Operation and Performance of an ATM based Demonstrator for the Sequential Option of the ATLAS Trigger

M. Abolins[2], R. Blair[1], D. Calvet[3], J. Dawson[1], Y. Ermoline[2], R. Hubbard[3]
M. Huet[3], P. Le Dû, I. Mandjavidze[3], D. Owen[2], J. Schlereth[1], B. Thooris[3]

[1]Argonne National Laboratory, Argonne, IL 60439-4812, USA

[2]Michigan State University, East Lansing, MI 48824-1321, USA

[3]CEA Saclay DAPNIA, 91191 Gif-sur-Yvette Cedex, France

Abstract

The current thrust within the ATLAS Second Level Trigger is the Demonstrator Program which seeks to facilitate the selection of an architecture for the Second Level Trigger of ATLAS with several combined hardware, emulation and simulation efforts. This paper describes the implementation of a small scale demonstrator of the sequential option for the ATLAS Second Level Trigger at Saclay. This demonstrator utilizes an eight port ATM switch connecting source modules, destination processors, and a supervisor unit. The source modules emulate a group of detector read-out buffers which would receive event data on a level 1 trigger. The hardware and software of the system and the performance of the demonstrator in various configurations and with different sets of input parameters are described. Scaling of the architecture of this demonstrator to the design of the large Second Level Trigger system for ATLAS is discussed.

I. INTRODUCTION

The ATLAS experiment at the CERN Large Hadron Collider is evaluating a number of possible schemes for its Trigger/DAQ system [1]. Their relative merits are being analyzed by means of spread sheet model, computer modeling, simulation and by building small scale prototypes and measuring their performance under controlled conditions. This "demonstrator" activity has the aims of demonstrating the validity of the designs proposed, measuring the performance of a real, albeit small scale, system and of understanding the underlying technologies. The resulting measurements are necessary inputs to modeling activities which strive to predict behavior of systems scaled to real size. This paper describes the implementation and performance of a system based on the sequential event selection principle described earlier [2].

II. OVERVIEW OF SEQUENTIAL ARCHITECTURE

In the sequential triggering scheme events passing the LVL1 trigger are subjected to a series of increasingly restrictive tests with failure resulting in rejection of the event. Data are transferred to the LVL2 processor only as they are needed. In this way relatively simple tests can serve to rapidly decrease the event rate with minimum traffic on the links. This scheme should be contrasted with the parallel selection approach where events are processed in parallel and all relevant data are immediately transferred.

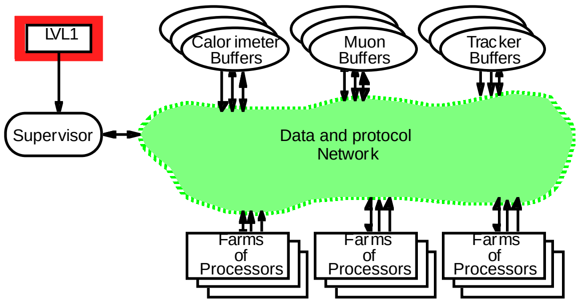

Figure 1: The Sequential Option in ATLAS

In the sequential option, shown schematically in Figure 1, the supervisor assigns each event to a single processor. The processor "pulls" event fragments from the data sources as it needs them in applying its sequence of tests to the event. This design permits a reduction in the data volume transferred at the expense of introducing additional complexity in the event handling logic. In this scheme it is possible to apply selection criteria to only a single detector, for example the calorimeter, before considering data from others. This should be contrasted with a "push" scheme where all data that may be needed are sent by the sources without intervention by the destination processors.

This architecture relies on a single communication network to link all nodes of the system. The network carries data as well as control information providing a flexible all-to-all connection which has to be rather large and complex to deal with the variety of traffic. In this demonstrator project we have chosen ATM as our switching standard. The appropriateness of this technology for this purpose is supported by reports from other groups [3], [4].

