Hands-on Labs

Instructors:
Stephen Ibanez, Gordon Brebner, Robert Halstead, Chris Neely, Tushar Swamy, Sean Choi
Outline

• P4-NetFPGA Workflow Overview
• P4 Compilation Using Xilinx P4-SDNet
• P4-NetFPGA Workflow Details
• Tutorial Assignments
P4-NetFPGA Workflow Overview
So what is NetFPGA?

NetFPGA = Networked FPGA

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

SUME

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

Stanford University
General Process for Programming a P4 Target

A P4-programmable switch differs from a traditional switch in two ways:

- The switch data plane is no longer fixed; P4 programs describe the data plane functionality. The data plane is configured at switch initialization time based on the P4 functionality (shown by the long red arrow). The data plane has no built-in knowledge of existing protocols.
- The control plane continues to interact with the data plane using the same channels; however, the set of tables and other objects driving the behavior of the data plane is no longer fixed, since the P4 programmer specifies it after the switch has been manufactured. The P4 compiler generates the API that the control plane uses to communicate with the data plane from the P4 program.

Hence, P4 itself is protocol independent but allows programmers to express a rich set of data plane behaviors and protocols.

The core abstractions provided by the P4 language are:

- **Header definitions** describe the format (the set of fields and their sizes) of each header within a packet.
- **Parsers** describe the permitted header sequences within received packets, how to identify those header sequences, and the headers and fields to extract from packets.
- **Tables** associate user-defined keys with actions. P4 tables generalize traditional switch tables; they can be used to implement routing tables, flow lookup tables, access-control lists, and other user-defined table types, including complex multivariable decisions.
- **Actions** are code fragments that describe how packet header fields and metadata are manipulated. Actions can also include data, which can be supplied by the control-plane at run time.
- **Match-action units** perform the following sequence of operations:
  - Construct lookup keys from packet fields or computed metadata,
  - Perform table lookup using the constructed key, choosing an action (including the associated data) to execute
  - Finally, execute the selected action

**Control flow** expresses an imperative program describing the data-dependent packet processing.
SimpleSumeSwitch Architecture Model for SUME Target

P4 used to describe parser, match-action pipeline, and deparser
Standard SUME Metadata in Architecture Model

/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> pkt_len; // unsigned int
    port_t src_port; // one-hot encoded
    port_t dst_port; // one-hot encoded
    bit<8> drop;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    digest_metadata_t digest_data;
}

// digest metadata to send to CPU
struct digest_metadata_t {
    bit<8> src_port;
    bit<48> eth_src_addr;
    bit<24> unused;
}
P4 Architecture Model Plugs Into SUME Reference Switch
P4 Architecture Model Plugs Into SUME Reference Switch
P4 Compilation Using Xilinx P4-SDNet
Xilinx SDNet programmable packet processor
(www.xilinx.com/sdnet)

Headline feature set, uniquely enabled by FPGA ‘white box hardware’ target:

- Scalable 1G to 100G line rate performance
- Exact-fit hardware for reduced cost and power
- Domain-specific programming abstraction
- Firmware for run time programmability
SDNet Design Flow and Use Model using PX language

Packet processing specification

Compiler

HDL description

Xilinx Vivado tools

FPGA bitstream

Customized packet processing data path

PX domain-specific programming language describing functions in packet-oriented terms

Throughput, latency, resource, programmability requirements

Firmware

Update firmware when required

Configure FPGA

Configure FPGA

FPGA platform

Tailored packet processor

Stanford University
Xilinx P4-SDNet Design Flow
Included in SDNet 2016.4 release, February 2017

Xilinx P4 Compiler

% p4c-fpga example.p4

Xilinx SDNet Compiler

% sdnet example.sdnet

Verification Environment

Top level Verilog wrapper
Verilog Engines (Encrypted)
System Verilog Testbench
Lookup Engine C++ Drivers
High level C++ Testbench

VIVADO
HLx Editions

example.img

Stanford University
P4-SDNet Compilation in P4-NetFPGA Workflow

- User P4 code is compiled with respect to SimpleSumeSwitch Architecture Model:
  - Code for Parser, Match-Action Pipeline, and Deparser
- Compiler outputs Verilog module for whole P4-described system
- Module has standard AXI-S packet input/output interfaces
- Output is engineered for 100G rate >> SUME switch aggregate 40G rate
- Supports P4 extern feature for user defined logic
P4-NetFPGA Workflow Details
P4-NetFPGA Workflow

