UNIVERSITY OF CAMBRIDGE

P51: High Performance Networking

Introduction to P4 and NetFPGA – Part II

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

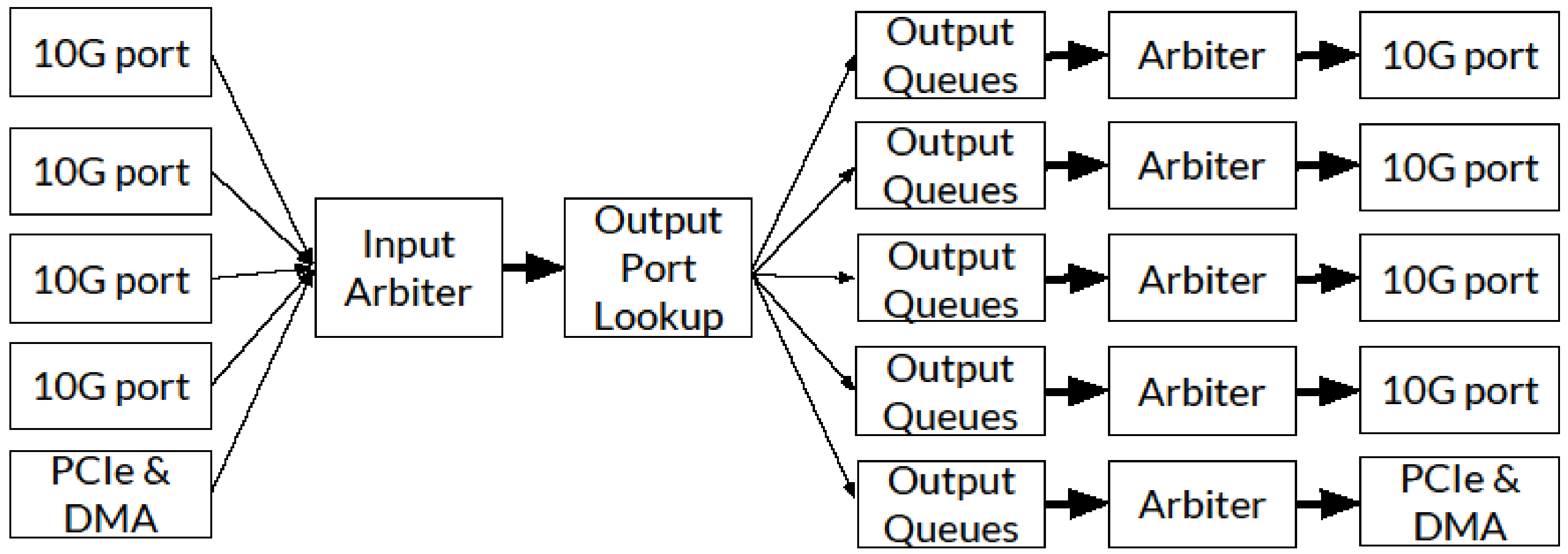
Lent 2018/19

Project

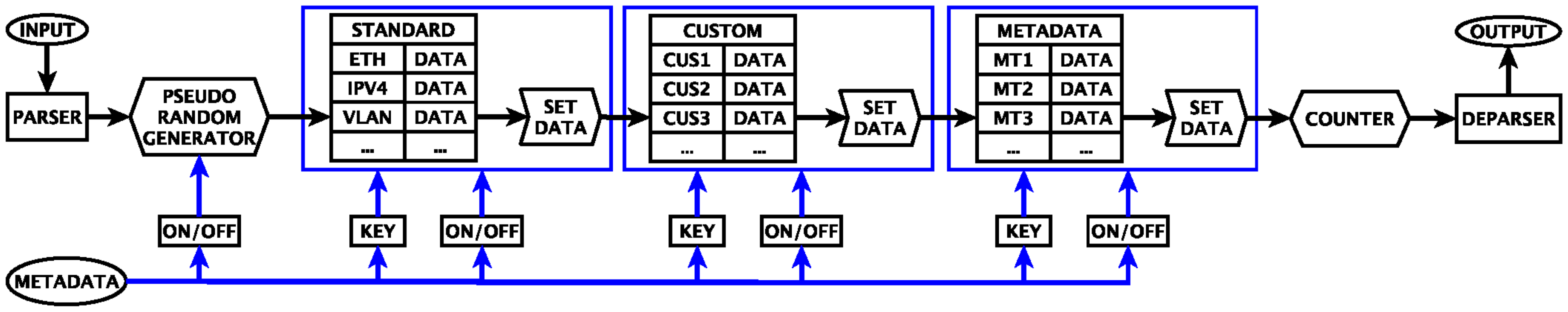
Project

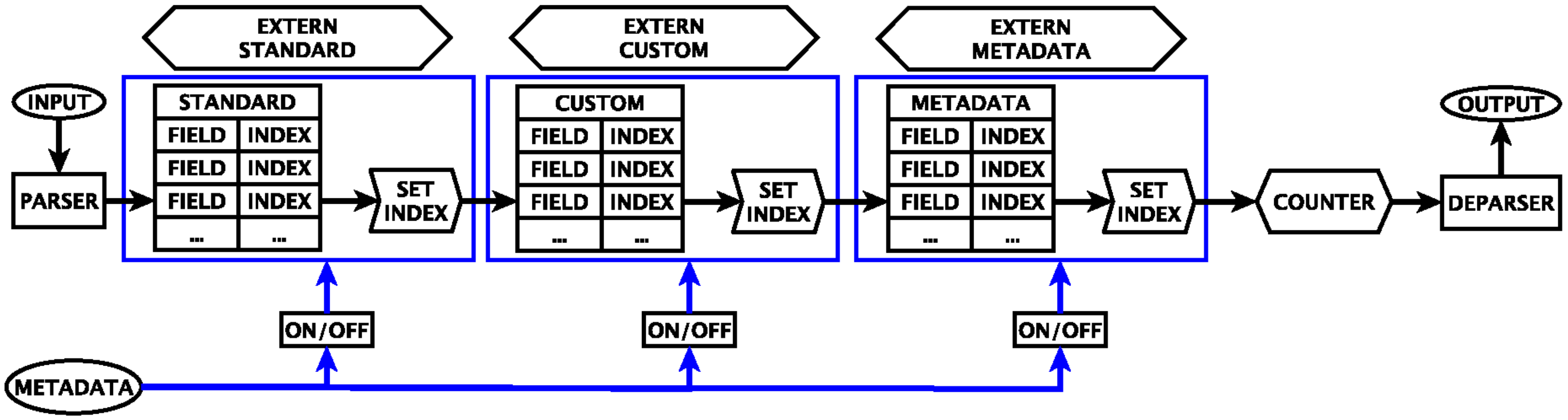
- Today
  - P4
  - Proposal and aims
- Next Week
  - Architecture
  - Dividing the effort
- Project test infrastructure

Architecture



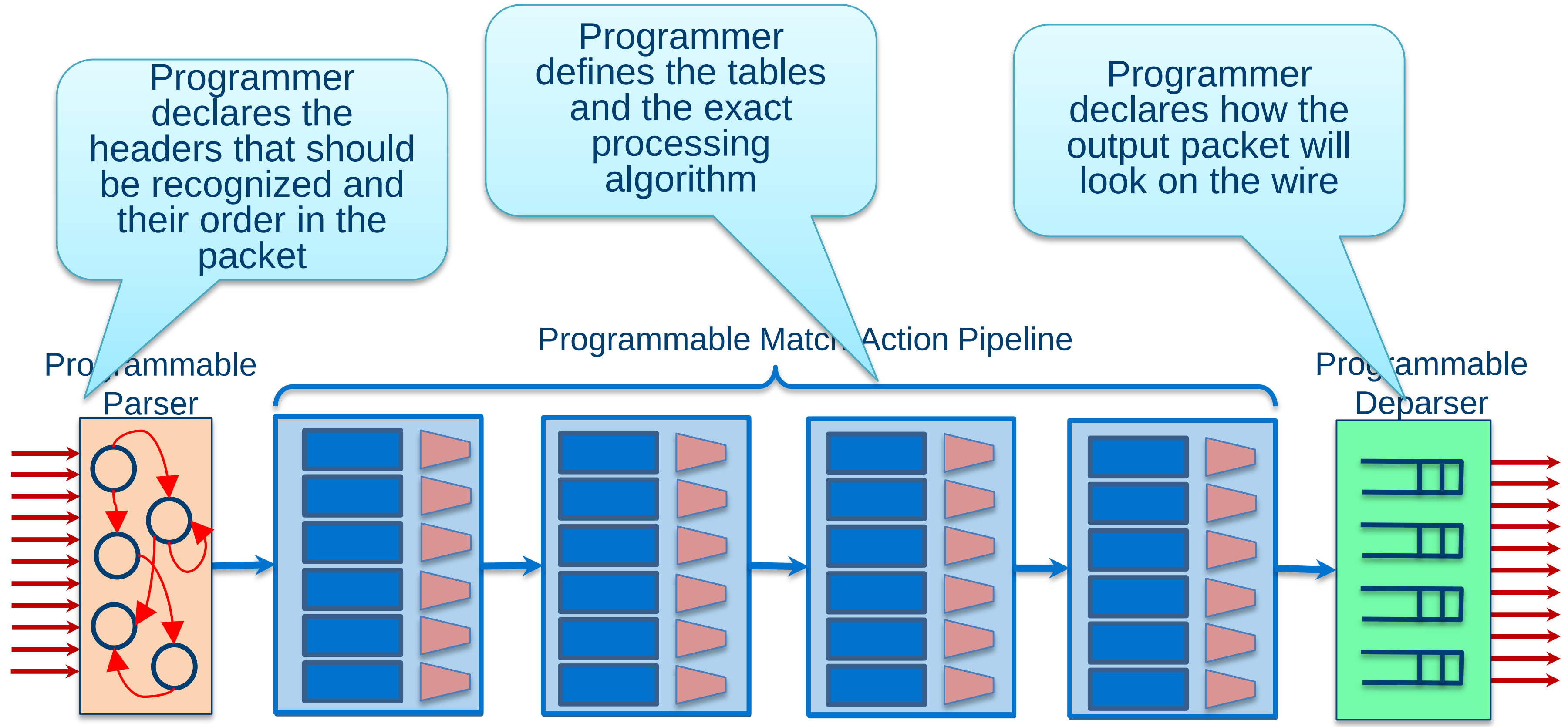
Architecture





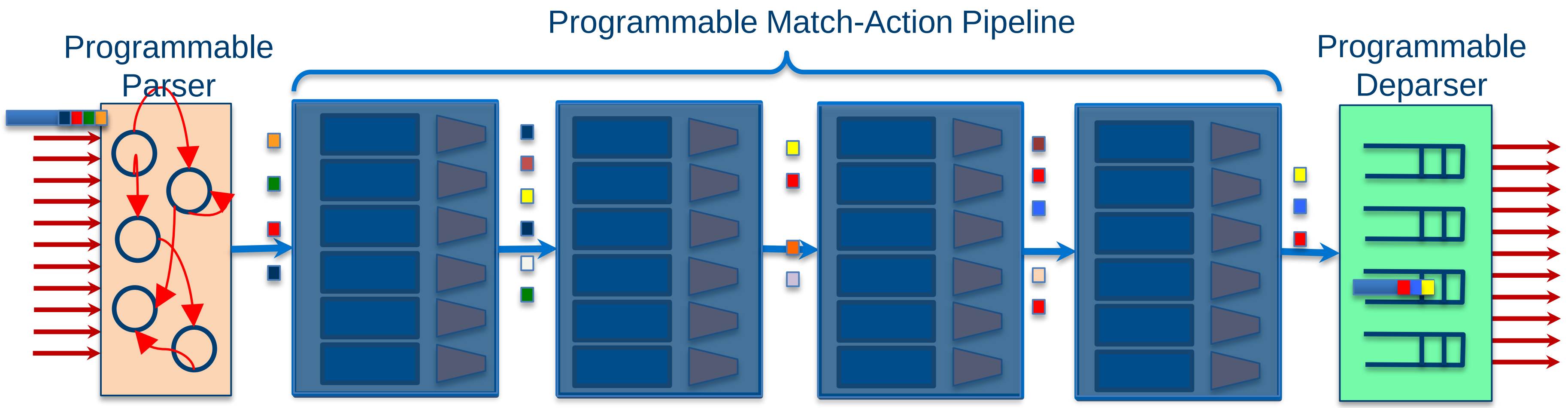
P4

PISA: Protocol-Independent Switch Architecture


Programmer declares the headers that should be recognized and their order in the packet
Programmer defines the tables and the exact processing algorithm
Programmer declares how the output packet will look on the wire
Programmable Match-Action Pipeline
Programmable Parser
Programmable Deparser

