1 Introduction

A previous document (*Eurogam Project—System Overview)* divides the Data Acquisition System into a number of logical components. This document takes that concept and proceeds to describe the components in terms of real physical elements.

2 System infrastructure and communications

The Data Acquisition System for Eurogam may be viewed as a distributed computing system having a number of discrete and identifiable processing elements. Apart from the more obvious elements such as workstations which provide the primary user interface into the system there are a number of data collection (such as the GE channel readout), data processing (such as the event builder) and equipment control (such as the HV unit and auto-fill system) elements which are built up from modules in either VME or VXI crates. These elements will be connected together to form a co-operating system by a broadcast communications network (Ethernet initially and later FDDI) which will be used for all control and monitoring functions during normal operation. In the first phase (at least) separate data paths will be provided for movement of event-by-event data but these will be co-ordinated by and form an integral part of the overall system.

A diagnostic port will be provided into each processing element (in addition to any 'on-line' diagnostics provided by the main system) but these will live outside the distributed environment.

Each processing element must support the communications functions required by the distributed environment. In order to implement this each VME or VXI crate forming a discrete processing element (and this probably means all crates) will have a **Crate Controller** which must also in the case of the VXI crates incorporate the required **Resource Manager** functions (see separate document). The crate controller will be based on a single VME board computer (the Motorola MVME 147 which has a 68030 processor running at 20 MHz and 4 Mbytes of integral RAM) having integral Ethernet, SCSI and RS232 ports running under a multi-tasking operating system (OS9). The SCSI port allows a small (40 Mbyte) 3.5 inch hard disc to be attached which can hold production software and support (using the RS232 ports if required) a comprehensive diagnostic environment. Current technology allows these discs to be treated as exchangeable and are hence almost as convenient as floppy discs but offer much greater reliability (5 year MTBF) and capacity at little extra cost.

Ethernet communications will be based an a low overhead datagram protocol (Type 1 LLC as specified by ISO 8802.2). This supports up to 256 ports at source and destination station and provides the length of the data following in the frame (needed since Ethernet must always transmit at least 60 bytes). Other communications media will use similar protocols as required to provide the same functionality.

Software running in each crate controller will provide a common communications interface to the applications level software independent of the communications hardware. Each processing element will provide one (or possibly more) resource(s) to the distributed environment. Each resource will be managed by a software module acting as a **server** for the resource. All applications software (**client**s) wishing to access such resources must communicate with the resource server.

Communications between all elements of the data acquisition system will follow this strict client/server model. The document *Eurogam Project - System Overview* describes in general terms the control paths between the logical components of the system. Each of these logical components contains invariably both a server and a client. Clients always initiate an activity and servers respond. For example: the event processor unit is a server to the event builder unit which in its turn is a client of the data storage unit.

The software applications will be built using a technique known as **Remote Procedure Call (RPC)** which is the logical extension into a distributed computing environment of a local procedure (ie subroutine or subprogram). In exactly the same way as a local procedure is called by supplying a procedure name and some arguments then a call to a remote procedure must supply the same information. This will be passed by the RPC support software to the appropriate remote server where the request will be executed and the results returned to the client. A RPC support library will be provided to aid generation of applications using RPC techniques.

The software system for Eurogam data acquisition is intended to be as flexible as is practical in order to place no unnecessary constraints on its users. For example: it is expected that more than one graphics station will concurrently wish to view experimental data. Also control of an experiment comes from the user via the graphics stations while the data comes from the front-end hardware. Conventional point to point links force the use of connection orientated network protocols and invariably require that control and data follow the same paths into the host (server). Using a broadcast network and RPC with a datagram (connectionless) network protocol places no such constraints on communication between clients and servers. This both simplifies the design of applications software and allows maximum advantage to be taken of the distributed system.

One problem which must be addressed by any distributed system adopting this approach is the question of access control and authentication. A general solution used by many distributed systems is the concept of a **Capability**. Before a client program can access a resource such as a front-end crate or a tape drive it must first obtain a capability for the resource from the server program which guards access to that resource. A capability is a 32 or 64 bit datum obtained by encrypting the server's own internal identification for the resource with a randomly chosen key. This makes it unlikely that capabilities for resources can be forged. Subsequent requests from the client to access the resource must be accompanied by a copy of the capability else the server refuses to grant access.

