SLAC-PUB-3073
CERN/DD/83/4
March 1983
(M)

DATA ACQUISITION USING THE 168/E

J. T. CARROLL*
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

and

CERN, 1211 Geneva 23, Switzerland

S. CITTOLIN, M. DEMOULIN, A. FUCCI, B. MARTIN
A. NORTON, J.P. PORTE, P. ROSSI AND K.M STORR
CERN, 1211 Geneve 23, Switzerland

ABSTRACT

Event sizes and data rates at the CERN $\bar{p}p$ collider compose a formidable environment for a high level trigger. A system using three 168/E processors for experiment UA1 real-time event selection is described. With 168/E data memory expanded to 512K bytes, each processor holds a complete event allowing a FORTRAN trigger algorithm access to data from the entire detector. A smart CAMAC interface reads five Remus branches in parallel transferring one word to the target processor every 0.5 μs. The NORD host computer can simultaneously read an accepted event from another processor.

Presented at the Three Day In-Depth Review on the
Impact of Specialized Processors in Elementary Particle Physics,
Padova, Italy, March 23-25, 1983.

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.

1. INTRODUCTION

The UA1 experiment uses a large image chamber central detector in a dipole magnetic field. The Central Detector (CD) surrounded by electromagnetic and hadronic calorimeters, muon chambers and forward detectors is described extensively elsewhere.[1] The full detector has about 20,000 channels generating 1.6M bytes of raw data for each event. For the CD which produces most of this, zero suppression and data compaction yield an average of 38K bytes which must be recorded on tape. There is no data reduction for electromagnetic and hadronic calorimeters which have $\simeq$2K channels producing 8K bytes per event. The 5K muon drift tubes produce only a few hundred bytes for a typical event. A distribution of event sizes from a December 1982 run is shown in Fig. 1.

All front end digitizer and trigger electronics is located in a Mobile Electronic Control room (MEC) close to the detector. Data is collected and concentrated using the CERN Romulus/Remus CAMAC branch system.[2] As shown in Fig. 2, five Remus branches transmit calorimeter, muon, forward detector and CD data from the MEC

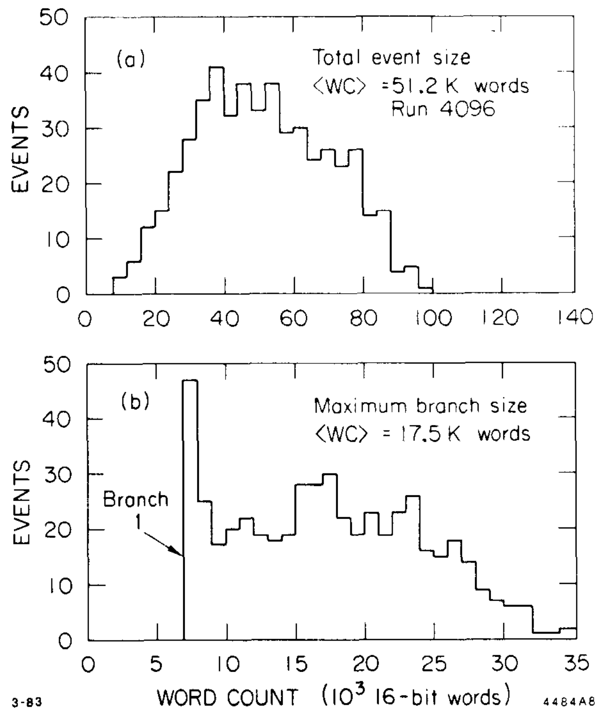


Fig. 1. UA1 event sizes.

to the main control room - a distance of 100 meters. The primary computing system at UA1 is based on two Norsk Data 100/500 machines referred to as NORD-A and NORD-B. Usually NORD-A is the master running a CERN Data Acquisition System (DAS) and the Remus routers in Fig. 2 are set to allow NORD-B to spy on the data transfer. An event trigger generates an interrupt to DAS running on NORD-A which then sends "Prepare and Go" commands to the five Remus branch drivers (RWBD) in the NORD-A CAMAC system crate. Data transfer from MEC to main control room then proceeds in parallel at an average rate of 2.5 μs per 16-bit word on each branch. Remus buffer memory can hold a single event while NORD-A reads the branches sequentially. A complete description of this UA1 data acquisition has been presented elsewhere.[3]

