UNIVERSITY OF CAMBRIDGE

P51: High Performance Networking

Introduction to NetFPGA – Part II

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

Lent 2017/18

Project

Project

- Practical Sessions
- Today
  - Registers
  - Proposal and aims
- Next Week
  - Architecture
  - Dividing the effort
- Project test infrastructure

Infrastructure

Infrastructure

- Tree structure

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

Tree Structure (1)

NetFPGA-SUME-live

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

Tree Structure (2)

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

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)

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)

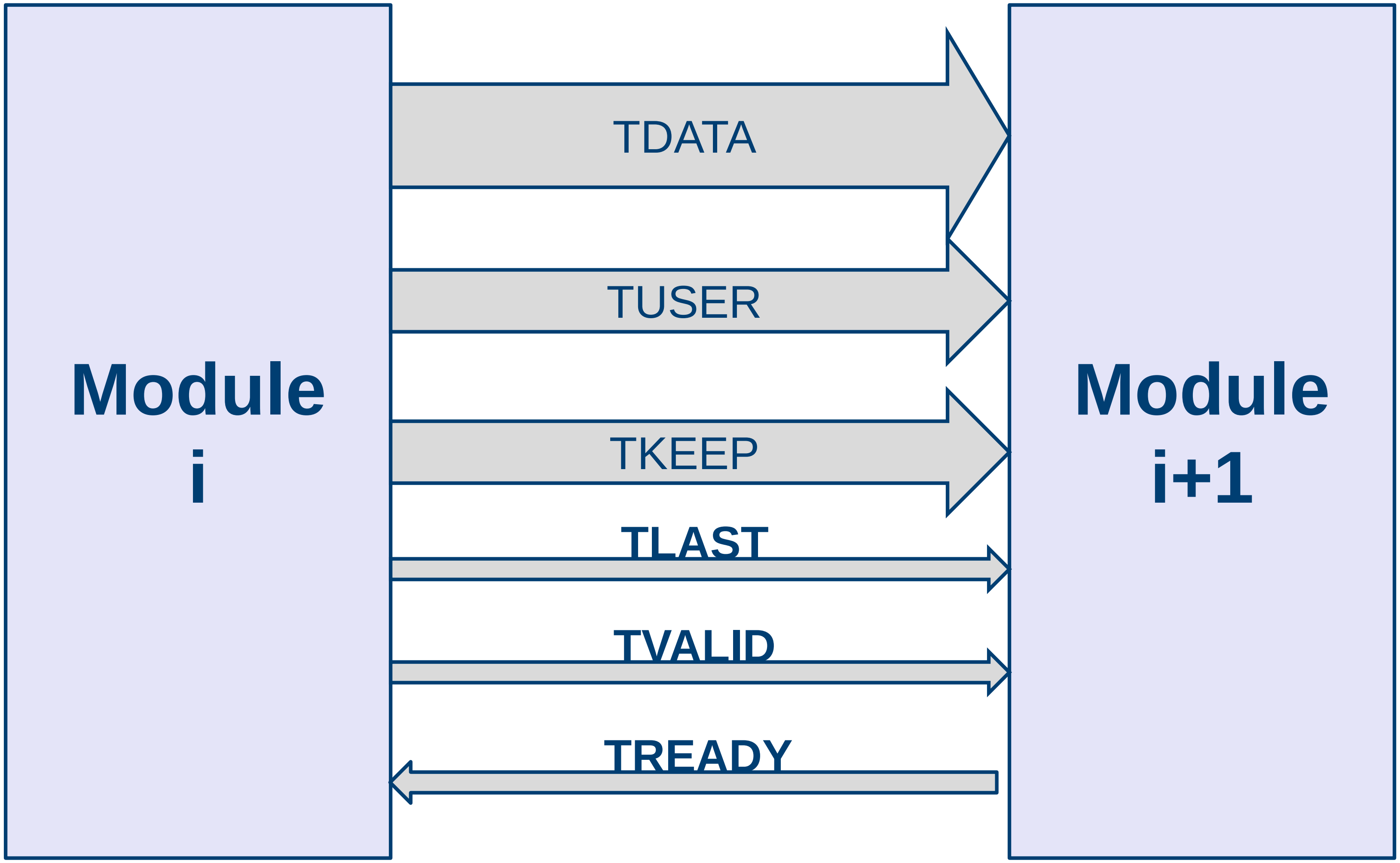
- NetFPGA cores
  - AXI-compliant
- AXI = Advanced eXtensible Interface
  - Used in ARM-based embedded systems
  - Standard interface
  - **AXI4/AXI4-Lite**: Control and status interface
  - **AXI4-Stream**: Data path interface
- Xilinx IPs and tool chains
  - Mostly AXI-compliant