III. THE DEMONSTRATOR PROJECT

A. Description of Hardware

The physical configuration of the demonstrator project described in this paper is shown in Figure 2. The implementation consists of 4 sources, 4 destinations, a supervisor and a monitor connected to the ports of an ATM switch. For the switch we have chosen the FORE

ASX 1000 which is equipped with twelve 155 Mb/s ports along with six ports of 25 MB/s. The switch has an aggregate bandwidth of 10 Gb/s and can accommodate many different kinds of interfaces. For sources we use 100 Mhz PowerPC VME single board computers (CES RIO 2) running LynxOS and for destination processors 200 Mhz PentiumPro PC's (DELL) running WindowsNT.

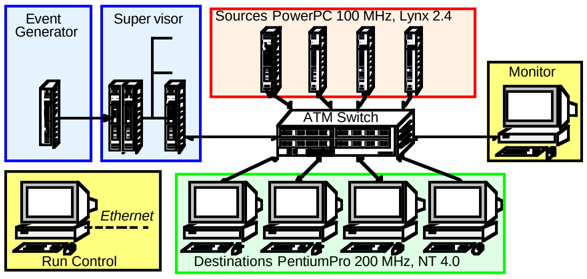
Event Generator
Super visor
Sources PowerPC 100 MHz, Lynx 2.4
Monitor
ATM Switch
Ethernet
Run Control
Destinations PentiumPro 200 MHz, NT 4.0

Figure 2: Demonstrator system configuration

The Supervisor is based on a RIO 2 along with special hardware and is described below. A primary goal of our implementation was to develop flexible and portable code to permit any host platform to serve as source, destination, supervisor and even monitor. For reasons of performance and space constraints, we chose LynxOS VME processors for source and supervisor modules. The choice of PC's for destination processors and monitor is dictated by cost and convenience.

Our run control system was based on Ethernet with a graphical user interface allowing starting and stopping runs from Workstations. The system configuration and the run input parameters were stored in a parameter file accessible to all nodes at the beginning of the run. At the end of the run nodes could print log files to screen or disk for system debugging and run analysis.

B. *Software*

Efforts were made to produce software that would be independent of specific platforms, operating systems and even networking technology. We chose a layered structure divided into three major blocks with more than 20,000 lines of C code. We describe them below.

1) *Core module*

This code is operation system and platform independent with large parts of the code common to all types of nodes. This permits rapid creation of new nodes through re-use of existing code.

2) *Platform/OS dependent modules*

All code of this type is placed in a module that also includes thread creation and manipulation routines, routines for synchronization and mutual exclusion and routines for timing. We have implemented this part for both LynxOS and WindowsNT.

C. *Network technology specific module*

This module contains functions to setup and tear down connections and to send and receive messages across the network. We have implemented this layer using an optimized ATM library [5]. These same functions can be implemented for other ATM network interface cards as well as for other switching technologies.

For data transport we adopted the internet standard network byte order (Big Endian) and we have defined functions to convert internal data structures to and from their network representations.

IV. Node Operation and Performance

A. *Source Operation*

A source module, shown in Figure 3, consists of a variable number of Read-out Buffers (RoB's) [6] on a bus (e.g. PCI bus) connecting them to a RoB to Switch Interface (RSI) network. The RoB's receive the event data from the detector at the level 1 trigger rate and buffer them during the event selection process. The RSI responds to requests arriving via the ATM network.