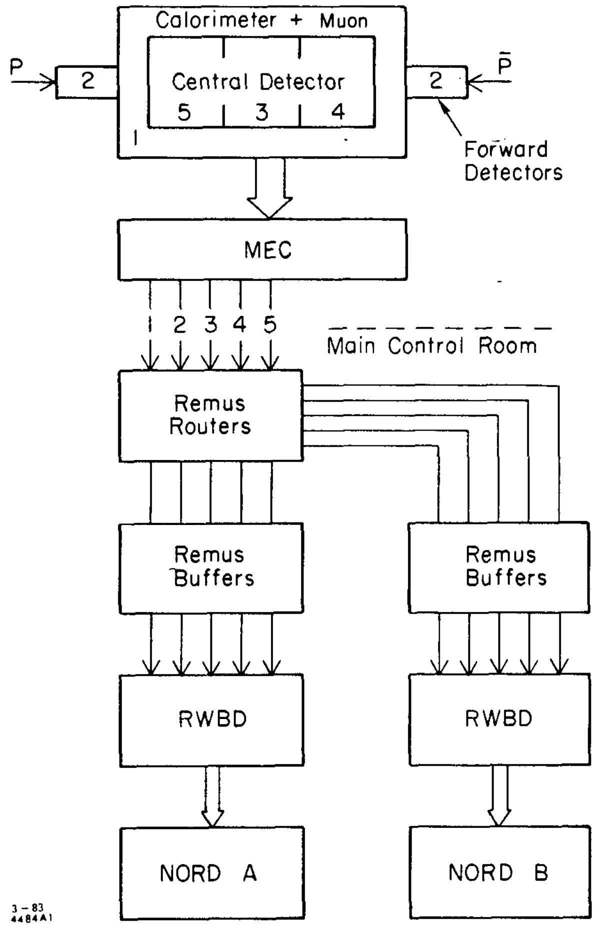


Fig. 2. NORD Remus system without 168/$_E$.

With six collider bunches there is a beam-beam crossing every 3.8 μs. At the design luminosity of $10^{30}\ cm^{-2}\ s^{-1}$ this should yield an interaction rate of 50K Hz while the maximum rate for recording production data on magnetic tape is about 5 Hz. There are 4 levels in UA1 trigger logic. The 0th level uses a ± 20 ns coincidence between hodoscopes in the p and $\bar{p}$ detector arms to select bunch-bunch crossings which have an interaction. This decision, made with NIM logic, provides a minimum bias trigger with rejection of background such as beam gas interactions. This is followed by 1st level calorimeter and muon triggers. A fast arithmetic processor groups and adds calorimeter FADC measurements providing total and transverse energy triggers in 2-3 μs. Another set of muon hardware processors searches for track candidates which point to within 100 mrad of the interaction vertex. A muon track is required to have three hits within a cone passing through a set of four parallel planes in one muon projection. This decision takes about 1 μs and uses only muon drift tube numbers - drift time is not available at this level. There is no dead time for the 0th and 1st level decisions which are completed between bunch crossings, but after a good 1st level trigger there is a dead time of 3-10 ms for CD data reduction. A muon 2nd level trigger using the FAMP[4] system is available during these few ms. It uses drift time to reconstruct muon tracks with 10 mrad accuracy. When an interaction produces a good 1st or 2nd level trigger, NORD-A initiates read-out on the 5 Remus branches and this transfer cannot be interrupted. All hardware for trigger levels 0, 1 and 2 is located in the MEC. The $168/E$ processors[5] in the main control room provide a 3rd level trigger.

As collider luminosity increases towards design value, the current 1st level trigger rate would soon exceed NORD capacity. Increasing the trigger rate by a factor of 5 over December 1982 rate of 0.7 Hz would require writing a 6250 bpi magnetic tape every 3-4 minutes and off-line processing costs probably prohibit running at this NORD limit. Under such saturated conditions another level of on-line filtering which reduced the NORD rate by only a factor of 2 would deserve consideration since it would double the effective luminosity for the experiment.

