EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/DRDC/91-23
DRDC/P16 Add. 2
22 March 1991

Addendum to

A SCALABLE DATA TAKING SYSTEM AT A TEST BEAM FOR LHC

This addendum contains the development of the ideas proposed in the DRDC/P16 [1] and is intended to answer the referees' requests about the scopes and the activities of the project.

It also presents the new configuration of the collaboration, which, as already indicated in [1], has grown to include new members and two new groups.

C.P. Bee, R. Bonino, S. Hellman, R. Jones, L. Mapelli*,
G. Mornacchi, M. Pentney, S. Stapnes, N. Zaganidis
CERN, Geneva, Switzerland

G. Ambrosini, R. Ferrari, G. Fumagalli, A. Lanza,
F. Pastore, G. Polesello
Dipartimento di Fisica dell'Universita' e Sezione INFN di Pavia, Italy

R. Cardarelli, M.L. Ferrer, G. Mirabelli
respectively
Dipartimento di Fisica dell'Universita' e Sezione INFN di Roma "Tor Vergata"
INFN - Laboratori Nazionali di Frascati
Dipartimento di Fisica dell'Universita' e Sezione INFN di Roma, Italy

M. Aguer, T. Choulette, M. Coret, J.L. Fallou,
M. Huet, J. Pain, J.R. Poggioli
SEIPE - STEN, C.E. Saclay, France

* Spokesperson

INTRODUCTION

Since the open presentation we have continued to work on the following items as a development of ideas proposed in DRDC/P16:

1. Hardware layout
2. Front-end system software
3. Software layout and development
    3.1. Software layout
    3.2. Software development
4. Use of system
    4.1. Detector data taking
    4.2. Event building studies
    4.3. Examples of applications
5. Summary of phase 1 activities.

1. HARDWARE LAYOUT

During **phase 1** detectors will gradually move from CAMAC based to VME based electronics. The minimal system to ensure data-taking for the detectors thus requires a host computer capable of acquiring data from VME and CAMAC front-ends, recording data on permanent storage and allowing online monitoring and control. The R&D activities of the project proper will put additional demands on the hardware configuration, in particular the resident front-end processor. During phase 1 we will investigate the feasibility of running a real-time UNIX operating system and the data acquisition software in the front-end processor (see sections 2 and 3.1).

The above requires a powerful resident processor. At present we are concentrating our investigation on the CES RAID8235 module which is built around the MIPS R3000 RISC processor. Our choice of this processor is motivated by CES's decision to support the LynxOS real-time UNIX operating system. The necessary software drivers for a number of other VME modules will thus become commercially available. We believe that the architecture of the RAID module is very well suited to real-time data acquisition in VME. For example, the board features a dedicated DMA controller circuit that frees the processor from the chore of moving large blocks of data and helps to ensure that the data and instruction caches are used efficiently.

Figure 1 shows one possible implementation which would meet our requirements. Although one might consider implementing such a system in a one-crate configuration, we have chosen a multi-crate system from the outset for several reasons. Firstly, this increases the autonomy of the detector prototypes. During setting up and tests one can easily decouple a detector crate from the system and interface it directly to a host. Secondly, for full functionality at later stages a multi-crate system will be essential. The requirement that the system should be able to scale smoothly into such a

configuration more than justifies the extra overhead of installing a multi-crate system from the start. Although no final decision has been taken as to exactly which modules will be used in the system, we show in figure 1 how existing modules could meet our requirements.

We have chosen the Sun SPARC Station 2 as host machine firstly because of the excellent price/performance ratio available from RISC based workstations and secondly because of the existence of a convenient interfaces to VME and to the VIC-bus. We have decided to use the VIC-bus to extend VME to a multi-crate system, using for example the CES VIC 8250/8251 modules. The VIC-bus is the subject of a forthcoming international standard and thus fits in with our open system philosophy. An advantage of directly interfacing the host to the VIC-bus rather than to VME is that it makes for a more flexible architecture. We plan to test a prototype of the CES Sbus-VIC interface, SVIC 7213, in collaboration with CES (see section 5).

For what concerns the handling of the event trigger, a simple VME interrupt generator is adequate for the system of fig. 1. Nevertheless, in view of the growing system complexity and the evolution towards a distributed system, we might envisage the use of more flexible hardware. A VME trigger module is being designed by Ph. Farthouat (ECP/EDU), for the Omega experiment, which, in addition to transmitting a trigger signal to a VME processor via an interrupt, includes several other features: busy handling, trigger counting (event identification), busy monitoring (dead time measurement), support for multiple trigger destination.

During **phase 2** the system will be expanded to incorporate the increased functionality needed for our proposed R&D. Figure 2 shows one way in which the VME backbone can be expanded to accommodate this. As stated in our proposal, we intend to add a second host to create parallel production and development systems. This also gives us the ability to test scaling of process control and monitoring tasks. Both hosts will be interfaced to the VIC-bus, possibly using reflective memories which allow a shared memory system to be implemented easily. With the VIC vertical bus the VME system can grow in a transparent way to meet increased demands from detector systems.

We emphasise that the VME top-crate is probably the best location for adding increased functionality. If the feasibility of real-time UNIX in the front-end is demonstrated, a natural next step would be to implement a multi-processor system and to study various methods of organising the processors. Possibilities include transparent distributed computing relying on the operating system to handle task distribution and architectures using a master-slave arrangement of processors.

Multiplying the number of resident processors in the VME top-crate will also allow us to test several event building techniques. One such technique would be "classical" event building, in which a master processor distributes computing tasks and data is transferred over the backplane bus. We note that many of the

proposed alternative techniques, such as cross-bar switches using HIPPI or barrel switches, at present use modules residing in VME.

Other interesting areas of investigation are higher level trigger modules (see section 4.3), again an area where most ongoing studies utilize modules based in VME. It is thus clear that the system described here, in which the versatility of the VME, VSB and VIC buses are combined with distributed processing power, represents the best choice today for a system to cope with future technological progress. In particular, advanced bus systems such as SCI and Futurebus+ are being designed with fast and efficient interfaces to VME, providing a natural evolution path for our system towards the requirements of an LHC detector.

2. FRONT END SYSTEM SOFTWARE

The evaluation of a modern operating system environment for the front-end processors is one of the activities we intend to pursue during the first phase of the project. We have therefore defined a list of requirements, which we have compared against the specifications of the commercially available products and the possible choices for the hardware. As a result of these studies, we have selected the integration of a hardware module and a real-time UNIX environment available from and supported by industry.

2.1. Front-end system software requirements

We categorize our list of requirements as follows:

- UNIX and related standards. As indicated in the proposal, we want to experiment in the direction that we consider closest to industry trends. Hence, UNIX for uniformity and compliance to operating system standards (with particular reference to the POSIX suite of operating system interface standards).