When the RSI receives a request for data, it transmits

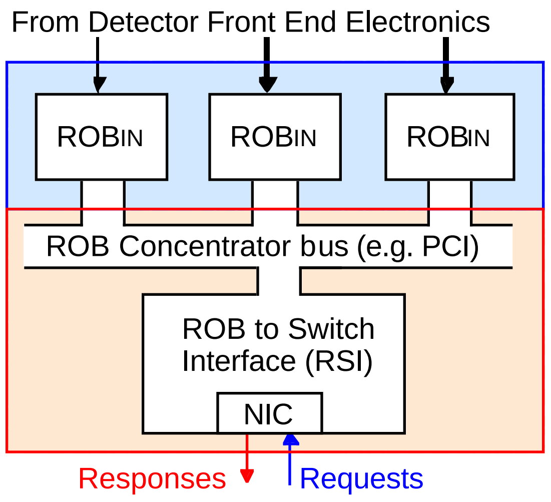
From Detector Front End Electronics
ROBIN
ROBIN
ROBIN
ROB Concentrator bus (e.g. PCI)
ROB to Switch Interface (RSI)
NIC
Responses
Requests

Figure 3: Schematic view of a source module.

this request to all affected RoB's. When all of these RoB's have replied with their data, the RSI formats and transmits the data to the requesting processor. A time-out mechanism has been implemented to prevent hang-ups due to malfunctioning RoB's. The RSI also has the function of distributing to RoB's the decisions received from the supervisor.

The RoB design choice has not been made and therefore we have not implemented hardware RoB's in the data sources. We have chosen instead to emulate the RoB functions in the RSI processor as a single thread application. We note that this approach is sufficient to test the demonstrator system but, it does not confront possible problems in the RoB/RSI interaction.

1) Performance

A good measure of the source performance is the maximum number of data requests, F_{src}, that the source can service as a function of the size of the returned packet size. This is shown in Figure 4 for 1, 2 or 4 RoB's connected to the RSI and with no pre-processing by the source.

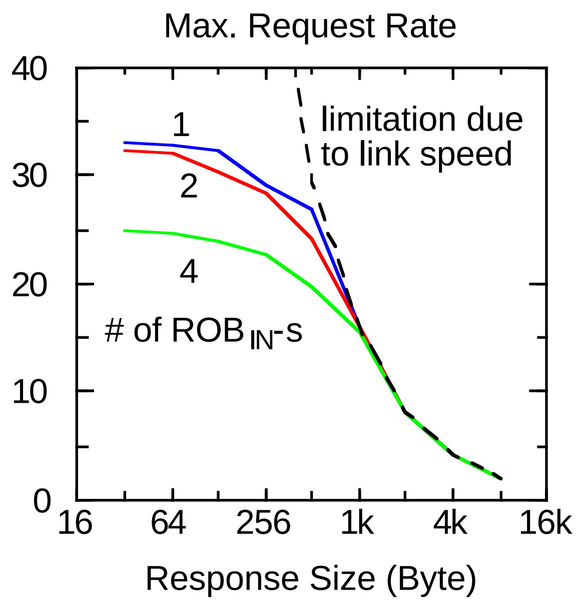

Figure 4: F_{src} vs. fragment size.

Saturation of the output link is reached for packets larger than about 1 kB where the transmission rate starts to exceed the link speed of 155 Mb/s.

On a 100 Mhz PowerPC, the source code execution time is given by:

$$T_{src} = 30\ \mu s + 2.5\ \mu s * n_{RoB} \qquad (1)$$

where n_{RoB} is the number of RoB's. Since there are no physical RoB's connected to the source, these numbers must be considered as lower limits. Modeling of the RoB/RSI interface on a spreadsheet suggests a more realistic relation to be:

$$T_{src} = 50\ \mu s + 20\ \mu s * n_{RoB} \qquad (2)$$

If pre-processing by the source is included, the maximum sustainable data rate is further reduced.

B. Destination module

1) Operation

The destination processor takes charge of processing the events selected by the supervisor. At each sequential step of processing it issues requests to the sources for the data it needs to execute its algorithms. Each processor handles several events concurrently, processing one event while waiting for data from another.