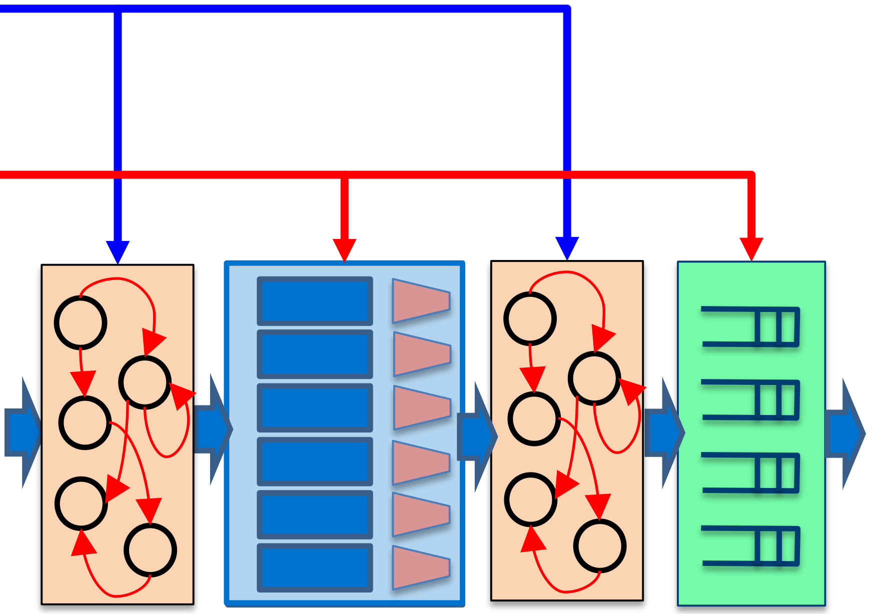
PISA in Action

- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)



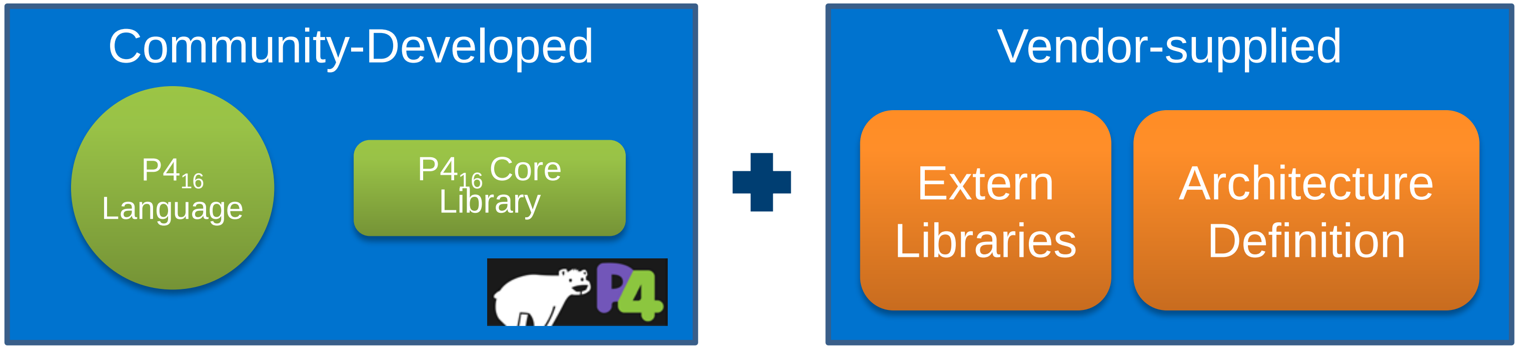
P4$_{16}$ Language Elements

Parsers	State machine, bitfield extraction
Controls	Tables, Actions, control flow statements
Expressions	Basic operations and operators
Data Types	Bistrings, headers, structures, arrays
Architecture Description	Programmable blocks and their interfaces
Extern Libraries	Support for specialized components

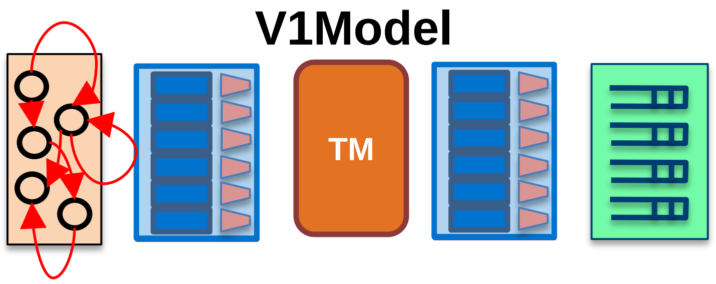


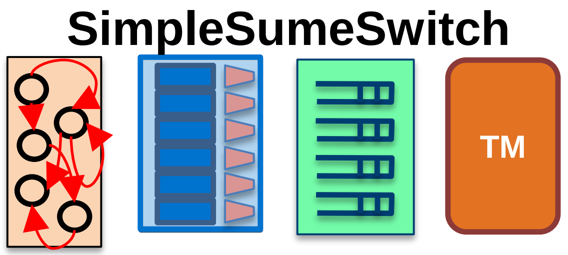
P4_16 Approach

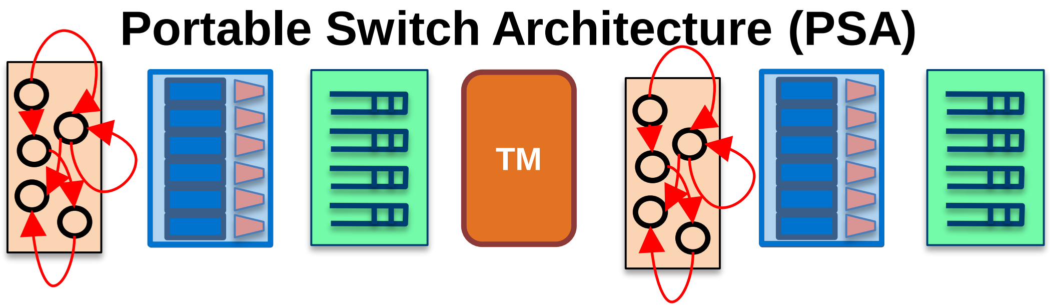
Term	Explanation
P4 Target	An embodiment of a specific hardware implementation
P4 Architecture	A specific set of P4-programmable components, externs, fixed components and their interfaces available to the P4 programmer
P4 Platform	P4 Architecture implemented on a given P4 Target



