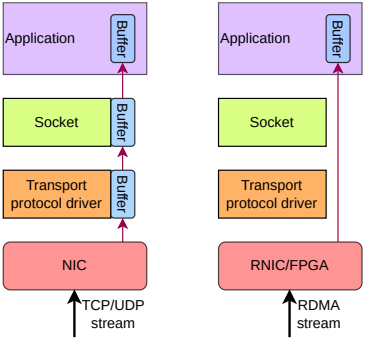
Front-End Rdma Over Converged Ethernet, real-time firmware simulation

BERGNOLI Antonio[1], BORTOLATO Damiano[2], BORTOLATO Gabriele[1,a,b], MENGONI Daniele[1,a], MIGLIORINI Matteo[1,a], MONTECASSIANO Fabio[1], PAZZINI Jacopo[1,a], TRIOSSI Andrea[1,a], VENTURA Sandro[1], ZANETTI Marco[1,a]

[1]INFN sezione Padova, [2]INFN LNL, [a]Università degli studi di Padova, [b]CERN

Introduction on RDMA and RoCE

In a DAQ system a large fraction of CPU resources is engaged in networking rather than in data processing; common network stacks that take care of network traffic usually manipulate data through several copies.



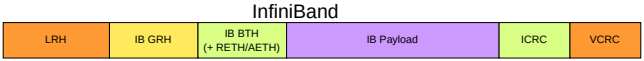
Remote Direct Memory Access (RDMA), as the name suggests, allows read and write operations directly in the target machine(s). This implies no OS involvement allowing high-throughput and low-latency applications.

This requires RDMA enabled NICs on both ends (RNIC) that perform the DMA, reducing the CPU load.

RDMA protocols

Many RDMA flavours are available:

- InfiniBand, it requires IB capable switches
- RoCEv1, it introduces the Ethernet framing, enable use of commodity switches
- RoCEv2, it adds the UDP/IP transport protocol

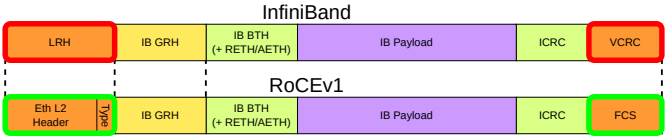


- Local and Global Route Headers
- Base and Extended Transport Headers

RDMA protocols

Many RDMA flavours are available:

- InfiniBand, it requires IB capable switches
- RoCEv1, it introduces the Ethernet framing, enable use of commodity switches
- RoCEv2, it adds the UDP/IP transport protocol

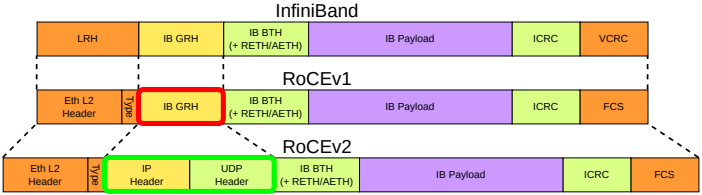


- Eth L2 Header instead of LRH

RDMA protocols

Many RDMA flavours are available:

- InfiniBand, it requires IB capable switches
- RoCEv1, it introduces the Ethernet framing, enable use of commodity switches
- RoCEv2, it adds the UDP/IP transport protocol

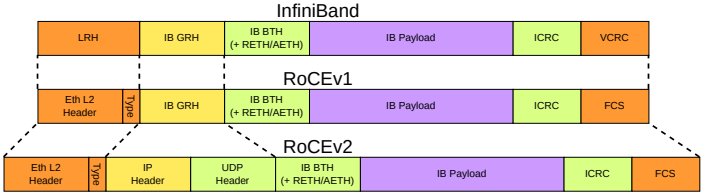


- Drop the use of Global ID (GID) in favour of IP (RoCEv2 UDP port number 4791)

RDMA protocols

Many RDMA flavours are available:

- InfiniBand, it requires IB capable switches
- RoCEv1, it introduces the Ethernet framing, enable use of commodity switches
- RoCEv2, it adds the UDP/IP transport protocol ←



RoCEv2 is the only industry-standard Ethernet-based RDMA solution with a multi-vendor ecosystem. For this reason it has been chosen as target protocol.