- Real Time. We need deterministic response to external events (pre-emptable kernel and priority scheduling) to efficiently perform tasks driven by external signals, and a set of real-time features (such as those proposed by POSIX 1003.4) for the multitasking typical of back-end applications.

- Distributed computing. Our foreseen applications address problems typical of distributed and multiprocessor systems: redistribution of a task over a number of processors (to cope with scaling), and system reconfiguration (to cope with a changing environment and possibly to investigate fault tolerance).

- ROMable system software. The basic system we have proposed consists of a front-end with embedded processors and a back-end for development, controls and the man-machine interface etc. We, therefore, need system software which can run diskless, and be possibly ROMable, with NFS and X11 client support as well as cross development and remote debugging.

- Platform independence. As stated in our proposal, such a system is meant to accommodate a number of external developments and so it is important that the selected front-end operating system be available for all the major platforms.

2.2. Commercial Solutions

The commercial solutions intended to satisfy the industrial demand for real-time UNIX can be categorized into three groups:

- Real Time executives with some degree of UNIX compatibility, for example cross development is supported on a UNIX system while the target processor runs a non UNIX kernel (e.g. VxWorks [2]).

- UNIX systems, both modified System V or proprietary, to which real time capabilities have been added (e.g. Real/IX [3], LynxOS [4], RTU [5]).

- Specialised real time kernels to which UNIX support (typically System V compatibility) has been added (e.g. Chorus [6], DuneIX [7]).

We remark that with the current trend towards operating system interface standardisation (all companies questioned said they are committed to POSIX compliance) will make a future choice less critical and justifies our insistence on UNIX and compliance to standards. For this reason we think that, while technically viable, the real time executives category is not suitable to our needs.

2.3 Environment selection

We have compared a number of existing products against our requirements list, by reading documentation, organising technical presentations from the companies and in some cases by direct evaluation of the software. Two solutions seem particularly promising:

- LynxOS: a fast, hard real time UNIX system, available on a wide spectrum of processor types, whose release 2.0 will be compatible with the current draft of the POSIX 1003.4 specifications.

- Chorus: a conceptually modern, distributed, real time micro-kernel for which a UNIX (System V compatible) sub-system is available.

Although architecturally very different, both these solutions can solve our problems. The interesting possibility of a supported, integrated solution to the front-end processor (i.e. the RIAD8235 from CES), as well as the high cost of acquiring the Chorus technology, lead us to consider LynxOS as our best candidate.

3. SOFTWARE LAYOUT AND DEVELOPMENT

We intend to address the problems of data acquisition (DAQ) and support software by the exploitation of modern software engineering and programming language technologies. We plan to use commercially available solutions and to build, whenever possible and suitable with respect to our future developments, on existing (within HEP) technology.

3.1. Software layout

Given the strong R&D objectives specified in our proposal we have taken a bottom up approach to deal with the DAQ software. In this sense we will initially address a limited number of issues, perhaps evaluating various independent solutions, without an immediate consideration for a detailed global design. Nevertheless we will emphasise the design of interfaces between components to allow extensibility (to cope with additional requirements), interchangeability of implementation (to accommodate new developments) and multi-processing (to cope with system scaling).

In the following sections we split the description of the DAQ software objectives by distinguishing between a **phase 1** (1st year activities), aiming at a minimal system, and a **phase 2** (2nd and 3rd years) when implementation and technologies will be gradually introduced.

3.1.1 General Objectives

Phase 1: the functionality of the minimal system will be specified, inter-module interfaces will be designed according to the above requirements and programming techniques supporting these requirements will be evaluated and applied (in particular Object Oriented Programming, see 3.2). The operating system environment, experimental controls and the use of commercial products to maintain an information system and build user interfaces will be studied in detail.

Phase 2: the DAQ software will be extended to cope with hardware scalability and the evolution of existing modules from initial ad hoc to modern implementations. We will address at this time the issues concerning distribution of software components over a scalable number of cooperating processors.

3.1.2 Software Components

A broad layout of the necessary software components includes: data acquisition, controls (slow and run control), management of the information available in the system and computerised assistance to the operator (i.e. the physicist on shift). In addition we also aim to evaluate and understand the use of UNIX, and

in particular of real-time UNIX, in our application domain. In the following sections we briefly consider the software components, state our activity during phase 1 and give possible directions for phase 2.

3.1.2.1. Data Acquisition

The data acquisition software will initially (phase 1) respond to the minimal requirements of the collaborating detectors. With the exclusion of the read-out component, which will be completely developed by the project, we intend to reuse existing modules (e.g. buffer manager, data logger) and build around them an interface hiding their internal implementation. In phase 2 we will consider substituting some of these components with new implementations while extending their scope to a distributed, multiprocessor system.

An essential part of a data acquisition system is a module responding to event triggers and performing the necessary steps to read out the event into some storage. While its presence is mandatory to take data, we also consider the read-out module as the vehicle to evaluate the response capabilities of a real-time UNIX system. The read-out module will be able to react to external events (e.g. event triggers, control signals), to read out from different data sources (e.g. CAMAC, special VME modules) and to support different read-out sequences. The read-out module will be written by the project, during phase 1, so as to allow support for new read-out sources to be added in phase 2.

The event manager distributes events, once read, to appropriate software processes for recording and analysis. The event manager will provide allocation of resources (i.e. store and events) and support for data reduction (to possibly try out high level triggering algorithms online). A distributed scheme for the allocation of resources will be implemented from the start (e.g. transparent access from both front-end and back-end). While this task has often been performed by software we plan to exploit both our hardware layout and new hardware developments (e.g. reflective memories) to assist these resource allocation tasks. During phase 1 we will specify the functionality needed, design an interface expandable to new services and independent of the internal implementation then select a suitable existing software package for the internal implementation (e.g. SPIDER or the OPAL skeleton). For phase 2 we will consider making the event manager evolve towards hardware supported implementations more suitable for distributed multiprocessor architectures (e.g. via shared memory or replicated shared memory).

For the data logger we require recording on a 3480 compatible cassette driver in whatever format is required by the detectors. We will adapt an existing data logger to our needs.

For the production of monitoring programs, a template will be provided (in phase 1) adaptable to users needs with libraries to interface to the services (event manager, control system, database). In addition we intend to evaluate the use of commercial products to build user interfaces (see 3.2).

