ATLAS Internal Note
DAQ-NO-084
26 March 1998

Demonstrator Results Architecture "A"

A.Kugel, K.Kornmesser, R. Lay, J. Ludvig, R. Männer, K.-H. Noffz
S. Rühl, M. Sessler, H. Simmler, H. Singpiel

Lehrstuhl für Informatik V
Universität Mannheim

V. Dörsing, W. Erhard, P. Kammel, A. Reinsch, T. Schober

Lehrstuhl für Rechnerarchitektur und -kommunikation
Universität Jena

March 26, 1998

1 Introduction

The ATLAS level-2 trigger system has to handle several tasks to process data from different detectors. Three of this tasks under investigation in demo A are described here

- The raw data has to be collected, synchronized, arranged to Regions-of-Interest (RoI) and transmited to the Feature Extraction for further processing.
- The high-luminosity trigger has to reduce the input event frequency of 100kHz to a output rate of 1kHz. This reduction can be done in several steps. One of these steps is the RoI-guided feature extraction of the Transition-Radiation-Tracker (TRT). This step was implemented in an FPGA processor called Enable++[HKL+95] to show the feasibility of using FPGAs.
- The low-luminosity trigger has to process the whole detector data of the TRT for doing B-physics studies. The feature extraction has to reconstruct tracks down to 0.5 GeV. This feature extraction has to be done with an event frequency of up to 10 kHz.

To cope with these high event frequencies and with its enormous data rates the FPGA processor Enable++ was developed at the university of Mannheim and the Region of Interest Collector (RoIC) at the university of Jena. Enable++ and RoIC are based on functional modules which are scalable and can be adjusted in computing and I/O functionality to the respective tasks.

To solve the tasks of the ATLAS second level trigger Enable++ and RoIC are embedded into the entire trigger architecture. Figure 1 shows a possible architecture.

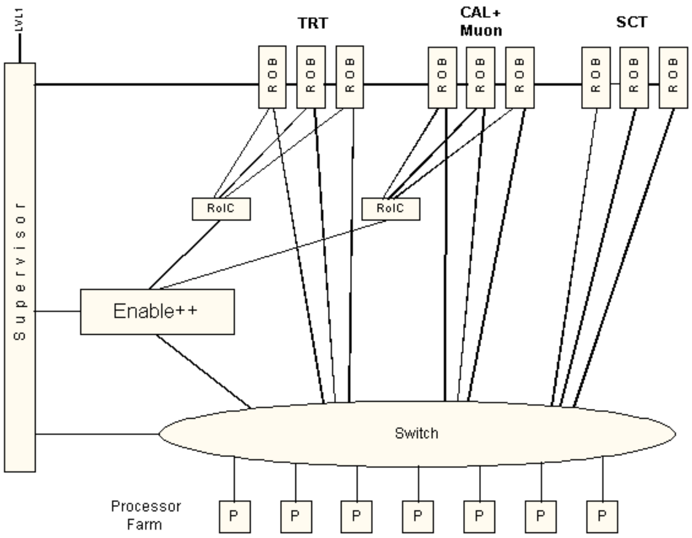

Figure 1: Possible ATLAS level-2 trigger architecture

The Enable++ data input is connected via the RoI-Collector (RoIC) to the ReadOutBuffers (ROB) which store data from the different detectors. A supervisor connection is necessary for the event frequency reduction, to initiate the ROB readout and to signal the event decision. The following switch network connects the supervisor with the processor farm to do a further reduction of the event rate, initiated by the supervisor.

The B-physics task at low-luminosity is initiated by a farm processor and the result of the feature extraction is send directly via the switch network to this farm processor.

2 Feature Extraction

2.1 Algorithm

Two tasks mentioned above are described here in more detail.

The **first task**, the **RoI-guided TRT Feature Extraction (FEX)**, is needed as one step for the overall reduction of the event rate from up to 100kHz after level-1 to about 1kHz after level-2. For this task full precision data from most of the detectors are available, but only regions of the detector identified by the level-1 as containing trigger objects are used. The TRT FEX is based on a pattern recognition algorithm searching for high p_t tracks. bins in the data stream. A processor farm, which was not part of the demo A program, combines information of several subdetectors and RoIs to generate an event decision.

The **second task**, the **B-physics**, occurs with lower frequency. However, due to the special character of B-physics events, the locality information from level-1, the RoI, is insufficient for selecting the desired B-physics processes.

Therefore for B-physics studies in the low and intermediate luminosity runs the whole detector volume has to be investigated. A similar pattern recognition algorithm as for the first task is used to investigate the TRT information. The reconstructed features are passed via the switch network to a farm processor to do the global decision.

Following, some steps of the TRT FEX algorithm used in both tasks of the level-2 trigger are explained. Calorimeter studies have already been done years ago with heavier algorithms than requested now [BBea95]. Muon studies could not be done because we had no reference algorithms and data available.

Preprocessing The preprocessing task for the TRT is the transformation of raw data obtained from the ROBs to a new data format that is easier to handle for the following steps of the level-2 trigger. This could profitably be done in an FPGA-based coprocessor[GHea97] next to the ROBs, which then reduces the required bandwidth between the ROBs and the 2nd level trigger [BNS97]. The execution time for the preprocessing is not negligible in comparison with the feature extraction, whereas in an FPGA architecture the preprocessing can easily be done in a pipelined fashion without introducing much latency.

