Project
Project

• Today
  • P4
  • Proposal and aims

• Next Week
  • Architecture
  • Dividing the effort

• Project test infrastructure
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.
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\textsubscript{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

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4 Target</td>
<td>An embodiment of a specific hardware implementation</td>
</tr>
<tr>
<td>P4 Architecture</td>
<td>A specific set of P4-programmable components, externs, fixed components and their interfaces available to the P4 programmer</td>
</tr>
<tr>
<td>P4 Platform</td>
<td>P4 Architecture implemented on a given P4 Target</td>
</tr>
</tbody>
</table>

### Community-Developed

- **P4**<sup>_16</sup> Language
- **P4**<sup>_16</sup> Core Library

### Vendor-supplied

- Extern Libraries
- Architecture Definition
Example Architectures and Targets

V1Model

SimpleSumeSwitch

Portable Switch Architecture (PSA)

Anything
Programming a P4 Target

- **P4 Architecture Model**
- **P4 Compiler**
- **P4 Program**

**Target**
- **Target-specific configuration binary**

**Control Plane**
- **Add/remove table entries**
- **Extern control**
- **Packet-in/out**

**Data Plane**
- **Tables**
- **Extern objects**

**User supplied**
- **Vendor supplied**

**RUNTIME**
Reminder: P4 on NetFPGA (P4-NetFPGA)

P4 Program

Xilinx P4_{16} Compiler

Xilinx SDNet Tools

SimpleSumeSwitch Architecture

NetFPGA Reference Switch
P4 used to describe parser, match-action pipeline, and deparser
**V1Model Standard Metadata**

```c
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

- \_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

```
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
}
```
#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;
```c
#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;
```
#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;
#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;

<table>
<thead>
<tr>
<th>Key</th>
<th>Action Name</th>
<th>Action Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>set_egress_spec</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>set_egress_spec</td>
<td>1</td>
</tr>
</tbody>
</table>
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
/** 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

- 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

```c
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};
```
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).
/* 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
P4\textsubscript{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
• 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)

```c
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**
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

- **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)
• **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

Control Plane

Headers and Metadata

Lookup Key

Key | Action ID | Action Data
---|---|---

Default Action ID | Default Action Data

Hit/Miss Selector

Hit

Action Code

Action Execution

Directional (Data Plane) Parameters

Directionless (Action Data) Parameters

Headers and Metadata (Output)

Headers and Metadata (Input)
Example: IPv4_LPM Table

- **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

<table>
<thead>
<tr>
<th>Key</th>
<th>Action</th>
<th>Action Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.1.1/32</td>
<td>ipv4_forward</td>
<td>dstAddr=00:00:00:00:01:01; port=1</td>
</tr>
<tr>
<td>10.0.1.2/32</td>
<td>drop</td>
<td></td>
</tr>
<tr>
<td>*`</td>
<td>NoAction</td>
<td></td>
</tr>
</tbody>
</table>
IPv4_LPM Table

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

- **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

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

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

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr, bit<9> port) {
    ...
}
```
control MyIngress(inout headers hdr,
    inout metadata meta,
    inout standard_metadata_t standard_metadata) {

table ipv4_lpm {
    ...
}
apply {
    ...
    ipv4_lpm.apply();
    ...
}
}
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();
}
/** 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\textsubscript{14} almost 1/3 of all the constructs were dedicated to specialized processing
  ◦ In P4\textsubscript{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:

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

- Stateless Externs

<table>
<thead>
<tr>
<th>Extern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Checksum</td>
<td>Given an IP header, compute IP checksum</td>
</tr>
<tr>
<td>LRC</td>
<td>Longitudinal redundancy check, simple hash function</td>
</tr>
<tr>
<td>timestamp</td>
<td>Generate timestamp (granularity of 5 ns)</td>
</tr>
</tbody>
</table>
Packet processing pseudo code:

```python
count[NUM_ENTRIES];

if (pkt.hdr.reset == 1):
    count[pkt.hdr.index] = 0
else:
    count[pkt.hdr.index]++
```
Using Externs in P4 – Resetting Counter

```c
#define REG_READ    0
#define REG_WRITE   1
#define REG_ADD     2

// count register
#define Xilinx_MaxLatency(1)
#define 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);

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;
}

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

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

Instantiate extern
Set metadata for state access
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)
# Reusable logic (IP cores)

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

• Simulation and Debugging
  • built on industry standard Xilinx “xSim” simulator and “Scapy”
  • Python scripts for stimuli construction and verification
• xSim
  • a High Level Description (HDL) simulator
  • performs functional and timing simulations for embedded, VHDL, Verilog and mixed designs

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

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

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

• “IP Core” in Vivado
  • Standalone Module
  • Configurable and reuseable

• HDL (Verilog/VHDL) + TCL files

• Examples:
  • 10G Port
  • SRAM Controller
  • NIC Output port lookup
HDL (Verilog)

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

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

<table>
<thead>
<tr>
<th>AXI4-Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDATA</td>
<td>Data Stream</td>
</tr>
<tr>
<td>TKEEP</td>
<td>Marks qualified bytes (i.e. byte enable)</td>
</tr>
<tr>
<td>TVALID</td>
<td>Valid Indication</td>
</tr>
<tr>
<td>TREADY</td>
<td>Flow control indication</td>
</tr>
<tr>
<td>TLAST</td>
<td>End of packet/burst indication</td>
</tr>
<tr>
<td>TUSER</td>
<td>Out of band metadata</td>
</tr>
<tr>
<td>TLAST</td>
<td>TUSER</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>V</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Position</td>
<td>Content</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>[15:0]</td>
<td>length of the packet in bytes</td>
</tr>
<tr>
<td>[23:16]</td>
<td>source port: one-hot encoded</td>
</tr>
<tr>
<td>[31:24]</td>
<td>destination port: one-hot encoded</td>
</tr>
<tr>
<td>[127:32]</td>
<td>6 user defined slots, 16bit each</td>
</tr>
</tbody>
</table>
TVALID/TREADY Signal timing

