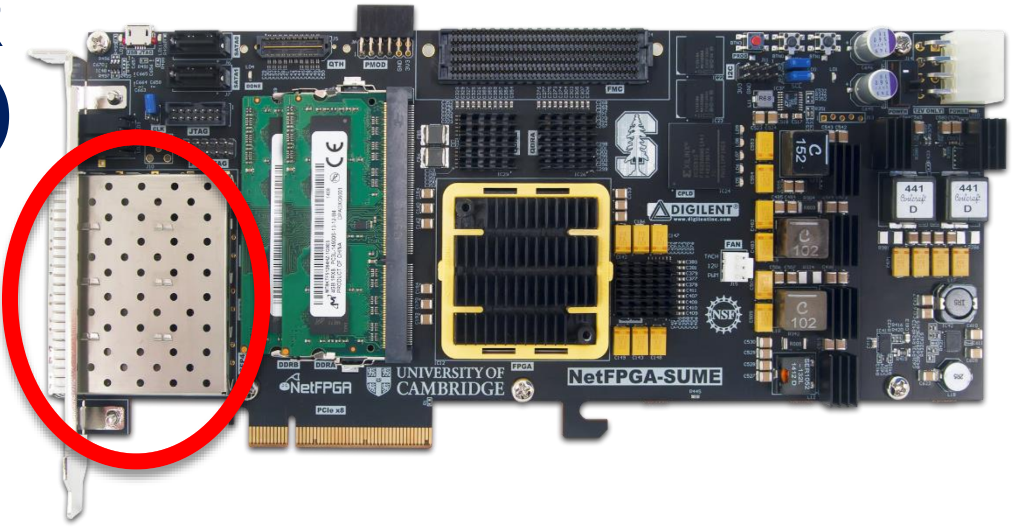
Host Interface

- PCIe Gen. 3
- x8 (only)
- Hardcore IP



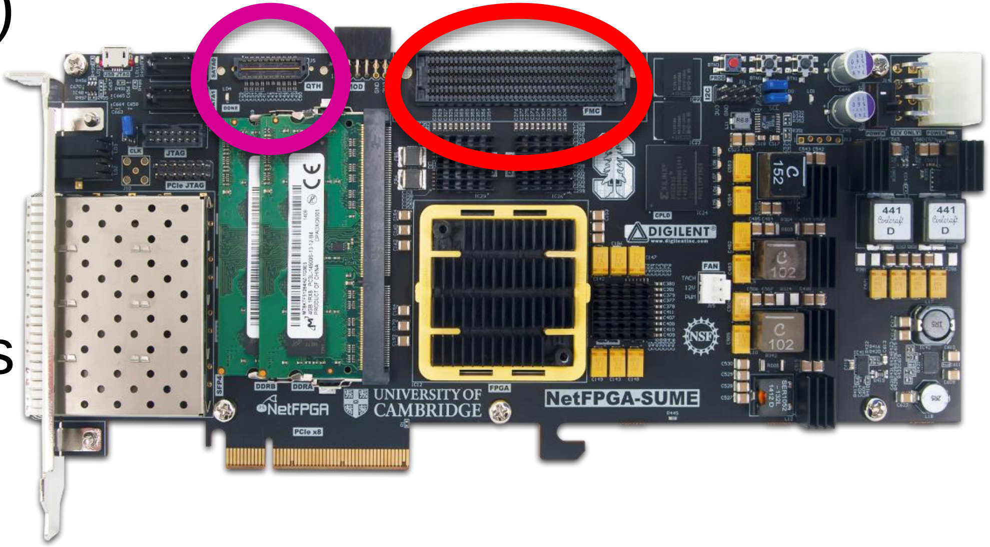
Front Panel Ports

- **4 SFP+ Cages**
- **Directly connected to the FPGA**
- **Supports 10GBase-R transceivers (default)**
- **Also Supports 1000Base-X transceivers and direct attach cables**



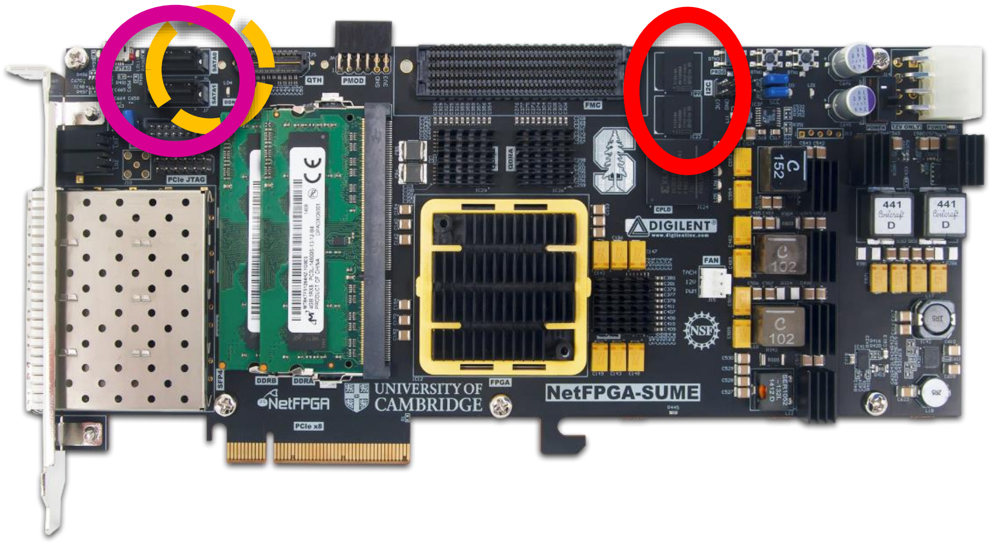
Expansion Interfaces

- **FMC HPC connector**
  - VITA-57 Standard
  - Supports Fabric Mezzanine Cards (FMC)
  - 10 x 12.5Gbps serial links
- **QTH-DP**
  - 8 x 12.5Gbps serial links

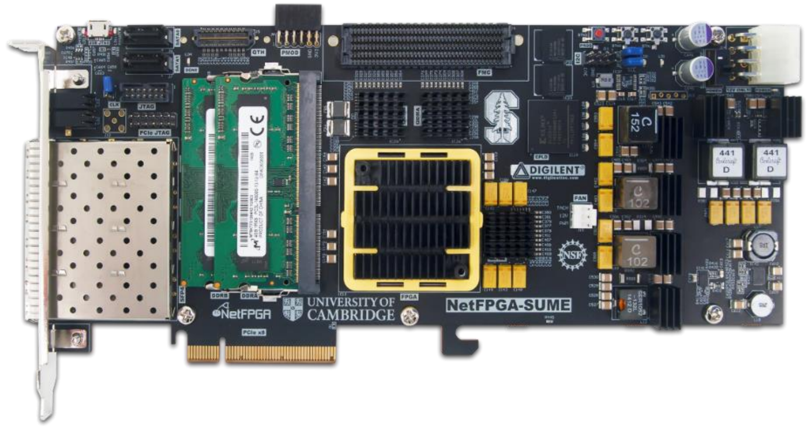


Storage

- 128MB FLASH
- 2 x SATA connectors
- Micro-SD slot
- Enable standalone operation

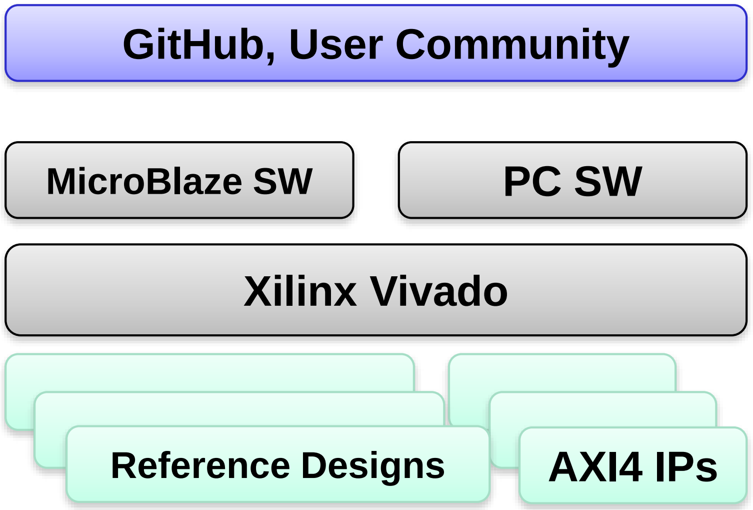


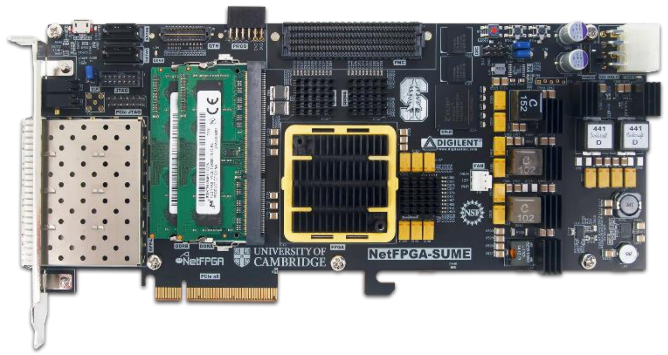
NetFPGA Board Comparison



NetFPGA SUME	NetFPGA 10G
Virtex 7 690T -3	Virtex 5 TX240T
8 GB DDR3 SoDIMM 1800MT/s	288 MB RLDRAM-II 800MT/s
27 MB QDRII+ SRAM, 500MHz	27 MB QDRII-SRAM, 300MHz
x8 PCI Express Gen. 3	x8 PCI Express Gen. 1
4 x 10Gbps Ethernet Ports	4 x 10Gbps Ethernet Ports
18 x 13.1Gb/s additional serial links	20 x 6.25Gb/s additional serial links

Beyond Hardware



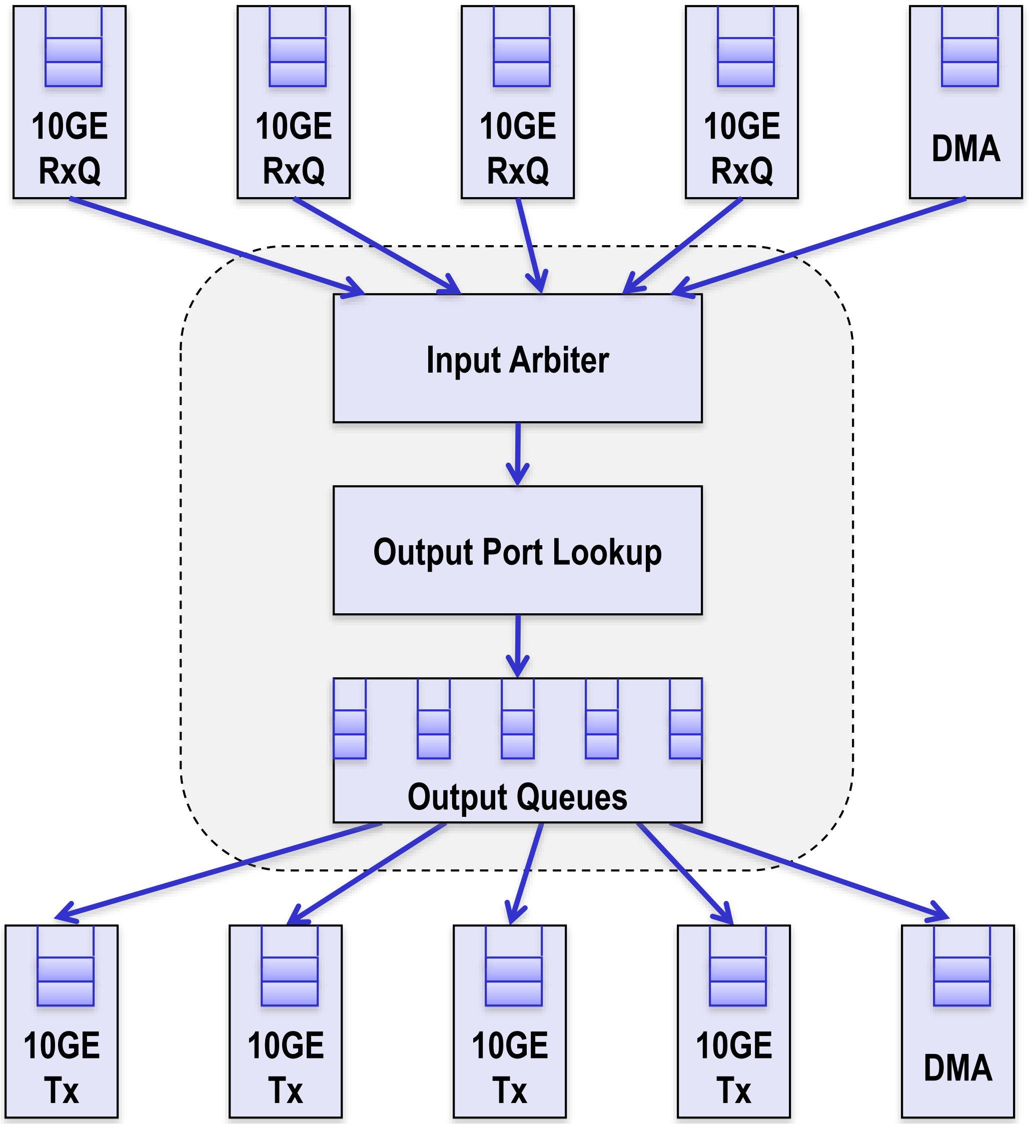


- NetFPGA Board
- Xilinx Vivado based IDE
- Reference designs using AXI4
- Software (embedded and PC)
- Public Repository
- Public Wiki

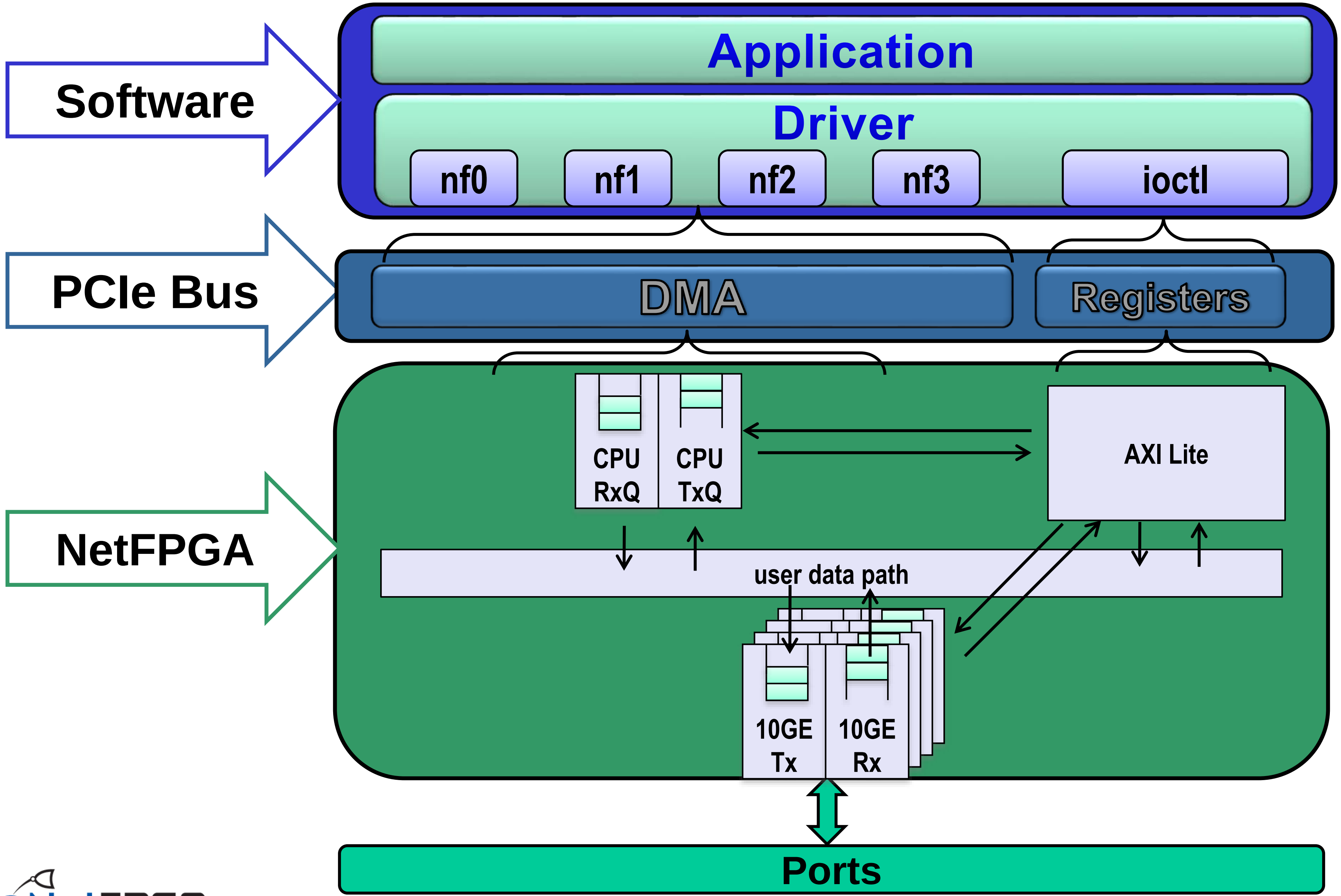
Section IV: Life of a Packet

Reference Switch Pipeline

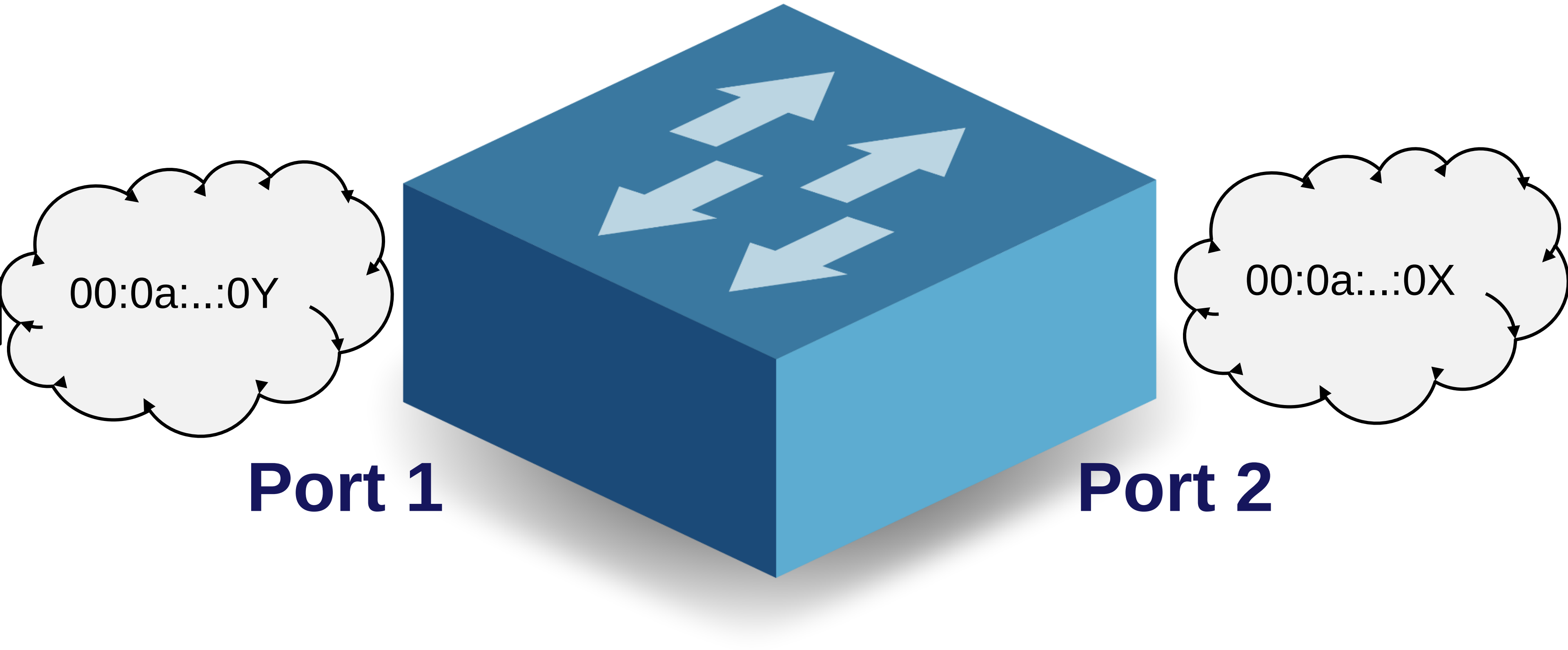
- **Five stages**
  - Input port
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output port
- **Packet-based module interface**
- **Pluggable design**