Initial Track finding Both the full scan for the B-physics and the RoI scan of the TRT are based on the histogramming method in the Hough space. The bins in the Hough space are predefined and stored in Look-Up-Tables (LUT). The histogramming on Enable++ is done in parallel and pipelined, so that within every clock cycle up to 864 histogram counter can be increased according to the incoming data. When all data are processed the counters are compared with a threshold to identify the found bins. These bins are send out as the result. The 864 bins are not sufficient for the B-physics due to the complete scan down to 0.5 GeV. Consequently, to increase the number of available bins the active straw information has to be passed several times. On the other hand, locality information from the ROBs is used to group the active straws to the corresponding Hough space. This allows a very fast operation with up to 80 000 bins.

The principle of working with LUTs allows to calculate any trajectory in the Hough space. This is useful both for the barrel, where the trajectories of physical interest are only approximately straight lines, and the end cap due to the inhomogeneous magnetic field.

Elimination of double tracks One physical track can be reconstructed several times due to the discrete straw information and the several pattern recognition. To facilitate the elimination of so-called double tracks, the LUT is ordered in a way that neighboring bins in the LUT represent spatially adjacent patterns instead of neighbored bins in the Hough space. This allow a more efficient and faster elimination of double tracks, because only local communication is required. In the current implementation this step reduces the number of reconstructed double tracks by a factor of 10.

Analysis of these and further algorithm steps showed that the initial track finding part took the biggest fraction of the overall computing time.

A **detailed description of the following algorithms steps and the analysis of the quality of the reconstruction algorithm** can be found in [KKL+98].

2.2 Hardware realization

Based on analysis and the software implementation, the initial track finding was ported to the Enable++ system. For this purpose this algorithm part was rewritten in PPC and compiled with the PPC compiler [Zoz97]. PPC is a C like high level language which was developed for Enable++. As a result it turned out that the compiled processes for the system used too much resources and was two times slower than an optimized process. So, to get a fast and efficient process for the FPGAs on Enable++ VHDL [1] was used.

The initial track finding algorithm is distributed across sixteen single FPGA processes which are almost similar. These processes are designed for the FPGA computing core on the computing array unit, a component of the Enable++ system. Each FPGA process consists of a RAM access port, histogrammer counters, a result compression unit and the main controller. The RAM has a size of 128k$\times$54 bit and holds the LUT with the predefined tracks. Therefore a change in the geometry effects only the LUT which can be easily changed by software. Only a change of the thresholds leads to new processes.

The process itself has a pipelined structure so that each incoming data word can be processed by the histogrammer within one clock cycle. Thus the histogramming is done during the data transmission. Furthermore, the search for the tracks is done in parallel, so each clock cycle detector data are compared with the 864 predefined tracks and the histogrammer are incremented accordingly. Nowadays, the processes runs with a clock rate of 20 MHz, but with state of the art FPGA devices clock rates up to 60 MHz can be achieved.

With this present VHDL processes we use only half of the gates in each FPGA device. So there is plenty of resources left to implement other algorithm parts like the elimination of the double tracks in the FPGAs. This will increase the speedup even more. Also some tasks of the following processor farm can be implemented in the FPGA processor.

Beside the FEX processes for the actual initial track finding algorithm there are other processes necessary to implement a data path from the data source to the destination. Some processes communicate with the SLinks and they are responsible for the data I/O while others pass the data via the backplane to the computation unit or create control signals for the algorithm.

2.3 Measurements

The main issues of the hardware measurements are:

- benchmarking of the Enable++ execution speed
- proof of the reliability of Enable++ results

To provide a testbed for those measurements, simulated TRT data were taken both for the RoI-guided scan and the full scan. Identical data sets were given to Enable++ and a reference C program, to ensure comparable conditions. The C program uses the same algorithm as Enable++ and runs on all standard processors. This allows a test of the hardware, showing that identical results are produced.

[1] VHDL – standardized hardware description language

The data set contains on average 560 active straws per event, which corresponds to 30 % occupancy in the RoI-guided case and 1 % occupancy for the full scan.

2.3.1 Software measurements

The reference software is a pattern recognition program written in C based on the same LUT-information as Enable++. This software can be adjusted only to execute the initial track finding, which then produced identical results as Enable++. A more detailed description of this program can be found in [KKL+98]. The benchmarking of this program in comparison to Enable++ is shown in table 1.

2.3.2 Hardware measurements

The measurements with the Enable++ were done to measure the data throughput rate and the latency, and to verify the identical behaviour of the software implementation and the Enable++ implementation. In the following chapter the test architecture is described followed by the measurements which were made with this architecture. Finally the integration tests with other ATLAS groups are described briefly.

2.3.3 System architecture

For the algorithm design verification as well as for measuring the date rate and the latency the same test architecture was used. This test architecture is shown in figure 2. The data source was either a Slate[Sla94] with a Slate-to-SLink daughter board and a fiber channel SLink sender onto it, or a MicroEnable board with a fiber channel SLink attached to it. Both data sources were able to send single data events or packets of several events as a continuous data stream or only once. The figure shows only the MicroEnable-SLink module solution which is described more detailed later in this chapter.