Example Architectures and Targets

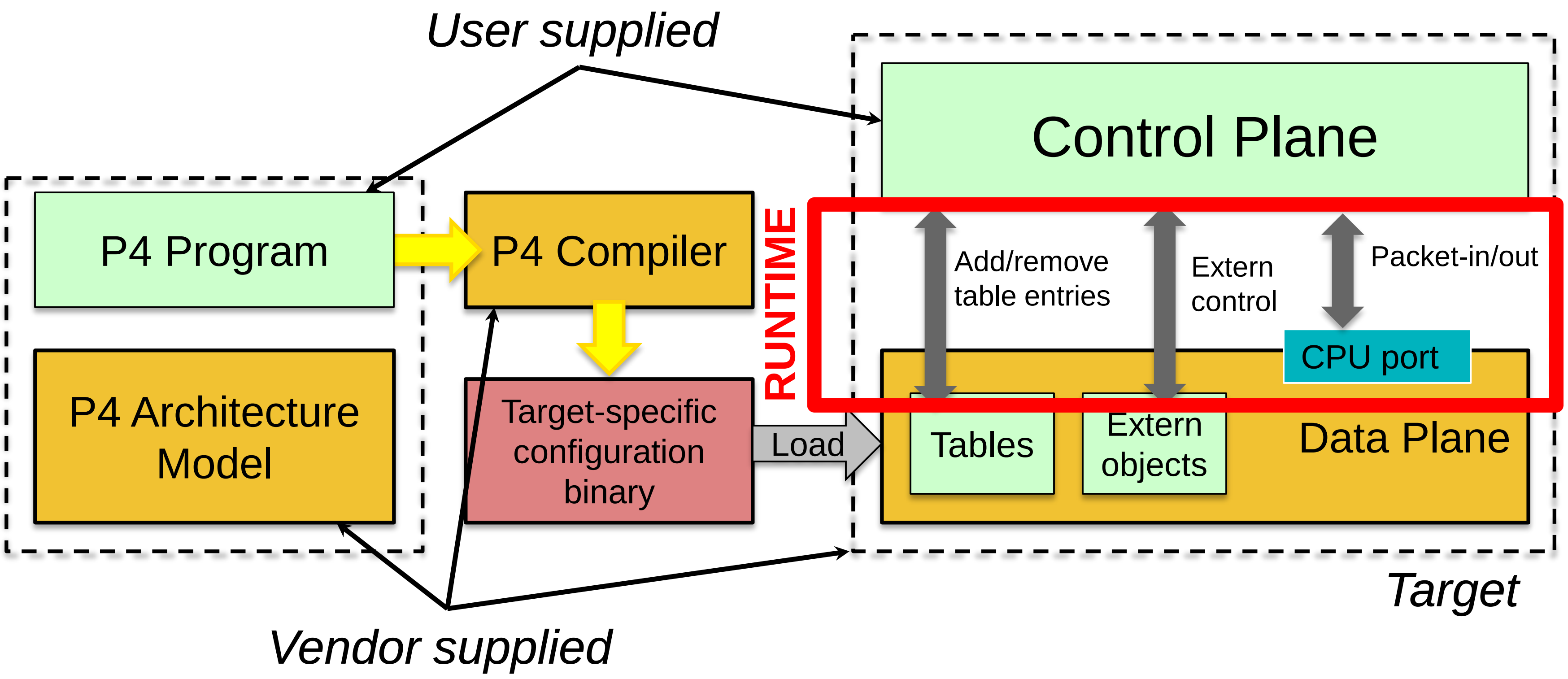




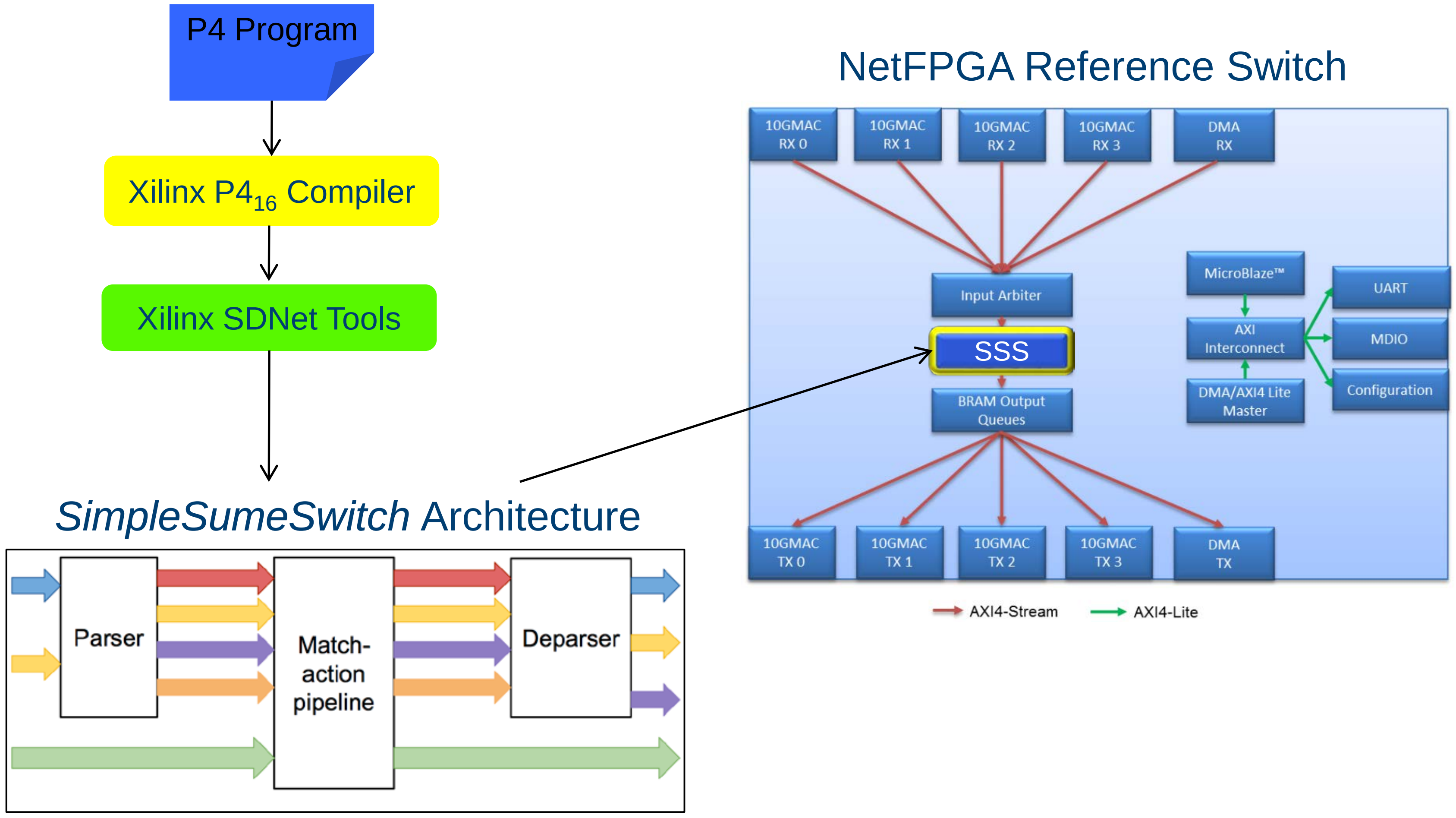


Anything

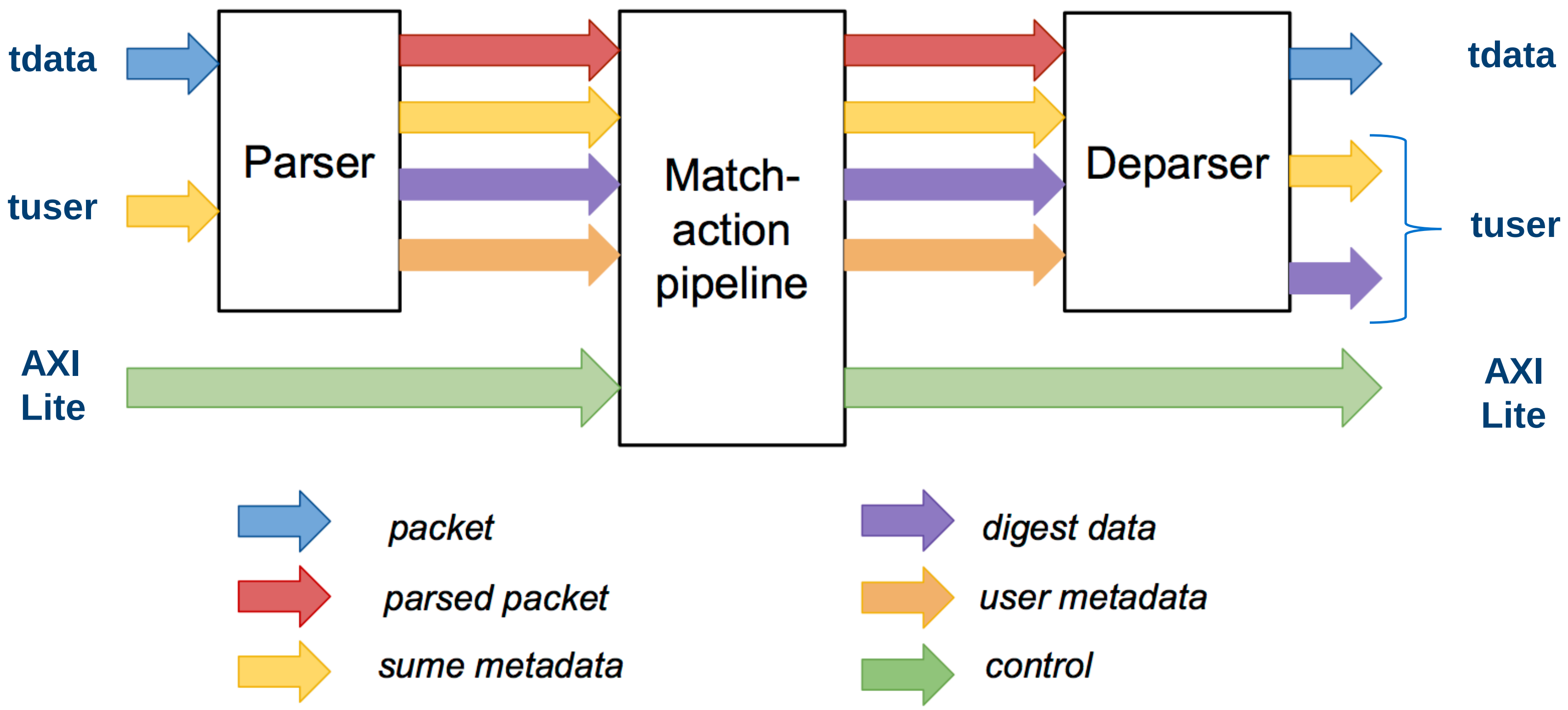
Programming a P4 Target



Reminder: P4 on NetFPGA (P4-NetFPGA)



SimpleSumeSwitch Architecture Model



P4 used to describe parser, match-action pipeline, and deparser

V1Model Standard Metadata

```
struct standard_metadata_t {
   bit<9>  ingress_port;
   bit<9>  egress_spec;
   bit<9>  egress_port;
   bit<32> clone_spec;
   bit<32> instance_type;
   bit<1>  drop;
   bit<16> recirculate_port;
   bit<32> packet_length;
   bit<32> enq_timestamp;
   bit<19> enq_qdepth;
   bit<32> deq_timedelta;
   bit<19> deq_qdepth;
   bit<48> ingress_global_timestamp;
   bit<32> lf_field_list;
   bit<16> mcast_grp;
   bit<1>  resubmit_flag;
   bit<16> egress_rid;
   bit<1>  checksum_error;
}
```

- **`ingress_port`** - the port on which the packet arrived
- **`egress_spec`** - the port to which the packet should be sent to
- **`egress_port`** - the port on which the packet is departing from (read only in egress pipeline)

Standard Metadata in SimpleSumeSwitch Architecture

```
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    bit<8> dst_port; // one-hot encoded
    bit<8> src_port; // one-hot encoded
    bit<16> pkt_len; // unsigned int
}
```

- *_q_size – size of each output queue, measured 32B words
  Taken when the packet starts being processed by the P4 program
- src_port/dst_port – one-hot encoded, easy to do multicast
- user_metadata/digest_data – structs defined by the user

UNIVERSITY OF CAMBRIDGE

P4$_{16}$ Program Template (V1Model)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t    ethernet;
  ipv4_t        ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

P4$_{16}$ Program Template (SimpleSumeSwitch)

```
#include <core.p4>
#include <sume_switch.p4>
/* HEADERS */
struct user_metadata_t { ... }
struct digest_data_t  { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser TopParser (packet_in b,
                  out Parsed_packet p,
                  out user_metadata_t user_metadata,
                  out digest_data_t digest_data,
                  inout sume_metadata_t sume_metadata) {
  ...
}
/* PROCESSING */
control TopPipe (inout Parsed_packet p,
                 inout user_metadata_t user_metadata,
                 inout digest_data_t digest_data,
                 inout sume_metadata_t sume_metadata) {
  ...
}
```

```
/* DEPARSER */
control TopDeparser (packet_out b,
                     in Parsed_packet p,
                     in user_metadata_t user_metadata,
                     inout digest_data_t digest_data,
                     inout sume_metadata_t sume_metadata) {
  ...
}

/* SWITCH */
SimpleSumeSwitch (
  TopParser(),
  TopPipe(),
  TopDeparser()
) main;
```

P4$_{16}$ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
   out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {

    state start { transition accept; }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {   apply {  }   }

control MyIngress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
apply {
        if (standard_metadata.ingress_port == 1) {
            standard_metadata.egress_spec = 2;
        } else if (standard_metadata.ingress_port == 2) {
            standard_metadata.egress_spec = 1;
        }
    }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {
     apply { }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch(
   MyParser(),
   MyVerifyChecksum(),
   MyIngress(),
   MyEgress(),
   MyComputeChecksum(),
   MyDeparser()
) main;
```

$P4_{16}$ Hello World (V1Model)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply {   forward.apply();   }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {   apply { }   }

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {   apply { }   }

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

$P4_{16}$ Types (Basic and Header Types)

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16>   etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
```

Basic Types

- **bit<n>**: Unsigned integer (bitstring) of size n
- **bit** is the same as **bit<1>**
- **int<n>**: Signed integer of size n (>=2)
- **varbit<n>**: Variable-length bitstring

Header Types: Ordered collection of members

- Can contain **bit<n>**, **int<n>**, and **varbit<n>**
- Byte-aligned
- Can be valid or invalid
- Provides several operations to test and set validity bit: **isValid()**, **setValid()**, and **setInvalid()**

Typedef: Alternative name for a type

P4$_{16}$ Types (Other Types)

```
/* Architecture */
struct standard_metadata_t {
  bit<9>  ingress_port;
  bit<9>  egress_spec;
  bit<9>  egress_port;
  bit<32> clone_spec;
  bit<32> instance_type;
  bit<1>  drop;
  bit<16> recirculate_port;
  bit<32> packet_length;
  ...
}

/* User program */
struct metadata {
  ...
}
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
```

Other useful types

- **Struct**: Unordered collection of members (with no alignment restrictions)
- **Header Stack:** array of headers
- **Header Union:** one of several headers