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

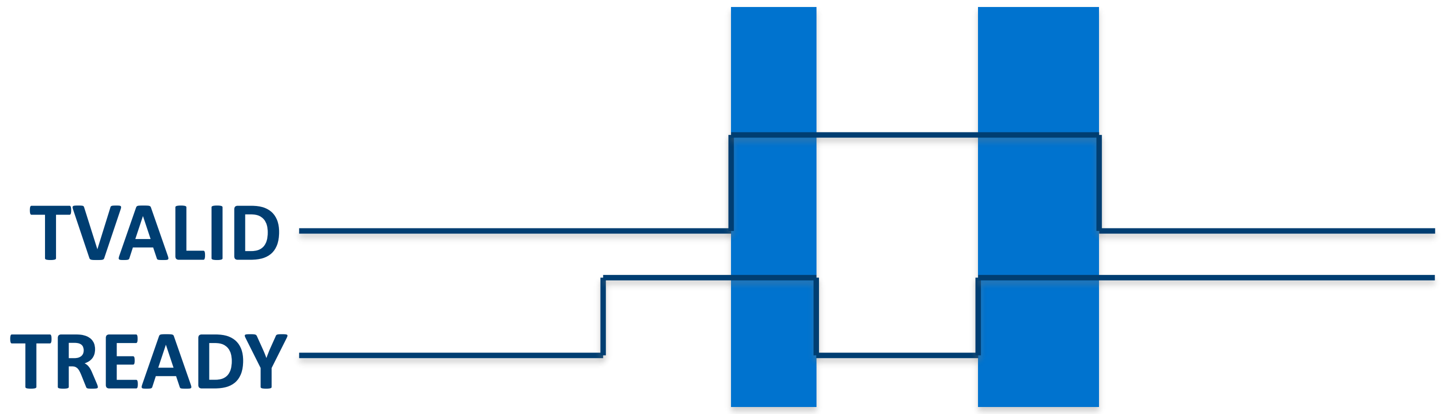
TLAST	TUSER	TKEEP	TDATA
0	V	0xFF…F	Eth Hdr
0	X	0xFF…F	IP Hdr
0	X	0xFF…F	…
1	X	0x0…1F	Last word

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 ***not*** depend on TREADY



Byte ordering

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

Developing a project

Embedded Development Kit

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

Xilinx Vivado

- A Vivado project consists of following:
  - **<project_name>.xpr**
    - top level Vivado project file
  - **tcl and HDL files that define the project**
  - **system.xdc**
    - user constraint file
    - defines constraints such as timing, area, IO placement etc.

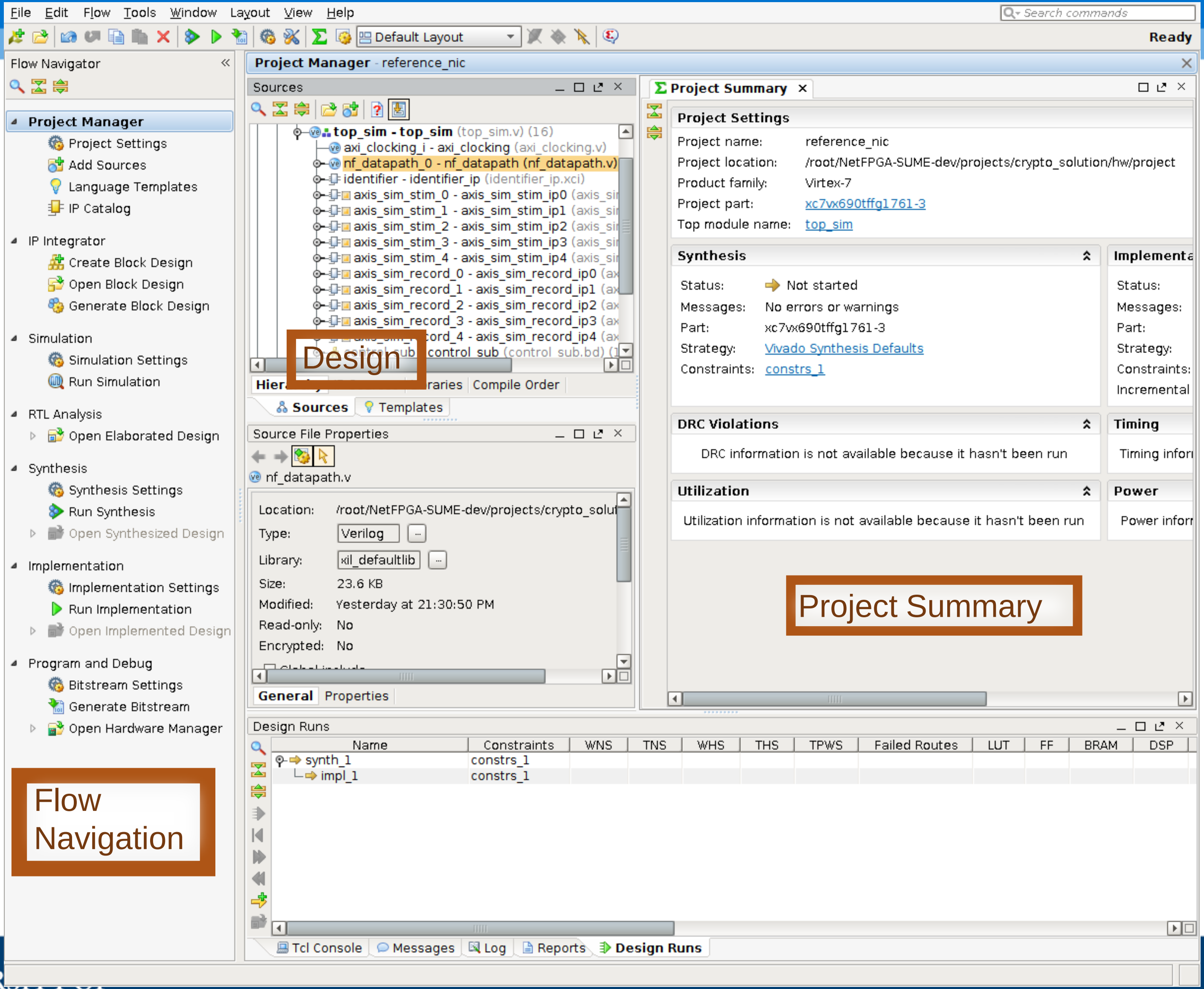
Xilinx Vivado (2)