We have used a multi-thread task to implement the processor function. An Rx thread on the ATM network side handles incoming packets responding to interrupts while a Tx thread transmits messages. Other processing threads, one per event, are charged with running the selection algorithms. A new event is passed to a processing thread which posts a request to the Tx thread, becoming idle, until data are delivered. If only a small number of sources of data are involved, individual messages are sent to each. If on the other hand a larger group of sources is involved such as would be the case for missing energy calculation or whole event building, a single message is sent by the Tx thread to the whole group of sources on a multi-cast channel. After all the requested data have been collected by the Rx thread, they are posted to the processing thread that executes the next step in the selection algorithm. This process continues until a decision to keep or reject the event is reached. To avoid system hang-ups, a time-out thread periodically scans the list of pending requests posted to sources. If any source has not replied within a specified interval, the processing thread is informed that its request has failed and the event is discarded. In the demonstrator we do not use actual selection code but, emulate its operation by dummy code of appropriate duration.

The software running in the destination modules has been designed to execute equally well on single processor platforms as well as on Symmetric Multi-Processor machines. Operation on a cluster of processors could be easily implemented as well since the processing threads are independent of the Tx, Rx and time-out threads.

An indication of destination performance is given in, Figure 5 where we plot the time necessary to handle one event, T_{dst}, versus the size of the fragment sent by each source. We use a single step algorithm with no time devoted to processing the event.

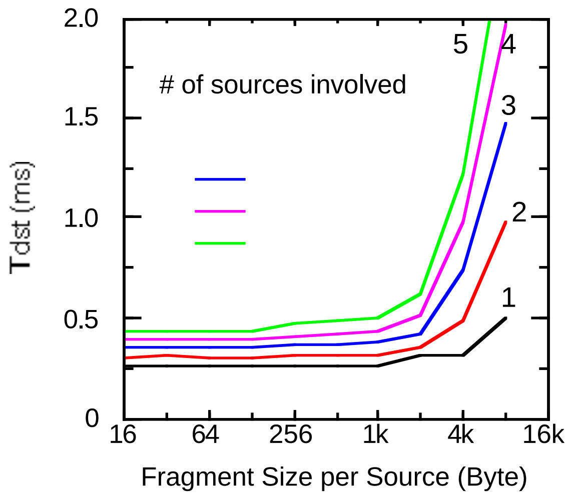

Figure 5: T_{dst} vs. event fragment size.

On a 200 Mhz PentiumPro, the code executes in:

$$T_{dst} = 215\ \mu s + 45\ \mu s * n_{req} \qquad (3)$$

where n_{req} is the number of requests.

Saturation in the graph indicates that data transfer time dominates for blocks larger than about ~2 kB. The maximum event rate sustainable by a single destination is $1/T_{dst}$. As an example, a single destination can accept short fragments spread over 4 sources at a rate of ~2.5 kHz.

When processing is turned on, the time per event increases proportionately and the maximum event rate is given by:

$$F_{dst} = 1/(T_{dst} + T_{algo}) \tag{4}$$

where T_{algo} is the time necessary to execute the algorithm. This applies in the regime where processing time dominates the data transfer time. When data transfer dominates the maximum event rate becomes:

$$F_{dst} = 1/T_{transfer} \tag{5}$$

The selection algorithm first re-formats the data and then executes the selection code. Our measurements for the speed of copying the data are ~36MB/s on our PC's and ~30MB/s on our VME hosts.

In order to avoid wasting processing resources or network bandwidth care should be taken to match the CPU power of the destination, running the appropriate algorithms, to the speed of the link. If we assume that a typical algorithm processes 4 kB of data from 4 sources in ~100 µs, the overhead from equation (3) is ~400 µs and the minimum link speed necessary is ~8 MB/s, a number well within the capabilities of a single 155 Mb/s link.

C. *Supervisor*

A detailed description of the supervisor can be found in the literature [7]. It consists of an arbiter/router plus one or more processors (RoI Cpu's) connected to the ATM switch. It is designed to be simply scalable by adding additional processors but, for this test we used a minimum configuration of 1 RoI Cpu to demonstrate its operation with the other elements in the system.