Access to a particular resource is controlled solely by this capability rather than other more conventional information such as the client's name, network access port or network address. This method allows a number of cooperating client programs, such as a component of the experimental monitoring software, to access a particular resource, such as a histogram held by the event processor, as long as the programs share knowledge of the capability.

When a resource is finally freed the server invalidates the capability by changing the encryption key associated with the resource thus rendering all copies of the capability **stale**.

3 Acquisition Control - D Brightly

This section presents an overview of the Eurogam software architecture. Because of the distributed nature of the system we have adopted the 'client-server' model—users make things happen by invoking application programs (the clients) which make requests on server programs to execute the necessary functions.

Server programs have an important 'information-hiding' role. Requests from client programs to servers are presented in terms of abstractions such as 'spectrum' and 'channel' rather than in specific hardware terms. The server programs elaborate these abstract functions into sequences of operations appropriate to the hardware in which they are embedded.

Server programs are also responsible for maintaining all of the 'state' of an experiment—what spectra exist, what Event-by-Event files are open, what acquisition channels are enabled, and so on. By removing state from the client side we can allow several clients (i.e., user interfaces) access to a single experiment. All information about the experiment needed by a client is obtained from the appropriate server program(s) by means of an Inquiry function.

In many cases, server programs will be non-trivial pieces of software, quite possibly further broken down into multiple subprocesses, and they must run in an environment which forms part of the distributed system. We thus require all server programs to be supported by a suitable multi-tasking operating system (OS9). The server program may run in the processor which provides the crate controller functions or in a separate compatible processor.

We use the concept of Capability as described previously to control access to the resources of the system. Each of the servers outlined below supports a pair of ClaimResource and FreeResource functions whereby capabilities are dispensed and revoked. Once in possession of a capability a client has full access rights over the resource. Capabilities may be stored in files and may be passed if desired between distinct clients so that a resource claimed by one client may also be accessed by another. This provides the basis for several users to access the resources of a single experiment (and potentially interfere with one another!)

The remainder of this section outlines the functions offered by each of the principal server types. It should be stressed that the lists are not complete but are intended to give an indication of the functions provided. Complete lists will be found in the functional specification documentation for each server. The notation used is derived from the **Z** specification language. We give just the 'signature' of each function, namely its arguments and their data types. By convention, input and output argument names end in '?' and '!' respectively. To keep the descriptions short we have omitted further details of argument data types. We have also left out the capability arguments (which are always required). This information will be provided in the functional specification document for each server.

3.1 Frontend Crate Servers

These include servers for crates containing GE cards, BG cards, histogrammers, trigger processor, event builder, and interfaces to apparatus control systems.

These servers offer a uniform client interface presented in terms of named abstract registers. For example, a VXI crate containing GE cards might support a register named 'G23.PoleZero', denoting the polezero adjuster on GE channel 23. The name space is constructed by the server using information derived from a configuration file supplied when the crate is claimed. This static configuration data can be correlated with information derived from auto-configuration procedures run at crate power up time to reveal broken or missing modules. Essentially, the register name space is determined by the numbers and types of modules present in the crate. Since we expect register names to appear in the user interface comments on a suitable naming convention are invited. The functions supported by these servers are:

- ClaimCrate == [crate?:ResourceName; cap!:Capability; status!:Report]
- FreeCrate == [crate?:ResourceName; cap?:Capability; status!:Report]

and

- InitialiseReg == [reg?:Register; status!:Report]
- ReadReg == [reg?:Register; val!:Value; status!:Report]
- WriteReg == [reg?:Register; val?:Value; status!:Report]
- InitialiseRegs == [pat?:Pattern; status!:Report]
- ReadRegs == [pat?:Pattern; rvlist!:seq (Register,Value); status!:Report]
- WriteRegs == [pat?:Pattern; val?:Value; status!:Report]
- InquireRegs == [pat?:Pattern; rlist!:seq Register; status!:Report]

The *Regs functions address registers by pattern matching on their names, using patterns based on regular expressions. The InquireRegs function returns a list of those register names matching a pattern and enables a client program to build a mapping between register names and the crates where they reside.

