NetFPGA Summer Course

Presented by:

Andrew W Moore, Noa Zilberman, Gianni Antichi
Stephen Ibanez, Marcin Wojcik, Jong Hun Han,
Salvator Galea, Murali Ramanujam, Jingyun Zhang,
Yuta Tokusashi

University of Cambridge

July 24 – July 28, 2017

http://NetFPGA.org

Day 1 Outline

- **The NetFPGA platform**
  - Introduction
  - Overview of the NetFPGA Platform
- **NetFPGA SUME**
  - Hardware overview
- **Network Review**
  - Basic IP review
- **The Base Reference Switch**
  - Example I: Reference Switch running on the NetFPGA
- **The Life of a Packet Through the NetFPGA**
  - Hardware Datapath
  - Interface to software: Exceptions and Host I/O
- **Infrastructure**
  - Tree
  - Verification Infrastructure
- **Examples of Using NetFPGA**
- **Example Project: Crypto Switch**
  - Introduction to a Crypto Switch
  - What is an IP core?
  - Getting started with a new project.
  - Crypto FSM
- **Simulation and Debug**
  - Write and Run Simulations for Crypto Switch
- **Concluding Remarks**

Section V: 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-live

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)**

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)

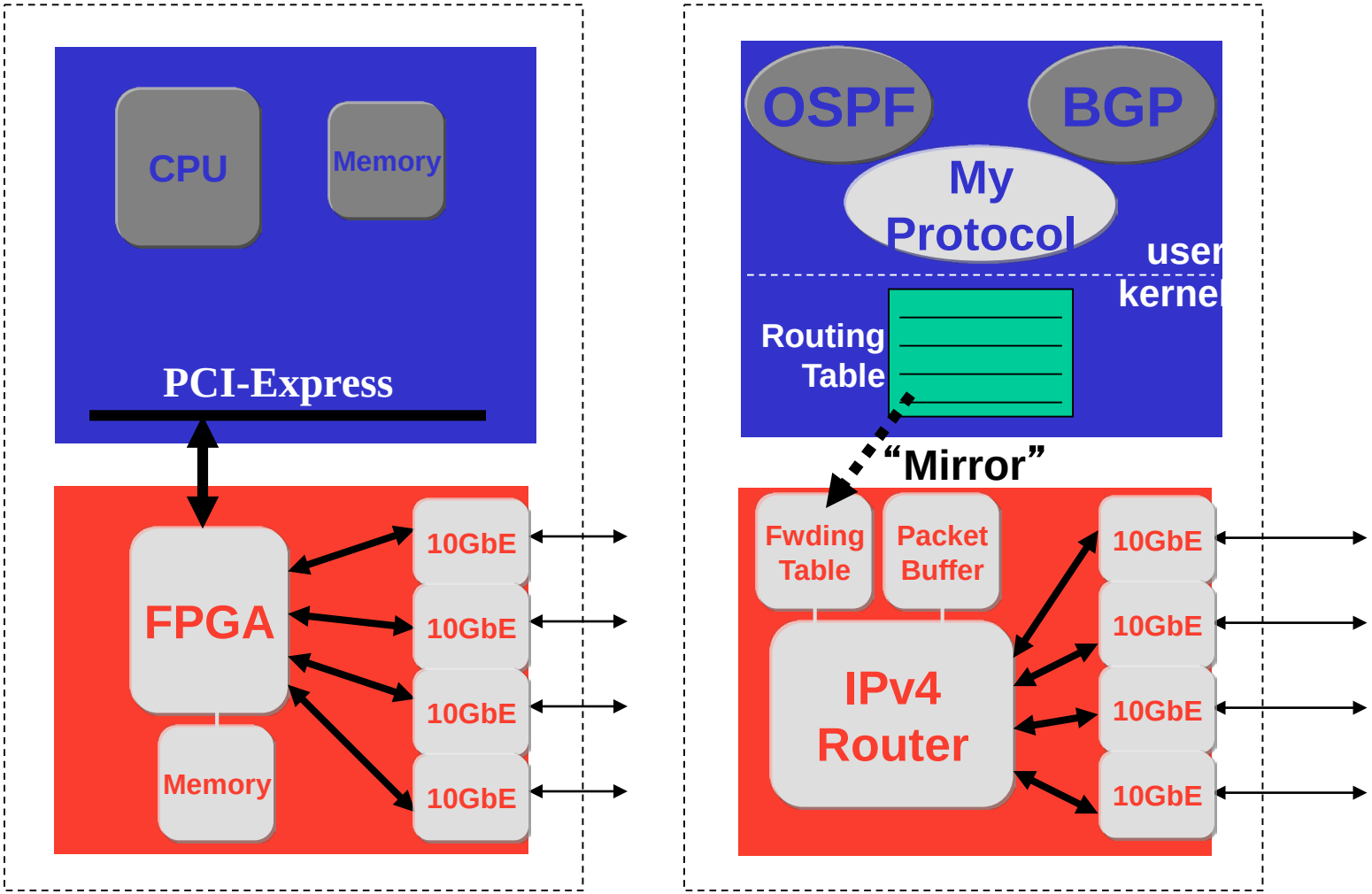
- **Register system**
  - collects and generates addresses for all the registers and memories in a project
  - uses integrated python and tcl scripts to generate HDL code (for hw) and header files (for sw)

Section VI: Examples of using NetFPGA

Usage #1

Running the Reference Router

User-space development, 4x10GE line-rate forwarding

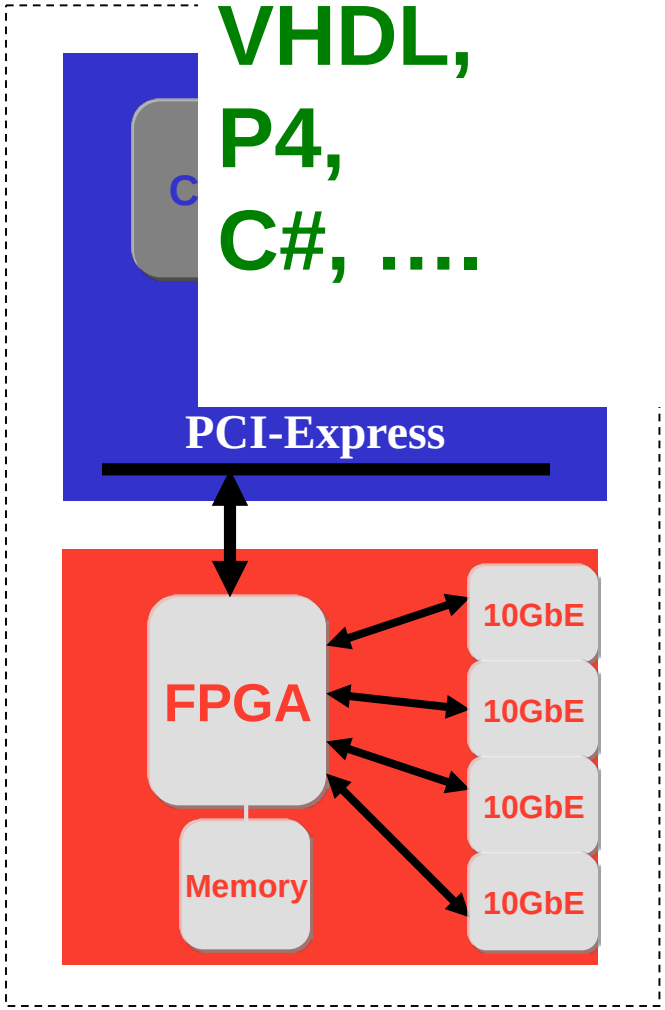


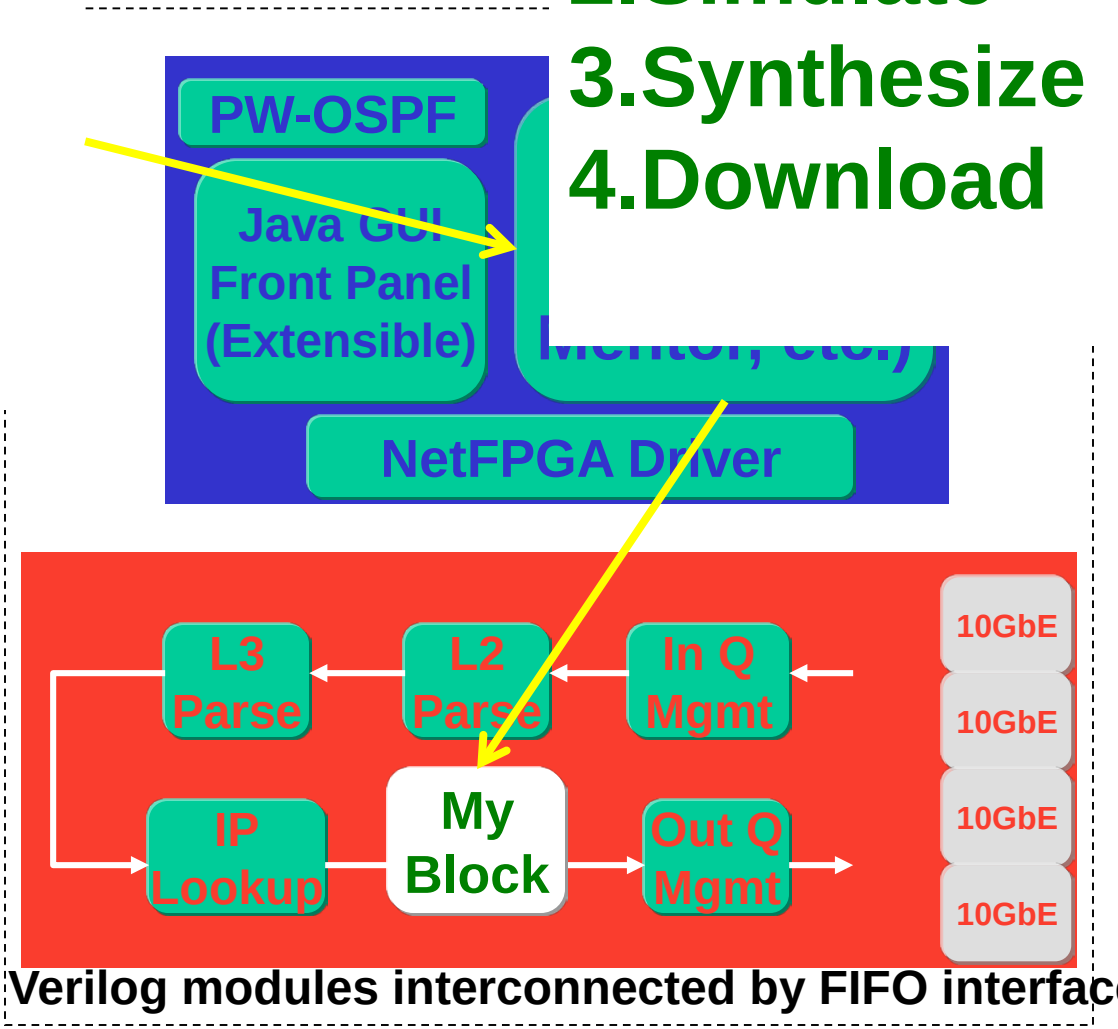
Usage #2

Enhancing Modular Reference Designs

Verilog,
VHDL,
P4,
C#, ...

