P4-SDNet Translator

User Guide

UG1252 (v2017.3) October 27, 2017

XILINX ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/27/2017	2017.3	Released with SDNet 2017.3 without changes from the previous version.
08/24/2017	2017.2	Updated the --*help* flag current tool information under Compiling with p4c-sdnet. Updated *@Xilinx_MaxPacketRegion()* default value from 8,192 to12,144 in Table 1. Changed *MaxPacketRegion* from 8192 to 1518*8, *ControlStruct* to *switch_metadata_t*, *DROP_PORT* to *0xF*, and *LPM* to *ternary* in Appendix A sample program.
05/15/2017	2017.1	Initial release

Table of Contents

XILINX
ALL PROGRAMMABLE™

P4-SDNet Translator User Guide

P4 and SDNet

P4 is an emergent language standard for describing programmable data planes that can target a wide range of technologies including CPUs, FPGAs, and NPUs. An article published in ACM SIGCOMM CCR in 2014, with authors from Stanford, Princeton, Intel, Microsoft, and Google, laid the foundation. Shortly thereafter, the P4 Language Consortium (P4.org [Ref 1]) was established, with Xilinx as a founding member. Currently the consortium has more than 65 member companies and 15 universities world-wide participating to shape the development of the language.

The initial language specification now called $P4_{14}$ was released in early 2015. However, limitations were quickly identified and the community began exploring new features. The new $P4_{16}$ specification was released in May, 2017.

The Xilinx SDNet high-level design environment was created earlier to simplify the design of packet processing data planes that target FPGA hardware. Some of SDNet's design goals were very similar to those of the P4 language. However, SDNet places more emphasis on custom architectures.

SDNet allows programmers to build new data planes by explicitly specifying the dataflow graph of processing engines. SDNet processing engines have specialized behavior and include: ParsingEngines, LookupEngines, EditingEngines, TupleEngines, and UserEngines; each generated according to an application-specific processing. The basic types are similar to the components found in P4 such as the parser and the control components. However, unlike P4, SDNet lets developers explicitly declare the architecture at the level of point-to-point connections (in P4, only control flow is specified and the architecture is abstracted away). To implement a P4 specification, the P4-SDNet translator maps the control flow onto a custom data plane architecture of SDNet engines. This mapping chooses appropriate engine types and customizes each of them based on the P4-specified processing. More information about the SDNet design environment can be found in the *SDNet Packet Processor User Guide* (UG1012) [Ref 2].

Xilinx $P4_{16}$ Feature Support

At the time of this publication, the P4 Language Consortium had only released a draft specification for $P4_{16}$ for comments from the community. Although the core features are well-defined and unlikely to change, there are still parts of the compiler in flux. Currently, p4c-sdnet does not fully support all features in the draft specification.

Features currently supported by p4c-sdnet:

- Parsing and Deparsing
- Match+Action control flow
- Multiple Lookup Table types (exact, ternary and longest-prefix match)
- Extracting data from a packet (no look-ahead, or variable size field support)
- Emitting data into a packet
- Extern functions
- Generic types
- Header Stacks

Features NOT currently supported by p4c-sdnet:

- Extern objects
- Error types
- Variable length fields

Xilinx Specific P4 Annotations

The P4 language strives to be agnostic from any underlying hardware architecture. This methodology enables p4c-sdnet to always assume worst-case parameters, and this will likely hurt overall performance. However, p4c-sdnet supports the custom P4 annotations listed in Table 1 to help improve performance.

Table 1: **Xilinx Specific P4 Annotations**

Annotation	Valid P4 Objects	Description
`@Xilinx_MaxPacketRegion()`	parser/control containers	The maximum packet depth (in bits). *Default: 12,144*
`@Xilinx_MaxLatency()`	extern functions	The maximum latency (in cycles) for an extern function call. *Default: 1*

Table 1: **Xilinx Specific P4 Annotations *(Cont'd)***

Annotation	Valid P4 Objects	Description
`@Xilinx_ControlWidth()`	extern functions	The width of the software control address space (in bits) for an extern function call. *Default: 0*
`@Xilinx_ExternallyConnected`	table objects	Pulls the table's request and response tuples to the top level of the design.

Xilinx $P4_{16}$ Supported Architectures

The $P4_{16}$ specification has architecture-language separation: this gives target providers the flexibility to describe the nature of P4-programmable components within their own packet forwarding architectures. This architecture description gives signatures for container holes that P4 developers can fill with their desired functionality. The containers are specified in a generic fashion to maintain P4's protocol independence. Xilinx's p4c-sdnet tool currently supports two architectures that developers can target:

- XilinxSwitch
- XilinxEngineOnly

XilinxSwitch Architecture Description

Designs targeting the XilinxSwitch architecture generate a pipeline with three customizable containers. Figure 1 shows how containers are connected within the pipeline. The container arguments are defined generically, which allows developers the flexibility to define how much header and control data is passing though the data pipeline.

