ATLAS Internal Note DAQ NO. 46
31 August 1995

Event Building Protocols

VERSION 1.0 June 1995

André Bogaerts[1], Manfred Liebhart[1], Jean F. Renardy[2], Ralf Spiwoks[1], Per Werner[1]

[1] CERN, ECP Division, 1211 Geneva-23 Switzerland
[2] CEA SACLAY DAPNIA/SPP F91191 Gif/Yvette CEDEX France

Abstract

Control Protocols for parallel Event Building in HEP DAQ Systems are described. It is shown that these can be based on two simple data structures suitable for a wide class of architectures. A few examples of different push and pull architectures are given.

1 Introduction

A *parallel Event Builder* is illustrated in Figure 1. Data is produced by n *Data Sources* each containing event *fragments*, numbered sequentially 1,2,..i,.., which are combined into whole events in one of the m *Data Destinations*.

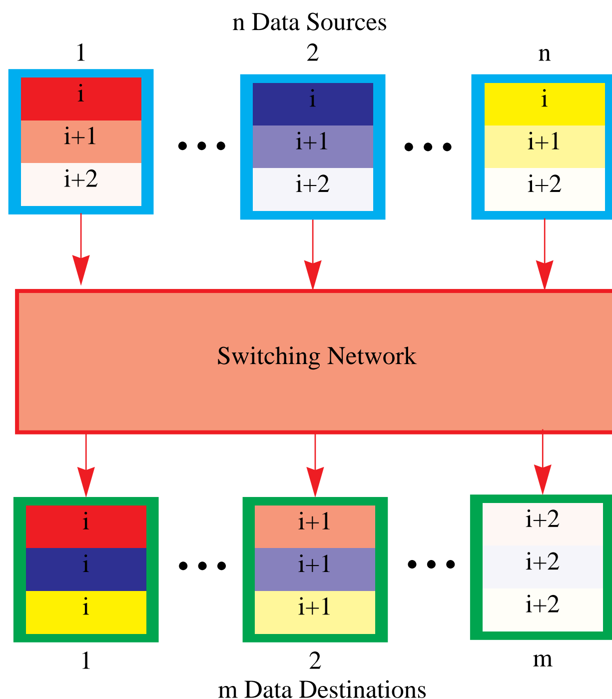

Fig 1. Parallel nxm Event Builder

We allow that only a subset of the sources contributes to the Event Building process. Examples of Event Builders can be found in the proposed implementation of the CMS [1] "Virtual Level-2" and Level-3 Trigger as well as the ATLAS [2] Second and Third Level Trigger. The sources may be thought of as Read-out Dual-Port Memories (RDPM, for CMS [1]) or LVL2 Buffers or Processors (ATLAS [2]), the destinations as Second or Third Level Trigger Processors, interconnected by a network.

First, we shall describe a broad class of Event Building protocols and show that these can be based on two elementary data structures which keep track of the location of the fragments which eventually will constitute the whole event. Event Building protocols may then be described by algorithms which update the data structures. These may either be executed directly by the sources and destinations if the tables are located in shared memory, or indirectly by the exchange of messages with an *Event Manager* that controls the tables. Event management and tables may be distributed if required for scalability or modularity of the detector. Protocols have to address a multitude of issues such as data copying or just data access, push/pull architectures in case of copying, trigger decision latencies, scheduling of processors, distributed or centralized event management and robustness/error recovery [3]. A strategy for choosing control protocols is given in [4]. The choice of actual protocols and implementation details may depend on the technology. A few examples will be given. Detailed results from demonstrator systems and simulations can be found in specialized literature [5],[6],[7].

2 Event Building Protocols

2.1 Terminology

Event building protocols are naturally layered in *data transmission, dataflow control* and *configuration*.

Event and control data may be *transmitted* over a unique or separate network for which drivers and higher level software are often required.

The *dataflow* is *controlled* by Event Building protocols which assign destination processors. They ensure correct movement of data and synchronisation of processors. Ideally, they make maximum use of the bandwidth of the interconnect, do not leave processors idle, minimize trigger decision latencies (to minimize memory), recover from malfunctioning of equipment and allow flexibility in the implementation of trigger algorithms.

Configuration is traditionally part of the Run Control and involves selection and initialization of all the components (processors, memories, interconnects, trigger algorithms). The Run Control will need to continuously monitor the system, log anomalies and reconfigure in case of persistent malfunctioning.

