FELIX: a High-Throughput Network Approach for Interfacing to Front End Electronics for ATLAS Upgrades

J Anderson[1], A Borga[4], H Boterenbrood[4], H Chen[2], K Chen[2], G Drake[1], D Francis[3], B Gorini[3], F Lanni[2], G Lehmann Miotto[3], L Levinson[6], J Narevicius[6], C Plessl[5], A Roich[6], S Ryu[1], F Schreuder[4], J Schumacher[3,5], W Vandelli[3], J Vermeulen[4], J Zhang[1]

[1] Argonne National Laboratory, USA
[2] Brookhaven National Laboratory, USA
[3] CERN, Switzerland
[4] Nikhef National Institute for Subatomic Physics / University of Amsterdam, Netherlands
[5] Department of Computer Science, University of Paderborn, Germany
[6] Department of Particle Physics, The Weizmann Institute of Science, Israel

E-mail: `joern.schumacher@cern.ch`

Abstract.
The ATLAS experiment at CERN is planning full deployment of a new unified optical link technology for connecting detector front end electronics on the timescale of the LHC Run 4 (2025). It is estimated that roughly 8000 GBT (GigaBit Transceiver) links, with transfer rates up to 10.24 Gbps, will replace existing links used for readout, detector control and distribution of timing and trigger information. A new class of devices will be needed to interface many GBT links to the rest of the trigger, data-acquisition and detector control systems. In this paper FELIX (Front End LInk eXchange) is presented, a PC-based device to route data from and to multiple GBT links via a high-performance general purpose network capable of a total throughput up to O(20 Tbps). FELIX implies architectural changes to the ATLAS data acquisition system, such as the use of industry standard COTS components early in the DAQ chain. Additionally the design and implementation of a FELIX demonstration platform is presented and hardware and software aspects will be discussed.

1. Introduction

The ATLAS experiment [1] is one of the four experiments at the Large Hadron Collider (LHC) at CERN, Geneva, Switzerland. During operation ATLAS generates data at O(100 Tb/s), mostly from well-known physics phenomena. Only a small fraction is considered suitable for potential new physics discoveries and therefore stored for offline analysis. It is the task of the trigger and data acquisition system to process and filter events in quasi real-time before they are written to permanent storage.

The trigger and data acquisition system is organized in layers. A first rejection (L1 trigger) system reduces the 40 MHz collision rate to approximately 100 kHz. The L1 trigger is entirely implemented using custom electronic components as it needs to accept or reject events within 2.5 microseconds. Upon accept, fragments are forwarded from the on-detector Front End electronics

(FE) to the back-end electronics (ReadOut Drivers, RODs), located in a separate service cavern, over custom point-to-point detector-specific links. The RODs perform data manipulation tasks like aggregation or compression before pushing the data to the circa 100 ReadOut System (ROS) PCs over 1800 point-to-point optical links (S-Link) [2]. ROS PCs buffer event fragments and forward them upon request to the High-Level Trigger (HLT) computer farm, consisting of 1500 servers. Here, the event rate is further reduced to the target event rate of about 1 kHz.

The Front End LInk eXchange (FELIX) project is a novel approach to interfacing the various ATLAS sub-detectors to the data-acquisition system. Starting from the next long shutdown of the LHC in 2018, new detector links will be based on CERN's Versatile Link technology [3] and GBT protocol [4]. A total of about 2000 new links will be deployed. Most of these links will be GBT links but for readout of new off-detector trigger systems FPGA links and simple 8b/10b coding will also be used. Full deployment of FELIX is planned for 2023.

FELIX is a multi-purpose routing device: data can be forwarded to or from multiple destinations connected to a network, e.g. data-acquisition, monitoring, detector control, calibration, or configuration systems. The system is also capable of handling special fixed latency channels, such as needed for transferring Timing, Trigger and Control (TTC) [5] signals via GBT links and low latency channels for Direct-Output-Low-Latency connections to first level trigger electronics. The software layer integrates support for data sampling, quality-of-service prioritization, load balancing based on a round-robin scheme and automatic failover. An overview of the architectural changes of the ATLAS data-acquisition system with FELIX is given in Figure 1.

