CERN-Data Handling Division
DD/85/24
October 1985

An intelligent approach to complex systems architectures

T.J. Berners-Lee and E.M. Rimmer

(To be presented at the "Nuclear Science Symposium 1985",
San Francisco, October 23-25, 1985)

An Intelligent Approach to Complex Systems Architectures

T.J. Berners-Lee and E.M. Rimmer
C.E.R.N., 1211 Geneva 23, Switzerland

Abstract

Powerful processors embedded in complex data acquisition and control systems can support system design methods based on message transfer as a bus-independent means of communication. The implications of this approach on FASTBUS interface and interconnect design, and on mixed-bus architectures, are discussed.

A Single Master System

The CFI CERN FASTBUS Interface [1], [2] is an M68000-based module designed for providing host access to FASTBUS. We first describe its use in data acquisition, in a single-master FASTBUS system. Later, we will draw conclusions about the future design and use of modules in a multi-processor data acquisition system.

The CFI - Host Interconnect

The interface between the CFI and the host consists of parallel I/O channels. Messages sent over these channels allow lists of FASTBUS operations to be loaded and subsequently executed as required. When a list is executed, all the data read in is concatenated, with optional formatting information, and sent back in a single large block. There is no "scattering" of data automatically within the host's memory. The CFI and host are quite unaware of the physical address of the data in each other's memory.

This interconnect has proved easy to implement on a number of hosts, being simple both in hardware and software. Whereas the interconnect does in fact use two channels, one for supervisory and one for data, it is now felt that this is unnecessary: one physical channel would be sufficient, and still simpler.

Programming Remote Operations

The CFI is programmed for a particular task in terms of a "list" of primitive FASTBUS operations. For fast data acquisition, the lists remain loaded throughout a run, whereas for simple testing they are often loaded each time they are used. The list is sent to the CFI in a common "F-Code" format [3]: the M68000 then converts it locally into instructions for execution by a hardware sequencer. The principle of loading and invoking a complex operation on a remote processor has proved useful and efficient.

There are, however, operations for which a list of FASTBUS operations is not sufficiently powerful. These operations include the handling of Service Requests, performing a "sparse data scan" function, or reading out data from a complex acquisition module. These operations require a certain amount of in-line computation: for instance, pointers must be read from the slave, and these used to determine addresses and word counts for following read operations. An example of this problem was highlighted by the recent Time Projection Digitiser tests undertaken by the Aleph collaboration [4].

The architecture of the CFI is specifically designed for the efficient handling of lists. The architectures of most embedded processors such as the Delphi General Purpose Master [5], the Aleph Event Builder [6] and Time Projection Processor [7], or the SLAC Scanner Processor [8], however, allow the intimate mixing of FASTBUS access and general computation. This has suggested extending a general purpose programming language (such as FORTRAN) to describe the remote operations. A special FORTRAN compiler [9] is used, for example, with the Event Builder, which recognises calls to FASTBUS routines and replaces them with in-line code to directly access FASTBUS. Unfortunately this requires compiler adaptation to each device, so more general solutions are being sought.

The use of messages to load and to invoke a remote operation is equally applicable when that operation is represented by a general software module, rather than by a list of FASTBUS operations.

Multiprocessor Systems

We now consider the extension of these techniques to a multiprocessor data acquisition system.

Requirements of the Software

In a large system, such as a data acquisition system for a modern high energy physics experiment, it is convenient to design the system software in terms of the passing of a message as the basic unit of communication between tasks. The advantages, which will not be discussed here in detail, include the following:

- The method is as useful between tasks on a different processors as between those on the same processor;

- No problems arise over simultaneous access to common resources;
- A message may be used for synchronisation, as well as information flow.

The passing of a message, then, is the function which the bus system must provide to the system software.

Further benefits arise in that message-passing is a function which is independent of the nature of the host, interface and processor modules involved. It is also independent of the particular bus used. These portability aspects are particularly important in the heterogeneous environment at CERN.

The PALABRE [10] system designed by the OPAL collaboration is an example of an existing message-based system. This system originally ran over CAMAC, and is being ported into a VME environment.

Host Interface Implications

In the case of a multiprocessor system, the remote actions to be invoked will not always be executed by the host interface, but by yet more remote processors. The host interface must therefore be able to pass on a message to or from a remote embedded processor transparently.

If speed allows the entire data acquisition system to be designed in terms of message passing, message handling becomes the principal function of the host interface. The task of reading data from unintelligent acquisition modules is confined to dedicated embedded processors.

The new role of the host interface suggests a changed architecture. A processor with a large buffer memory is equipped with a FASTBUS interface which is sufficient for it to send and receive messages over FASTBUS. It is also equipped with a separate message interface to the host, which may be a simple DMA channel. This technique is used by the VIRTUS project [11], for example, using a GPM module with a parallel interface to a VAX UNIBUS adapter.