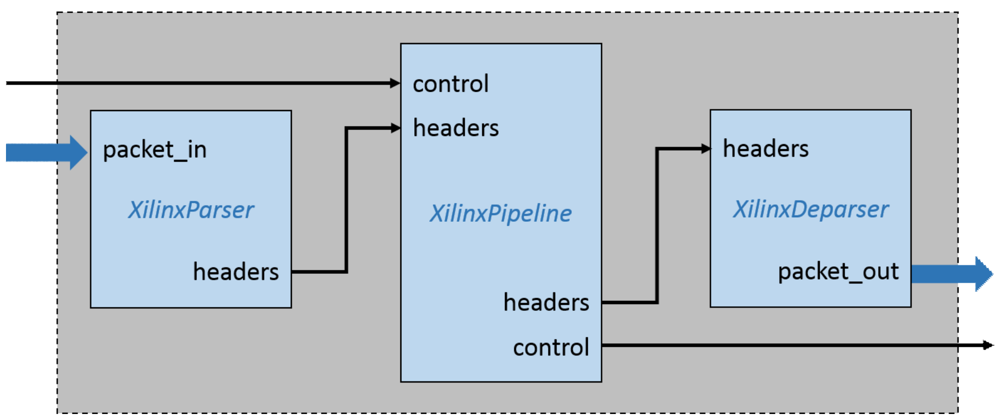

Figure 1: **XilinxSwitch Layout**

The first container (XilinxParser) is a parsing block that extracts headers from the packet. The next container (XilinxPipeline) is a control block that can be used to modify header and control data. The last container (XilinxDeparser) is another control block specifying the order headers that should be de-parsed back into the packet.

The $P4_{16}$ source code for the XilinxSwitch architecture description is as follows:

```
parser XilinxParser<H>(packet_in pkt, out H headers);
control XilinxPipeline<H,C>(inout H headers, inout C control);
control XilinxDeparser<H>(in H headers, packet_out pkt);

package XilinxSwitch<H,C>(XilinxParser<H> prsr,
                          XilinxPipeline<H,C> pipe,
                          XilinxDeparser<H> dprsr);
```

The source code can be found at `SDNet/2017.3/data/p4include/xilinx_model.p4`. An example of P4-programmed components for this architecture is provided in Appendix A.

XilinxEngineOnly Architecture Description

The XilinxEngineOnly architecture does not generate a complete forwarding pipeline. It can be used to generate stand-alone SDNet engines without any system building logic (i.e., the generated SDNet code will not specify multiple engines with connections). For generality, the architecture has no pre-defined P4 containers or packages. Developers can define these components in accordance with their needs.

This architecture is useful for developers needing only a single container (e.g., only a packet parser), or advanced developers wishing to do their own SDNet system building. The following code outlines a $P4_{16}$ parser design that can be used to extract a 5-tuple from network packets. Compiling this design with p4c-sdnet generates a single SDNet ParsingEngine.

```
struct five_tuple_t {
     bit<8>       proto;
     bit<32>      ip_src;
     bit<32>      ip_dst;
     bit<16>      sport;
     bit<16>      dport; }

parser FiveTuple_t(packet_in p, inout five_tuple_t t);
package XilinxEngineOnly(FiveTuple_t container);

parser MyParser(packet_in pkt, inout five_tuple_t tup) {
     ...
}

XilinxEngineOnly(MyParser()) main;
```

Translating $P4_{16}$ to SDNet

As of release 2017.1, the SDNet design environment offers a $P4_{16}$ to SDNet translator. A beta version of the tool can be found in some releases of SDNet 2016.4. The translator can be used in conjunction with the SDNet tools to compile $P4_{16}$ sources down to Xilinx FPGA targets.

- The translator is located at: `SDNet/2017.3/bin/p4c-sdnet`
- Example programs can be found under: `SDNet/2017.3/examples/p4examples/`
- Standard include files can be found under: `SDNet/2017.3/data/p4include/`

Operating System Support

The P4 translator is currently only supported on 64-bit Linux operating systems. Although it might run on many distributions, the following are officially supported:

- Ubuntu
- CentOS

Compiling with p4c-sdnet

Use the following command to compile $P4_{16}$ source code at the Linux command prompt. The resulting `output.sdnet` file can then be used as normal as a source file within the SDNet design environment.

```
$ p4c-sdnet input.p4 -o output.sdnet
```

Use the *-v* flag when debugging or reporting errors. Multiple *-v* flags can be used together to increase the verbosity level of the output.

```
$ p4c-sdnet -v input.p4
$ p4c-sdnet -v -v input.p4
```