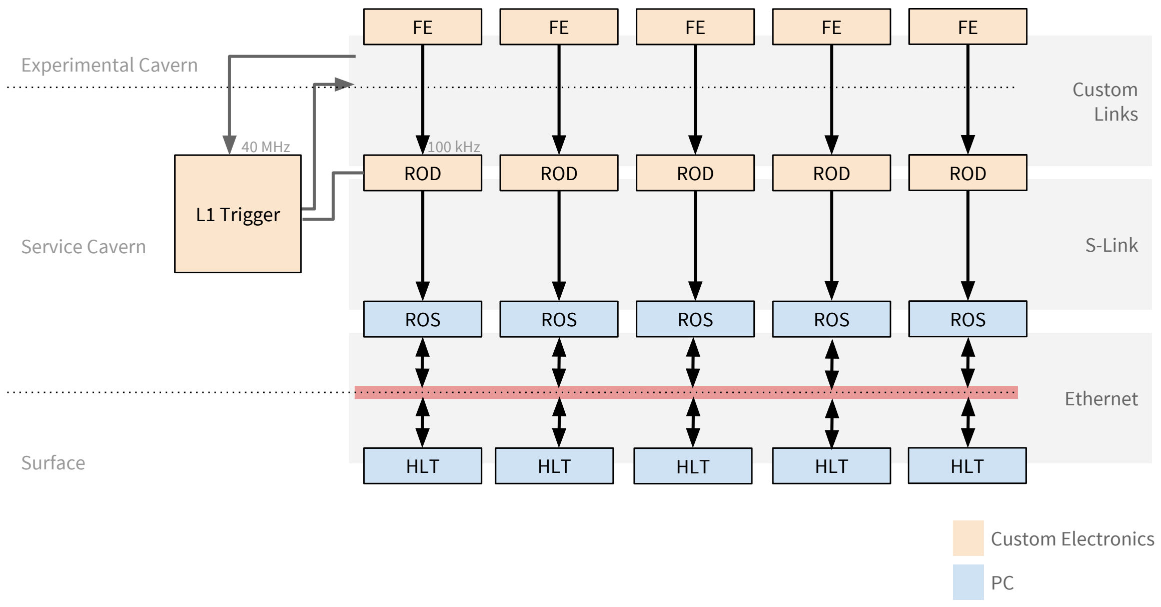
There are multiple improvements that FELIX can bring to the ATLAS data acquisition system concerning operation and maintenance. Scalability is one: the introduction of switched networks early in the DAQ chain allows connecting additional components to the network as needed. In addition the use of standard industry links makes it easy to implement data processors based on commercial off-the-shelf components instead of custom designed electronics. Moreover the flow of data through the system is not fixed and can be configured dynamically. Hence FELIX acts as a central data distribution layer supporting different types of traffic: not only physics event data, but also calibration, control, or trigger and timing data. FELIX supports the use of *E-links*, a feature of the GBT protocol[6]. E-links are serial electrical links, with configurable bandwidth, that are aggregated onto a single high speed optical link. E-links can be used to logically separate different types of data traffic. Data received via different E-links can be sent to different network destinations and data from different network sources can be passed via different E-links multiplexed on the same GBT link.