- To invoke Vivado design tool, run:

  `# vivado <project_root>/hw/project/<project_name>.xpr`

- This will open the project in the Vivado graphical user interface
  - `open a new terminal`
  - `cd <project_root>/projects/ <project_name>/`
  - `source /opt/Xilinx/Vivado/2016.4/settings64.sh`
  - `vivado hw/project/<project name>.xpr`

Vivado Design Tool (1)



Project Manager - reference_nic

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
  ord_4 - axis_sim_record_ip4 (ax
  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

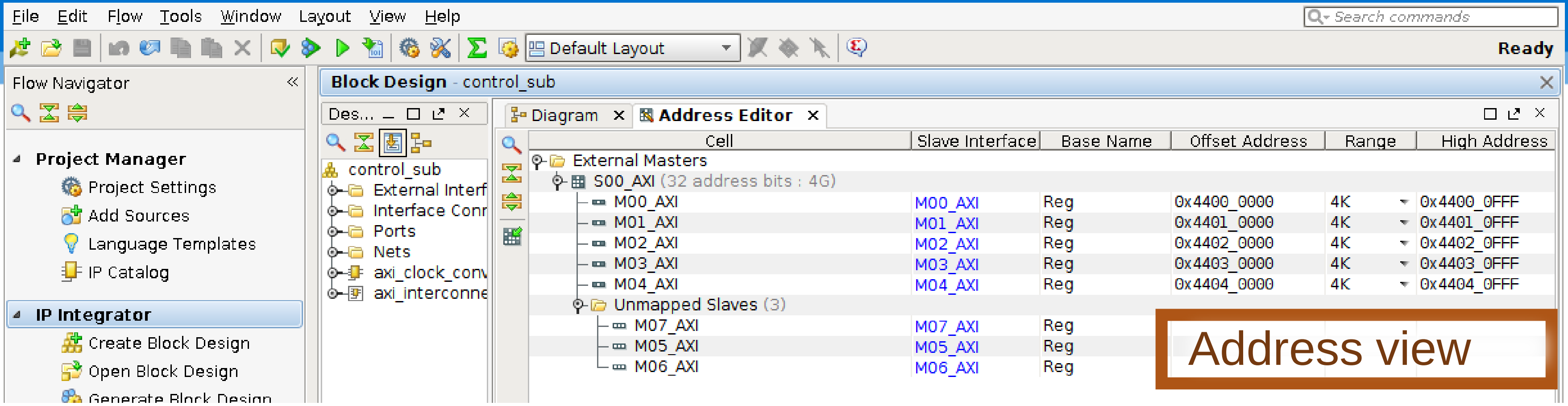
Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP
synth_1	constrs_1										
impl_1	constrs_1										

UNIVERSITY OF CAMBRIDGE

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**

UNIVERSITY OF CAMBRIDGE

Getting started with a project (1)

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

Getting started with a project (2)

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

Getting started with a project (3)

Preparing a module:

1. cd $IP_FOLDER/<ip name>

2. Write and edit files under <ip_name>/hdl Folder

3. make

Notes:

1. review ~/P51/NetFPGA-SUME-live/tools/settings.sh
2. If you make changes:
source ~/P51/NetFPGA-SUME-live/tools/settings.sh