Intrinsic Metadata

- The data that a P4-programmable components can use to interface with the rest of the system
- Defined in the files supplied by the vendor

Declaring and Initializing Variables

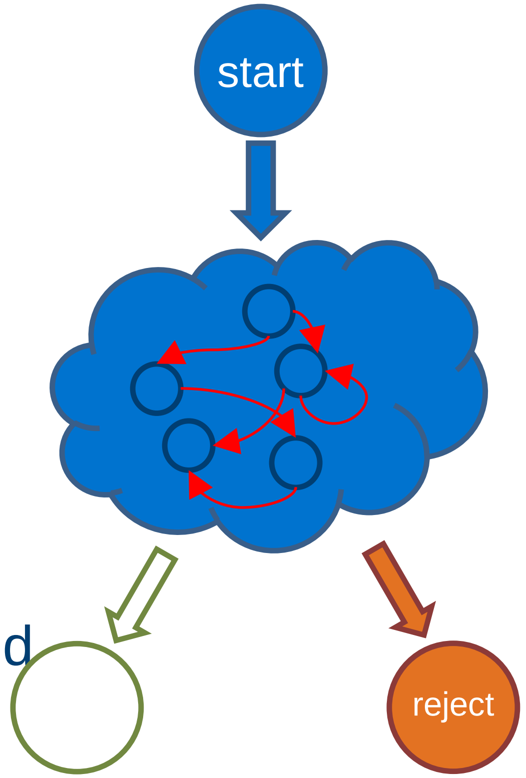
```
bit<16>         my_var;
bit<8>          another_var = 5;

const bit<16> ETHERTYPE_IPV4 = 0x0800;
const bit<16> ETHERTYPE_IPV6 = 0x86DD;

ethernet_t      eth;
vlan_tag_t      vtag = {3w2, 0, 12w13, 16w0x8847};
```

Safe constants with explicit widths

- In $P4_{16}$ you can instantiate variables of both base and derived types
- Variables can be initialized
  - Including the composite types
- Constant declarations make for safer code
- Infinite width and explicit width constants

P4$_{16}$ Parsers

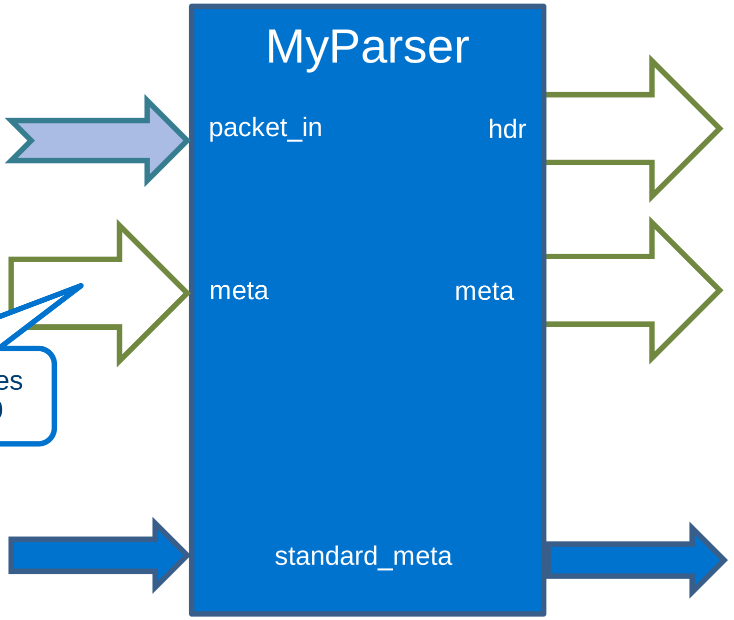
- Parsers are functions that map packets into headers and metadata,
  - written in a state machine style
- Every parser has three predefined states
  - `start`
  - `accept`
  - `reject`
- Other states may be defined by the programmer
- In each state, execute zero or more statements, and then transition to another state (loops are OK)



UNIVERSITY OF CAMBRIDGE

Parsers (V1Model)

```
/* From core.p4 */
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T variableSizeHeader,
                  in bit<32> variableFieldSizeInBits);
  T lookahead<T>();
  void advance(in bit<32> sizeInBits);
  bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t std_meta) {

  state start {
     packet.extract(hdr.ethernet);
     transition accept;
  }

}
```



The platform Initializes User Metadata to 0

UNIVERSITY OF CAMBRIDGE

Select Statement

```
state start {
  transition parse_ethernet;
}

state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.etherType) {
    0x800: parse_ipv4;
    default: accept;
  }
}
```

$P4_{16}$ has a `select` statement that can be used to branch in a parser

Similar to `case` statements in C or Java, but without "fall-through behavior"—i.e., `break` statements are not needed

In parsers it is often necessary to branch based on some of the bits just parsed

For example, etherType determines the format of the rest of the packet

Match patterns can either be literals or simple computations such as masks

P4$_{16}$ Controls

- **Similar to C functions (without loops)**
- **Can declare variables, create tables, instantiate externs, etc.**
- **Functionality specified by code in `apply` statement**
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional forms of packet processing (updating checksums)
- **Interfaces with other blocks are governed by user- and architecture-specified types (typically headers and metadata)**

Example: Reflector (V1Model)

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  /* Declarations region */
  bit<48> tmp;

  apply {
    /* Control Flow */
    tmp = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
    hdr.ethernet.srcAddr = tmp;
    std_meta.egress_spec = std_meta.ingress_port;
  }
}
```

Desired Behavior:

- **Swap source and destination MAC addresses**
- **Bounce the packet back out on the physical port that it came into the switch on**

Example: Simple Actions

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  action swap_mac(inout bit<48> src,
                  inout bit<48> dst) {
    bit<48> tmp = src;
    src = dst;
    dst = tmp;
  }

  apply {
    swap_mac(hdr.ethernet.srcAddr,
             hdr.ethernet.dstAddr);
    std_meta.egress_spec = std_meta.ingress_port;
  }
}
```

- **Very similar to C functions**
- **Can be declared inside a control or globally**
- **Parameters have type and direction**
- **Variables can be instantiated inside**
- **Many standard arithmetic and logical operations are supported**
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Non-standard operations:**
  - Bit-slicing: [m:l] (works as l-value too)
  - Bit Concatenation: ++

Operating on Headers

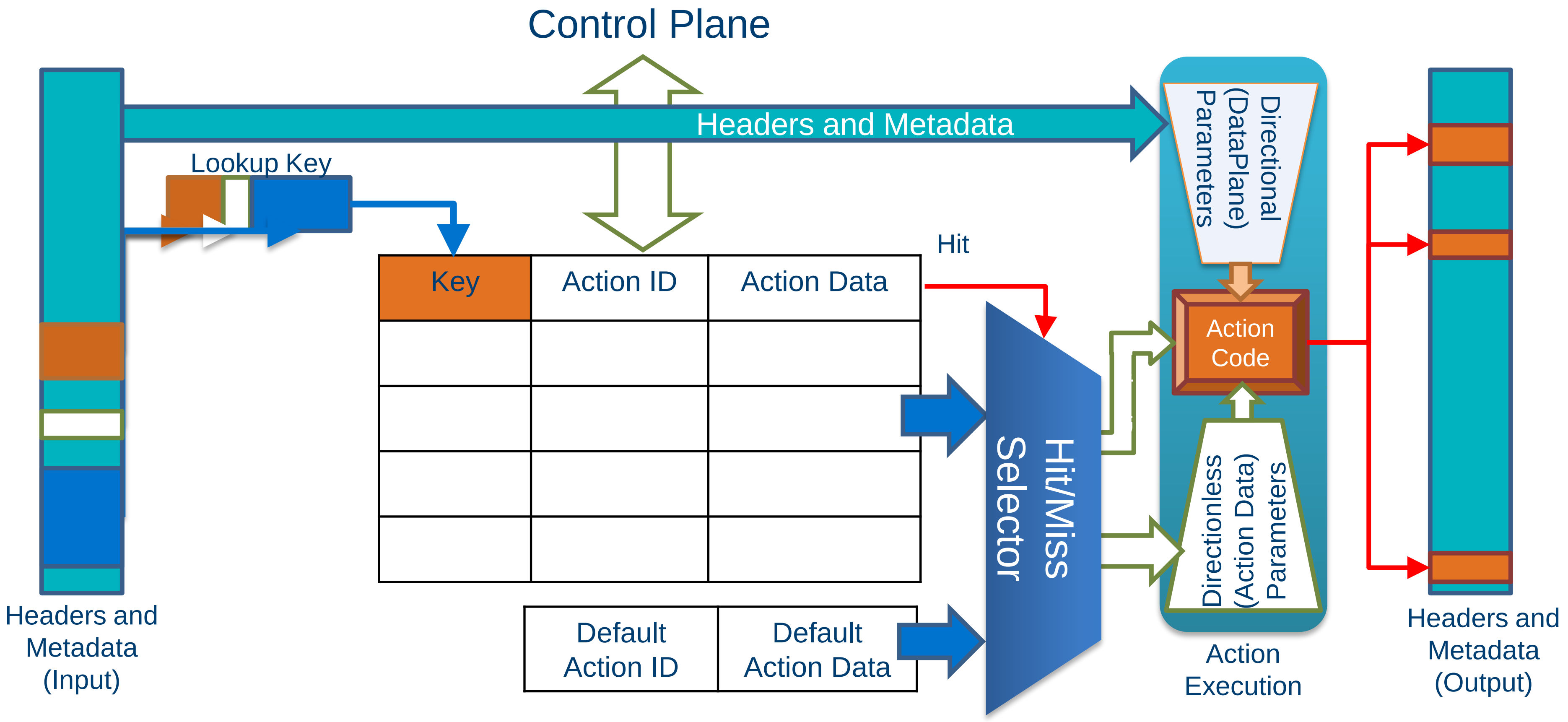
```
action decap_ip_ip() {
    hdr.ipv4 = hdr.inner_ipv4;
    hdr.inner_ipv4.setInvalid();
}

action pop_mpls_label() {
    hdr.mpls.pop_front(1);
}

action push_mpls_label(in bit<20> label, in
bit<3> exp) {
    hdr.mpls.push_front(1);
    hdr.mpls[0].setValid();
    hdr.mpls[0] = { label, exp, 0, 64 };
}
```

- Header Validity bit manipulation:
  - header.**setValid**() – add_header
  - header.**setInvalid**() – remove_header
  - header.**isValid**()
- Header Assignment
  - header = { f1, f2, ..., fn }
  - header1 = header2
- Special operations on Header Stacks
  - In the parsers
    - header_stack.**next**
    - header_stack.**last**
    - header_stack.**lastIndex**
  - In the controls
    - header_stack[i]
    - header_stack.**size**
    - header_stack.**push_front**(**int** count)
    - header_stack.**pop_front**(**int** count)

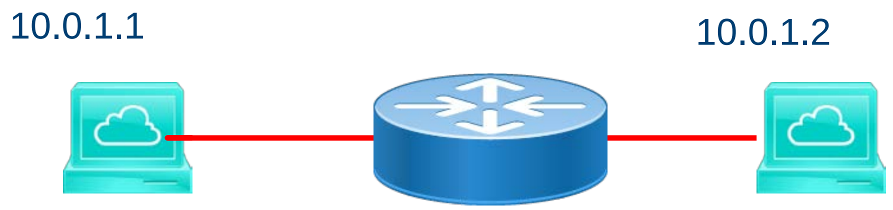
P4$_{16}$ Tables

- **The fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - Static entries
    - etc.
- **Each table contains one or more entries (rules)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action that is executed when a packet matches the entry
  - Action data (possibly empty)