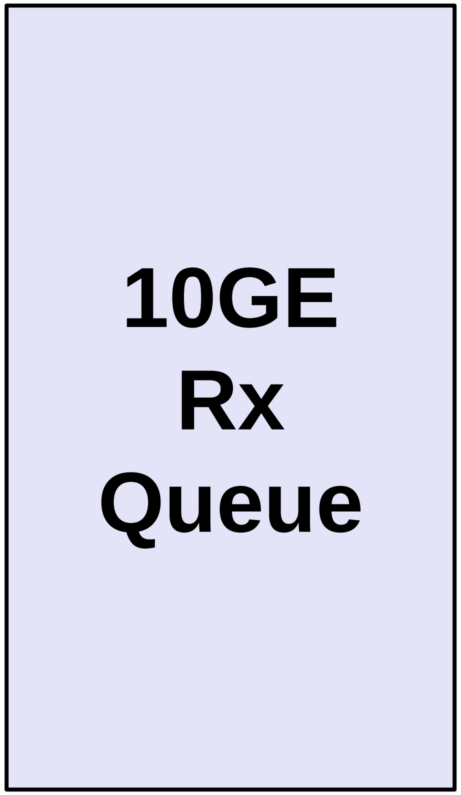
Full System Components



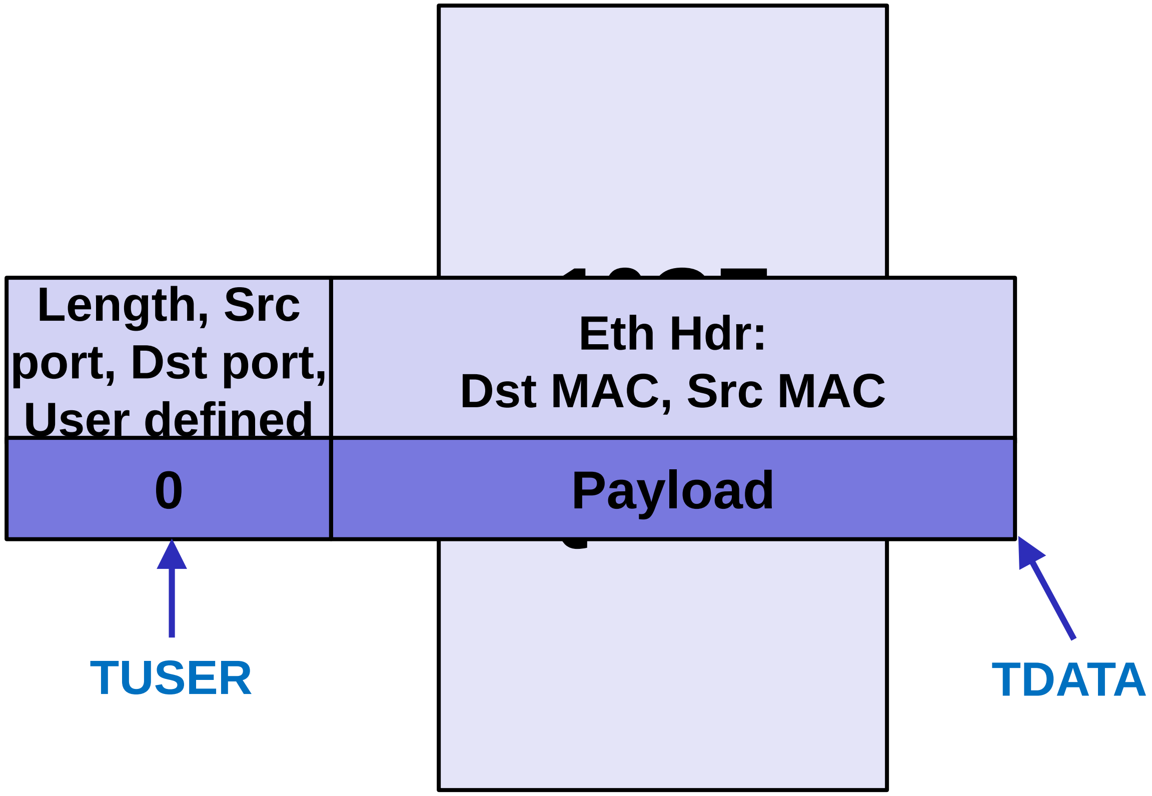
Life of a Packet through the Hardware



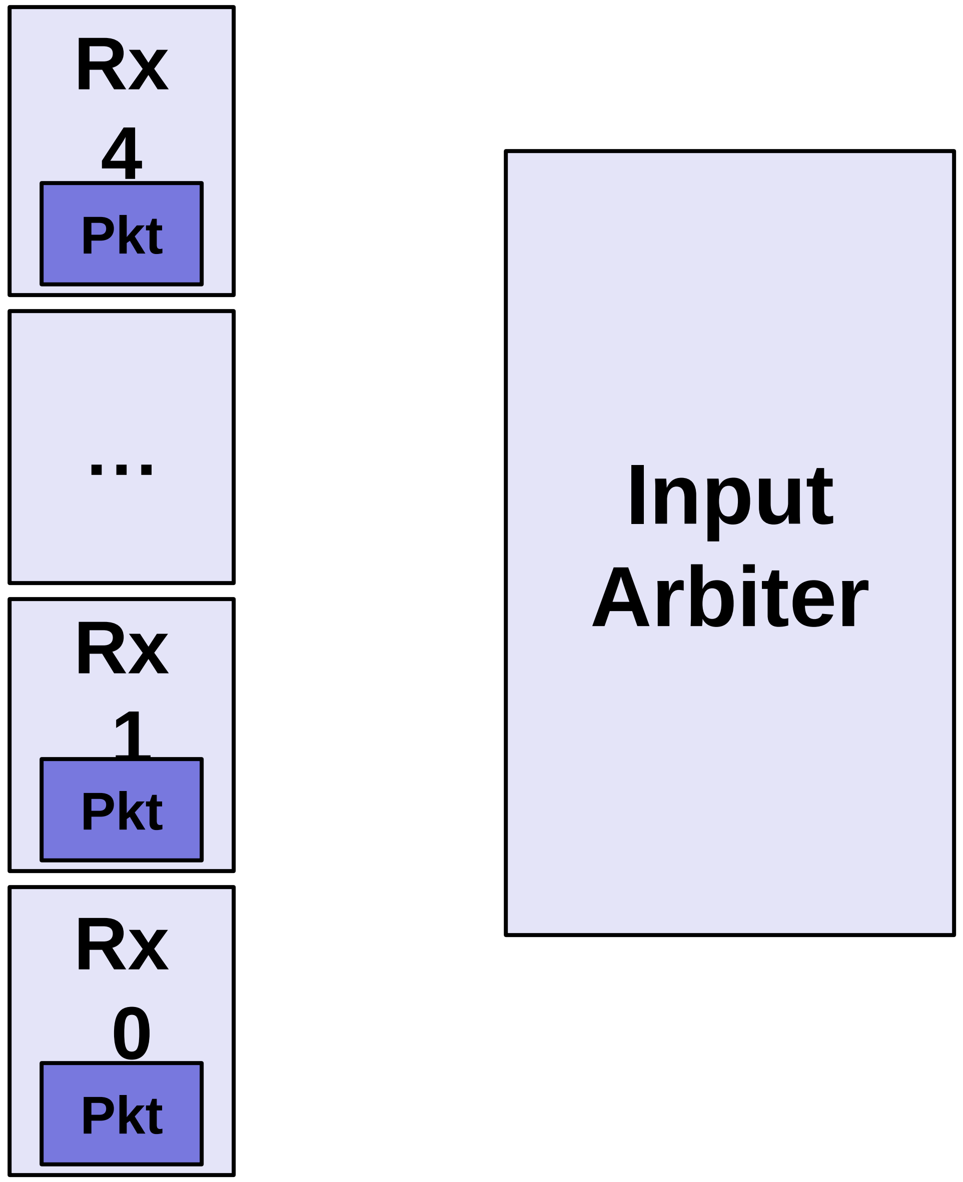
10GE Rx Queue



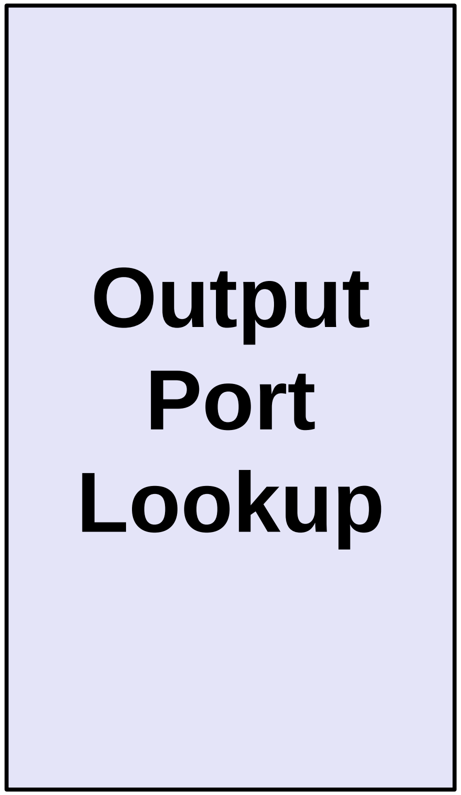
10GE Rx Queue



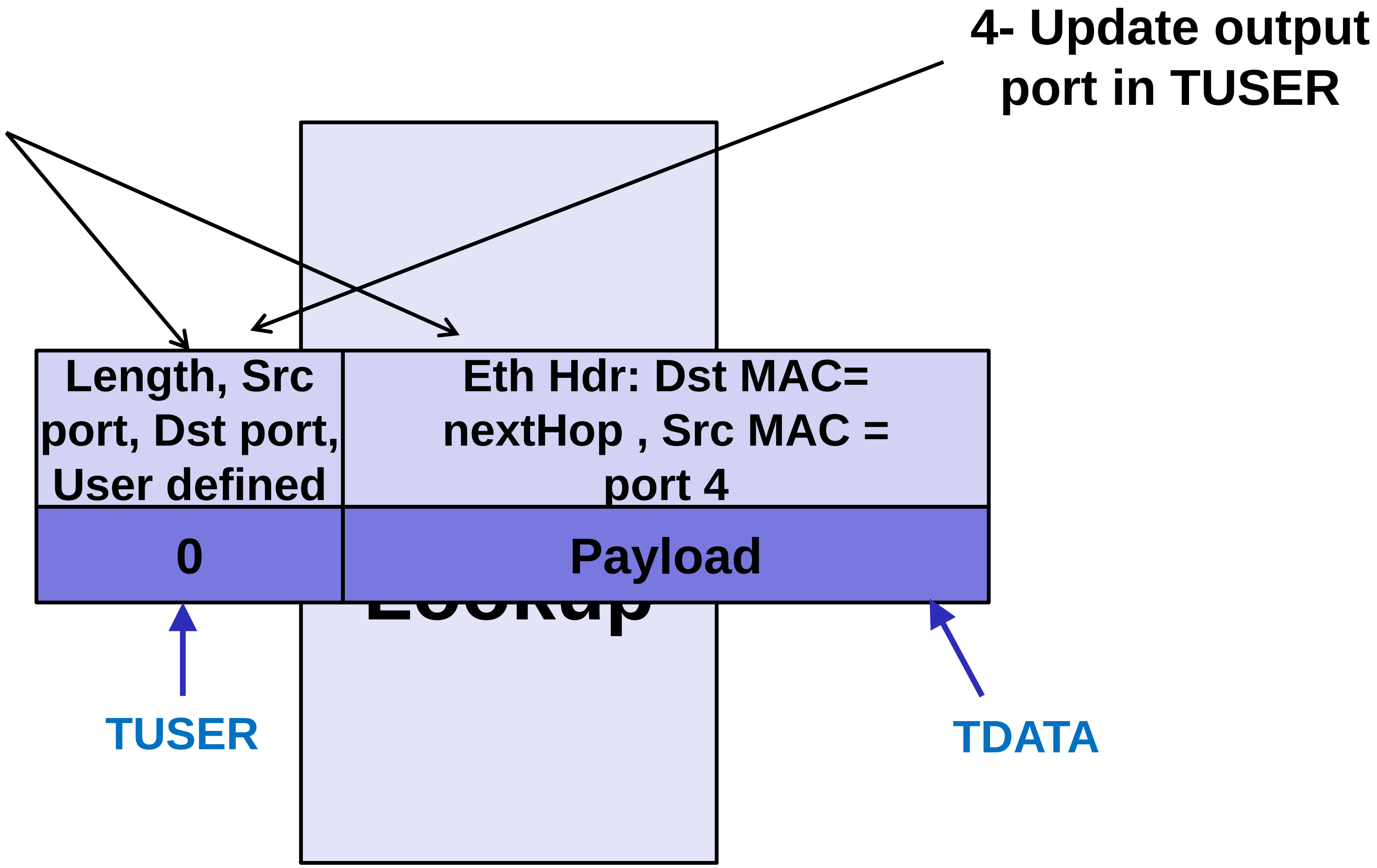
Input Arbiter



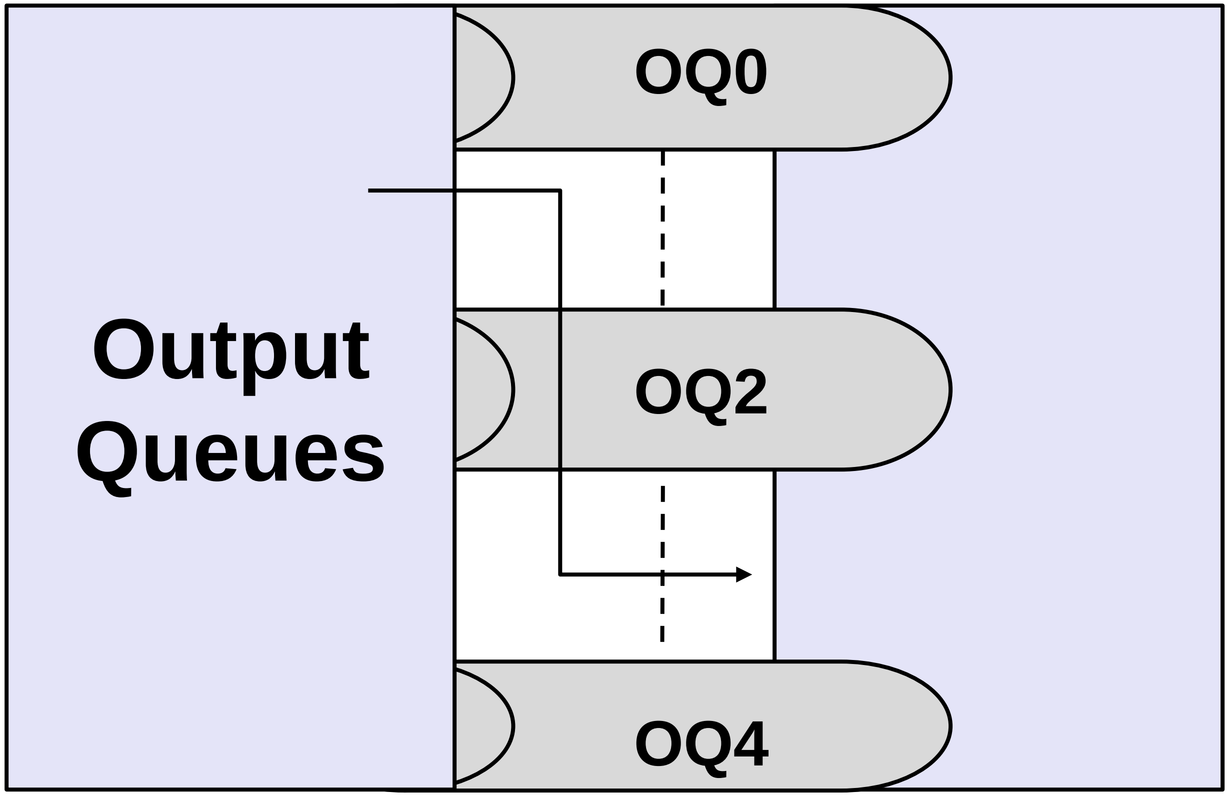
Output Port Lookup



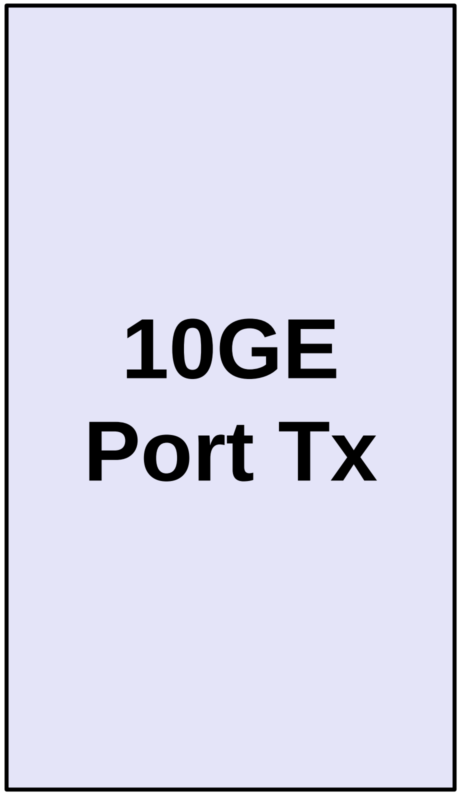
Output Port Lookup


1- Parse header: Src MAC, Dst MAC, Src port
2 - Lookup next hop MAC& output port
3- Learn Src MAC & Src port
4- Update output port in TUSER
Length, Src port, Dst port, User defined
Eth Hdr: Dst MAC= nextHop , Src MAC = port 4
0
Payload
Lookup
TUSER
TDATA

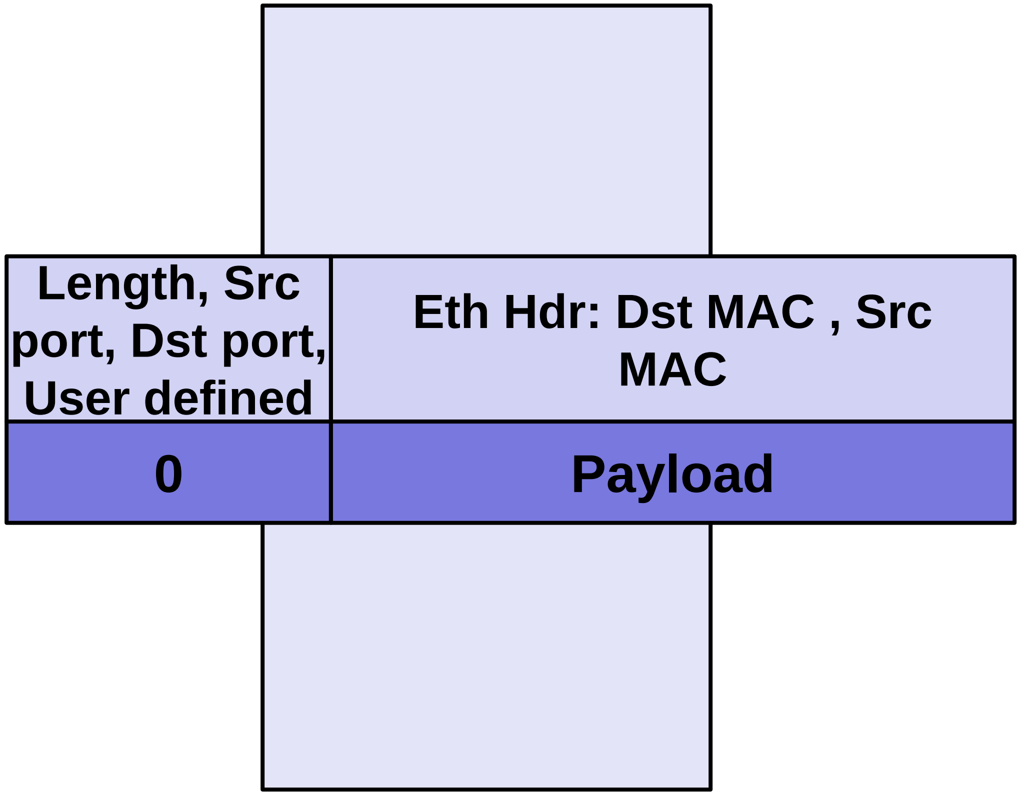
Output Queues



10GE Port Tx



MAC Tx Queue


Length, Src port, Dst port, User defined
Eth Hdr: Dst MAC , Src MAC
0
Payload

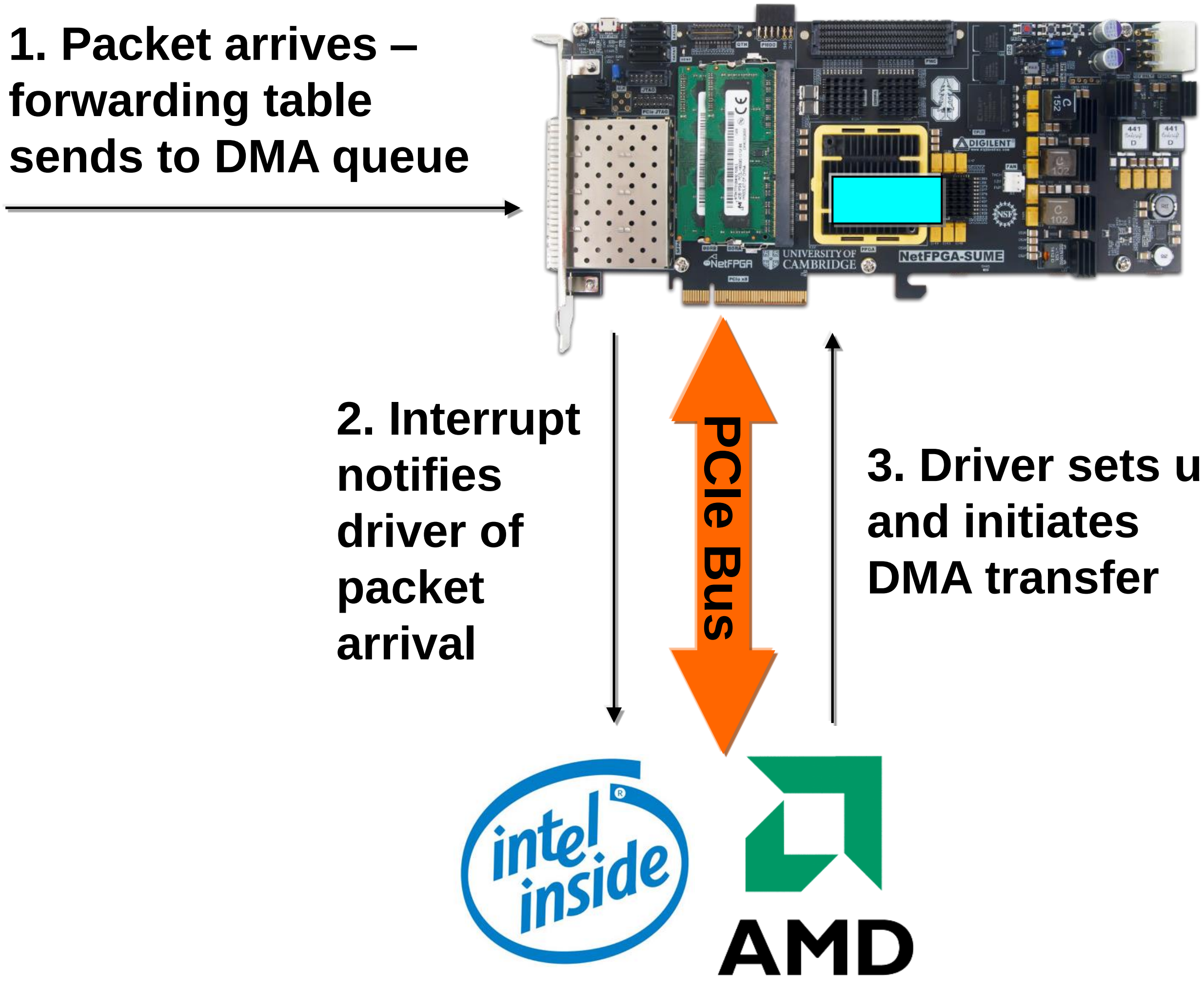
NetFPGA-Host Interaction

- **Linux driver interfaces with hardware**
  - Packet interface via standard Linux network stack
  - Register reads/writes via ioctl system call with wrapper functions:
    - rwaxi(int address, unsigned *data);

      eg:
      rwaxi(**0x7d4000000**, &val);

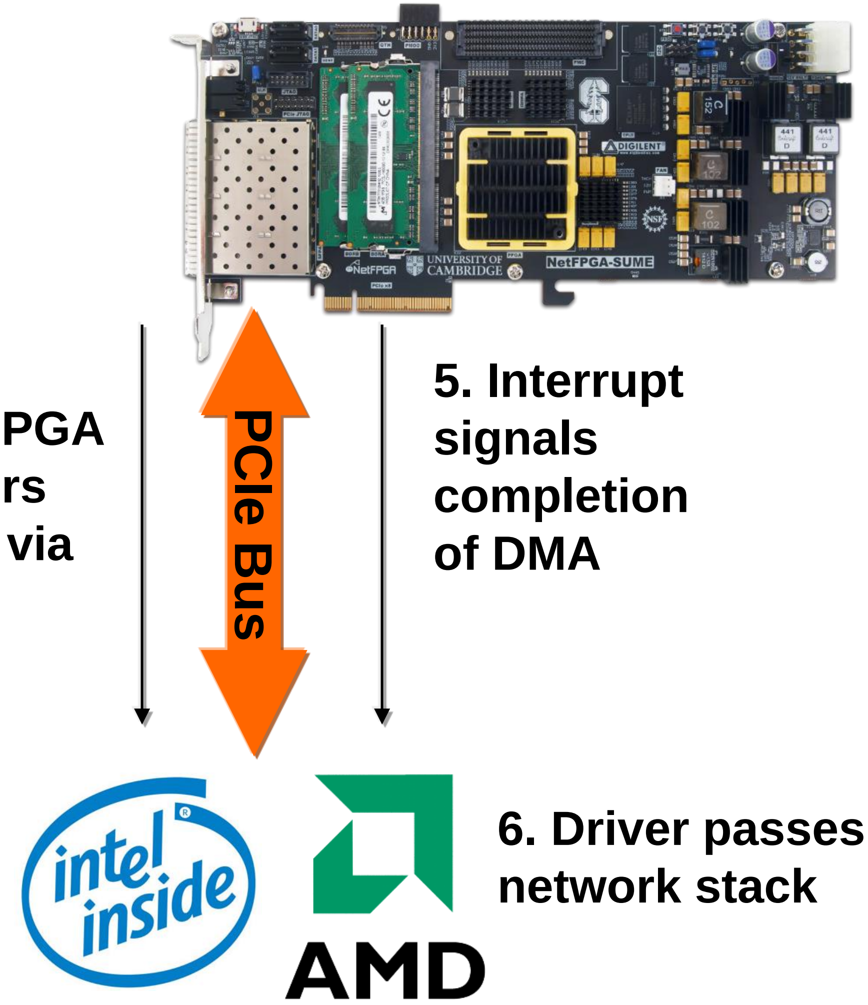
NetFPGA-Host Interaction

NetFPGA to host packet transfer


1. Packet arrives – forwarding table sends to DMA queue
2. Interrupt notifies driver of packet arrival
PCIe Bus
3. Driver sets up and initiates DMA transfer
NetFPGA-SUME
UNIVERSITY OF CAMBRIDGE
DIGILENT
intel inside
AMD

NetFPGA-Host Interaction

NetFPGA to host packet transfer (cont.)