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



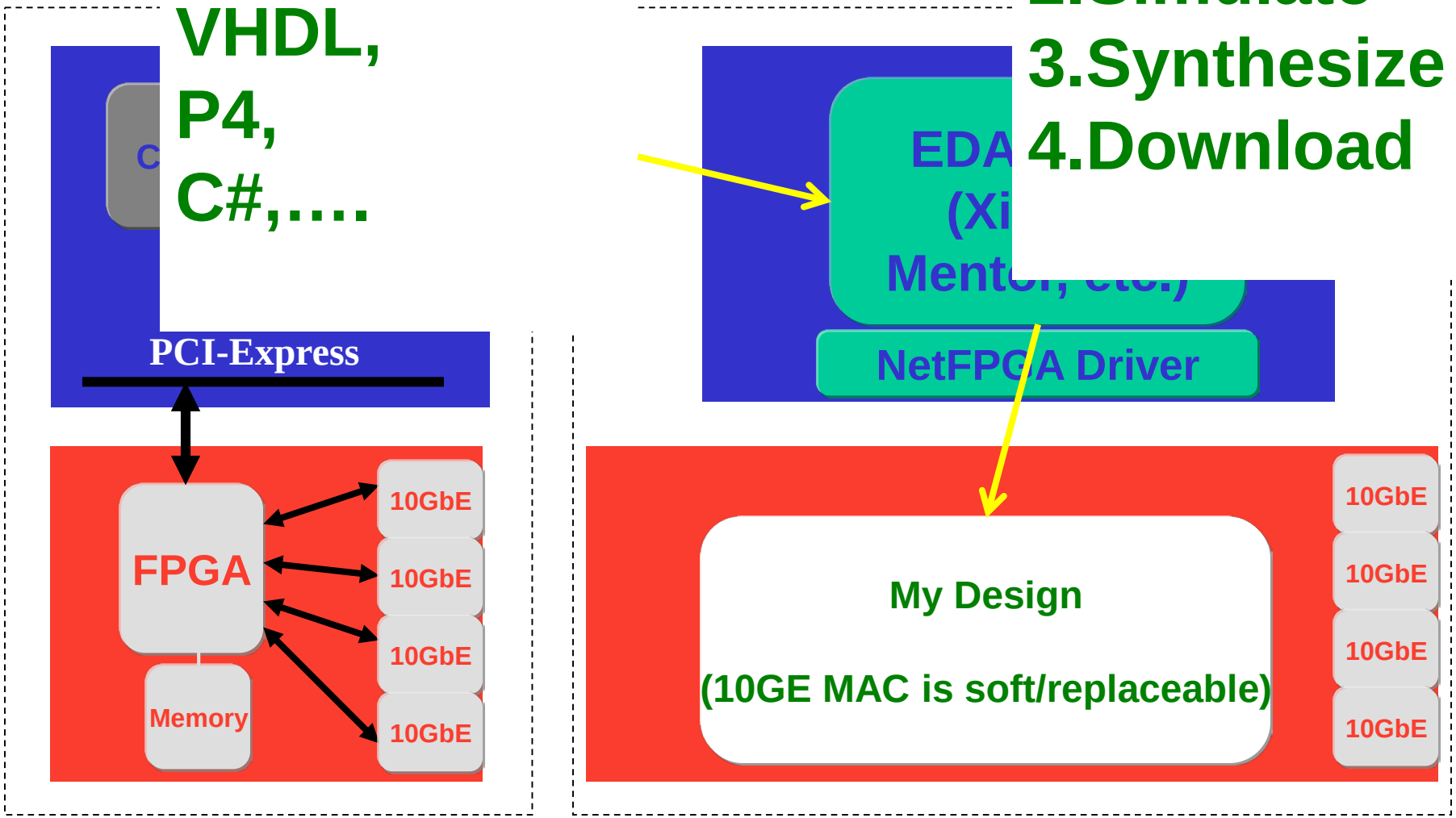


Verilog modules interconnected by FIFO interface

Usage #3

Creating new systems

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



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 / SUME	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
	NetSoC	UCAM/SRI International
	OSNT	UCAM/Stanford/GTech/CNRS

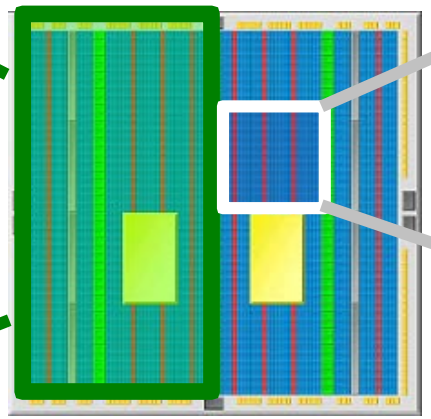
OpenFlow

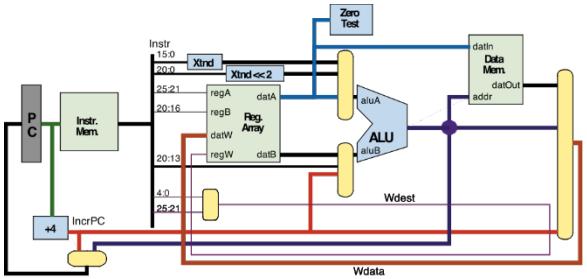
- **The most prominent NetFPGA success**
- **Has reignited the Software Defined Networking movement**
- **NetFPGA enabled OpenFlow**
  - A widely available open-source development platform
  - Capable of line-rate and
- **Was, until its commercial uptake, the reference platform for OpenFlow.**

NetSoC: Soft Processors in FPGAs

DDR controller

Ethernet MAC



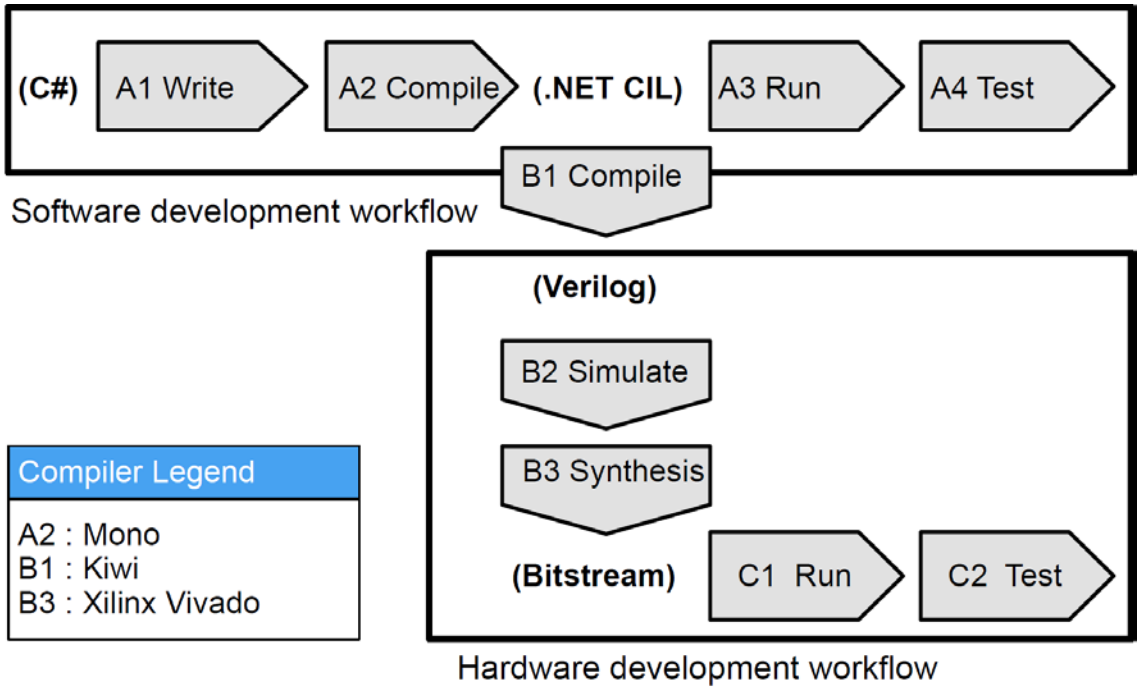


Processor(s)

- 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
- RISC-V, Linux OS

Emu : Accelerating Network Services

- Compiling .Net programs
  - To x86
  - To simulation environment
  - To multiple FPGA targets



How might we use NetFPGA?

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
  - Fibre 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
- Cluster
- 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 powerr
- 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 rout
- h/w openflow and (simple) NOX controller in one…
- Network RAMP (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 fo
- Prototype platform for NON-Ethernet or near-Ethernet MACs
  - Optical LAN (no buffers)

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 powerr
- 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 rout
- h/w openflow and (simple) NOX controller in one…
- Network RAID (multicast TCP with redundancy)
- inline compression
- hardware accelorator 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 fo
- Prototype platform for NON-Ethernet or near-Ethernet MACs
  - Optical LAN (no buffers)

Section VII: Example Project: Crypto Switch

Project: Cryptographic Switch

Implement a learning switch that encrypts upon transmission and decrypts upon reception

Cryptography

XOR function

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

XORing a value with itself *always* yields 0

XOR written as: ^ ⊻ ⊕

XOR is *commutative:* (A ^ B) ^ C = A ^ (B ^ C)

Cryptography (cont.)

Simple cryptography:

- Generate a secret key
- Encrypt the message by XORing the message and key
- Decrypt the ciphertext by XORing with the key

Explanation:

$$
\begin{aligned}
(M \wedge K) \wedge K &= M \wedge (K \wedge K) \quad \leftarrow \text{Commutativity} \\
&= M \wedge 0 \quad \leftarrow A \wedge A = 0 \\
&= M
\end{aligned}
$$

Cryptography (cont.)

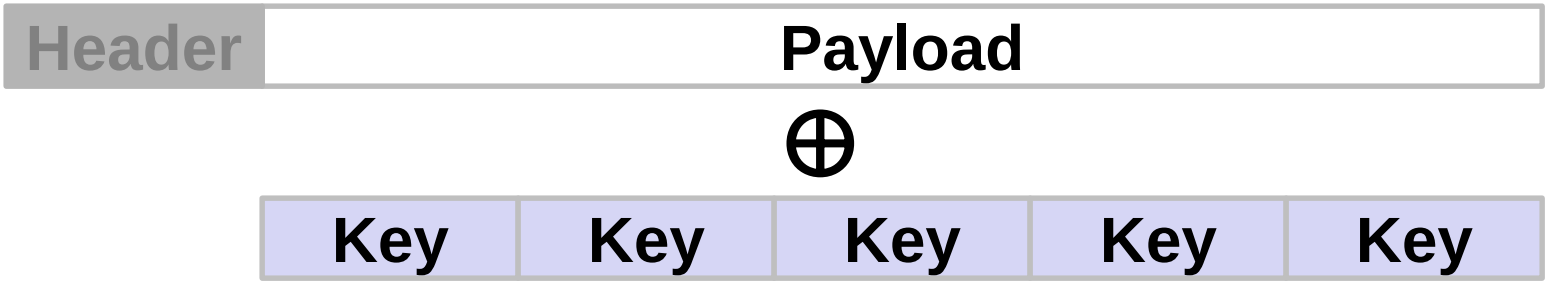
Example:

Message:	**00111011**
Key:	**10110001**
Message ^ Key:	**10001010**
Key:	**10110001**
Message ^ Key ^ Key:	**00111011**

Cryptography (cont.)