3.1.2.2. Operating System Support

The presence of a UNIX (-like) operating system in the front-end provides us with a number of important points for study. While UNIX itself may already provide a certain degree of uniformity between the front-end and the back-end, we want to go further concerning real-time services not directly related to interrupts and direct I/O. We will build on an existing package, VOS [8], which already runs on VMS, OS9 and several UNIX flavours. This work will also be in preparation for the availability of interfaces conforming to POSIX 1003.4.

Another issue concerning the operating system is that of drivers: while a number are already available with the operating system, we need to acquire experience in this kind of system programming in view of future developments (e.g. special external devices, etc.). We also need to understand the similarities between the system interfaces for drivers, the degree of driver portability a UNIX-like system can provide and the impact of standardisation efforts in this direction. Finally, we want to evaluate, in terms of performance and ease of use, a real-time UNIX system in our application.

Subsequently, we will address the problems related to the support of distributed applications and of multiprocessing.

3.1.2.3. Controls and information management

We intend to build on the expertise of the LEP experiments in the areas of controls and information management. A good starting point could be the L3 slow control system and the ALEPH online data base and software design principles. To this end, we have initiated discussions with J.M. Legoff and P. Palazzi from the ECP/PT group. The technologies involved are: expert systems, communications, graphical user interfaces, object oriented languages and data modelling. As already mentioned, these fields are evolving rapidly and need to be followed closely. Preliminary investigations are discussed in section 3.2.

3.1.2.4. Operator Assistance

System documentation and online help are two vital aspects of a user friendly data acquisition, especially if we consider the expected complexity and the particular sociology of an LHC-like experiment. Hypertext techniques, for which commercial products exist, are a technically interesting solution to the problem of providing powerful and easy-to-use online help, directly issued from the system documentation (see also 3.2). We plan to apply this technology to our system and have initiated discussions with Tim Berners-Lee of CN division.

3.2. Software development

3.2.1 Programming languages and Tools

While it is our intention to use as much commercially available software as possible, we will obviously have to produce some software ourselves. In an attempt to take a professional approach to software developments we have investigated many available tools, methodologies and environments.

There are a number of Computer Aided Software Engineering (CASE) tools available for the development of software, some of which have or will be evaluated at CERN, such as Software through Pictures (StP) from IDE, Geode from Verilog [9] and RTEE from Westmount. Using one of these tools implies adopting the associated software methodologies (such as Structured Analysis and Design) at a time when the software industry is starting to look beyond these techniques and investigate an object oriented approach. Object oriented programming claims to ease software maintenance by reducing the dependency between software modules and encourage software reuse, an important point for the capitalisation of existing investments. While there is a multitude of object oriented languages (Smalltalk [10], C++ [11], objective-C [12], CLOS [13], Eiffel [14], etc.) the associated software design techniques are lagging behind. IDE claim to be extending StP to incorporate Object Oriented Structured Design (OOSD) an extension of the work done in Ada by G. Booch [15] and that of Hierarchical Object Oriented Design(HOOD) from the ESA [16].

In order to allow for portability of code across the front-end and back-end we need to write applications in a language for which compilers and software development tools are readily available. C has been the traditional programming language of UNIX systems. Part of C's popularity comes from its flexibility and the lack of constraint it puts on the programmer but this is also the cause of many problems. Object oriented extensions to C provide extra control and consistency checking. The advantage of using a C hybrid language, such as C++, is the combination of general availability and efficiency. Applications can make use of the object oriented extensions to ease their implementations and resort to straight C code for sections that need to be optimised. Pure object oriented languages, such as Smalltalk and Eiffel, do not offer this flexibility and tend to be less efficient due to their pedantic implementation of object oriented principles.

To aid the implementation phase of writing software we are evaluating Saber C++ from Saber Software Inc [17]. Saber C++ offers a programming environment for developing C++ applications by providing various support tools, such as an interpreter, extra compile time and run time error checking, graphical browsers for the program code and data structures, an incremental linker to reduce the compile-link-run cycle time and a source level debugger.

3.2.2 User Interfaces

MIT's X11 window system [18] will be the basis of all our human interface needs thereby providing distributed access to applications using the industry de facto standard. We will not write any low level X code directly, but intend to rely on a suitable commercial interface builder, similar to Digital's VUIT, or use a combination of OSF's high level descriptive language UIL [19] and C++. A C++ binding for the Motif toolkit is available from Lowell University. We intend to follow OSF rather than SUN's Open Look for reasons of portability.

For application programmers, a User Interface Management System (UIMS), which sits on top of a Graphical User Interface (GUI) library, eases the creation of user friendly interfaces. Several commercial software products have been evaluated at CERN [20]. We plan to investigate these products further, in particular Teleuse, UIMX and VUIT.

Another approach is that of programming the user interface using a toolkit. InterViews, an object oriented GUI toolkit based on X11, provides a variety of interactive objects in a C++ library that runs on several platforms. It is already being used at CERN (e.g. in ALEPH) and we are currently evaluating its applicability to our case.

We plan to apply a two-pronged approach to the user interface problem: a UIMS tool for the more common case where a simple interface (e.g. some menus, dialogue boxes) is needed while an object oriented, extensible toolkit such as InterViews will be applied to cases where highly sophisticated graphic interfaces are needed (e.g. event display, iconic control panels).

3.2.3. Databases

The complexity of the data management problem in a modern HEP experiment makes the usage of a relational data base management system (DBMS) unavoidable. This was pioneered by the LEP collaborations (in particular ALEPH), who employed an entity-relationship model, also linked to a commercial data base product (SQL/ORACLE), to build up an integrated Data Management System (ADAMO). ADAMO could constitute the minimal solution for our problems and could be the starting point for the investigations.

On the other hand, the approach to system design outlined in the previous sections, together with the further increase in size and complexity of the problems to be faced, call for an even more flexible implementation extending the basic relational model to include object-oriented concepts and data types. This is a new field for which we need, in collaboration with other groups having common interests and objectives, to build up the necessary know how and to survey and monitor available commercial products while taking into account the additional constraint of their application to a real time environment.

4. USE OF THE SYSTEM

4.1. Detector data taking

We would like to test the following:

- detector data taking with various generations of electronics (integration of new readout hardware).
- combination of detectors: phase 1 configuration is naturally scalable in number of detectors (data sources).
- possibility of use of front-end processor for algorithm studies involving several detectors: kind of level 2/level 3 trigger simulation.