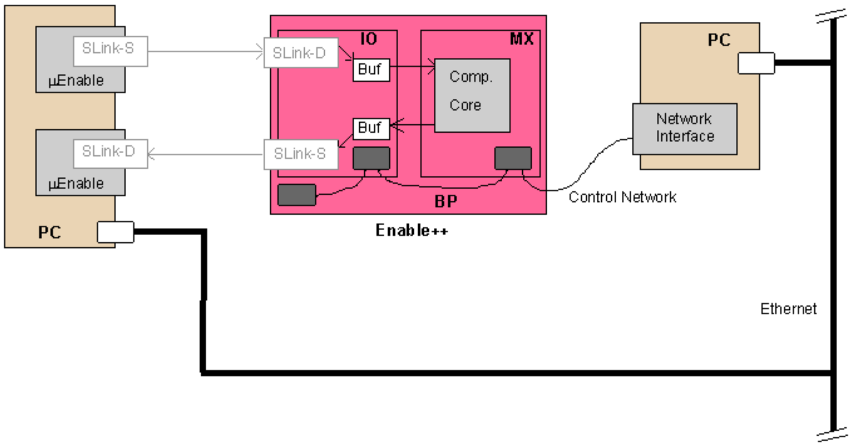

Figure 2: The Enable++ test architecture.

For the tests and measurements data were received with the Enable++ system and processed in the computing core. The result readout was done either via the control network of the system or it was send back via a second SLink connection and compared with a PC.

The other PC, connected to the control network acts as host for the Enable++ system. This host is responsible for booting the system, setting up the parameters and initializing the LUTs.

MicroEnable as slate As an alternative to the slate we use a MicroEnable [LNK+98] with an SLink module. MicroEnable is a FPGA-based coprocessor board for PCs equipped with a state of the art FPGA, up to 2MByte RAM, a PCI host interface and another free programmable interface. The board is plugged into a PC and the SLink module is attached to it and connected to the programmable interface.

Together with a MicroSlate design data can be stored in the RAM on the card and send through the SLink connection. This can be done either as single data packets or as data packets which are send in a endless loop to achieve a continuous data stream. Also the gaps between the single events can be specified. The data rate is limited by the SLink specification and reaches the maximum data rate of ≈95MByte/s.

2.3.4 Data transmission

To find the limits of the data transmission some measurements with the SLinks and the Enable++ system were made. Therefore a process emulating the execution times of the algorithm was used to measure the maximum data throughput rates of this architecture. The process time for each event packet was fixed set to 1.2μs. The measurements itself were made with two different system clock frequencies. The results of this measurements are shown in figure 3 and indicate the maximum achievable data rates for this test setup. The decreasing data rates at lower event packages are an effect of the SLink protocol.

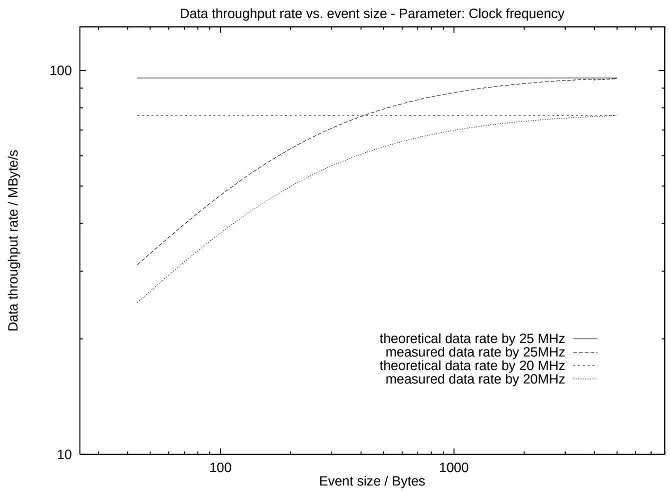

Figure 3: The Enable++ test architecture (extrapolated graph).

2.3.5 Algorithm results

The measurements mentioned in the previous chapter and the data throughput and latency measurements were all made with this system architecture described above. For the following measurements the Enable++ system clock frequency was set to 20 MHz.

The **verification** of the algorithm was made with a set of 250 events covering the whole detector. It showed that the **initial track finding algorithm implemented on Enable++ produces the same results as the software implementation** on some high end computers. The processing times for the algorithm on different platforms are given in table 1 below.

EXECUTIVE UNIT	PROCESSING TIME [2]
100 MHz HP Workstation	2,000 μs
400 MHz Alpha	400 μs
20 MHz Enable++ System	26 μs

Table 1: Processing times for one barrel RoI on different systems.

For the **data throughput** measurements we took a subset of the test data to get a continuously data stream from the data source. The average data event packet had $\approx$450 data words. With this data sets and the board settings we measured a data throughput rate of 71.9 MByte/s, close to the theoretical data rate of 76 MByte/s. Based on the data sets and this result an event rate of 38 kHz can be calculated. The estimated event rates for several event lengths can be seen in figure 4. This figure shows the event rate with ideal and with real transmission rates. Due to the SLink protocol the real event rate is smaller than the theoretical event rate.

This measurements indicate that the data transmission time, which is almost equal to the histogramming time, takes $\approx 22\mu$s for the average event packet. The time to prepare the result and pass it to the output needs only $\approx 3\mu$s. This time to achieve the result is for every data packet a fix overhead which slow down the system minimally.

Based on this measurements we can estimate the event rates for both the RoI scan and the full scan.