The term *source/destination driven* indicates how the data flows. In a source driven architecture, data is *written* by the sources. In a destination driven architecture, data is *read* by the destinations. In systems supporting shared memory it is possible that data is (remotely) accessed (read) rather than transferred in its entirety. This is a particular case of a destination driven architecture.

The term *push/pull* relates to the dataflow control and indicates that the source/destination *initiates* the transfer. It is thus possible to have a destination driven push architecture, e.g. when the source has the possibility to activate a DMA device in the destination.

This is perhaps the place to dispel a few myths. Push architectures allow multiple parallel data transfers to the same destination, but this may unfortunately overload the receiving port resulting in data loss (ATM), long delays (in establishing a connection, Fiberchannel) or thrashing (caused by retry traffic, SCI). Sources cannot start pushing data as soon as this is available but must wait till the destination is ready to accept it. Equivalently, in a pull architecture a destination may initiate a transfer when it is ready and data is available in the source. Push architectures cannot therefore guarantee low latencies in general and the choice depends on detailed considerations.

Destination processors can be assigned centrally by an Event Manager or the task may be distributed over sources and destinations. Several strategies are possible to determine the destination for an event. In addition, it must be known if the destination processor has queue space available and in some cases the address of the destination buffer must be supplied.

Non-deterministic algorithms (e.g. based on pseudo random number generators to associate a destination with an event) are for this reason not very suitable. Deterministic algorithms may investigate the queue sizes of the destinations to make the "best" decision. A very simple deterministic algorithm is round-robin scheduling where the destination is a simple function of the event number that can be calculated by any source and destination. The algorithm can easily be distributed. Another popular algorithm is to choose a processor which has no or the least number of events queued. This minimises latencies.

3 Data Structures

It is assumed that data is collected and buffered in the data sources. In push architectures destinations must also buffer events. How this is done (random access buffer, fifo, circular buffer, linked list, ...) and what the buffer size should be is outside the scope of this paper (see [4] for more details). Event Building and data management becomes much simpler if it is not required that an event occupies a contiguous block of memory. This allows sources to push blocks of data of unknown size, or for certain pull architectures, direct access to data without memory remapping. Fortunately, most Physics Analysis Software assumes that data is organized in smaller logical entities ("banks").

Two data structures form the basis for the Event Building protocols. Although we shall refer to these structures as tables they may be implemented in different forms such as linked lists or fifos.

A local, one-dimensional *Fragment Table* (indexed by event number) in each source or destination keeps track of the buffered event fragments. Each entry consists of an event fragment descriptor which contains all the information necessary to access a data fragment such as: a pointer to the event fragment data, its size and status.

A global, two-dimensional *Event Table* keeps track of all stored events (and its fragments) which are being processed ("alive"). The elements are also event fragment descriptors. One dimension ("row") is indexed by event number and associates all fragments belonging to one event with a destination. The second dimension ("column") is indexed by source identification and associates fragments with the source from which the data originated.

Figure 2 shows an example of these tables for a *pull* architecture. Rows in the Event Table contain pointers to fragments in all *sources* that contribute data to the event (some may be empty). A destination can start building an event as soon as it has the information of a complete row. A scheduling algorithm assigns rows to destination processors.

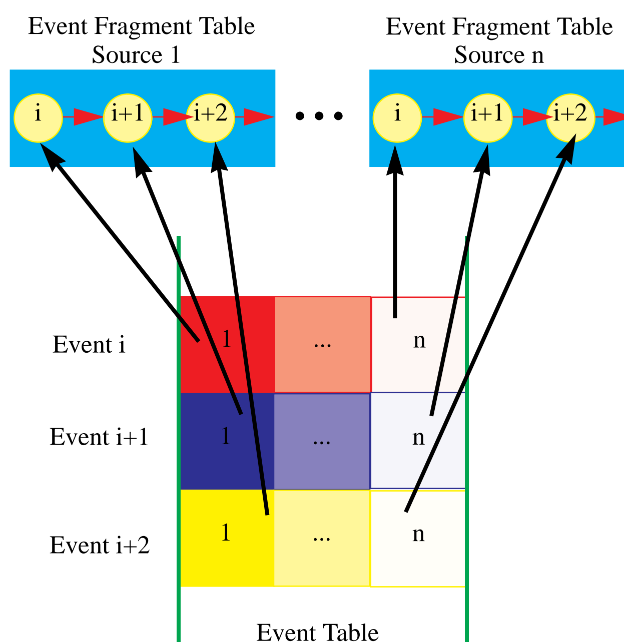

Fig 2. Event and Fragment Table for a *pull* architecture

Figure 3 shows an example for a *push* architectures. Each row of the Event Table contains pointers to the buffers in the *destinations* to which each source must transfer the data. A source can start transferring data to the various destinations once the column information becomes available; a destination can start event analysis as soon as a row is complete (some fragments may be empty).