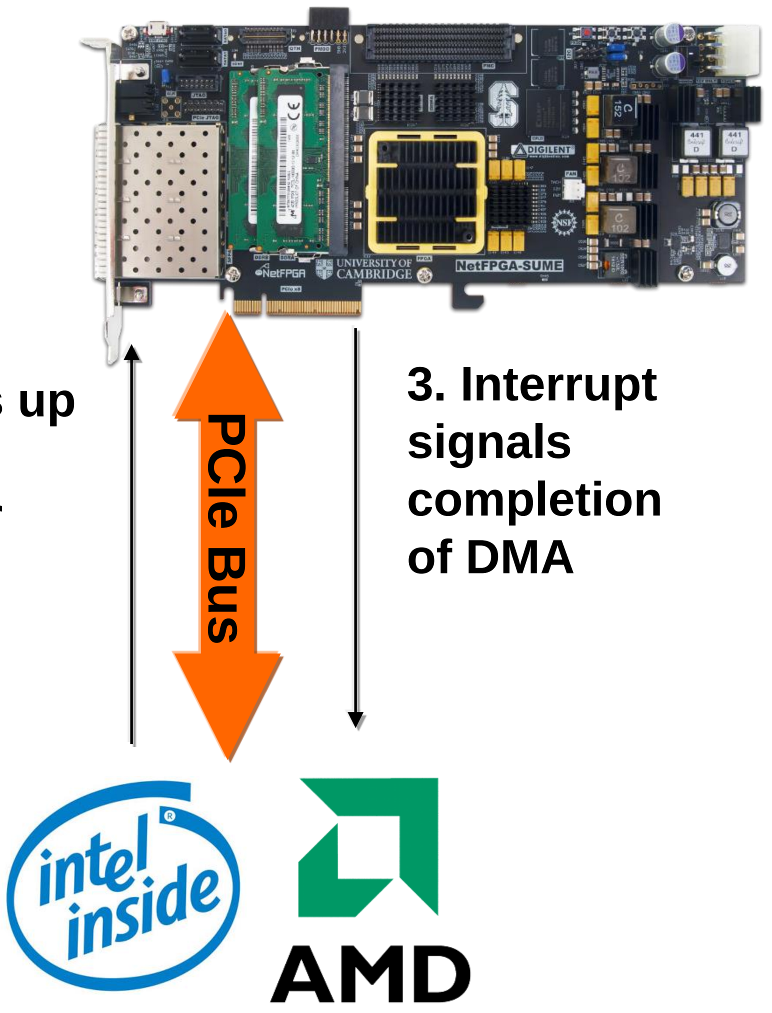
4. NetFPGA transfers packet via DMA

5. Interrupt signals completion of DMA

6. Driver passes packet to network stack

NetFPGA-Host Interaction

Host to NetFPGA packet transfers



2. Driver sets up and initiates DMA transfer

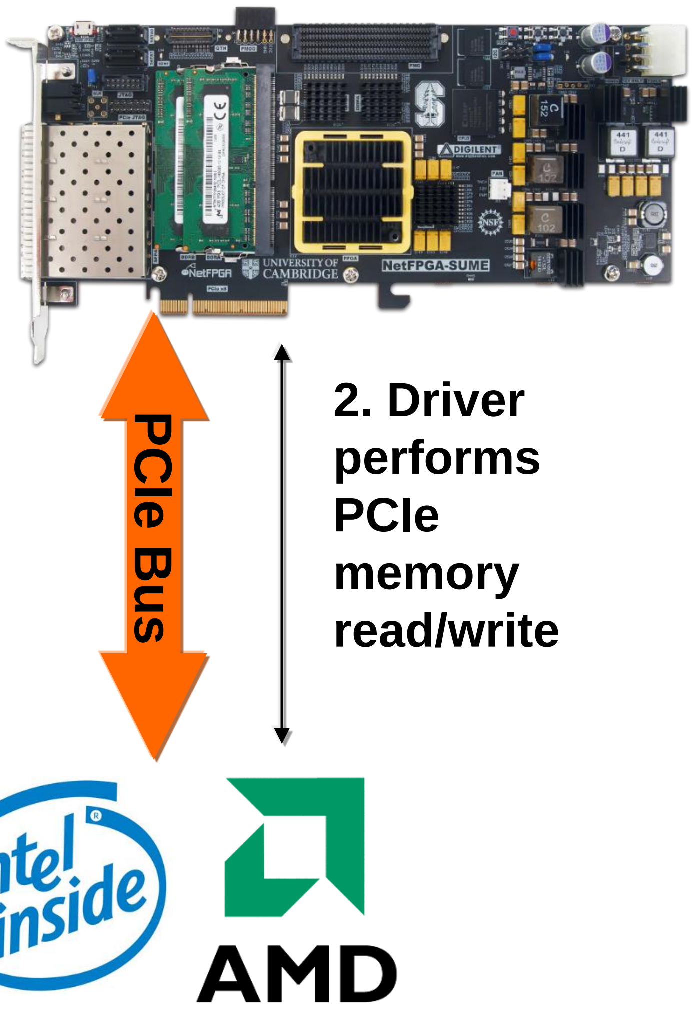
3. Interrupt signals completion of DMA

1. Software sends packet via network sockets

   Packet delivered to driver

NetFPGA-Host Interaction

Register access



2. Driver performs PCIe memory read/write

1. Software makes ioctl call on network socket

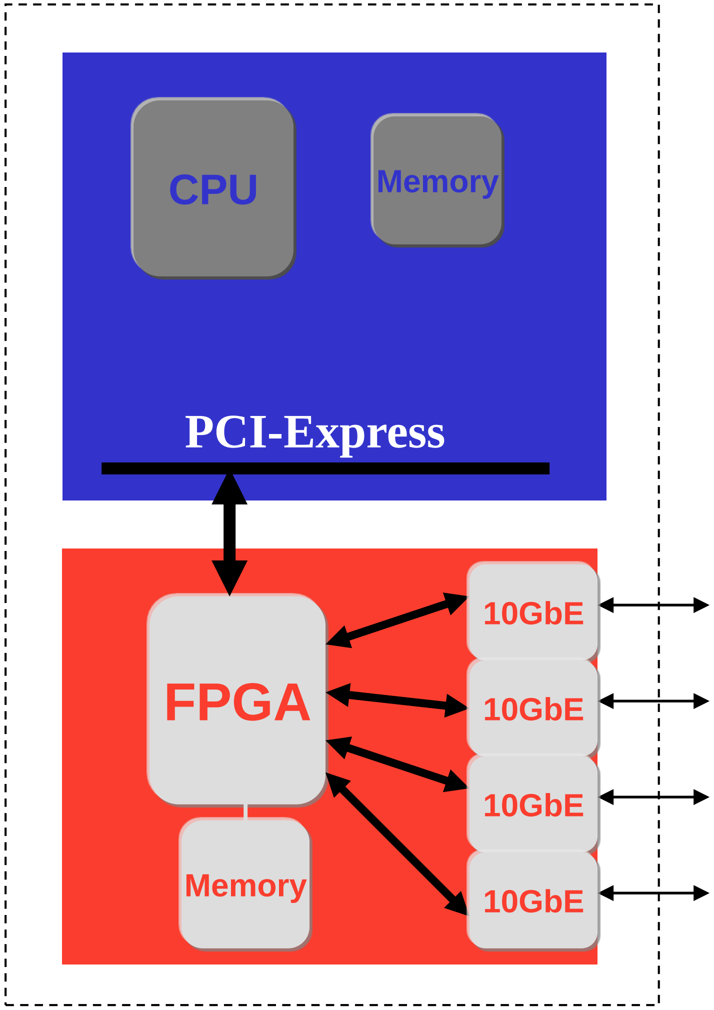
ioctl passed to driver

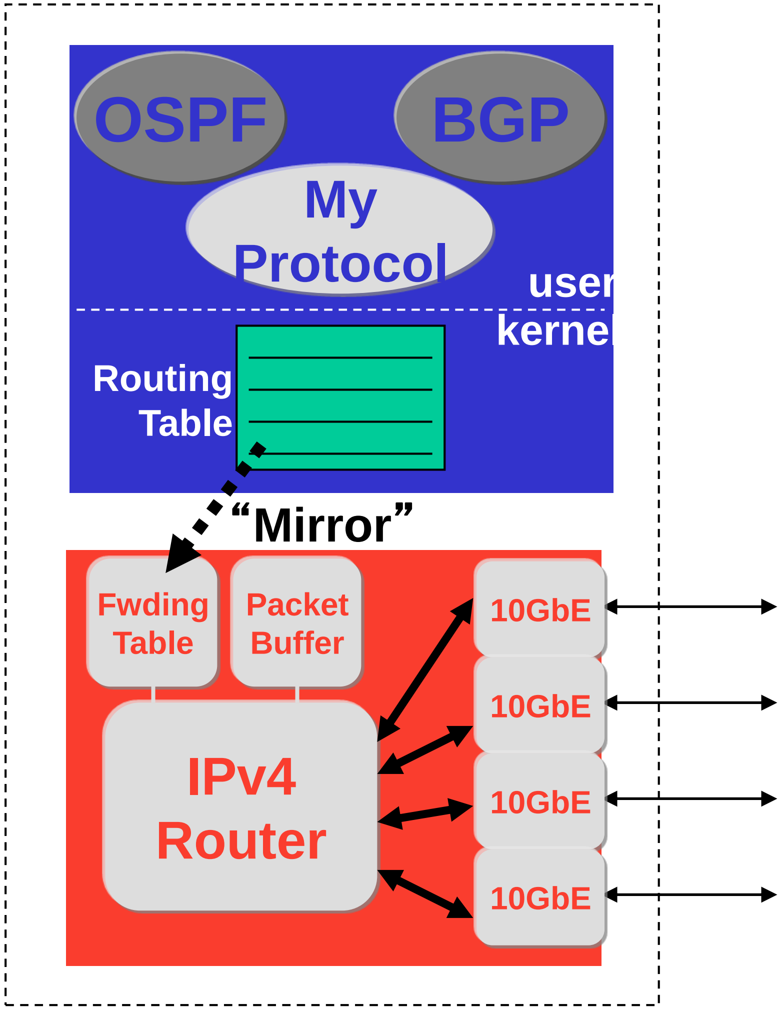
Section V: Examples of using NetFPGA

Usage #1

Running the Reference Router

User-space development, 4x10GE line-rate forwarding



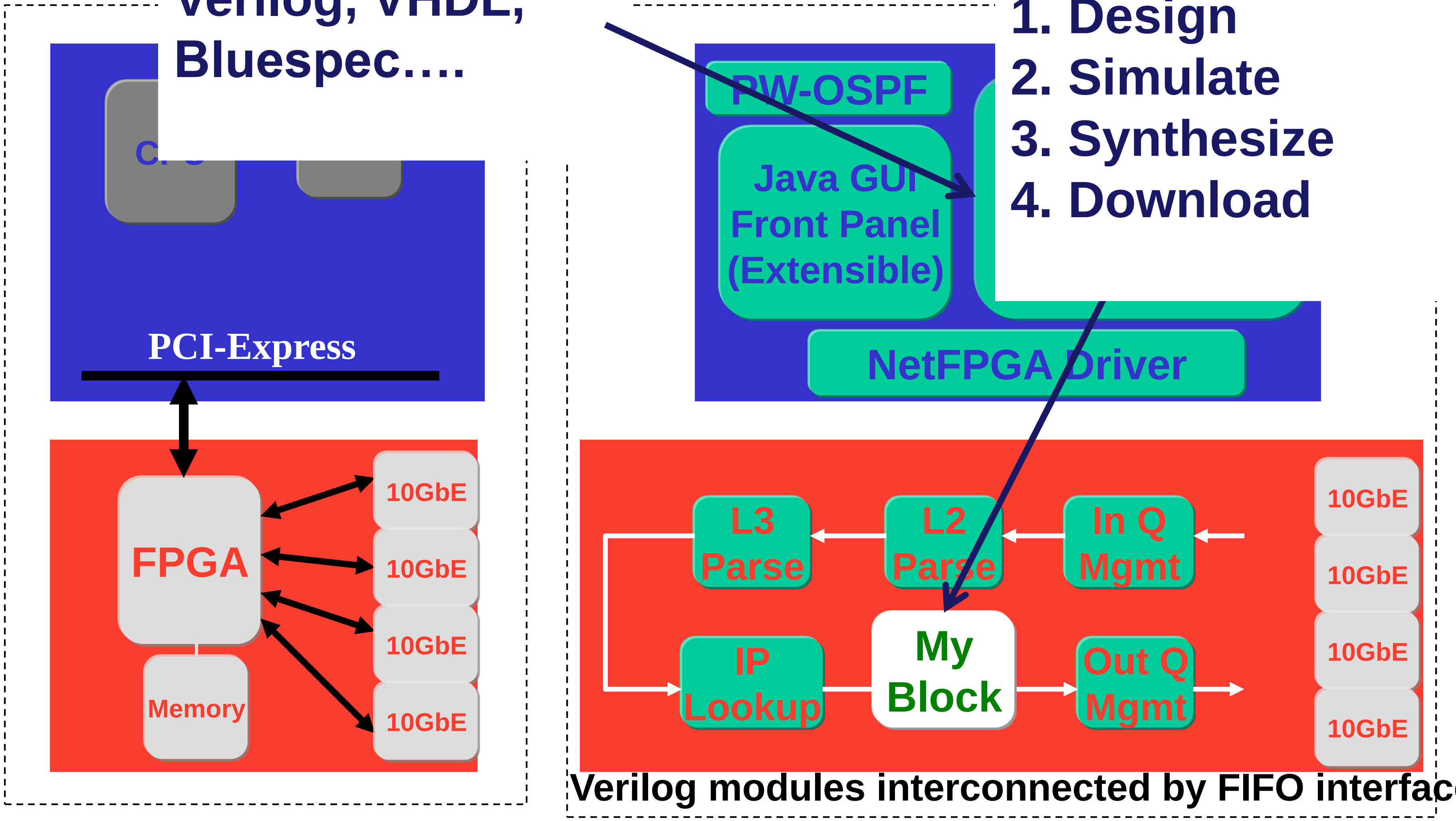


Usage #2

Enhancing Modular Reference Designs

Verilog, System Verilog, VHDL, Bluespec….

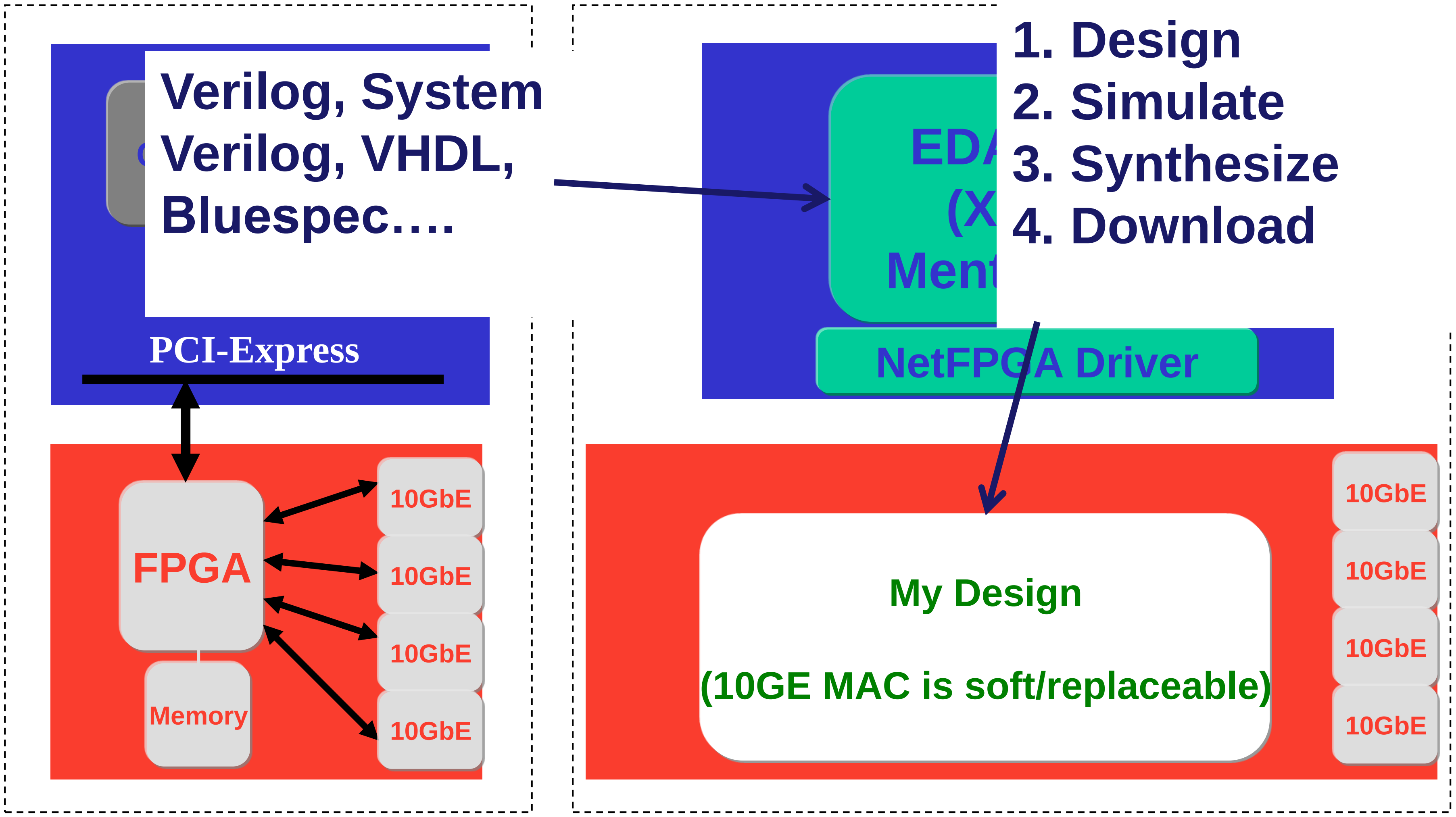
1. Design
2. Simulate
3. Synthesize
4. Download



Verilog modules interconnected by FIFO interface

Usage #3

Creating new systems

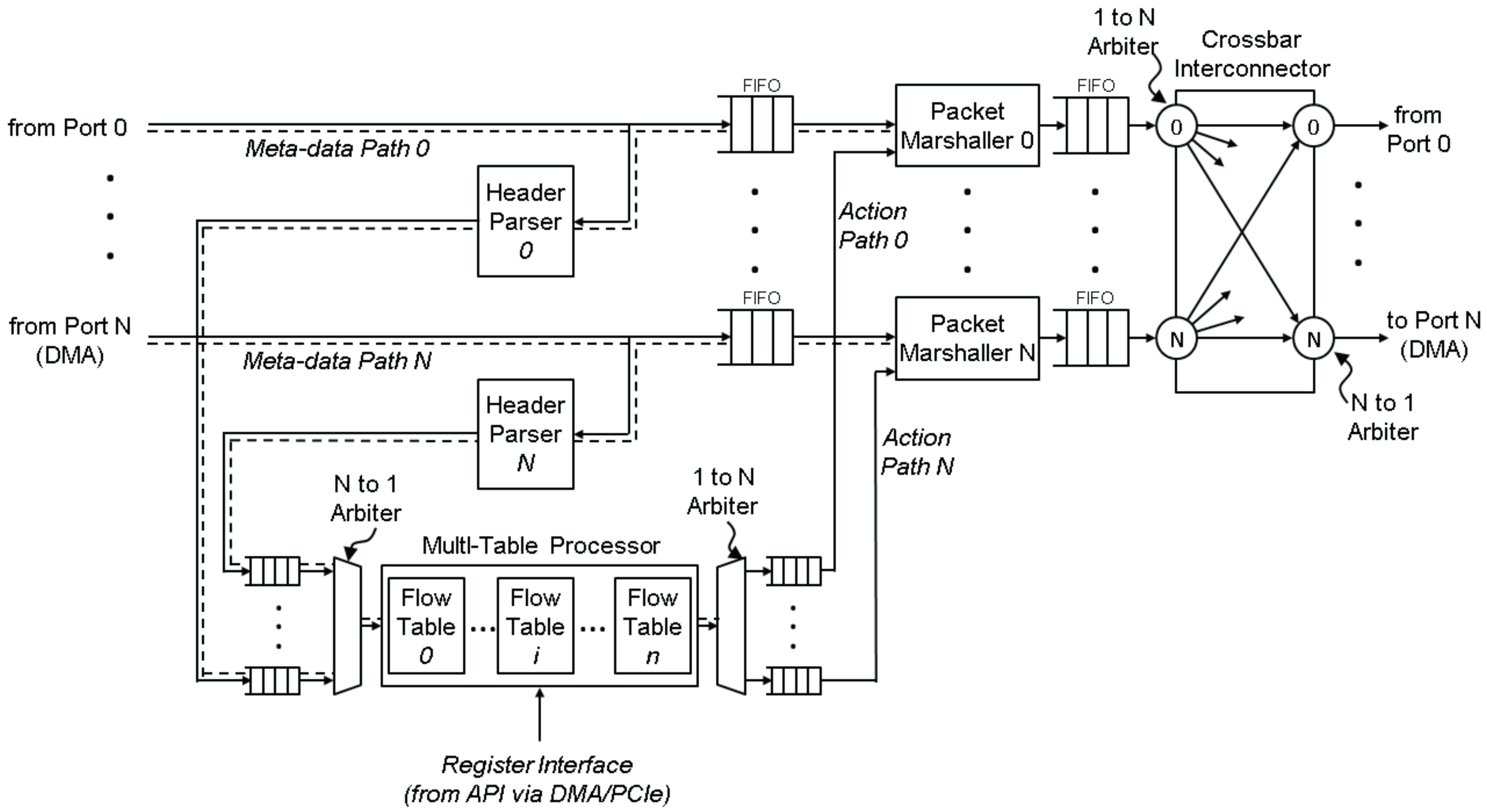


Verilog, System Verilog, VHDL, Bluespec….

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

BlueSwitch: A Multi Table OpenFlow Switch

Your design can look completely different!



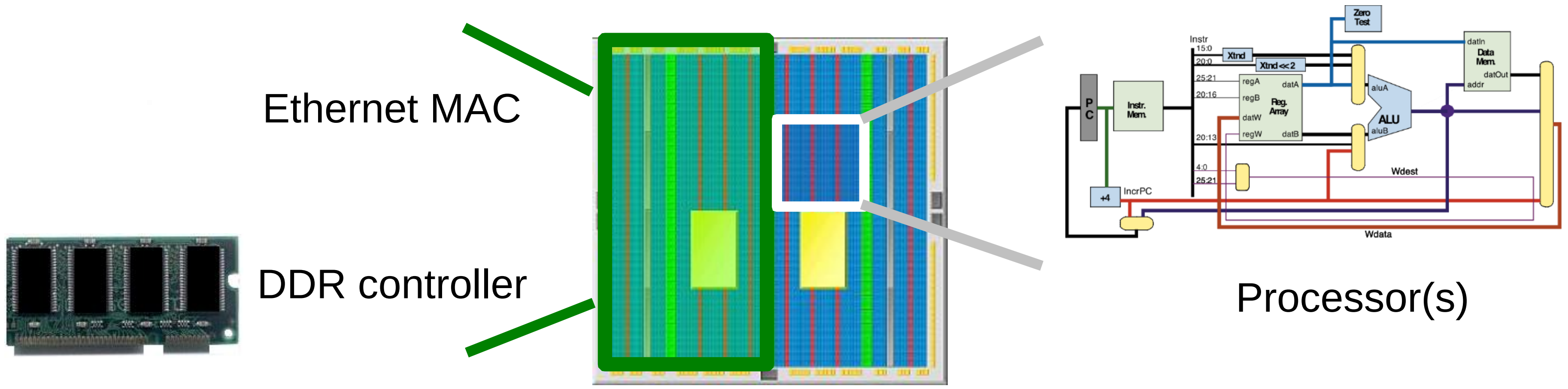
Contributed Projects

Platform	Project	Contributor
1G	OpenFlow switch	Stanford University
	Packet generator	Stanford University
	NetFlow Probe	Brno University
	NetThreads	University of Toronto
	zFilter (Sp)router	Ericsson
	Traffic Monitor	University of Catania
	DFA	UMass Lowell
10G	Bluespec switch	UCAM/SRI International
	Traffic Monitor	University of Pisa
	NF1G legacy on NF10G	Uni Pisa & Uni Cambridge
	High perf. DMA core	University of Cambridge
	BERI/CHERI	UCAM/SRI International
	OSNT	UCAM/Stanford/GTech/CNRS

OpenFlow

- **The most prominent NetFPGA success**
- **Has reignited the Software Defined Networking movement**
- **NetFPGA enabled OpenFlow**
  - A widely available open-source development platform
  - Capable of line-rate and
- **was, until its commercial uptake, the reference platform for OpenFlow.**

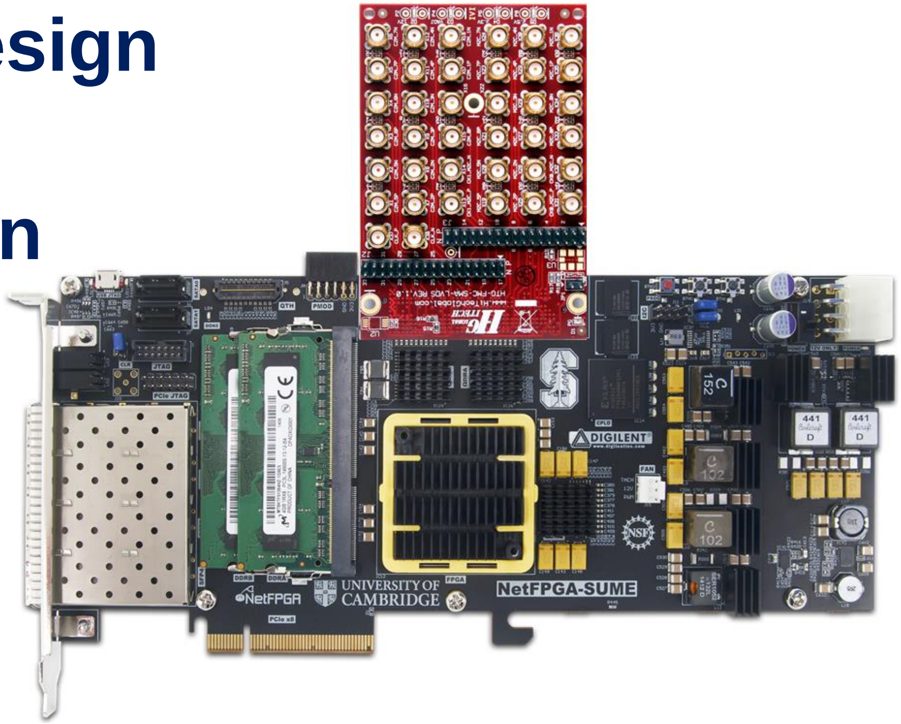
Soft Processors in FPGAs



- 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
- CHERI – 64bit MIPS soft processor, BSD OS

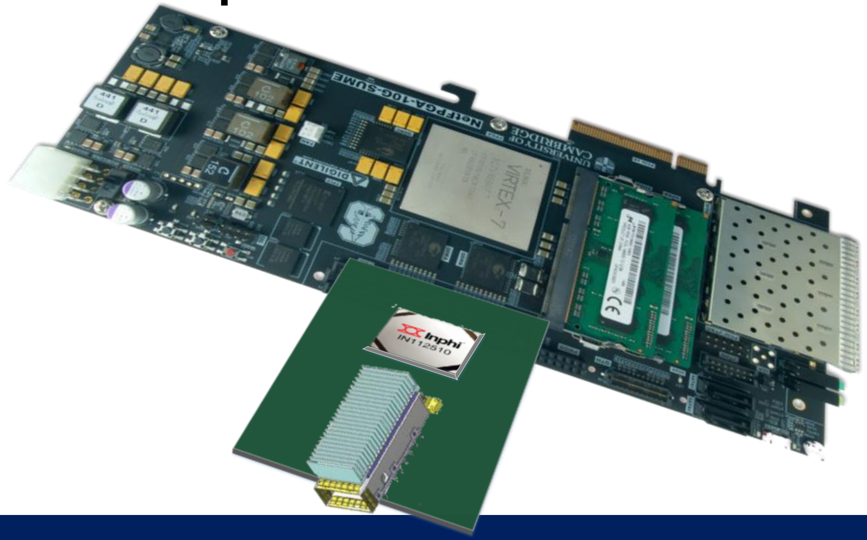
Physical Interface Design

- A deployment and interoperability test platform
  - Permits replacement of physical-layer
  - Provides high-speed expansion interfaces with standardised interfaces
- Allows researchers to design custom daughterboards
- Permits closer integration



Power Efficient MAC

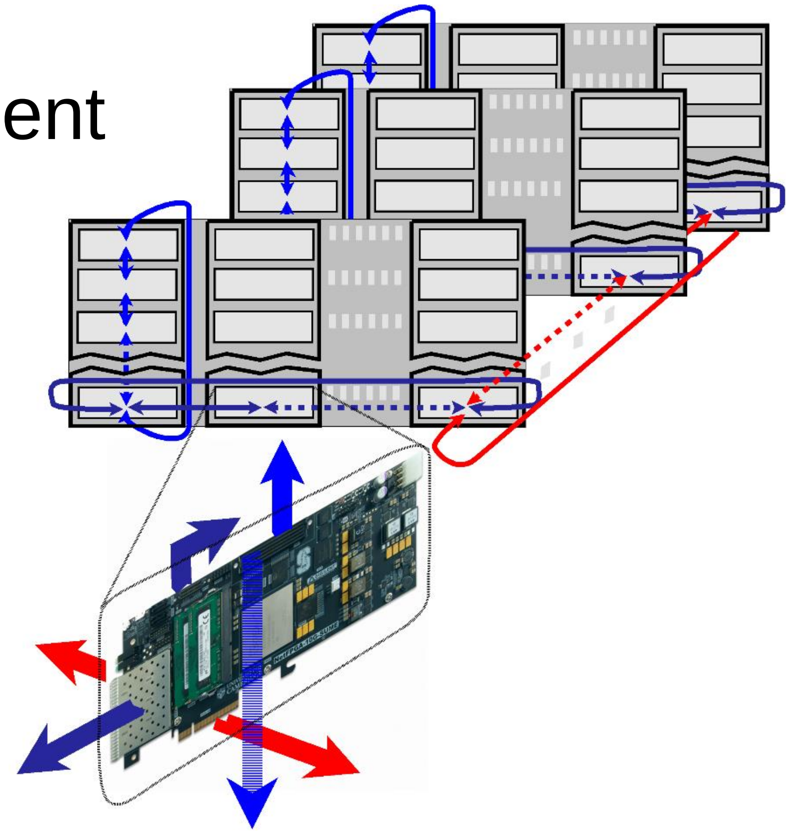
- **A Platform for 100Gb/s power-saving MAC design (e.g. lights-out MAC)**
- **Porting MAC design to SUME permits:**
  - Power measurements
  - Testing protocol's response
  - Reconsideration of power-saving mechanisms
  - Evaluating suitability for complex architectures and systems



Interconnect

- **Novel Architectures with line-rate performance**
  - A lot of networking equipment
  - Extremely complex

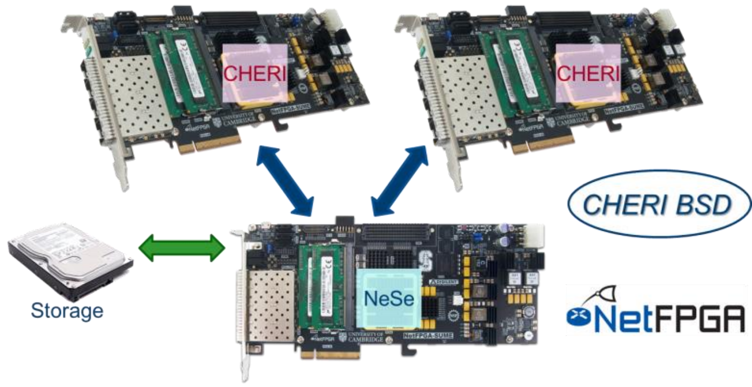
- **NetFPGA SUME allows prototyping a *complete* solution**

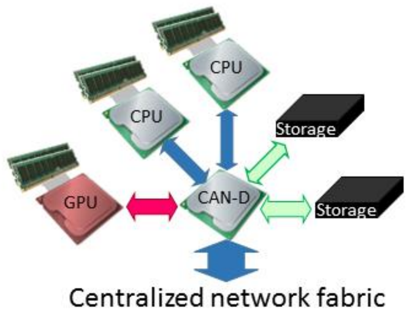


N x N xN Hyper-cube

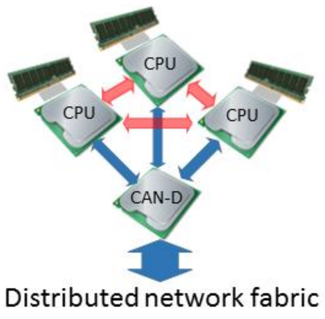
Tiny Terabit Datacenter*

- **Bridging performance gap between networking and computing**
  - Bandwidth
  - Latency
  - Performance guarantees
  - Supporting 10K-100K of VMs/Processes

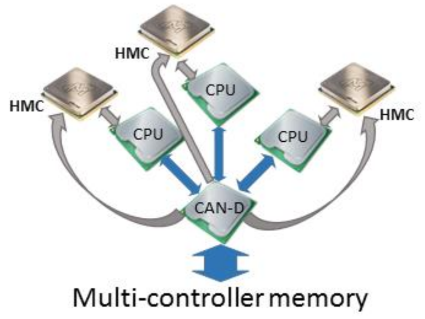




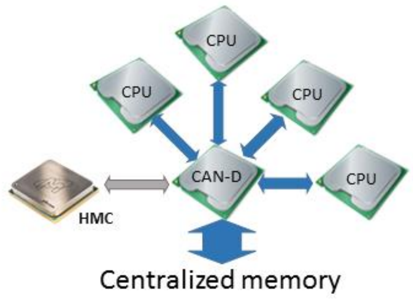
Centralized network fabric



Distributed network fabric



Multi-controller memory



Centralized memory

***No relation to Tiny Tera switch**

Section VI: Infrastructure

Infrastructure

- **Tree structure**

- **NetFPGA package contents**
  - Reusable Verilog modules
  - Verification infrastructure
  - Build infrastructure
  - Utilities
  - Software libraries

NetFPGA package contents

- **Projects:**
  - HW: router, switch, NIC
  - SW: router kit, SCONE
- **Reusable Verilog modules**
- **Verification infrastructure:**
  - simulate designs (from AXI interface)
  - run tests against hardware
  - test data generation libraries (eg. packets)
- **Build infrastructure**
- **Utilities:**
  - register I/O
- **Software libraries**

Tree Structure (1)

NetFPGA-SUME

- **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)

https://github.com/NetFPGA/NetFPGA-SUME-alpha

Tree Structure (2)

lib

- ***hw*** **(hardware logic as IP cores)**
  - **std** **(reference cores)**
  - **contrib** **(contributed cores)**
- ***sw*** **(core specific software drivers/libraries)**
  - **std** **(reference libraries)**
  - **contrib** **(contributed libraries)**

Tree Structure (3)

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)

Category	IP Core(s)
I/O interfaces	Ethernet 10G Port PCI Express UART GPIO
Output queues	BRAM based
Output port lookup	NIC CAM based Learning switch
Memory interfaces	SRAM DRAM FLASH
Miscellaneous	FIFOs AXIS width converter

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**
  - collates 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

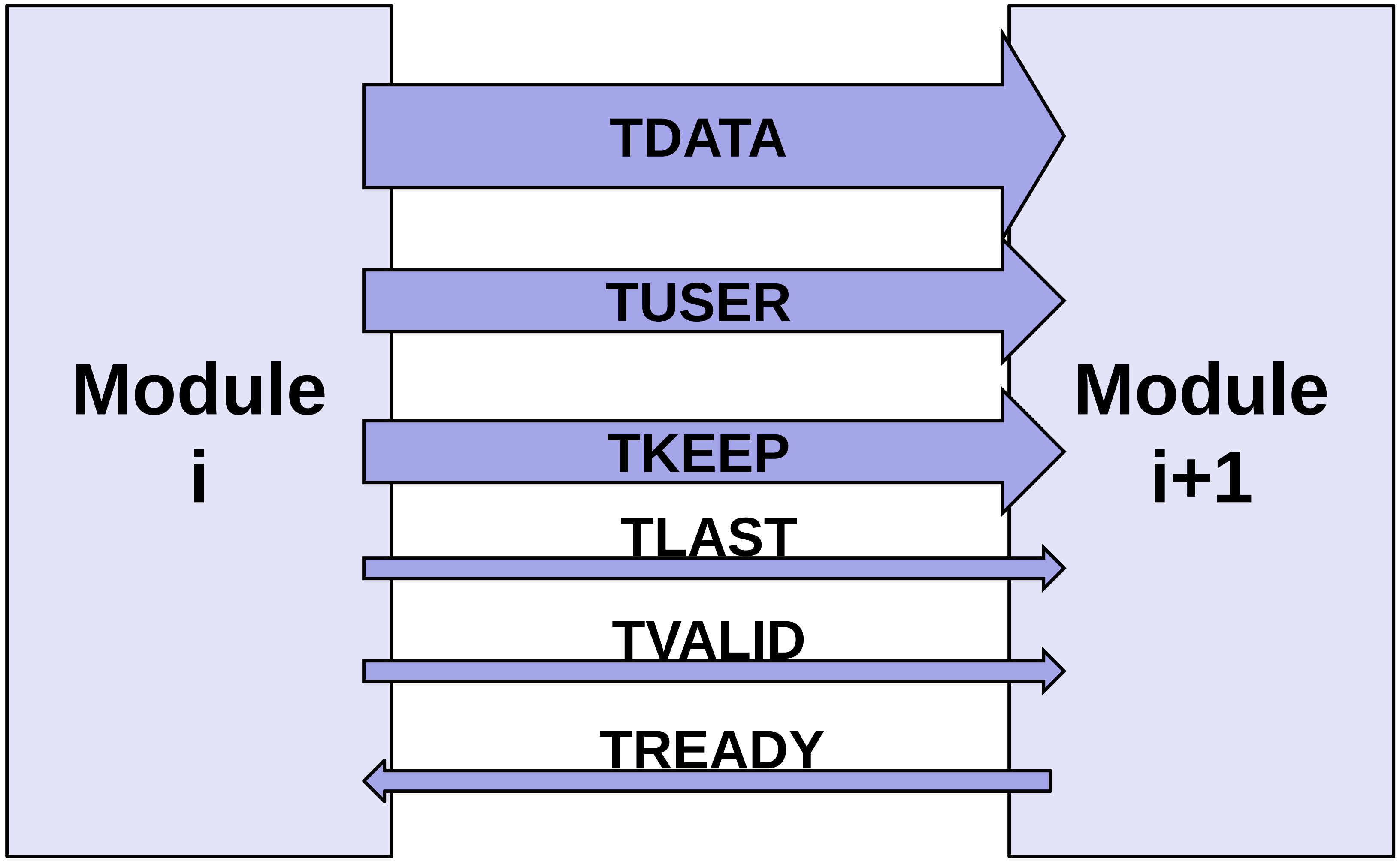
- **Has a large number of uses:**
  - Create projects
  - Set properties
  - Generate cores
  - Define connectivity
  - Etc.

