Microprocessors and Microsystems 56 (2018) 22–33

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

P4-To-VHDL: Automatic generation of high-speed input and output network blocks

Pavel Benáček*[a], Viktor Puš[a], Hana Kubátová[b], Tomáš Čejka[a]

[a] CESNET a. l. e. Zikova 4, 160 00 Prague, Czech Republic
[b] Faculty of Information Technology Czech Technical University in Prague Thákurova 9, 160 00 Prague, Czech Republic

ARTICLE INFO

Keywords:
FPGA
High-level language
P4
100 Gbps
Parser
Deparser

ABSTRACT

High-performance embedded architectures typically contain many stand-alone blocks which communicate and exchange data; additionally a high-speed network interface is usually needed at the boundary of the system. The software-based data processing is typically slow which leads to a need for hardware accelerated approaches. The problem is getting harder if the supported protocol stack is rapidly changing. Such problem can be effectively solved by the Field Programmable Gate Arrays and high-level synthesis which together provide a high degree of generality. This approach has several advantages like fast development or possibility to enable the area of packet-oriented communication to domain oriented experts. However, the typical disadvantage of this approach is the insufficient performance of generated system from a high-level description. This can be a serious problem in the case of a system which is required to process data at high packet rates. This work presents a generator of high-speed input (Parser) and output (Deparser) network blocks from the P4 language which is designed for the description of modern packet processing devices. The tool converts a P4 description to a synthesizable VHDL code suitable for the FPGA implementation. We present design, analysis and experimental results of our generator. Our results show that the generated circuits are able to process 100 Gbps traffic with fairly complex protocol structure at line rate on Xilinx Virtex-7 XCVH580T FPGA. The approach can be used not only in networking devices but also in other applications like packet processing engines in embedded cores because the P4 language is device and protocol independent.

1. Introduction

Embedded hardware is nowadays used in almost every advanced technical system. Very typically, the embedded architecture contains more processing cores which need to communicate and exchange data. This communication can be inspired from the field of computer networks. Moreover, developers of NoC (Network on Chip) can reuse knowledge from this area which leads to faster development of final product. The number of computers and mobile devices is still rising which leads to higher bandwidth utilization. There is also a requirement to analyze all data in real time which is quite complicated to fulfil in pure software tools. Therefore, it is typical to embed the time-critical parts of networking operations into hardware accelerators. The advantage of this approach is the performance and native parallelism of hardware. However, development of networking hardware is quite complicated discipline and it isn't easy to meet all required parameters like frequency, throughput or used resources. Very common are hardware accelerators with FPGA chips because such solutions are flexible and fast enough for processing of high-speed traffic even at speed of 100 Gbps.

From above description, we feel that it is quite complicated to find an expert who understands the FPGA technology and domain-specific problems (like computer network security, etc.). Therefore, high-level languages (HLL) seem to be suitable for such developers because they allow describing the hardware in simpler languages than VHDL or Verilog. Unfortunately, this level of abstraction suffers from performance penalty and there is a need to have a tool which is capable to meet a trade-off between the level of abstraction and reached performance.

P4: Programming Protocol-independent Packet Processors [1,2] is novel and open source language which seems to be suitable for the description of modern network devices. It also evades the typical problem of classical network approaches, such as a fixed set of supported protocols, fixed set of actions, and so on. The main idea of this language is the capability to map a P4 program onto computational resources like FPGAs, graphics cards or programmable switches. We contribute towards the vision of P4 by designing and evaluating a generator of high-speed packet parser and deparser suitable for FPGAs. The generator's

* Corresponding author.
E-mail addresses: benacek@cesnet.cz (P. Benáček), pus@cesnet.cz (V. Puš), hana.kubatova@fit.cvut.cz (H. Kubátová), cejkat@cesnet.cz (T. Čejka).

http://dx.doi.org/10.1016/j.micpro.2017.10.012
Received 12 May 2016; Received in revised form 3 October 2016; Accepted 27 October 2017
Available online 31 October 2017

output is a synthesizable VHDL code that performs packet parsing and deparsing as defined by the P4 program. Internal structure of both modules is inspired by hand-written modules which were developed by a skilled HDL programmer.