Idea: Implement simple cryptography using XOR

- 32-bit key
- Encrypt every word in payload with key



Note: XORing with a one-time pad of the same length of the message is secure/uncrackable. See: http://en.wikipedia.org/wiki/One-time_pad

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

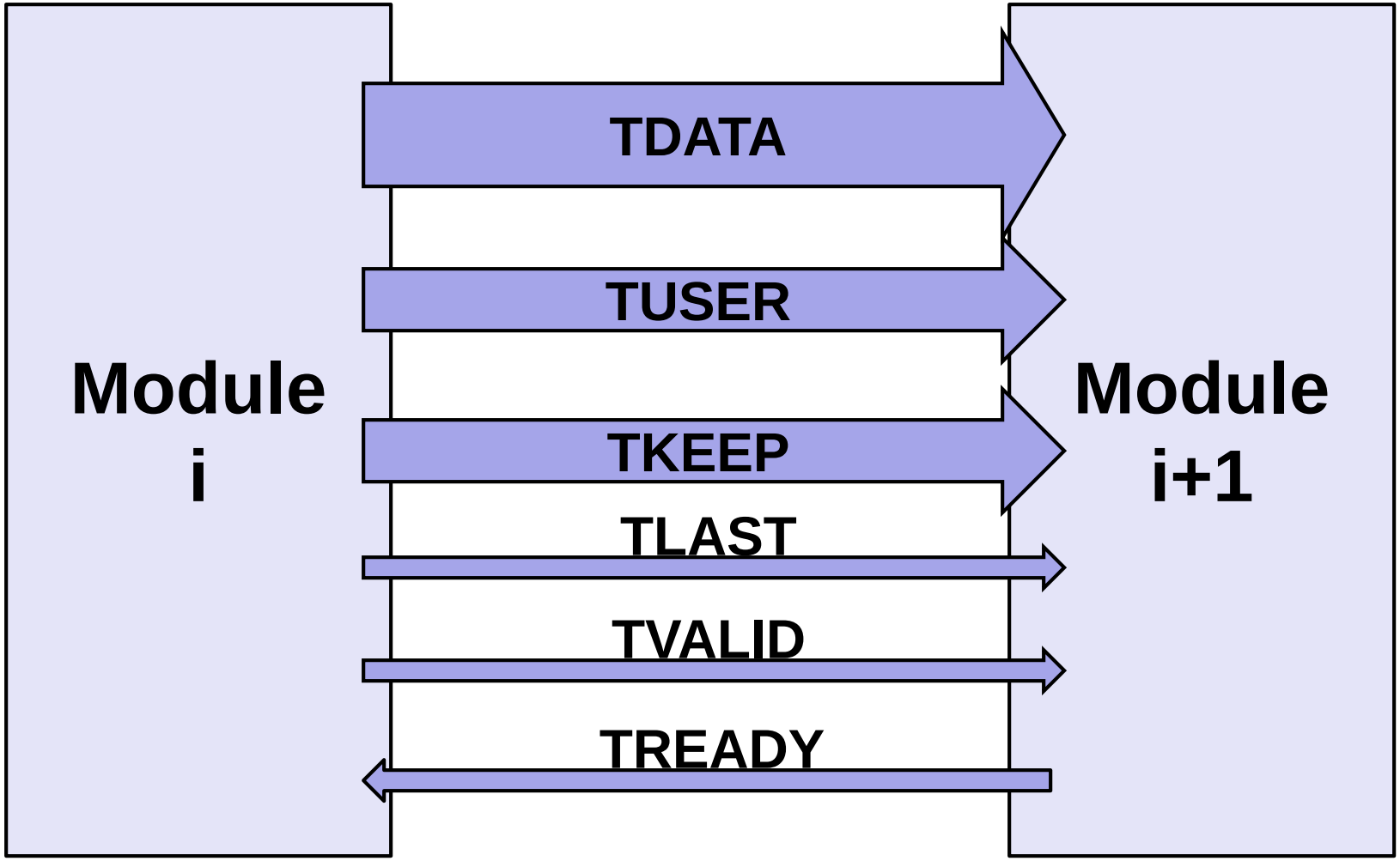
Scripts (TCL)

- **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

AXI4-Stream	Description
TDATA	Data Stream
TKEEP	Marks qualified 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

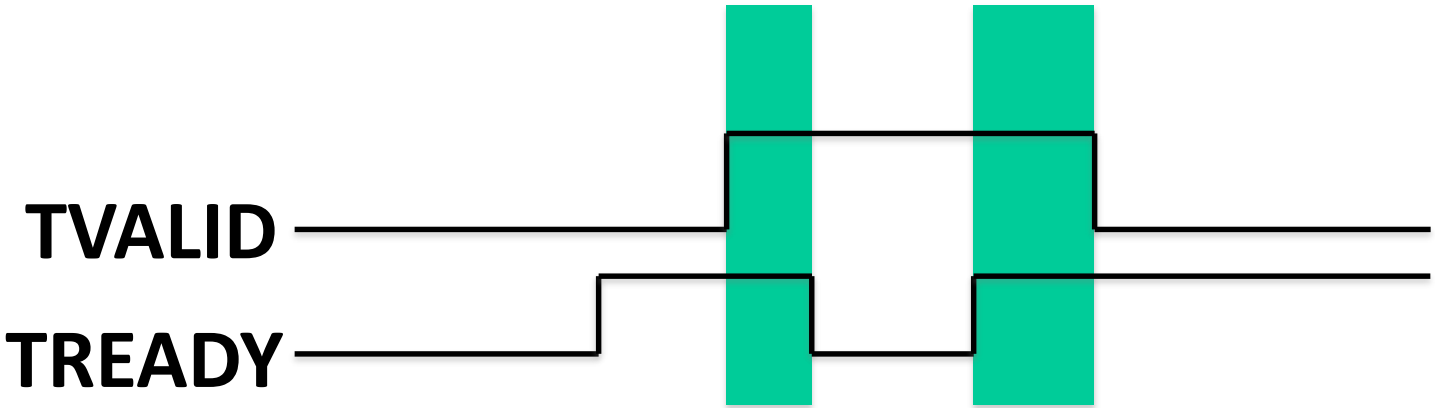
TLAST	TUSER	TKEEP	TDATA
0	V	0xFF…F	Eth Hdr
0	X	0xFF…F	IP Hdr
0	X	0xFF…F	…
1	X	0x0…1F	Last word

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 ***<u>not</u>*** 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]

Getting started with a new 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)