Several options for using $168/E$ processors were considered and in Fall 1981 a single processor was installed with 128K bytes of data memory and CAMFAST[6] interface as shown in Fig. 3. This system was similar to the SLAC Hybrid Facility $168/E$ trigger.[7] The CAMFAST module can spy on NORD CAMAC I/O and transfer data to a $168/E$

with no additional dead time. Since this interface is quite passive, considerable on-line development and testing was done while the experiment was in progress.[8] For an algorithm using only one Remus branch and with execution time limited to 25 ms, the system shown in Fig. 3 could at best achieve a factor of two improvement in data rate.

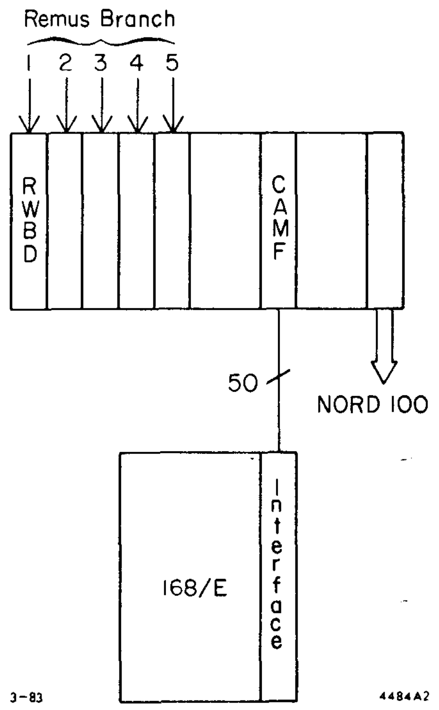


Fig. 3. 168/E with CAMFAST interface.

Much stronger improvement in performance can be obtained with a 3rd level trigger which provides event buffering and parallel processing. CAMAC memory buffers were considered but using 168/E processor memory as the event buffer gives the algorithm access to an entire event and is less expensive than Remus buffer memory. A 168/E software filter at this level provides easy implementation of new ideas as one acquires experience with the UA1 detector. Algorithms written in IBM FORTRAN can be fully tested on any IBM compatible machine and incorporated into DAS with minimal overhead for on-line tests. A new 3rd level trigger using the 168/E has been developed with the following objectives:

- support algorithms using calorimeter, muon and CD data,
- improve performance using event buffering and parallel processing,

- maintain compatibility with Remus branch structure and transfer data from MEC to 3rd level processors at the optimum Remus rate,
- minimize overhead on the NORD-A host computer,
- process 20 triggers per second for an algorithm with rejection rate of .75 and execution time of 50-100 ms, i.e. a factor of 4 in effective luminosity for a saturated system.

In order to achieve these objectives, a new PAX-GREYHOUND interface to the 168/E has been developed.

2. NEW HARDWARE FOR THE 168/E AT UA1

2.1 PAX

The 168/E interface developed for UA1 employs a CAMAC module called PAX which provides both autonomous read-out of Remus branches and all 168/E control functions for the host computer. Only four of the PAX CAMAC functions defined in Table 1 are required for the host computer to load a 168/E program, start execution, check processor status and read results from 168/E data memory. For example, the basic sequence of PAX functions to load and execute a program is as follows:

(a) F16 A2 - load PAX control register with target CPU# and select program memory,

(b) F16 A4 - load PAX address register with start address in 168/E program memory,

(c) F16 A6 - write to 168/E program memory,

(d) repeat steps (a)-(c) to load 168/E data memory,

(e) F16 A2 - load PAX control register with target CPU# and program control bits set,

(f) F16 A6 - write 168/E program counter and start execution. *

Six of the functions in Table 1 refer to the sequencer and address file which are the key logic for PAX operation as an auxiliary controller. A block diagram of sequencer

* This step is described in the Greyhound interface section.