In the case of the *RoI scan* the algorithm has to search only for 240 different patterns in the RoI. Therefore the algorithm can either be executed 3 times in parallel, or only 30 % of the computing board is used. In order to fulfill the required event rate of 100 kHz the Enable++ system has to run the algorithm in parallel on the same computing board, but this was not implemented at the demo program.

The *full scan* fills 80 000 bins with the detector data information to find the tracks. This can be done with an extended system where the number of FPGAs processing the data are increased or by modifying the LUTs and passing the event data several times. When the event data are passed three times, which

[2] The given times for all measurements are only for the histogramming part of the entire track search algorithm. Only the Enable++ measurement includes data I/O.

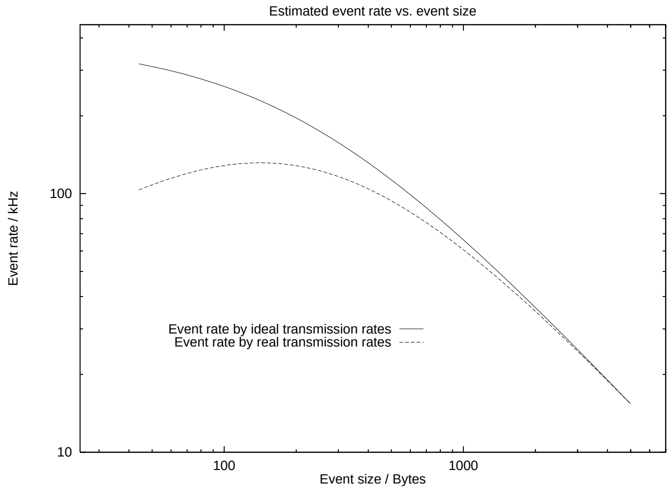

Figure 4: The Enable++ test architecture (extrapolated graph).

is enough to search for 2400 bins, the event rate will be reduced to ≈6 kHz for events with the size of 1k words. With the use of locality information it is sufficient to search in a reduced patterns space (2400 bins) per active straw without losing any quality.

The following table 2 shows the estimated event rates for a Enable++ system with one I/O unit and one computing unit. The use of three additional computing boards enables the parallel processing of barrel and end cap.

	Occupancy	Event Rate	Resource usage
RoI Scan	30 %	38 kHz	30 %
Full Scan	1 %	6 kHz	100 %

Table 2: Event rates (full barrel) respectively RoI rates.

For the **latency** measurements we took the same data set and the same system parameters as for the data throughput measurement. The latency was the time a event packet took from the SLink destination module on the Enable++ until it came out of the algorithm on the computation unit.

For this data path we got a latency of $\approx 2.3\mu$s. Additional to this measured time the transmission time of the event packet and the algorithm overhead of $\approx 3\mu$s has to be added to that time. So for a event with ≈450 data words the latency through the Enable++ is only $\approx 28.3\mu$s. The biggest part of this latency is the time for transmitting the event data.

3 Region of Interest Collection

The collection of all RoI fragments belonging to a given event is an essential task for the RoI-guided and full scan feature extraction algorithms. The task to collect, synchronize and transmit a complete RoI, consisting of several RoI fragments, to the FEX is performed by the Regions of Interest Collector (RoIC). Each RoIC receives a stream of concentrated, but unarranged RoI fragments from a number of ROBs. On real time conditions with 100 kHz event rate the RoIC arranges a pre-concentrated stream of RoI fragments to complete RoIs. The RoIC real time data rearrangement is guided by information which is extracted from the data header of each RoI fragment.

3.1 RoIC Algorithm

In the L2-scheme of ATLAS, the RoI-corresponding data is spread over a certain number of individual Readout Buffers (ROBs). These buffers are requested asynchronously by a supervisor to push each data set to the FEX, that corresponds to a given RoI, called RoI fragment. But due to a random distribution of RoIs over some neighbouring ROBs it is not a trivial task to transmit at 100 kHz event rate a full RoI, consisting of several RoI fragments, completely to the FEX. This task is performed by the Regions of Interest Collector (RoIC) [Kam98].

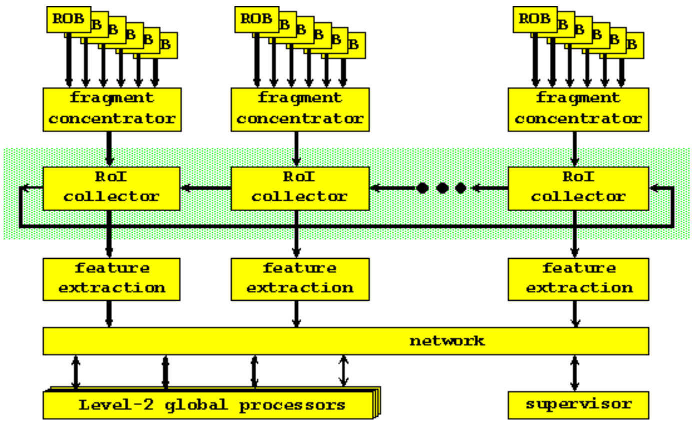

Figure 5: Architecture using RoI collector (RoIC)

The RoIC system takes advantage from the fact, that the ROBs can be grouped into crates using their geographical location within the detector. An optimized layout can guarantee that almost all possible RoIs can be assembled from ROBs in neighbouring crates. Considering the performance of the available components (links, memory speed etc) a demonstrator system with 4 Slink inputs per RoIC was chosen. Three of these input being fed by the concentrated output of three ROB crates and one by the preceding RoIC system thus closing the loop (see figure 5).