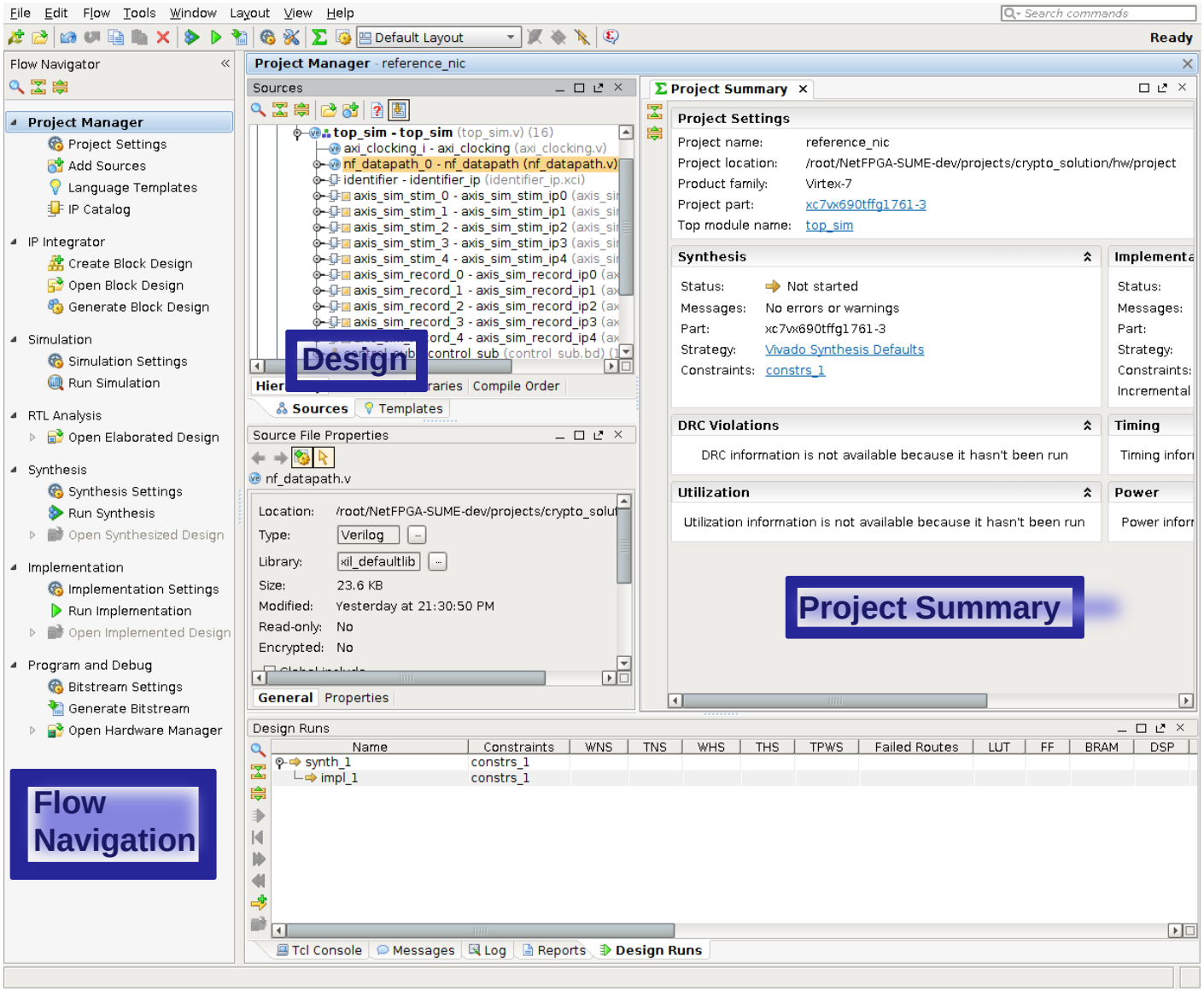
- **To invoke Vivado design tool, run:**

```
# 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/Xilinx/Vivado/2016.4/settings64.sh
  - vivado hw/project/<project name>.xpr

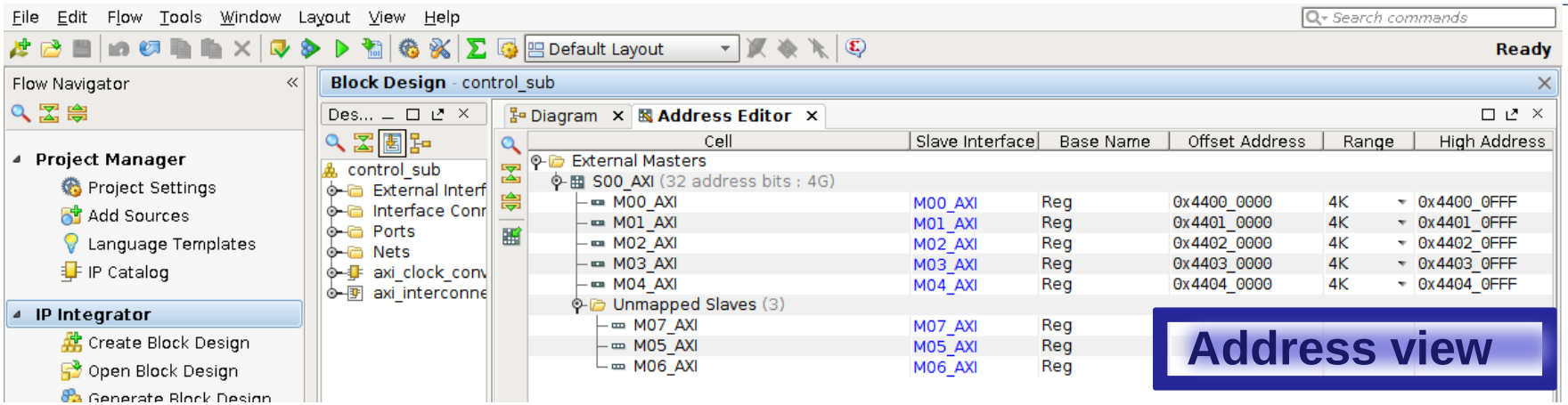
Vivado Design Tool (1)



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**

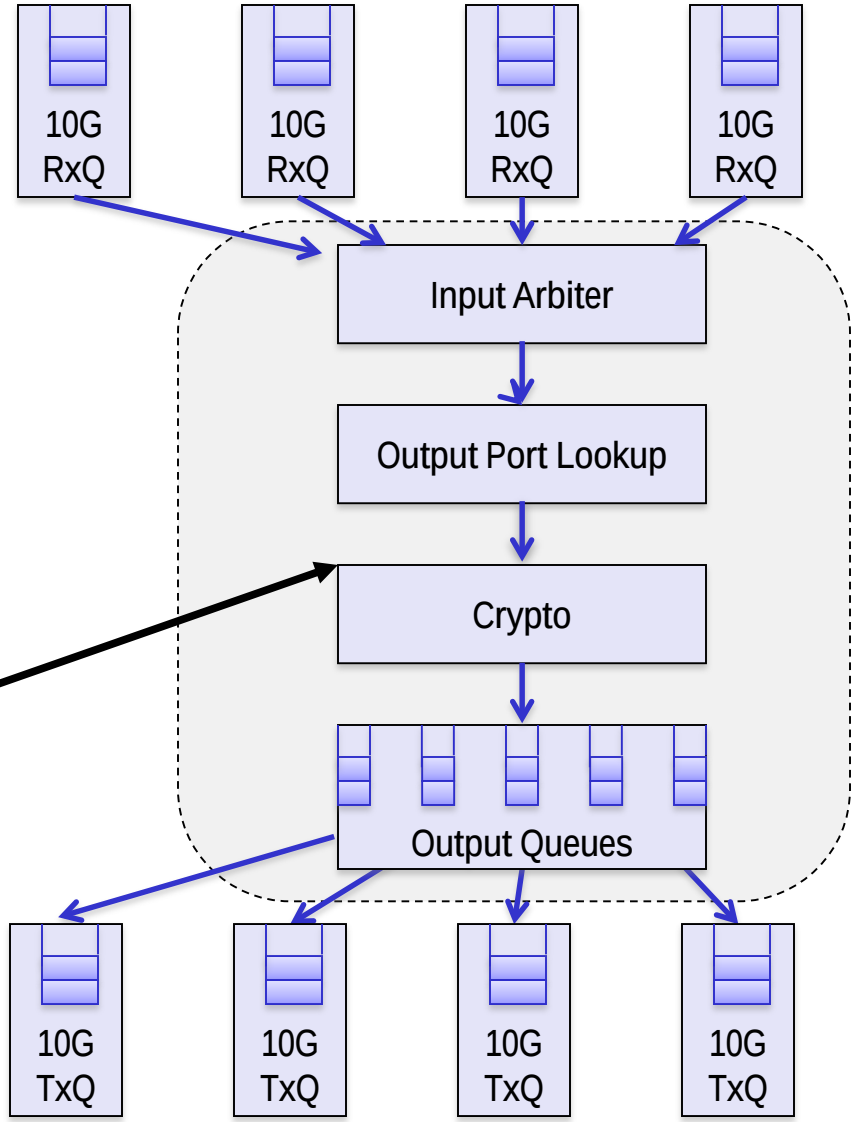
Getting started with a new project (1)

- **Projects:**
  - 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
    - Today:
      - pre-created project (crypto_switch)
  - Consists of:
    - Verilog source
    - Simulation tests
    - Hardware tests
    - Optional software

Getting started with a new project (3)

Typically implement functionality in one or more modules under the top wrapper

Crypto module to encrypt and decrypt packets



Getting started with a new project (4)

– Shared modules included from netfpga/lib/hw
  - Generic modules that are re-used in multiple projects
  - Specify shared modules in project's tcl file

– crypto_switch:

Local	Shared
crypto	Everything else

Getting started with a new project (5)

We already created the core for you:

1. cd $NF_DESIGN_DIR/hw/local_ip/crypto_v1_0_0
2. Write and edit files under crypto_v1_0_0/hdl Folder
   hint: TODO indicates where you should add your code
3. cd $NF_DESIGN_DIR/hw/local_ip/crypto_v1_0_0
4. make

<u>Notes:</u>

1. review ~/NetFPGA-SUME-live/tools/settings.sh
2. make sure NF_PROJECT_NAME=crypto_switch
3. If you make changes: source ~/NetFPGA-SUME-live/tools/settings.sh
4. Check that make passes without errors

crypto.v

```
Module crypto
   #(
      parameter C_M_AXIS_DATA_WIDTH = 256,
      parameter C_S_AXIS_DATA_WIDTH = 256,
       ...)
   (
      ...
   )

 //------------------------ regs/wires ----------------------------
   ...
   //------------------------ modules -------------------------------
   ...
   //------------------------ logic -------------------------------
   ...

endmodule
```

Module port declaration

crypto.v (2)

```
//------------------------- Modules------
```

Packet data dumped in a FIFO. Allows some "decoupling" between input and output.

```
  fallthrough_small_fifo #(
     .WIDTH(...),
     .MAX_DEPTH_BITS(2)
  ) input_fifo (
     .din        ({fifo_out_tlast, fifo_out_tuser,..}), // Data in
     .wr_en      (s_axis_tvalid & s_axis_tready),  // Write enable
     .rd_en      (in_fifo_rd_en),        // Read the next word
     .dout       ({s_axis_tlast, s_axis_tuser, ..}),
     .full       (),
     .nearly_full(in_fifo_nearly_full),
     .prog_full  (),
     .empty      (in_fifo_empty),
     .reset      (!axi_aresetn),
     .clk        (axi_aclk)
  );
```

crypto.v (3)

```
//------------------------- Logic-------------------------------

assign s_axis_tready = !in_fifo_nearly_full;
assign m_axis_tuser = fifo_out_tuser;
...

always @(*) begin
     // Default value
     in_fifo_rd_en = 0;

