BittWare a molex company

White Paper

Building BittWare's Packet Parser Using HLS vs. P4

Introduction

One of the features of both BittWare's SmartNIC Shell and BittWare's Loopback Example is a packet parser/classifier that extracts protocol fields from packets. The fields are called tuples and are added to each packet's metadata that the user's application can utilize. See the sidebars for more details on how packet parsers and the tuple fields are used.

With this white paper, we not only wanted to describe our Parser, but explain how using HLS to build and configure it has resulted in a better implementation than using the P4 language. The Parser code is available on the BittWare developer website for free to Xilinx Ultrascale+ card owners as part of our Loopback Example (January 2020 availability)

Moving Beyond P4 for Programming Smart-NICs

Today the Parser component of BittWare's SmartNIC Shell is built using the Xilinx HLS C++ development environment. But an earlier revision of BittWare's SmartNIC Shell used the P4 language though the Xilinx SDNet tool.

One reason to use P4 is that it's an emerging standard popular among people embracing software-defined networking (SDN) on commodity Intel servers. However, Xilinx later restricted the availability of SDNet. Our use of P4 was specifically for end-users of SmartNIC Shell, so this restriction caused us to search for a more open solution. Following the success of our RSS implementation using HLS, we were motivated to re-implement the SmartNIC Shell parser using this same HLS approach (specifically the Xilinx HLS C++ environment).

What is a "Tuple"?

In networking, tuples are fields extracted from networking packets and grouped together. The most common is the "5-tuple" which combines source and destination IP address, source and destination IP port (if the IP protocol has them), and the IP protocol number.

The BittWare Parser in the SmartNIC offering examines packets and extracts up to a 4-tuple if available. It places that data into a 96-bit field added to the packet metadata. That field width provides enough bits for the IPv4 source and destination address as well as source and destination port. Our Parser provides zeros for fields that are not available in the packet. If a packet does not include any IP payload, the full 96-bit tuple field is zero.

A full 5-tuple would require an additional 8 bits to accommodate the protocol number. HLS users of BittWare's parser can easily accommodate that change with minor source code changes.

BittWare's RSS is an example of a block that might follow the parser in the packet pipeline and consume the 96-bit tuple data. Read about that block in our white paper comparing RTL to HLS C++, which is available on the BittWare website.

The Parser is used differently in BittWare's Loopback example. The Loopback uses three copies of the Parser, as opposed to a single copy that expands metadata. This approach was taken because the Parser is actually quite small.

Building BittWare's Packet Parser Using HLS vs. P4

What is a Packet Parser?

The protocols used over Ethernet are challenging for hardware to leverage. This challenge exists because the protocols have many optional fields. Those options make it complicated to find, for example, the start of an IP header. Why? In the IP header case, there can be zero, one, or two VLAN tags in front of it. There can also be MPLS tags. Thus hardware needs to understand the protocol just enough to find the IP header. Hardware needs the IP header in order to find IP addresses which are often used in hardware filters and tables. Similar problems exist at the next level as the IP header itself has optional fields.

BittWare's HLS C++ packet parser can deal with:

- 0 to 2 VLAN tags (the old SDNet code allowed 0 or 1)
- 0 to 5 MPLS tags (BittWare's old SDNet code did not recognize MPLS)
- IP fragments
- IPv4 headers (not IPv6)
- It assumes port IDs are found in these IP protocols: TCP, UDP, DCCP, and STCP

Having essentially created two versions of a packet parser, we noted some differences between using P4 versus HLS C++. Overall, the HLS flow is less abstract than P4, but the tool is far more mature.

Details of resource usage are in the table:

Characteristic	P4/SDNet	HLS C++
CLBs	3,185	3,391
BRAM	22	0
Registers	10,361	5,975
Lines of Code	206	1,154

You can see that across all FPGA resources, HLS is either similar or better. While the source code does require more lines, part of that is impacted by comments and formatting. However, it is true that an HLS C++ implementation is always going to need more lines of code than P4. That's for a packet parser/classifier though, which falls under the scope of what P4 can describe—HLS C++ can do more. HLS is very general purpose and can pretty much do anything. P4 is very specialized.

Even better, now that the HLS implementation exists, any follow-on effort to modify it to digest an Ethernet protocol variation is roughly the same as doing the modification in the P4 language. This is because our HLS C++ implementation is structured as a sequence of calls to low-level parser functions that we created. This approach is analogous to directly manipulating the runtime that sits under P4.

As noted, the source code for the Loopback Example, including its Parser block, is available for free to Ultrascale++ owners through the BittWare Developer site. It is an excellent illustration of how to use AXI interfaces within HLS C++ code. Want to see it but don't have a BittWare FPGA card? Get in touch with us for where to buy.