The Event Building process can be viewed as read and update operations of the Event and Fragment Tables which requires communication between sources and destinations. There are two practical ways to achieve this.

The tables may be accessed through messages to a unique owner e.g. a data source or destination for the Fragment Table, or an Event Manager for the Event Table.

Alternatively, tables may be kept in shared memory. If data items are modified by several processors access may have to be protected by (global) semaphores. This method can be very efficient and is particularly suited to SCI. Updates may have to be accompanied by messages, or even simply interrupts, if immediate action is required though simple polling in the "event loop" is often sufficient.

An important implementation issue is the physical location of these tables. For instance, the Event Table could be kept inside a unique Event Manager (which communicates through messages) or a *globally shared memory segment* (for SCI). For large systems this may not be scalable and the Event Manager or shared memory may be overloaded, apart from the practical connection problems to provide a communication path. It might therefore be necessary to decompose an Event Manager or distribute the table over shared memory. The table need not have a fixed size and might be implemented as a list of linked lists. For SCI, coherent caching would automate the distribution.

.

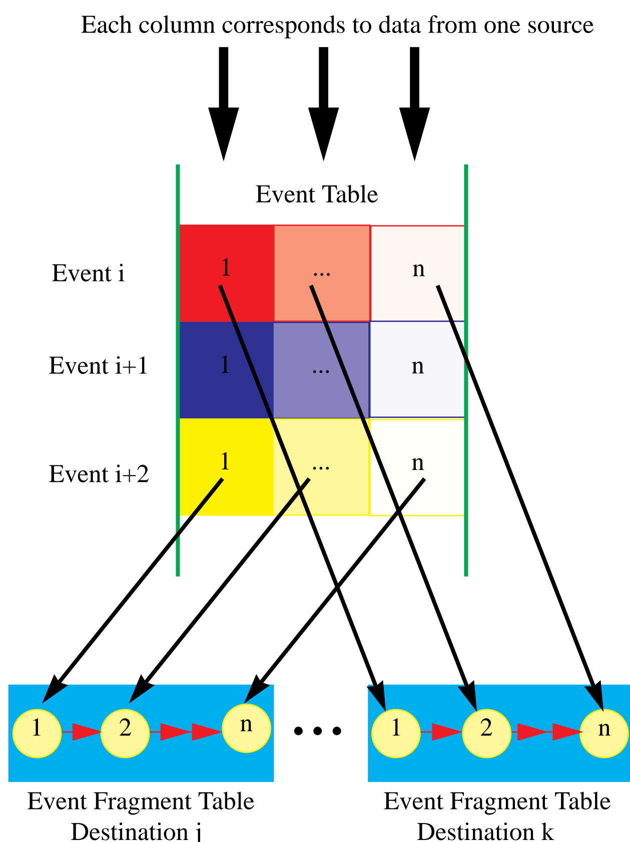

Fig 3. Event and Fragment Table for a *push* architecture

3.1 Event Fragment Table

For pull architectures it is convenient to locate Fragment Tables in each source. Since new elements are inserted in the table for each new event by the source, accessed sequentially, and deleted under instigation of the destination, a practical implementation is a simply linked list ordered by increasing event number as illustrated in Figure 2. Elements are a inserted at the tail and deleted in the vicinity of the head.

For push architectures the destinations manage the memory which is to receive the data and maintain therefore the Fragment Table. Fragments are grouped into events and several events may be queued which suggests that a two-dimensional table might be used. Otherwise, Fragment Tables could be integrated in standard Buffer Managers or implemented by software packages for Physics Data Management.

3.2 Event Table

The Event Table must keep track of all “live” events in the system. It is reasonable to keep an upper bound N on their number. Stale events may be timed out and forcefully deleted. Taking the ATLAs Second Level Trigger as an example, a reasonable value is N ~ 1000. This corresponds to 10ms data taking at 10µs mean event arrival time and results in a size of a ~ 1 Mbyte for ~ 200 sources. A small fraction of the events may stay much longer in the system. T1 should ensure unique event numbers during this time. The Event Table can than be organized as a linear array of size N and indexed by event # mod N for efficient access to each event. Each element of the Event Table must provide pointers to elements in the Event Fragment Table as indicated in Figure 2 and Figure 3.