     if (m_axis_tready && !in_fifo_empty) begin
        in_fifo_rd_en = 1;
     end
  end
```

Combinational logic to read data from the FIFO. (Data is output to output ports.)

You'll want to add your state in this section.

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**

Project Integration

- **Edit the TCL file which generates the project:**
- **vi $NF_DESIGN_DIR/hw/tcl/ crypto_switch_sim.tcl**
- **Add the following lines (line 96):**

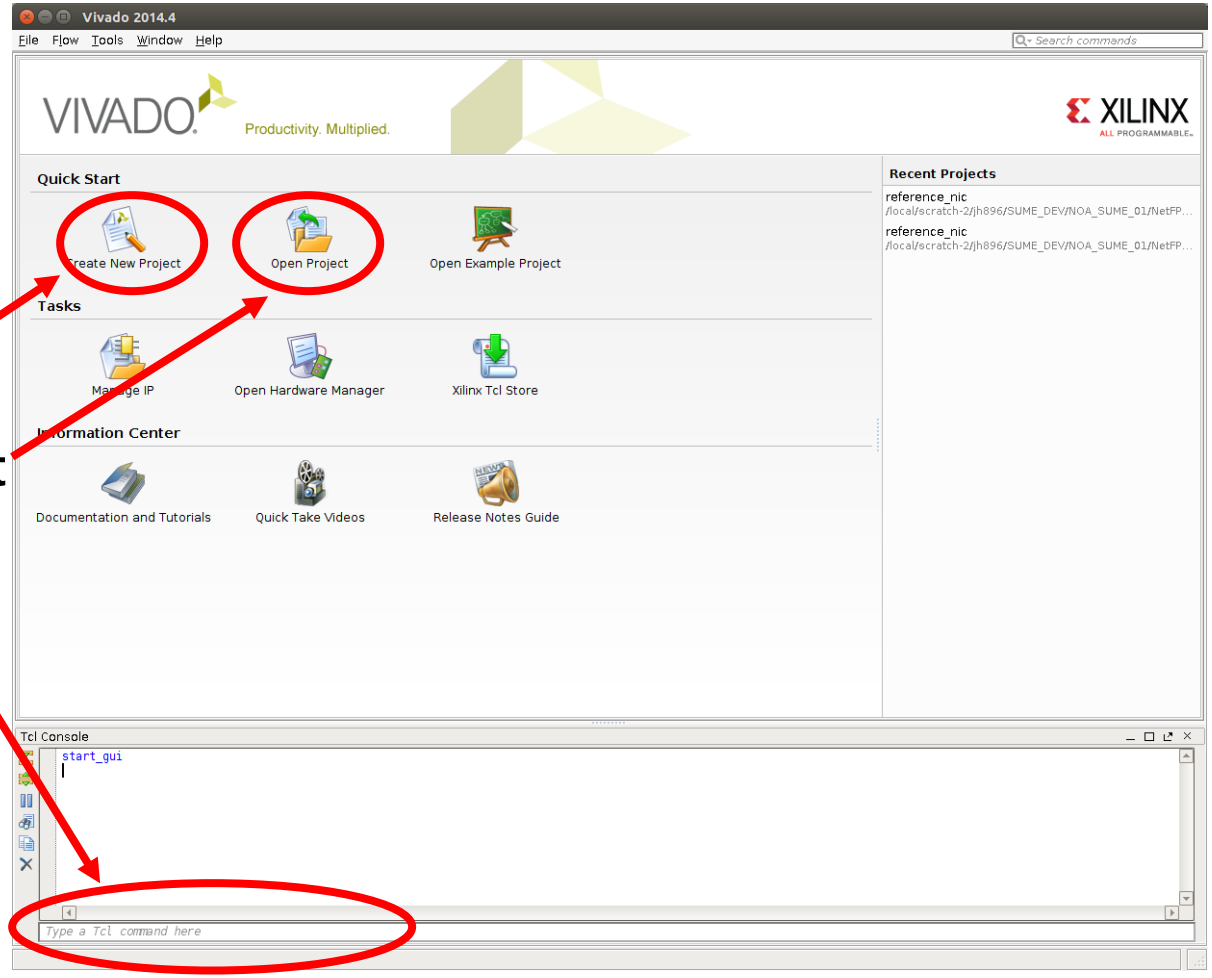
```
create_ip -name crypto -vendor NetFPGA -library NetFPGA -module_name crypto_ip
set_property generate_synth_checkpoint false [get_files crypto_ip.xci]
reset_target all [get_ips crypto_ip]
generate_target all [get_ips crypto_ip]
```

- **name and module_name should match your hdl**
- **Save time for later, add the same text also in:**

  $NF_DESIGN_DIR/tcl/crypto_switch.tcl (line 98)

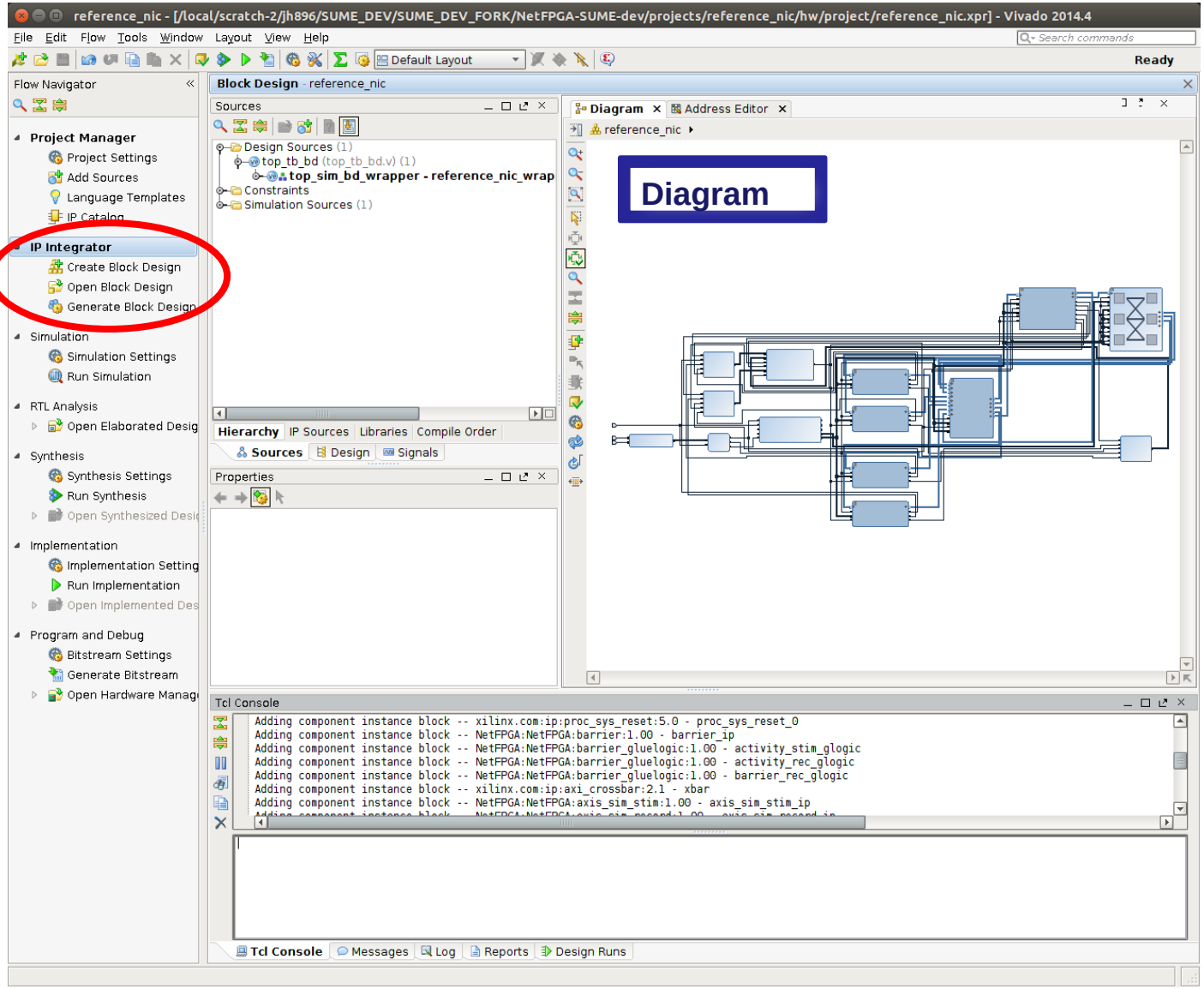
Project Integration – Block Design

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

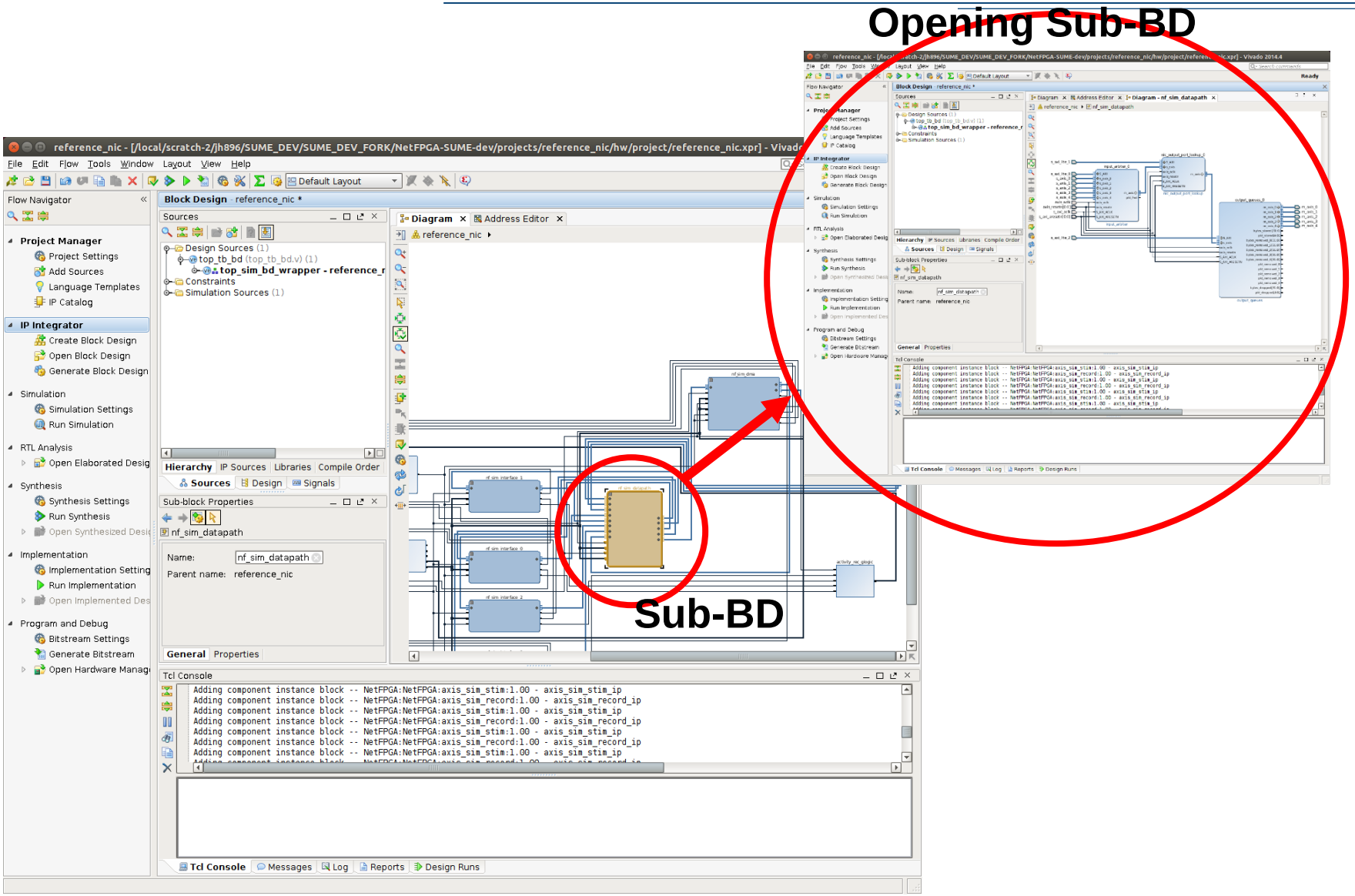


Project Integration – Block Design (2)

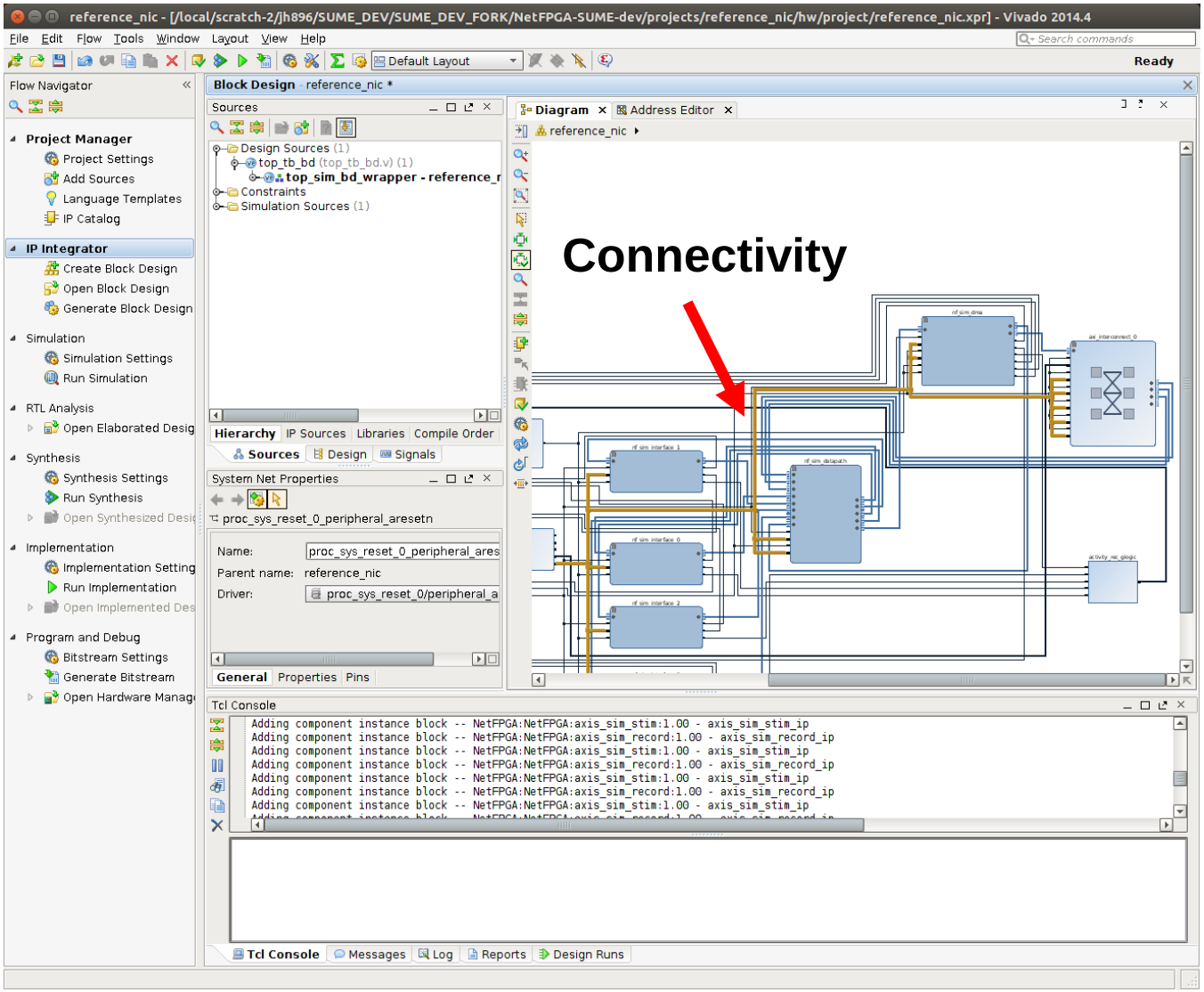
Open block design



Project Integration – Block Design (3)

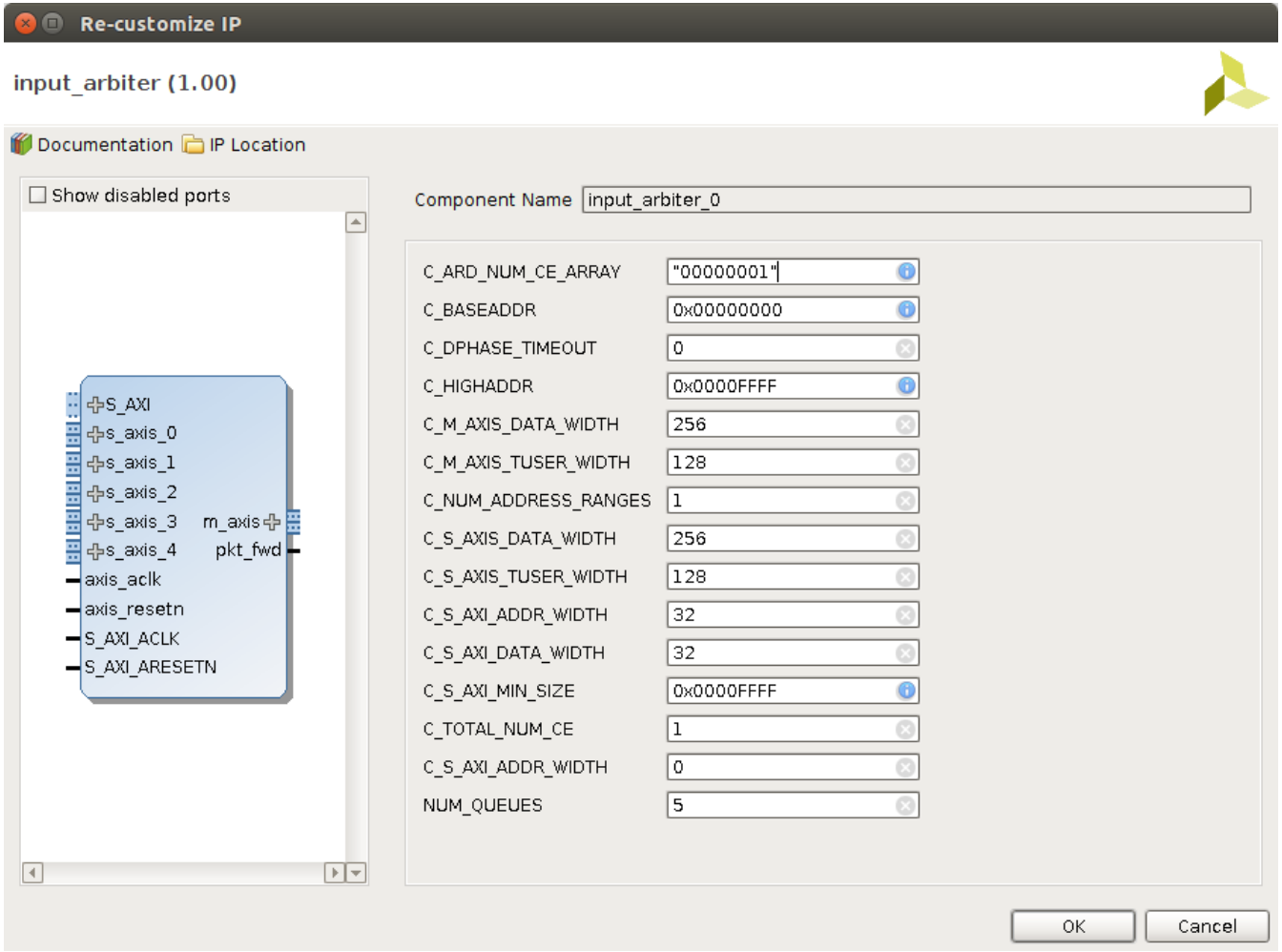


Project Integration – Block Design (4)



Project Integration – Block Design (5)

Setting module parameters



Project Integration – Block Design (6)

reference_nic - [/local/scratch-2/jh896/SUME_DEV/NOA_SUME_01/NetFPGA-SUME-dev/projects/reference_nic/hw/project/reference_nic.xpr] - Vivado 2014.4

File Edit Flow Tools Window Layout View Help

Default Layout — write_bitstream Out-of-date — more info

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 Desig
- Synthesis
  - Synthesis Settings
  - Run Synthesis
  - Open Synthesized Desi
- Implementation
  - Implementation Setting
  - Run Implementation
  - Open Implemented Des
- Program and Debug
  - Bitstream Settings
  - Generate Bitstream
  - Open Hardware Manag

Block Design - reference_nic

Design

- reference_nic
  - External Interfaces
  - Interface Connections
  - Ports
  - Nets
  - axi_iic_0 (AXI IIC:2.0)
  - axi_interconnect_0
  - axi_uartlite_0 (AXI Uartl
  - input_arbiter_0 (input_
  - mbsys
  - nf_10g_interface_0 (nf_
  - nf_10g_interface_1 (nf_
  - nf_10g_interface_2 (nf_
  - nf_10g_interface_3 (nf_
  - nf_sume_dma
  - nic_output_port_lookup
  - output_queues_0 (outp
  - proc_sys_reset_0 (Proc

Block Properties

input_arbiter_0

Name: input_arbiter_

Parent name: reference_nic

General Properties IP

Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Addre
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

Address Editor

Offset **Range**

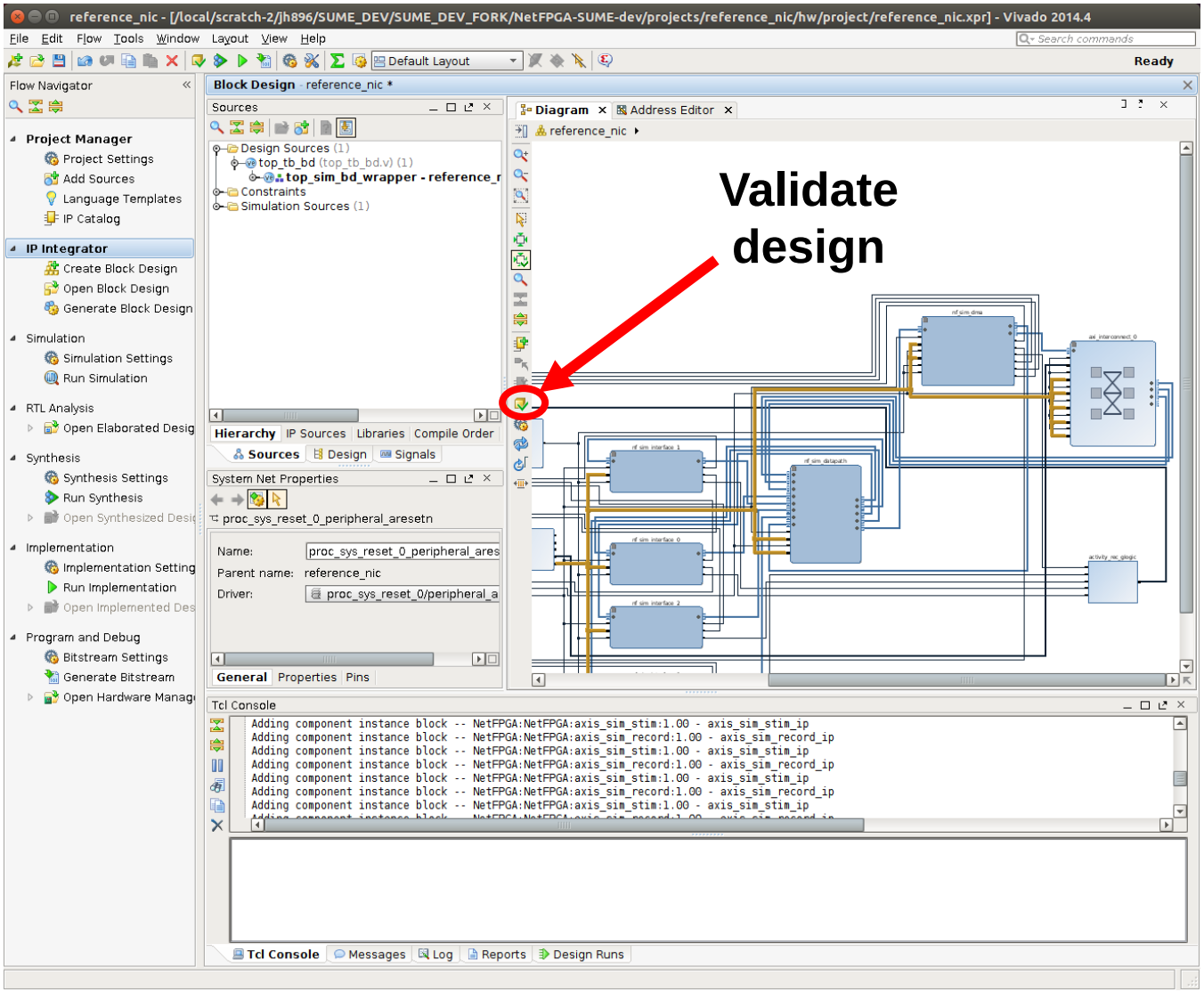
Tcl Console