Table 1. PAX CAMAC Functions

Function		Definition
F0 A0		READ PAX Status Register:
		bits 1-7 = sequencer address,
		bit 8 = time out,
		9 = not end of read sequence as defined by LAM mask in control register,
		10 = overflow,
		11 = not used,
		12 = interrupt process.
F0 A1	*	READ PAX Address File:
		32 Base and 32 Current 168/E addresses indexed by the PAX EXEC register.
F0 A2		READ from 168/E and Increment Address Reg.
		Read program and data memory, CSR and PC.
F0 A3		Test LAM, Q=1 if LAM set.
F16 A0		WRITE Sequencer Memory and Increment Counter.
F16 A1		ENABLE LAM.
F16 A2	*	WRITE PAX Control Register:
		bits 1-3 = 168/E CPU # (0-7)
		4 = Read/Write (1/0)
		5 = Control/Memory (0/1)
		6 = Program/Data (1/0)
		7 = Clear Greyhound
		8 = Overflow enabled
		9 = Sequencer time-out enabled
		10 = End condition enabled
		11-18 = LAM Mask (0-377 octal)
		19 = Cycle time (1=> 0.5 μs)
F16 A3		START PAX Sequencer.
F16 A4	*	WRITE PAX Address Register and Word Count.
F16 A5		WRITE Sequencer Transparent.
		Same as F16 A0 with sequencer instruction executed immediately instead of stored in memory.
F16 A6		WRITE to 168/E and Increment Address Reg.
F16 A7		INHIBIT LAM.
F16 A8		RESET LAM and Sequencer Logic.
F16 A9		WRITE Address File to 168/E + Inc. Register.
		An entry selected by a previous Seq-Internal (F16 A5) is transferred with 2 instructions. Upper/lower halfword is selected by address register LSB = 0/1.
F16 A10	*	WRITE Address File.
		The station # address file entry can be set with a Seq-Internal (F16 A5.)

* => 24-bit CAMAC I/O

logic is shown in Fig. 4. Sequencer memory provides a maximum of 128 16-bit instructions. As shown in Table 2, sequencer instructions are divided into four classes defined by bits 15-16 in sequencer memory. A Sequencer-External instruction can read any CAMAC module in the same crate with the PAX and transfer a 16-bit word to $168/E$ memory in one CAMAC cycle. Sequencer-Internal instructions provide internal PAX control, transfer address file data to $168/E$ memory and start the processor. Loops within sequencer memory and tests for end of a CAMAC read sequence are obtained with Sequencer-Branch instructions. The rather complex Seq-Branch instruction has six types of branch and test operations. Timing and physical constraints on PAX design forced some complexity in sequencer instructions - there were only two free slots in the main NORD-A system crate when PAX design was initiated. The Write-Block

