Optimizing a Hardware Network Stack to Realize an In-Network ML Inference Application

TECHNISCHE UNIVERSITÄT DARMSTADT

SC21 Workshop H2RC'21

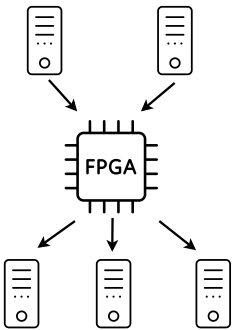
Marco Hartmann, Lukas Weber, Johannes Wirth, Lukas Sommer, Andreas Koch

Background

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- In-Network Processing (INP)
  - Move computation onto specialized hardware within the network
  - Process in-flight data → Reduce network traffic
  - Free up CPU resources → Reduce costs of data center applications
- Application offload onto network-attached FPGAs
  - More flexibility and performance than programmable switches (P4, Tofino, …)
  - Requires a high-throughput hardware network stack
  - Acceleration of both packet processing and application layer



Background

TECHNISCHE UNIVERSITÄT DARMSTADT

- Related Work: FPGA-based TCP/IP Offload increasingly popular
  - More energy/cost efficient than software implementations
  - Few available projects support 100G Ethernet
  - Many HFT-optimized solutions ($\rightarrow$ low latency, few sessions)
- HLS-based hardware TCP/IP stack developed at ETH Zürich[1]
  - Implements: UDP/TCP, IP, ICMP, ARP
  - Supports: 100G, up to 10k concurrent sessions
  - Stack is used in this work

[1] `https://github.com/fpgasystems/Vitis_with_100Gbps_TCP-IP`

Background (2)

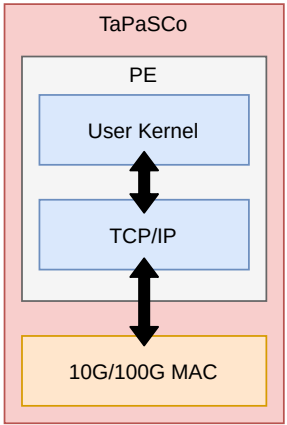
TECHNISCHE UNIVERSITÄT DARMSTADT

- Task Parallel System Composer[2] (ESA at TU Darmstadt)
  - Integrates FPGA-based accelerators into heterogeneous compute platforms
  - User provides HLS kernels or custom HDL cores ($\rightarrow$ PE)
  - Provides software API for dispatching compute tasks to FPGA
- TaPaSCo Network plugin
  - Adds an Ethernet subsystem to the FPGA design
  - Ethernet-level access to a network (10G or 100G)
  - Upper-layer protocols must be implemented manually

[2] https://github.com/esa-tu-darmstadt/tapasco

Goals

1. Enable the development of portable TCP/IP-capable FPGA designs using TaPaSCo
2. Evaluation of TCP/IP-capable FPGA designs w.r.t. throughput, latency, and resource utilization
3. Field study: In-Network acceleration of Sum-Product Network inference



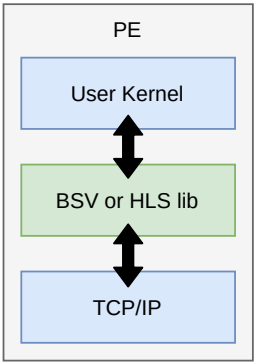
Contributions: Tools for developing TCP/IP-capable FPGA designs

Bluespec Library for Designing new User Kernels

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

- TCP/IP stack user interface: **16** AXI Stream ports
  - HLS User Kernels → Existing HLS C++ Library
  - HDL User Kernels, e.g. Bluespec
- Contribution: Bluespec Network Library[3]
  - Idiomatic user interface
    - UDP: TX/RX
    - TCP: TX/RX, session control, port control
  - Abstracts away data width of stack → Portability 10G, 100G



[3] https://git.esa.informatik.tu-darmstadt.de/net/bluenet

Bluespec Library for Designing new User Kernels (2)

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bluespec code example: UDP echo server

```
1  import NetworkLib :: *;
2  ...
3
4  NetworkLib#(512, 512) nw <- mkNetworkLib(cfg);
5
6  rule echo;
7      // Reply to the sender of the received packet ...
8      let m <- nw.udp.getMeta();
9      nw.udp.setMeta(m.dst_port, m.src_port, m.their_addr, m.length);
10
11     // ... with the contents of the received packet.
12     let d <- nw.udp.datawordRx.get();
13     nw.udp.putData(d.data, d.word_bytes);
14 endrule
```

Bluespec Library for Designing new User Kernels (2)