We have investigated, as an example, the next phase of the Silicon Track Preshower (SITP) readout and its DAQ implications [21]. At present an apparatus consisting of 6 printed circuit boards is foreseen forming a beam telescope. Each of these boards will carry an array of 4x4 silicon detectors with 64 channels per detector. Each detector will be read by two 32 channel ASIC front-end circuits. In addition to an amplifier and an analogue pipeline for each channel, these ASICs will contain some control logic and an output multiplexer (see figure 3a). On receipt of a trigger, arriving at a fixed time (nominally 1 μs) after the particle causing the trigger, the ASIC freezes one or more time slices in the pipeline corresponding to the time of the trigger. The channels corresponding to each of these time slices are then multiplexed to one of four analog to digital converters (ADCs) mounted on each printed circuit board. The multiplexing will be controlled either by a digital ASIC designed to work in tandem with the analog front end ASICs, or by a control circuit implemented in standard logic and mounted on the printed circuit board. The outputs of the ADCs will be buffered before being transferred to the VME data acquisition system via a fast digital link.

To avoid developing sparse data scan logic specifically for the test beam, one can take advantage of the power of the front end processor and the high system bandwidth and implement software zero suppression. Hence all of the data from the silicon counters will be read out for each trigger. There are 16x64=1024 channels per board and 10-bit digitisation will be used. To compensate for the lack of time correlation between the arrival of beam particles and the clocking of the analog pipelines, 3 or 4 consecutive time slices will presumably have to be read to ensure that all of the charge is recorded. Hence there will be up to 4x1024x10=40kbits per event before data compaction. If one aims at a trigger rate of 100-200Hz, then the bandwidth needed per board is up to about 8Mbits per second.

To supply this bandwidth the use of digital fibre optic links is planned at the moment. The receiver end of the link would then be a VME module with a suitable buffer memory. In the final detector one read-out fibre per board is expected, but for the test beam it may be more cost effective to have a single fibre, since the total bandwidth needed (i.e. 6x8=48Mbits per second) is less than

the available bandwidth from many commercial systems. Investigations of the availability and cost of various VME based fibre optic I/O modules are under way. The UA1 experiment has already developed a link which seems very well matched to the SITP needs, and also a number of commercial options exist.

Once in the VME module the data would be compacted by the zero suppression routine running on the front-end processor before being written to tape. It may also be interesting to investigate various 2nd level trigger algorithms in conjunction with data from the calorimeter, since this is where we expect the SITP data to be used in the trigger of a real LHC experiment.

The general DAQ and trigger structure is illustrated in figure 3b.

4.2. Event building studies

Of the various aspects of an event builder for a high rate experiment we have started by focussing our attention on the communication channels. The event building studies will center around two activities: simulation and testing components in a prototype system.

4.2.1. Simulation

The following aspects have to be addressed:

- the influence on the event builder architecture of external conditions such as the number of channels needed per event, the amount of information per channel, the local and global delay of the second level trigger, and the number of processors participating in the third level trigger
- how the standard protocols (SCI, HIPPI, etc.), already defined, which take into account the very high speed channel characteristics, can meet the external requirements listed above. In particular, we intend to study how SCI and HIPPI can be used in a fast trigger system. To this extent we propose to use VERILOG to simulate the physical channels implementing the SCI and HIPPI protocols and to evaluate the overall performance by testing the software interfaces between RISC processors and the standard channels.
- the possibility of defining a new special protocol dedicated to event building, which should take into account the fact that an event builder only needs an almost unidirectional channel and better optimisation is achieved by separating the data and control channels.

4.2.2. System prototyping

Two approaches will be pursued.

First, we plan to build on the experience gained in the recent INFN STARNET experiment [22] concerning the realization of a fibre optic network with a passive star topology. This activity stimulated the development of a high frequency device laboratory located in the Physics Department of the Rome

"Tor Vergata" University, where instrumentation required to test optical and electrical GHz components is available.

The event building studies will be done in close collaboration with a newly funded INFN project (STARDAQ), which is now testing switch matrices realized in GaAs technology with 2 ns switching time. This test will allow us to study the use of the switch matrices as an interesting alternative to passive star couplers. A GaAs transceiver gate-array for optical communications is currently under development. This gate-array allows transmission of 32 or 39 parallel bits per word. It could be used to implement control and/or data channels.

The second approach, shown in Fig. 4, is the integration of a HIPPI based system, currently being developed by R. McLaren and E. Van der Bij and also applied in read out systems for DRDC/P12. The integration of this system will require that each VME Detector crate will be equipped with an intelligent high speed link interface (RHS). The interface will collect the event data from other modules in the VME Detector crate, add header information and transmit the event data over the high speed link. The Event Builder Switch (EBS) will keep a list of "free" Destination modules. Upon receipt of event data from a Detector crate, the EBS will assign a Destination to that event. Any subsequent information pertaining to that event will then be routed by the switch to the same Destination. Once the Destination (RHD) has received the complete event, the on-board processor begins analysis. Only "good" events are then transmitted via the SVIC to the Host.

For the HIPPI implementation, it is planned to use two commercially available VMEbus modules from CES. The two boards both contain the same MIPS R3051 RISC processor system and VMEbus interface and differ only in that one is a HIPPI Source (RHS) whilst the other is a HIPPI Destination (RHD). For the Event Builder Switch, it may be possible to use the HIPPI switch from Network Systems Corporation. If this does not provide the required functionality an in-house design may be required.

Further to the two approaches described, the possibility of using the SCI for non-traditional event building can be envisaged. In such a scheme, only the fraction of data needed for the higher level selection are accessed thanks to the SCI protocol: the virtual memory scheme and the SCI cache coherence can provide a synchronisation mechanism between the local memories and the higher level processor, thus simplifying the event building process [23].

4.3. Other examples of applications

The further applications envisaged for the proposed system are concerned with phase 2 and after. We do not have any formal engagement for the following activities and our contacts with the relevant people have, at this stage, only concerned the investigation of technical details involved in a possible future integration. They should rather be considered as non-exclusive examples of

integration of Trigger/DAQ sub-system developments, which we intend to perform in collaboration with the relevant people.

4.3.1 Integration of specific sub-systems.

We have been contacted by groups involved in specialised development who foresee the need to study integration of their developments after the first test bench work. Amongst these are:

- 'Studies in Fast Calorimeter Trigger Design', by N. Ellis et al., funded in UK [24] (see letter from N. Ellis and J. Garvey to Prof. Iarocci on November 2nd 1990).

- 'A neural trigger for experimental HEP' [25], development proposed at the Niels Bohr Institute for the use of commercially available chips as a neural coprocessor for a transputer based trigger system.

4.3.2. Integration of 'level 2' trigger techniques

Apart from the ones studied in [26] and already discussed in the proposal, we have investigated the technical aspects of the use of the Fast Digital Parallel Processing Modules (FDPP) [27] to explore a flexible parallel architecture.