2. The FELIX demonstrator

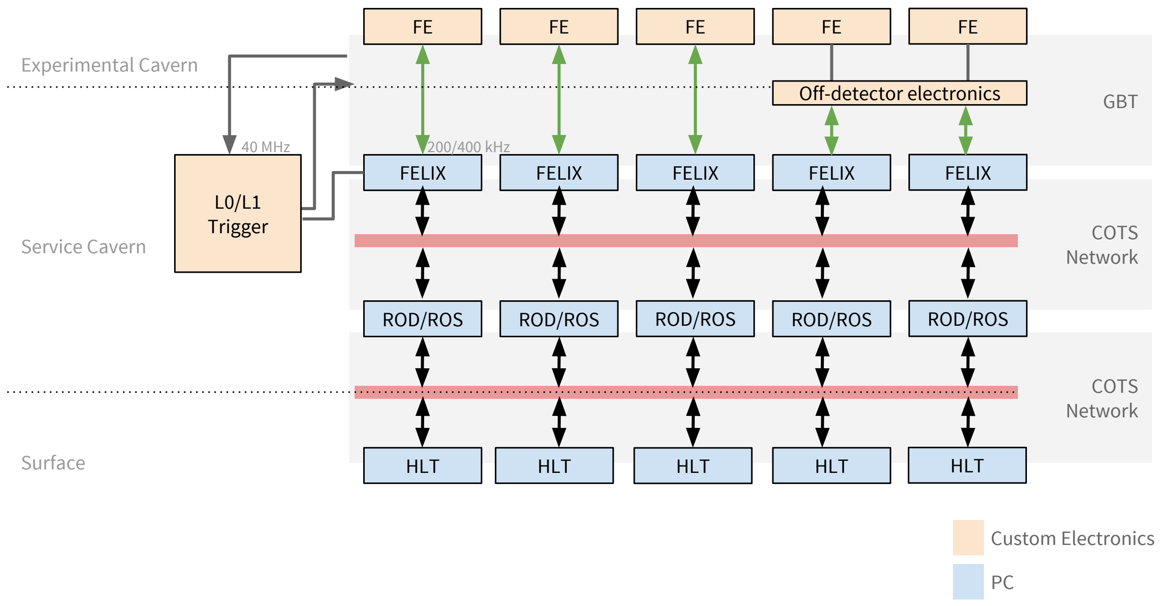
To prove that the FELIX approach is viable, a PC-based demonstration platform implementing the FELIX functionality has been prepared. A commercial PCIe FPGA development card [7] interfaces the detector links. The board is shown in Figure 3. Up to 24 bi-directional optical links can be connected to the card via two 12-way CXP optical transceivers. A custom FMC mezzanine card, TTCfx, provides an additional optical input from the legacy LHC TTC (Trigger, Timing and Control) system [5]. The TTCfx hosts TTC clock and data recovery circuitry and a clock jitter cleaner, as well as a connector for a NIM output that can be used as BUSY signal for throttling the L1 trigger in case of need. The PCIe card is equipped with a Xilinx Virtex-7 FPGA and is connected to the host system via an 8-lane PCIe Gen-3 slot. Network connectivity is provided by a dual-port Mellanox ConnectX-3 card [8], which can be configured either as 40 Gb/s Ethernet or as 56 Gb/s Infiniband interface.

3. The PCIe card and its firmware

The core component of the PCIe card is a Xilinx Virtex-7 690 FPGA [9] responsible for: (i) GBT link handling (implemented in the GBT wrapper component), (ii) data routing and (iii) data transfer to and from the host. The FPGA also handles the timing and trigger information distributed via the TTC system. TTC information is routed with fixed and low latency to multiple E-links on connected GBT links. In addition slow control (configuration and



(a) Current architecture



(b) Anticipated architecture for LHC Run 4 (2025), some detectors will deploy FELIX for Run 3 (2019).

Figure 1. ATLAS data-aquisition architecture with and without FELIX.

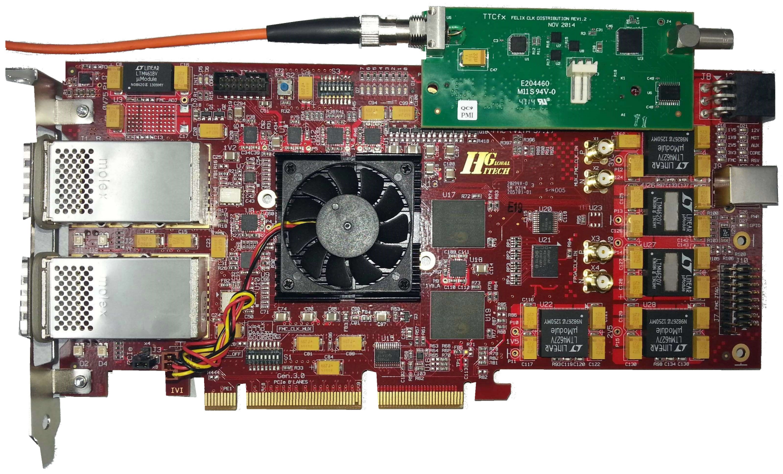


Figure 2. The HTG-710 FPGA development card from HiTech Global used in the demonstrator system. For each of the two CXP cages at the left twelve bi-directional links can be connected via a CXP transceiver installed in the cage. The custom TTCfx board to connect to the TTC system is inserted in an FMC connector at the top of the board.