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bluespec code example: UDP echo server

```
1  import NetworkLib :: *;
2  ...
3
4  NetworkLib#(512, 512) nw <- mkNetworkLib(cfg);
5
6  rule echo;
7      // Reply to the sender of the received packet ...
8      let m <- nw.udp.getMeta();
9      nw.udp.setMeta(m.dst_port, m.src_port, m.their_addr, m.length);
10
11     // ... with the contents of the received packet.
12     let d <- nw.udp.datawordRx.get();
13     nw.udp.putData(d.data, d.word_bytes);
14 endrule
```

Bluespec Library for Designing new User Kernels (2)

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bluespec code example: UDP echo server

```
1  import NetworkLib :: *;
2  ...
3
4  NetworkLib#(512, 512) nw <- mkNetworkLib(cfg);
5
6  rule echo;
7      // Reply to the sender of the received packet ...
8      let m <- nw.udp.getMeta();
9      nw.udp.setMeta(m.dst_port, m.src_port, m.their_addr, m.length);
10
11     // ... with the contents of the received packet.
12     let d <- nw.udp.datawordRx.get();
13     nw.udp.putData(d.data, d.word_bytes);
14 endrule
```

Bluespec Library for Designing new User Kernels (2)

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bluespec code example: UDP echo server

```
1  import NetworkLib :: *;
2  ...
3
4  NetworkLib#(512, 512) nw <- mkNetworkLib(cfg);
5
6  rule echo;
7      // Reply to the sender of the received packet ...
8      let m <- nw.udp.getMeta();
9      nw.udp.setMeta(m.dst_port, m.src_port, m.their_addr, m.length);
10
11     // ... with the contents of the received packet.
12     let d <- nw.udp.datawordRx.get();
13     nw.udp.putData(d.data, d.word_bytes);
14 endrule
```

Simulator for TCP/IP-capable PEs

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

- TCP/IP-capable PEs are complex and interactive
  - → Need simulator for efficient development and debugging
- Simulation abstraction layer
  - HDL-sim is slower than C-sim / Bluesim …
  - …but simulates fully integrated mixed-language design (HLS, BSV, SystemVerilog)
- ”In-Circuit” Emulation of PEs[4]
  - DUT interacts with network like the real hardware would
  - Hardware debugging using ”Wireshark”, ”netcat”, ”ping” etc.

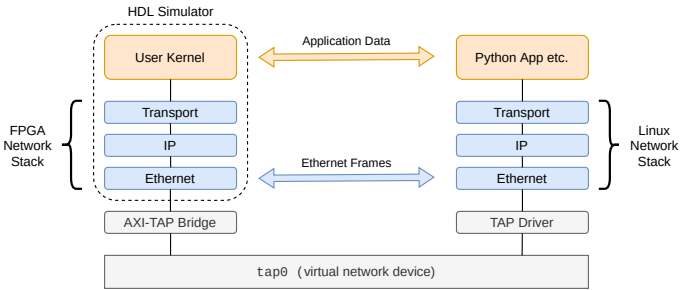
[4] `https://git.esa.informatik.tu-darmstadt.de/net/net-sim`

Simulator for TCP/IP-capable PEs (2)

Contribution

TECHNISCHE UNIVERSITÄT DARMSTADT

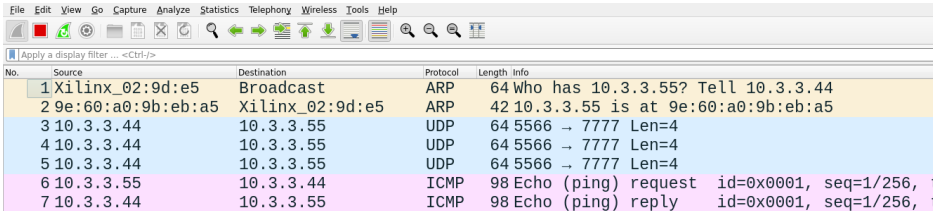
- C++-based testbench, Xilinx XSI
- Top-level AXI ports ⇔ virtual NIC (TAP) of host OS (”AXI-TAP-Bridge”)
- Packet-level interaction **and** HDL signal-level insight



Simulator for TCP/IP-capable PEs (3)