The FDPP is a modular system in which each node consists of one Digital Signal Processor (DSP) tightly coupled to one transputer, housed on a VME mother board. Any number of nodes can be interconnected using the four serial links of the transputer, while any data source can communicate in a parallel manner to the DSP or directly to the node memory. Such a parallel architecture then provides a MIMD parallel system loosely coupled via both serial (1.2 Mbyte/s) and parallel (up to 150Mbyte/s) data exchange, and with the possibility of dynamically changing the topology of the system (e.g. from a tree structure to a mesh structure).

A parallel processing system of this kind allows an easy distribution of processes over several processors and, at the same time, the concentration of processing power in critical areas of the detector. Algorithms developed on the DSP and transputer can be run for real time analysis with data loading via the fast parallel I/O channel as well as for offline analysis (debugging) with data loaded via the transputer serial links. Two environments are available for the development of the software, the OCCAM toolset for the transputer and cross software tools for the DSP32C running under several operating systems (UNIX included). Application programs can be loaded from the host computer or stored in a ROMTRAM from INMOS and automatically loaded into the DSP and the transputer at power on.

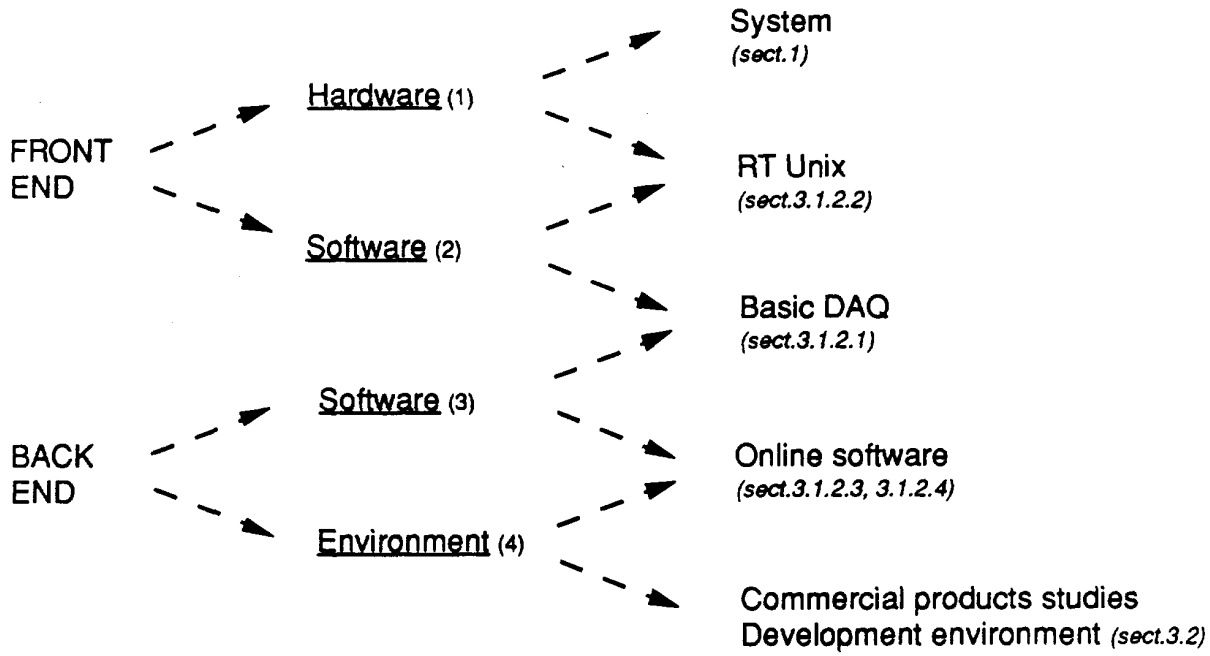
5. SUMMARY OF PHASE 1 ACTIVITIES

5.1 Work Plan

The project activities during the first year (phase 1) are subdivided into three main areas:

• **Interface to detectors.** (R. Bonino, R. Ferrari, M.Pentney).
This involves keeping in contact with the detector projects, relaying their requirements for the 1992 data taking and assuring successful 1991 data taking with the existing H2 data acquisition.

• **Phase 1 system implementation.** (see list of names below).
The activities are described in chapters 1 and 3 above. The work plan in this area can be best described by the following graph:



(1) M. Coret, A. Lanza, J. Pain, M. Pentney, G. Polesello.
(2) M. Aguer, C. Bee, J.L. Fallou, G. Fumagalli, M. Huet, G. Mornacchi, S. Stapnes.
(3) M. Aguer, G. Ambrosini, R. Bonino, T. Choulette, G. Fumagalli, M. Huet, L. Mapelli, G. Mornacchi, F. Pastore
(4) R. Ferrari, S. Hellman, R. Jones, J.R. Poggioli, N. Zaganidis.

• **Event Building Activities.** (R. Cardarelli, M.L. Ferrer, G. Mirabelli).
This is done by the Frascati-Rome group, in collaboration with the relevant part of P12, as described in section 4.2.

5.2 Collaboration with industry and other groups

The use of new products from CES (SBUS-VIC interface, RAID running LynxOS) is accompanied by an offer of support and collaboration with the common objective of evaluating the use of modern hardware (a RISC processor) and software (real-time UNIX) technology in a real-time application. This offer, which is not limited to the implementation of the phase 1 system, consists of:

- the transfer, free of charge, to our project of all software used and developed by CES relevant to the hardware environment
- the availability of a contact person within CES for the software aspects of the collaboration
- the continuation of the collaboration in view of technological upgrades (e.g. Futurebus+ based developments).

As mentioned in the proposal, the development of software common to DRDC/P12 [26] and DRDC/P15 [28] will be organised jointly. We have initiated discussions to establish a 'pseudo-formal' collaboration and we intend to propose the creation of an inter-project group, with members from the three proposals, for the development of specific common software.

References

[1] L. Mapelli et al. A scalable data taking system at a test beam for LHC, CERN/DRDC/90-64, DRDC/P16.

L. Mapelli et al. Addendum to DRDC/P16, CERN/DRDC/91-1.

[2] T. Williams. Unix gets groomed for real time duty. Computer Design, June 1st,1990.

[3] REAL/IX, concepts and characteristics. AEG-Modcomp. Order no. 205-855001-001.

[4] Lynx to open systems. Information Week, June 12, 1990..

[5] J.F. Muratore et al. Real time data acquisition at mission control. CACM, Vol. 33, No. 12, Dec 1990.

[6] M. Rozier et al. Overview of the Chorus distributed operating system. CS/TR-90-25, Chorus Systemes.

[7] Dune 3000 / Dune-IX. IX-Magazine, Vol. 6, No. 40, Feb. 1991.

[8] R.D. Russel, G. Mornacchi. VOS - A Virtual Operating System. CERN-ECP/DS unpublished.