1. Write P4 program
2. Write python gen_testdata.py script
3. Compile to verilog / generate API & CLI tools
   $ make
4. Run initial SDNet simulation
   $ ./vivado_sim.bash
5. Install SDNet output as SUME library core
   $ make install_sdnet
6. Run NetFPGA simulation
   $ ./nf_test sim –major switch –minor default
7. Build bitstream
   $ make
8. Test the hardware
Directory Structure of $SUME_FOLDER

NetFPGA-SUME-SDNet/
  |- contrib-projects/
  |   |- sume-sdnet-switch/ → the main directory for P4 dev
  |
  |- lib/ → contains all of the SUME IP cores
  |
  |- tools/ → various NetFPGA scripts for test infra.
  |
  |- Makefile → builds all of the SUME IP cores
Directory Structure of $SUME_SDNET

sume-sdnet-switch/
  |
  |- bin/ → scripts used to automate workflow
  |
  |- templates/ → templates for externs, wrapper module,
  |                  CLI tools, new projects
  |
  |- projects/ → all of the P4 project directories
  |  |- switch_calc/
Directory Structure of $P4_PROJECT_DIR

switch_calc/
  |
  |- src/ → P4 source files
  |
  |- testdata/ → scripts to generate testdata used for verifying functionality of P4 program
  |
  |- simple_sume_switch/ → main SUME project directory, top level HDL files and SUME sim scripts
  |
  |- sw/ → populated with API files and CLI tools
  |
  |- nf_sume_sdnet_ip/ → SDNet output directory
API & Interactive CLI Tool Generation

- Both Python API and C API
  - Manipulate tables and stateful elements in P4 switch
  - Used by control-plane program
- CLI tool
  - Useful debugging feature
  - Query various compile-time information
  - Interact directly with tables and stateful elements in real time
P4-NetFPGA Extern Function Library

- Verilog modules invoked from within P4 programs

Examples:
- Atoms for writing stateful P4 programs based on packet transactions (SIGCOMM 2016)

<table>
<thead>
<tr>
<th>Atom</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>
</tbody>
</table>

- LRC16 checksum function
- Timestamp generation
- More to come…
Using Atom 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]++
```
# define REG_READ 0
# define REG_WRITE 1
# define REG_ADD 2

// count register
@CYCLES(1)
@CONTROLBITS(16)
extern void count_reg_raw(in bit<16> index,
                          in bit<32> newVal,
                          in bit<32> incVal,
                          in bit<8> opCode,
                          in 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
count_reg_raw(index, newVal, incVal, opCode, result);
Tutorial Assignments

https://bitbucket.org/sibanez/netfpga-sume-sdnet/wiki/Tutorial%20Assignments

http://tinyurl.com/p4-netfpga-dev-day

Username: p4user
Password: p4Rocks!
Assignment 1: Switch as a Calculator
Switch as Calculator

Supported Operations:
- **ADD** – add two operands
- **SUBTRACT** – subtract two operands
- **ADD_REG** – add constant to current value in register
- **SET_REG** – overwrite the current value of the register
- **LOOKUP** – Lookup the given key in the table

```c
header Calc_h {
    bit<32> op1;
    bit<8> opCode;
    bit<32> op2;
    bit<32> result;
}
```
Switch as Calculator

User PC

- Ethernet
  - DST: MAC1
  - SRC: MAC2
  - Type: CALC_TYPE

- Calc
  - op1: 1
  - opCode: ADD
  - op2: 2
  - result: 0
  - Payload...

NetFPGA SUME

Stanford University
Switch as Calculator

User PC

NetFPGA SUME

- DST: MAC1
- SRC: MAC2
- Type: CALC_TYPE

Calc

- op1: 1
- opCode: ADD
- op2: 2
- result: 3

Payload...

Stanford University
Switch as Calculator

User PC

NetFPGA SUME

Ethernet

DST: MAC2
SRC: MAC1
Type: CALC_TYPE

Calc

op1: 1
opCode: ADD
op2: 2
result: 3
Payload...

Stanford University
Switch Calc Operations

**ADD**