Contribution

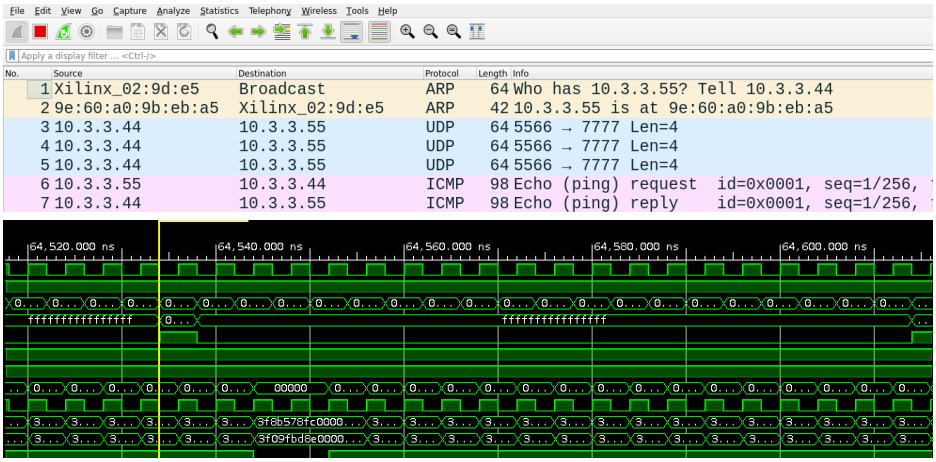
TECHNISCHE UNIVERSITÄT DARMSTADT



Simulator for TCP/IP-capable PEs (3)

Contribution

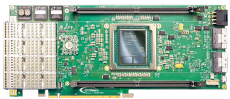
TECHNISCHE UNIVERSITÄT DARMSTADT



Evaluation of TCP/IP-capable FPGA designs

Evaluation Setup

Evaluation



Source: www.bittware.com/files/XUPVVH-flat-800px1.svg

Source: www.xilinx.com/content/dam/xilinx/imgs/products/alveo/alveo.png

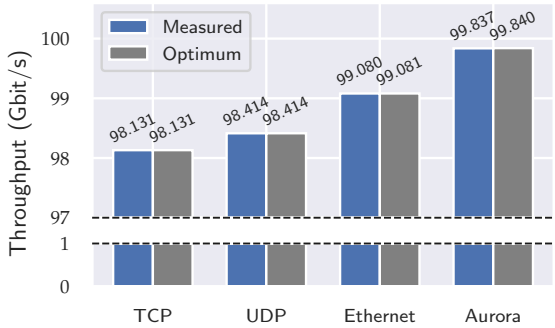
- Determine throughput, latency, resource utilization using synthetic benchmarks
- Comparison of different configurations and protocols
- Protocols: TCP, UDP, plain Ethernet, Xilinx Aurora (lightweight PTP link-layer protocol)

Throughput, Payload Size 4 KiB

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Throughput: inversely proportional to protocol overhead
- All protocols perform close to theoretical maximum (”goodput”)
- Aurora is not packet-based
  - Zero header overhead
  - ”Clock compensation” overhead (Utilization < 100 %)

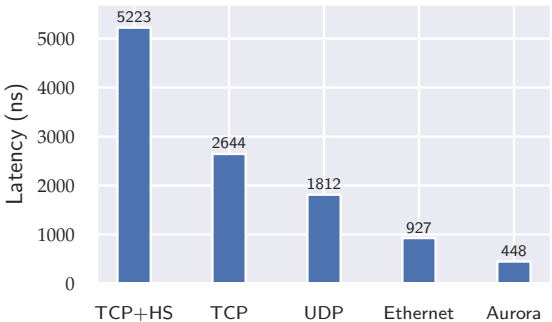


RTT Latency

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- Hardware has deterministic runtime
  $\Rightarrow$ Low variance in results
- Latency: proportional to protocol overhead
  - TCP and UDP inherit latency of Ethernet
- TCP Handshake is slightly faster than RTT

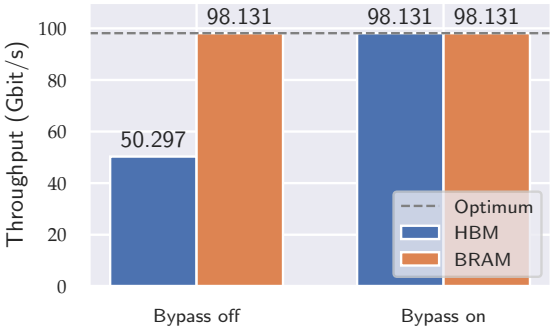


RX Buffer Implementation: TCP throughput, MSS 4 KiB

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- **HBM**-based TCP buffers
  - HBM: 12GB/s combined read/write
    $\Rightarrow$ Throughput is memory-limited
- **BRAM**-based TCP buffers
  - No throughput limitation
- **Buffer bypass**: directly deliver data to app
  - No throughput limitation
  - RTT latency decreases by 27 % (BRAM) and 29 % (HBM)