Projects

- **Projects:**
  - Each design is represented by a project
  - Location: NetFPGA-SUME-alpha/projects/<proj_name>
  - Create a new project:
    - Normally:
      - copy an existing project as the starting point
    - Today:
      - pre-created project
  - Consists of:
    - Verilog source
    - Simulation tests
    - Hardware tests
    - Optional software

Inter-Module Communication

– Using AXI-4 Stream (*Packets are moved as Stream)*



AXI4-Stream

AXI4-Stream	Description
TDATA	Data Stream
TKEEP	Marks NULL bytes (i.e. byte enable)
TVALID	Valid Indication
TREADY	Flow control indication
TLAST	End of packet/burst indication
TUSER	Out of band metadata

Packet Format

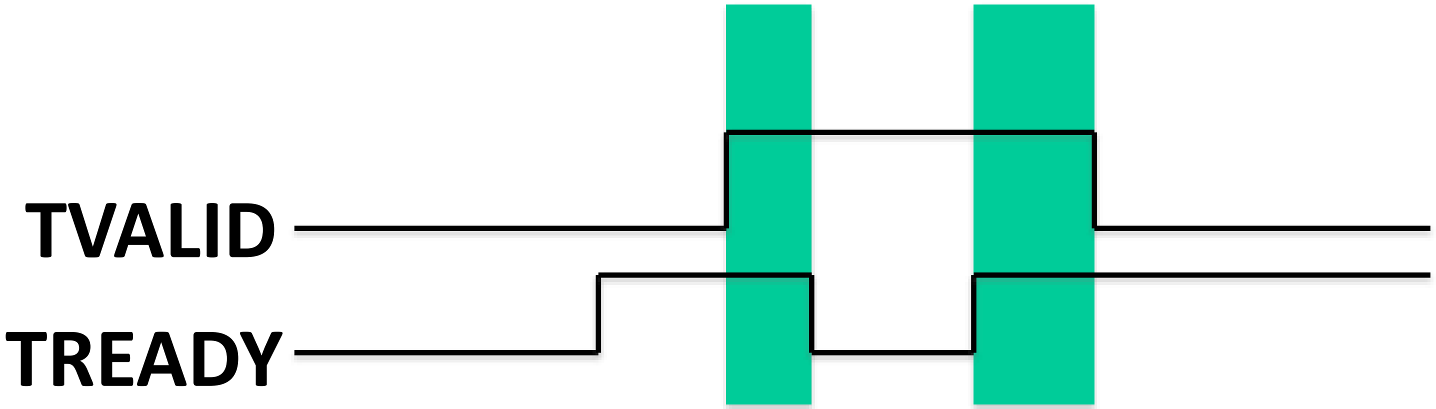
TLAST	TUSER	TKEEP	TDATA
0	X	0xFF…F	Eth Hdr
0	X	0xFF…F	IP Hdr
0	X	0xFF…F	…
1	X	0x0…1F	Last word

TUSER

Position	Content
[15:0]	length of the packet in bytes
[23:16]	source port: one-hot encoded
[31:24]	destination port: one-hot encoded
[127:32]	6 user defined slots, 16bit each

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]

Section VII: Example Project: Controlled Packet Loss

Exercise

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: Drop 1 in N Packets….

Exercise

Drop 1 in N Packets

Objectives

- Add counter and FSM to the code
- Simulate and test design

Execution

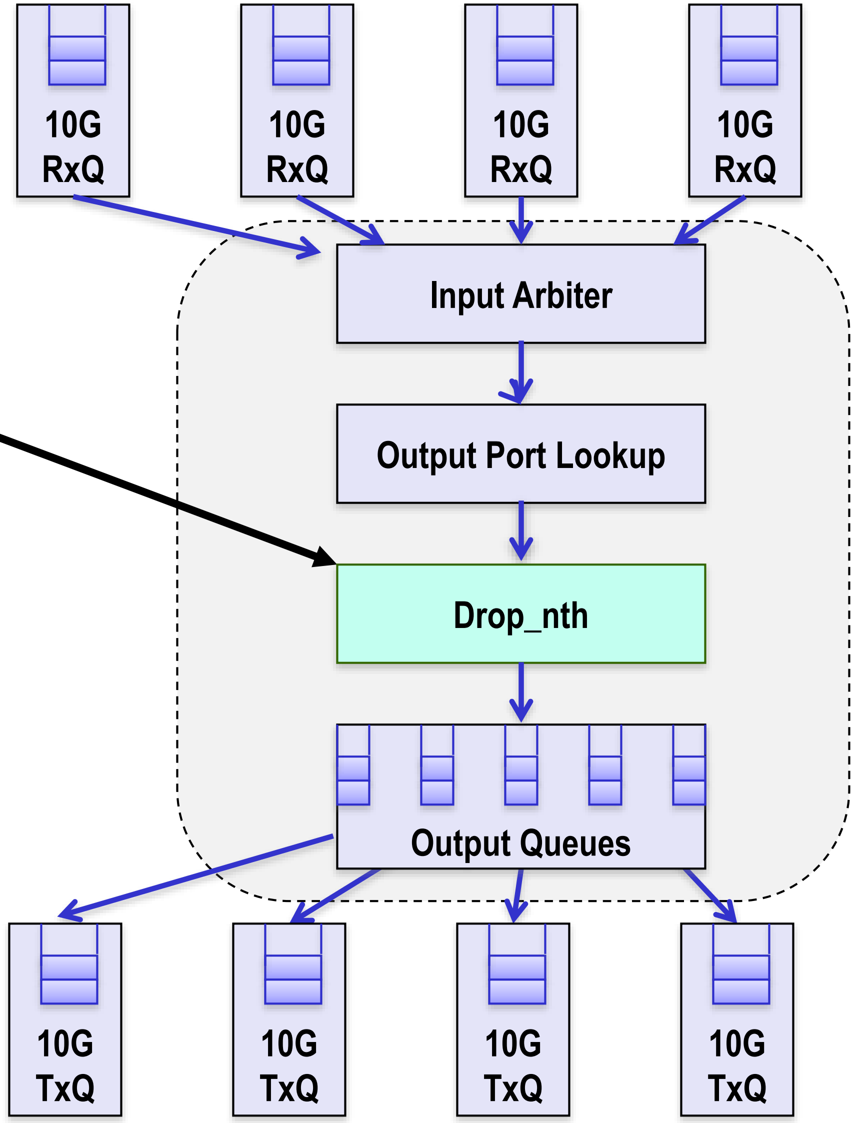
- Open drop_nth_packet.v
- Insert counter code
- Simulate
- Synthesize
- Test the new system

Exercise

Our Enhanced Switch Pipeline

One module added

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



Exercise

Step 1 - Open the Source

We will modify the Verilog source code to add drop_nth module

We will simply comment and uncomment existing code

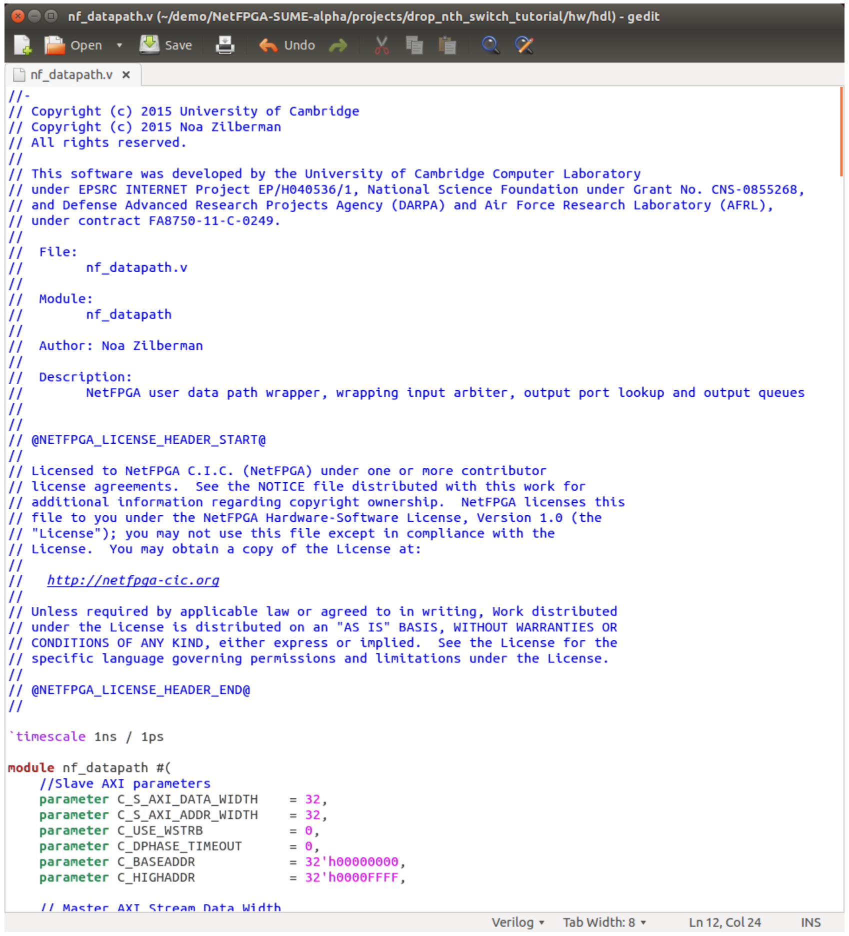
Open terminal

>> cd NetFPGA-SUME-alpha

>> source tools/settings.sh

>> cd $NF_DESIGN_DIR

>> vim hw/hdl/nf_datapath.v



nf_datapath.v (~/demo/NetFPGA-SUME-alpha/projects/drop_nth_switch_tutorial/hw/hdl) - gedit

```
//-
// Copyright (c) 2015 University of Cambridge
// Copyright (c) 2015 Noa Zilberman
// All rights reserved.
//
// This software was developed by the University of Cambridge Computer Laboratory
// under EPSRC INTERNET Project EP/H040536/1, National Science Foundation under Grant No. CNS-0855268,
// and Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL),
// under contract FA8750-11-C-0249.
//
//  File:
//        nf_datapath.v
//
//  Module:
//        nf_datapath
//
//  Author: Noa Zilberman
//
//  Description:
//        NetFPGA user data path wrapper, wrapping input arbiter, output port lookup and output queues
//
//
// @NETFPGA_LICENSE_HEADER_START@
//
// Licensed to NetFPGA C.I.C. (NetFPGA) under one or more contributor
// license agreements.  See the NOTICE file distributed with this work for
// additional information regarding copyright ownership.  NetFPGA licenses this
// file to you under the NetFPGA Hardware-Software License, Version 1.0 (the
// "License"); you may not use this file except in compliance with the
// License.  You may obtain a copy of the License at:
//
//   http://netfpga-cic.org
//
// Unless required by applicable law or agreed to in writing, Work distributed
// under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
// CONDITIONS OF ANY KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations under the License.
//
// @NETFPGA_LICENSE_HEADER_END@
//

`timescale 1ns / 1ps

module nf_datapath #(
    //Slave AXI parameters
    parameter C_S_AXI_DATA_WIDTH    = 32,
    parameter C_S_AXI_ADDR_WIDTH    = 32,
    parameter C_USE_WSTRB           = 0,
    parameter C_DPHASE_TIMEOUT      = 0,
    parameter C_BASEADDR            = 32'h00000000,
    parameter C_HIGHADDR            = 32'h0000FFFF,

    // Master AXI Stream Data Width
```

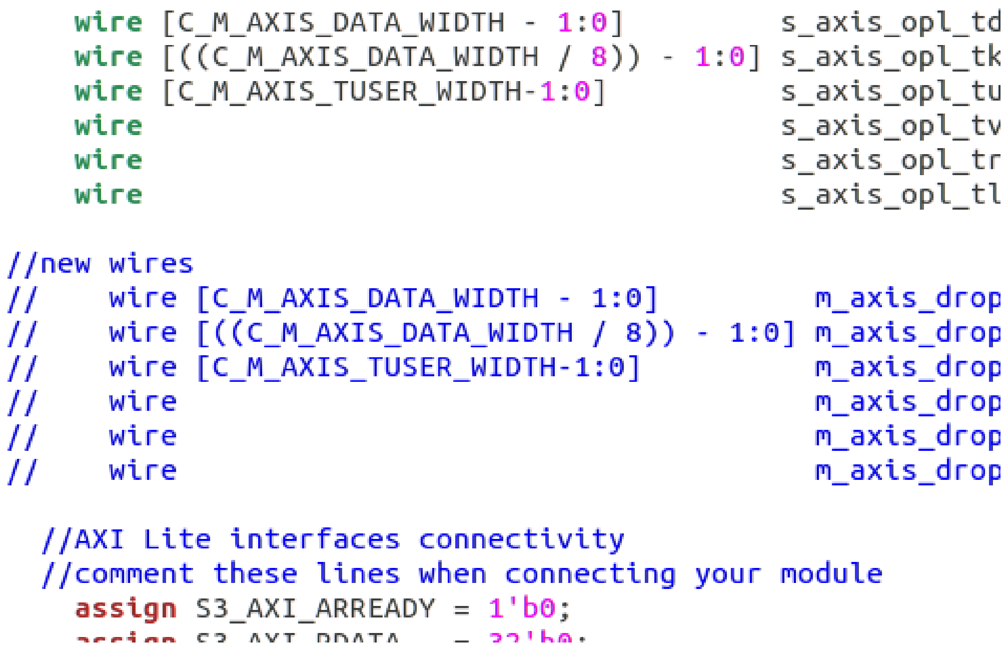
Verilog ▾   Tab Width: 8 ▾   Ln 12, Col 24   INS

Step 2a - Add Wires

Now we need to add wires to connect the new module

Search for "new wires"

Uncomment the wires



```
    wire [C_M_AXIS_DATA_WIDTH - 1:0]         s_axis_opl_td
    wire [((C_M_AXIS_DATA_WIDTH / 8)) - 1:0] s_axis_opl_tk
    wire [C_M_AXIS_TUSER_WIDTH-1:0]          s_axis_opl_tu
    wire                                     s_axis_opl_tv
    wire                                     s_axis_opl_tr
    wire                                     s_axis_opl_tl

//new wires
//    wire [C_M_AXIS_DATA_WIDTH - 1:0]         m_axis_drop
//    wire [((C_M_AXIS_DATA_WIDTH / 8)) - 1:0] m_axis_drop
//    wire [C_M_AXIS_TUSER_WIDTH-1:0]          m_axis_drop
//    wire                                     m_axis_drop
//    wire                                     m_axis_drop
//    wire                                     m_axis_drop

  //AXI Lite interfaces connectivity
  //comment these lines when connecting your module
    assign S3_AXI_ARREADY = 1'b0;
```

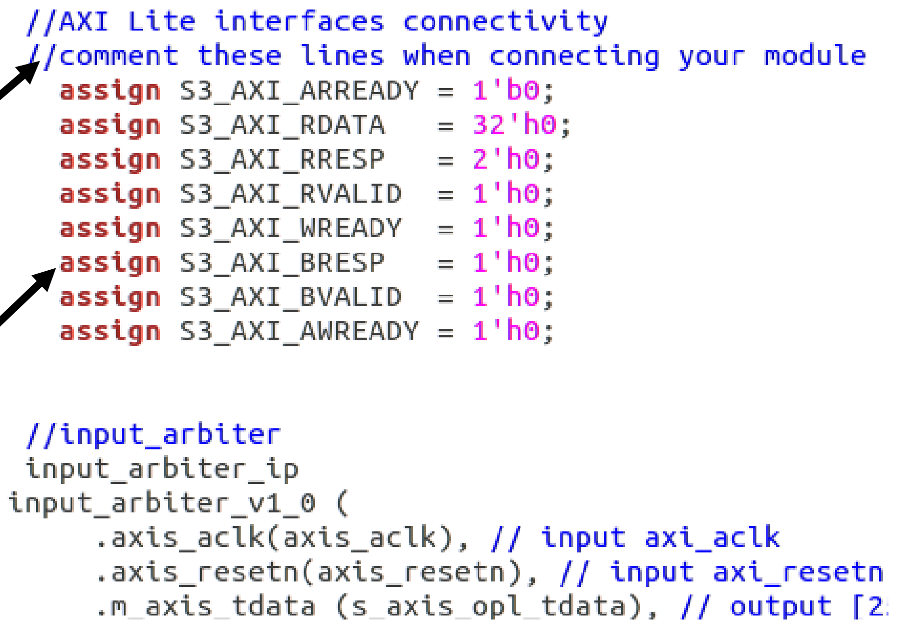
Exercise

Step 2b – Comment assignments

Now we need to comment assignments, used when the module is not connected

Search for “comment these lines”

Comment the assignments



```
//AXI Lite interfaces connectivity
//comment these lines when connecting your module
  assign S3_AXI_ARREADY = 1'b0;
  assign S3_AXI_RDATA   = 32'h0;
  assign S3_AXI_RRESP   = 2'h0;
  assign S3_AXI_RVALID  = 1'h0;
  assign S3_AXI_WREADY  = 1'h0;
  assign S3_AXI_BRESP   = 1'h0;
  assign S3_AXI_BVALID  = 1'h0;
  assign S3_AXI_AWREADY = 1'h0;


 //input_arbiter
 input_arbiter_ip
input_arbiter_v1_0 (
     .axis_aclk(axis_aclk), // input axi_aclk
     .axis_resetn(axis_resetn), // input axi_resetn
     .m_axis_tdata (s_axis_opl_tdata), // output [2
```

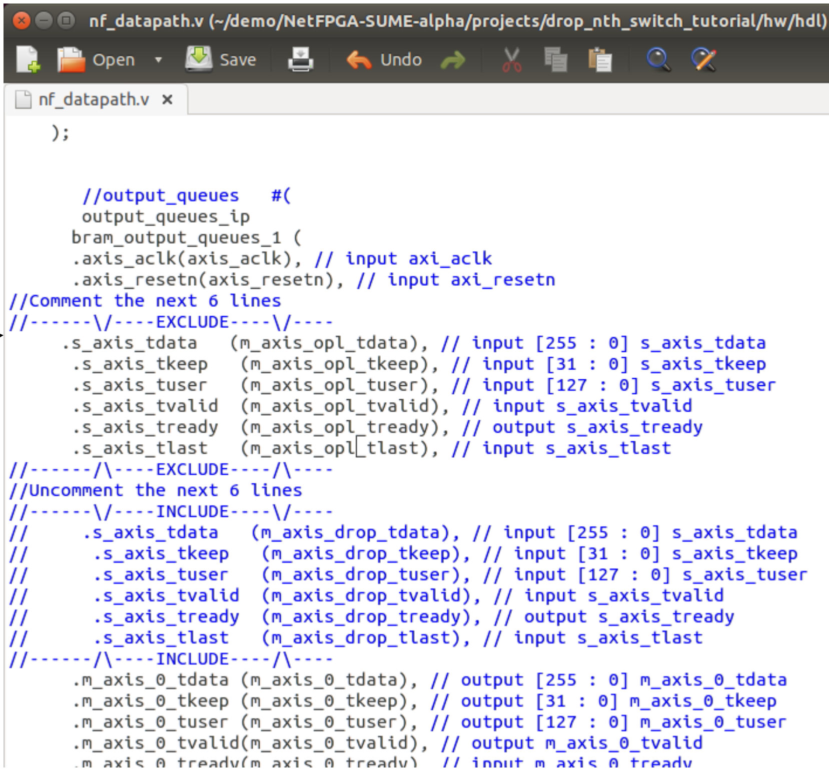
Exercise

Step 3a - Connect Drop_nth

Search for output_queues_ip

Comment the six lines above

Uncomment the block below to connect the outputs


nf_datapath.v (~/demo/NetFPGA-SUME-alpha/projects/drop_nth_switch_tutorial/hw/hdl)
Open
Save
Undo
nf_datapath.v

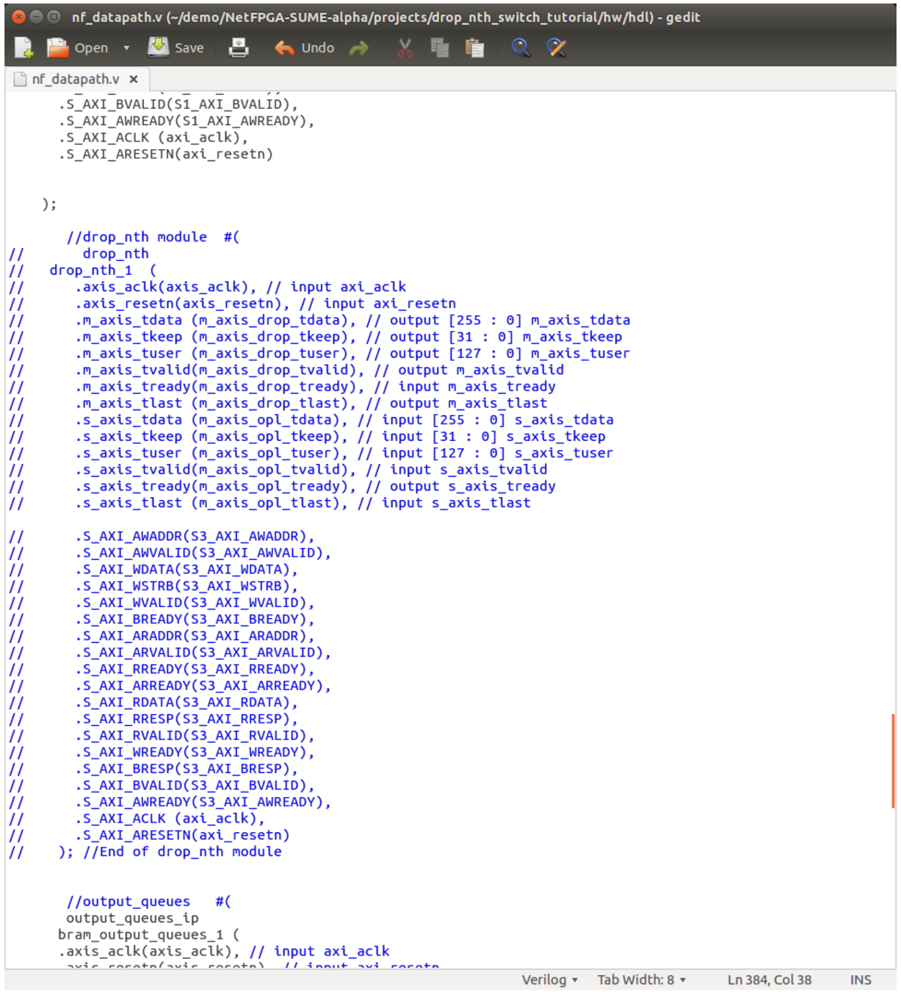
```
    );


      //output_queues   #(
      output_queues_ip
     bram_output_queues_1 (
     .axis_aclk(axis_aclk), // input axi_aclk
     .axis_resetn(axis_resetn), // input axi_resetn
//Comment the next 6 lines
//------\/----EXCLUDE----\/----
    .s_axis_tdata   (m_axis_opl_tdata), // input [255 : 0] s_axis_tdata
     .s_axis_tkeep   (m_axis_opl_tkeep), // input [31 : 0] s_axis_tkeep
     .s_axis_tuser   (m_axis_opl_tuser), // input [127 : 0] s_axis_tuser
     .s_axis_tvalid  (m_axis_opl_tvalid), // input s_axis_tvalid
     .s_axis_tready  (m_axis_opl_tready), // output s_axis_tready
     .s_axis_tlast   (m_axis_opl_tlast), // input s_axis_tlast
//------/\----EXCLUDE----/\----
//Uncomment the next 6 lines
//------\/----INCLUDE----\/----
//     .s_axis_tdata   (m_axis_drop_tdata), // input [255 : 0] s_axis_tdata
//      .s_axis_tkeep   (m_axis_drop_tkeep), // input [31 : 0] s_axis_tkeep
//      .s_axis_tuser   (m_axis_drop_tuser), // input [127 : 0] s_axis_tuser
//      .s_axis_tvalid  (m_axis_drop_tvalid), // input s_axis_tvalid
//      .s_axis_tready  (m_axis_drop_tready), // output s_axis_tready
//      .s_axis_tlast   (m_axis_drop_tlast), // input s_axis_tlast
//------/\----INCLUDE----/\----
     .m_axis_0_tdata (m_axis_0_tdata), // output [255 : 0] m_axis_0_tdata
     .m_axis_0_tkeep (m_axis_0_tkeep), // output [31 : 0] m_axis_0_tkeep
     .m_axis_0_tuser (m_axis_0_tuser), // output [127 : 0] m_axis_0_tuser
     .m_axis_0_tvalid(m_axis_0_tvalid), // output m_axis_0_tvalid
     .m_axis_0_tready(m_axis_0_tready), // input m_axis_0_tready
```