With these primitives, a user interface can offer two functions for saving and restoring complete experimental configurations:

- SaveSetup == [file!:RegisterValueDataBase; status!:Report]
- RestoreSetup == [file?:RegisterValueDataBase; status!:Report]

In principle, all features of the frontend hardware can be accessed by these functions but in addition higher level functions are provided to support singles spectra and control of Event-by-Event data flow:

- InquireSpectra == [pat?:Pattern; slist!:seq Spectrum; status!:Report]
- InquireAttributes == [s?:Spectrum; attribs!:SpectrumAttributes; status!:Report]
- ReadSpectrum == [s?:Spectrum; dom?:Domain; counts!:seq Int; status!:Report]

- WriteSpectrum == [s?:Spectrum; dom?:Domain; counts?:seq Int; status!:Report]
- ZeroSpectrum == [s?:Spectrum; status!:Report]
- ReadAnnotation == [s?:Spectrum; anno?:Int; val!:seq Byte; status!:Report]
- WriteAnnotation == [s?:Spectrum; anno?:Int; val?:seq Byte; status!:Report]
- CreateSpectrum == [s?:Spectrum; attribs?:SpectrumAttributes; status!:Report]
- DeleteSpectrum == [s?:Spectrum; status!:Report]

and

- Go == [status!:Report]
- Halt == [status!:Report]

The spectrum functions are further described in the Event Processor Server section.

3.2 Event Processor Server

We view the Event-by-Event data path as a pipeline effectively starting at the **Trigger Unit** (TU), passing through an **Event Builder** unit (EB), and continuing to an **Event Processor** unit (EP), where it may split into several parallel pipelines, terminating in **Event Storage** (ES) units. Since both the EB and EP are capable of transforming the Event-by-Event data under the control of a downloaded program, we treat them both as logical event processor units. We intend to construct the software supporting Event-by-Event data paths to run autonomously without the need for explicit go and halt operations directed to individual units, since this gives rise to synchronising problems. We expect data flow to be controlled by a single enabling register in the trigger unit which inserts START and STOP tokens into the pipeline. All pipeline stages propagate these tokens and may take local action (e.g., change state) if necessary.

Both EB and EP must be capable of accepting a downloaded event processing algorithm. Typically this is code cross-compiled for a specific event processing platform by utilities running in the user interface. This code is dynamically linked into the environment provided by the event processing unit. The environment supports the input and output of events to the downloaded code and addressing mechanisms for memory resident spectra, for example.

Experience at the NSF has shown the value of a facility for making incremental changes to the event sorting procedure which avoids the need to recompile, link, and download a complete program. Such a facility is convenient for the majority of small changes to sorting parameters. We have generalised this idea to the concept of a ‘sort object’— essentially a named and typed piece of data that can be independently constructed by a user interface program and downloaded to an event processor. Typical objects include simple types such as integer bit masks and floating point constants, as well as composite types such as the set of coefficients of a gain matching polynomial and sets of one and two dimensional polygons representing sorting ‘windows’.

We also recognise the need to selectively enable the routing of processed events onto any of several event storage streams, identified by small integers.

In summary, the set of functions recognised by event processing servers is:

- ClaimEbyEProcessor == [proc?:ResourceName; cap!:Capability; status!:Report]
- FreeEbyEProcessor == [proc?:ResourceName; cap?:Capability; status!:Report]

and

- HaltProgram == [status!:Report]
- GoProgram == [status!:Report]
- DeleteProgram == [status!:Report]
- LoadProgram == [prog?:CompiledProgram; status!:Report]

and

- InquireObjects == [pat?:Pattern; objlist!:seq Object; status!:Report]
- InquireAttributes == [obj?:Object; attribs!:ObjectAttributes; status!:Report]
- CreateObject == [obj?:Object; attribs?:ObjectAttributes; status!:Report]
- DeleteObject == [obj?:Object; status!:Report]
- ReadObject == [obj?:Object; val!:ObjectValue; status!:Report]
- WriteObject == [obj?:Object; val?:ObjectValue; status!:Report]
- ReadObjects == [pat?:Pattern; objlist!:seq (Object,ObjectAttributes,ObjectValue); status!:Report]
- WriteObjects == [pat?:Pattern; val?:ObjectValue; status!:Report]

end

- AssociateStream == [str?:Stream; f?:FileHandle; cap?:Capability; status!:Report]
- DissociateStream == [str?:Stream; status!:Report]
- EnableStream == [str?:Stream; status!:Report]
- DisableStream == [str?:Stream; status!:Report]

In addition, the following functions support access to histograms generated by event processing:

- InquireSpectra == [pat?:Pattern; slist!:seq Spectrum; status!:Report]

- InquireAttributes == [s?:Spectrum; attribs!:SpectrumAttributes; status!:Report]
- ReadSpectrum == [s?:Spectrum; dom?:Domain; counts!:seq Int; status!:Report]
- WriteSpectrum == [s?:Spectrum; dom?:Domain; counts?:seq Int; status!:Report]
- ZeroSpectrum == [s?:Spectrum; status!:Report]
- ReadSpectrumProjected == [s?:Spectrum; poly?:Polygon; dom?:Domain; counts!:seq Int; status!:Report]
- ReadAnnotation == [s?:Spectrum; anno?: Int; val!:seq Byte; status!:Report]
- WriteAnnotation == [s?:Spectrum; anno?: Int; val?:seq Byte; status!:Report]
- CreateSpectrum == [s?:Spectrum; attribs?:SpectrumAttributes; status!:Report]
- DeleteSpectrum == [s?:Spectrum; status!:Report]

The ReadSpectrum and WriteSpectrum functions provide access to blocks of contiguous channels or 'Domains', ie intervals [n..m] in one dimensional spectra and rectangles [n..m]×[p..q] in two dimensional spectra. The ReadSpectrumProjected function provides an efficient way of extracting X and Y projections of a two dimensional spectrum above a polygonal window—a client program does not have to read a large array of counts in order to obtain such projections. Simple one dimensional cuts are a special case of this function. Spectrum 'annotations' are pieces of opaque data identified by small integers that the server can associate with a spectrum. They are typically used to attach titles and calibrations to spectra.

3.3 Event Storage Server

This server is the final stage in an Event-by-Event pipeline. The server can claim exclusive use of individual drives, mount and dismount labelled media, open and close data files, and read and write data records:

- ClaimDrive [d?:Drive; cap!:Capability; status!:Report]
- FreeDrive [d?:Drive; cap?:Capability; status!:Report]
- MountVolume [d?:Drive; volume?:Name; status!:Report]
- DismountVolume == [d?:Drive; status!:Report]
- OpenFile == [d?:Drive; file?:Name; mode?:Mode; f!:FileHandle; status!:Report]
- CloseFile == [f?:FileHandle; status!:Report]
- ReadFile == [f?:FileHandle; rec!:Record; status!:Report]
- WriteFile == [f?:FileHandle; rec?:Record; status!:Report]

- PositionFile == [f?:FileHandle; offset?:Int; status!:Report]

The FileHandle data type returned by OpenFile is an identifier for a file open on some drive. Note how a FileHandle obtained by a client program from a storage server is passed to an event processor via an AssociateStream request. The event processor also needs to know the capability for the corresponding drive before it can write data successfully.

4 The Event Processor - J Cresswell

This section provides a description of the features of the Sorter. This is an Event Processor sub-system in the Eurogam data acquisition architecture. To simplify the discussions it is assumed that the following refers to the implementation for Daresbury Laboratory. It is possible that a further development may be necessary to support the complete array at Strasbourg.

Input will consist of blocks of event data from the Event Builder module. The processing element will be based on the current offline Sort Engine and language as currently in use within the UK. Output will consist of blocks of event data for writing to the storage medium.

