A Programmable and Scalable OpenFlow Switch using Heterogeneous SoC Platforms

Shijie Zhou
U. of Southern California
shijiezh@usc.edu

Weirong Jiang
Xilinx Research Labs
weirongj@acm.org

Viktor K. Prasanna
U. of Southern California
prasanna@usc.edu

ABSTRACT

This work presents a hardware-software co-design approach of an OpenFlow switch using a state-of-the-art heterogeneous System-on-chip (SoC) platform. Specifically, we implement the OpenFlow switch on a Xilinx Zynq ZC706 board. The Xilinx Zynq SoC family provides a tight coupling of field programmable gate array (FPGA) fabric and ARM processor cores, making it an attractive on-chip implementation platform for SDN switches. High-performance, yet highly-programmable, data plane processing can reside in the programmable logic (PL), while complex control software can reside in ARM processor. Our proposed architecture scales across a range of possible packet throughput rates and a range of possible flow table sizes. Post-place-and-route results show that our design targeted at Zynq can achieve a total 88 Gbps throughput for a 1K flow table which supports dynamic updates. Correct operation has been demonstrated using a ZC706 board.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Packet-switching networks*

Keywords

Software Defined Networking; OpenFlow switch; Heterogeneous SoC

1. INTRODUCTION

An OpenFlow switch is mainly composed of an OpenFlow agent which interacts with the controller over OpenFlow protocol, and an OpenFlow data plane which performs packet lookup and forwarding [1].

The reconfigurability and massive parallelism of field programmable gate array (FPGA) technology have made it an attractive option for implementing real-time network processing engines. There has been a growing interest in employing FPGA to implement the OpenFlow switch to achieve high throughput [2].

A heterogeneous SoC architecture typically consists of multiple components of different types (e.g. ARM, FPGA, DSP) interconnected together. Zynq SoC is a system on chip platform which tightly integrates a dual ARM Cortex^{TM}-A9 MPcores processing system (PS) with the most advanced 28 nm FPGA fabric. The ARM processor and FPGA are connected via high-speed AXI4 [3] interfaces, eliminating the common bottleneck for data transmission between these two subsystems. It is an ideal platform for hardware-software co-designs.

Based on the hardware-software integration of the heterogeneous SoC platforms, we devise an OpenFlow switch based on the OpenFlow 1.0.0 [1] specification for the heterogeneous SoC platforms and implement it on a Zynq board. Our architecture is scalable to support a range of possible packet throughput rates by varying the input data width. The post-place-and-route results show that our OpenFlow switch can achieve a high throughput of 88 Gbps while supporting 1K flow entries.

2. ARCHITECTURE

Our OpenFlow switch architecture consists of a software-based agent and a hardware-based data plane. Figure 1 depicts the overview of the architecture.

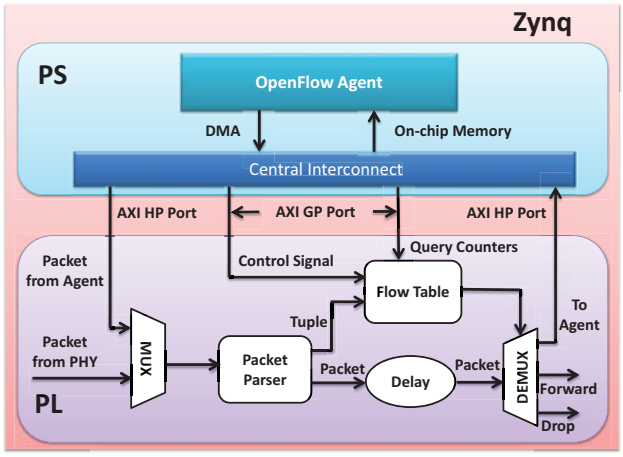

Figure 1: Architecture Overview

HotSDN'14, August 22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2989-7/14/08.
http://dx.doi.org/10.1145/2620728.2620767.

2.1 Software

The OpenFlow agent is a Linux application running in PS. It can send packets to the data plane, update/delete flow entries, query the counters and capture the packets forwarded from the data plane.

Updating the flow table and querying the counters are achieved through writing and reading the specific registers, respectively. To send packets to the data plane, the agent first transfers packets to the central interconnect of PS via a direct memory access (DMA) controller. Packets are then sent to PL through AXI interfaces in a streaming fashion. To receive the packets from the data plane, the agent reserves specific on-chip memory blocks where PL can write through AXI interfaces.