[9] W. Bozzoli, P. Vande Vyvre. Geode: An evaluation. CERN/ECP-EVAI 91-1.

[10] A. Goldberg. Smalltalk 80: the language and its implementation. Addison-Wesley, 1983.

[11] B. Stroustrup. The C++ programming language: reference manual. Bell Labs, Jan. 1984.

[12] B.J. Cox. Object Oriented Programming: an evolutionary approach. Addison-Wesley, 1983.

[13] S.E. Keene. Object oriented Programming in Common LISP: a programmer's guide to CLOS. Addison-Wesley, 1989.

[14] B. Meyer. Object Oriented Software Construction. Prenctice-Hall, 1988.

J.P. Matheys. Fifth International Eiffel user group conference. CERN ECP/CONF 90-3.

[15] G. Booch. Object Oriented Development. IEEE Tans. Software Eng. SE-12, No. 12, Dec. 1986, 211-221.

[16] HOOD reference Manual, Issue 3.0. HOOD working group, ESTEC, Sep. 1989.

[17] Saber-C++ Reference. Saber Software Inc.

[18] R.W. Schifler, J. Gettys. The X Window System. ACM Trans. on Comp. Graphics, 5, (1987), 79.

[19] OSF/Motif Programmer's guide, revision 1.1. Open Software Foundation.

[20] F.Di Maio et al. Report of the study group for the selection of a software package for synoptics. CERN/PS/CO 91-02.

[21] SITP Collaboration. A Proposal to Study a Tracking/Preshower Detector for the LHC. CERN/DRDC/P3.

[22] P. Bacilieri et al. STARNET, a Fiber Optic Metropolitan Area Network with Centralized Control. International Conference on Computing on High Energy Physics, Oxford 12-14 April (1989), Computer Physics Communications 57 (1989) 459-465.

P. Bacilieri et al. STARNET, A Fiber Optic MAN with Star Topology. Conference EFOC/LAN, Amsterdam June 12-16, 139(1989)

P. Bacilieri et al. STARNET, a high speed fiber optical network for particle physics application. NIM, Proceedings of the 4th Pisa meeting on advanced detectors, Isola d'Elba

[23] J.F. Renardy et al. SCI at LHC Large Hadron Collider Workshop, G. Jarlskog and D. Rein eds., Aachen 4-9 October 1990, p. 165.

[24] N. Ellis et al. Studies in Fast Calorimeter Trigger Design. Proposal approved by the UK selection panel.

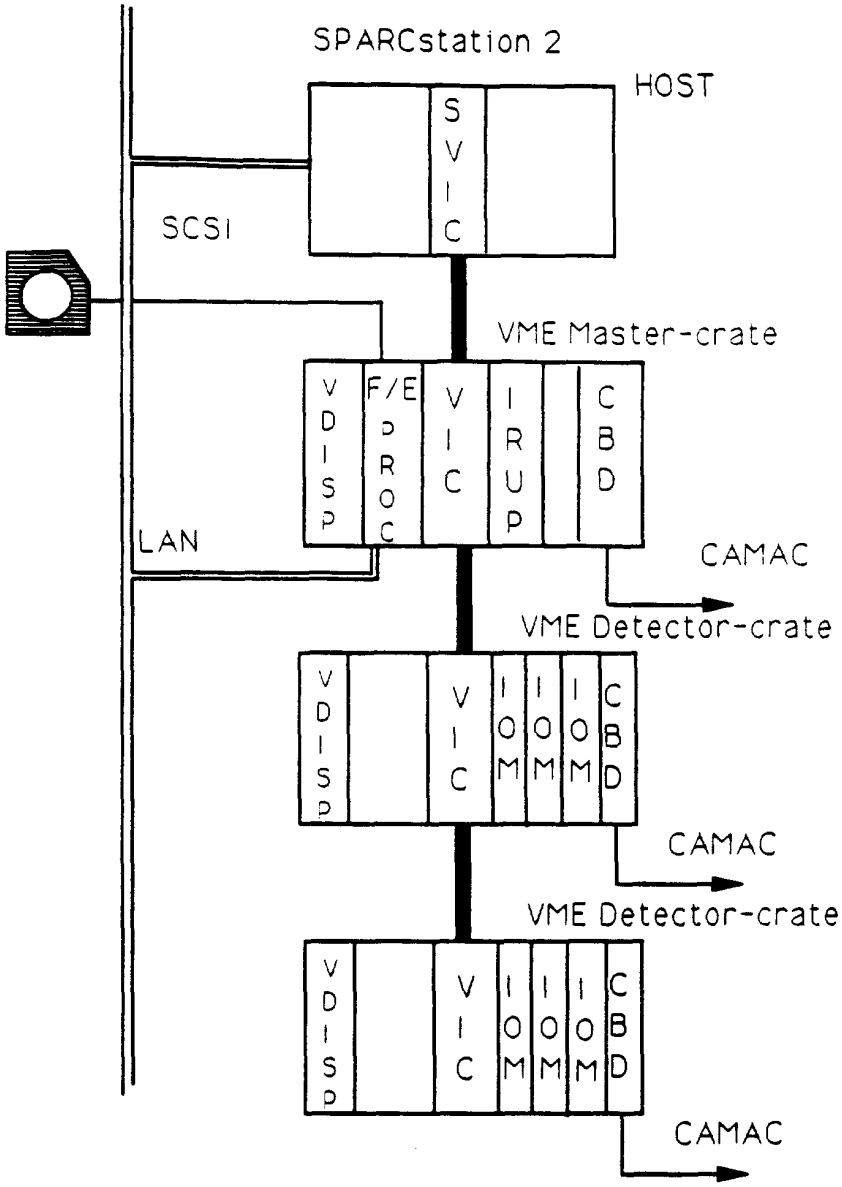
[25] J.R. Hansen. Private communication.

[26] R. Bock et al. Embedded Architectures for Second-level Triggering in LHC Experiments. CERN/DRDC/P12.

[27] D. Crosetto.Fast Digital Parallel Processing module FDPP. CERN-DD/89-33 AC.