Exercise

Step 4 - Add the Drop_nth Module

Search for drop_nth

Uncomment the block


nf_datapath.v (~/demo/NetFPGA-SUME-alpha/projects/drop_nth_switch_tutorial/hw/hdl) - gedit

```
      .S_AXI_BVALID(S1_AXI_BVALID),
      .S_AXI_AWREADY(S1_AXI_AWREADY),
      .S_AXI_ACLK (axi_aclk),
      .S_AXI_ARESETN(axi_resetn)


    );

       //drop_nth module  #(
//       drop_nth
//   drop_nth_1  (
//      .axis_aclk(axis_aclk), // input axi_aclk
//      .axis_resetn(axis_resetn), // input axi_resetn
//      .m_axis_tdata (m_axis_drop_tdata), // output [255 : 0] m_axis_tdata
//      .m_axis_tkeep (m_axis_drop_tkeep), // output [31 : 0] m_axis_tkeep
//      .m_axis_tuser (m_axis_drop_tuser), // output [127 : 0] m_axis_tuser
//      .m_axis_tvalid(m_axis_drop_tvalid), // output m_axis_tvalid
//      .m_axis_tready(m_axis_drop_tready), // input m_axis_tready
//      .m_axis_tlast (m_axis_drop_tlast), // output m_axis_tlast
//      .s_axis_tdata (m_axis_opl_tdata), // input [255 : 0] s_axis_tdata
//      .s_axis_tkeep (m_axis_opl_tkeep), // input [31 : 0] s_axis_tkeep
//      .s_axis_tuser (m_axis_opl_tuser), // input [127 : 0] s_axis_tuser
//      .s_axis_tvalid(m_axis_opl_tvalid), // input s_axis_tvalid
//      .s_axis_tready(m_axis_opl_tready), // output s_axis_tready
//      .s_axis_tlast (m_axis_opl_tlast), // input s_axis_tlast

//      .S_AXI_AWADDR(S3_AXI_AWADDR),
//      .S_AXI_AWVALID(S3_AXI_AWVALID),
//      .S_AXI_WDATA(S3_AXI_WDATA),
//      .S_AXI_WSTRB(S3_AXI_WSTRB),
//      .S_AXI_WVALID(S3_AXI_WVALID),
//      .S_AXI_BREADY(S3_AXI_BREADY),
//      .S_AXI_ARADDR(S3_AXI_ARADDR),
//      .S_AXI_ARVALID(S3_AXI_ARVALID),
//      .S_AXI_RREADY(S3_AXI_RREADY),
//      .S_AXI_ARREADY(S3_AXI_ARREADY),
//      .S_AXI_RDATA(S3_AXI_RDATA),
//      .S_AXI_RRESP(S3_AXI_RRESP),
//      .S_AXI_RVALID(S3_AXI_RVALID),
//      .S_AXI_WREADY(S3_AXI_WREADY),
//      .S_AXI_BRESP(S3_AXI_BRESP),
//      .S_AXI_BVALID(S3_AXI_BVALID),
//      .S_AXI_AWREADY(S3_AXI_AWREADY),
//      .S_AXI_ACLK (axi_aclk),
//      .S_AXI_ARESETN(axi_resetn)
//    ); //End of drop_nth module


       //output_queues   #(
       output_queues_ip
      bram_output_queues_1 (
      .axis_aclk(axis_aclk), // input axi_aclk
```

Exercise

Step 5 - Open the Source

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

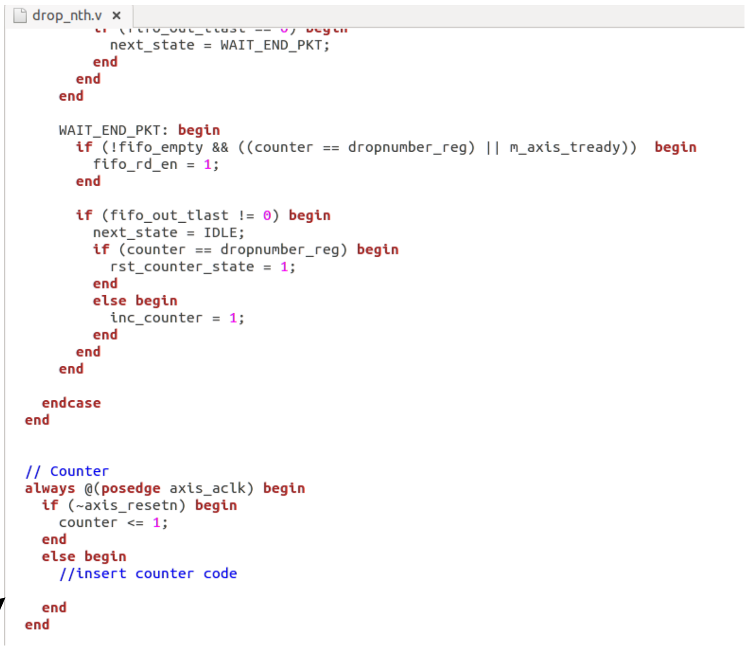
Return to terminal

```
>> cd NetFPGA-SUME-alpha
>>  cd $NF_DESIGN_DIR/local_ip/drop_nth/hw/hdl
>> vim drop_nth.v
```

Step 6 - Add Counter to Module

Add counter using the following signals:

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



drop_nth.v

```
          next_state = WAIT_END_PKT;
        end
      end
    end

    WAIT_END_PKT: begin
      if (!fifo_empty && ((counter == dropnumber_reg) || m_axis_tready))  begin
        fifo_rd_en = 1;
      end

      if (fifo_out_tlast != 0) begin
        next_state = IDLE;
        if (counter == dropnumber_reg) begin
          rst_counter_state = 1;
        end
        else begin
          inc_counter = 1;
        end
      end
    end

  endcase
end


// Counter
always @(posedge axis_aclk) begin
  if (~axis_resetn) begin
    counter <= 1;
  end
  else begin
    //insert counter code

  end
end
```

1. Search for insert counter code

2. Insert counter and save

Section VIII: Simulation and Debug

Testing: Simulation

- **Simulation allows testing without requiring lengthy synthesis process**
- **NetFPGA simulation environment allows:**
  - Send/receive packets
    - Physical ports and CPU
  - Read/write registers
  - Verify results
- **Simulations run in xSim**
- **We provides an unified infrastructure for both HW and simulation tests**

Testing: Simulation

- **We will simulate the "drop_nth_switch" design under the "simulation framework"**
- **We will**
  - create simple packets using scapy
  - transmit and reconcile packets sent over 10G Ethernet and PCIe interfaces
  - the code can be found in the "test" directory inside the drop_nth_switch project

Exercise

Step 7 - Simulation

Test your project:

Return to terminal

```
>> cd ~/NetFPGA-SUME-alpha/
```

Build IP cores:

```
>> make
```

Run simulation:

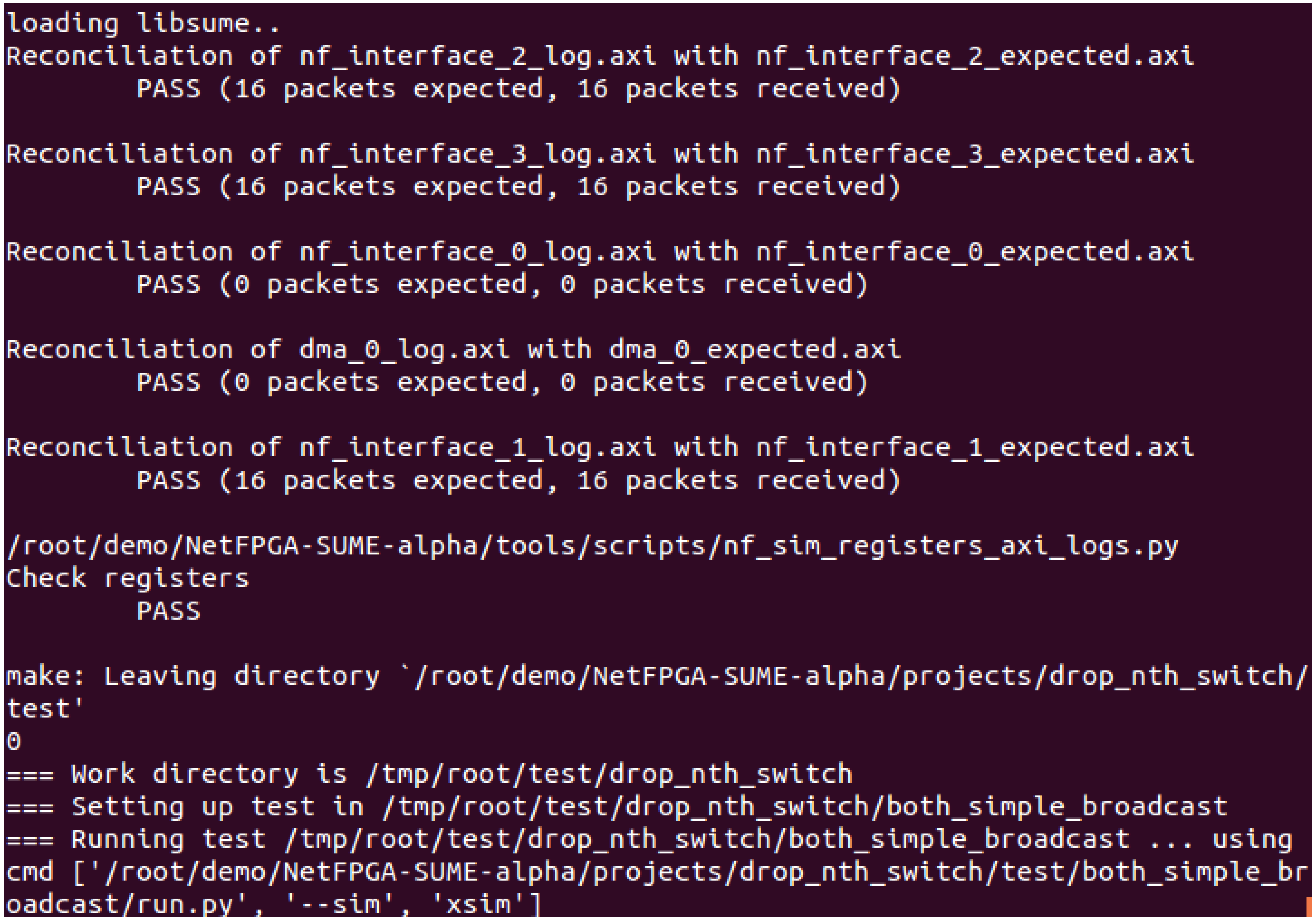
```
>> cd tools/scripts
>>  ./nf_test.py sim --major simple --minor broadcast
```

-OR- with GUI:

```
>>  ./nf_test.py sim --major simple --minor broadcast --gui
```

Exercise

The results are in…



```
loading libsume..
Reconciliation of nf_interface_2_log.axi with nf_interface_2_expected.axi
        PASS (16 packets expected, 16 packets received)

Reconciliation of nf_interface_3_log.axi with nf_interface_3_expected.axi
        PASS (16 packets expected, 16 packets received)

Reconciliation of nf_interface_0_log.axi with nf_interface_0_expected.axi
        PASS (0 packets expected, 0 packets received)

Reconciliation of dma_0_log.axi with dma_0_expected.axi
        PASS (0 packets expected, 0 packets received)

Reconciliation of nf_interface_1_log.axi with nf_interface_1_expected.axi
        PASS (16 packets expected, 16 packets received)

/root/demo/NetFPGA-SUME-alpha/tools/scripts/nf_sim_registers_axi_logs.py
Check registers
        PASS

make: Leaving directory `/root/demo/NetFPGA-SUME-alpha/projects/drop_nth_switch/
test'
0
=== Work directory is /tmp/root/test/drop_nth_switch
=== Setting up test in /tmp/root/test/drop_nth_switch/both_simple_broadcast
=== Running test /tmp/root/test/drop_nth_switch/both_simple_broadcast ... using
cmd ['/root/demo/NetFPGA-SUME-alpha/projects/drop_nth_switch/test/both_simple_br
oadcast/run.py', '--sim', 'xsim']
```

- **As expected, total of 20 packets are sent but only 16 are received on each interface**

Drop Nth Switch simulation

```
cd $NF_DESIGN_DIR/test/both_simple_broadcast
vim run.py
```

- The “**isHW**” statement enables the HW test
- Let’s focus on the “**else**” part of the statement
- **make_IP_pkt** fuction creates the IP packet that will be used as stimuli
- **pkt.tuser_sport** is used to set up the correct source port of the packet
- **encrypt_pkt** encrypts the packet
- **pkt.time** selects the time the packet is supposed to be sent
- **nftest_send_phy/dma** are used to send a packet to a given interface
- **nftest_expected_phy/dma** are used to expect a packet in a given interface
- **nftest_barrier** is used to block the simulation till the previous statement has been completed (e.g., send_pkts -> barrier -> send_more_pkts)

it is time for the first synthesis!!!

Synthesis

- **To synthesize your project:**

```
cd ~/$NF_DESIGN_DIR/
make clean; make
```

Section IX: What to do Next?

Well I'm not sure about you but here is a list I created:

- Build an accurate, fast, line-rate NetDummy/nistnet element
- A flexible home-grown monitoring card
- Evaluate new packet classifiers
  - (and application classifiers, and other neat network apps….)
- Prototype a full line-rate next-generation Ethernet-type
- Trying any of Jon Crowcrofts' ideas (Sourceless IP routing for example)
- Demonstrate the wonders of Metarouting in a different implementation (dedicated hardware)
- Provable hardware (using a C# implementation and kiwi with NetFPGA as target h/w)
- Hardware supporting Virtual Routers
- Check that some brave new idea actually works
  - e.g. Rate Control Protocol (RCP), Multipath TCP,
- toolkit for hardware hashing
- MOOSE implementation
- IP address anonymization
- SSL decoding "bump in the wire"
- Xen specialist nic
- computational co-processor
- Distributed computational co-processor
- IPv6 anything
- IPv6 – IPv4 gateway (6in4, 4in6, 6over4, 4over6, ….)
- Netflow v9 reference
- PSAMP reference
- IPFIX reference
- Different driver/buffer interfaces (e.g. PFRING)
- or "escalators" (from gridprobe) for faster network monitors
- Firewall reference
- GPS packet-timestamp things
- High-Speed Host Bus Adapter reference implementations
  - Infiniband
  - iSCSI
  - Myranet
  - Fiber Channel
- Smart Disk adapter (presuming a direct-disk interface)
- Software Defined Radio (SDR) directly on the FPGA (probably UWB only)
- Routing accelerator
  - Hardware route-reflector
  - Internet exchange route accelerator
- Hardware channel bonding reference implementation
- TCP sanitizer
- Other protocol sanitizer (applications… UDP DCCP, etc.)
- Full and complete Crypto NIC
- IPSec endpoint/ VPN appliance
- VLAN reference implementation
- metarouting implementation
- virtual <pick something>
- intelligent proxy
- application embargo-er
- Layer-4 gateway
- h/w gateway for VoIP/SIP/skype
- h/w gateway for video conference spaces
- security pattern/rules matching
- Anti-spoof traceback implementations (e.g. BBN stuff)
- IPtv multicast controller
- Intelligent IP-enabled device controller (e.g. IP cameras or IP powerm
- DES breaker
- platform for flexible NIC API evaluations
- snmp statistics reference implementation
- sflow (hp) reference implementation
- trajectory sampling (reference implementation)
- implementation of zeroconf/netconf configuration language for route
- h/w openflow and (simple) NOX controller in one…
- Network RAID (multicast TCP with redundancy)
- inline compression
- hardware accelerator for TOR
- load-balancer
- openflow with (netflow, ACL, ….)
- reference NAT device
- active measurement kit
- network discovery tool
- passive performance measurement
- active sender control (e.g. performance feedback fed to endpoints for
- Prototype platform for NON-Ethernet or near-Ethernet MACs
  - Optical LAN (no buffers)

- Build an accurate, fast, line-rate NetDummy/nistnet element
- A flexible home-grown monitoring card
- Evaluate new packet classifiers
  (and application classifiers, and other neat network apps….)
- Prototype a full line-rate next-generation Ethernet-type
- Trying any of Jon Crowcrofts' ideas (Sourceless IP routing for example)
- Demonstrate the wonders of Metarouting in a different implementation (dedicated hardware)
- Provable hardware (using a C# implementation and kiwi with NetFPGA as target h/w)
- Hardware supporting Virtual Routers
- Check that some brave new ideas actually works
  e.g. Rate Control Protocol (RCP), Multipath TCP

How might YOU use NetFPGA?

- Build an accurate, fast, line-rate NetDummy/nistnet element
- A flexible home-grown monitoring card
- Evaluate new packet classifiers
  - **(and application classifiers, and other neat network apps….)**
- Prototype a full line-rate next-generation Ethernet-type
- Trying any of Jon Crowcrofts' ideas (Sourceless IP routing for example)
- Demonstrate the wonders of Metarouting in a different implementation (dedicated hardware)
- Provable hardware (using a C# implementation and kiwi with NetFPGA as target h/w)
- Hardware supporting Virtual Routers
- Check that some brave new idea actually works

  e.g. Rate Control Protocol (RCP), Multipath TCP,
- toolkit for hardware hashing
- MOOSE implementation
- IP address anonymization
- SSL decoding "bump in the wire"
- Xen specialist nic
- computational co-processor
- Distributed computational co-processor
- IPv6 anything
- IPv6 – IPv4 gateway (6in4, 4in6, 6over4, 4over6, ….)
- Netflow v9 reference
- PSAMP reference
- IPFIX reference
- Different driver/buffer interfaces (e.g. PFRING)
- or "escalators" (from gridprobe) for faster network monitors
- Firewall reference
- GPS packet-timestamp things
- High-Speed Host Bus Adapter reference implementations
  - **Infiniband**
  - **iSCSI**
  - **Myranet**
  - **Fiber Channel**
- Smart Disk adapter (presuming a direct-disk interface)
- Software Defined Radio (SDR) directly on the FPGA (probably UWB only)
- Routing accelerator
  - **Hardware route-reflector**
  - **Internet exchange route accelerator**
- Hardware channel bonding reference implementation
- TCP sanitizer
- Other protocol sanitizer (applications… UDP DCCP, etc.)
- Full and complete Crypto NIC
- IPSec endpoint/ VPN appliance
- VLAN reference implementation
- metarouting implementation
- virtual <pick-something>
- intelligent proxy
- application embargo-er
- Layer-4 gateway
- h/w gateway for VoIP/SIP/skype
- h/w gateway for video conference spaces
- security pattern/rules matching
- Anti-spoof traceback implementations (e.g. BBN stuff)
- IPtv multicast controller
- Intelligent IP-enabled device controller (e.g. IP cameras or IP powerm
- DES breaker
- platform for flexible NIC API evaluations
- snmp statistics reference implementation
- sflow (hp) reference implementation
- trajectory sampling (reference implementation)
- implementation of zeroconf/netconf configuration language for route
- h/w openflow and (simple) NOX controller in one…
- Network RAID (multicast TCP with redundancy)
- inline compression
- hardware accelerator for TOR
- load-balancer
- openflow with (netflow, ACL, ….)
- reference NAT device
- active measurement kit
- network discovery tool
- passive performance measurement
- active sender control (e.g. performance feedback fed to endpoints for
- Prototype platform for NON-Ethernet or near-Ethernet MACs
  - **Optical LAN (no buffers)**

To get started with your project

1. **New Software ideas? get familiar with the host-systems of the current reference (C and java)**
2. **replace them at will; no egos will be hurt**

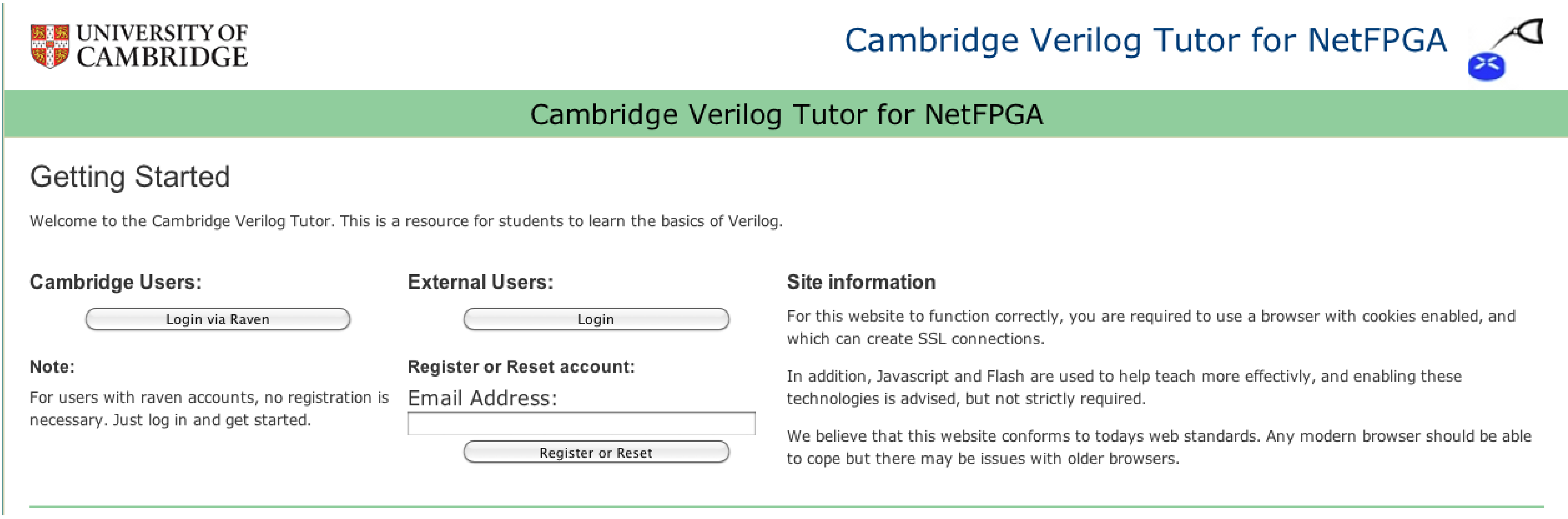
OR

1. **New Hardware ideas? get familiar with hardware description language**
2. **Prepare for your project**
   a) Become familiar with the NetFPGA yourself
   b) Go to a hands-on event

Good practice is familiarity with hardware and software…. (and it isn't that scary - honest)

Scared by Verilog? Try our Online Verilog tutor (with NetFPGA extensions)

www-netfpga.cl.cam.ac.uk



UNIVERSITY OF CAMBRIDGE

Cambridge Verilog Tutor for NetFPGA

Cambridge Verilog Tutor for NetFPGA

Getting Started

Welcome to the Cambridge Verilog Tutor. This is a resource for students to learn the basics of Verilog.

Cambridge Users:

Note:

For users with raven accounts, no registration is necessary. Just log in and get started.

External Users:

Register or Reset account:

Email Address:

Site information

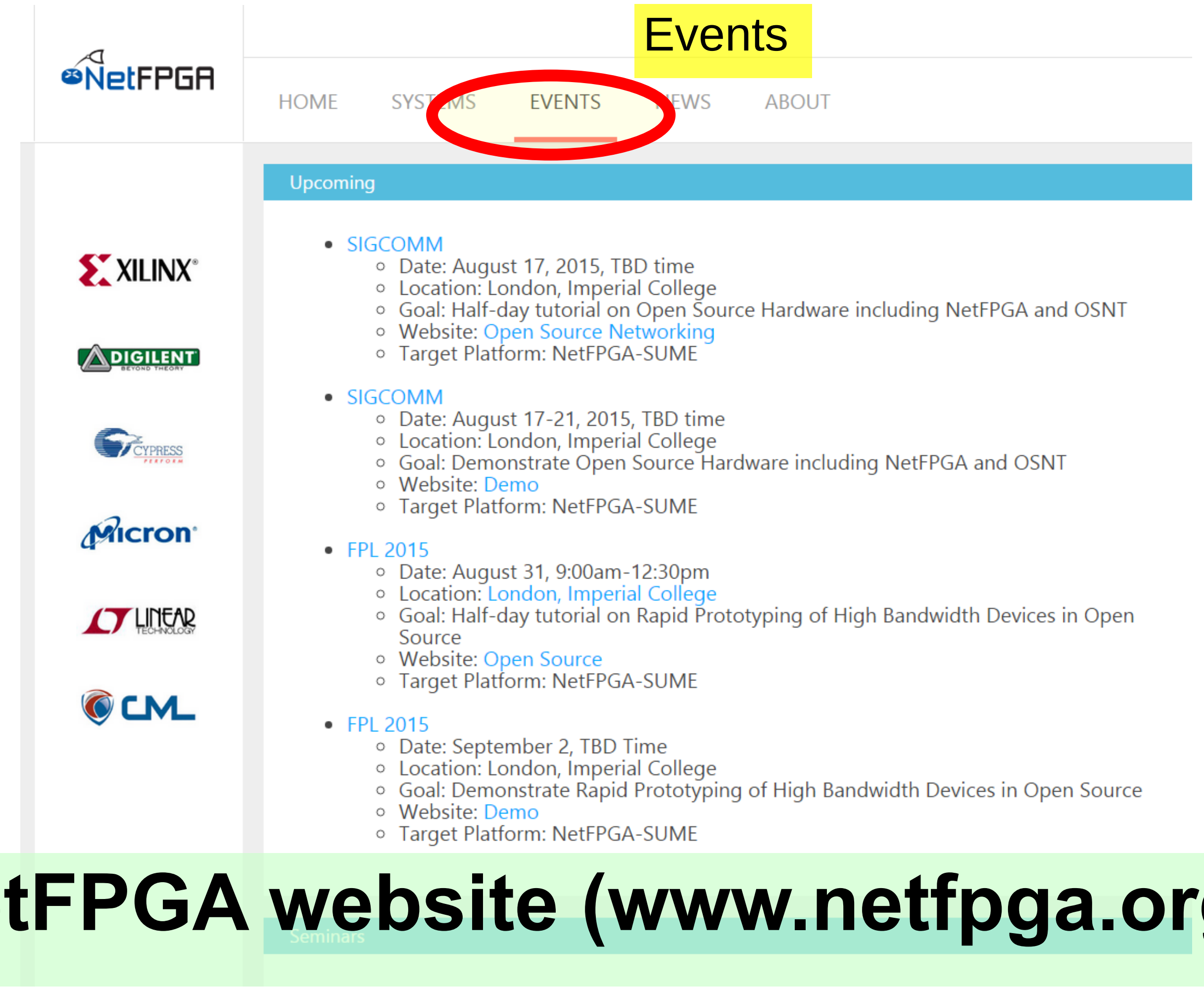
For this website to function correctly, you are required to use a browser with cookies enabled, and which can create SSL connections.

In addition, Javascript and Flash are used to help teach more effectivly, and enabling these technologies is advised, but not strictly required.

We believe that this website conforms to todays web standards. Any modern browser should be able to cope but there may be issues with older browsers.

Support for NetFPGA enhancements provided by redgate® software

Get a hands-on camp



- SIGCOMM
  - Date: August 17, 2015, TBD time
  - Location: London, Imperial College
  - Goal: Half-day tutorial on Open Source Hardware including NetFPGA and OSNT
  - Website: Open Source Networking
  - Target Platform: NetFPGA-SUME
- SIGCOMM
  - Date: August 17-21, 2015, TBD time
  - Location: London, Imperial College
  - Goal: Demonstrate Open Source Hardware including NetFPGA and OSNT
  - Website: Demo
  - Target Platform: NetFPGA-SUME
- FPL 2015
  - Date: August 31, 9:00am-12:30pm
  - Location: London, Imperial College
  - Goal: Half-day tutorial on Rapid Prototyping of High Bandwidth Devices in Open Source
  - Website: Open Source
  - Target Platform: NetFPGA-SUME
- FPL 2015
  - Date: September 2, TBD Time
  - Location: London, Imperial College
  - Goal: Demonstrate Rapid Prototyping of High Bandwidth Devices in Open Source
  - Website: Demo
  - Target Platform: NetFPGA-SUME

NetFPGA website (www.netfpga.org)

All the slides from the first NetFPGA SUME camp, held earlier this month, are available online!

Start with a board….