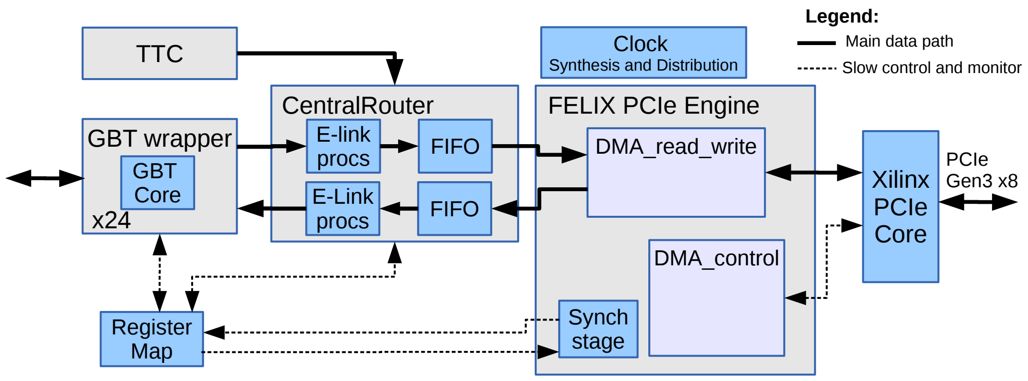


Figure 3. Simplified data flow in the FPGA. Elements are explained in the text.

calibration) data must be transmitted to the Front Ends. There are therefore three main data paths: TTC and slow control data to the Front Ends and event data from the Front Ends to the host PC. Figure 3 shows a simplified block diagram of the data flow in the FPGA.

Each of the GBT links has up to 42 E-links for which serial data are accumulated, packet boundaries discovered and, if required, 8b/10b decoding (or encoding) is performed. Each E-link has a dedicated FIFO for queuing its packets for transfer to the host, managed in the CentralRouter component in the firmware. Similar logic operates in the reverse direction. The Central Router will also handle the HDLC encoding/decoding required for communication to Slow Control Adapter ASICs that may be connected to an E-link at the Front End. There are potentially about 1000 E-link FIFOs for 24 GBTs. The Central Router formats the E-link data into blocks for transfer and arbitrates their transfer to the PCIe engine. (See Section 4.2)

Data are streamed from the FPGA to host PC memory for packet processing and routing, as described in more detail in Section 4. In order to sustain the high data rates (61.4 Gb/s for 8b/10b coded data received via 24 GBT links each with 40 80 Mb/s E-links, after decoding), a high-bandwidth interface is required towards the PC. A custom PCIe engine has been developed for this purpose, which has been published as an OpenCore project [10].

The PCIe engine provides a simple Direct Memory Access (DMA) interface for the Xilinx Virtex-7 PCIe Gen3 hard block [11], specifically designed for the 256 bit wide ARM AMBA [12] AXI4-Stream interface [13] of the hard block. The FELIX PCIe Engine provides an interface to a standard FIFO with the same width as the Xilinx AXI4-Stream interface (256 bits) and runs at 250 MHz, providing a theoretical throughput of 65 Gb/s. The user application side of the FPGA then simply reads or writes to the FIFO; the PCIe Engine handles the transfer into or from host PC memory, according to the addresses and transfer direction specified in the DMA descriptors. Another functionality of the core is to manage a set of DMA descriptors, with an $address$, a $read/\overline{write}$ flag, the $transfersize$ (number of 32 bit words) and an $enable$ line. These descriptors are mapped as normal PCIe memory or IO registers. A status register for every descriptor is provided in the register map in addition to the descriptors and the enable lines (one per descriptor). This allows detecting pending/processed requests and also permits descriptor reuse, so that automatic cyclic transfer operations with wrap-around addressing can be performed. In this case DMA transfers halt after an address, supplied by the host, is used for a transfer and continue after a new address is supplied. This way the host can throttle the DMA transfers as needed. The PCIe Engine also supports user configurable MSI-X interrupts and further user defined IO registers can be instantiated for custom purposes. The register map is synchronized inside the PCIe Engine to an independent lower clock frequency to ease design timing closure.