3. Check that make passes without errors

Register Infrastructure

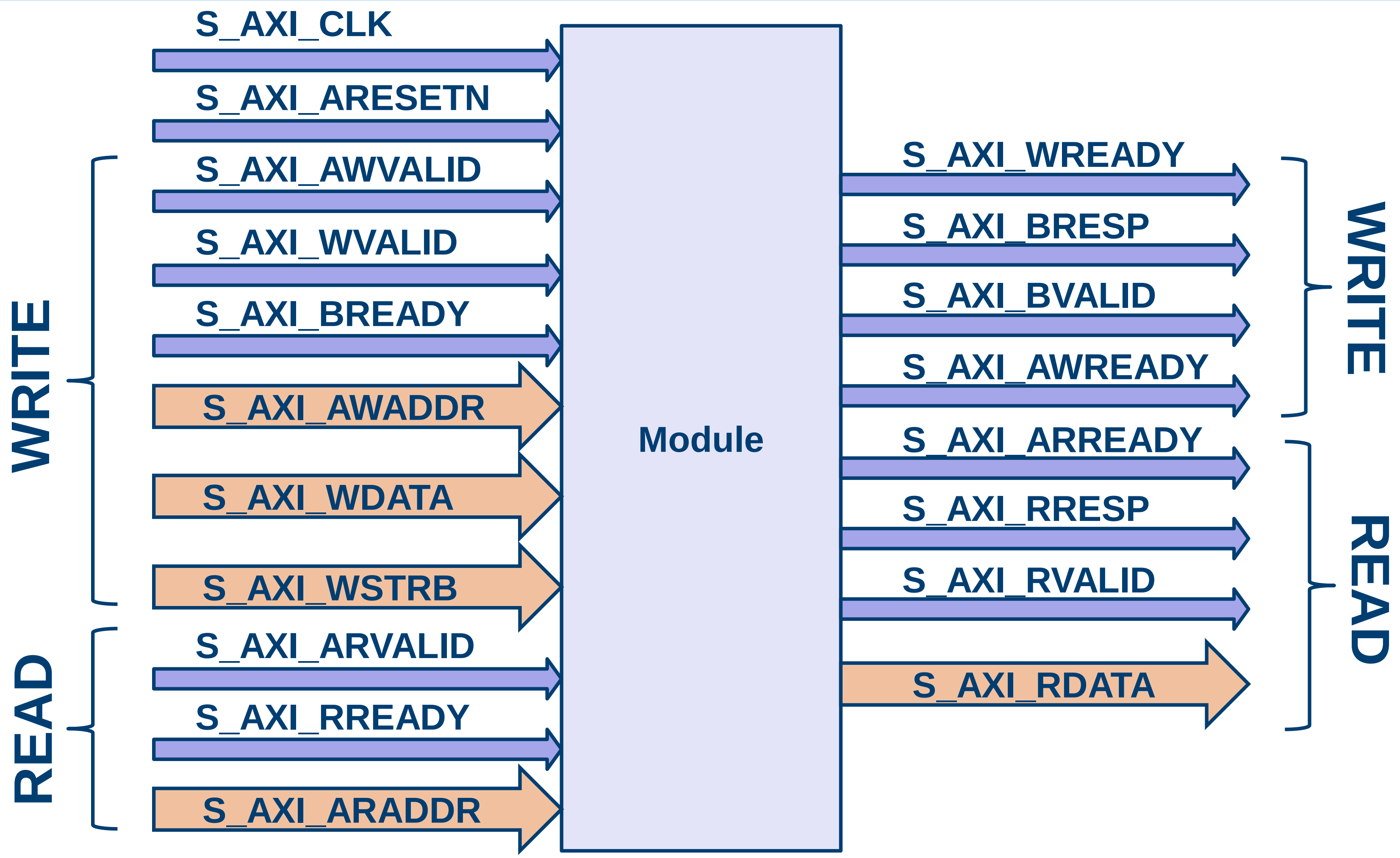
Registers

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

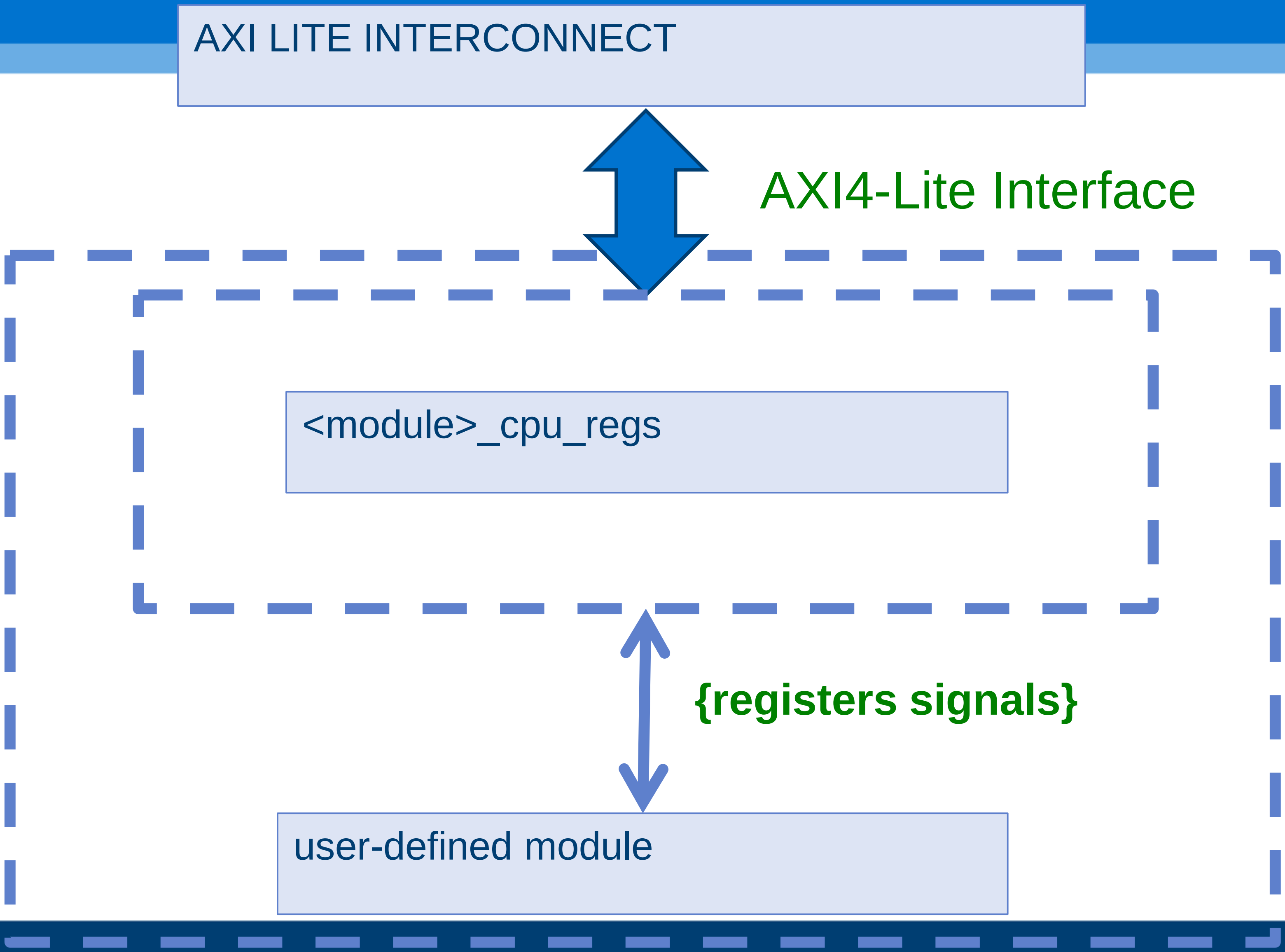
Registers bus

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

Register bus (AXI4-Lite interface)