Tables: Match-Action Processing



Example: IPv4_LPM Table



Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

- **Data Plane (P4) Program**
  - Defines the format of the table
    - Key Fields
    - Actions
    - Action Data
  - Performs the lookup
  - Executes the chosen action
- **Control Plane (IP stack, Routing protocols)**
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations

IPv4_LPM Table

```
 table ipv4_lpm {
    key = {
      hdr.ipv4.dstAddr: lpm;
    }
    actions = {
      ipv4_forward;
      drop;
      NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

Match Kinds

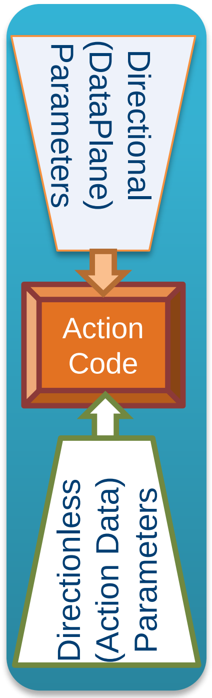
```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */
match_kind {
    range,
    selector
}

/* Some other architecture */
match_kind {
    regexp,
    fuzzy
}
```

- **The type `match_kind` is special in P4**
- **The standard library (`core.p4`) defines three standard match kinds**
    - Exact match
    - Ternary match
    - LPM match
- **The architecture (`v1model.p4`) defines two additional match kinds:**
    - range
    - selector
- **Other architectures may define (and provide implementation for) additional match kinds**
- **P4-SDNet supports** `exact, ternary, lpm and direct.`

Defining Actions for L3 forwarding

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
  mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
  ...
}
```

- **Actions can have two different types of parameters**
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- **Actions that are called directly:**
  - Only use directional parameters
- **Actions used in tables:**
  - Typically use directionless parameters
  - May sometimes use directional parameters too


Directional (DataPlane) Parameters
Action Code
Directionless (Action Data) Parameters

Action Execution

Applying Tables in Controls

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  table ipv4_lpm {
    ...
  }
  apply {
     ...
     ipv4_lpm.apply();
     ...
  }
}
```

Table Initialization

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
  table ipv4_lpm {
    ...
  }
  apply {
     ...
  }
  const entries = {
   /*{ dstAddr, srcAddr, vlan_tag[0].isValid(), vlan_tag[1].isValid() } : action([action_data])*/
          { 48w000000000000, _,  _, _ } : malformed_ethernet(ETHERNET_ZERO_DA);
          { _, 48w000000000000, _, _ } : malformed_ethernet(ETHERNET_ZERO_SA);
          { _, 48w010000000000 &&& 48w010000000000, _, _ } : malformed_ethernet(ETHERNET_MCAST_SA);
          { 48wFFFFFFFFFFFF, _, 0, _ } : broadcast_untagged();
          { 48wFFFFFFFFFFFF, _, 1, 0 } : broadcast_single_tagged();
          { 48wFFFFFFFFFFFF, _, 1, 1 } : broadcast_double_tagged();
          { 48w010000000000 &&& 48w010000000000, _, 0, _ } : multicast_untagged();
          { _, _, 0, _ } : unicast_untagged();
          { _, _, 1, 0 } : unicast_single_tagged();
  }
}
```

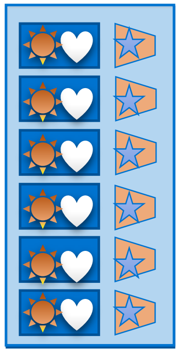
P4$_{16}$ Deparsing

```
/* From core.p4 */
extern packet_out {
  void emit<T>(in T hdr);
}

/* User Program */
control DeparserImpl(packet_out packet,
                     in headers hdr) {
  apply {
    ...
    packet.emit(hdr.ethernet);
    ...
  }
}
```

- **Assembles the headers back into a well-formed packet**
- **Expressed as a control function**
  - No need for another construct!
- **packet_out extern is defined in core.p4:** emit(hdr): serializes header if it is valid
- **Advantages:**
  - Makes deparsing explicit...
    ...but decouples from parsing

The Need for Externs

- Most platforms contain specialized facilities
  - Differ from vendor to vendor
  - Can’t be expressed in the core language
  - Might have control-plane accessible state or configuration
- The language should stay the same
  - In $P4_{14}$ almost 1/3 of all the constructs were dedicated to specialized processing
  - In $P4_{16}$ all specialized objects use the same interface
- Objects can be used even if their implementation is hidden
  - Through instantiation and method calling



Stateless and Stateful Objects

- Stateless Objects: Reinitialized for each packet
  - Variables (metadata), packet headers, packet_in, packet_out

- Stateful Objects: Keep their state between packets
  - Tables
  - Externs
    - V1 architecture: Counters, Meters, Registers, Parser Value Sets, Selectors, etc.

P4-NetFPGA Extern Function library

- Implement platform specific functions
  - Black box to P4 program
- Implemented in HDL
- Stateless – reinitialized for each packet
- Stateful – keep state between packets
- Xilinx Annotations
  - `@Xilinx_MaxLatency()` – maximum number of clock cycles an extern function needs to complete
  - `@Xilinx_ControlWidth()` – size in bits of the address space to allocate to an extern function

P4-NetFPGA Extern Function library

- HDL modules invoked from within P4 programs
- Stateful atomic operations:

Extern	Description
R/W	Read or write state
RAW	Read, add to, or overwrite state
PRAW	Predicated version of RAW
ifElseRAW	Two RAWs, one each for when predicate is true or false
Sub	IfElseRAW with stateful subtraction capability

- Stateless Externs

Extern	Description
IP Checksum	Given an IP header, compute IP checksum
LRC	Longitudinal redundancy check, simple hash function
timestamp	Generate timestamp (granularity of 5 ns)

Using Externs in P4 – Resetting Counter

Packet processing pseudo code:

```
count[NUM_ENTRIES];

if (pkt.hdr.reset == 1):
    count[pkt.hdr.index] = 0
else:
    count[pkt.hdr.index]++
```

Using Externs in P4 – Resetting Counter

- State can be accessed exactly *1 time*
- Using RAW here

```
#define REG_READ    0
#define REG_WRITE   1
#define REG_ADD     2
// count register
@Xilinx_MaxLatency(1)
@Xilinx_ControlWidth(3)
extern void count_reg_raw(in bit<3> index, in bit<32> newVal,
                          in bit<32> incVal,in bit<8> opCode,
                          out bit<32> result);
```

Instantiate extern

```
bit<16> index = pkt.hdr.index;
bit<32> newVal; bit<32> incVal; bit<8> opCode;
if(pkt.hdr.reset == 1) {
    newVal = 0;
    incVal = 0; // not used
    opCode = REG_WRITE;
} else {
    newVal = 0; // not used
    incVal = 1;
    opCode = REG_ADD;
}
```

Set metadata for state access

```
bit<32> result; // the new value stored in count reg
count_reg_raw(index, newVal, incVal, opCode, result);
```

Single state access!

FAQs

- **Can I apply a table multiple times in my P4 Program?**
    - No (except via resubmit / recirculate)
- **Can I modify table entries from my P4 Program?**
    - No (except for direct counters)
- **What happens upon reaching the `reject` state of the parser?**
    - Architecture dependent
    - Currently not supported by P4-SDNet
- **How much of the packet can I parse?**
    - Architecture dependent

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)

UNIVERSITY OF CAMBRIDGE

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)

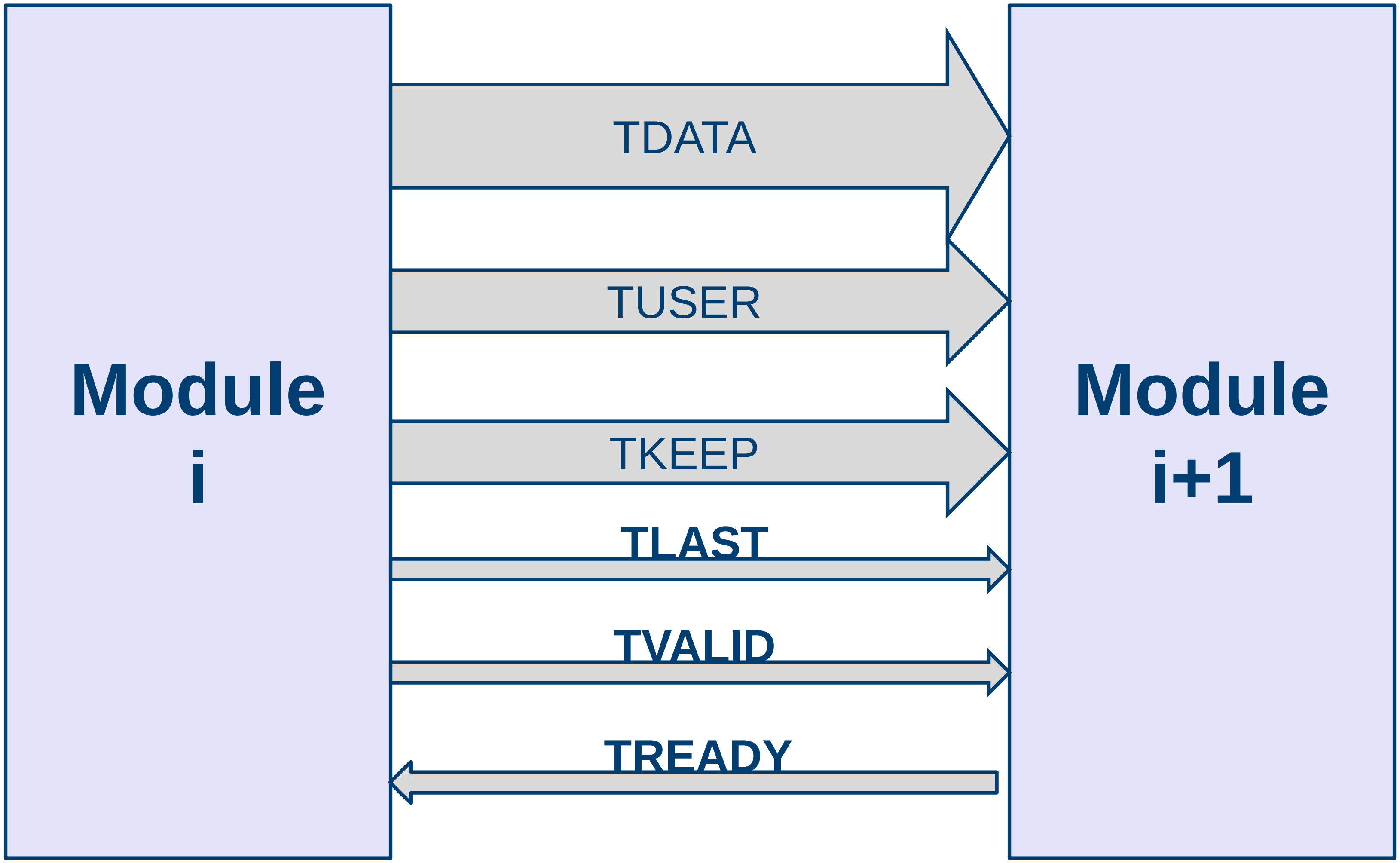
- NetFPGA cores
  - AXI-compliant
- AXI = Advanced eXtensible Interface
  - Used in ARM-based embedded systems
  - Standard interface
  - **AXI4/AXI4-Lite**: Control and status interface
  - **AXI4-Stream**: Data path interface
- Xilinx IPs and tool chains
  - Mostly AXI-compliant