- **Go to NetFPGA website**
  - www.netfpga.org
- **Go to Digilent website**
  - digilentinc.com

- **NetFPGA contributors get price priority!**

Section X: Concluding remarks

Conclusions

- **Open-source network hardware provides unprecedented flexibility to explore evolvability**
  - *NetFPGA* is a low-cost open source network hardware framework, allowing implementation for a wide range of functionalities at unprecedented rates
- **Open-source hardware enhances experimentation and testing capabilities**
- **Open-source hardware enables hardware/software co-evolution for dynamic functionality offload/onload**
- **Community is everything for open-source**

Acknowledgments (I)

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

Andrew Moore, David Miller, Muhammad Shahbaz, Martin Zadnik, Matthew Grosvenor, Yury Audzevich, Neelakandan Manihatty-Bojan, Georgina Kalogeridou, Jong Hun Han, Noa Zilberman, Gianni Antichi, Charalampos Rotsos, Hwanju Kim, Marco Forconesi, Jinyun Zhang, Bjoern Zeeb, Robert Watson

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

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, Lisa Donatini, Sergio Lopez-Buedo

Steve Wang, Erik Cengar, Michael Alexander, Sam Bobrowicz, Garrett Aufdemberg, Patrick Kane, Tom Weldon

Patrick Lysaght, Kees Vissers, Michaela Blott, Shep Siegel, Cathal McCabe

Acknowledgements (II)

UNIVERSITY OF CAMBRIDGE

XILINX

EPSRC Pioneering research and skills

BROADCOM

Micron

CYPRESS PERFORM

ALGO-LOGIC

DIGILENT

Google

Atomic Rules

Global HiTech

<u>*Disclaimer:*</u> ***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.***

This effort is also sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. This material is approved for public release, distribution unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

BACKUP SLIDES

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)

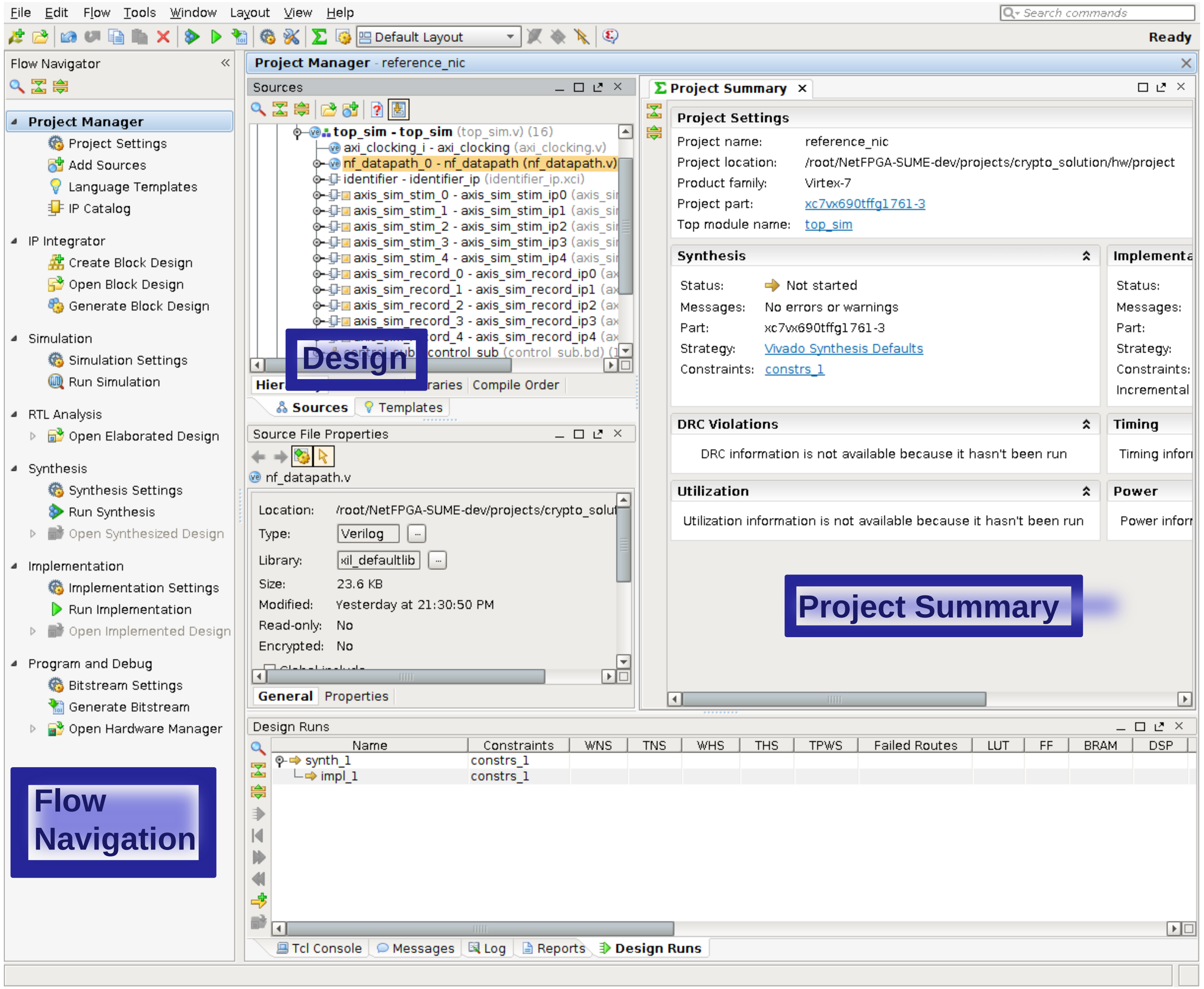
- **To invoke Vivado design tool, run:**

```
# 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/Xilinx/Vivado/2014.4/settings64.sh`
  - `vivado hw/project/<project name>.xpr`

Vivado Design Tool (1)



Flow Navigator

- Project Manager
  - Project Settings
  - Add Sources
  - Language Templates
  - IP Catalog
- IP Integrator
  - Create Block Design
  - Open Block Design
  - Generate Block Design
- Simulation
  - Simulation Settings
  - Run Simulation
- RTL Analysis
  - Open Elaborated Design
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Design
- Implementation
  - Implementation Settings
  - Run Implementation
  - Open Implemented Design
- Program and Debug
  - Bitstream Settings
  - Generate Bitstream
  - Open Hardware Manager

Sources

```
top_sim - top_sim (top_sim.v) (16)
  axi_clocking_i - axi_clocking (axi_clocking.v)
  nf_datapath_0 - nf_datapath (nf_datapath.v)
  identifier - identifier_ip (identifier_ip.xci)
  axis_sim_stim_0 - axis_sim_stim_ip0 (axis_si
  axis_sim_stim_1 - axis_sim_stim_ip1 (axis_si
  axis_sim_stim_2 - axis_sim_stim_ip2 (axis_si
  axis_sim_stim_3 - axis_sim_stim_ip3 (axis_si
  axis_sim_stim_4 - axis_sim_stim_ip4 (axis_si
  axis_sim_record_0 - axis_sim_record_ip0 (ax
  axis_sim_record_1 - axis_sim_record_ip1 (ax
  axis_sim_record_2 - axis_sim_record_ip2 (ax
  axis_sim_record_3 - axis_sim_record_ip3 (ax
  axis_sim_record_4 - axis_sim_record_ip4 (ax
  control_sub - control_sub (control_sub.bd) (1
```

Hier... raries Compile Order | Sources Templates

Source File Properties

nf_datapath.v

Location:	/root/NetFPGA-SUME-dev/projects/crypto_solut
Type:	Verilog
Library:	xil_defaultlib
Size:	23.6 KB
Modified:	Yesterday at 21:30:50 PM
Read-only:	No
Encrypted:	No

General Properties

Project Summary

Project Settings

Project name:	reference_nic
Project location:	/root/NetFPGA-SUME-dev/projects/crypto_solution/hw/project
Product family:	Virtex-7
Project part:	xc7vx690tffg1761-3
Top module name:	top_sim

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7vx690tffg1761-3
Strategy:	Vivado Synthesis Defaults
Constraints:	constrs_1

Implementa

Status:
Messages:
Part:
Strategy:
Constraints:
Incremental

DRC Violations

DRC information is not available because it hasn't been run

Timing

Timing infor

Utilization

Utilization information is not available because it hasn't been run

Power

Power infor

Design Runs

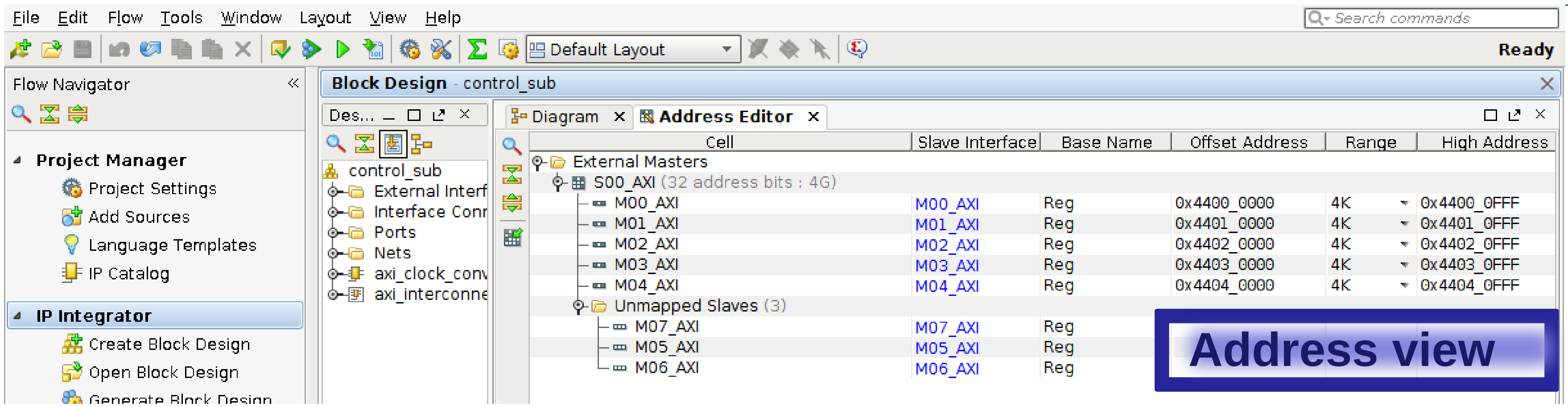
Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP
synth_1	constrs_1										
impl_1	constrs_1										

Tcl Console Messages Log Reports Design Runs

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)



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
External Masters					
S00_AXI (32 address bits : 4G)					
M00_AXI	M00_AXI	Reg	0x4400_0000	4K	0x4400_0FFF
M01_AXI	M01_AXI	Reg	0x4401_0000	4K	0x4401_0FFF
M02_AXI	M02_AXI	Reg	0x4402_0000	4K	0x4402_0FFF
M03_AXI	M03_AXI	Reg	0x4403_0000	4K	0x4403_0FFF
M04_AXI	M04_AXI	Reg	0x4404_0000	4K	0x4404_0FFF
Unmapped Slaves (3)					
M07_AXI	M07_AXI	Reg			
M05_AXI	M05_AXI	Reg			
M06_AXI	M06_AXI	Reg			

- **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**

Project Design Flow

- **There are several ways to design and integrate a project, e.g.**
  - Using Verilog files for connectivity and TCL scripts for project definition
  - Using Vivado’s Block Design (IPI) flow

- **We will use the first, but introduce the second**

Project Integration

- vi $NF_DESIGN_DIR/hw/nf_datapath.v
- Add the new module between the output port lookup and output queues
- Connect S3_AXI to the AXI_Lite interface of the block

Project Integration

- **Edit the TCL file which generates the project:**
- **vi $NF_DESIGN_DIR/hw/tcl/<project_name>_sim.tcl**
- **Add the following lines:**

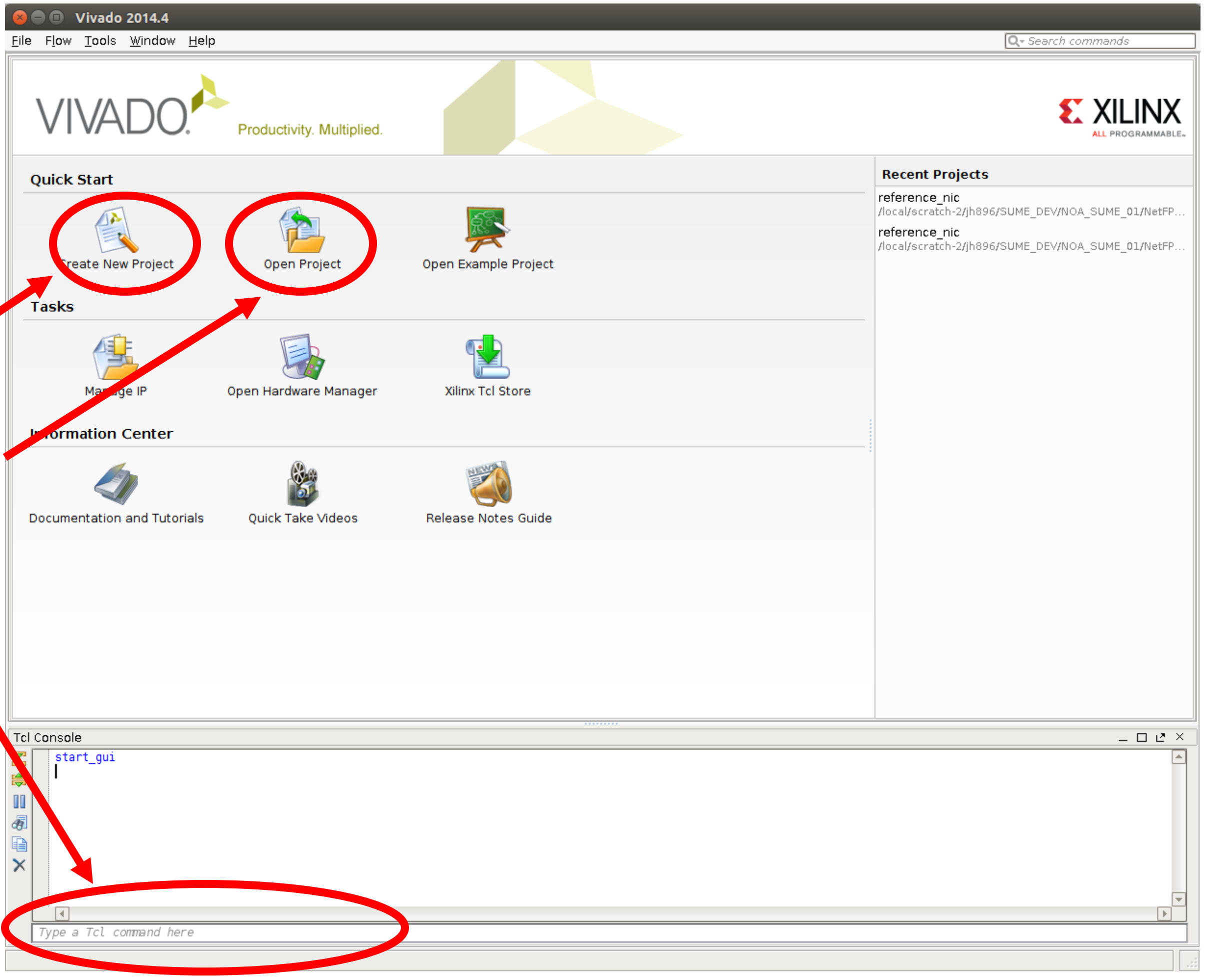
```
create_ip -name <core_name> -vendor NetFPGA -library NetFPGA -module_name <core>_ip

set_property generate_synth_checkpoint false [get_files <core>_ip.xci]

reset_target all [get_ips <core>_ip]

generate_target all [get_ips <core>_ip]
```

- **Save time for later, add the same text also in:**

$NF_DESIGN_DIR/tcl/<project_name>.tcl

Project Integration – Block Design

Create a new project
OR
Open an existing project
OR
run a TCL script
(also through tools)

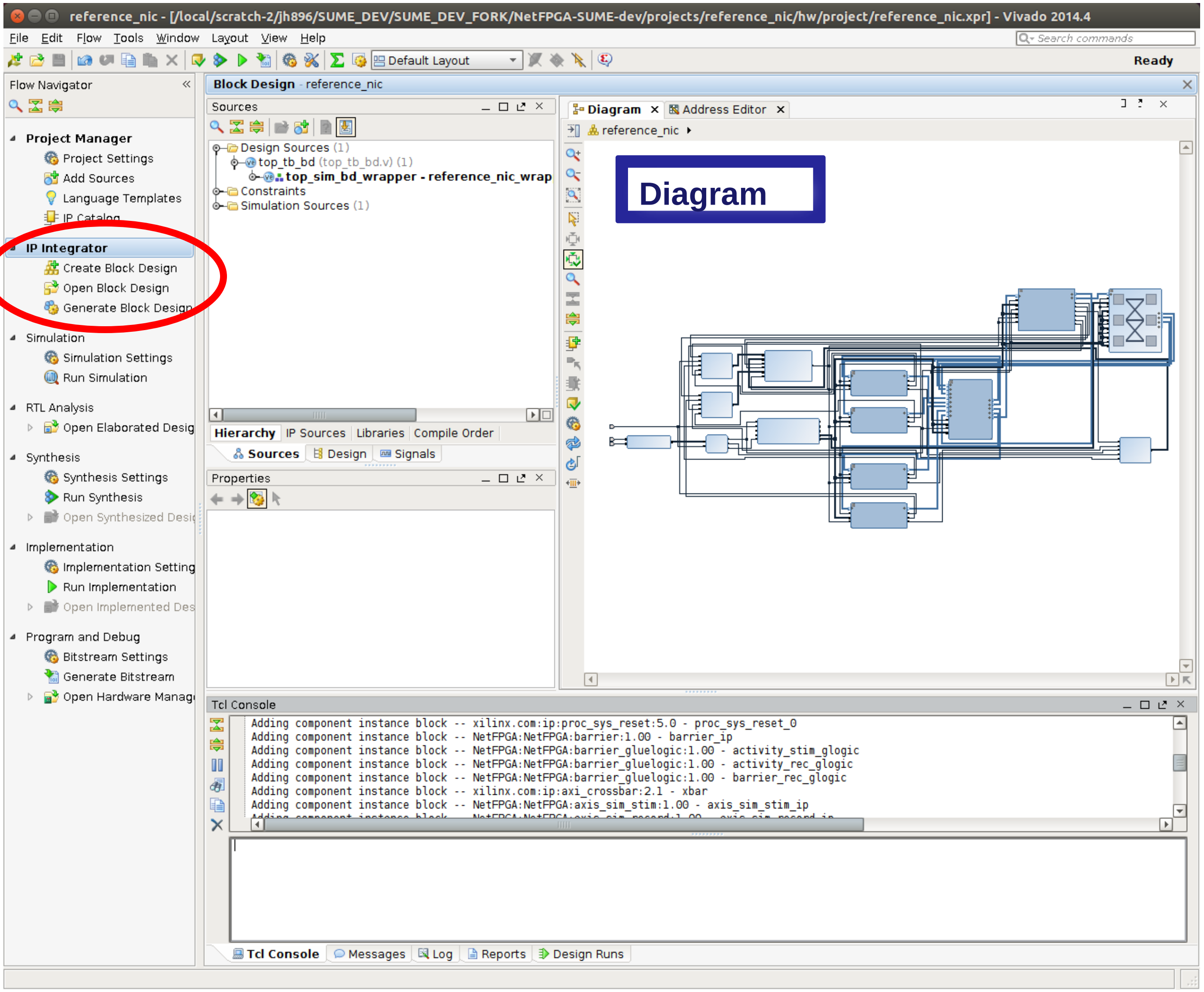


Recent Projects:
reference_nic
/local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFP...
reference_nic
/local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFP...

```
start_gui
```

Project Integration – Block Design (2)

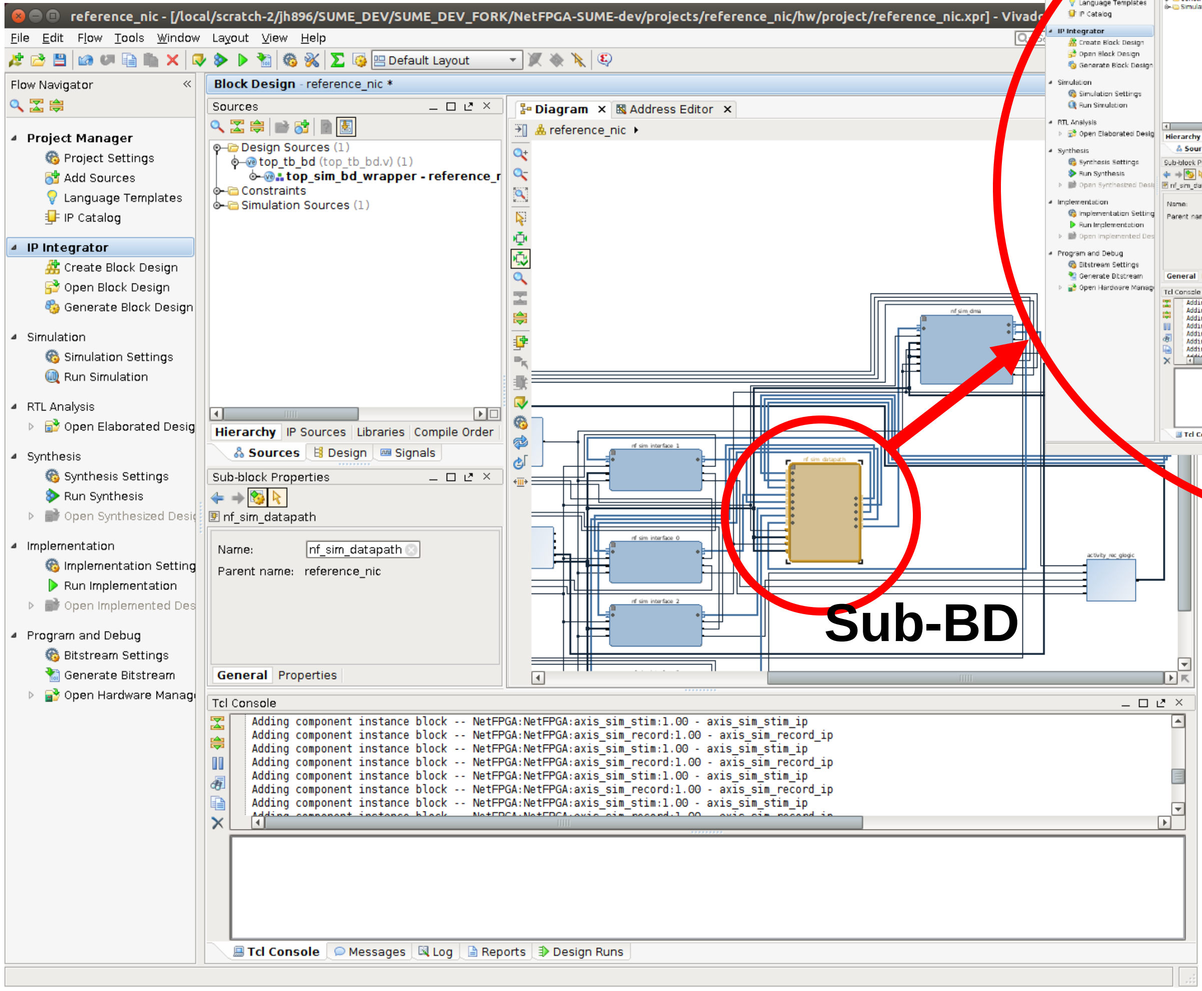
Open block design



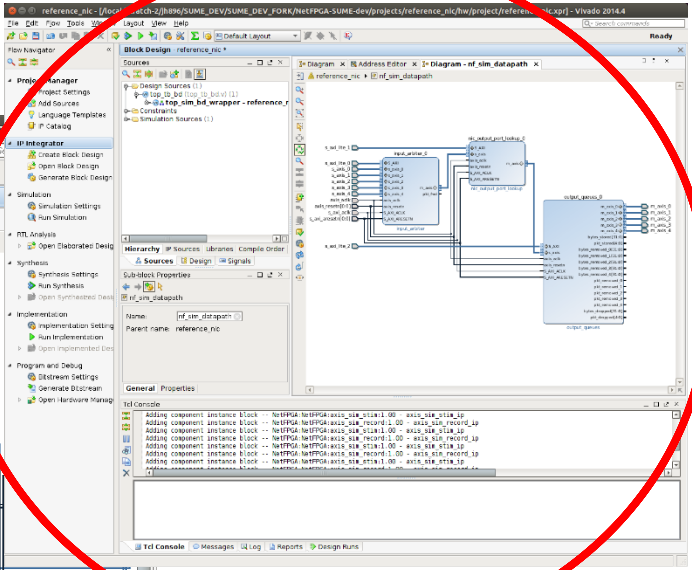
```
Adding component instance block -- xilinx.com:ip:proc_sys_reset:5.0 - proc_sys_reset_0
Adding component instance block -- NetFPGA:NetFPGA:barrier:1.00 - barrier_ip
Adding component instance block -- NetFPGA:NetFPGA:barrier_gluelogic:1.00 - activity_stim_glogic
Adding component instance block -- NetFPGA:NetFPGA:barrier_gluelogic:1.00 - activity_rec_glogic
Adding component instance block -- NetFPGA:NetFPGA:barrier_gluelogic:1.00 - barrier_rec_glogic
Adding component instance block -- xilinx.com:ip:axi_crossbar:2.1 - xbar
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
```

Project Integration – Block Design (3)

Opening Sub-BD



```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
```

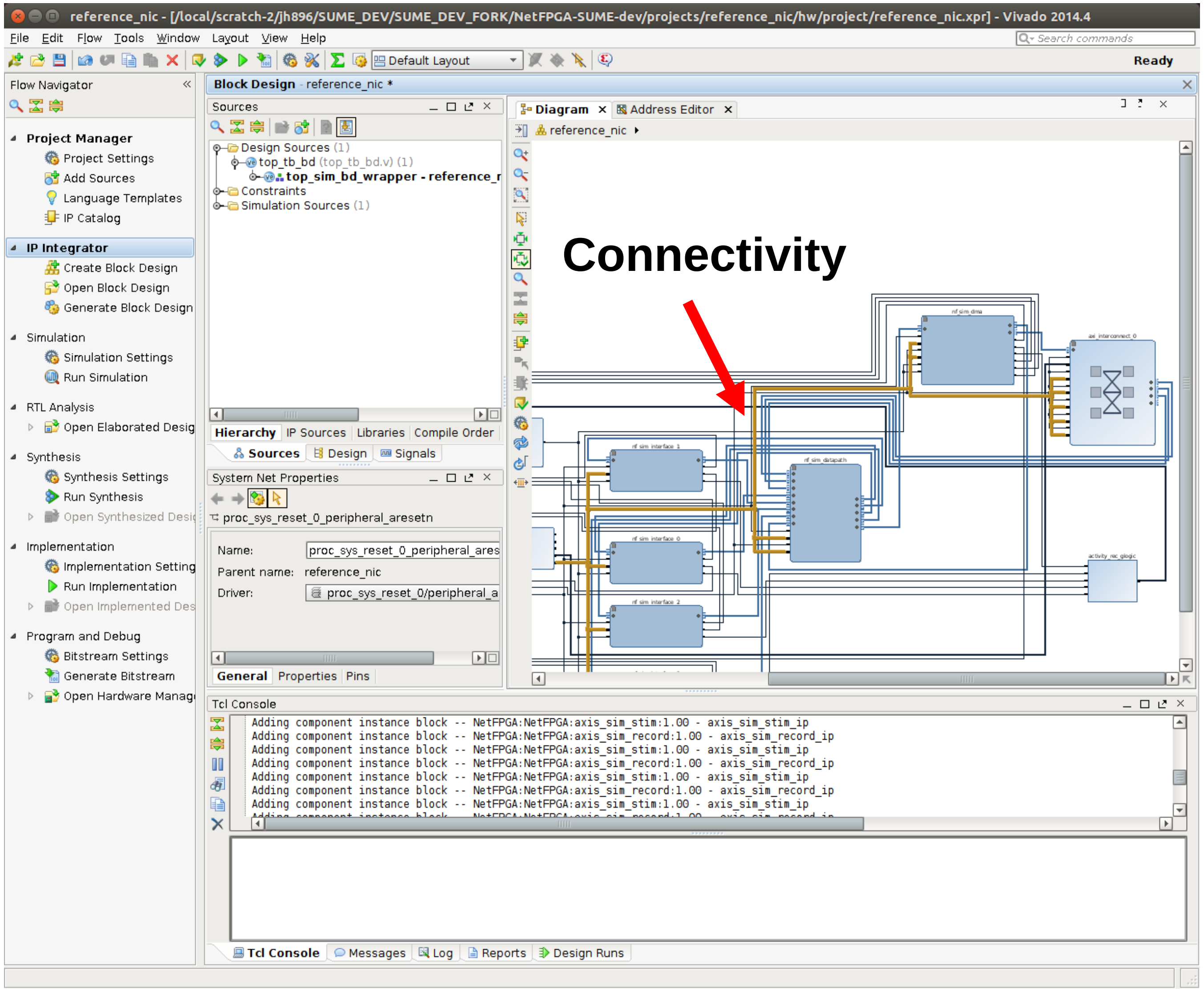


reference_nic - [/local/scratch-2/jh896/SUME_DEV/SUME_DEV_FORK/NetFPGA-SUME-dev/projects/reference_nic/hw/project/reference_nic.xpr] - Vivado 2014.4
Ready
Block Design - reference_nic *
Diagram
Address Editor
Diagram - nf_sim_datapath
reference_nic
nf_sim_datapath
input_arbiter_0
nic_output_port_lookup_0
output_queues_0
Sub-block Properties
nf_sim_datapath
Name: nf_sim_datapath
Parent name: reference_nic
Tcl Console