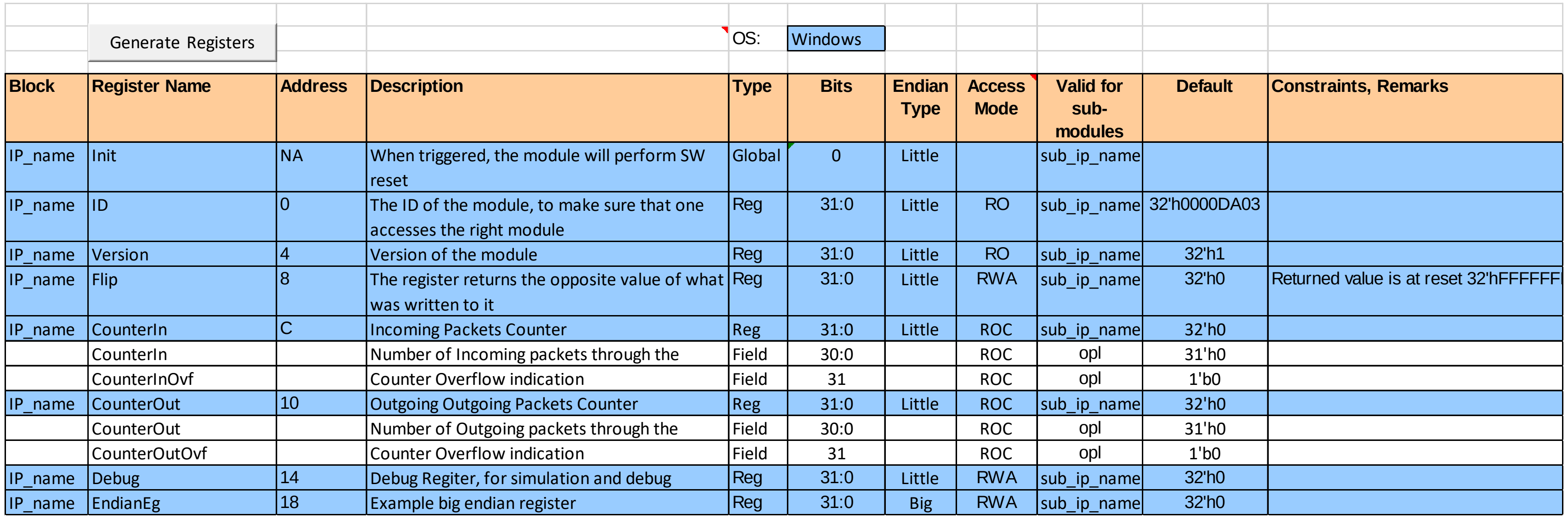


Register bus



Registers – Module generation

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

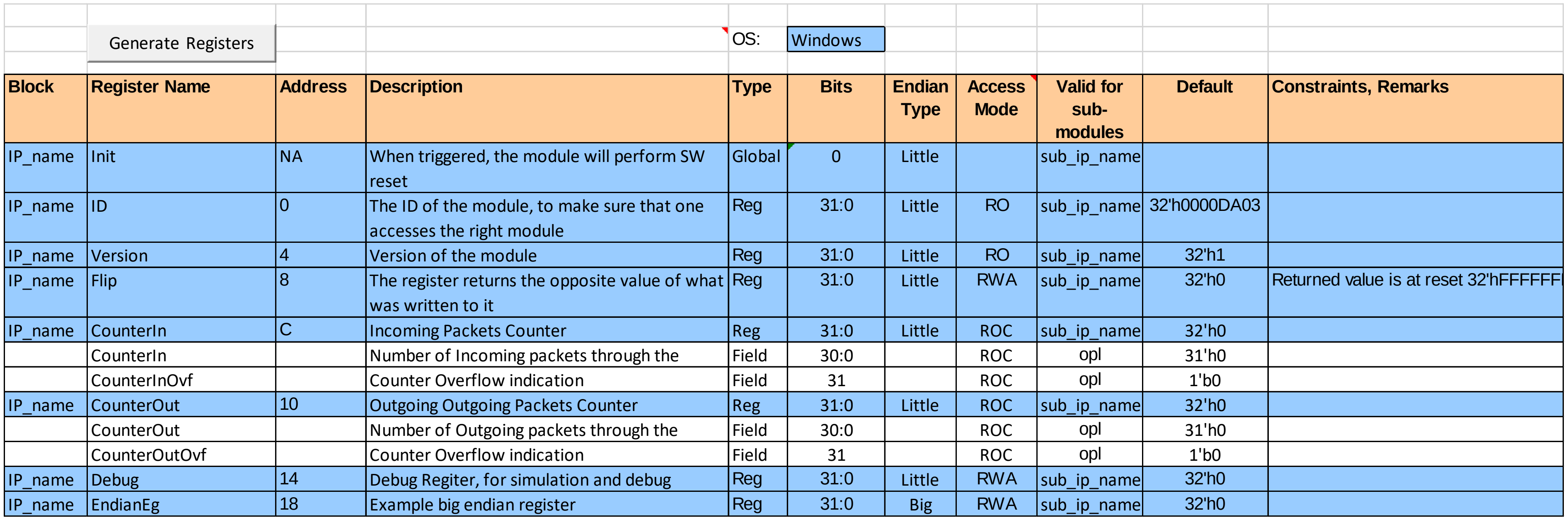


Generate Registers

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'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