Each RoIC receives a stream of pre-concentrated, but unarranged RoI fragments. The average data rates on the input and the output of the RoIC are identical, the RoIC works without any data reduction. The RoIC real time data rearrangement is guided by information which is extracted from the data header of each RoI fragment. It is also possible to supply the RoI-ID through the control and monitoring path. Three information are essential:

1. EVENT-ID
2. FRAGMENT-ID (ROB-ID)
3. RoI-ID (fragments to combine)

Based on these information the RoIC can build a complete RoI from a various number of RoI fragments with different individual lengths.

3.2 RoIC Implementation

Main functional part of the RoIC is an intelligent memory, consisting of a huge dual port memory block, controlled by FPGA based logic. The choice for Field Programmable Gate Arrays (FPGAs) as controller hardware results from the real-time requirements combined with high flexibility. FPGAs contain configurable logic blocks (CLBs) with programmable interconnections. Both, logic and connections, are programmable by software, giving high flexibility for adaption of different tasks to the same hardware. Given algorithms can be expressed in VHDL [IEE87] and implemented directly in hardware and thus, run with extremely high speed.

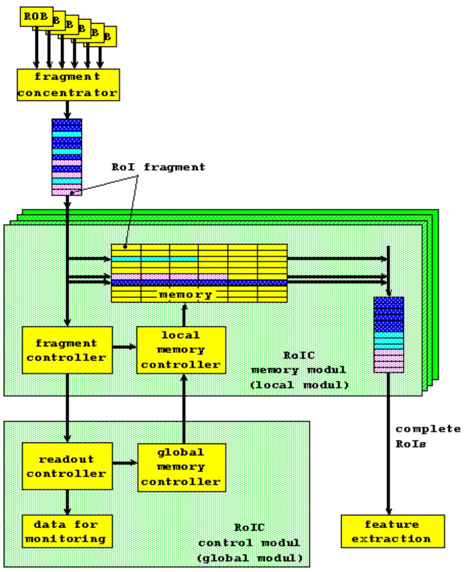

Figure 6: Functional diagram of RoI collector (RoIC)

The RoIC contains two individual modules, a local and a global module, each having two FPGAs (see figure 6). The local module is responsible for receiving the RoI fragments and storing them into the memory. A certain number of such local modules can be controlled by one global module, that watches all received RoI fragments and initializes the readout of complete RoIs. The fragments are assigned to pages according to the fragment(ROB)-ID in such an order used for output. Each event is stored in a different memory space.

The currently evaluated demonstrator of the RoIC accepts individual RoI fragments with any size below 4 kBytes. A RoI is built from up to 24 RoI fragments. Each event contains up to eight RoIs and up to 64 individual events are managed simultaneously by the RoIC.

All this functionality is implemented directly in hardware and thus, runs with high speed. Processing of the RoI fragments occurs "on the fly", only little latency is introduced from receiving the last fragment of a RoI until the header information is analyzed and the output of the complete RoI can start. An additional latency results from buffering the incoming data and grows linear with increasing fragment size (about 20 μs per kByte).

The RoIC can also handle very small RoI fragments (0 to 80 Byte), without problems. The only restriction is, that data cannot be accepted at the maximum bandwidth of 104 MByte/s (resulting in a theoretical fragment rate higher than 1 MHz). The reachable bandwidth slows down. In this case, for fragments less than 60 Byte, the latency may increase up to a maximum, the higher the fragment rate is. A flow control signal (xoff) signalizes if the RoIC is busy and cannot accept new fragments.

Fragments with sizes below 12 Byte are suppressed because they don't contain the full address information. In principle, the RoIC can handle RoI fragments with smaller size, but fragments with incomplete data-header (less than three data-words) will produce an error message. The upper limit of the fragment size is only restricted by the size of the memory. By defining a maximum fragment size, the memory can be partitioned for handling a certain number of fragments. The total number of RoI fragments inside the memory then can be split into the maximum number of ROBs per RoIC times the number of events, that can be handled simultaneously.

Due to the spacious ROB system, a final RoIC requires a high modularity, distributed over different crates. Identical RoIC modules can be used in the L2-trigger of ATLAS, to connect all ROBs with a small number of FEX-interfaces. In this case, the output from the right neighbour RoIC is multiplexed to the input data stream. There is also a demultiplexer necessary, to push only complete RoIs to FEX and incomplete RoIs to the left neighbour RoIC.

A prototype of the RoIC is evaluated in the Demonstrator Programme 1997 [DEK+97].

3.3 RoIC Measurements

The data for the RoIC demonstrator were injected by Slate (**S**econd **L**evel **A**rchitecture **T**est **E**nvironment) or alternative MicroSlate with FCS/S-link interface Link Source Card (LSC) using Fibre Channel with optical cables as physical layer. The RoIC synchronizes, sorts and transmits the data sets to the Enable++ system for further processing or analyzing. The RoIC resides on an I/O board of the Enable++ system (see figure 7).

Furthermore there is a bidirectional monitor path to control the RoIC. Information about all incoming and outgoing fragments are there available as well as the RoIC status and error messages. It is also possible to inject readout requests or information by the host system (supervisor) to the RoIC via monitor path.