This paper introduces results of our continuing research which was initially published in [3,4]. In this paper, we provide details of output network block (Deparser), the detailed description of Parser-Deparser approach and analysis of three use cases. The rest of the paper is organized as follows: Section 2 provides more details about the P4 language aspects that are relevant for this work. Section 3 introduces the basic ideas of Parser-Deparser approach and compares our work to conventional packet editor. Section 3.1 provides details about Parser's architecture and transformation algorithm from P4 to VHDL. The section also provides a description of available optimizations. Section 3.2 provides details about Deparser's architecture and transformation from P4 to VHDL. Section 4 provides results of our generator and compares them to a hand-written parser. The section also provides results for generated Deparsers which are supporting the same set of protocols. Section 5 presents other papers related to our work. Finally, Section 6 concludes the paper with important outcomes.

2. P4 Language

P4: Programming Protocol-independent Packet Processors [1,2] is a high-level, platform-agnostic language. It represents a recent contribution to the broader idea of Software-Defined Networking (SDN) and its ecosystem. The main purpose of P4 is to provide a way to define packet processing functionality of network devices, paying attention to reconfigurability in the field, protocol independence and target (platform) independence. Using relatively simple syntax, P4 allows to define five basic aspects of packet processing:

- **Header Formats** describe protocol headers recognized by the device.
- **Packet Parser** describes the (conceptual) state machine used to traverse packet headers from start to end, extracting field values as it goes.
- **Table Specification** defines how the extracted header fields are matched in possibly multiple lookup tables (e.g., exact match, prefix match, range search).
- **Action Specification** defines compound actions that may be executed for packets.
- **Control Program** puts all of the above together, defining the control flow mainly among the tables.

All proposed aspects have to be defined for each network device like routers, switches, monitoring probes, etc. The first aspect, Header Formats, is used for the definition of all supported protocol headers. The P4 language defines the following syntax:

```
header ethernet {
 fields {
  dst_addr : 48; // width in bits
  src_addr : 48;
  ethertype : 16;
 }
}
```

The definition simply lists fields of the packet header and their width in bits. The example above shows a protocol with static header structure, where the header length is the sum of lengths of all fields. This can't be done for protocols with variable header length. The P4 language solves this situation by the `length` statement definition in the form of an expression which uses protocol fields (from the Header Format definition) to compute the header length. Header Format definition with variable length may look like this:

```
header_type ipv6_ext_t {
 fields {
  nextHdr  : 8;
  totalLen : 8;
  frag  : 12;
  padding  : 3;
  fragLast : 1;
 }
 length : (totalLen + 1) * 8;
 max_length : 1024; // Bytes
}
```

Packet Parser definition constructs a parse graph using the Header Format definition, for example:

```
header ethernet eth;
parser ethernet {
 extract(eth);
 switch(eth.ethertype) {
  case 0x8100: vlan;
  case 0x9100: vlan;
  case 0x0800: ipv4;
  case 0xA100 mask 0xF100 : myProto;
 }
}
```

The provided example uses `switch` and `extract` statements. The `extract` statement instructs the parser to examine input packets and look for data defined in the header. Parsed data is then used in the `switch` statement to determine the next state (protocol) to process. There are also situations when we don't want to use the whole value from the protocol field. The P4 language solves this by the `mask` statement which is used in the `case` statement together with a mask value. In our example, the `mask` statement instructs the P4 parser to take the `ethertype` field, perform logical *and* operation between the value and mask. Finally, the result is compared to `0xA100`.

In our approach, we want to use first two aspects (Header Formats and Packet Parser definition) for automatic generation process. The remaining P4's aspects (definition of tables, actions and control program) define a more dynamic and general transformation process. The example of such transformation process can be VLAN or MPLS tagging, decrementation of the TTL value, insertion of new protocol header, etc. The next section describes the structure of the device in detail.

3. Device structure

The basic idea of the Parser-Deparser approach was introduced by Gibb in [5] and we adopted it into our approach. The generated device consists of three basic modules. The first module, Parser, is used to break the incoming network data into individual header fields. The output of this module is a set of extracted fields and corresponding valid bits. The valid bit is used for presence indication of extracted protocol fields in the currently processed packet. All extracted protocol fields and valid bits are passed to the second module, Transformation, which implements the general transformation process (VLAN tagging or traffic analysis for example). This block has to set a validity information for each inserted/removed protocol and it has to filter out unused header data (this data will not be used in Deparser). Finally, the last module, Deparser, is used for the construction of network packet back from incoming protocol headers and valid bits.

The brief architecture is shown in Fig. 1. The nearest comparable solution is a conventional packet editor. The packet editor is a general device which allows us to read or modify/insert bytes of the currently processed packet. Therefore, we can use this device for usual network operations like VLAN tagging, modification of protocol fields, removing of unwanted protocols, and so on. However, the Parser-Deparser approach is more elegant and simpler. We will demonstrate the flexibility of new approach on two common use cases — VLAN tagging and