For SCI the Event Table could be located on a shared memory segment, physically distributed over the sources or destinations. Event Management becomes then part of the event building process using the same network as for the data transport. Such an implementation is scalable and provides the redundancy required for robustness.

4 Examples of Event Building with SCI

As a practical illustration we describe a very simple SCI based system. This is not a production DAQ system but a test vehicle to study event building protocols. Initially, the system consists of 4 SUN Workstations and an SCI switch to form a 2x2 event builder with two sources and destinations. The system is being expanded to incorporate a composite SCI switch (4 four-switches) and C40.

The system is constructed from three simple, single threaded programs: n Data Producers, m Event Builders and a rudimentary Run Control Program (see Figure 4). Event Table, Fragment Tables and event data are located in shared memory. For simplicity, we assume that all Data Producers contribute fragments to the full event. Destinations are assigned in round-robin fashion. Event management is fully distributed over the sources and destinations which communicate using variables in shared memory.

All SUN Workstations run SunOs and are equipped with an SBUS/SCI interface from Dolphin, based on the first generation of CMOS interface chips. Despite the rather ineffecient first generation hardware this results in fast, scalable protocols with a measured rate of ~ 25 KHz (pull) - 50 KHz (push) for zero-length events (measuring protocol overhead only) [8].

4.1 Destination driven Shared Memory Pull Architecture.

Each source has a segment containing a Fragment Table and event data which is mapped in all destinations. Events may be accessed directly without transferring all the data. Because of the round-robin scheduling, destination i only needs access to every i^{th} entry in the Event Table which can therefore be physically distributed over all destinations. Each source maps to the complete Event Table. Rows in the Event Table provide a “handle” to a complete event by following the pointer chain. The format of both the Event and Fragment Table is compatible with ZEBRA-like data structures allowing an event to be scattered in memory.

All communication is through shared memory. There are neither interrupts nor calls to the Operating System to avoid

spending time in context switching for normal event processing.

For a pull architecture, event building consists of reading the event data fragments serially. Several choices are available: transfer the totality of the event (using DMA), transfer blocks piecemeal or just access the data in situ (SCI transparent read/write memory access).

The task of event management is distributed over the sources and destinations. The number of messages (read/write operations which must traverse the switch) in this implementation is 4n (could be reduced two 2n). Assuming n ~ 200 and an 10µs event arrival time this results in a data rate of hundreds of Megabytes/s just for synchronisation.