4.1 Requirements

The Project Scientist Committee has asked for the system to be able to cope with a minimum of 200 KWords/second, with feedback required on the possibility of upgrading to 2 MWords/second.

Other requirements that have been assumed include ease of programming and production phase maintainability. All components suggested are readily available from stable and internationally known companies.

4.2 Background

The Sorter lies in the chain of event data flow between the Event Builder and Data Storage. Some of the decisions involve consideration of the Sorter and Event Builder together.

The sorting language currently in use in the UK will be used to provide a user friendly interface for the physicist. As processing systems get more complicated with the use of multi-processing techniques, it is thought that user-written Fortran sorting programs will become more and more inappropriate. The advantage of a high level sorting language lies in its ease of use and computer-independence.

4.3 Components

It is proposed that the Sorter consist of boards contained in one VME crate. The current offline Sort Engine provides the fundamental design basis of using a VME crate and Motorola 68K series processors.

The crate will contain a standard Crate Controller module providing the necessary interface to the distributed system as described earlier in this document. The interface between the Event Builder and Sorter will be a fibre optic point-to-point link. There are two versions readily available in the catalogues of Struck and CES. Both have maximum quoted rates of 6 Mbytes/second, and both have VSB interfaces. On first sight either product would be satisfactory and some further research will be necessary to make a choice.

The interface for tape storage at Daresbury will be a parallel link a GEC 4100 system. The board in current use (Struck 302) will not be good enough for Eurogam purposes and it proposed to build a more suitable version. This will have a VSB interface to remove the transfer load from VMEbus. The exact proposals will be described elsewhere.

The data processing cpus have to be Motorola 68K series cpus to be compatible with current Sort Engine software. The Motorola 165 (25 or 33MHz 68040 with 4Mbytes memory) seems a good choice as a general processing engine with sufficient performance. It has been designed specifically for embedded controller functions. The onboard memory is ported to both VMEbus and VSBbus. This board is orderable now with production availability in Q3 1990. The 68040 has a quasi-risc design which provides high processing power whilst keeping the powerful instruction set of the 68K series.

The final component of the sort system is the spectrum memory. This memory will be VME accessible by all processing cpus. The memory boards used will be standard ones available from several manufacturers. It is important that they be configurable for extended addressing mode only (A24 must be disabled). The system is proposed to have 64 Mbytes enabling 2D matrices to be updated online.

4.4 Data Rates

The input data rate is limited by the bandwidth of the optical fibre link from the Event Builder to a maximum of 6 Mbytes/second.

It is difficult to estimate the processing power required for sorting. One MVME165 may be sufficiently powerful to cope with the minimum requested 200 KWords/second. Since the system is modular and more cpus may be added with no change of software, there is an easy upgrade path.

However, other criteria affect the performance. The data output path is currently via the Struck 302 parallel interface into a GEC4190. This parallel interface is only accessible over VMEbus with a limitation of 16bit wide data. This will have to be replaced, or a severe bandwidth limitation will occur on VMEbus. It is proposed to build a more efficient interface accepting 32bit wide block transfers over VSBbus. The GEC interface can accept data at up to 4 Mbytes/sec.

Spectrum updates will occur over VMEbus to globally accessible memory. Current experience places a limit approaching 1 Mupdates/second assuming there is no other use of VMEbus. This cannot be the case as spectrum reads for display purposes and local slicing will expect some reasonable response.

It should be noted that high fold events contain many combinations of doubles and triples. Hence the volume of spectrum updates will increase considerably over current levels. This is likely to become the major bottleneck to future expansions.

4.5 Modus Operandi

Data will arrive in the fibre optic controller and be transferred over VSBbus to one of several sorting processors. Data blocks to be output to Storage will be transferred over VSBbus again to the GEC interface controller. As in the Event Builder design, VSB is used to modularise the system and to partition total bus bandwidth. As output to storage will necessarily be limited, then two passes over VSB should be allowable.