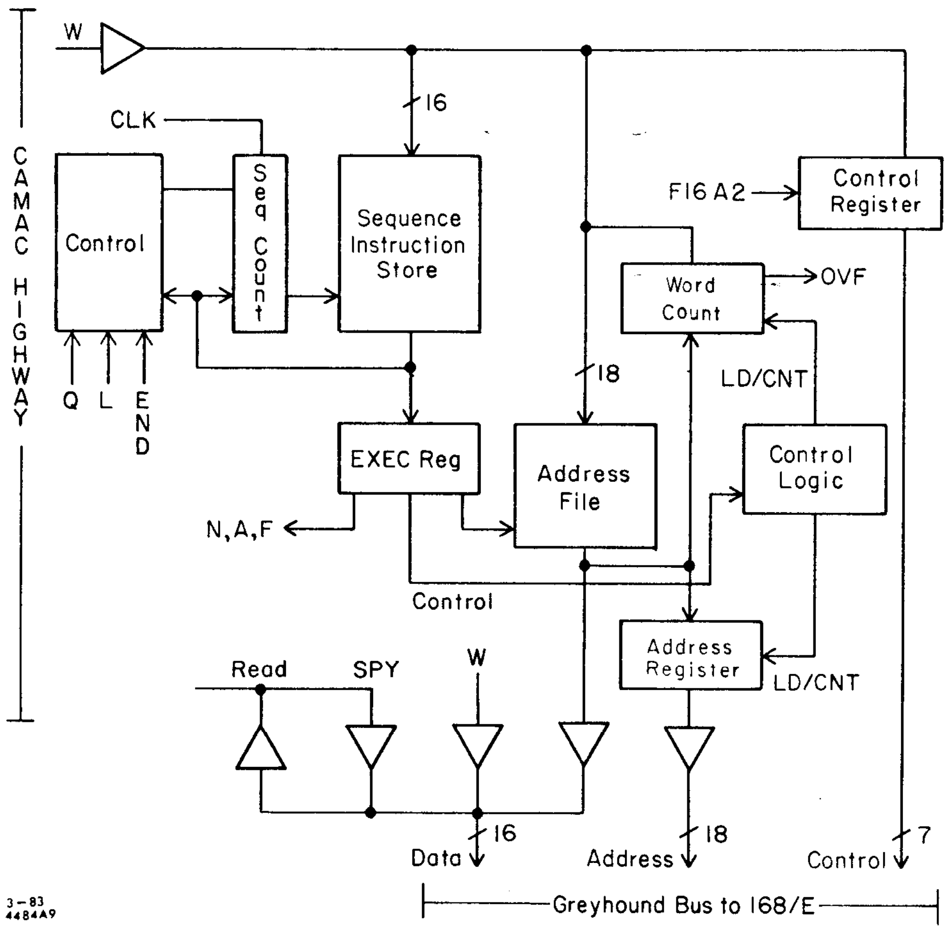


Fig. 4. PAX logic diagram.

Table 2. Sequencer Instructions

Instruction	Bits 1111111 6543210987654321	Definition
External	00NNNNNAAAAFFFFF	
	CAMAC function; CAMAC subaddress; CAMAC station #	
Internal	01AAAAA X X	
	Load address register; Load word count (WC); Write WC to address file; Write address file to 168/E; Start 168/E; Halt sequencer; Base/Current address (0/1); Address file index (0-31)	
Branch	10NNNNNTTAAAAAAA	
	Branch Address; 00000 – Unconditional branch; 10100 – Start 2 μs wait cycle; 11000 – Test for end of wait cycle; 01 – Branch if LAM(N) = 0; 10 – Branch if Q = 0; 11 – Branch if PAX end condition false. Bits 8-9 define the branch type. For branch type 01 bits 10-14 define the station #. For branch type 00 bits 10-12 define the type of wait operation and bits 13-14 are not used. Bits 1-7 contain the sequencer branch address (0-127 decimal.)	
Write Block	11XXXXXXXXXXXXXX	Enable block transfer.

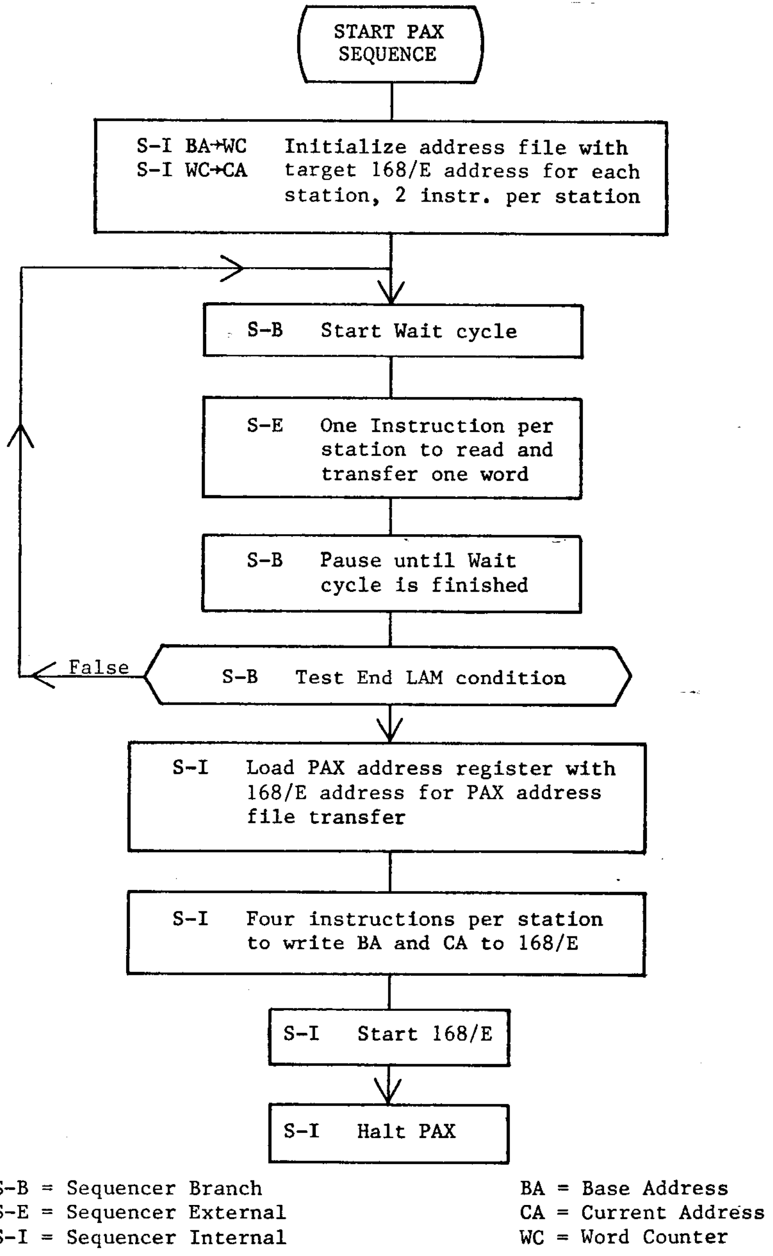
instruction enables a block transfer from $168/E$ memory to CAMAC highway. Once executed, it freezes the data path from $168/E$ memory in read mode allowing external write instructions to direct data to any CAMAC module.