Honourable mention

- iWARP, congestion-aware protocols, but higher complexity

Front-End RDMA over Converged Ethernet

Front-End RDMA over Converged Ethernet

Constant trend in producing larger and larger dataset in almost every experimental physics field, new requirements arise form that:

- High throughput, low latency
- Efficient data movement

Front-End RDMA over Converged Ethernet

Constant trend in producing larger and larger dataset in almost every experimental physics field, new requirements arise form that:

- High throughput, low latency
- Efficient data movement

Such requirements lead to clever ideas and features:

- Zero-copy protocols such as InfiniBand or RoCE
- Move network protocol directly in the front-end electronics (FPGA)
- Need to be scalable 1/10/100 Gb/s to target different scenarios
- Multi-vendor ecosystem Xilinx/Microchip/Altera

Front-End RDMA over Converged Ethernet

Constant trend in producing larger and larger dataset in almost every experimental physics field, new requirements arise form that:

- High throughput, low latency
- Efficient data movement

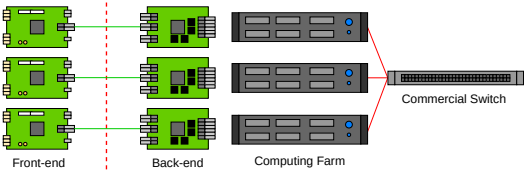
Such requirements lead to clever ideas and features:

- Zero-copy protocols such as InfiniBand or RoCE
- Move network protocol directly in the front-end electronics (FPGA)
- Need to be scalable 1/10/100 Gb/s to target different scenarios
- Multi-vendor ecosystem Xilinx/Microchip/Altera

What can we achieve?

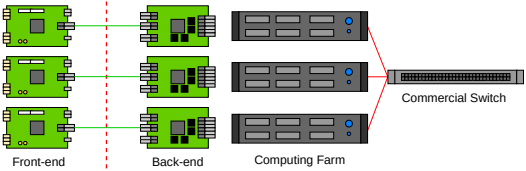
- Front-end initiates the RDMA transfer
- No point-to-point connection between front-end back-end
- Dynamical switching routing with COTS (lowering the costs and maintenance)

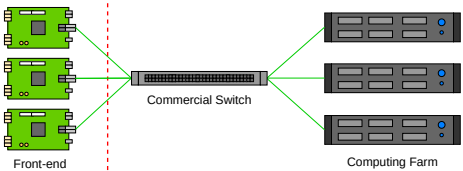
What is FERoCE?



Back-end boards required to get the data, and send it to the computing farms. This requires multiple custom cards and custom boards

What is FERoCE?

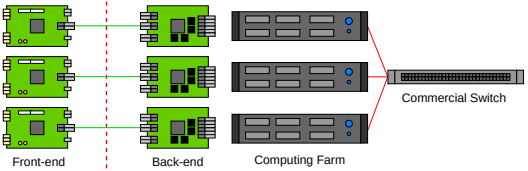


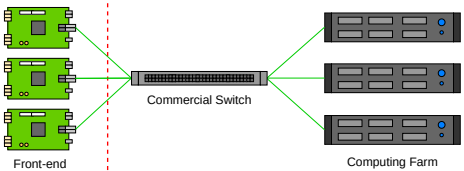


Back-end boards required to get the data, and send it to the computing farms. This requires multiple custom cards and custom boards

Front-end boards send data already packaged within an ethernet frame allowing switching and routing. Choosing the proper protocol allows the use of COTS switches

What is FERoCE?





Back-end boards required to get the data, and send it to the computing farms. This requires multiple custom cards and custom boards

Front-end boards send data already packaged within an ethernet frame allowing switching and routing. Choosing the proper protocol allows the use of COTS switches

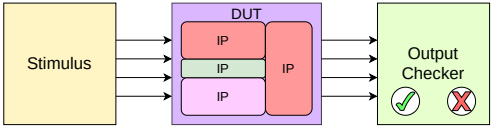
ETH RDMA network stack library has been chosen for the first prototype. Some of its characteristics:

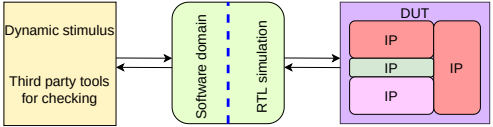
- Entirely written in HLS (Vivado 2019.1)
- It targets Xilinx FPGA with PCIe connection
- 10/100 Gb/s speeds
- It supports UDP, TCP and RDMA

Systems @ ETH zürich

Real-time Firmware simulation

Why a dynamic firmware simulation is needed?





- Narrow test-case, limited by the stimulus
- Difficult to evaluate the RoCE stream produced
- Easy to set-up

- Explore wider test-case phase space
- Feed/Get ethernet frames directly to/from the code
- Simulate the HDL produced starting from the HLS code
- Capture frames with third party programs (e.g. Wireshark)
- Possibility to treat it as a device and send frames to Soft-RoCE or to a physical RNIC

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

- Works on Linux machines: Tun/Tap devices

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

- Works on Linux machines: Tun/Tap devices
- It makes use of DPI-C interface of SystemVerilog: C code in our testbench!

