A high-throughput network approach for interfacing to front end electronics for ATLAS upgrades

Jörn Schumacher
CERN / University of Paderborn, Germany
jorn.schumacher@cern.ch

On behalf of the ATLAS FELIX Developer Team
**ATLAS DAQ:**

Custom point-to-point links

Point-to-point S-links

- **Custom electronic components**
  - FE
  - ROD
  - Frontend
  - ReadOut Driver
  - ReadOut System\(^1\) (Data Buffer)

- **PCs (COTS)**
  - Ethernet
  - High-Level Trigger Farm
  - HLTPU
  - ~1500 servers
  - ~100 servers

- **Timing**

**Run**
- Run 2
- Run 3
- Run 4

\(^1\)Including ROS RobinNP
ATLAS DAQ:

Versatile Link, GBT
PCs
40 Gb Ethernet, Infiniband

Custom electronic components
PCs (COTS)

High-Level Trigger Farm

Run 2 Run 3 Run 4
2015 2017 2019 2021 2023 2025
ATLAS DAQ:

Versatile Link, GBT, LpGBT

COTS network technology

~10,000 links

~200 systems

40 MHz

HPC Network

less than 10 TB/s

PCs (COTS)

High-Level Trigger Farm

2015 2017 2019 2021 2023 2025
**Functionality**

- **Calibration**
- **DCS**
- **FE config**
- **event readout**

Optionally, different IP ports in same HW module.

**TTC**

**COTS network switch**

40Gb/s

Routing map:

- GBT id, E-link id, trigtype...
- $\rightarrow$ IPaddr:port

- N x GBT links per FELIX
  @ 3.2Gb/s each

**GBT**

GBT E-links per function, per FE chip

**E-link**: variable-width logical link on top GBT. Can be used to logically separate different streams on a single link.

**Scalable architecture**

Routing of multiple traffic types: physics events, detector control, configuration, calibration, monitoring

**Industry standard links**: data processors/handlers can be SW in PCs - Less custom electronics, more COTS components

**Reconfigurable data path**, multi-cast, cloning, QoS

**Automatic failover and load balancing**

**TTC integration, LHC clock distribution**
Development Platform

HiTech Global PCIe development
Xilinx Virtex-7
PCIe Gen-2/3 x8
24 bi-directional links
http://hitechglobal.com/Boards/PCIE-CXP.htm
With custom TTCfx FMC

SuperMicro X10DRG-Q
2x Haswell CPU, up to 10 cores
6x PCIe Gen-3 slots
64 GB DDR4 Memory

Mellanox ConnectX-3 VPI
FDR/QDR Infiniband
2x10/40 GbE
**FELIX Demonstrator System**

- **FPGA Card**
  - TTC FMC
  - DMA
  - Config
  - GBT
  - Elink router
- **Memory**
  - Large Buffers per group of elinks
- **CPU**
  - DMA
  - MSI-X
  - Custom Device Driver
  - FELIX Application
- **NIC**
  - PCIe Gen-3
  - 64 Gb/s
- **Optical Links**
  - 64 Gb/s
  - TTC is the Timing, Trigger and Control System

**Key Details**

- 24 – 48 links
- 2 – 4 40-Gb/s ports

**Technical Specifications**

- PCIe Gen-3
- DMA
- MSI-X
- Custom Device Driver
- FELIX Application
Demonstrator Firmware Design

- Different GBT modes (normal, wide, …) supported
- Connection to legacy TTC system via FMC
- Internal data generator for testing

E-link data is encoded in fixed-size blocks before transmitting over PCIe

Legend:
- Main data path
- Slow control and monitor
FELIX DMA Core

DMA interface to the Xilinx Virtex-7 PCIe Gen3 hard block

MSI-X compatible interrupt controller

Developed at Nikhef for use in FELIX

Published as OpenSource (LGPL) on OpenCores
http://opencores.org/project,virtex7_pcie_dma
DMA Core Benchmarks

Bandwidth for single DMA transfers

Speed of transfers into/from a single buffer in host memory was measured for different block sizes.
CPU Data Processing Pipeline

- Read Blocks from File Descriptors
- Block Decode
- Statistics
- Extract Meta Information
- Route
- Load Balancing, Error Handling
- Send To Network
Program DMA transfers and read fixed blocks of data that have been encoded for the transfer over PCIe.

Decode into variable sized chunks for transmission over network.

e-link streams share the same pipeline.
Program DMA transfers, read and decode data that has been encoded for the transfer over PCIe.

### CPU Data Processing Pipeline

- **Optimizations targeting memory throughput**
- **More efficient data layout**
- **Better use of STL containers**

---

<table>
<thead>
<tr>
<th>Read Blocks from File Descriptors</th>
<th>Block Decode</th>
<th>Statistics</th>
<th>Extract Meta Information</th>
<th>Route</th>
<th>Load Balancing, Error Handling</th>
<th>Send To Network</th>
</tr>
</thead>
</table>

---

![Graph showing Total Throughput vs. # Threads](image1)

- **Hyperthreading**

![Graph showing Average Processing Time per Block vs. Chunk size](image2)
Program DMA transfers and read fixed blocks of data that have been encoded for the transfer over PCIe.

Decode into variable sized chunks for transmission over network.

Count processed blocks, transfer rates, etc.
Program DMA transfers and read fixed blocks of data that have been encoded for the transfer over PCIe.

Decode into variable sized chunks for transmission over network.

Count processed blocks, transfer rates, etc.

Metainformation, for example event ID, is extracted and matched against a routing table.
Program DMA transfers and read fixed blocks of data that have been encoded for the transfer over PCIe.

Decode into variable sized chunks for transmission over network.

Count processed blocks, transfer rates, etc.

Metainformation, for example event ID, is extracted and matched against a routing table.

Distribute load among multiple systems.
Handle automatic failover in case of system failures.
Summary

general-purpose data-routing device
connects ATLAS detector frontends to the ATLAS DAQ system
load balancing, automatic failover, routing based on type of data (physics events, control, calibration, …)

Technology demonstrator currently under development
BACKUP
Examples for Functions

**Forward**
- All incoming data of e-link 1
  - to
  - pc-daq-00.cern.ch

**Forward**
- Packets with data[5] == 0xAB
  - to
  - 10.0.0.1:1234

**Forward**
- A random 10% sample of e-link 1
  - to
  - my-monitoring-server

**Forward**
- All incoming data of e-link 2
  - to any of
  - {pc-daq-00, pc-daq-01, pc-daq-02}

Send a complete e-link to a destination

Send only certain packets, marked by a flag in the data stream, to a certain destination

Send a certain percentage of incoming data to a monitoring system

Load balancing: Distribute load among multiple systems