Thus with a 6 slot VSB backplane, two slots will be occupied by the Event Builder and Storage interfaces leaving upto four slots available for sorting processors.

This design leaves VMEbus free for spectrum access. Spectrum updates from the sorting processors will occupy most of the available VMEbus bandwidth. The Control and Setup network interface cpu will require spectrum access for read,clear, and slicing operations.

4.6 Software

A task running in the Crate Controller processor will accept control commands from the user as outlined in section 3.2.

The software running in the fibre optic controller cpu will control the link and transfer data blocks only. This will be an independent subsystem, accepting data blocks transferred from the Event Builder and initiating block transfers to one of several processors. It will receive acknowledgements from the processors, and send acknowledgements to the Event Builder. The code will be a fixed single task system, written in C or assembler.

The processing cpus for the Sorter will all execute the same code. This consists of a foundation layer allowing command and data transfer from the controller. This code will be essentially the same as running currently on the offline Sort Engine. Some changes will be necessary to accommodate the extra requirements of an online system.

On top of the foundation lies a layer of code to process an event. This code will be generated as at present, from the output of the Sort compiler. It then passes through a cross-assembler and the result downloaded into each processor. The basic system will be the same as is currently in use in the offline version. The command set that the Sort compiler understands will be enhanced to accommodate the different trigger types and event formats of Eurogam, together with the agreed requirement for multiple storage streams output from the Sorter. There will be other changes to the sort language, but these will be the subject of another document.

It is proposed to provide online matrix slicing operations locally. The code to execute the slicing operations will be part of the sort control task executing in the Crate Controller processor. The slicing operations will include horizontal, vertical, diagonal and polygonal window projections on 2D matrices.

4.7 Costs

Costs are best estimates and are quoted including VAT. The total cost of the crate and contents for the proposed system is estimated to be £K30. This estimate includes two MVME165 processors and 64 Mbytes of spectrum memory. Should further processing be necessary, two further MVME165 cpus could be added for an extra £K8. External cabling costs are not included.

5 The Event Builder - J Cresswell

This section provides a description of the features of the Event Builder data acquisition sub-system. Conceptually, this sub-system can be thought of as having an input section, a transformation section and an output section.

Input will consist of blocks of event data from each of the acquisition readout controllers. The transformation section is to some extent optional and consists of one or more cpus processing the incoming event data stream. Transformations include event formatting, gain adjustment, etc. The output will be blocks of complete events in the standard event format (see elsewhere) for transmission to the online Event Processor. It is assumed that most data transformations are to be applied in the Event Processor. The only criteria for processing data in the Event Builder is assumed to be for requirements more easily accomplished prior to event formatting, or where significant processing or data reduction of a fixed nature can be performed.

5.1 Requirements

The Project Scientists Committee has asked for the system to be able to cope with a minimum of 200 KWords/second, with feedback required on the possibility of upgrading to 2 MWords/second. For the purposes of the Event Builder, it is assumed that these defined rates apply to output and not input rates.

Other requirements that have been assumed include ease of programming and production phase maintainability. All components suggested are readily available from stable and internationally known companies.

5.2 Background

The Event Builder lies between the data acquisition hardware and the Sorting and Storage components of the whole system. It has to accept event by event data from the Readout Controllers at high rate and with little or no deadtime. It has a primary task of reorganising the format of events ready for passing forward to the Sorting and Storage components. Other transformations could be applied to the data stream at this stage and so sufficient processing power needs to be provided.

Event readout from each data acquisition crate will occur sequentially resulting in one data stream arriving at the Event Builder. This data stream will consist of sub-event contributions from each acquisition crate. All sub-event sections of one event will therefore arrive together and separated from those of other events.

The connection between the Readout Controllers and the Event Builder crate will be a 32bit FERA-like link with Strobe, Acknowledge, Busy and Readout Inhibit signals. This definition is compatible with the CES HSM8170 fast memory board with the 32bit input option.

A further requirement is that the link from the Readout Controllers to the Event Builder be as short as possible. This forces the Event Builder crate to be positioned close to the acquisition hardware. In handshake mode, any increase of the cable length will reduce the overall bandwidth. For the system envisaged this should not be a problem.