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

- Works on Linux machines: Tun/Tap devices
- It makes use of DPI-C interface of SystemVerilog: C code in our testbench!
- Tap device exchanges raw ethernet frames between simulation and Linux network stack

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

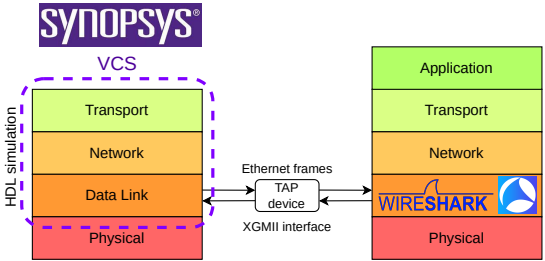
- Works on Linux machines: Tun/Tap devices
- It makes use of DPI-C interface of SystemVerilog: C code in our testbench!
- Tap device exchanges raw ethernet frames between simulation and Linux network stack
- We can capture such frames and study them

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

- Works on Linux machines: Tun/Tap devices
- It makes use of DPI-C interface of SystemVerilog: C code in our testbench!
- Tap device exchanges raw ethernet frames between simulation and Linux network stack
- We can capture such frames and study them

Simulation with Synopsys VCS. XGMII interface directly form Xilinx MAC



Capture and analyze packets, are they malformed? Are the RoCE parameters sent correctly?

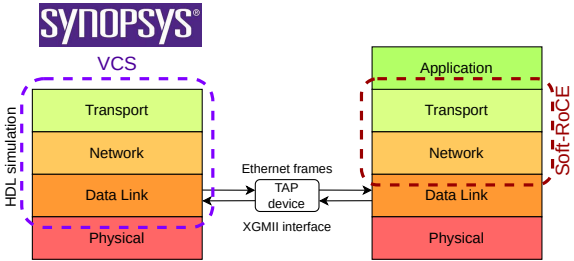
Once the stack has been verified, firmware can be eventually built (Resources? Performances? Is timing closure reached?)

Real-time Firmware Simulation

Start form ETH network stack entirely developed in HLS. Functionalities and features must be understood: real-time firmware simulation with real network traffic.

- Works on Linux machines: Tun/Tap devices
- It makes use of DPI-C interface of SystemVerilog: C code in our testbench!
- Tap device exchanges raw ethernet frames between simulation and Linux network stack
- We can capture such frames and study them

Simulation with Synopsys VCS. XGMII interface directly form Xilinx MAC



Soft-RoCE used to capture and store in memory data sent. Enable fast verification of the stack without going through sythesis/implementation every time.

Once the stack has been verified, firmware can be eventually built (Resources? Performances? Is timing closure reached?)

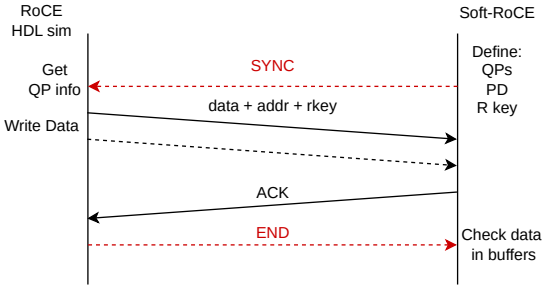
Changes implemented

Some changes have to be made to the stack to enable us to use the AXI-stream port:

- Update FSM for RMDA WRITE:
  - AXI-stream port did not work properly with RDMA WRITE, FSM get stuck if message is too big: support only for WRITE ONLY, WRITE FIRST-MIDDLE-LAST are needed!
- Re-enable and update iCRC computation:
  - By default iCRC was disabled
  - The mask for its computation was wrong
  - Need to solve timing violation here (time multiplex the computation?)
- Add prefix in each IP:
  - Simulator doesn't like IP with same name but different functionality..

RoCE

RoCEv2 is a complex protocol, but not all its features are required for this project. RoCE supports many operations such as: RDMA SEND, RDMA WRITE, RDMA READ, ATOMIC OPERATIONS.



SYNC and END* messages are outside the RoCE protocol

*RDMA WRITE IMMEDIATE command exists that trigger a completion message upon finishing. ETH RDMA network stack doesn't support that.

The goal is only to push data and initiate the RDMA transfer, for this reason only RDMA WRITE is considered.