The RoIC and the Enable++ system are configured, controlled and monitored by a common controlling network based on local processor units. The data source and the controlling network are managed alternatively by a Sparc-Unix system in Jena or a PC running WindowsNT software.

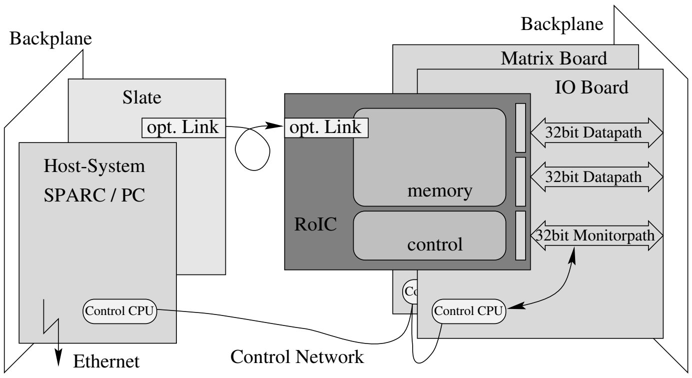

Figure 7: Demonstrator Setup

The data sets (events and fragments), sent by the source in an infinite loop to the RoIC, are emulating up to 24 different senders (ROBs and adjacent RoIC). Due to the limited memory of the Slate the data sets had a size up to 256 kByte containing up to 64 events. Measurements with varying assembled events, RoIs and fragments were made for evaluating the functionality of RoIC.

The data sets for the following diagrams were identified and measured with a hardware monitor. Each event contains 1 RoI composed of 6 fragments already in sorted order for a simpler understanding. This is justifiable since the sorting algorithm takes no extra time.

3.3.1 RoIC Throughput and Packet Frequencies

Within the demonstrator program the RoIC is used in a push architecture. The data are pushed by the source into the RoIC and the RoIC is pushing the data to the Enable++ system. The communication model is similar to so called "Pairs" communication which is defined in abstract basic communication benchmarks for the Second Level Trigger [BCDH95]. The interfaces of the RoIC are equivalent to the "user buffers".

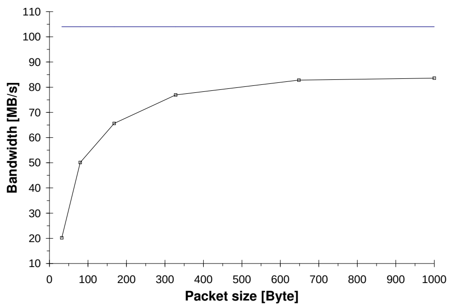

Figure 8: Slate bandwidth

The Slate with SLink interfaces sends at a theoretical bandwidth of 104

MByte/s. The maximum throughput is the Slate bandwidth measured to 83.6 MByte/s and for the MicroSlate to 91.6 MByte/s. The packet size is ≈60 Byte at half the maximum throughput (see figure 8).

The reason for the gap between measured and theoretical throughput is the insertion of idle times during the transmission of the packets and between the packets by the Slate-SLink system. This effect don‘t arise by RoIC. The RoIC xoff signal only slows down the bandwidth of Slate for fragment sizes below 80 Byte.

The frequency of sending data sets is again determined by the Slate bandwidth. For the RoIC the packet frequency covers the range from 100 kHz for 1 kByte packets to 740 kHz for packet sizes below 80 Byte. This upper limit is caused by the basic RoIC reaction time.

The packets at the output are grouped together for 1 RoI with idle time between different RoIs for fragment sizes larger than 80 Byte. The output fragment frequency shows the situation in such a burst whereas the output packet frequency shows the reciprocal mean packet period (see figure 9).

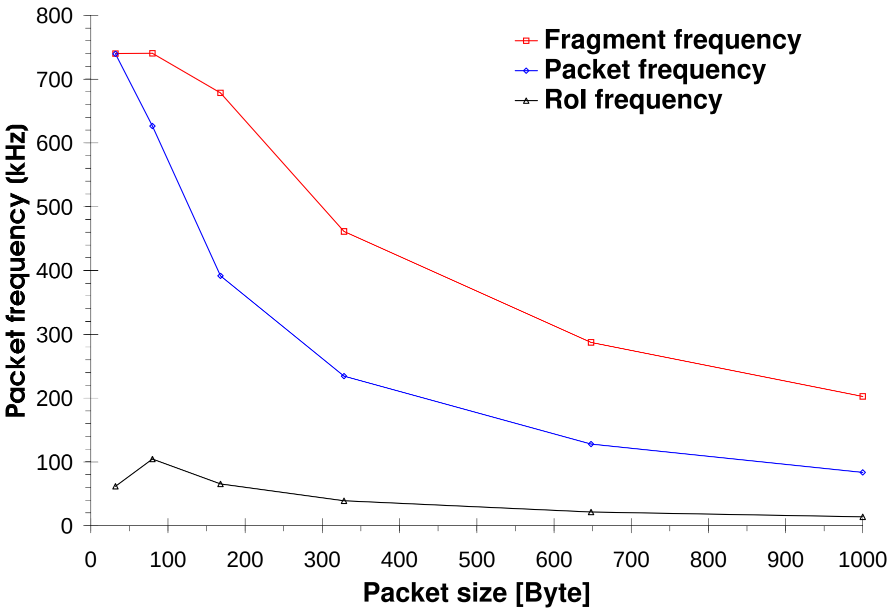

Figure 9: Packet frequencies

The distance between the curves is an expression for the RoIC utilization. If the RoIC could run with full load, the output frequency for 1 kByte fragments is about 200 kHz. The input bandwidth in the demonstrator setup was the bottleneck enabling a minimal frequency of 83.6 kHz for 1 kByte fragments.