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

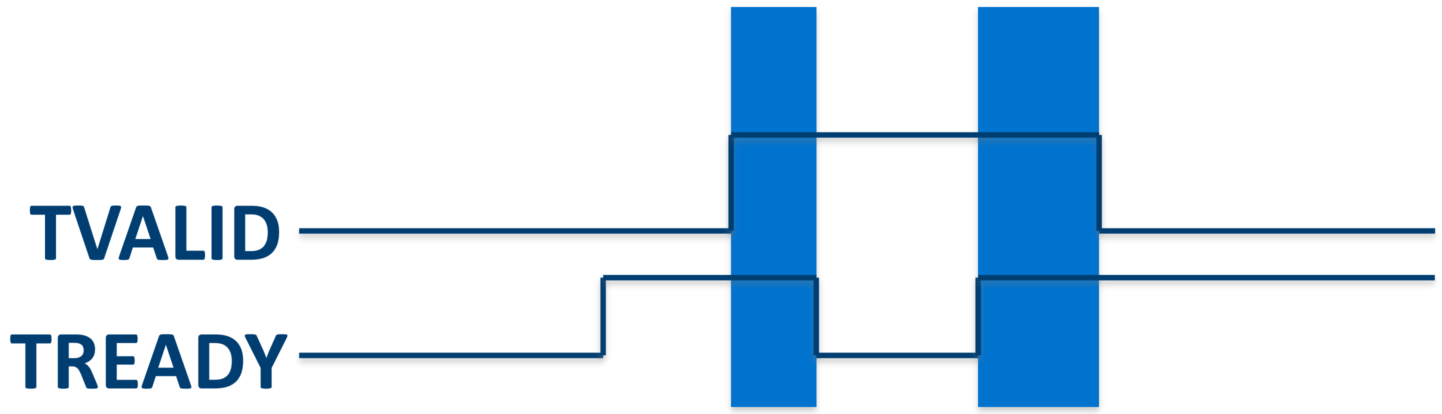
TLAST	TUSER	TKEEP	TDATA
0	V	0xFF…F	Eth Hdr
0	X	0xFF…F	IP Hdr
0	X	0xFF…F	…
1	X	0x0…1F	Last word

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.

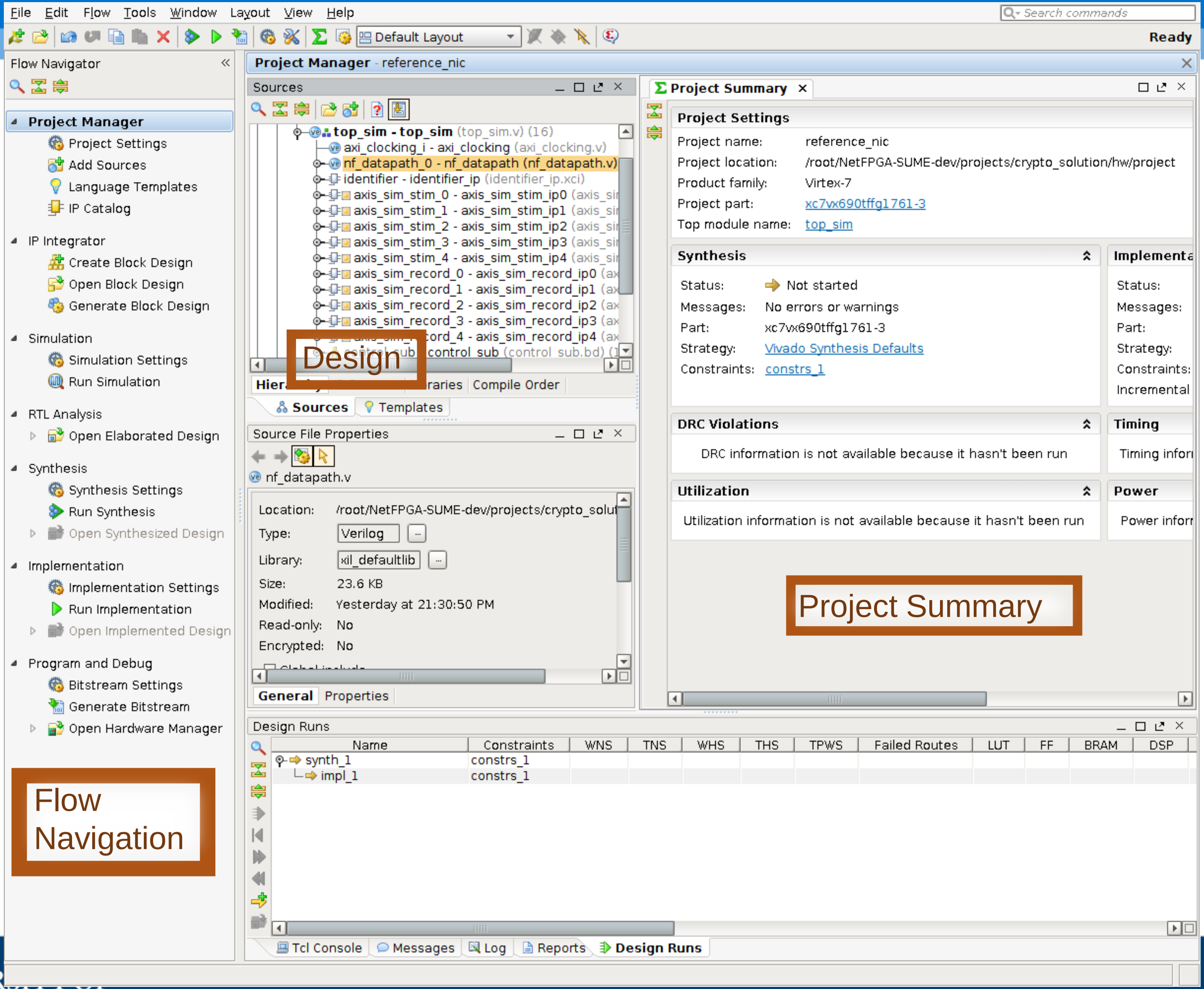
Xilinx Vivado (2)

- 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)