Use the *--help* flag for more information about the tool. The current information given is:

```
$ p4c-sdnet --help

Compile a P4 program
--help                  Print this help message
--version               Print compiler version
-I path                 Specify include path (passed to preprocessor)
-D arg=value            Define macro (passed to preprocessor)
-U arg                  Undefine macro (passed to preprocessor)
-E                      Preprocess only, do not compile (prints program
                        on stdout)
-o outfile              Write output to outfile
--Werror                Treat all warnings as errors
--no_arch               Generate only SDNet engines (i.e. no system)
--sdnet_info outfile    Write SDNet switch information to outfile
```

Appendix A

This is a sample $P4_{16}$ program. A copy can be found at: `SDNet/2017.3/examples/p4examples/forward.p4`.

```
typedef bit<48>     MacAddress;
typedef bit<32>     IPv4Address;
typedef bit<128>    IPv6Address;

header ethernet_h {
    MacAddress          dst;
    MacAddress          src;
    bit<16>             type; }

header ipv4_h {
    bit<4>              version;
    bit<4>              ihl;
    bit<8>              tos;
    bit<16>             len;
    bit<16>             id;
    bit<3>              flags;
    bit<13>             frag;
    bit<8>              ttl;
    bit<8>              proto;
    bit<16>             chksum;
    IPv4Address         src;
    IPv4Address         dst; }

header ipv6_h {
    bit<4>              version;
    bit<8>              tc;
    bit<20>             fl;
    bit<16>             plen;
    bit<8>              nh;
    bit<8>              hl;
    IPv6Address         src;
    IPv6Address         dst; }

header tcp_h {
    bit<16>             sport;
    bit<16>             dport;
    bit<32>             seq;
    bit<32>             ack;
    bit<4>              dataofs;
    bit<4>              reserved;
    bit<8>              flags;
    bit<16>             window;
    bit<16>             chksum;
    bit<16>             urgptr; }

header udp_h {
    bit<16>             sport;
    bit<16>             dport;
    bit<16>             len;
    bit<16>             chksum; }
```

```
struct headers_t {
    ethernet_h          ethernet;
    ipv4_h              ipv4;
    ipv6_h              ipv6;
    tcp_h               tcp;
    udp_h               udp; }

@Xilinx_MaxPacketRegion(1518*8)  // in bits
parser Parser(packet_in pkt, out headers_t hdr) {

    state start {
        pkt.extract(hdr.ethernet);
        transition select(hdr.ethernet.type) {
            0x0800  : parse_ipv4;
            0x86DD  : parse_ipv6;
            default : accept;
        }
    }
    state parse_ipv4 {
        pkt.extract(hdr.ipv4);
        transition select(hdr.ipv4.proto) {
            6       : parse_tcp;
            17      : parse_udp;
            default : accept;
        }
    }
    state parse_ipv6 {
        pkt.extract(hdr.ipv6);
        transition select(hdr.ipv6.nh) {
            6       : parse_tcp;
            17      : parse_udp;
            default : accept;
        }
    }
    state parse_tcp {
        pkt.extract(hdr.tcp);
        transition accept;
    }
    state parse_udp {
        pkt.extract(hdr.udp);
        transition accept;
    }
}


control Forward(inout headers_t hdr, inout switch_metadata_t ctrl) {
    action forwardPacket(switch_port_t value) {
        ctrl.egress_port = value;
    }
    action dropPacket() {
        ctrl.egress_port = 0xF;
    }

    table forwardIPv4 {
        key             = { hdr.ipv4.dst : ternary; }
        actions         = { forwardPacket; dropPacket; }
        size            = 63;
        default_action  = dropPacket;
    }
```

```
    table forwardIPv6 {
        key             = { hdr.ipv6.dst : exact; }
        actions         = { forwardPacket; dropPacket; }
        size            = 64;
        default_action  = dropPacket;
    }

    apply {
        if (hdr.ipv4.isValid())
            forwardIPv4.apply();
        else if (hdr.ipv6.isValid())
            forwardIPv6.apply();
        else
            dropPacket();
    }
}


@Xilinx_MaxPacketRegion(1518*8)  // in bits
control Deparser(in headers_t hdr, packet_out pkt) {
    apply {
        pkt.emit(hdr.ethernet);
        pkt.emit(hdr.ipv4);
        pkt.emit(hdr.ipv6);
        pkt.emit(hdr.tcp);
        pkt.emit(hdr.udp);
    }
}

XilinxSwitch(Parser(), Forward(), Deparser()) main;
```

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. *P4 Language Consortium website* (http://p4.org)
2. *SDNet Packet Processor User Guide* (UG1012)

Please Read: Important Legal Notices