The provision of a buffered interconnect poses few problems in an environment already based on message passing. A buffered interconnect has essentially the same functionality as a host interface module, though the physical interconnect may be different.

Implementation in FASTBUS

Using Existing Modules

Given a scenario in which message passing is a primary

function, we now look at ways in which this has been, or could be, implemented using modules available today.

Firstly, consider the passing of a message from a processor with FASTBUS master capability to a slave device which is not specifically designed for receiving messages.

Typical schemes which have been proposed involve implementing, using software, a First-In First-Out (FIFO) buffer in the destination module's FASTBUS addressable memory space. The pointers associated with this buffer are also accessible from FASTBUS. The operation of the buffers is performed completely by software. The operation of sending a message, in this case, involves

- Reading the pointers
- Checking that space is available
- Writing the data into the buffer
- Updating the pointers.
- Alerting the destination processor.

This, incidentally, is an example of mixed FASTBUS access and computation.

The buffer may be in control or data space. It is faster, however, if the pointers and the buffer are in the same space, so as not to require disconnection and reconnection to the destination module. Alerting the processor may be done in a number of ways, using CSR registers. A FASTBUS interrupt message is a possibility, or simply a bit set operation.

Now consider the problem of sending a message from a processor which only has a slave interface with FASTBUS. In this case, the only option is to put the data into a data structure, or hardware FIFO, and raise the Service Request line. Delivery of the message then has to wait for service by a suitable master. In this way, the logical passing of a message has occurred, but in a slow and inefficient manner.

Hardware Techniques

Various hardware techniques could be used in a purpose-designed module to speed the message passing function.

Firstly, the processor sending the message must have FASTBUS master capability. For a device with two FASTBUS ports, it will ideally have master capability on both.

Secondly, the processor receiving the message should be able to allocate a part of its memory as an automatic

message buffer. A dedicated FASTBUS port, a single register address, is used for sending messages to the module. Data written into this port is automatically stored in the next place available in the allocated memory. The sender is unaware of the physical address into which the data is being written. A non-zero FASTBUS slave response may be used to indicate insufficient buffer space. Error recovery procedures for such a "Uni-address" device are discussed in [1].

Finally, the ability of the receiving processor to be alerted automatically on disconnection also saves time.

These features reduce the action required of the sender to a single FASTBUS block write operation, avoiding the necessity to read and write pointers. This is similar to the FASTBUS interrupt system, but differs in that memory usage is more efficient: one single port (buffer) may accept many messages until all the available space is used.

Also, to be of general use, a message must carry data of arbitrary content and arbitrary length, up to some maximum. This maximum varies from system to system, but tends to be of the order of 10^3 bytes. The 54 bytes of general information in a FASTBUS interrupt message, for example, while sufficient for use in transmitting a request for service, are not sufficient for general information flow.

Summary

The passing of a message is a useful primitive operation on which to base distributed system software, and provides portability.

If message transfer is used as a building-block for multiprocessor system software, a host interconnect must be capable of transparently forwarding messages.

Although messages may be transferred between existing processors using software techniques, careful design of embedded processor hardware can make message passing simpler and more efficient.

Acknowledgements

Many of the ideas in this paper derive from research into the future requirements of LEP experiments by a working group led by W. Bozzoli of CERN DD division.

References

- [1] "FASTBUS, A Modular High-Speed data Acquisition System for High-Energy Physics and Other Applications", U.S. NIM Committee, December 1983
- [2] D. Jacobs et al., "CFI, CERN FASTBUS Interface", CERN DD/ED FB/MAN1, December 1983
- [3] E.M. Rimmer, "CERN FASTBUS Software", CERN, 1 August 1984
- [4] G. Sanguinetti, CERN/EP, private communication.
- [5] H. Müller, CERN/EP, "The GPM general purpose FASTBUS Master/Slave, Multiwire version", CERN EP Note 85-02
- [6] A. Marchioro, "The Aleph Event Builder", *Proceedings of the FASTBUS Software Workshop*, CERN, 1985.
- [7] S.R. Amendolia, "The Readout Controller of the Aleph Time Projection Chamber", *Proceedings of the FASTBUS Software Workshop*, CERN, 1985.
- [8] H. Brafman, J. Olsen, "The SLAC Scanner Processor (SSP)", SLAC, August 1984
- [9] H. Von der Schmitt, "RTF68K Fortran Compiler", private communication.
- [10] OPAL Saclay On-line Group, "PALABRE", 28 Jan 85
- [11] J. Ellet, R. Ritter, D. Yaeger, J. Zweizig, "VIRTUS Software Design Document", UCLA October 1984