Design and Implementation of a VME Based Event Building Protocol for the New ALEPH Data Acquisition System

Josep A. Perlas
CERN, CH-1211 Geneva 23, Switzerland

ALEPH 94-062
DATACQ 94-001
J. A. Perlas
06.05.1994

Abstract

This paper describes and discusses the design and implementation of the protocol for event building in the new ALEPH VME readout system. Special emphasis is given to the consequences of dealing with a large and complex multi-crate system, a feature that is not incorporated in the specification of the VME standard.

INTRODUCTION

The new ALEPH readout system contains 20 VME crates. The original system was implemented using Fastbus [1] and has recently been replaced by commercial VME hardware, while keeping Fastbus in the front-end digitizers [2]. The readout protocol had to be revised and a new implementation made.

Data generated in each detector component has to be collected and assembled correctly. The readout protocol guarantees that this is done properly. The protocol is implemented as a software library, the *readout library*, that ensures that it is strictly obeyed by detecting protocol violations during event assembly.

To implement a multi-crate VME based protocol to perform event building for ALEPH, different possibilities have been considered. This includes protocols based on pure VMEbus and VICbus resources and also protocols based on processor-specific resources. The advantages and inconveniences of the considered cases will be discussed and the final choice justified and described.

ABSTRACT PROTOCOL OVERVIEW

In the old Fastbus readout system, the *abstract* event building protocol was as follows. The readout function was separated into two independent and asynchronous tasks: the sender and the receiver, which ran in different processors in the different stages of the event building "tree". The sender was responsible for detecting the presence of a new event and asserting a request indicating data available. The receiver read the different pieces of data according to the requests received from its sources until the whole event had been assembled.

The complete description of the abstract data transfer protocol, together with the Finite State Machine model used to handle the different phases of it, can be found elsewhere [3,4].

A PROTOCOL FOR A VME MULTI-CRATE SYSTEM

After having decided to move to a new readout hardware, a revision of the old readout protocol had to be made. We started by looking at the function-

alities that the new hardware could provide, classifying the considered possibilities into two main groups: protocols based on resources of pure external busses (VMEbus and VICbus) and protocols based on processor-specific resources[1].

In the first group we can shortly mention the following:

- *Using VME interrupts.* The sources in a VME crate generate a VME interrupt which is then translated to a VIC interrupt and finally to a new VME interrupt in the destination crate. This needs a VME interrupt handler to identify the source. In addition, the identification information of the different sources is not easy to handle.

- *Using VIC mailboxes.* A source in a VME crate writes in one of the mailboxes of the VIC module its identification information, generating a VIC interrupt. This generates a VME interrupt in the destination crate. It has the limitation of eight sources per crate and needs also an interrupt handler.

In the second one we can emphasize:

- *Using FIC mailboxes.* A source writes directly its identification information into the receiver mailbox RAM, generating a CPU interrupt. This allows for a big number of requests but it doesn't provide enough freedom for the source identification information data structure.

- *Using FIC fifos.* The sources also write directly into the receiver using the VIC module in *transparent access* but in this case they write into the fifo port of the FIC. This generates a CPU interrupt in the receiver who will then read the source identification information in a special memory accessible through VME.

Clearly, a protocol based on the first group offers the advantage of being processor-independent, which means that it can be replaced by any other VME processor without changing anything in the readout configuration. Nevertheless, the poor resources offered by VME (e.g. the difficulties in handling the identification of the different sources) means the protocol would not be robust, which was considered essential in the design.

On the other hand, a protocol based on the second group, although it is processor-dependent, offers more robustness since it is 'simpler' and doesn't rely on additional software to support, for instance, VME interrupt handling.

At this phase of the design, we didn't consider it as absolutely necessary to follow the specifications given by [4]. Nevertheless, since we had a positive experience running with this approach, we wanted the new design not to be very different from the old one, assuming that a correct implementation could be found. For instance, although a *push* protocol[2] is also a valid one, it introduces completely new schemes of work with the corresponding new problems and our lack of experience. Consequently, rather than investigating further the *push* protocols, we decided to

[1] See [5,6] for an explanation of the technical concepts used in the following discussion.

[2] In a *push* protocol it is the source who writes its data into the receiver instead of the receiver reading the source.

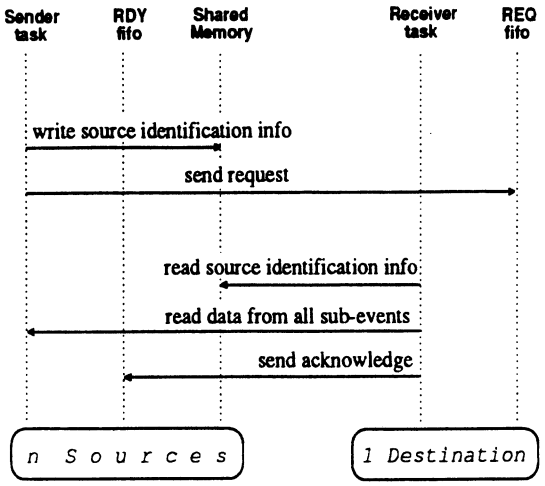

Figure1: 'Event trace' diagram for the ALEPH readout protocol.

chose the old and safe *pull* protocol, as described in the previous section.