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

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

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

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

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

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

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

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

Preparing a module:

1. cd $IP_FOLDER/<ip name>

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

3. make

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

3. Check that make passes without errors
Register Infrastructure
Registers

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

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

AXI LITE INTERCONNECT

AXI4-Lite Interface

<module>_cpu_regs

{registers signals}

user-defined module
Registers – Module generation

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

<table>
<thead>
<tr>
<th>Block</th>
<th>Register Name</th>
<th>Address</th>
<th>Description</th>
<th>Type</th>
<th>Bits</th>
<th>Endian Type</th>
<th>Access Mode</th>
<th>Valid for sub-modules</th>
<th>Default</th>
<th>Constraints, Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_name</td>
<td>Init</td>
<td>NA</td>
<td>When triggered, the module will perform SW reset</td>
<td>Global</td>
<td>0</td>
<td>Little</td>
<td></td>
<td>sub_ip_name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>ID</td>
<td>0</td>
<td>The ID of the module, to make sure that one accesses the right module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name 32'h0000DA03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Version</td>
<td>4</td>
<td>Version of the module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name 32'h1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Flip</td>
<td>8</td>
<td>The register returns the opposite value of what was written to it</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name 32'h0</td>
<td>Returned value is at reset 32'hFFFFFFF</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterIn</td>
<td>C</td>
<td>Incoming Packets Counter</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name 32'h00</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CounterInOvf</td>
<td></td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opl</td>
<td>31'h0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOut</td>
<td>10</td>
<td>Outgoing Outgoing Packets Counter</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name 32'h0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CounterOutOvf</td>
<td></td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opl</td>
<td>1'b0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Debug</td>
<td>14</td>
<td>Debug Register, for simulation and debug</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name 32'h0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>EndianEg</td>
<td>18</td>
<td>Example big endian register</td>
<td>Reg</td>
<td>31:0</td>
<td>Big</td>
<td>RWA</td>
<td>sub_ip_name 32'h0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block</td>
<td>Register Name</td>
<td>Address</td>
<td>Description</td>
<td>Type</td>
<td>Bits</td>
<td>Endian Type</td>
<td>Access Mode</td>
<td>Valid for sub-modules</td>
<td>Default</td>
<td>Constraints, Remarks</td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>IP_name</td>
<td>Init</td>
<td>NA</td>
<td>When triggered, the module will perform SW reset</td>
<td>Global</td>
<td>0</td>
<td>Little</td>
<td></td>
<td>sub_ip_name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>ID</td>
<td>0</td>
<td>The ID of the module, to make sure that one accesses the right module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h0000DA03</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Version</td>
<td>4</td>
<td>Version of the module</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RO</td>
<td>sub_ip_name</td>
<td>32'h1</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Flip</td>
<td>8</td>
<td>The register returns the opposite value of what was written to it</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td>Returned value is at reset 32'hFFFFFFF</td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterIn</td>
<td>C</td>
<td>Incoming Packets Counter</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CounterInOvf</td>
<td></td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opl</td>
<td>31'h0</td>
<td>1'b0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>CounterOut</td>
<td>10</td>
<td>Outgoing Packets Counter</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>ROC</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CounterOutOvf</td>
<td></td>
<td>Counter Overflow indication</td>
<td>Field</td>
<td>31</td>
<td>ROC</td>
<td>opl</td>
<td>31'h0</td>
<td>1'b0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>Debug</td>
<td>14</td>
<td>Debug Register, for simulation and debug</td>
<td>Reg</td>
<td>31:0</td>
<td>Little</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
<tr>
<td>IP_name</td>
<td>EndianEg</td>
<td>18</td>
<td>Example big endian register</td>
<td>Reg</td>
<td>31:0</td>
<td>Big</td>
<td>RWA</td>
<td>sub_ip_name</td>
<td>32'h0</td>
<td></td>
</tr>
</tbody>
</table>
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.
Same contents as <module>_cpu_regs_defines.v, but in different formats, used by software, build and test harness:

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

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

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

• Usage examples:

```verilog
always @(posedge axi_aclk)
    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_registers.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

```
reg_stim.axi
```

2. The testbench executes the stimulus

```
system_axisim_tb
```

3. Simulation accesses are written to a log file

```
reg_stim.log
```

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

```
compare
```

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

• An example of write format:

```
# Ten DWORD writes to nic_output_port_loopup interface. Each waits for completion.
77000000, deadc0de, f, -
77000004, acce55ed, f, -
77000008, add1c7ed, f, -
770000c, ca0ebabe, f, -
7700010, c0dedead, f, -
7700014, 55edacce, f, -
7700018, babeca1e, f, -
770001c, abcde9ab, f, -
7700020, cde2abcd, f, -
7700024, e4abcde3, f, -
```

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

less reg_stim.axi

- An example read format:

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

# 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
Section II: Testing Hardware
• To synthesize your project:

```bash
  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
```

```python
major         minor
```
Running Hardware Tests

• Having problems?

• Take advantage of the wiki!

• Detailed explanations

• Tips for debug