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

Adding Registers Logic - Example

- Usage examples:

```
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 $APPS_FOLDER/sume_riffa_v1_0_0/
./rwaxi –a 0x44010000
./rwaxi –a 0x44010000 –w 0x1234
```

You must program the FPGA and load the driver before using these commands!

Can I collect the registers addresses in a unique .h file?

NetFPGA-Host Interaction

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

- The sume_register_defines.h file will be placed under `$NF_DESIGN_DIR/sw/embedded/src`

NetFPGA-Host Interaction

Required steps:

- Generate .h file per core
  - Automatically generated by the python script
- Edit $NF_DESIGN_DIR/hw/tcl/ $NF_PROJECT_NAME_defines.tcl
  - Indicate the address mapping you use
- Edit $NF_DESIGN_DIR/hw/tcl/ export_regiters.tcl
  - Indicate the location of all IP cores used
    - Default path assumed is under \lib\hw\cores

NetFPGA-Host Interaction

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

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

Step by step

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

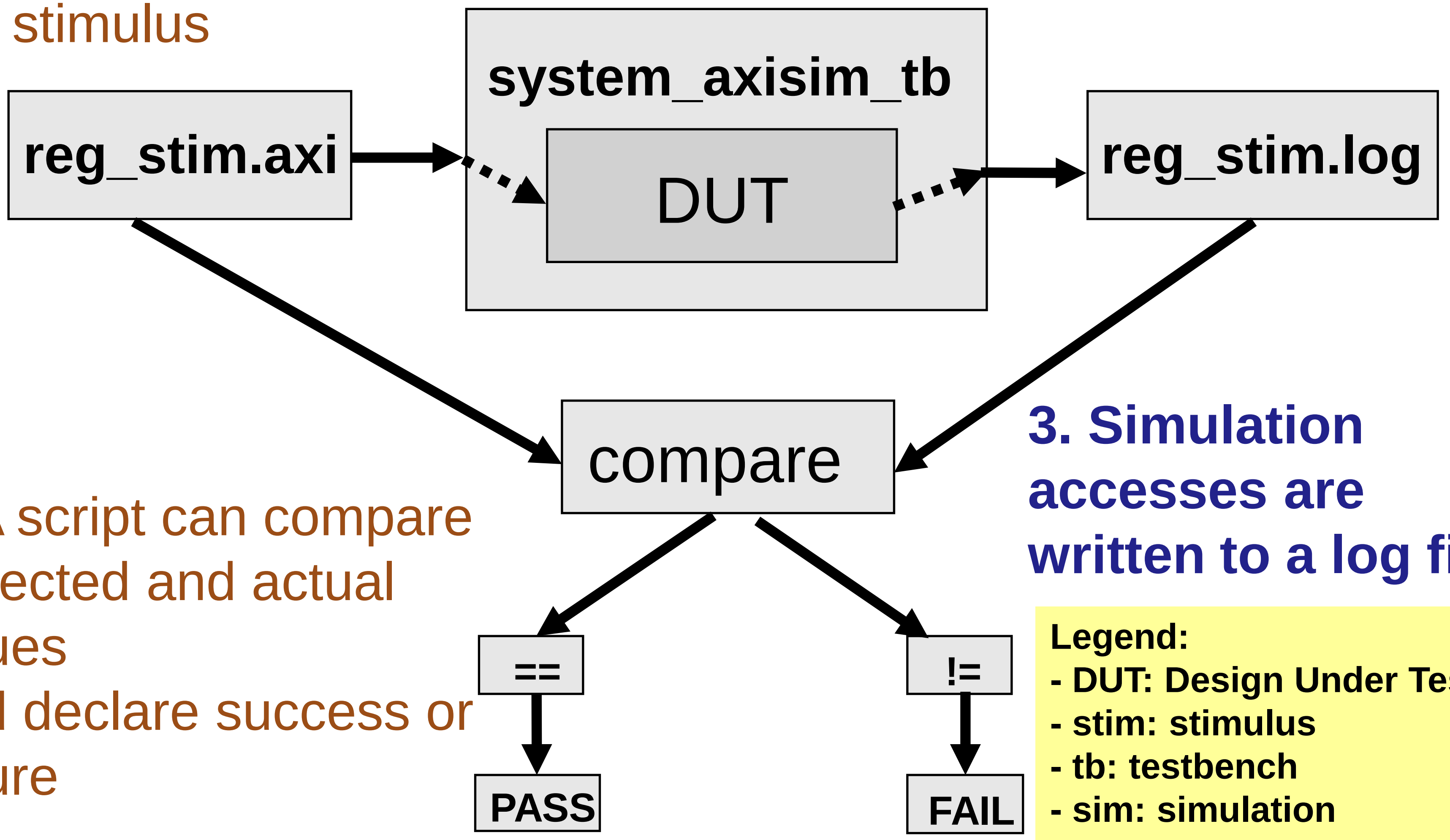
Testing Registers with Simulation

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

Simulating Register Access

1. Define register stimulus

2. The testbench executes the stimulus



3. Simulation accesses are written to a log file

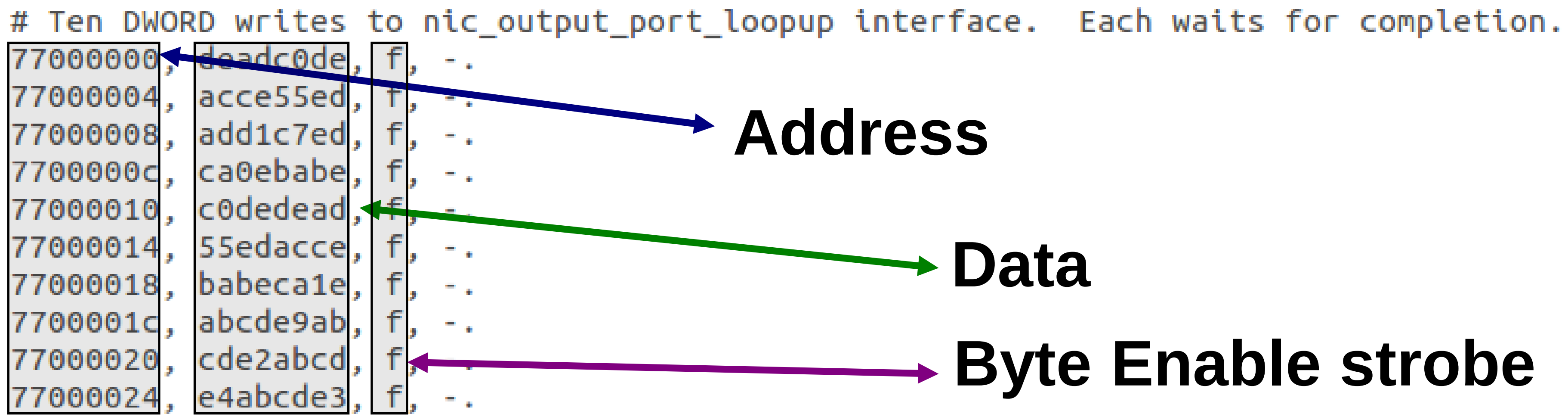
4. A script can compare expected and actual values
And declare success or failure

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

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, -.
```

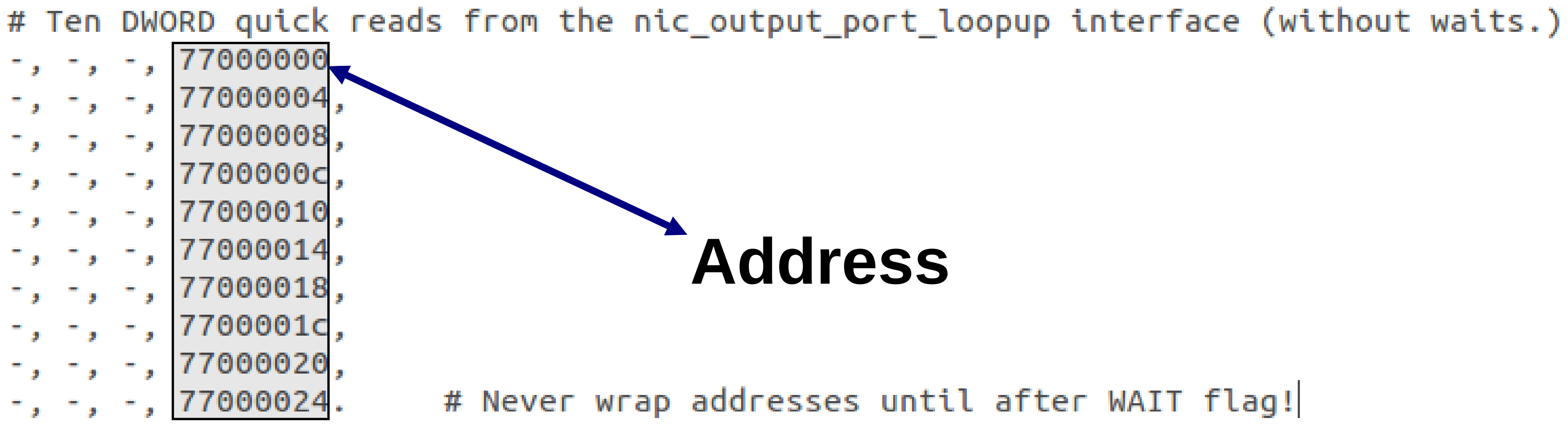
with other useful information like, time, barriers etc..

Registers Stimulus (2)