Taking into account its anticipated robustness and simplicity, and given the fact that we had decided to chose a FIC8234 as processor after an evaluation performed on several processor boards [2], we finally decided to use the *"FIC fifos"* option to implement our readout protocol. Furthermore, the *'enable request'* phase as specified in [4] was considered not necessary and has been removed from the protocol.

DESCRIPTION OF THE IMPLEMENTATION

In the design of an event building protocol, two important aspects have to be considered: how the sources notify the receiver of available data and how the receiver identifies the different sources.

For the first function, an interprocessor interrupt facility has been used. This is based on the FIC8234 *fifos* [5], a powerful resource of this processor. In this approach, the source has simply to write a longword (with a pre-determined format) into the fifo port of the receiver. This is done by writing to the local VME bus at a certain address such that if the VIC8251F module [6] is used in *transparent access*, this generates a remote VME cycle through the VIC bus.

To implement this facility, an OS9 device driver and library have been written. Its purpose is to buffer the fifo messages and dispatch them to the requesting processes. Furthermore, since our data acquisition system supports the concept of *partitioning* [1], the driver uses different *channels* to dispatch the fifo messages. A channel is a number used by the driver to create a link between two communicating tasks. In this way, the library implements multi-user access to any fifo.

A limitation in the functionality of the fifo port has influenced the protocol implementation: it is not possible to perform DMA transfers into it. Consequently, our original idea of writing the source identification information (consisting on 4 longwords: ID number, event buffer offset, event length and trigger number) into the fifo port was discarded. Instead, we use a special memory (separated from the OS9 system memory and having a colour number) called *shared memory*, accessible through VME.

To implement the protocol, our first approach was to allocate three fifos in each processor of the readout tree. One to signal *Data Available* (called REQ fifo), another for *Data Ready* (RDY fifo) and a third one used to store the indices to the shared memory of the receiver (IDX fifo); this last one is not attached to the driver and it is merely used as a FIFO memory.

When a source has data available, it takes an index from the IDX fifo and

uses it as a pointer to the shared memory where it writes its identification information. Then it writes the index into the REQ fifo of its receiver to interrupt it and be read-out. Once all the sources are read, the receiver writes into the RDY fifo of its slaves who then free their event buffer and prepare for the next event.

After having exercised this implementation in the real experiment (see below), we decided to locate the shared memory in the sources instead of in the receiver and not to use the IDX fifo. The aim was to minimize the number of 'dead locks' in the VIC bus. See Fig. 1 for the corresponding 'event trace' diagram.

Tests and performance

Extensive tests in the laboratory have been performed to check the viability, reliability and performance of the protocol described in the previous section. In a first test a single crate was used, containing 1 receiver reading 3 sources. The sources generated small events (64 bytes) at a rate of about 240 Hz (these conditions were chosen to generate very frequent VME arbitrations). The test showed almost no error after more than 14 million events.

In a second test, two VME crates interconnected with a VIC bus were used: one housing the receiver and the other the 3 sources. This time the aim was to gain experience with the potential *dead lock* situations, expected as 'normal' errors when more than one bus is used. As expected, after having enabled the 'VME retry' mechanism we only found *Bus Errors* in some DMA transfers (when the receiver was reading a source) which were cured by a single software retry at the level of the readout library[3].

Although the tests done in the laboratory gave us some confidence that the protocol was working very nicely, the real experiment with the real readout configuration was the ultimate check. Preliminary tests performed with a small partition of the ALEPH detector containing 1 receiver and 6 sources showed immediately hardware imperfections in the VME Bus Requester module of the VME Master Interface of the FIC and also in its VME Slave Interface and in the Slot 1 Function provided by the CES VMDIS 8004 display module. All these problems were temporarily overcome by adding checks in all the VME actions performed by the readout library and retrying if a Bus Error was found.

After having solved the hardware problems, made more robust readout software and tuned the different parameters in the VME system (arbitration scheme, requester mode and bus timeout values), the event building was running well. The cosmic runs made during the commissioning period add another prove to this.

Conclusions

The upgrade of the ALEPH readout system from Fastbus to commercial VME hardware motivated a revision of the old event building protocol and led to a new implementation of it. Amongst several considered possibilities we have chosen one that adapts well to our needs while being robust and acceptably simple for a multi-crate VME system. The mechanisms used to achieve this have also been described. The extensive tests performed

[3]The BMA hardware provided with the FIC8234 to perform DMAs does *not* have the 'retry feature'.

in the laboratory and in the real experiment during the commissioning phase show that this design adequately fulfills the original requirements.

ACKNOWLEDGEMENTS

I would like to thank B. Jost, of the ALEPH online group, for many useful discussions.

REFERENCES

1. W. von Rüden, "The ALEPH data acquisition system", IEEE Trans. on Nucl. Science **Vol. 36, no. 5**, 1444-1448 (1989).
2. D. Botterill et al., "Report of the Study Group for a New ALEPH Readout Processor", ALEPH internal communication (1991).
3. J.A. Perlas et al., "The new implementation of the event building protocol for the ALEPH data acquisition system", IEEE Trans. on Nucl. Science (1993).
4. ALEPH Dataflow Group, "ALEPH Data Acquisition System Hardware Functional Specifications", ALEPH DATACQ note 85-21, CERN (1985).
5. Creative Electronic Systems S.A. (Geneva), "FIC8234 Dual 68040 Fast Intelligent Controller, User's Manual" (1993).
6. Creative Electronic Systems S.A. (Geneva), "VIC8251F VIC to VME Interface with Mirrored Memory, User's Manual" (1993).