In normal operation the LVL1 trigger sends summary information (region of interest or RoI data) to the arbiter/router where it is gated sequentially to an RoI Cpu. The RoI Cpu assigns the event to an available destination processor, transfers the RoI data and then listens for a trigger decision which it transmits to the affected sources. All transactions take place via the ATM network. To avoid overloading any processors, we implemented a credit based flow control mechanism as well as a time-out mechanism to detect and isolate faulty processors.

We implemented two modes of supervisor operation, a normal mode where events where externally generated and an emulation mode where the RoI Cpu generated events internally at the desired rate and the arbiter/router was bypassed. In this mode, the RoI Cpu could be a desktop PC or any other processor.

The supervisor task is implemented in a single thread with polling used to check for input from the router and the ATM network. In normal mode a supervisor with a single RoI Cpu (100 Mhz PowerPC) achieved an event throughput of ~8 kHz. In emulation mode, however, with the same Cpu this rate went to ~16 kHz and to ~22 kHz when the PowerPC was exchanged fro a 200 Mhz PentiumPro. Our studies indicate that a target rate of ~100 kHz can be reached using several, faster RoI Cpu's as well as implementing a number of optimizations.

D. *Monitor*

The function of the monitor was to make regular checks of the nodes to see that they were working and to take corrective action if they were not. It was also responsible for periodic gathering of relevant statistical data and presenting them to a display program. The monitor itself was designed without a graphical user interface to permit implementation on various platforms. In our application a separate visualization program communicating with the monitor via shared memory provided the user interface. The monitor could also be used without graphical output to display results at regular intervals on the screen and to print statistical summaries at ends of runs. Provision was made for user input to the monitor to dynamically change run parameters. All monitoring and statistics gathering functions were implemented via ATM.

The performance of this node was not critical to any of the measurements taken.

V. SYSTEM PERFORMANCE

While the performance of the individual components, discussed in the previous section, is interesting in its own right, we now turn to an analysis of the full system as shown in Figure 2. An important measurement of system performance is the event latency, T_{dec}, defined as the time interval from event allocation by the supervisor until the return of a decision to the supervisor by the destination processor. In this test a processor fetched 1 kB of data from each of 4 sources and made a selection in a single step of 100 µs. The event rate was fixed at 3.6 kHz corresponding to the utilization of ~20% of link bandwidth and ~50% of destination processor power. Our results show that the average trigger latency is 1.2 ms and that 99% of decisions arrive in less than 1.4 ms. These numbers may be understood by noting that the latency between the supervisor and the destination processor is ~200 µs, the latency introduced by the destination processor is ~500 µs consisting of 4 sent requests and a 100 µs algorithm, the transmission of requests to sources takes ~60 µs and the transfer of 4 kB of data over a 155 Mb/s link takes ~250 µs which add up to 1.1 ms, the minimum observed value. Additional delays are contributed by queuing, system scheduling etc.

Measurement of T_{dec} do not include the time needed to communicate the decision from the supervisor to the sources and finally to the RoB's.

Event latency sets the depth of event buffering necessary in the RoB's. Assuming a target LVL1 rate of 100 kHz, an average event size of 1 MB and 1,000 RoB's implies an average data rate of 100 MB/s into a RoB. Using 10 MB of memory per RoB provides 100 ms of buffering which far exceeds our latency measurements.

The performance of the system as the event rate was increased is shown in Figure 6 which shows that the system operates safely at a rate of 1.5 kHz per destination

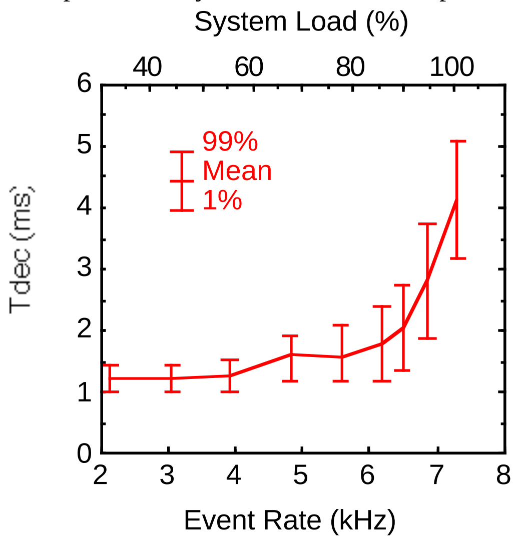

Figure 6: T_{dec} vs. event rate.