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
```

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

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

Implementation: 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

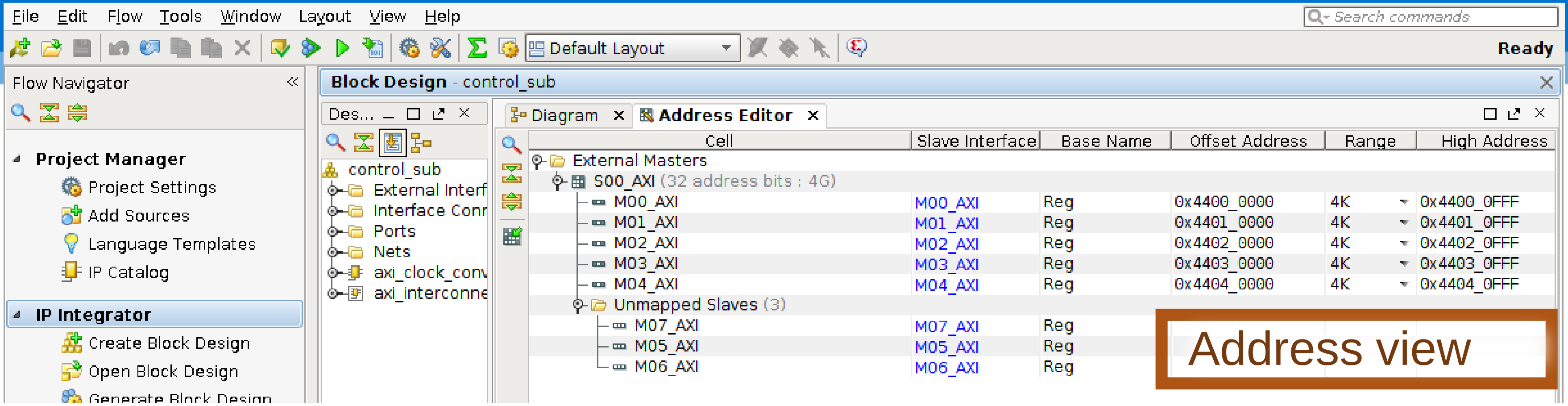
Design Runs

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

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

3. Check that make passes without errors

Register Infrastructure

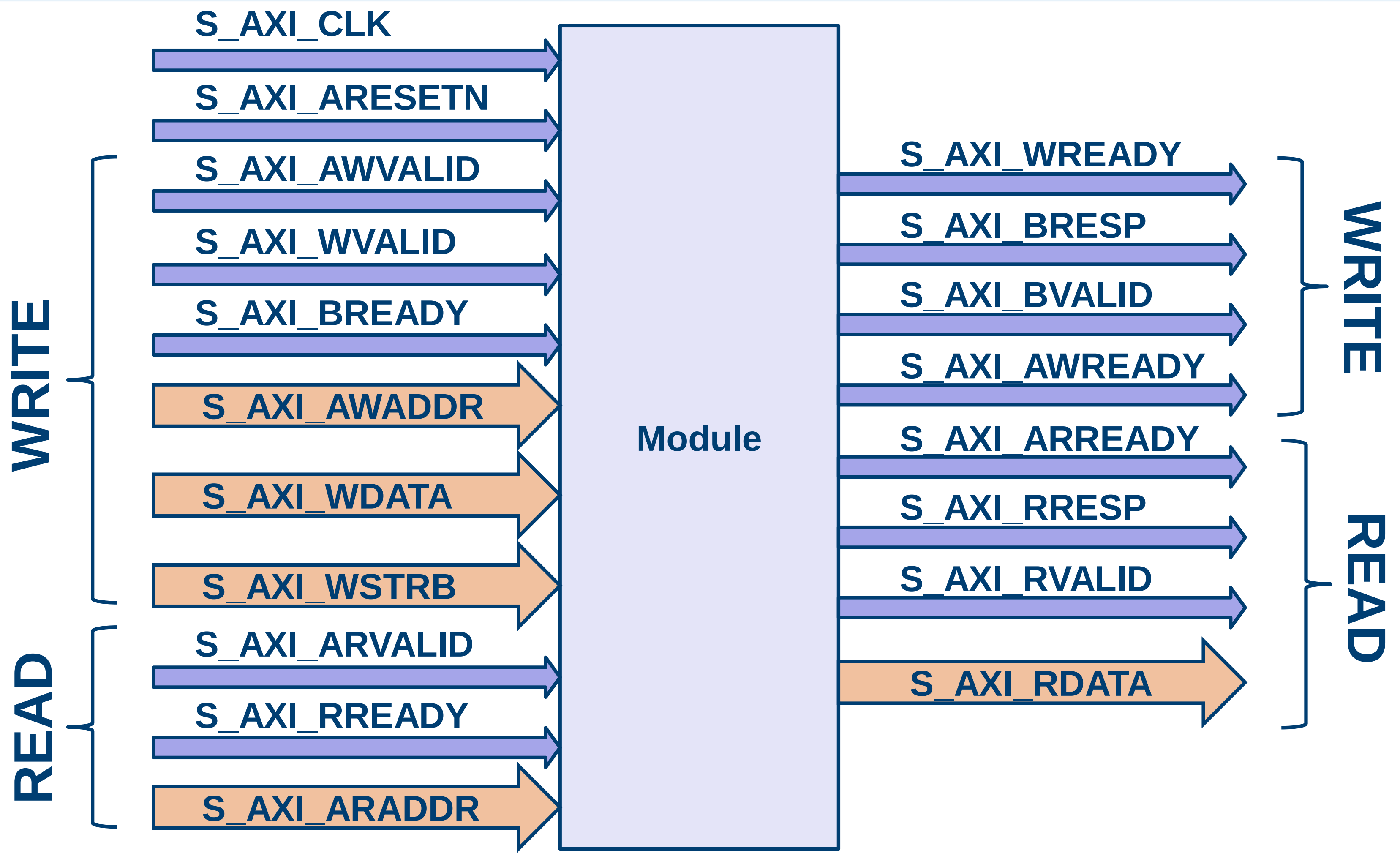
Registers

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

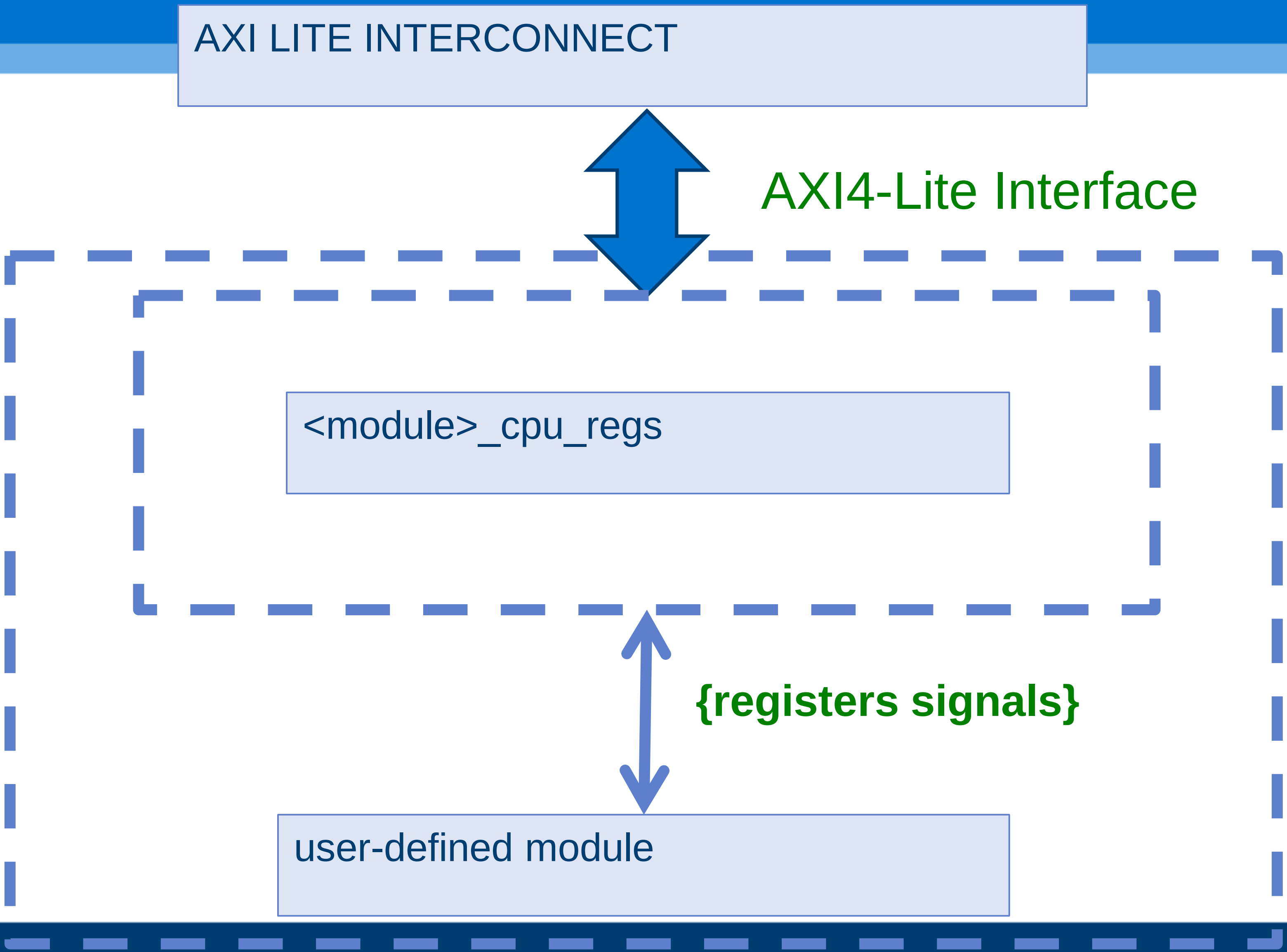
Registers bus

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

Register bus (AXI4-Lite interface)



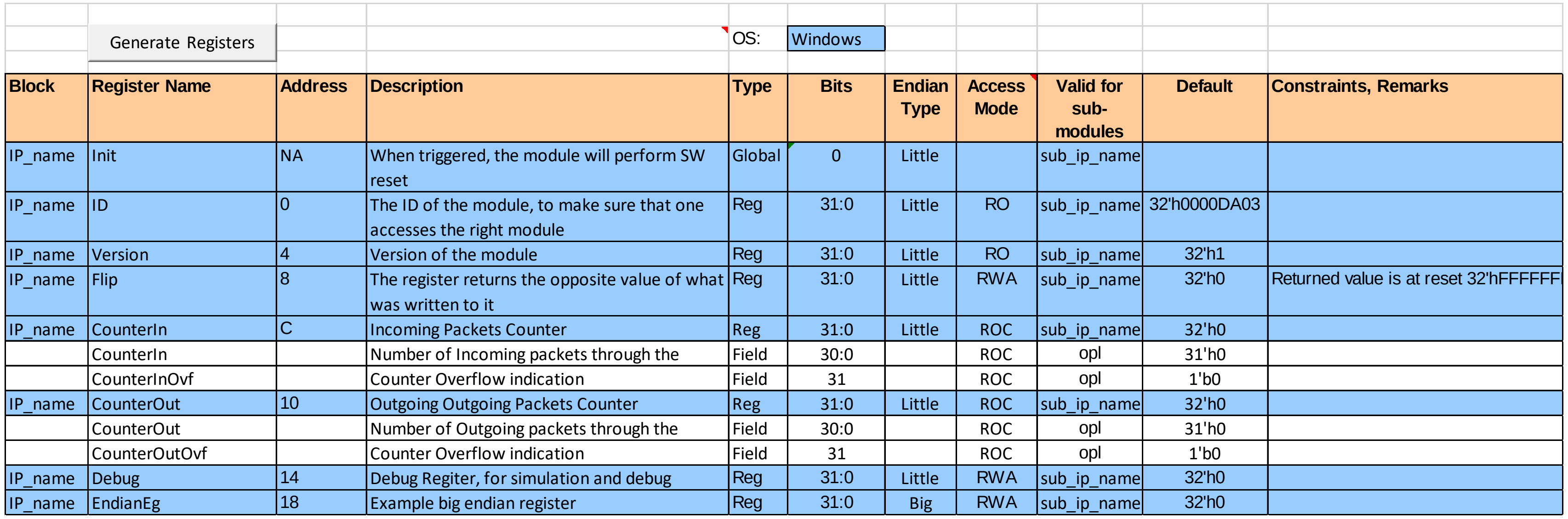
Register bus



UNIVERSITY OF CAMBRIDGE

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

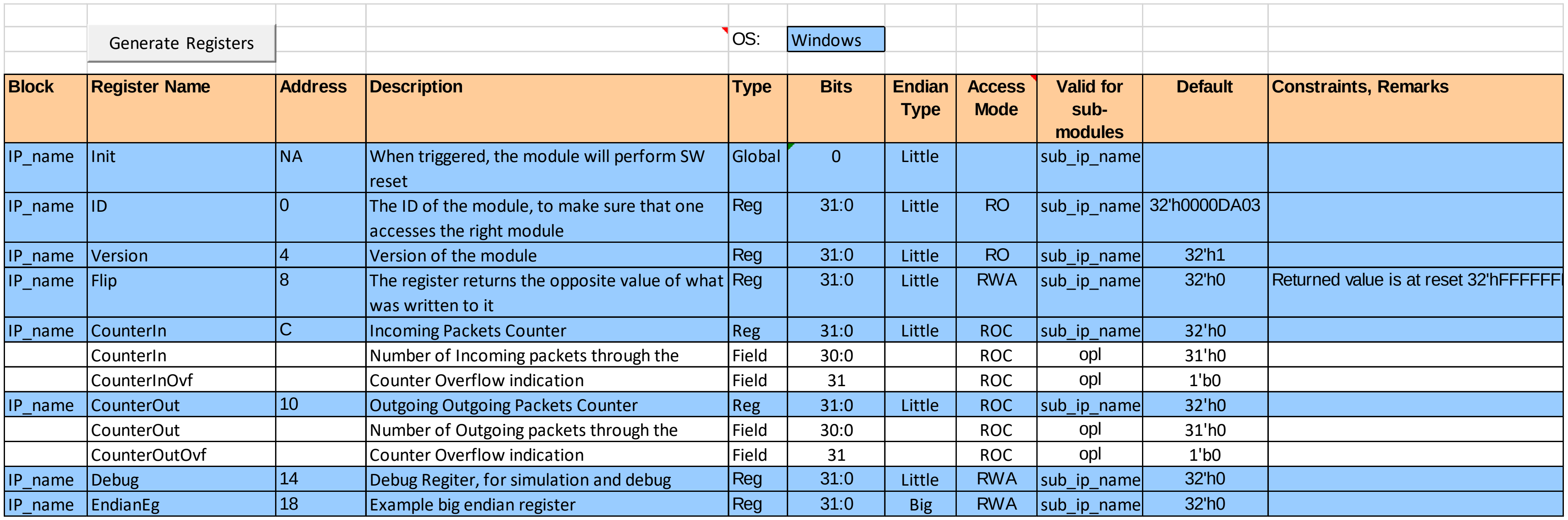


Generate Registers

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'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



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

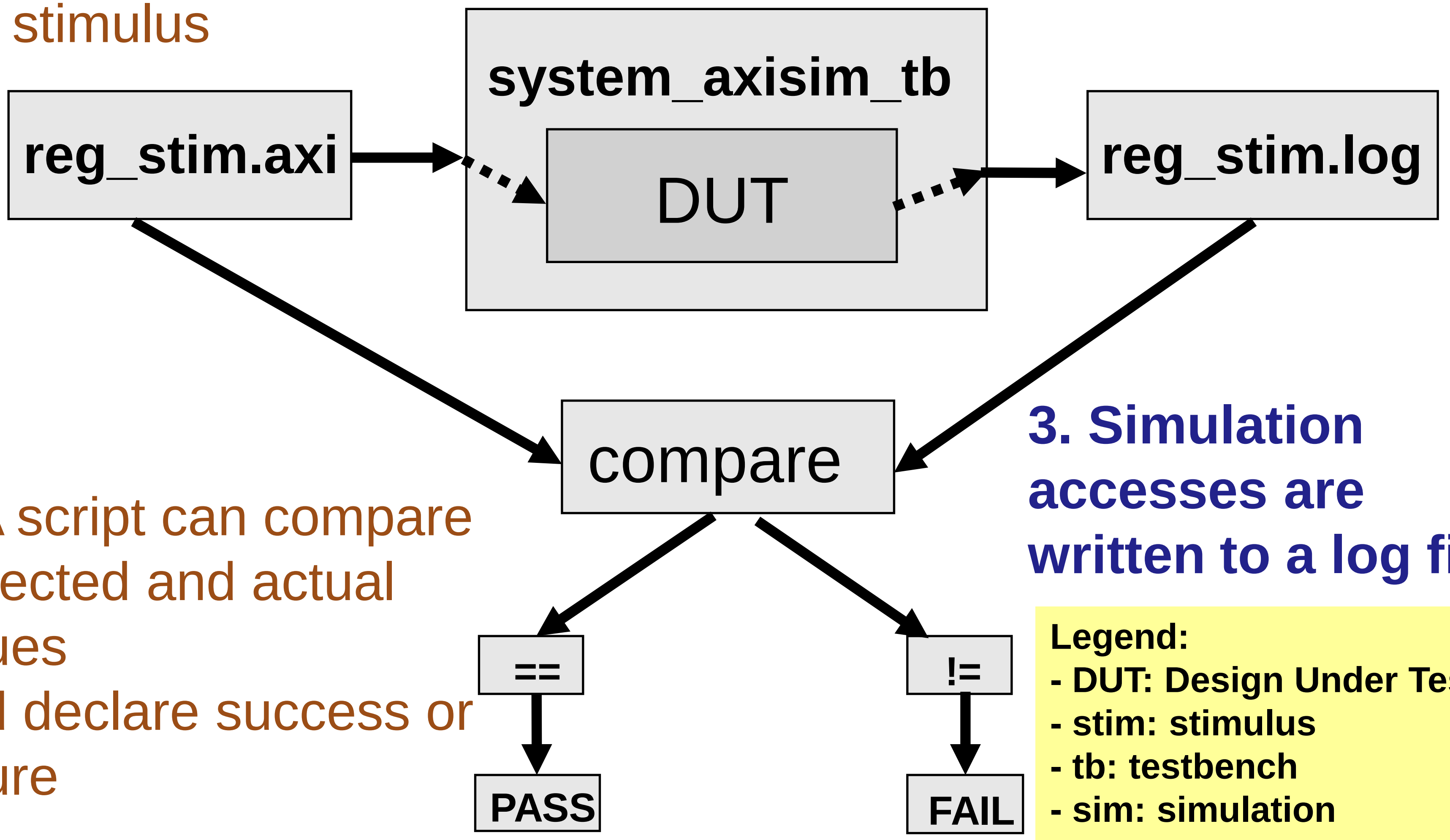
Testing Registers with Simulation

- 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

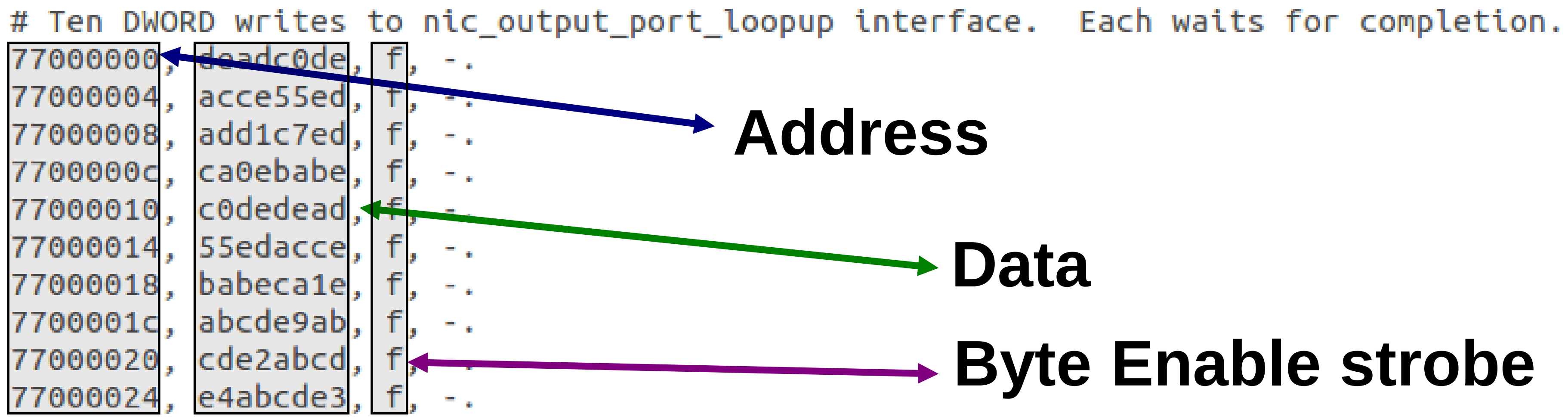
4. A script can compare expected and actual values
And declare success or failure

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 :



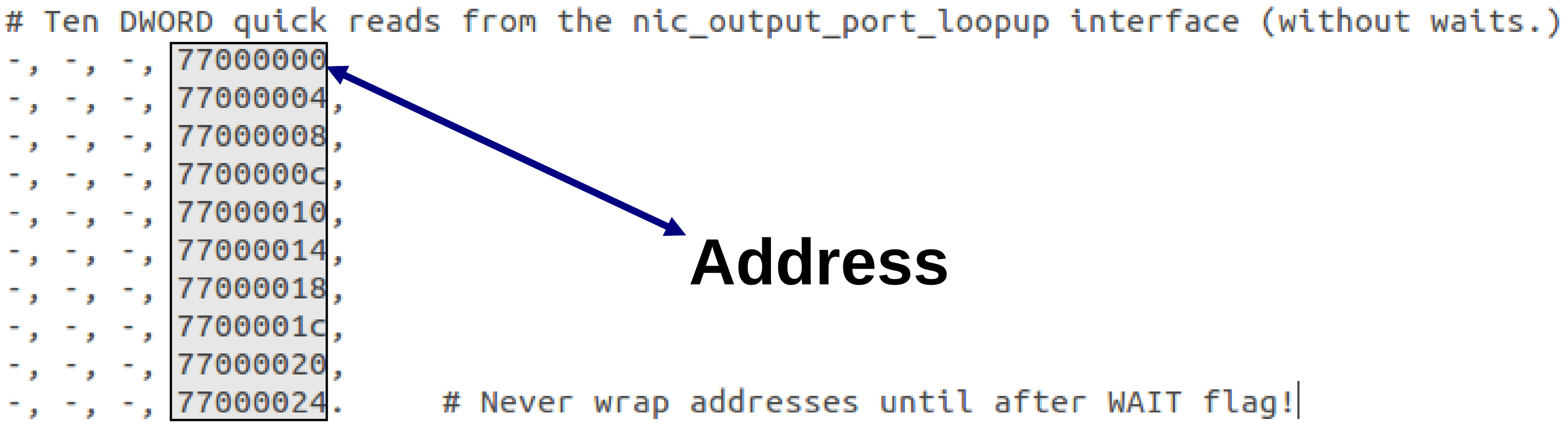
```
# Ten DWORD writes to nic_output_port_loopup interface.  Each waits for completion.
77000000, deadc0de, f, -.
77000004, acce55ed, f, -.
77000008, add1c7ed, f, -.
7700000c, ca0ebabe, f, -.
77000010, c0dedead, f, -.
77000014, 55edacce, f, -.
77000018, babeca1e, f, -.
7700001c, abcde9ab, f, -.
77000020,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