The RoI frequency illustrates how the RoI frequency slows down for 1 RoI with 6 fragments and RoI sizes up to 6 kByte. This slow down is only caused by the data source bandwidth and goes below 14 kHz for a 6 kByte RoI except for the smallest fragment size.

3.3.2 RoIC Latency

The lowest curve in figure 10 describing the RoIC reaction time of 2.3 μs from the recognition of the last to the output of the first fragment of a RoI.

If the transmission time of a fragment goes below this basic RoIC reaction time, the latency of the RoIC is increasing dramatically. This situation is to observe below 80 Byte fragment size and frequencies of $\approx$740 kHz resulting in latencies of more than 75 μs.

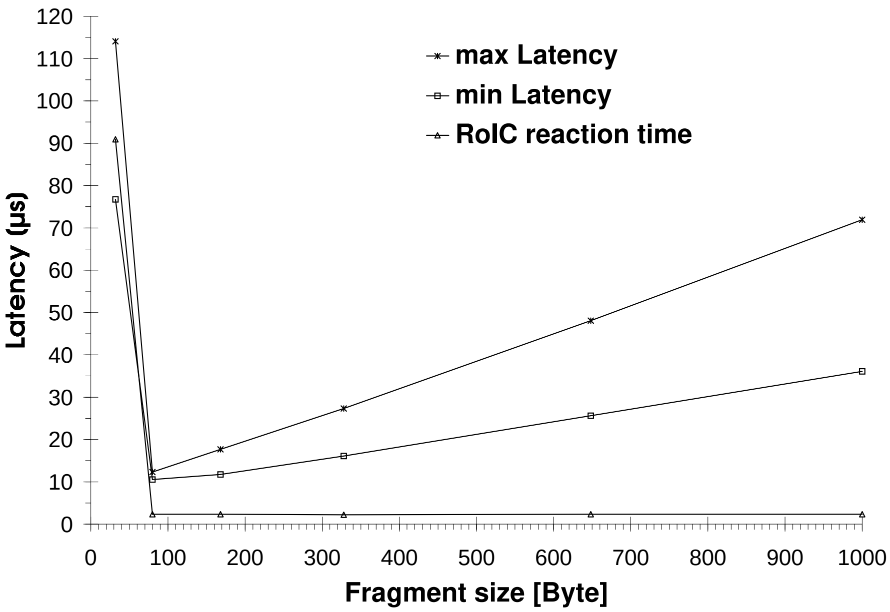

Figure 10: Basic and Algorithm Latency

The latency is a very ambiguous term. Now we define the latency as the time from the first word of a fragment on the input interface to the first word of the same fragment on the output interface.

The transmission time is the determining latency for fragment sizes larger 80 Byte. This is mainly the time to buffer the data. For larger fragment sizes all other effects are negligible and the latency is increasing linear by the fragment size. In figure 10 the both upper curves show the difference between the latency of the first and last fragment of a RoI. This difference represents an algorithm latency.

3.3.3 Algorithm Latency

Additional to the basic latency increasing linear by the fragment size there is an algorithm latency depending on how many fragments contains 1 RoI. Only when all fragments of a RoI arrived, the RoIC is allowed to push the first out. The first arriving fragment has the largest latency. For later fragments of a RoI this latency is decreasing (see figure 11).

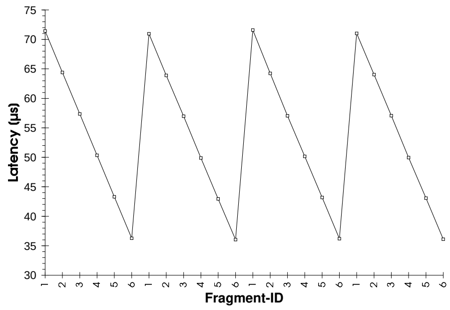

Figure 11: Latency of 1kByte fragments (6 per RoI)

The measurements are performed by complete events, that is minimal algorithm latency. If this condition isn't true, the algorithm introduces an extended latency.

For illustration of the extended algorithm latency depending by the order of events we assume that 250 fragments of 50 different events are received except one of every event. Now every last fragment of 50 different events arrive one after another. The completed RoIs of average 6 fragments are to submit with every arriving fragment. There is building up a traffic jam, which results again in a huge increasing algorithm latency.

3.4 RoIC Results

The RoIC proved the full functionality including monitoring. It is possible to manage up to 8 RoIs and 64 events simultaneously with 0 to 4 kByte fragment size and up to 24 fragments per RoI. The RoIC demonstrator allows for synchronization a latency of at least 0.64 ms.

The input is a bottleneck. The packet size at half of the throughput is about 60 Byte. The maximal fragment frequency is 740 kHz. If the RoIC runs with full load, the output frequency for 1 kByte fragments is about 200 kHz.

The RoIC reaction time is $\approx$2.3 μs. For fragment sizes larger 80 Byte the latency is increasing with the transmission time. The algorithm latency depends on how many fragments contain 1 RoI. But the order of arriving fragments and events adds a extended algorithm latency.