```
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 ;
```

Type a Tcl command here

Tcl Console | Messages | Log | Reports | Design Runs

Project Integration – Block Design (7)



Summary to this Point

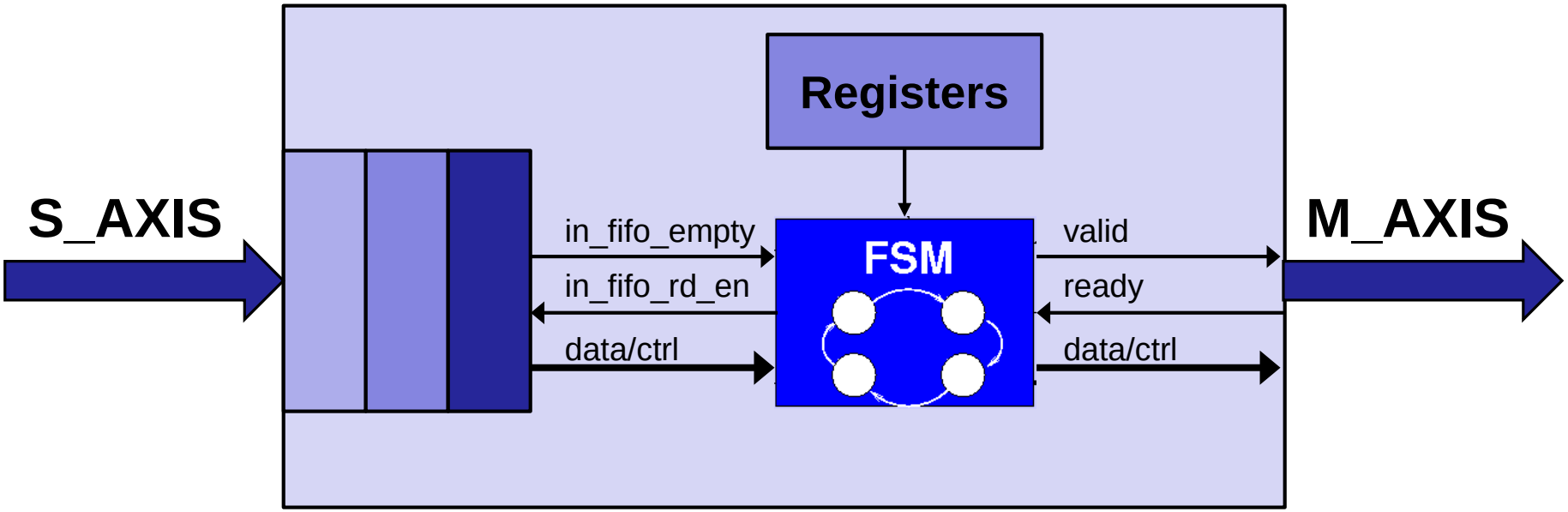
- **Created a new project**
- **Created a new core named crypto**
- **Wired the new core into the pipline**
  - After output_port_lookup
  - Before output_queues
- **Next we will write the Verilog code!**

Implementing the Crypto Module (1)

- **What do we want to encrypt?**
  - IP payload only
    - Plaintext IP header allows routing
    - Content is hidden
  - Encrypt bytes 35 onward
    - Bytes 1-14 – Ethernet header
    - Bytes 15-34 – IPv4 header (assume no options)
    - Remember AXI byte ordering
  - For simplicity, assume all packets are IPv4 without options

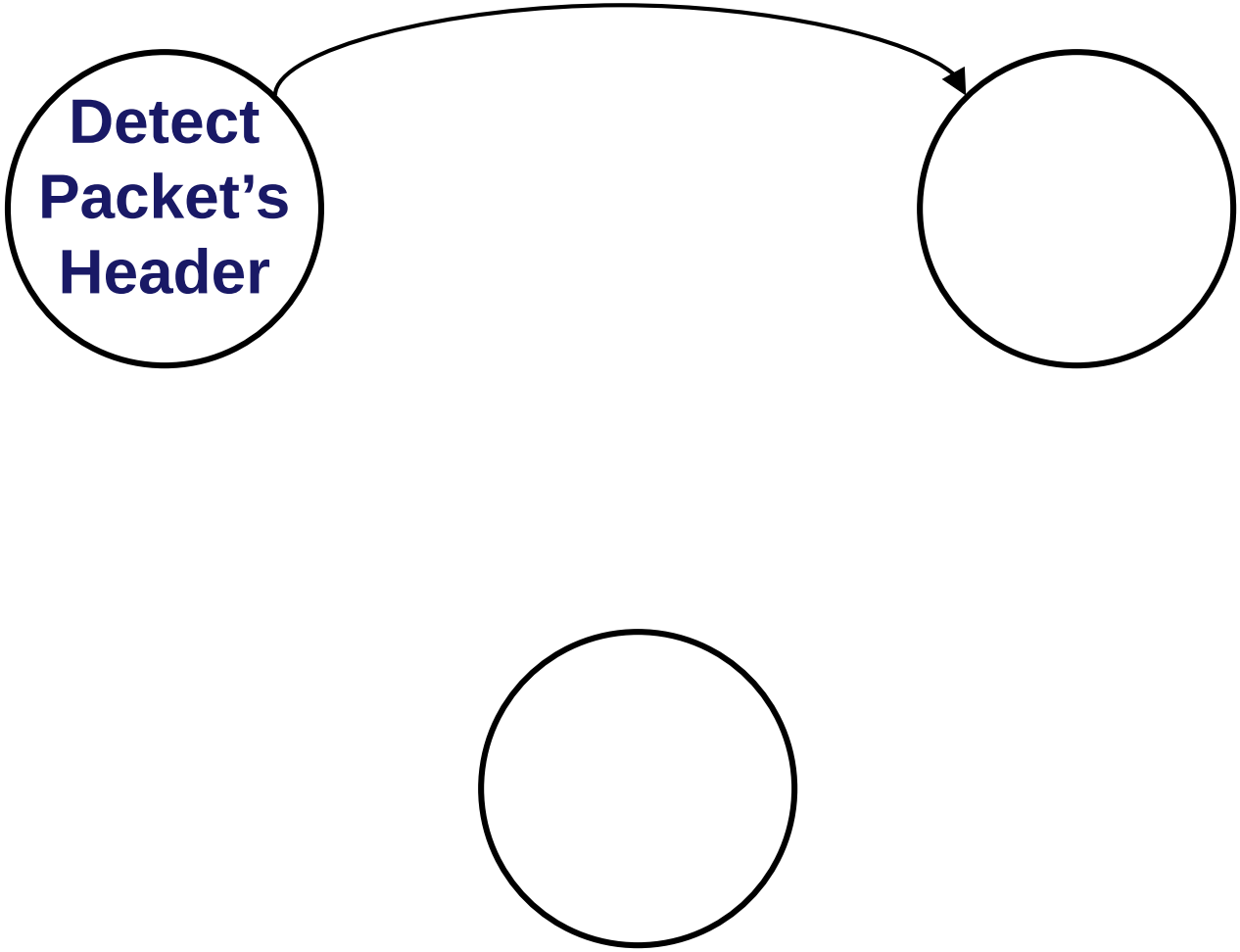
Implementing the Crypto Module (2)

- **State machine (shown next):**
  - Module headers on each packet
  - Datapath 256-bits wide
    - 34 / 32 is not an integer! ☹
- **Inside the crypto module**



Crypto Module State Diagram

Hint: We suggest 3 states



Implementing the Crypto Module (3)

Implement your state machine inside crypto.v

Suggested sequence of steps:

1. Set the key value
   - set the key = 32'hffffffff;
2. Write your state machine to modify the packet by XORing the key and the payload
   - Use eight copies of the key to create a 256-bit value to XOR with data words
3. Do not pay attention to the register infrastructure that will be explained later.

Afraid of Verilog? Start with easy_crypto.v

More Verilog: Assignments 1

- **Continuous assignments**
  - appear *outside* processes (`always @` blocks):