or a total event rate of 6 kHz for 4 destinations. The saturation point is ~1.9 kHz per destination which is imposed by the destination processors and can be understood from equation [4] which predicts in this case:

$$F_{dst} = 1 / (215 + 4*45 + 100)\ 10^{-6}) = 2\ \text{kHz} \qquad (6)$$

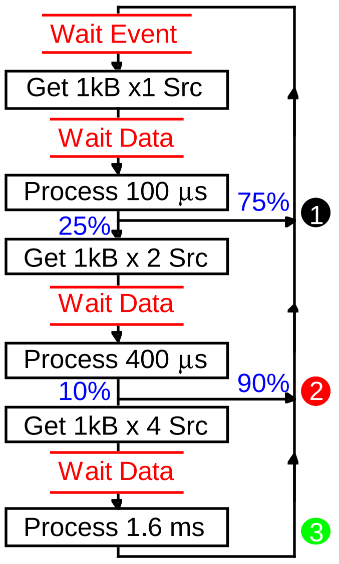

Figure 7: algorithm flow chart.

We have also performed tests using algorithms with several steps and different execution times and a variable number of sources. One such algorithm flow chart is shown in Figure 7.

A histogram of T_{dec} for each step is shown in Figure 8 with an event rate of 3.35 kHz corresponding to 50% system loading. Step 1 executes at a higher priority than subsequent steps so that new events are not delayed by events with longer processing times. This test demonstrates an implementation of sequential processing with simultaneous analysis of several events in destination processors. It also serves to demonstrate the capabilities of this network in handling heterogeneous traffic including, requests, event data, monitoring and run statistics. We use Unspecified Bit Rate (UBR) channels to transfer data for the first and second steps of the algorithm and Constant Bit Rate (CBR) channel for the last step which is full event building. Monitoring uses low priority CBR channels. This bandwidth allocation scheme was successful in avoiding network congestion and cell losses.

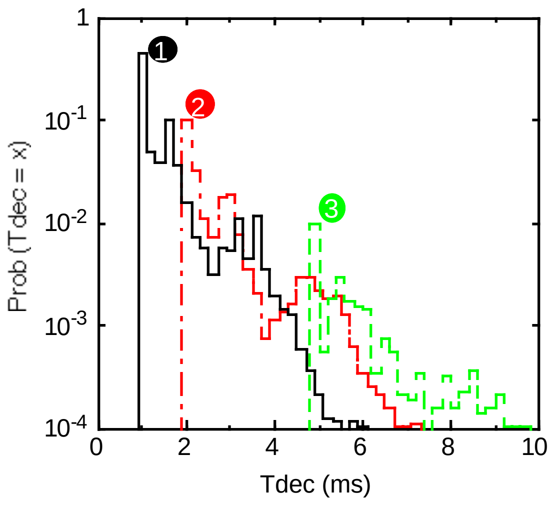

Figure 8: T_{dec} histogram per algorithmic step.

VI. System Scaling

An important issue is the scalability of the system and its components. The supervisor is expected to scale linearly with the number of RoI Cpu's. The performance requirements for source modules are set by their frequency of participation in events. This depends on the sub-detector and on the number and granularity of the RoB's. Early estimates indicate that some sources may participate in as many as 20% of all events. Although the implied 20 kHz request rate is compatible with our measurements, some improvement in source performance will probably be needed.