2.2 Hardware

The data plane on the PL is implemented as modular stages connected together in a pipeline. The packets may come from the OpenFlow agent (PS) or the physical interfaces on the PL. A packet parser extracts the fields of interest from the packet headers and concatenates them into a tuple. The tuple is passed into a flow table for lookup. A few delay blocks are added to ensure the synchronization of the packets and their lookup results. Once a final lookup decision is reached, the counters for that flow entry are updated and the action is taken. If there is no matching result, the packets are encapsulated and forwarded to the agent.

The hardware system (packet parser and lookup table) is generated by the PX language [4], which is a high-level domain-specific language. The PX language treats packets in an object-oriented style and can achieve a similar functionality as a P4 [5] program does. It provides designers the flexibility to define packet processing policies for FPGA-based switches.

2.3 SW-HW Interface

The OpenFlow agent utilizes one high-performance port for sending software-generated packets to PL, and another high-performance port for receiving packets with no lookup match from PL. We adopt AXI-stream protocol to provide high-speed packets transmission between PS and PL. We use the general purpose port with AXI-Lite interfaces for the transferring control signals and reading counters.

3. EVALUATION RESULTS

We evaluate the performance based on the Xilinx Vivado 2013.2. The experiments are targeted at the Zynq ZC706 featuring xc7z045. We use throughput and slice utilization as the performance metrics.

We first explore the impact of varying the input data width. In these experiments, we fix the flow table size at 128 entries, and vary the input data width from 64 to 512 bits. The results are shown in Table 1. The clock rate is not affected negatively due to the pipelined architecture. The slice usage increases as the input data width increases, but not in a linear fashion.

Then we fix the input data path width at 512 bits and increase the flow table size from 128 to 1024 entries. As shown in Table 2, the clock rate and throughput show a gently dropping trend as the flow table becomes larger. The deterioration is caused by a higher routing complexity and more long wires. With a 1024-entry flow table and a 512-bit input data width, the slice resource is consumed by 78%, and the throughput is 88 Gbps.

Table 1: Increasing the input data width

Data Width (bit)	64	128	256	512
Clock Rate (MHz)	204.8	202.6	202.4	202.5
Throughput (Gbps)	13.1	25.9	51.8	103.7
Slice Utilization	13%	14%	15%	17%

Table 2: Increasing the number of flow entries

No. of flow entries	128	256	512	1024
Clock Rate (MHz)	202.5	202.1	200.5	173.1
Throughput (Gbps)	103.7	103.5	102.7	88.6
Slice Utilization	17%	25%	42%	78%

Finally, we deploy the switch on a Zynq ZC706 board. Traffic enters the switch through a 4×10 Gbps daughter card. We configure the data plane with 256-bit data width and 512 flow entries. The frequency of the buses connecting PS and PL is set to 250 MHz. Besides the physical traffic, the OpenFlow switch can also process 30 million software-generated per second. It takes 120 ns for the insertion/-modification control signal while 10 ns for deletion signal to reach the data plane; another 20 ns is needed to complete an update in PL.

4. CONCLUSION

In this paper, we devised an OpenFlow switch architecture targeted on the Zynq SoC, a platform of efficient integration of ARM and FPGA. The experimental results showed that our design achieved 88 Gbps throughput for 1K flow entries and scaled well with larger input data width and flow table. The on-board deployment demonstrated the correct operation.

5. ACKNOWLEDGEMENTS

This work is supported by U.S. National Science Foundation under grant CCF-1320211. Equipment grant and toolset from Xilinx, Inc. are gratefully acknowledged.

6. REFERENCES

[1] "OpenFlow Switch Specification V1.0.0" `http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf`

[2] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown,"Implementing an OpenFlow switch on the NetFPGA platform," in Proc. ANCS, 2008, pp. 1-9.

[3] "AMBA AXI4 Interface Protocol" `http://www.xilinx.com/ipcenter/axi4.htm`

[4] G. Brebner and W. Jiang, "High-Speed Packet Processing using Reconfigurable Computing," in Proc. Micro, 2014, pp. 8-18.

[5] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "Programming protocol-independent packet processors," December 2013. `http://arxiv.org/abs/1312.1719`