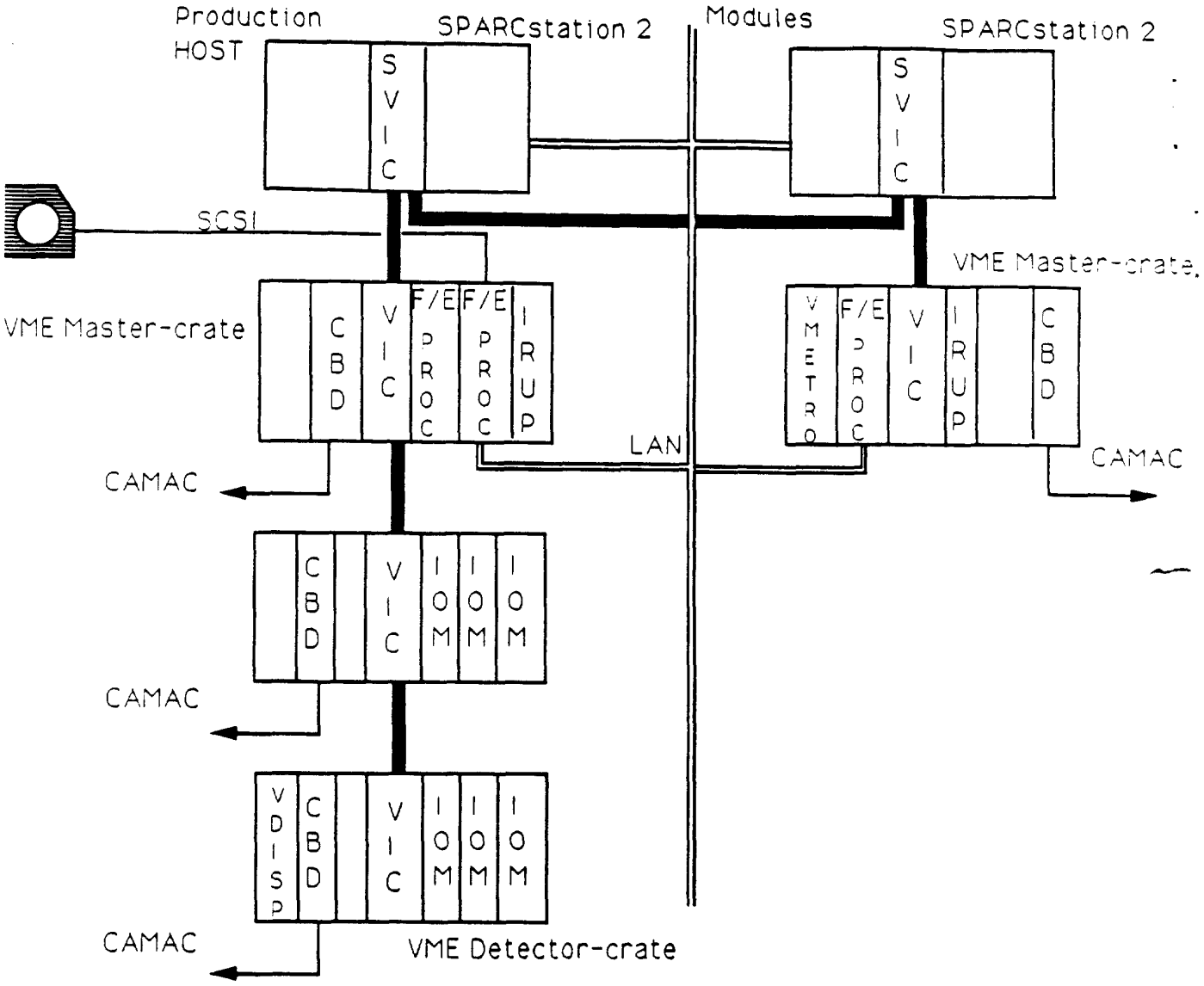
D. Crosetto. FDPP applications. Application on Transputers 2, D.J. Princhard and C.J. Scott (eds.), Southampton, IOS Press, Amsterdam (1990).

[28] S. Cittolin et al. Read Out System Test Benches. CERN/DRDC/P15.



Modules	
SVIC	VIC/Sbus interface, CES SVIC7213
VIC	VMV Master/slave interface, CES VIC8250/1
F/E PROC	MIPS3000 based processor, CES RAID8235
VDISP	Display module, CES VMDIS 8003A
CBD	CAMAC branch driver, CES CBD8210
IRUPT	Interrup generator board, ELTEC V-IGEN-A100
IOM	Detector I/O Modules

Fig. 1. **Phase 1 system**



Modules	
SVIC	VIC/Sbus interface, CES SVIC7213
VIC	VMV Master/slave interface, CES VIC8250/1
F/E PROC	MIPS3000 based processor, CES RAID8235
VDISP	Display module, CES VMDIS 8003A
VMETRO	VMETRO Display/Logic State Analyser Module
CBD	CAMAC branch driver, CES CBD8210
IRUPT	Interrup generator board, ELTEC V-IGEN-A100
IOM	Detector I/O Modules

Fig. 2 Phase 2 system

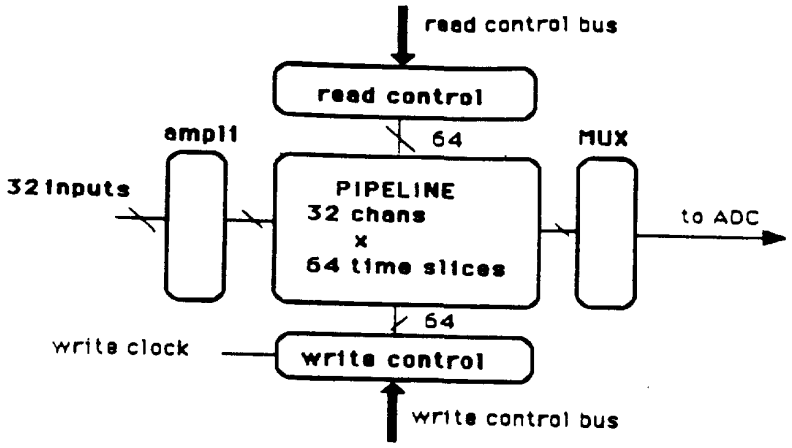


Fig. 3a. SITP front end circuit.