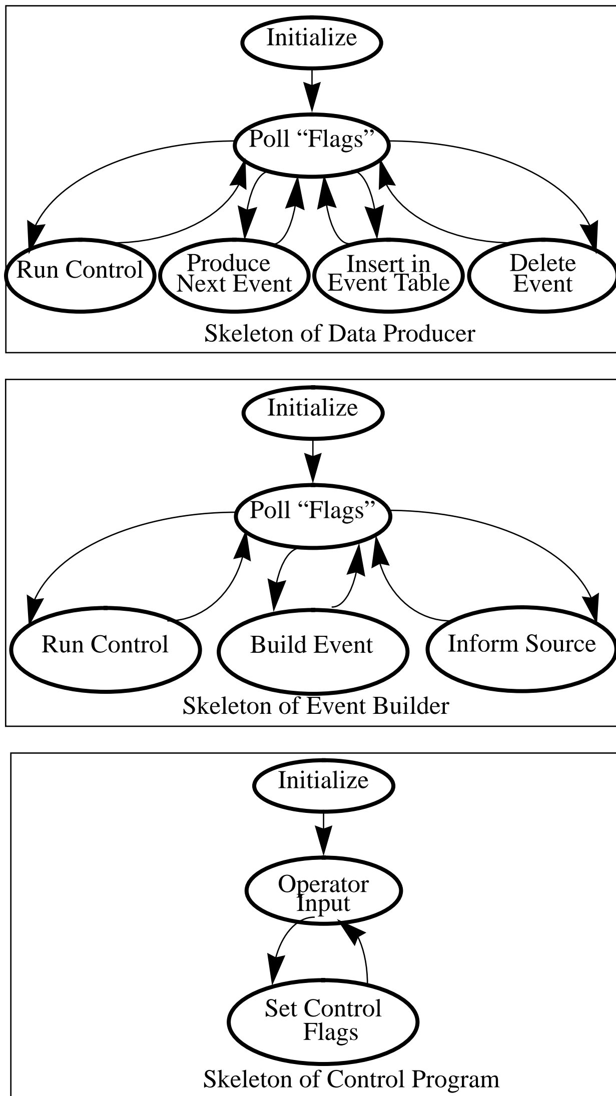

Fig 4. Data Producer, Event Builder and Run Control

4.2 Source driven Push Architecture

Many data networks can only support source driven architectures and these have therefore received more attention. To benefit from the potentially low latencies one has to be careful in the design to avoid overloading the I/O system of the destination as well as waiting times in the receiving data queues.

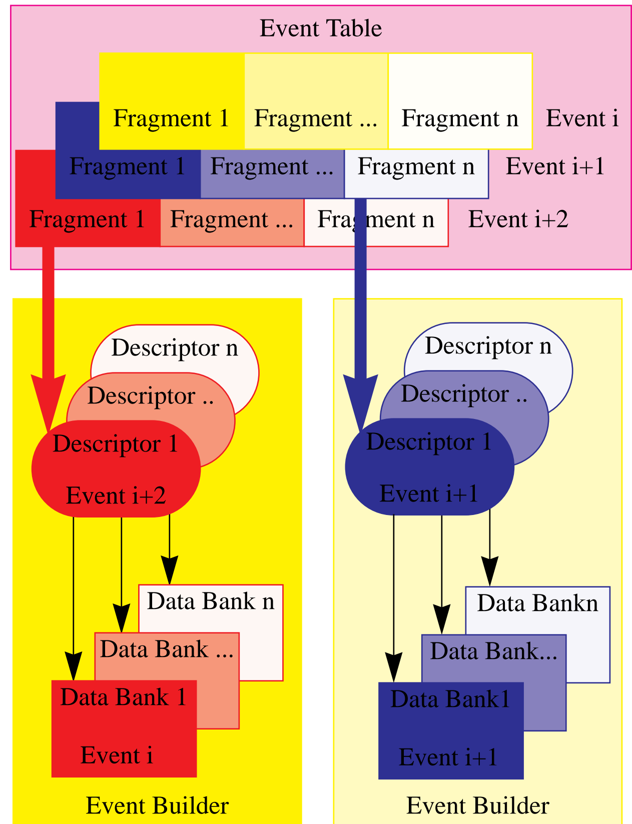

Fig 5. Data Structure in Destination Processors

In this SCI implementation we shall use shared memory for the passing of messages, as in the preceding example. Since the destinations need to buffer events, they each hold a Fragment Table accessible by the sources. Each entry in the Fragment Table now contains a pointer where to receive the data of the fragment as well as a data ready status indicator. This indicator must be read and tested by the source which, when it indicates ready, transfers the data and sets the status to data available. The Data Destination should fill the Event Table with valid pointers ahead of time for as many events as it wishes to buffer. The Data Banks and Fragment Table combined with the corresponding entry in the Event Table creates a ZEBRA-like scattered data structure in each Destination Processor as illustrated in Figure 5.

5 Examples of Event Building with HIPPI

5.1 Source-Driven Push Architecture based on HiPPI

Another source-driven push architecture can be implemented using the HiPPI standard [9].The data sources and data destinations are connected via a HiPPI switch [10].

Each source holds an Event Fragment Table for all the arriving event fragments. This table can be implemented as a linked list of event descriptors which contain pointers to the actual data. The sources assign the destination to a given event fragment in a round-robin manner or use a field in the event

descriptor filled by some earlier stage (e.g. T2 Trigger Supervisor). The sources then send the data to the destinations using the HiPPI protocol. In this protocol the sources can "camp-on" the switch waiting till the requested destination becomes available. Several camping-on requests are arbitrated by the switch in a round-robin manner. The end of a transfer is defined in the HiPPI protocol by a destination signal propagating back to the source which can then start "pushing" the next event fragment.

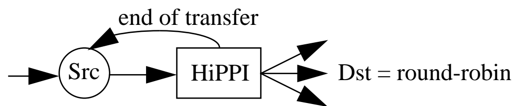

Fig 6. Push Architecture based on HiPPI Protocol

The event assembly is done in the destinations which hold each one its part of the Event Table, this table being implemented as a linked list of the event descriptors of full events which point to a linked list of event fragment descriptors.