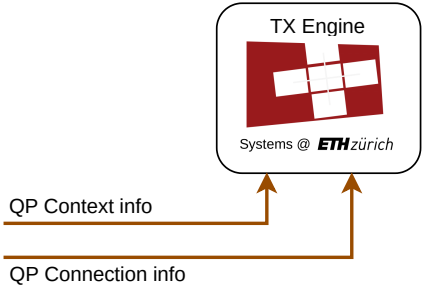
ETH TX engine details

QP Context and connection info contains:

- QP numbers
- Remote and local PSNs
- Remote key
- Virtual address
- Remote IP address

TX metadata contains:

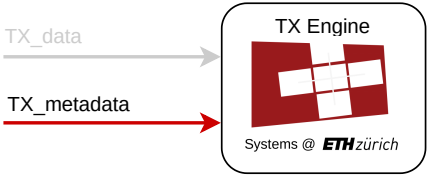
- Operation type
- QP number
- Remote address
- Local address
- DMA length



ETH TX engine details

QP Context and connection info contains:

- QP numbers
- Remote and local PSNs
- Remote key
- Virtual address
- Remote IP address



TX metadata contains:

- Operation type
- QP number
- Remote address
- Local address
- DMA length

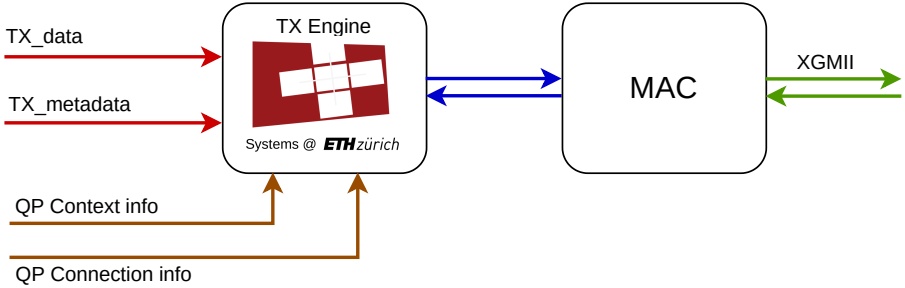
ETH TX engine details

QP Context and connection info contains:

- QP numbers
- Remote and local PSNs
- Remote key
- Virtual address
- Remote IP address

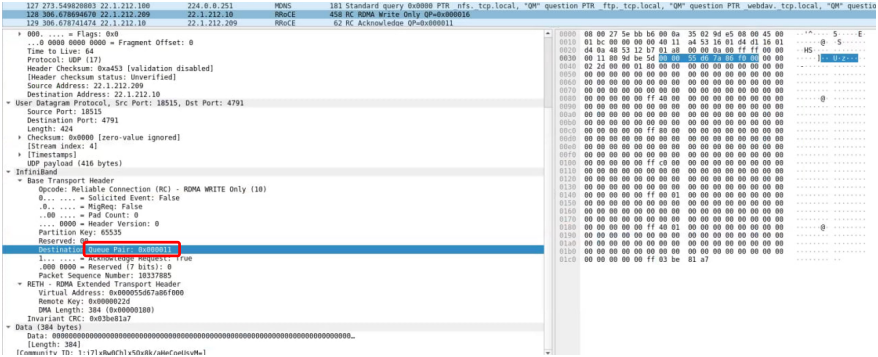
TX metadata contains:

- Operation type
- QP number
- Remote address
- Local address
- DMA length



Some results -Wireshark-

Used Wireshark to capture Ethernet frames coming out of the simulation.

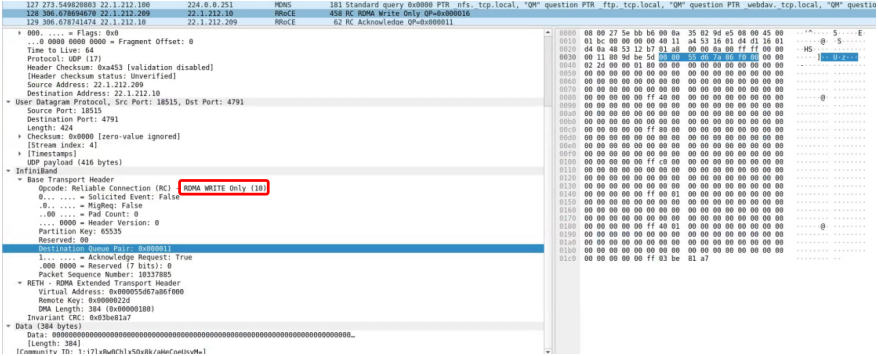


In this frame we can check:

- Queue Pair number
- RDMA OP Code
- IP addresses
- Memory addresses

Some results -Wireshark-

Used Wireshark to capture Ethernet frames coming out of the simulation.