4 Integration Tests

For the integration of the Enable++ system into the whole ATLAS trigger system some corporate tests have been made. The cooperation partners are the group in Jena/Germany and the group in Saclay/France.

The **Jena group** is responsible for the **RoI-Collector** which synchronize the data from several ROBs to achieve complete and continuous RoI packets.

The test was intended to check that both the RoIC and the Enable++ system can work together and that the RoIC collects the RoI fragments correctly. As a result of this test it came out that the RoIC works correct with a set of RoI fragments and that the collected RoIs were transfered properly to the input port of Enable++. A performance measurement was executed.

Both RoI-Collector and Enable++ used a common controlling network with compatible software.

The **Saclay group** works on the **processor farm** and the **switch network** which connects the farm to the ROBs and to the supervisor. The goals for the common measurements with the Saclay group were to attach the Enable++ system to the Saclay ATM network via a special node called COP, to adapt the Saclay software module to this COP and to run the complete Saclay framework including the COP and Enable++.

Data from the ROBs were emulated as data packages which were transfered via the switch network to the COP. These data were passed and processed in the Enable++ system and the result was send back via the COP and the network to the requested farm processor.

A **successful integration of both demonstrators at the protocol level was performed**. The software framework from Saclay was ported to the COP and some measurements were made. The complete demo C system, including supervisor emulation and monitoring was running stable up to 1kHz message rate with an increased latency from 600μs to 800μs.

5 Conclusions

The RoIC is easily able to operate at high event frequencies. Processing of the RoI fragments occurs “on the fly”, only $\approx$2.3 μs latency is introduced from RoIC. A latency resulting from buffering the data grows linear with 20 μs per kByte fragment size. The RoIC demonstrator allows for RoI collection and synchronization a latency of at least 0.64 ms. The trigger control system has full access for controlling and monitoring the RoIC.

With the TRT track finding algorithm a concrete example is given to assess the fields of Enable++ in the second level trigger. It is shown that Enable++ is able both to cope with high event frequencies and computing-intensive tasks. High event frequencies occur at the high luminosity runs, and computing-intensive tasks occur especially in the B-physics trigger at low luminosities. A further field of FPGAs is the pre-processing of the ROB data.

References

[BBea95] D. Belosloudtsev, P. Bertin, and R. Bock et al. Programmable active memories in real time tasks: implementing data-driven triggers for LHC experiments . In *Nuclear Instruments and Methods in Physics Research, A 356 (1995) 457-467* , 1995.

[BCDH95] R. Bock, F. Chantemargue, R. Dobinson, and R. Hauser. Benchmarking communication systems for trigger applications. Technical Report DAQ note 48, CERN, November 1995. `http://atlasinfo.cern.ch/Atlas/documentation/notes/DAQTRIG/notes.html`.

[BNS97] R. Bock, K.-H. Noffz, and M. Sessler. TRT Preprocessing – Algorithm, Implementations and Benchmarks .

Technical Report DAQ note 66, CERN , March 1997. http://atlasinfo.cern.ch/Atlas/documentation/notes/DAQTRIG/notes.html.

[DEK+97] V. Dörsing, W. Erhard, P. Kammel, A. Reinsch, and H.U. Zühlke. RoI Collector implementation for the Level-2 Trigger of ATLAS. In *10. IEEE Real Time Conference*, Beaune, September 1997.

[GHea97] Y. Gal, H. Hoegl, and A. Kugel et al. A FPGA-based Coprocessor proposed for Preprocessing in ATLAS. In *Computing in High Energy Physics (CHEP)*, April 1997.

[HKL+95] H. Högl, A. Kugel, J. Ludvig, R. Männer, K.H. Noffz, and R. Zoz. Enable++: A Second Generation FPGA Processor. In P. Athanas and K. L. Pocek, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 45–53, Los Alamitos, California, April 1995. IEEE Computer Society Press.

[IEE87] IEEE. The IEEE Standard VHDL Language Reference Manual . Technical report, Institute of Electrical and Electronics Engineers, 1987. Standard 1076.

[Kam98] Peter Kammel. *Rekonfigurierbare Einheit zur Datenreduktion für den Level-2 Trigger von ATLAS*. PhD thesis, Friedrich-Schiller-Universität Jena, 1998. to submit March 98.

[KKL+98] A. Kugel, K. Kornmesser, R. Lay, J. Ludvig, R. Männer, K.H. Noffz, S. Rühl, M. Sessler, H. Simmler, H. Singpiel, and M. Smizanska. Fullscan of the Barrel TRT for the ATLAS second-level Trigger. Technical Report future DAQ note, CERN, March 1998. http://atlasinfo.cern.ch/Atlas/documentation/notes/DAQTRIG/notes.html.

[LNK+98] R. Lay, K.H. Noffz, K. Kornmesser, A. Kugel, and R. Männer. The microEnable FPGA Coprocessor . In *IEEE Symposium on FPGAs for Custom Computing Machines* , Los Alamitos, California, April 1998. IEEE Computer Society Press. submitted.

[Sla94] ERT 2.28.02.0 SLATE2 Module , 1994. http://sunserv.kfki.hu/~mszkihp/lad/products.htm.

[Zoz97] Ralf Zoz. *Eine Hochsprachen Programmierumgebung für FPGA-Prozessoren*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 1997.