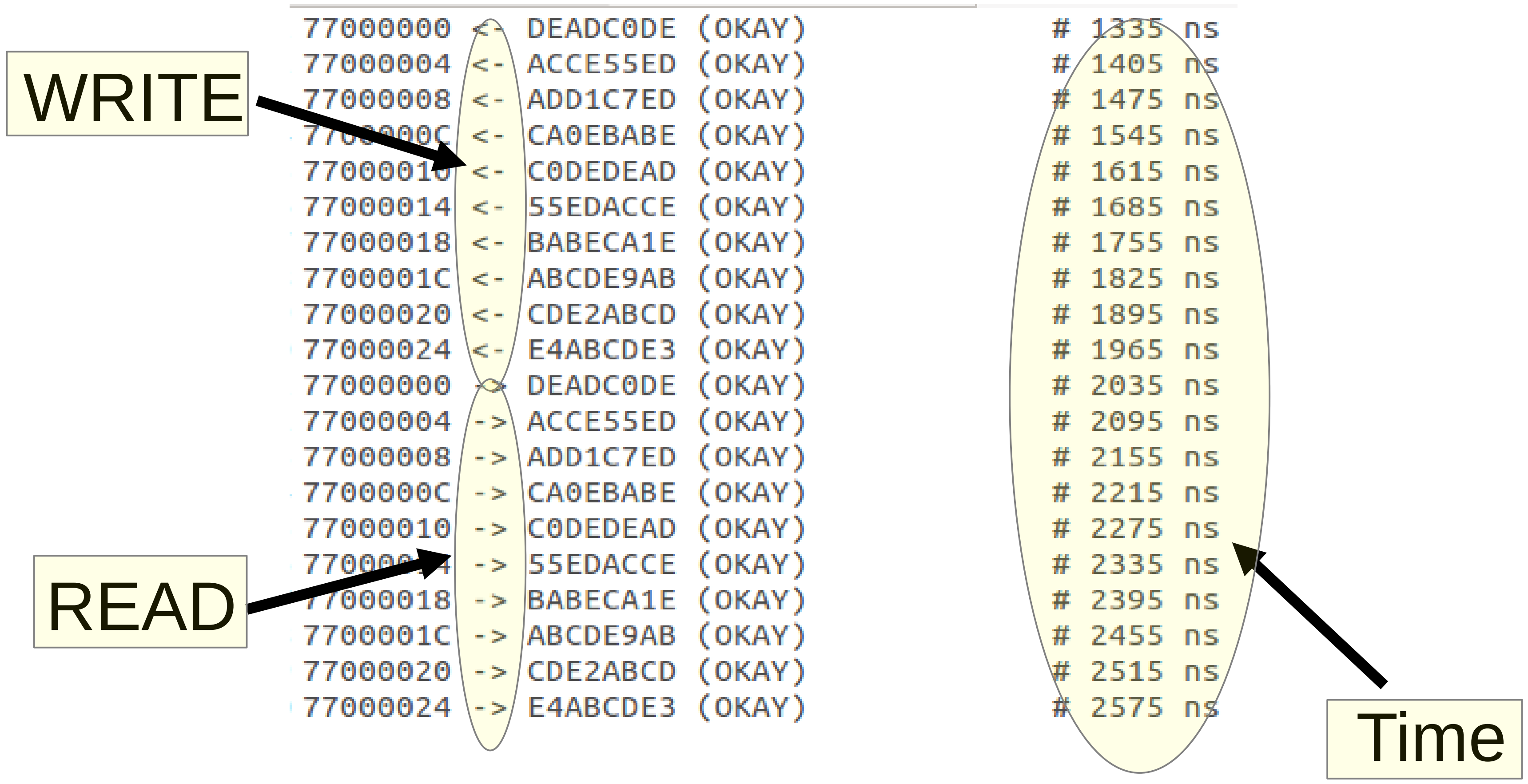


```
# Ten DWORD quick reads from the nic_output_port_loopup interface (without waits.)
-, -, -, 77000000
-, -, -, 77000004,
-, -, -, 77000008,
-, -, -, 7700000c,
-, -, -, 77000010,
-, -, -, 77000014,
-, -, -, 77000018,
-, -, -, 7700001c,
-, -, -, 77000020,
-, -, -, 77000024.      # Never wrap addresses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

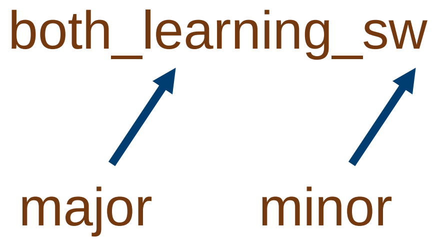
- `cd $NF_DESIGN_DIR/test/both_learning_sw`
- `vim run.py`
- "isHW" indicates HW test
- "connections/conn" file declares the physical connections

```
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>



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