The final requirement relates to the connection between the Event Builder and the Sorting and Storage components. The latter will be placed in the Control room some 100 metres from the acquisition hardware. At Daresbury Laboratory this would mean connecting two different electrical grounds. An optical solution will surmount this difficulty.

5.3 Components

It is proposed that the Event Builder consist of boards contained in one VME crate. VME is the most sensible choice for this application as it supports the necessary processing power and data transfer bandwidth. For use at Daresbury Laboratory, the crate will conform to their standard specification.

The crate will contain a standard Crate Controller module providing the necessary interface to the distributed system as described earlier in this document.

The interface with the acquisition Readout Controllers will be the CES HSM8170 fast memory module. This module has a 10MHz 32bit FERA-like input port to fast static RAM. The module also has a VSB interface.

The hardware aspects of the link betwen the ROCs and the HSM 8170 will be the subject of another document.

The original proposal for the connection between the Event Builder crate and the Sorting and Storage components was Ethernet. This is now discarded as not having sufficient bandwidth. It is now proposed to use a fibre optic point-to-point link. There are two versions readily available in the catalogues of Struck and CES. Both have maximum quoted rates of 6 Mbytes/second, and both have VSB interfaces. On first sight either product would be satisfactory and some further research will be necessary to make a choice.

The cpus involved in the event builder will be standard VME cpu modules utilising MC68020/30/40 microprocessors. The Motorola 165 (MC68040 at around 15-20 mips) seems a good choice as a general processing engine with sufficient performance. It has been designed specifically for embedded controller functions and is provided with 4 Mbytes of memory ported to both VMEbus and VSBbus. This board is orderable now with production availability in Q3 1990. The 68040 has a quasi-risc design which provides high processing power whilst keeping the powerful instruction set of the 68K series.

5.4 Data Rates

Tests have been carried out to quantify the performance of the chosen components. As the MVME165 is not yet available a MVME147 was used for the first test. A DMA transfer initiated by the FIC8230 from HSM8170 memory via VSB to MVME147 dual-ported memory via VME occupied about 50% of VME bandwidth at 1 MWords/second rate. A similar test transferring between two HSM8170 via VSB achieved a rate of nearly 1.5 MWords/second.

The intended approach is to setup DMA transfers between HSM8170 and MVME165 via VSB. This is estimated to be nearer the case of the first test as the MVME165 dual ported memory presumably is similar to that of the MVME147. There may be some improvement due to better designed VME and VSB interface chips reportedly used in the MVME165.

A conservative approach, therefore, says that a HSM8170 and FIC8230 combination can cope with 1 MWord/second throughput.

A comparison with the existing Sort Engine can serve as a guide to the power required for event-reformatting. The current task of event unpacking is roughly equivalent to the required task of event formatting. A 16 MHz 68020 cpu currently performs event unpacking at about 200 KWords/second. A 25MHz 68040 is approximately 6 times more powerful, which indicates that one cpu is able to re-format events at 1MWord/secomd. Some reduction can be expected due to memory bandwidth contention during data block transfers. Any further processing requires additional processing power. Further processing will include reducing the data volume by calculating the total energy collected in the event, possibly gain matching, filtering on TDCs etc. However, some of these transformations could be applied in the Event Processor.

5.5 Modus Operandi

One HSM/FIC pair of boards has been shown to be capable of an input rate to the Event Builder of 1 MWord/second. To achieve this data rate then VMEbus and VSBbus will both have to be used. Two data transfers are necessary per block of event data. The first involves a transfer from the HSM8170 memory to dual-ported memory of a processor. The second transfer is from the dual-ported memory of the processor to the output interface controller. The proposal is to use VSB for the first and VME for the second transfer.

The maximum number of slots on a VSBbus is six, which can be filled by one HSM8170, one FIC8230 and upto four processing cpus. The first stage would be to use one VSB backplane. This would support 1MWord/second input and 1.5 MWord/second output to the Event Processor.