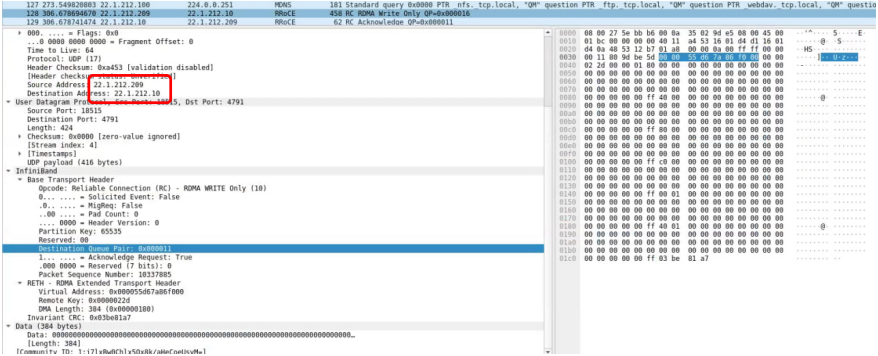


In this frame we can check:

- Queue Pair number
- RDMA OP Code
- IP addresses
- Memory addresses

Some results -Wireshark-

Used Wireshark to capture Ethernet frames coming out of the simulation.

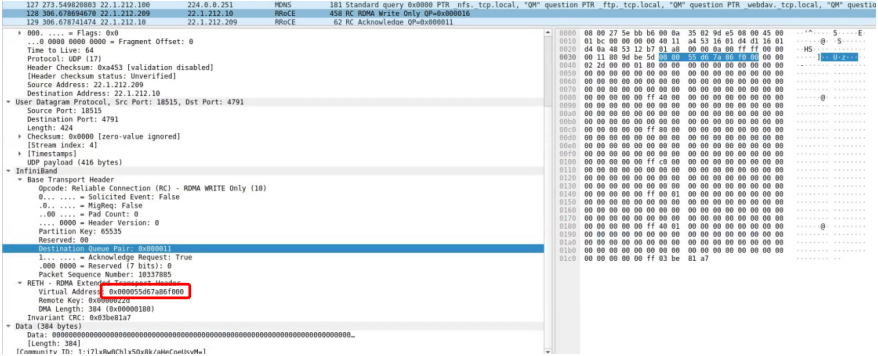


In this frame we can check:

- Queue Pair number
- RDMA OP Code
- IP addresses
- Memory addresses

Some results -Wireshark-

Used Wireshark to capture Ethernet frames coming out of the simulation.



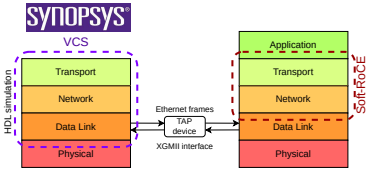
In this frame we can check:

- Queue Pair number
- RDMA OP Code
- IP addresses
- Memory addresses

Summary and Outlook

Summary

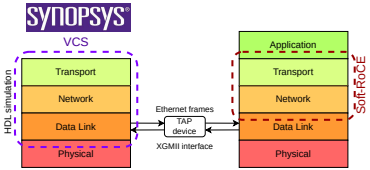
- Developed a dynamic simulation
- Tested and verified ETH network stack
- Fed simulation RoCE data to Soft-RoCE end-point



Summary and Outlook

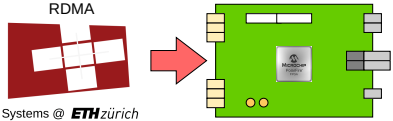
Summary

- Developed a dynamic simulation
- Tested and verified ETH network stack
- Fed simulation RoCE data to Soft-RoCE end-point



Outlook

- Cut ETH library to reduce the FPGA resource footprint
- Move from Xilinx HLS to a more agnostic HLS and/or rewrite a stack's subset in HDL (only RDMA WRITE)
- Delopy the light-RoCE in a Microchip FPGA



References (I)

- System@ETHzürich network stack repository, `https://github.com/fpgasystems/fpga-network-stack`
- System@ETHzürich Distributed OS , `https://github.com/fpgasystems/davos`
- Modified network stack repository (work in progress), `https://github.com/Gabriele-bot/fpga-network-stack`
- CRC mask fix network stack repository, `https://github.com/Nayib/fpga-network-stack`