The number of destination processors can be scaled to cope with the rate of incoming events. In sequential selection the first step will be confirmation of the primary RoI (~ 1 per event) which involves the transfer of ~4 kB of data from 4 to 8 sources. Our model indicates that current processors running a 100 µs algorithm should be able to process events at ~1.6 kHz implying a farm of ~66 such processors for this first step. Further consideration suggests that a farm of a few hundred processors should be adequate for all our needs.

Switching networks with capacity of 10 Gb/s are available from a number of vendors at moderate cost e.g. ATM switches for ~$1.0k per 155 Mb/s port. We estimate that the needs for ATLAS can be met with a switch in the 40 - 80 Gb/s range. Switches in this performance range are readily available in the telecommunications market although currently their cost per port is much higher than for smaller switches. Furthermore, simulation and modeling studies indicate that the bandwidth allocation scheme used in the demonstrator system is adequate to avoid network congestion in a large system [8].

VII. Summary

We have demonstrated a sequential trigger option for the ATLAS level 2 trigger. We have implemented a "pull" mode of operation and shown the feasibility of sequential data transfer using partial and full event data. We have demonstrated concurrent processing of several events in a single processor and have shown the viability of a number of mechanisms for error detection and recovery. The feasibility of merging different kinds of traffic, including monitoring and gathering of run statistics, in the same ATM network has been demonstrated. We have derived parameters that permit system characterization in terms of simple formulas.

A number of points have not been addressed by this demonstrator including implementation of real algorithms in destination processors and the interaction of RoB's with RSI's.

The demonstrator results presented in this paper are an important step in validating the sequential triggering concept for the ATLAS trigger. The architectural principles proposed are realistic and the ATM switching technology is well suited for this application.

VIII. Acknowledgments

We thank D. Laugier and C. Rondot of CPPM, Marseilles for their help in run-control implementation and J. P. Dufey of CERN for his support.

IX. References

[1] ATLAS Level-2 Trigger Groups "Options for the ATLAS Level-2 Trigger", in Proceedings of the IEEE Conference on Computing in High Energy Physics, Lichtenberger Congress Center, Berlin, Germany, 7-11 April, 1997.

[2] J. Bystricky et al., "A Sequential Processing Strategy for the ATLAS Event Selection", in Proceedings of the Nuclear Science Symposium, Anaheim, California, 3-9 November 1996. Also in IEEE Trans. on Nuclear Science, **44**, No. 3, June 1997, pp. 342-347.

[3] M. Costa et al., "Results from an ATM-based Event Builder Demonstrator", IEEE trans. on Nuclear Science, **43**, No. 4, June 1996, pp. 1814 - 1820.

[4] D. C. Doughty et al., "An ATM Event Builder for the CLAS Detector", in Proc. Second International Data Acquisition Workshop on Networked Data Acquisition Systems, Osaka, Japan, 13-15 November 1996, World Scientific Publishing 1997, pp. 124-131.

[5] D. Calvet et al., "Performance Analysis of ATM Network Interfaces for Data Acquisition Applications", in Proc. Second International Data Acquisition Workshop on Networked Data Acquisition Systems, Osaka, Japan, 13 - 15 November, 1996, World Scientific Publishing 1997, pp. 73-80.

[6] O. Gachelin et al., "ROBIN: A Functional Demonstrator for the ATLAS Trigger/DAQ Read-out Buffer", in Proc. Second Workshop on Electronics for LHC Experiments, Balatonfüred, Hungary, 23-27 September, 1996, pp. 204-207.

[7] R. E. Blair et al., "The ATLAS Level2 Trigger Supervisor", in Proc. Second Workshop on Electronics for LHC Experiments, Balatonfured, Hungary, 23-27 September, 1996, pp. 191-193.

[8] M. Costa et al., "Lessons from ATM-based event builder demonstrators and challenges for LHC-scale systems", in Proc. Second Workshop on Electronics for LHC Experiments, Balatonfüred, Hungary, 23-27, September, 1996, pp. 208-214.