```
+     
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op2</td>
</tr>
</tbody>
</table>
```

```
+     
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
result
```

**SUB**

```
-     
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op1</td>
<td>op2</td>
</tr>
</tbody>
</table>
```

```
-     
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
result: op1-op2
```

**ADD_REG**

```
+     
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
+     
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
result
```

**SET_REG**

```
const[op1]  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
const[op1]  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**LOOKUP**

```
<table>
<thead>
<tr>
<th>key</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>16^2</td>
</tr>
<tr>
<td>3</td>
<td>16^3</td>
</tr>
</tbody>
</table>
```

```
result: val
```

Stanford University
Assignment 2: TCP Monitor
TCP Monitor

- Practice writing stateful P4 programs for NetFPGA SUME
- Compute TCP flow size distribution in the data-plane
- Flow is determined by 5-tuple and delimited by SYN/FIN
- Fine grained flow monitoring capabilities with P4

Flow Size Distribution

![Flow Size Distribution Chart]

# Flows

Flow Size (bytes)
### byte_cnt

<table>
<thead>
<tr>
<th>1K</th>
</tr>
</thead>
</table>

### distribution

<table>
<thead>
<tr>
<th>0-16K</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-32K</td>
<td>0</td>
</tr>
<tr>
<td>32-48K</td>
<td>0</td>
</tr>
<tr>
<td>48-64K</td>
<td>0</td>
</tr>
<tr>
<td>64-80K</td>
<td>0</td>
</tr>
<tr>
<td>80-96K</td>
<td>0</td>
</tr>
<tr>
<td>96-112K</td>
<td>0</td>
</tr>
<tr>
<td>&gt;112K</td>
<td>0</td>
</tr>
</tbody>
</table>
Assignment 3: In-band Network Telemetry (INT)
In-band Network Telemetry (INT)

- One of the most popular applications for programmable data-planes
- All about gaining more visibility into network
- Basic idea:
  - Source requests each switch along path to insert some desired metadata into packet (using a bitmask)
- Example metadata:
  - Switch ID
  - Ingress Port
  - Egress Port
  - Timestamp
  - Queue Occupancy
In-band Network Telemetry (INT)

- Bitmask format (5 bits):
  `<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>`

![Diagram showing payload, INT bitmask, and Ethernet](image)
In-band Network Telemetry (INT)

- Bitmask format (5 bits):
  
  `<SWITCH_ID><INGRESS_PORT><Q_SIZE><TSTAMP><EGRESS_PORT>`
P4-NetFPGA Example
Learning by example – L2 Learning Switch

- Parses Ethernet frames
- Forwards based on Ethernet destination address
- Frame broadcast (with ingress port filtering) if address not in forwarding database
- Learns based on Ethernet source address
- If source address is unknown, the address and the source port are sent to the control-plane (which will add an entry to the forwarding database)
Learning Switch – Header definitions

```c
// standard Ethernet header
header Ethernet_h {
    EthernetAddress dstAddr;
    EthernetAddress srcAddr;
    bit<16> etherType;
}

// IPv4 header without options
header IPv4_h {
    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;
    IPv4Address srcAddr;
    IPv4Address dstAddr;
}
```

// List of all recognized headers
```c
struct Parsed_packet {
    Ethernet_h ethernet;
    IPv4_h ip;
}
```

- Explicit specification of headers, fields, and their bit widths
- The headers that can be parsed, manipulated, or created by the switch
Learning Switch – Parser

// Parser Implementation
parser TopParser(packet_in b, 
    out Parsed_packet p, 
    out user_metadata_t user_metadata, 
    inout sume_metadata_t sume_metadata) {

  state start {
    b.extract(p.ethernet);
    transition select(p.ethernet.etherType) {
      IPV4_TYPE: parse_ipv4;
      default: reject;
    }
  }

  state parse_ipv4 {
    b.extract(p.ip);
    transition accept;
  }

  // State machine
  // Extracts headers from incoming packets
  // Produces parsed representation of packet for use in match-action pipeline
Learning Switch – Control Flow

apply {
    // try to forward based on destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}
apply {
  // try to forward based on
  // destination Ethernet address
  if (!forward.apply().hit) {
    // miss in forwarding table
    broadcast.apply();
  }