4. Software Data Processing Pipeline

A application written in C^{++} is running on the FELIX host and processes data arriving on the detector links and network interfaces. The main task of the application is to process packets sent by the detector Front Ends, which is implemented in a software pipeline, depicted in Figure 4.

4.1. Device Driver: Interfacing the FPGA

The application software running on the PC communicates with the firmware via a dedicated device driver, the FELIX driver. This driver has been derived from the one used in the current ReadOut System (ROS) PCs for communicating with the RobinNP PCIe cards [14]. It provides basic support for reading from and writing to the registers of the PCIe interface and for interrupt handling. On loading the driver initializes an interrupt vector table so that application software can wait for an interrupt to occur by means of a call to the ioctl function. The driver does not allocate memory for storing data to be transferred from or to a FELIX PCIe card under DMA control and also does not handle DMA transfers, rather these are setup by the user software. Also the memory to or from which data is transferred under DMA control must be managed by the user software. This is facilitated by a second driver, the cmem driver that reserves, at the time it is loaded, contiguous memory in the physical address space, so that no scatter-gather DMA functionality and associated management is needed. An available API abstracts lower layers of the PCIe interface and of the cmem driver. For test purposes a third driver is used

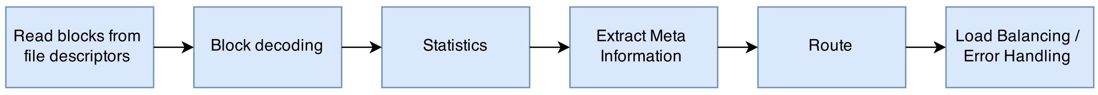

Read blocks from file descriptors
Block decoding
Statistics
Extract Meta Information
Route
Load Balancing / Error Handling

Figure 4. The pipeline architecture of the FELIX application for data being sent from detector Front Ends to the ATLAS data acquisition system.

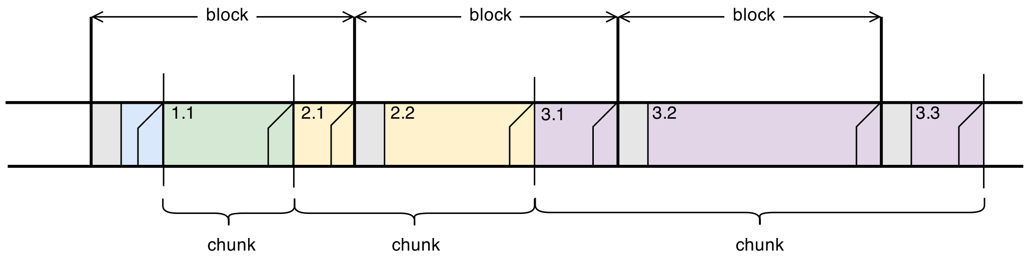


Figure 5. The data format in which variable-length chunks are encoded in fixed-length blocks prior to transmission over the PCIe bus. Chunks are broken into so-called subchunks. Each block has a 4-byte header (grey rectangle), each subchunk has a 2-byte trailer (slanted shape).

(the io driver) that only provides simple access to PCIe configuration space and BAR regions. This driver has originally been developed for use in the ROS.

4.2. PCIe Packet Processing

Variable-length packets (so-called *chunks*) arriving on the GBT links are encoded in the FPGA into fixed-length 1 kB blocks which are then transmitted over PCIe. The data format of the blocks is depicted in Figure 5. Chunks may need to be split up in smaller parts, called *subchunks*, to fit in the blocks. A subchunk can be of one of four types: either it is a full chunk or it is the beginning, middle or end part of a chunk. Subchunks have a 2-byte trailer containing information about their type and length.

The encoded 1 kB blocks are transmitted to circular buffers in the host memory. The encoding of chunks in blocks has to be undone in software: the original variable-length chunks have to be recreated before being transmitted over the network.

The processing of a block starts at the end of a block, where a subchunk trailer is always found. Using the length information in the trailer, the subchunk can be reconstructed and depending on the type (full chunk, beginning, middle, end of a chunk) it is either added to the current chunk or forms a new chunk. The algorithm avoids copying data: when adding a subchunk to a chunk, no data is copied, rather only a pointer to the subchunk data is stored.

