# API for Configuration of P4 Pipeline

## Overview

P4 program allows custom definition of packet processing actions. These actions are searched based on rules which are stored in Match+Action tables. Each rule is identified by a unique key which is constructed from extracted packet headers. A structure of the key is described in a P4 table definition (in `reads` section of a `table` definition). The example of such definition is following:

```
table table_ipv4_filter {
        reads {
             ipv4.srcaddr      : lpm;
             tcp.dstport       : exact;}
        action {
             _drop;
             add_vlan_tag;}
}
```

```
action add_vlan_tag(vid) {
    add_header(vlan);
    modify_field(vlan.ethtype,...);
    // ... Implementation of
    //     the action ...
 }
```

The provided example introduces a table named `table_ipv4_filter` which uses a two-element key: IPv4 source address field (using the Longest Prefix Match) and destination port of TCP protocol (using the exact match). The `action` section contains a list of available actions which can be performed on incoming packets upon rule match. Each action can have one or more parameters of different bit length.

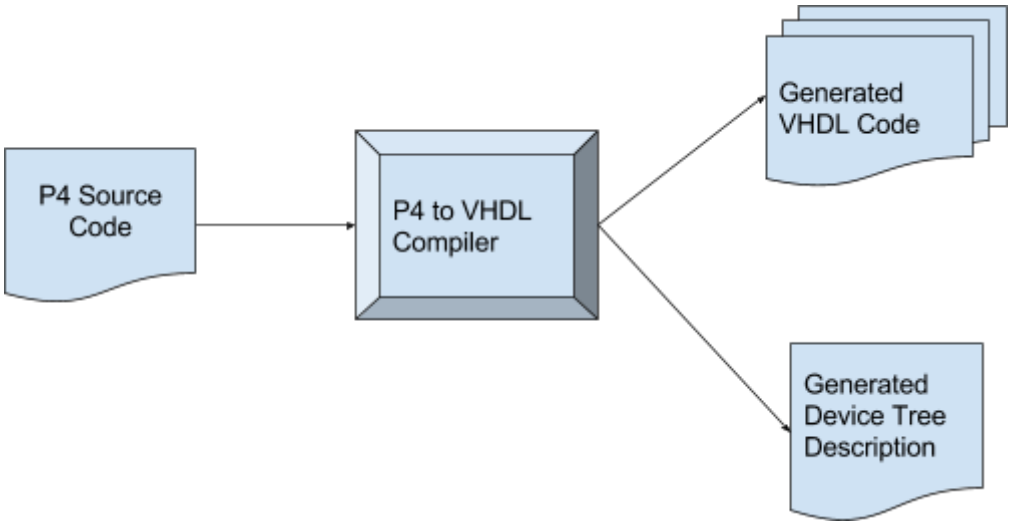


Figure 1 - Main outputs from P4-to-VHDL compiler.

Our solution uses the library which accepts the description of controlled hardware in the form of Device Tree [1]. The Device Tree is a standardized data structure for describing hardware. In our case, the Device Tree description contains information about Match+Action tables, structure of search keys, mapping of actions to opcodes which are understood by the P4 pipeline, and so on. The description of P4 pipeline is generated during the processing of P4 source code by the P4 to VHDL Compiler (see Figure 1). The generated description in

human readable format is then compiled to binary form by the `dtc` [2] tool and used in the library. Notice that the library is not recompiled after the firmware change (thanks to the description of the pipeline in the Device Tree).

## P4 Configuration Library

The P4 configuration library contains generic enough code to control any P4 pipeline. It uses the information in Device Tree to control the hardware target. The main functionality of the library is following:

1. Initialization/deinitialization of P4 device
2. Enabling/disabling the P4 pipeline
3. Loading/deleting rules to/from Match+Action tables
4. Reading counters
5. Reading/writing registers

# Example of Rule Configuration

Example of library usage:

```
/* Declare variables */
uint32_t xret;
uint32_t table_capacity;
p4dev_t p4;

/* Prepare one rule which will be configured */
/* Values and masks */
uint8_t ip_key_val[]   = {0x0A,0x00,0x00,0xA0};
uint8_t ip_mask_val[]  = {0xFF,0xFF,0xFF,0xFF};
uint8_t tcp_key_val[]  = {0x50,0x00};
uint8_t tcp_key_mask[] = {0xFF,0xFF};
uint8_t vid_val[]      = {0x0C};
/* Rule conditions (last param creates linked list) */
p4key_elem_t ktcp  = {“tcp.dstport”, tcp_key_val, tcp_mask_val, 2, NULL};
p4key_elem_t kipv4 = {"ipv4.srcaddr", ip_key_val, ip_mask_val, 4, &ktcp};
/* Rule action */
p4param_t actparam = {"vid", vid_val, 1, NULL};
/* Final Rule */
p4rule_t rule      = {"table_ipv4_filter", &kipv4, "add_vlan_tag", &actparam};

/* Initialize the device */
xret = p4dev_direct_init(read_dt(“device_tree.dtb”), &p4);
if(xret != P4DEV_OK) {   // Code for error handling
}

/* Disable the P4 pipeline */
xret = p4dev_disable(&p4);
if(xret != P4DEV_OK) {   // Code for error handling
}

/* Print the capacity of table */
xret = p4dev_get_table_capacity(&p4, "table_ipv4_filter", &table_capacity);
```

```
if(xret != P4DEV_OK) {   // Code for error handling
}

printf("Capacity of the table is %d \n", table_capacity);

/* Load a rule */
xret = p4dev_insert_rule(&p4, rule);
if(xret != P4DEV_OK) {      // Code for error handling
}

/* Enable the P4 pipeline */
xret = p4dev_enable(&p4);
if(xret != P4DEV_OK) {      // Code for error handling
}

/* Cleanup */
p4dev_free(&p4);
```

## Configuration Tool

The tool uses the P4 configuration library as a backend and it adds the frontend for parsing of inserted rules, possibility to enable/disable the device, and so on. Therefore, it brings all the functionality of the library to the command line. The following text provides the example of rule insertion. The rule is inserted into the `table_ipv4_filter` where all incoming traffic to destination `75.114.210.196` and TCP `80` is being dropped. The next example shows the read operation of maximal rule capacity in `table_ipv4_filter`.

Rule.txt file:

```
table_ipv4_filter; key = {ipv4.srcaddr = 75.114.210.196, tcp.dstport = 80}; action = {_drop, {}}
```

Call of the command line tool (insert rules into P4 device):

```
> p4ctl -t device_tree.dtb -r Rule.txt
< DONE!
```

Example of table capacity read:

```
> p4ctl -t device_tree.dtb -c “table_ipv4_filter”
< 128
```

## References

[1] The Device Tree Specification. Available from: https://www.devicetree.org
[2] DTC (Device Tree Compiler). Available from: https://git.kernel.org/pub/scm/utils/dtc/dtc.git/tree/