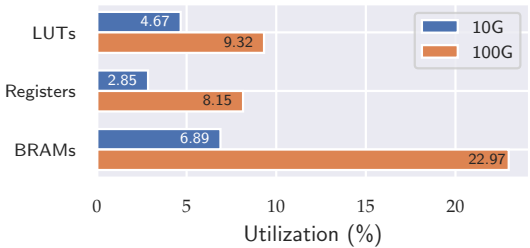


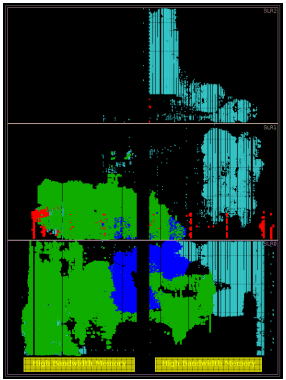
Resource Utilization

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

- TCP/IP stack variants
  - 10G: 64 bit data width
  - 100G: 512 bit data width
- Utilization on XUP-VVH:





Floorplan: XUP-VVH, 100G, TCP Echo Server

Case Study: In-Network Acceleration of Sum-Product Network Inference

SPN Background

Case Study

TECHNISCHE UNIVERSITÄT DARMSTADT

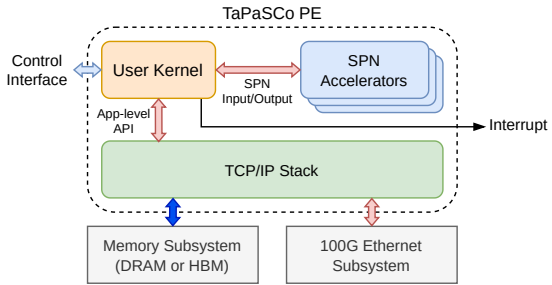
- ML technique from the class of probabilistic graphical models
  - Capture joint probability over a set of random variables
  - Node types: Sum, Product, Leaf
- Prior work
  - Automatic toolflow for FPGA acceleration of SPN inference
  - Maps SPN graph to fully spatial, pipelined accelerator
- Adapted for streaming based data model in this work
  - Free-running kernel with AXI-Stream interfaces

SPN Accelerator Architecture

Case Study

TECHNISCHE UNIVERSITÄT DARMSTADT

- Replicate those SPN accelerator variants with short input-vectors
- Process 50 Byte per cycle at 250 MHz $\rightarrow$ 12.5 GB/s = 100 Gb/s
- ”User Kernel”: contains control logic
  - MIMO: width conversion TCP/IP $\leftrightarrow$ SPN
- TaPaSCo: easily switch memory subsystems (DRAM, HBM)



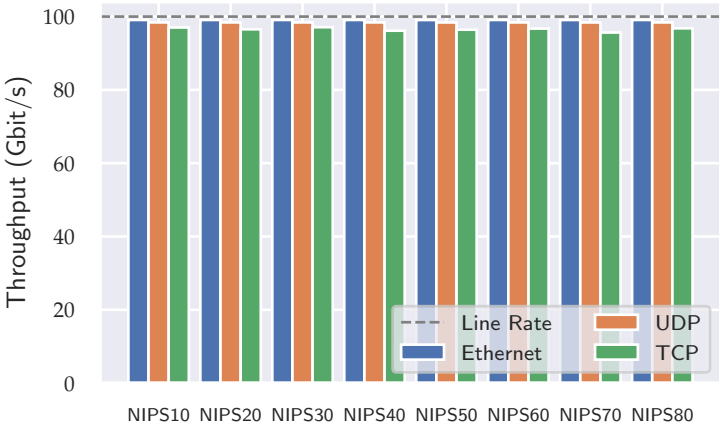
TaPaSCo PE Architecture

SPN Benchmark Results

Case Study

TECHNISCHE UNIVERSITÄT DARMSTADT

- Inference throughput: TCP, UDP and plain Ethernet
- Results are close to the 100G line rate
- Real-world application delivers same performance as synthetic benchmarks



Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Successfully integrated TCP/IP stack with the TaPaSCo framework
   - Open source Bluespec network library [5]
   - Open source HW/SW Co-Simulator [6]
2. Evaluation of TCP/IP-capable FPGA designs using synthetic benchmarks
   - Close-to-optimal throughput and low, deterministic latency
3. Field study: In-Network Acceleration of Sum-Product Network Inference
   - Close-to-optimal inference performance
   - Streaming-based SPN accelerators shown to keep up with 100G line rate

[5] https://git.esa.informatik.tu-darmstadt.de/net/bluenet
[6] https://git.esa.informatik.tu-darmstadt.de/net/net-sim

Thank you for your attention!