Analysis of the block processing algorithm revealed that its performance is memory-bound. Several optimizations targeting memory accesses improved the performance of the implementation. An additional observation was that for blocks containing many small chunks rather than fewer large chunks it is much more likely that a chunk can be represented by a single subchunk. Specifically optimizing for this scenario by adding a specialized datatype for single-subchunk chunks could further improve performance. Figure 6 shows the average block processing time per block for the original implementation, an optimized version, and an optimized version with using the additional single-subchunk datatype. Especially for small chunksizes the performance improved significantly due to the new datatype and the other optimizations. A more detailed analysis of the block processing algorithm performance and the performed optimizations is given in [15].

The FPGA copies blocks into circular buffers in main memory using DMA transfers. When multiple buffers are used, the block processing algorithm can be parallelized by assigning one processing thread to each buffer. This was simulated on a 12-core system (24 cores with HyperThreading enabled), the result is shown in Figure 7. As expected, the block processing throughput scales linearly with the number of threads before it saturates when one thread per cpu core is reached.

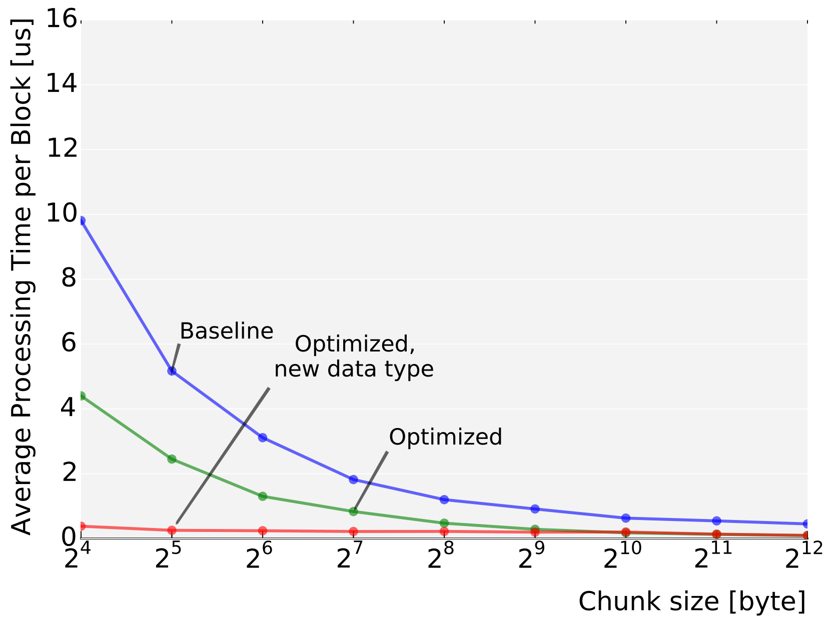


Figure 6. Average runtime per processed block for different stages of the optimization process.