```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
```

Sub-BD

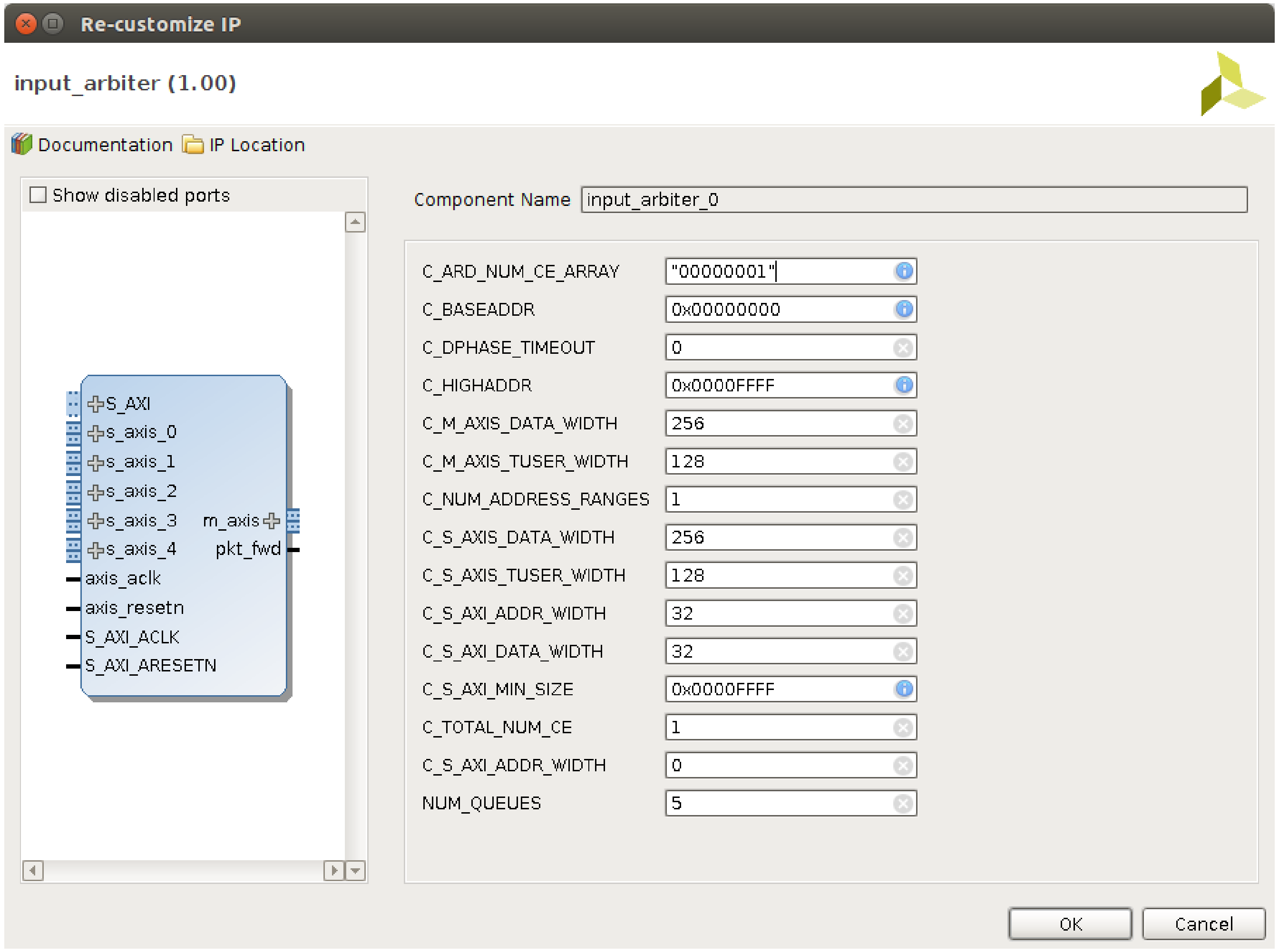
Project Integration – Block Design (4)



```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
```

Project Integration – Block Design (5)

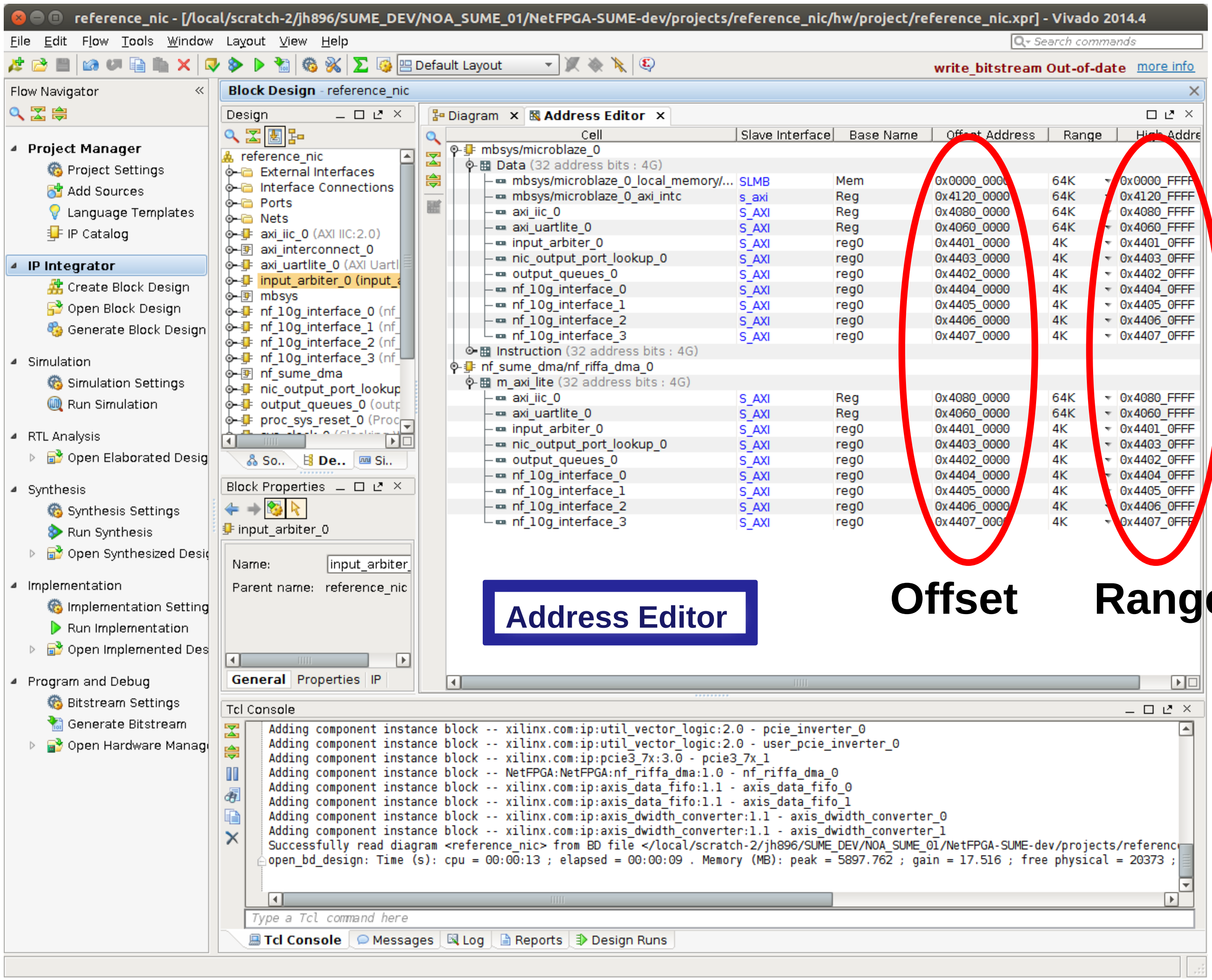
Setting module parameters



Component Name input_arbiter_0

C_ARD_NUM_CE_ARRAY	"00000001"
C_BASEADDR	0x00000000
C_DPHASE_TIMEOUT	0
C_HIGHADDR	0x0000FFFF
C_M_AXIS_DATA_WIDTH	256
C_M_AXIS_TUSER_WIDTH	128
C_NUM_ADDRESS_RANGES	1
C_S_AXIS_DATA_WIDTH	256
C_S_AXIS_TUSER_WIDTH	128
C_S_AXI_ADDR_WIDTH	32
C_S_AXI_DATA_WIDTH	32
C_S_AXI_MIN_SIZE	0x0000FFFF
C_TOTAL_NUM_CE	1
C_S_AXI_ADDR_WIDTH	0
NUM_QUEUES	5

Project Integration – Block Design (6)



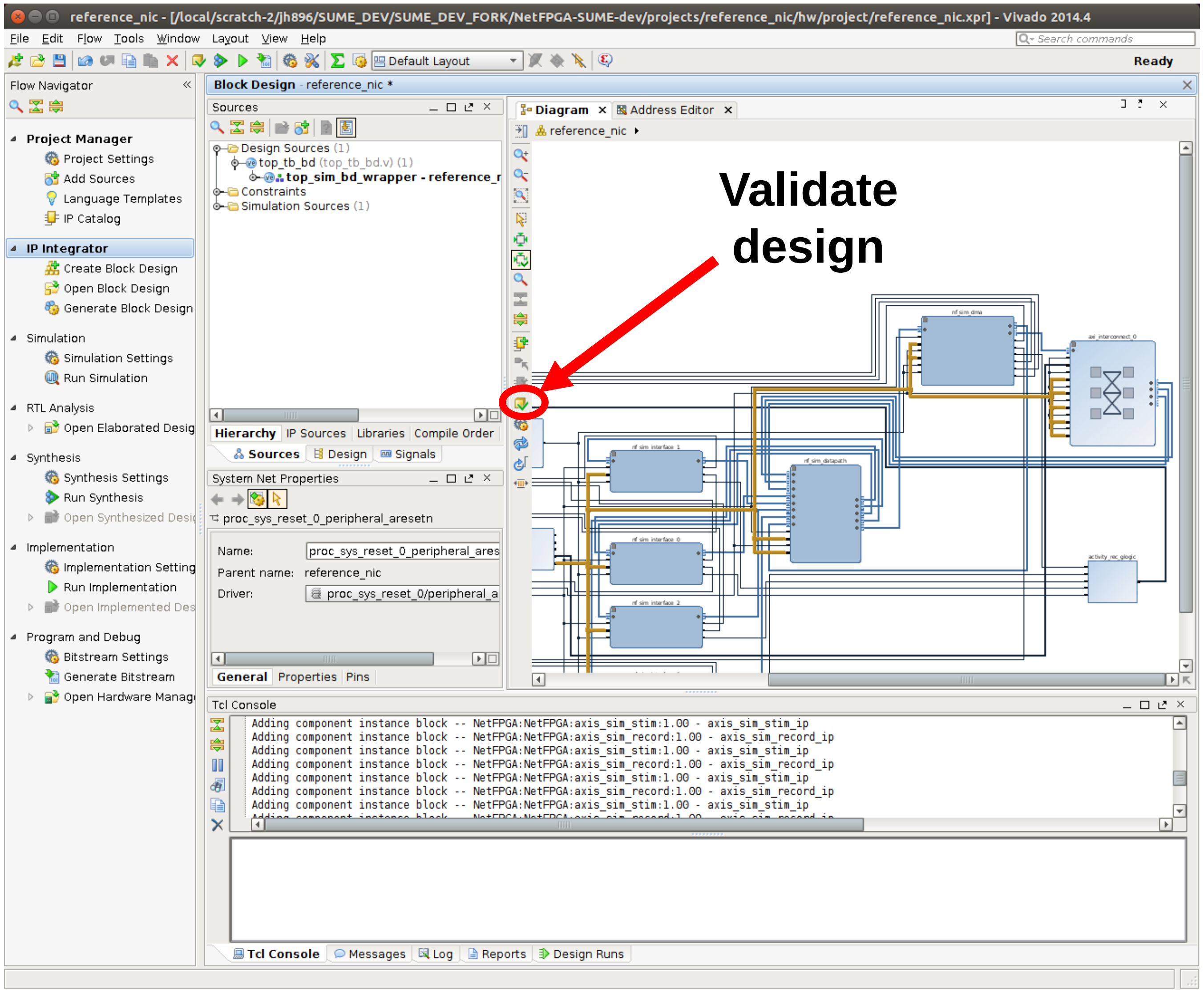
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
mbsys/microblaze_0					
Data (32 address bits : 4G)					
mbsys/microblaze_0_local_memory/...	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
mbsys/microblaze_0_axi_intc	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
axi_iic_0	S_AXI	Reg	0x4080_0000	64K	0x4080_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
input_arbiter_0	S_AXI	reg0	0x4401_0000	4K	0x4401_0FFF
nic_output_port_lookup_0	S_AXI	reg0	0x4403_0000	4K	0x4403_0FFF
output_queues_0	S_AXI	reg0	0x4402_0000	4K	0x4402_0FFF
nf_10g_interface_0	S_AXI	reg0	0x4404_0000	4K	0x4404_0FFF
nf_10g_interface_1	S_AXI	reg0	0x4405_0000	4K	0x4405_0FFF
nf_10g_interface_2	S_AXI	reg0	0x4406_0000	4K	0x4406_0FFF
nf_10g_interface_3	S_AXI	reg0	0x4407_0000	4K	0x4407_0FFF
Instruction (32 address bits : 4G)					
nf_sume_dma/nf_riffa_dma_0					
m_axi_lite (32 address bits : 4G)					
axi_iic_0	S_AXI	Reg	0x4080_0000	64K	0x4080_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
input_arbiter_0	S_AXI	reg0	0x4401_0000	4K	0x4401_0FFF
nic_output_port_lookup_0	S_AXI	reg0	0x4403_0000	4K	0x4403_0FFF
output_queues_0	S_AXI	reg0	0x4402_0000	4K	0x4402_0FFF
nf_10g_interface_0	S_AXI	reg0	0x4404_0000	4K	0x4404_0FFF
nf_10g_interface_1	S_AXI	reg0	0x4405_0000	4K	0x4405_0FFF
nf_10g_interface_2	S_AXI	reg0	0x4406_0000	4K	0x4406_0FFF
nf_10g_interface_3	S_AXI	reg0	0x4407_0000	4K	0x4407_0FFF

Name: input_arbiter
Parent name: reference_nic

```
Adding component instance block -- xilinx.com:ip:util_vector_logic:2.0 - pcie_inverter_0
Adding component instance block -- xilinx.com:ip:util_vector_logic:2.0 - user_pcie_inverter_0
Adding component instance block -- xilinx.com:ip:pcie3_7x:3.0 - pcie3_7x_1
Adding component instance block -- NetFPGA:NetFPGA:nf_riffa_dma:1.0 - nf_riffa_dma_0
Adding component instance block -- xilinx.com:ip:axis_data_fifo:1.1 - axis_data_fifo_0
Adding component instance block -- xilinx.com:ip:axis_data_fifo:1.1 - axis_data_fifo_1
Adding component instance block -- xilinx.com:ip:axis_dwidth_converter:1.1 - axis_dwidth_converter_0
Adding component instance block -- xilinx.com:ip:axis_dwidth_converter:1.1 - axis_dwidth_converter_1
Successfully read diagram <reference_nic> from BD file </local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFPGA-SUME-dev/projects/referenc
open_bd_design: Time (s): cpu = 00:00:13 ; elapsed = 00:00:09 . Memory (MB): peak = 5897.762 ; gain = 17.516 ; free physical = 20373 ;
```

Address Editor

Project Integration – Block Design (7)



reference_nic - [/local/scratch-2/jh896/SUME_DEV/SUME_DEV_FORK/NetFPGA-SUME-dev/projects/reference_nic/hw/project/reference_nic.xpr] - Vivado 2014.4

Block Design - reference_nic *

Sources
- Design Sources (1)
  - top_tb_bd (top_tb_bd.v) (1)
    - top_sim_bd_wrapper - reference_r
- Constraints
- Simulation Sources (1)

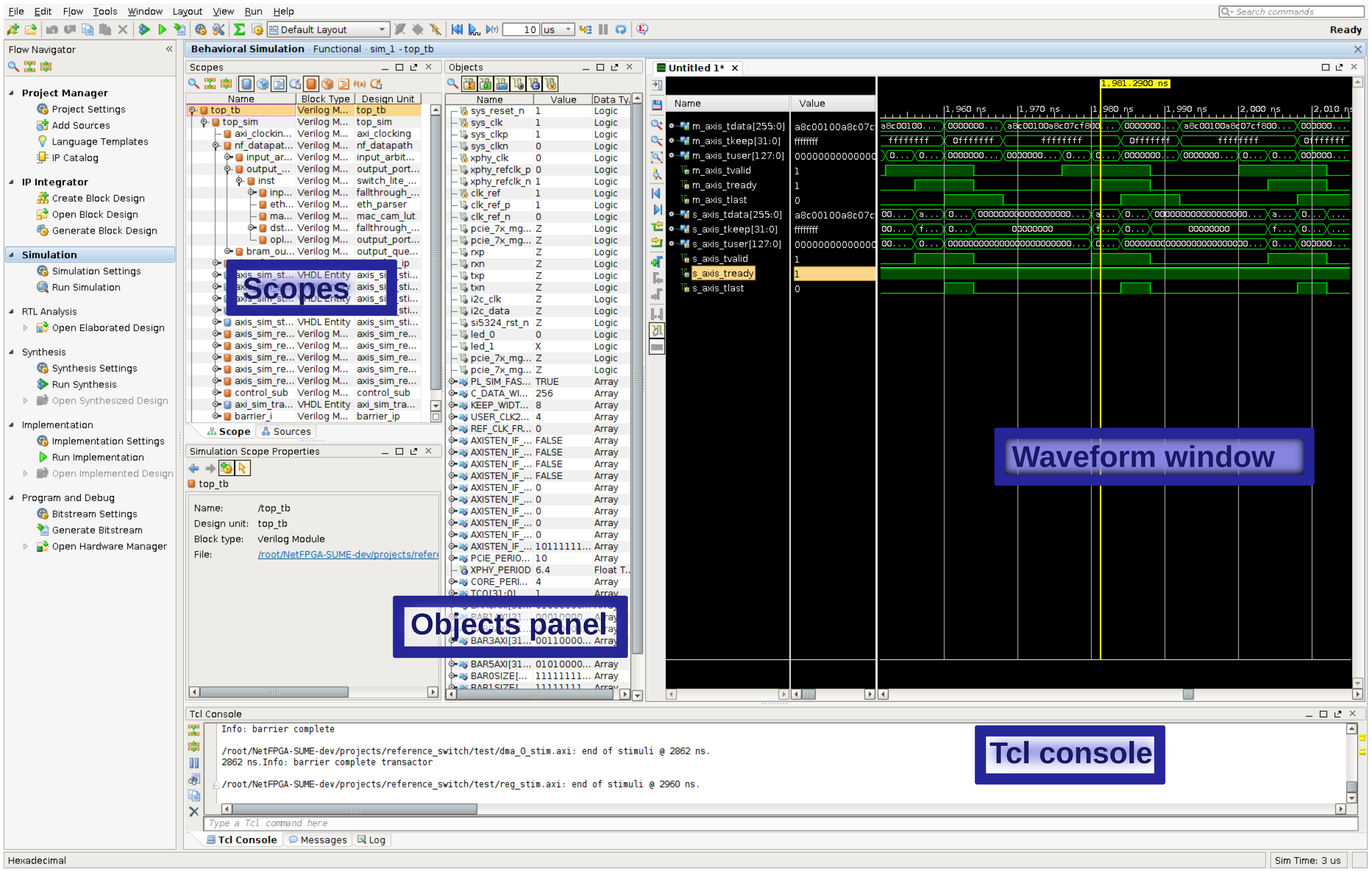
System Net Properties

proc_sys_reset_0_peripheral_aresetn

Name: proc_sys_reset_0_peripheral_ares
Parent name: reference_nic
Driver: proc_sys_reset_0/peripheral_a

```
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_record:1.00 - axis_sim_record_ip
Adding component instance block -- NetFPGA:NetFPGA:axis_sim_stim:1.00 - axis_sim_stim_ip
```

Running simulation in xSim



Name: /top_tb
Design unit: top_tb
Block type: Verilog Module
File: /root/NetFPGA-SUME-dev/projects/refere

```
Info: barrier complete

/root/NetFPGA-SUME-dev/projects/reference_switch/test/dma_0_stim.axi: end of stimuli @ 2862 ns.
2862 ns.Info: barrier complete transactor