```
assign foo = baz & bar;
```

  - targets must be declared as `wires`
  - always "happening" (*ie*, are concurrent)

More Verilog: Assignments 2

- **Non-blocking assignments**
  - appear *inside* processes (`always @` blocks)
  - use only in *sequential* (clocked) processes:

```
always @(posedge clk) begin
    a <= b;
    b <= a;
end
```

  - occur in next *delta* ('moment' in simulation time)
  - targets must be declared as `regs`
  - never clock any process other than with a clock!

More Verilog: Assignments 3

- **Blocking assignments**
  - appear *inside* processes (`always @` blocks)
  - use only in *combinatorial* processes:
    - (combinatorial processes are much like continuous assignments)

```
always @(*) begin

    a = b;
    b = a;
end
```

  - occur one after the other (as in sequential langs like C)
  - targets must be declared as `regs` – even though not a register
  - never use in sequential (clocked) processes!

More Verilog: Assignments 3

- **Blocking assignments**
  - appear *inside* processes (`always @` blocks)
  - use only in *combinatorial* processes:
    - (combinatorial processes are much like continuous assignments)

```
always @(*) begin
        tmp = a;
        a = b;
        b = tmp;
end
```

unlike non-blocking, have to use a temporary signal

  - occur one after the other (as in sequential langs like C)
  - targets must be declared as `regs` – even though not a register
  - never use in sequential (clocked) processes!

(hints)

- **Never assign one signal from two processes:**

```
always @(posedge clk) begin
    foo <= bar;
end

always @(posedge clk) begin
    foo <= quux;
end
```

(hints)

- **In combinatorial processes:**
  - take great care to assign in all possible cases

```
always @(*) begin
    if (cond) begin
        foo = bar;
    end
end
```

  - (latches ‹as opposed to flip-flops› are bad for timing closure)

(hints)

- **In combinatorial processes:**
  - take great care to assign in all possible cases

```
always @(*) begin
    if (cond) begin
        foo = bar;
    else
        foo = quux;
    end
end
```

(hints)

- **In combinatorial processes:**
  - (or assign a default)

```
always @(*) begin
    foo = quux;