```
cd $NF_DESIGN_DIR/test/both_learning_sw

less reg_stim.axi
```

- An example read format :

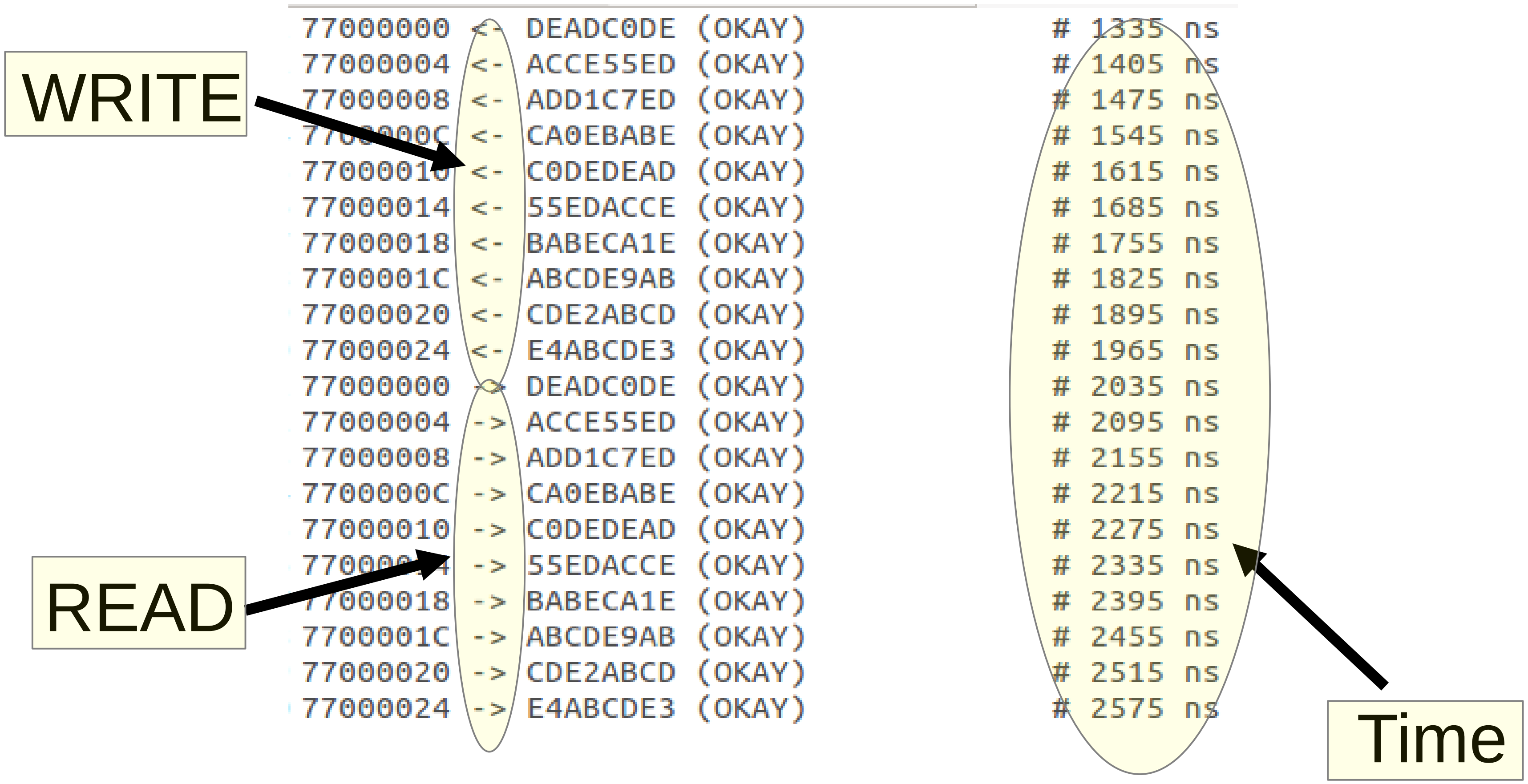


```
# Ten DWORD quick reads from the nic_output_port_loopup interface (without waits.)
-, -, -, 77000000
-, -, -, 77000004,
-, -, -, 77000008,
-, -, -, 7700000c,
-, -, -, 77000010,
-, -, -, 77000014,
-, -, -, 77000018,
-, -, -, 7700001c,
-, -, -, 77000020,
-, -, -, 77000024.       # Never wrap addresses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

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

Registers Access Log

```
cd $NF_DESIGN_DIR/test/both_learning_sw
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
```

Section II: Testing Hardware

Synthesis

- To synthesize your project:

```
cd $NF_DESIGN_DIR

make
```

Hardware Tests

- Test compiled hardware

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

Python Libraries

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

Creating a Hardware Test

Useful functions:

- Packet generation:
  - make_IP_pkt(…) – see wiki
  - encrypt_pkt(key, pkt)
  - decrypt_pkt(key, pkt)
- Packet transmission/reception:
  - nftest_send_phy(interface, pkt)
  - nftest_expect_phy(interface, pkt)
  - nftest_send_dma(interface, pkt)
  - nftest_expect_dma(interface, pkt)
- Register access:
  - nftest_regwrite(addr, value)
  - nftest_regread_expect(addr, expect)

Understanding Hardware Test

- `cd $NF_DESIGN_DIR/test/both_learning_sw`
- `vim run.py`
- “isHW” indicates HW test
- “connections/conn” file declares the physical connections

```
nf0:eth1
nf1:eth2
nf2:
nf3:
```

- “global/setup” file defines the interfaces

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

Your task:

  - Remember to source the settings.sh file
  - Edit run.py to create your test
  - Edit setup and conn files

Running Hardware Tests

- Use command nf_test.py
  - Required Parameter
    - sim hw or both (right now only use hw)
  - Optional parameters
    - --major <major_name>
    - --minor <minor_name>

both_learning_sw

major minor

Running Hardware Tests

- Having problems?

- Take advantage of the wiki!

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

  - Detailed explanations
  - Tips for debug