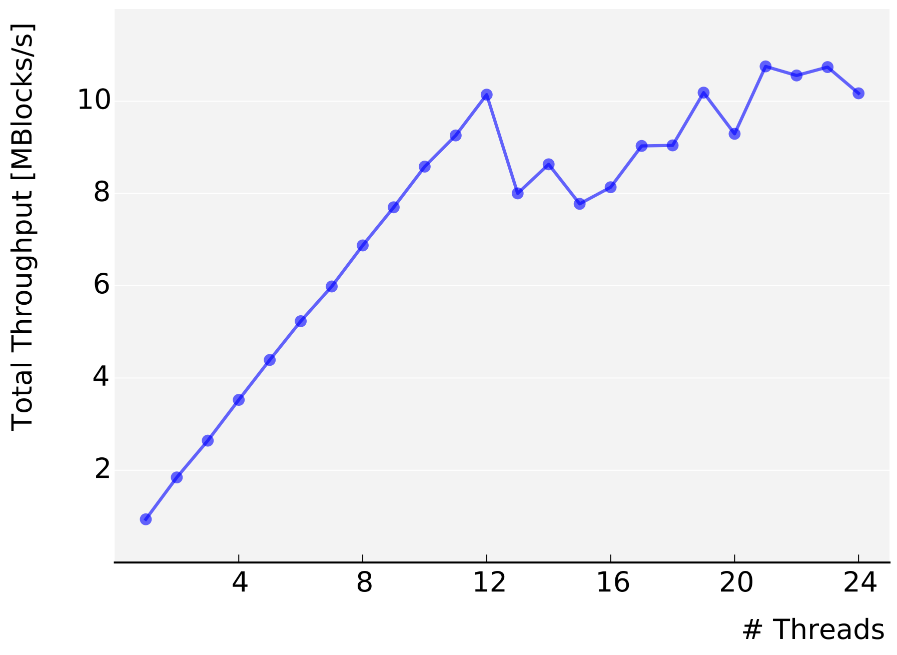


Figure 7. A benchmark of the packet processing algorithm that decodes the fixed-length blocks transmitted over PCIe via the DMA engine of the FPGA. Shown is the total aggregated throughput for the optimized packet processing as function of the number of threads utilized. The throughput saturates at 12 threads, the number of physical cores of the benchmark system.

4.3. Routing and Load Balancing
A simple routing scheme is used in the FELIX demonstrator. Multiple network endpoints can be set for each E-link. Chunks on a particular E-link are forwarded to one of these configured network endpoints. When multiple possible network endpoints are given the destination is chosen based on round-robin scheme, resulting in a simple load balancing among multiple hosts.

In future production versions of FELIX it is planned to support other routing schemes, e.g. routing by traffic type (calibration, physics events, etc.) or load balancing based on event identifiers, as well as features like quality-of-service or monitoring.

4.4. Network Connection
The network connectivity of FELIX is implemented in a library *netio*, which abstracts the low level network implementation and provides a unified interface to the FELIX application. Netio has a pluggable architecture and can support multiple implementations or network technologies via different backends. For the FELIX technology demonstrator a portable backend based on synchronous POSIX sockets is implemented. Future backends could, for example, support RDMA over Infiniband or other network technologies.

5. Conclusions
FELIX is a general-purpose data-routing device to be integrated in the ATLAS data acquisition system beginning in Run 3 of the LHC (2018). It introduces the usage of industry standard switched networks early in the DAQ chain. The goal of the FELIX project is to provide a standard data distribution layer between the ATLAS detector Front Ends and the various data processing components. A DAQ architecture with FELIX allows to easily scale data processing capabilities by adding data processors as needed.

A technology demonstrator is currently in preparation to show the viability of the FELIX concept. The demonstrator is based on a PC platform using a custom-firmware FPGA-based PCIe card for the detector link connectivity and packet sorting. Data routing and the connection to the COTS network is implemented in a software pipeline running on the FELIX host PC. Targeted optimizations improved the throughput of the PCIe packet processing algorithm. The packet processing performance satisfies the FELIX requirements.

References

[1] The ATLAS Collaboration *Journal of Instrumentation* **3** S08003
[2] S-Link URL `http://hsi.web.cern.ch/HSI/s-link/`
[3] L Amaral et al 2009 *Journal of Instrumentation* **4** P12003
[4] P Moreira et al 2009 *Topical Workshop on Electronics for Particle Physics* 342–346
[5] TTC URL `http://ttc.web.cern.ch/TTC`
[6] The GBT Project 2015 URL `https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf`
[7] HiTech Global URL `http://www.hitechglobal.com/Boards/PCIE-CXP.htm`
[8] Mellanox Technologies URL `http://www.mellanox.com/`
[9] Xilinx Virtex-7 URL `http://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html`
[10] Borga A and Schreuder F 2015 URL `http://opencores.org/project,virtex7_pcie_dma,overview`
[11] Xilinx URL `http://www.xilinx.com/support/documentation/ip_documentation/pcie3_7x/v3_0/pg023_v7_pcie_gen3.pdf`
[12] ARM URL `http://www.arm.com/products/system-ip/amba/amba-open-specifications.php`
[13] Xilinx URL `http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf`
[14] Andrea Borga et al 2014 Evolution of the ReadOut System of the ATLAS experiment Tech. rep. URL `https://cds.cern.ch/record/1710776`
[15] John Anderson et al 2015 (forthcoming, expected publication in 2015) URL `https://cds.cern.ch/record/1984495`