This architecture which uses no other feedback from the destinations than the one defined in the HiPPI protocol can equally be implemented using FibreChannel class1 connections [7],[11].

5.2 Source-Driven Pull Architecture based on HiPPI

A source-driven pull architecture can be implemented with the same setup as described in section 5.1 on page 5. Instead of assigning the destination statically in a round-robin manner the data will wait for a destination to send a request. This request can be generated by the destination whenever it has enough buffer space to build a new full event. The request is broadcasted to all the sources by a network which has to ensure that no request gets lost and that they arrive at all the sources in the same time-order they were produced at the destinations. In this way the request can be used by the source to assign the "next" event fragment. A possible implementation is done using the VMEbus for this purpose [12].

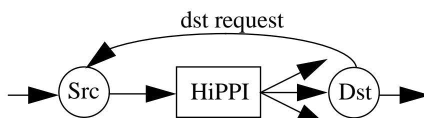

Fig 7. Pull Architecture based on HiPPI Protocol

The Event Fragment Tables are kept in the sources, the Event Table in parts in the destination and they are both implemented as linked lists.

This architecture can equally be implemented using FibreChannel class 1 connections in which case the FibreChannel links could also be used to transport the destinations' requests.

6 Event Building with ATM

Examples of ATLAS Leve-l2 and Level-3 architectures are given in [13].

7 Trigger Scenarios

The protocols discussed above are fairly general and applicable to both second and third level triggering. To be discussed: ATLAS/CMS model, RoIs & ATLAS T2, T2 buffering and event management, assignment of destination processors, local processing, parallelism, farms, massively parallel systems, phased event building, role of latencies, impact on memory usage, switch capacity, CPU usage, algorithms, software aspects, drivers, libraries, operating systems, flexibility.

8 Robustness and Error Recovery

Design of a robust system requires careful analysis of the cause of expected errors, their frequency and the amount of error recovery which is built into the communication network. Treatment of residual errors and, in some cases, recovery of data complicates and obscures production quality code considerably. It may also lead to the choice of a particular control protocol. For clarity, this aspect has been ignored in all the examples.

References

[1] CMS Collaboration, "Technical Proposal", *CERN/LHCC 94-38 LHCC/P1 CERN, Geneva, Switzerland 15 Dec. 1994*

[2] ATLAS Collaboration, "Technical Proposal", *CERN/LHCC 94-43 LHCC/P2 CERN, Geneva, Switzerland 15 Dec. 1994*

[3] R. Spiwoks, Requirements of an Event Building System, *RD13 Technical Note 111, April 1994*

[4] J.F. Renardy, "A taxonomy of control protocols for event building with switches", *To appear in the Proceedings of RT95, IEEE Conference, Michigan*, 1995

[5] S. Hunt, F. Harris, A. Bogaerts, J. Carter, R. Hauser, I. Legrand, "SIMDAQ - A System for Modelling DAQ/Trigger Systems", *To appear in the Proceedings of RT95, IEEE Conference, Michigan*, 1995

[6] B. Wu, A. Bogaerts, H. Li, B. Skaali, "Modelling of the ATLAS Data Acquisition and Trigger System with SCI", *To appear in the Proceedings of RT95, IEEE Conference, Michigan*, 1995

[7] W. Greiman et al., Design and Simulation of FibreChannel Based Event Builders, RD13 Technical Note 132, Oct. 1994

[8] M. Liebhart, A. Bogaerts, P. Werner, "Event Building with SCI", *RD24 Report*, 1995

[9] HiPPI-PH, ANSI standard X3T9.3/91-005

[10] S. Buono et al. (RD13 Collaboration), Prototype of an Event Building System based on HiPPI, to appear in Proc. of. Int DAQ Conf., Fermilab, 1994

[11] Fibre Channel Standard, ANSI working group X3T911/FC-PH

[12] G. Ambrosini et al. (RD13 Collaboration), Studies on Switch-Based Event Building Systems in RD13, submitted to CHEP 95, Rio de Janeiro, 1995.

[13] D. Calvet et al. "A Study of Performance Issues of the ATLAS Event Selection System Based on an ATM Switching Network", *To appear in the Proceedings of RT95, IEEE Conference, Michigan*, 1995