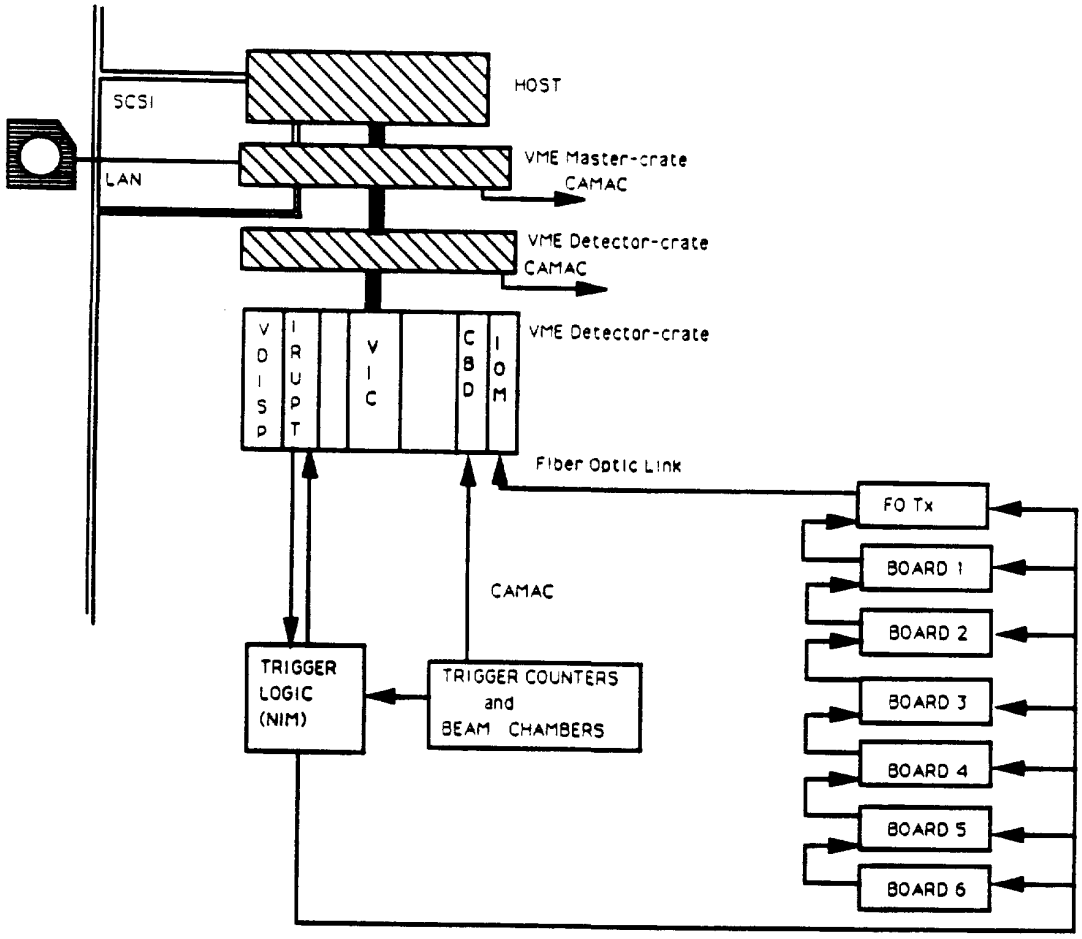
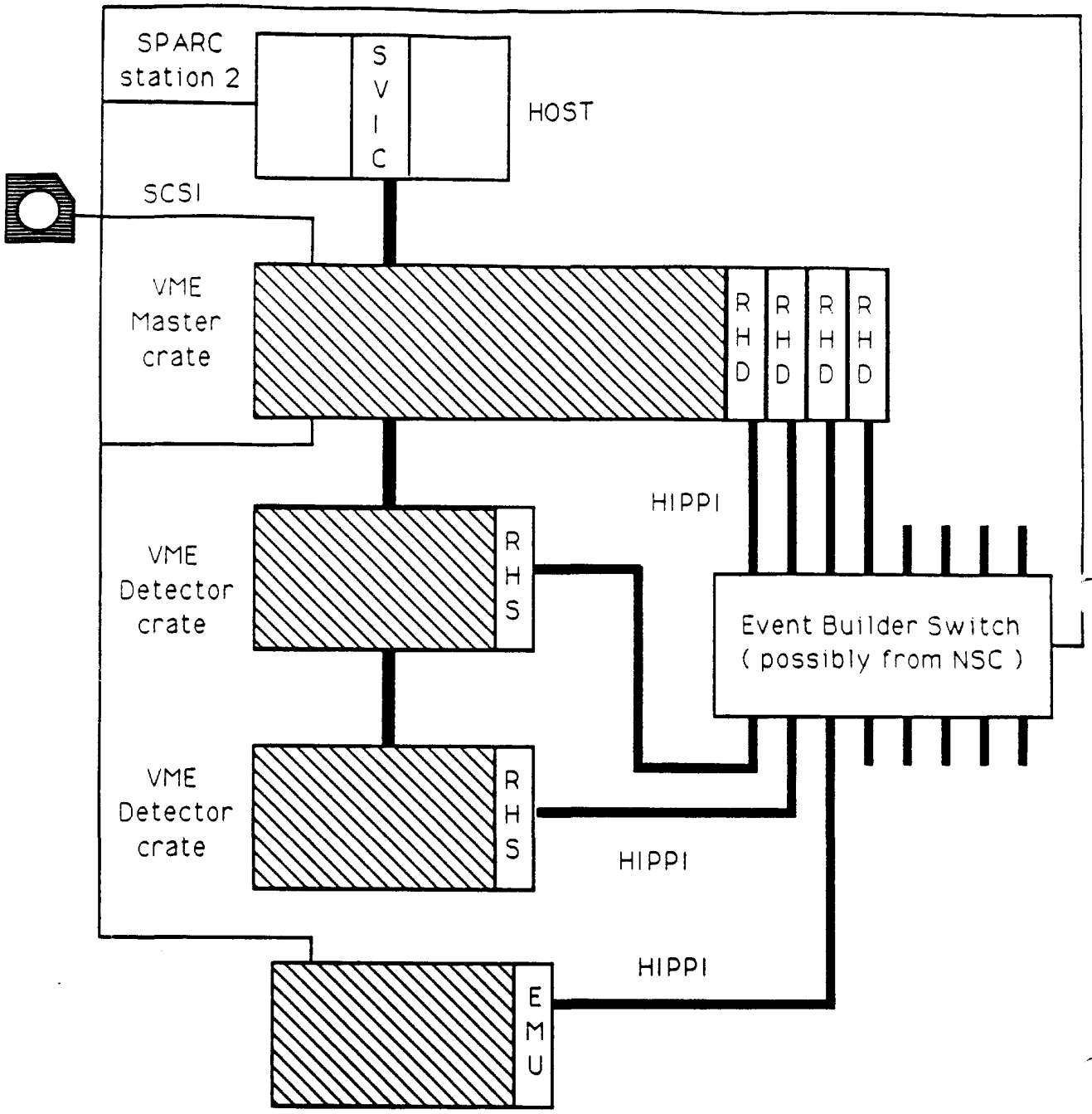


Fig. 3b Integration of SITP readout and trigger



Modules	
RHS	RISC I/O Processor with HIPPI Source CES RIOP 8260/HS
RHD	RISC I/O Processor with HIPPI Destination CES RIOP 8260/HD
EMU	VMEbus based "dummy detector" (under development in DRDC/P12)

Fig. 4 Integration of a HIPPI based event builder