  // check if src Ethernet address
  // is in the forwarding database
  if (!smac.apply().hit) {
    // unknown source MAC address
    send_to_control();
  }
}

action set_output_port(port_t port) {
  sume_metadata.dst_port = port;
}

table forward() {
  key = {
    headers.ethernet.dstAddr: exact;
  }

  actions = {
    set_output_port;
  }
  size = 64;
  default_action = nop;
}

◆ Tables:
◆ Which fields (header and/or metadata) to match on
◆ List of valid actions that can be applied
◆ Resources to allocate to table
◆ Match types: exact, ternary, LPM
apply {
  // try to forward based on destination Ethernet address
  if (!forward.apply().hit) {
    // miss in forwarding table
    broadcast.apply();
  }

  // check if src Ethernet address is in the forwarding database
  if (!smac.apply().hit) {
    // unknown source MAC address
    send_to_control();
  }
}

◆ Actions:
  ◆ Modify header/metadata fields
  ◆ Parameters may be provided by data-plane or control-plane

action set_output_port(port_t port) {
  sume_metadata.dst_port = port;
}

table forward() {
  key = {
    headers.ethernet.dstAddr: exact;
  }

  actions = {
    set_output_port;
  }
  size = 64;
  default_action = nop;
}
Learning Switch – Control Flow

```cpp
apply {
  // try to forward based on
  // destination Ethernet address
  if (!forward.apply() .hit) {
    // miss in forwarding table
    broadcast.apply();
  }

  // check if src Ethernet address
  // is in the forwarding database
  if (!smac.apply() .hit) {
    // unknown source MAC address
    send_to_control();
  }
}
```

```cpp
action set_broadcast(port_t port) {
  sume_metadata.dst_port = port;
}

table broadcast() {
  key = {
    sume_metadata.src_port: exact;
  }

  actions = {
    set_broadcast;
    nop;
  }

  size = 64;
  default_action = nop;
}
```
Learning Switch – Control Flow

```plaintext
apply {
    // try to forward based on destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}

table smac() {
    key = {
        headers.ethernet.srcAddr: exact;
    }
    actions = {
        nop;
    }
    size = 64;
    default_action = nop;
}
```
apply {
    // try to forward based on
    // destination Ethernet address
    if (!forward.apply().hit) {
        // miss in forwarding table
        broadcast.apply();
    }

    // check if src Ethernet address
    // is in the forwarding database
    if (!smac.apply().hit) {
        // unknown source MAC address
        send_to_control();
    }
}

action send_to_control() {
    sume_metadata.digest_data.src_port = sume_metadata.src_port;
    sume_metadata.digest_data.eth_src_addr = headers.ethernet.srcAddr;
    sume_metadata.sendDigToCPU = 1;
}
Learning Switch – Deparser

// Deparser Implementation
control TopDeparser(packet_out b,
    in Parsed_packet p,
    in user_metadata_t user_metadata,
    inout sume_metadata_t sume_metadata) {

    apply {
        b.emit(p.ethernet);
        b.emit(p.ip);
    }
}

◆ Reconstruct the packet
◆ May append headers or arbitrary data
Learning Switch – Control-Plane

```python
def learn_digest(pkt):
    dig_pkt = Digest_data(str(pkt))
    add_to_tables(dig_pkt)

def add_to_tables(dig_pkt):
    src_port = dig_pkt.src_port
    src_addr = dig_pkt.eth_src_addr
    (found, val) = table_cam_read_entry('forward', [src_addr])
    if (found == 'False'):
        table_cam_add_entry('forward', [src_addr], 'set_output_port', [src_port])
        table_cam_add_entry('smac', [src_addr], 'nop', [])

def main():
    sniff(iface=DMA_IFACE, prn=learn_digest, count=0)
```

- Auto generated Python API
- Some other API functions:
  - `table_cam_delete_entry()`
  - `table_cam_get_size()`
  - `reg_read()`
  - `reg_write()`
- C API also available