Adding a second VSB backplane would allow 2 MWords/second to be processed. This would have to be reduced to 1.5 MWords/second on output. The two HSM8170 boards would then be used in flip-flop mode.

There would be space in the VME crate for a third VSB backplane if necessary. Three HSM8170 connected together in cascade would work in principle.

The amount of processing power required for Event Building depends on two parameters - the data volume and the nature of the processing involved. A maximum of 4 cpus processing data at 1MWord/second would allow a reasonable amount of computation to be accomplished.

Data is strobed into fast memory in the HSM8170 in an automatically incrementing mode. The base address and count are settable. The memory accumulates events until a programmed count point is reached (say 16 Kbytes). This signals an interrupt to the controller cpu. This interrupt does not inhibit readout until the BUSY input is cleared. Event data still arrives until all data for that particular event has been transferred.

An *event-readout-in-progress* signal would need to be supplied and connected to the BUSY input. This signal would be derived from the REN/PASS signals of the FERA-like readout. The cpu then activates the *readout-inhibit* signal while it changes the address pointers to a free area of memory and resets the counter to re-activate the readin. With buffered Readout Controllers, this does not contribute any deadtime to event collection since it should be accomplishable very quickly.

Upon re-activating readin, the already collected buffer is DMAed to a processor for event building.

5.6 Software

A task running in the Crate Controller processor will accept control commands from the user as outlined in section 3.2.

The software running in the FIC8230 controls the HSM8170 and initiates DMA transfers to one of several processors. This software is not very complicated, but has to operate as quickly as possible when a memory block in the HSM8170 has filled. For these reasons it is proposed not to use an operating system but to code a simple task in C and assembler as necessary.

There is also code running in the controller for the optical fibre link. This is an independent subsystem, accepting requests for data block transfers and initiating block transfers down the link. It receives acknowledgements from the link, and sends acknowledgements to the sending cpu.

The processing cpus for the Event Builder and the Event Processor are the same. There are advantages to this approach for the software. Both execute a single task, and the foundation software to enable this should be identical. In principle this allows either a version of the sort software or purpose written code to be executed.

The transformations to be applied to the input data stream include event re-formatting, total energy calculation, possibly gain matching, TDC filtering and some Compton suppression related tasks. The nature of these transformations do not fit neatly onto the general sort language, and are better accomplished by purpose written code. Some of these transformations are better left to the Event Processor.

Hence it is proposed to code the necessary transformations in C as a single task. It is proposed that some options be allowed. If a suitable cross C compiler on a workstation were used then it would be possible to create a code module that could be downloaded onto 68K series microprocessors. The same code could be compiled to run on the workstation in a test form. It is proposed to download the code as opposed to being fixed as it may change from time to time, and maybe from experiment to experiment for particular requirements.

5.7 Limitations

It should be noted that a single crate VME-based Event Builder saturates at around 1.5MWords/second output rate using one fibre optic interface.

The input saturation rate is somewhat higher. Serial readout on a single cable into one HSM8170 saturates at 3MWords/second for a 20 metre cable assuming signal handshaking is used. If strobing were used instead of handshaking then a similar limit still applies , due to the number of HSM8170 modules and associated builder cpus that will fit into a crate.

To increase the input rate above these limits it would probably be necessary to revert to parallel sub-event readout, in which case the event merging would have to be accomplished in purpose-built hardware. There would still be one input stream to the Event Builder crate, but at a rate that would be too high to disperse to builder cpus. Therefore other crate technologies would be necessary, and we may have to wait for something like Futurebus.

It can be seen that any increase required over this rate will incur significant penalties in cost, development time and availability.

5.8 Costs

Costs are best estimates and are quoted including VAT. The total cost of the crate and contents for the proposed first stage of 1 MWord/second input rate is estimated to be £K30. This above estimate includes two 33MHz MVME165 processors. Should further processing be required, two further MVME165 cpus can be added for an extra £K8. The additional cost to enhance the Event Builder to 2 MWord/second input rate would be £K16. Also, if further data processing were demanded then extra MVME165 cpus can be added as above. External cabling costs are not included.