    if (cond) begin
        foo = bar;
    end
end
```

Getting started: step by step

Preparing the crypto module:

1. cd $NF_DESIGN_DIR/hw/local_ip/crypto_v1_0_0
2. Write and edit files under crypto_v1_0_0/hdl Folder
   hint: TODO indicates where you should add your code
3. cd $NF_DESIGN_DIR/hw/local_ip/crypto_v1_0_0
4. make

<u>Notes:</u>

1. review ~/NetFPGA-SUME-live/tools/settings.sh
2. make sure NF_PROJECT_NAME=crypto_switch
3. If you make changes: source ~/NetFPGA-SUME-live/tools/settings.sh
4. Check that make passes without errors

Project Integration: step by step

1. vi $NF_DESIGN_DIR/hw/nf_datapath.v
2. Add the new module between the output port lookup and output queues

   Afraid of Verilog? Start with easy_crypto.v

Edit the TCL file which generates the project:

1. vi $NF_DESIGN_DIR/hw/tcl/ crypto_switch_sim.tcl
2. Add the following lines (line 96):

   create_ip -name crypto -vendor NetFPGA -library NetFPGA -module_name crypto_ip
   set_property generate_synth_checkpoint false [get_files crypto_ip.xci]
   reset_target all [get_ips crypto_ip]
   generate_target all [get_ips crypto_ip]

3. name and module_name should match your hdl

4. Save time for later, add the same text also in:

   $NF_DESIGN_DIR/tcl/crypto_switch.tcl (line 98)

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 provide a unified infrastructure for both HW and simulation tests**

Testing: Simulation

- **We will simulate the "crypto_switch" design under the "simulation framework"**
- **We will show you how to**
  - 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 crypto_switch project

Testing: Simulation(2)

Run a simulation to verify changes:

1. make sure “NF_DESIGN_DIR” variable in the tools/settings.sh file located in ~/NetFPGA-SUME-live points to the crypto_switch project.
2. source ~/NetFPGA-SUME-live/tools/settings.sh
3. cd ~/NetFPGA-SUME-live/tools/scripts
4. ./nf_test.py sim --major crypto –minor test

   Or ./nf_test.py sim --major crypto –major test --gui (if you want to run the gui)

Now we can simulate the crypto functionality

Crypto Switch simulation

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

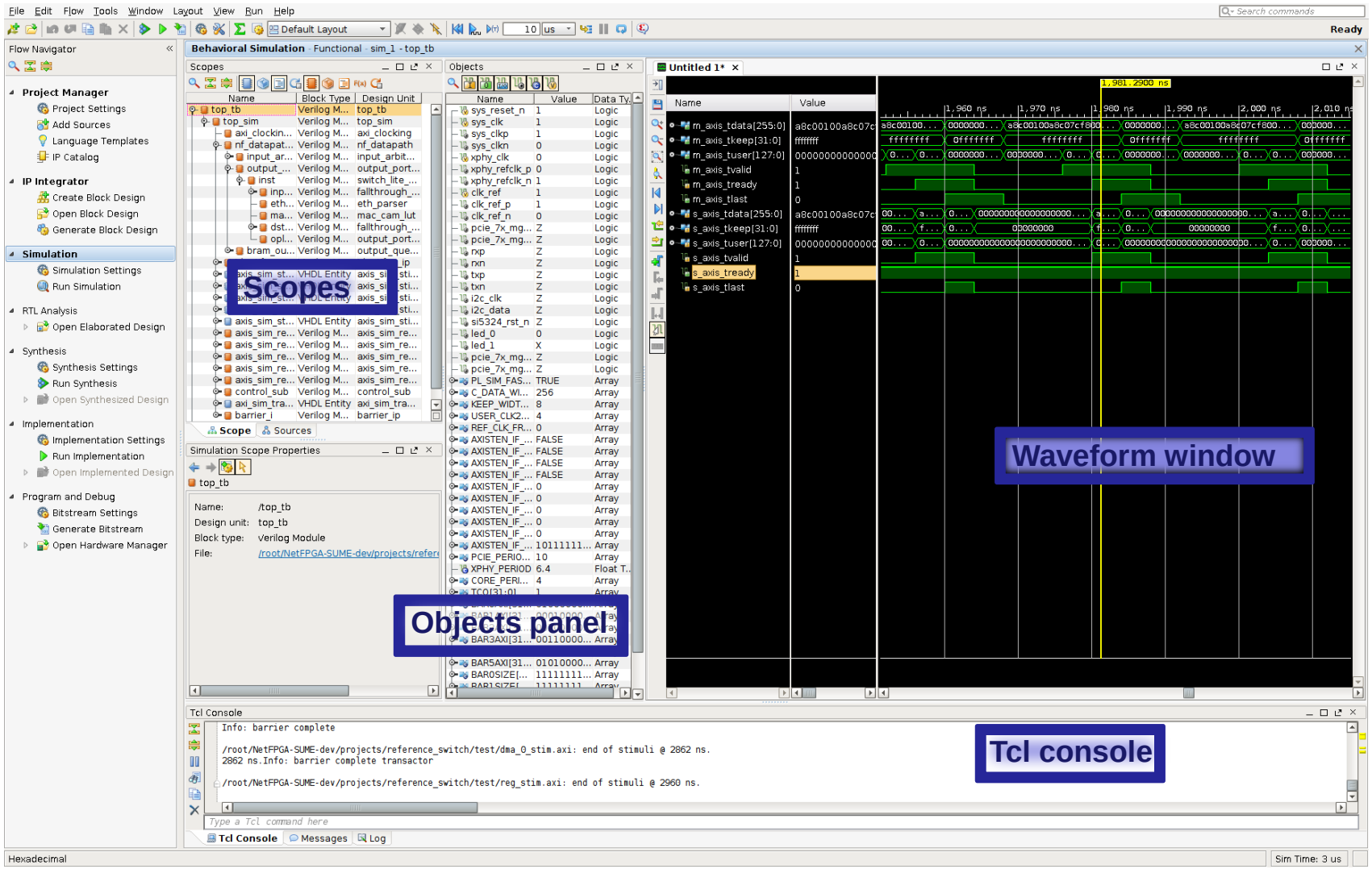
- The "**isHW**" statement enables the HW test (we will look into it tomorrow)
- Let's focus on the "**else**" part of the statement
- **make_IP_pkt** fuction creates the IP packet that will be used as stimuli
- **pkt.tuser_sport** is used to set up the correct source port of the packet
- **encrypt_pkt** 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)

4 The results are in…

```
loading libsume..
Reconciliation of nf_interface_2_log.axi with nf_interface_2_expected.axi
        PASS (10 packets expected, 10 packets received)

Reconciliation of nf_interface_3_log.axi with nf_interface_3_expected.axi
        PASS (10 packets expected, 10 packets received)

Reconciliation of nf_interface_0_log.axi with nf_interface_0_expected.axi
        PASS (10 packets expected, 10 packets received)

Reconciliation of nf_interface_1_log.axi with nf_interface_1_expected.axi
        PASS (10 packets expected, 10 packets received)

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

- **As expected, total of 10 packets are received on each interface**

Running simulation in xSim



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

make core going wild

CRITICAL WARNING: [filemgmt 20-742] The top module "crypto" specified for this project can not be validated. The current project is using automatic hierarchy update mode, and hence a new suitable replacement top will be automatically selected. If this is not desired, please change the hierarchy update mode to one of the manual compile order modes first, and then set top to any desired value.

…..

ERROR: [filemgmt 20-730] Could not find a top module in the fileset sources_1.

……

You've got syntax errors!!!

Start by checking ports and parameters syntax

Simulation gone wild

When "./nf_test.py sim ….."

1

source /opt/Xilinx/Vivado/2016.4/settings64.sh

2

Edit and source NetFPGA-SUME-live/tools/settings.sh

3

Run "make core" under projects/crypto_switch/hw/

4

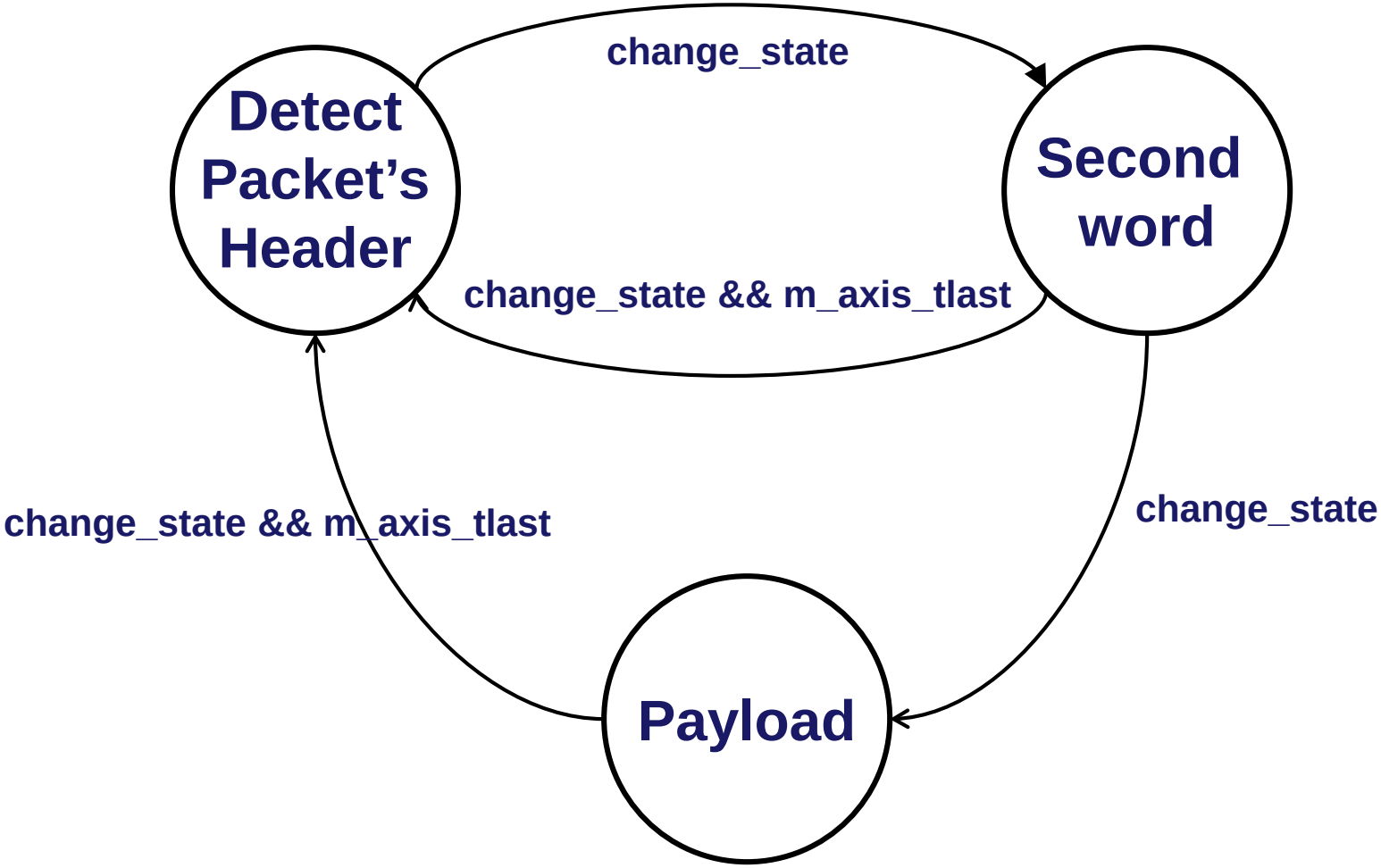
Check that crypto_switch.tcl, crypto_switch_sim.tcl, export_registers.tcl are all up to date with your changes

5

if sim finishes but complains that each test passes 10 packets but all tests FAIL – this means your static key is different between your code and your run.py file, check the log

Crypto Module State Diagram: Solution

change_state = m_axis_tvalid && m_axis_tready



it is time for the first synthesis!!!

Synthesis

- **To synthesize your project:**

```
cd $NF_DESIGN_DIR
make
```

Important Notes

- **Make sure to backup your work**
- **You can fork the course's repo to your user and push updates**
- **Careful – running simulation erases previous $NF_DESIGN_DIR/hw/project created by a synthesis**

Section IX: Conclusion

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, Salvator Galea, Marcin Wojcik, Diana Andreea Popescu, Murali Ramanujam

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, Stephen Gabriel Ibanez

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 , Andreas Fiessler, Robert Soule, Pietro Bressana, Yuta Tokusashi

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

EPSRC
Pioneering research and skills

NSF

XILINX

LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

Micron

CYPRESS
PERFORM

DIGILENT

Isaac Newton Trust

The Leverhulme Trust

BROADCOM

HUAWEI

Google

Atomic Rules

HGLOBAL HITECH

ALGO-LOGIC

imc