BittWare IP Block Interfaces

The HLS C++ tool flow needs to have built-in awareness of the interface protocols used. IP blocks from BittWare generally use Advanced eXtensible Interface (AXI) to communicate. Specifically, an AXI4-Stream to pass packet data and AXI4-Lite as a control plane. Xilinx documents AXI here:

https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v13_4/ug761_axi_reference_guide.pdf

For 100 GbE, BittWare uses an AXI4-Stream interface that is 512 bits wide and clocked at 300 MHz. The metadata associated with each packet follows on its own bus that is valid at the end of a packet, when the packet data's TLAST signal is asserted. The packet metadata evolves between blocks and between releases.

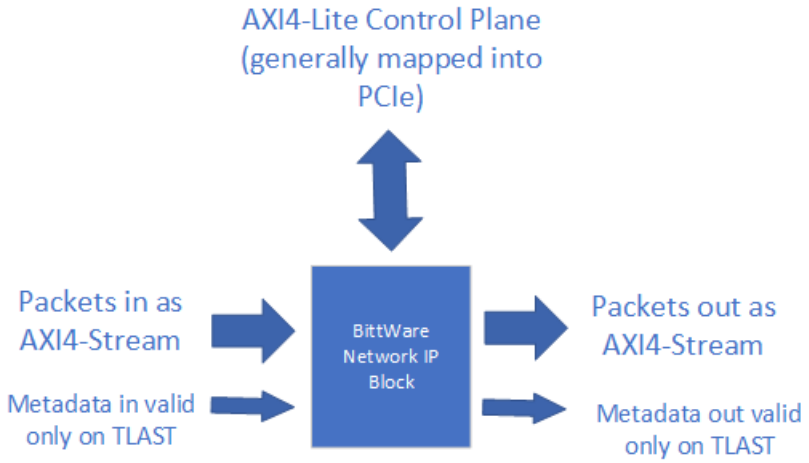
Building BittWare's Packet Parser Using HLS vs. P4

It usually includes information about:

- The number of the physical Ethernet connector that the packet arrived on
- Any errors the MAC identified associated with the packet
- A timestamp in 80-bit IEEE-1588 format or sometimes in a shortened 64-bit format
- A "deleted" bit to indicate the packet needs to be removed from the stream at the next opportunity
- A number we usually call "queue" to indicate a destination for the packet. It is calculated by one of the IP blocks in the pipeline (maybe even this block)

Our control plane for the parser block includes:

- An enable/disable bit
- A bit that forces generating a 2-tuple even if the packet contains 4-tuple data



Will P4 Become Common for FPGA Hardware?

The P4 language was created to define a "packet forwarding data plane" (or network switch) using software. The language is particularly associated with hardware vendor Barefoot Networks. The P4 language is distinct from something called "P4 Runtime" which Google helps promote. P4 Runtime presents a standard runtime API that enables manipulating the control plane of solutions compiled by P4.

P4 does make it easy to define a packet classifier/parser for a new protocol. P4 also specifies a complete set of table lookup functions, and it can rewrite packets that flow through, eliminating VLAN tags, for example.

Does this mean that the flexibility of P4 will lead to adoption for FPGAs? There are several reasons we see against this happening.

Commercial options to provide a subset of P4 on FPGA hardware exist, however they are currently limited in scope. Furthermore, as noted earlier, the commercial terms make it difficult for BittWare to leverage these to create an example program that we can provide free with our products.

It's important to note that no real-world FPGA application can be exclusively written in P4. For example, the Receiver Side Scaling (RSS) block that follows our Parser in some examples cannot be authored in P4. However, HLS C++ can be used to author either block, or even a single block that combines the two functions.

Also, the P4 table lookup functions are basically a wrapper on hardware-specific runtime libraries written in RTL or HLS C++. Programmers can call such runtimes directly from HLS C++ with no penalty.

The bottom line is that after using both P4 and HLS C++ to implement a parser, we actually favor the HLS C++ approach. It isn't clear that demand for P4 on FPGAs will grow large enough to support a mature tool. HLS C++ can do more and is more mature.

Building BittWare's Packet Parser Using HLS vs. P4

Portability of HLS and Conclusion

We hope the explanation of two implementations of a packet parser on an FPGA, one in the P4 language and then using HLS C++, are helpful in evaluating the right approach for you.

One final note is regarding portability between our FPGA cards. Between our Xilinx FPGA-based cards, HLS provides an easy method with few, if any, changes needed. For moving to an Intel-based card, such as our 520N-MX, source code changes will be required, particularly with respect to compliler pragmas. However, the basic concepts are identical. In both cases we are structuring C++ based upon our knowledge of FPGA translation challenges. Arbitrary C++ code will run very poorly inside an FPGA. However, C++ code restructured and anointed with pragmas works very well. The changes required for Xilinx or Intel are very similar but just expressed a little differently.

As part of BittWare's SmartNIC Shell, our Parser helps teams get up to speed quickly for building network packet processing applications on our FPGA cards. Learn more about SmartNIC for our cards or get in touch with us to talk about your application needs.

BittWare's Loopback example redeploys a subset of the SmartNIC shell that we can offer at no charge. That subset includes our Parser library.

To learn more, visit the BittWare website at www.bittware.com.

Rev 2019.12.12 | December 2019

BittWare
a molex company