/root/NetFPGA-SUME-dev/projects/reference_switch/test/reg_stim.axi: end of stimuli @ 2960 ns.
```

Running simulation in xSim (2)

- Scopes panel: displays process and instance hierarchy
- Objects panel: displays simulation objects associated with the instance selected in the instance panel
- Waveform window: displays wave configuration consisting of signals and busses
- Tcl console: displays simulator generated messages and can executes Tcl commands

Register Infrastructure

Specifying the Key via a Register

- **Set the key via a register**
  - Instead of a constant value
- **Requires understanding the registers system ☺**
- **Registers system:**
  - Automatically generated
  - Implementing registers in a module
    - Use automatically generated cpu_regs module
  - Need to implement the registers' functional logic

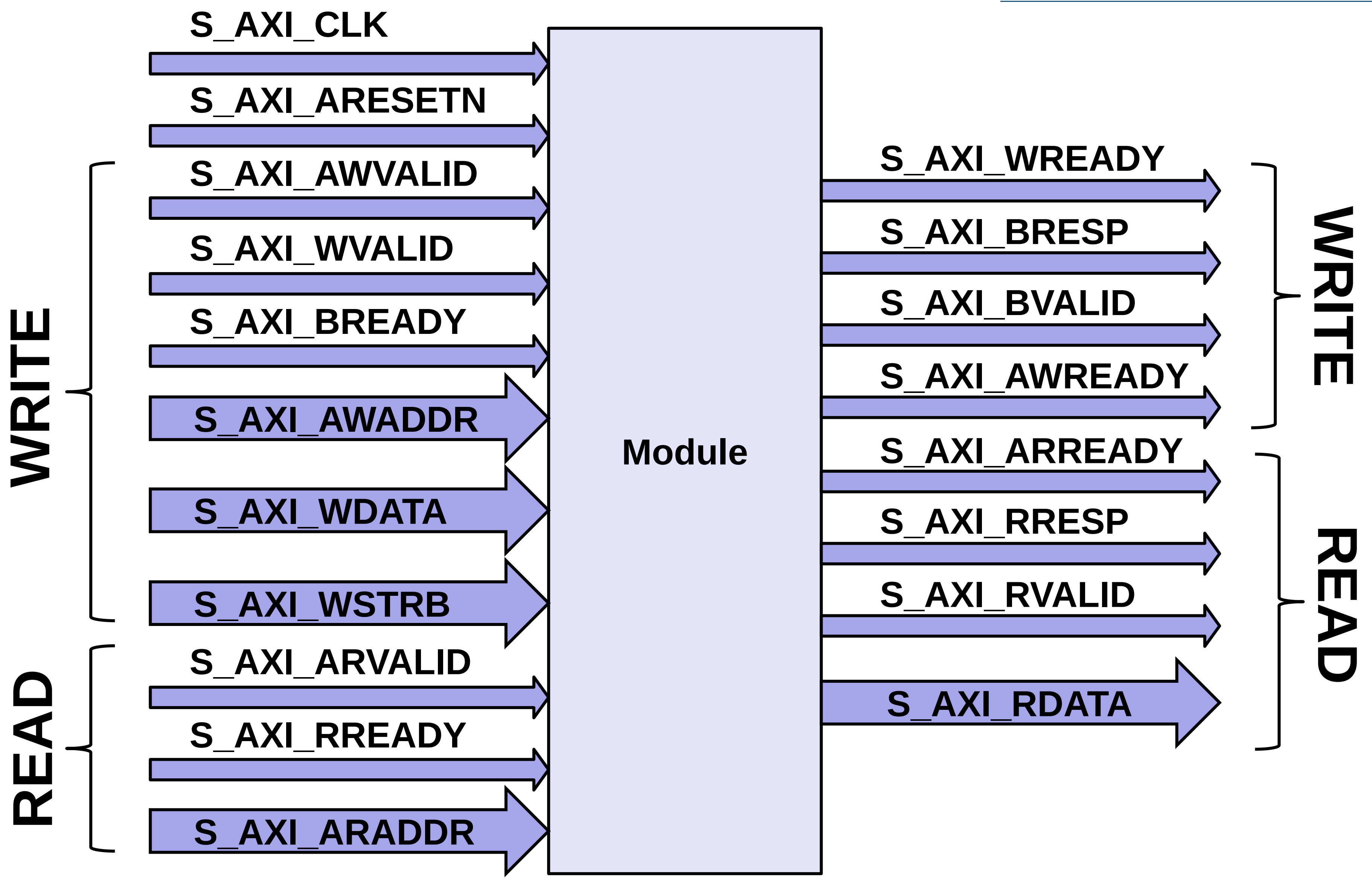
Registers bus

- We learnt that packets stream follows the AXI4-Stream paradigm

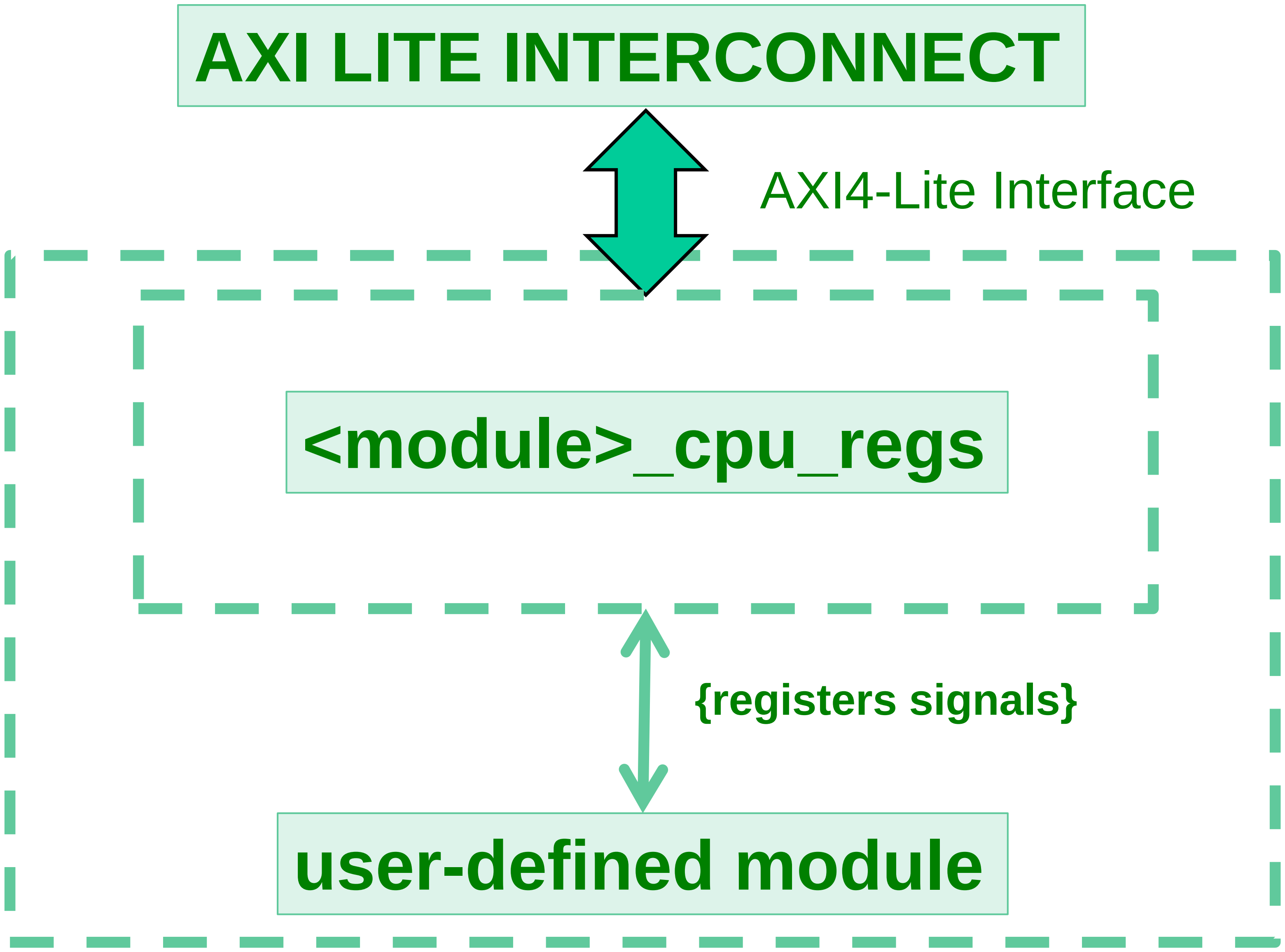
- Register communication follows the AXI4-Lite paradigm

- 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)

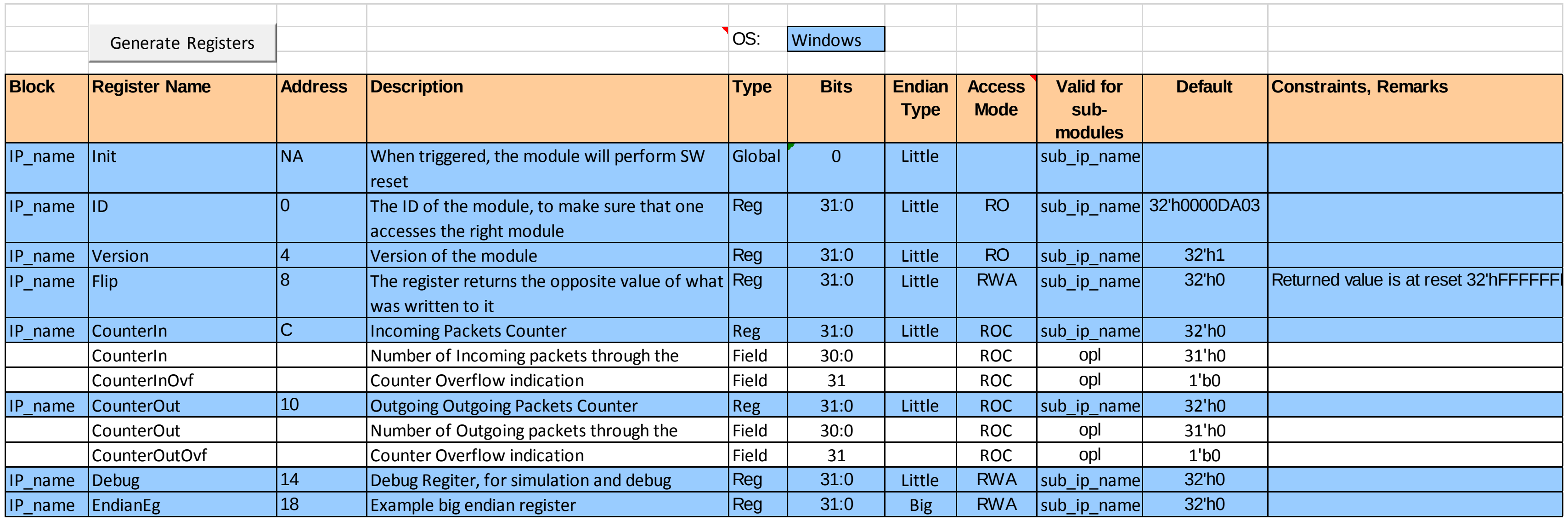


Register bus



Registers – Module generation

- **Spreadsheet based**
- **Defines all the registers you intend to support and their properties**
- **Generates a python script (regs_gen.py), which generates the outputs**

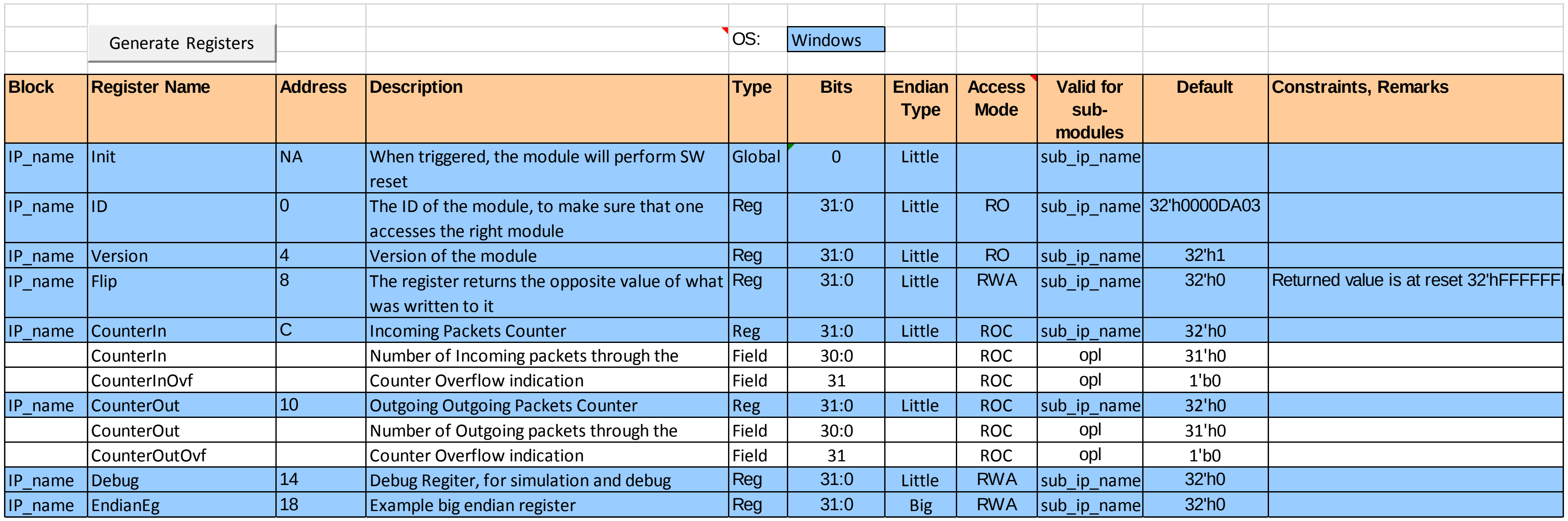


Generate Registers

OS: Windows

Block	Register Name	Address	Description	Type	Bits	Endian Type	Access Mode	Valid for sub-modules	Default	Constraints, Remarks
IP_name	Init	NA	When triggered, the module will perform SW reset	Global	0	Little		sub_ip_name		
IP_name	ID	0	The ID of the module, to make sure that one accesses the right module	Reg	31:0	Little	RO	sub_ip_name	32'h0000DA03	
IP_name	Version	4	Version of the module	Reg	31:0	Little	RO	sub_ip_name	32'h1	
IP_name	Flip	8	The register returns the opposite value of what was written to it	Reg	31:0	Little	RWA	sub_ip_name	32'h0	Returned value is at reset 32'hFFFFFFF
IP_name	CounterIn	C	Incoming Packets Counter	Reg	31:0	Little	ROC	sub_ip_name	32'h0	
	CounterIn		Number of Incoming packets through the	Field	30:0		ROC	opl	31'h0	
	CounterInOvf		Counter Overflow indication	Field	31		ROC	opl	1'b0	
IP_name	CounterOut	10	Outgoing Outgoing Packets Counter	Reg	31:0	Little	ROC	sub_ip_name	32'h0	
	CounterOut		Number of Outgoing packets through the	Field	30:0		ROC	opl	31'h0	
	CounterOutOvf		Counter Overflow indication	Field	31		ROC	opl	1'b0	
IP_name	Debug	14	Debug Regiter, for simulation and debug	Reg	31:0	Little	RWA	sub_ip_name	32'h0	
IP_name	EndianEg	18	Example big endian register	Reg	31:0	Big	RWA	sub_ip_name	32'h0	

Registers – Module generation



Generate Registers

OS: Windows

Block	Register Name	Address	Description	Type	Bits	Endian Type	Access Mode	Valid for sub-modules	Default	Constraints, Remarks
IP_name	Init	NA	When triggered, the module will perform SW reset	Global	0	Little		sub_ip_name		
IP_name	ID	0	The ID of the module, to make sure that one accesses the right module	Reg	31:0	Little	RO	sub_ip_name	32'h0000DA03	
IP_name	Version	4	Version of the module	Reg	31:0	Little	RO	sub_ip_name	32'h1	
IP_name	Flip	8	The register returns the opposite value of what was written to it	Reg	31:0	Little	RWA	sub_ip_name	32'h0	Returned value is at reset 32'hFFFFFF
IP_name	CounterIn	C	Incoming Packets Counter	Reg	31:0	Little	ROC	sub_ip_name	32'h0	
	CounterIn		Number of Incoming packets through the	Field	30:0		ROC	opl	31'h0	
	CounterInOvf		Counter Overflow indication	Field	31		ROC	opl	1'b0	
IP_name	CounterOut	10	Outgoing Outgoing Packets Counter	Reg	31:0	Little	ROC	sub_ip_name	32'h0	
	CounterOut		Number of Outgoing packets through the	Field	30:0		ROC	opl	31'h0	
	CounterOutOvf		Counter Overflow indication	Field	31		ROC	opl	1'b0	
IP_name	Debug	14	Debug Regiter, for simulation and debug	Reg	31:0	Little	RWA	sub_ip_name	32'h0	
IP_name	EndianEg	18	Example big endian register	Reg	31:0	Big	RWA	sub_ip_name	32'h0	

Registers – Module generation

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)
- RWCW - 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 under <core name>/data
- **<module>_regs_defines.tcl**
- To be placed under under <core name>/data
- **<module>_regs_defines.txt – used by test harness**
- To be placed under under <core name>/data

Adding Registers Logic - Example

- **Usage examples:**

```
always @(posedge axi_aclk)
    if (~resetn_sync) begin
            id_reg <= #1   `REG_ID_DEFAULT;
            ip2cpu_flip_reg <= #1   `REG_FLIP_DEFAULT;
            pktin_reg <= #1   `REG_PKTIN_DEFAULT;
            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

```
cd ~/NetFPGA-SUME-
  alpha/lib/sw/std/apps/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

Testing Registers with Simulation

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)
- **Use within run.py**
- **You don’t need to edit any other file**

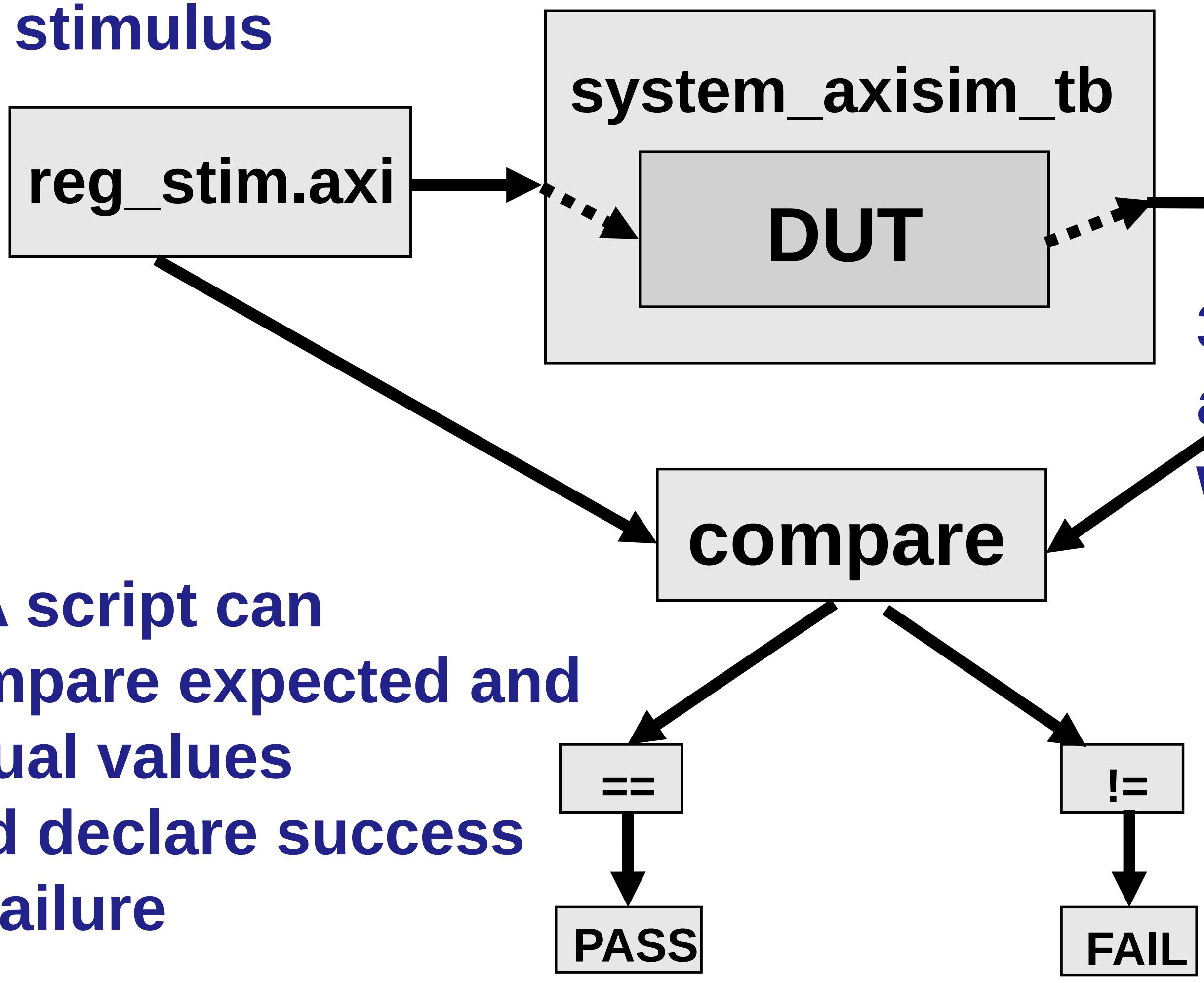
Simulating Register Access

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

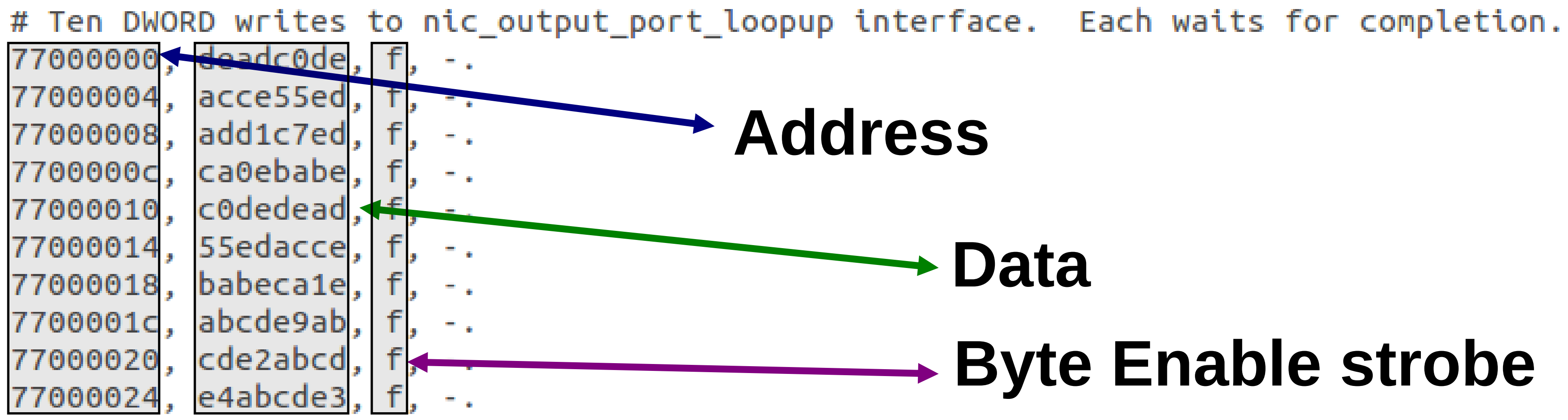


Legend:
- DUT: Design Under Test
- stim: stimulus
- tb: testbench
- sim: simulation

Registers Stimulus (1)

```
cd $NF_DESIGN_DIR/test/
less reg_stim.axi
```

- **An example of write format :**



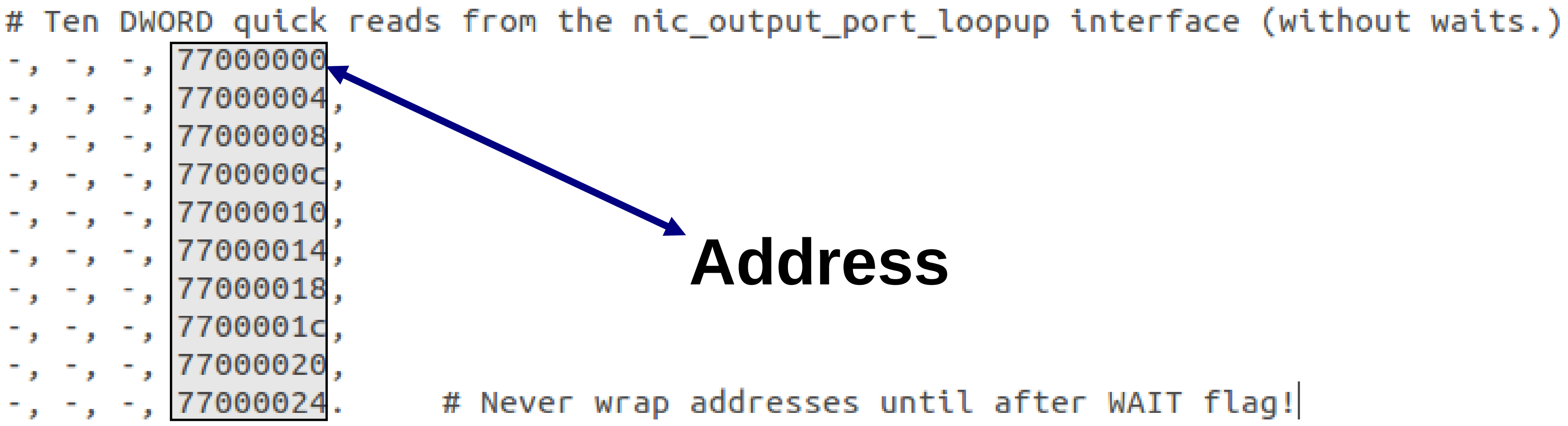
```
# Ten DWORD writes to nic_output_port_loopup interface.  Each waits for completion.
77000000, deadc0de, f, -.
77000004, acce55ed, f, -.
77000008, add1c7ed, f, -.
7700000c, ca0ebabe, f, -.
77000010, c0dedead, f, -.
77000014, 55edacce, f, -.
77000018, babeca1e, f, -.
7700001c, abcde9ab, f, -.
77000020, cde2abcd, f, -.
77000024, e4abcde3, f, -.
```

<u>with other useful information like, time, barriers etc..</u>

Registers Stimulus (2)

```
cd $NF_DESIGN_DIR/test/
less reg_stim.axi
```

- **An example read format :**

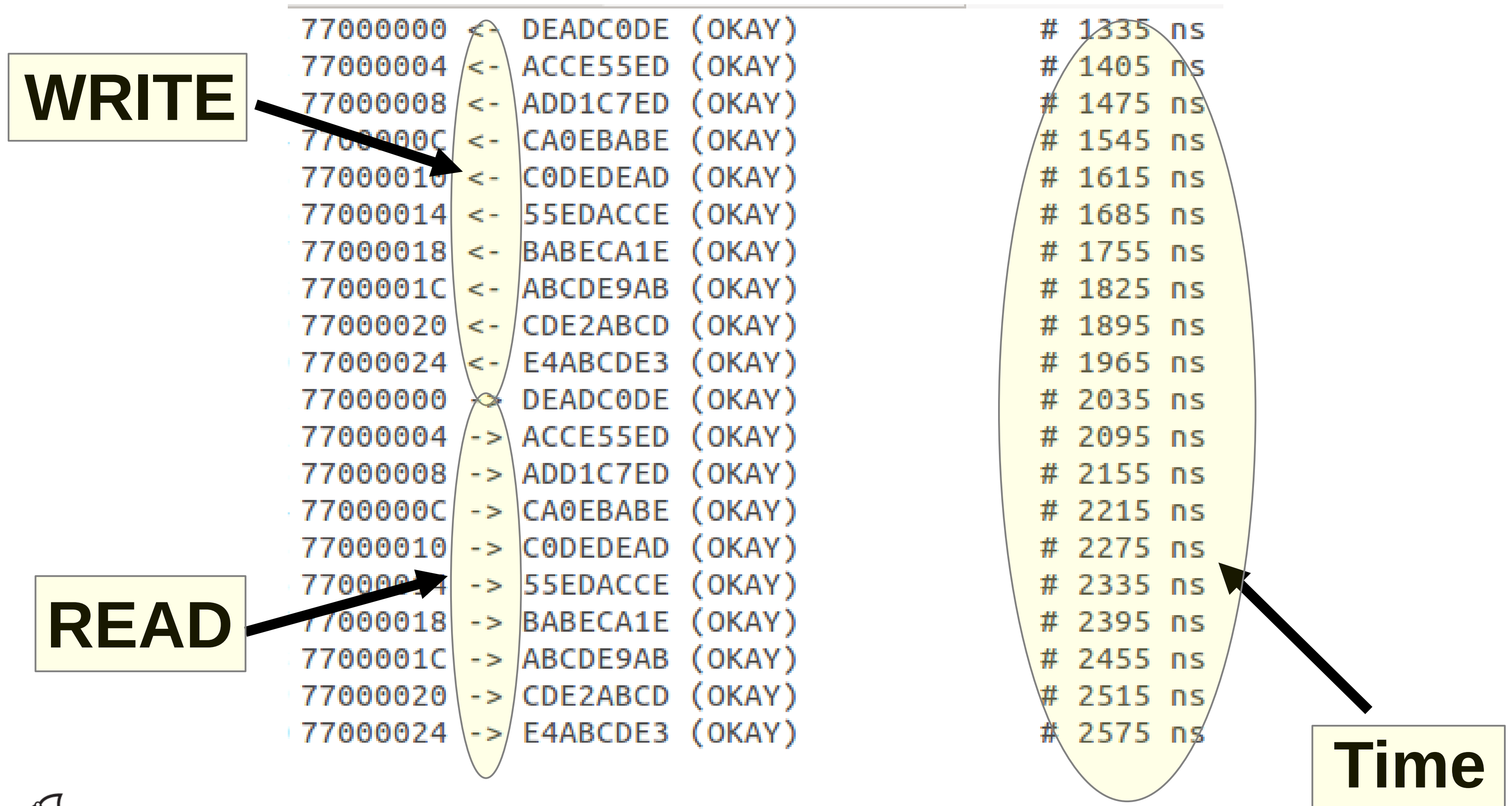


```
# Ten DWORD quick reads from the nic_output_port_loopup interface (without waits.)
-, -, -, 77000000
-, -, -, 77000004,
-, -, -, 77000008,
-, -, -, 7700000c,
-, -, -, 77000010,
-, -, -, 77000014,
-, -, -, 77000018,
-, -, -, 7700001c,
-, -, -, 77000020,
-, -, -, 77000024.        # Never wrap addresses until after WAIT flag!
```

with other useful information like, time, barriers etc..

Registers Access Log

```
cd $NF_DESIGN_DIR/test/
less reg_stim.log
```



```
77000000 <- DEADC0DE (OKAY)                 # 1335 ns
77000004 <- ACCE55ED (OKAY)                 # 1405 ns
77000008 <- ADD1C7ED (OKAY)                 # 1475 ns
7700000C <- CA0EBABE (OKAY)                 # 1545 ns
77000010 <- C0DEDEAD (OKAY)                 # 1615 ns
77000014 <- 55EDACCE (OKAY)                 # 1685 ns
77000018 <- BABECA1E (OKAY)                 # 1755 ns
7700001C <- ABCDE9AB (OKAY)                 # 1825 ns
77000020 <- CDE2ABCD (OKAY)                 # 1895 ns
77000024 <- E4ABCDE3 (OKAY)                 # 1965 ns
77000000 -> DEADC0DE (OKAY)                 # 2035 ns
77000004 -> ACCE55ED (OKAY)                 # 2095 ns
77000008 -> ADD1C7ED (OKAY)                 # 2155 ns
7700000C -> CA0EBABE (OKAY)                 # 2215 ns
77000010 -> C0DEDEAD (OKAY)                 # 2275 ns
77000014 -> 55EDACCE (OKAY)                 # 2335 ns
77000018 -> BABECA1E (OKAY)                 # 2395 ns
7700001C -> ABCDE9AB (OKAY)                 # 2455 ns
77000020 -> CDE2ABCD (OKAY)                 # 2515 ns
77000024 -> E4ABCDE3 (OKAY)                 # 2575 ns
```

Build and Test Hardware

Synthesis

- **To synthesize your project:**

```
cd $NF_DESIGN_DIR
make
```

Hardware Tests

- **Test compiled hardware**
- **Test infrastructure provided to**
  - Read/Write registers
  - Read/Write tables
  - Send Packets
  - Check Counters

Python Libraries

- **Start packet capture on interfaces**

- **Clear all tables in hardware**

- **Create packets**
  - MAC header
  - IP header
  - PDU

- **Read/Write registers**

- **Read/Write reference router tables**
  - Longest Prefix Match
  - ARP
  - Destination IP Filter

- **The same libraries used in the simulation infrastructure…**

Creating a Hardware Test

Useful functions:

- Register access:
  - libsume.regwrite(addr, value)
  - libsume.regread_expect(addr, expect)
- 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)

Understanding Hardware Test

- `cd $NF_DESIGN_DIR/test/both_simple_broadcast`
- `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_simple_broadcast

$$\begin{tikzcd}
\text{simple} & \text{broadcast} \\
\textbf{major} \arrow[u] & \textbf{minor} \arrow[u]
\end{tikzcd}$$

- **Run the command**

```
./nf_test.py hw --major simple --minor broadcast
```

Running Hardware Tests

- **Having problems?**

- **Take advantage of the wiki!**

  https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/Hardware-Tests

  – Detailed explanations
  – Tips for debug

…and now let's program the board!!!

Program the NetFPGA

Several options:

- **Program the bit file using Vivado's Hardware Manager**
- **Load a bit file for FPGA programming using impact script**

```
 ~/NetFPGA-SUME-alpha/tools/scripts/load_bitfile.py \
   -i $DESIGN_DIR/bitfiles/drop_nth_switch.bit
```

- **Use Xilinx Microprocessor Debugger (XMD)**

```
xmd
fpga –f <filename.bit>
```

WHILE YOU WAIT…. Here is one we built earlier:

```
~/NetFPGA-SUME-alpha/tools/scripts/load_bitfile.py -i \
~/NetFPGA-SUME-alpha/projects/drop_nth_switch_solution/bitfiles/
drop_nth_switch.bit
```

Loading the driver

- **Compile SUME driver:**
  - cd ~/NetFPGA-SUME-alpha/lib/sw/std/driver/sume_riffa_v1_0_0
  - make
  - make install
  - modprobe sume

- **Must reset the computer after programming the FPGA**
  - For proper detection and enumeration of PCIe
- **If you already had a running board**
  - cd $SUME_FOLDER/tools/scripts/reconfigure
  - source pci_rescan_run.sh
  - rescans the pcie bus (does not always succeed)