In order for the sequencer to read Remus branches in parallel, the PAX must maintain a set of target addresses in $168/E$ memory. For each CAMAC station, the address file in Fig. 4 has a Base Address (BA) and Current Address (CA) in $168/E$ memory (a total of 64 18-bit addresses.) The PAX EXEC register holds the function, station and subaddress for CAMAC I/O and points to the corresponding address file entry. The 18-bit address register in Fig. 4 holds the target address in $168/_E$ memory. Seq-External loads this register with CA for station N, loads the word counter with the same CA, increments the address and stores the result back in the file if $Q = 1$.

An 8-bit sequence counter selects the next instruction for execution. It increments after external or internal operations and is set/cleared by a Seq-Branch instruction. A 7-bit control register holds the target CPU# (maximum of 8 processors) and Greyhound interface control lines. During NORD CAMAC I/O to a $168/_E$ the sequencer and address file operate in a transparent mode. For example, when the NORD loads PAX address register (F16 A4) the 18-bit address is transferred through the address file in unused address space - sequencer instructions access 64 out of 128 words in this file. Similarly, F16 A5 can execute any sequencer instruction immediately.

A flowchart of a PAX sequence to read UA1 Remus branches is shown in Fig. 5. Before entering a CAMAC read loop, the PAX executes Seq-Internals which copy BA to corresponding CA for each CAMAC station. The CAMAC read loop executes five Seq-Externals, one per Remus branch. A Seq-Branch instruction tests a 2.0 μs wait cycle to guarantee a minimum delay between successive read cycles on the same RWBD as required by Remus specifications. If this wait cycle is not complete, the sequencer pauses and reexecutes this wait test every 125 ns. At UA1 the end of an event transfer has been defined by the logical AND of a LAM mask in the PAX control register (F16 A2 in Table 1) and Remus End LAM's for the five branches. The station numbers for each bit in this mask are set by jumpers in the PAX. When the end condition is true signalling completion of transfer on all branches, Seq-Internals copy BA and CA for each branch to $168/_E$ memory. The $168/_E$ algorithm needs these addresses to calculate word counts for each branch. Finally Seq-Internal instructions start the processor and halt the PAX sequencer. Such a sequence to read 5 UA1 branches takes 43 instructions and the sequence to start the Remus transfer is 16 instructions. The mean transfer time, fixed by the largest branch, is 44 ms for the distribution in Fig.1.

The PAX sequencer executes with an internal cycle of 125 ns. If the external CAMAC cycle is 0.5 μs there are four PAX internal cycles for each CAMAC cycle. Only the first internal cycle is required to initiate an external CAMAC operation. A maximum of 3 Seq-Branch instructions immediately following a Seq-External can be executed while the PAX-CAMAC cycle is in progress. Consequently a sequence read loop like that shown in Fig. 5 can easily be organized with no overhead for branch operations. Seq-Internal requires all 4 internal PAX cycles.



START PAX SEQUENCE
S-I BA→WC
S-I WC→CA
Initialize address file with target 168/E address for each station, 2 instr. per station
S-B Start Wait cycle
S-E One Instruction per station to read and transfer one word
S-B Pause until Wait cycle is finished
False
S-B Test End LAM condition
S-I Load PAX address register with 168/E address for PAX address file transfer
S-I Four instructions per station to write BA and CA to 168/E
S-I Start 168/E
S-I Halt PAX
S-B = Sequencer Branch
S-E = Sequencer External
S-I = Sequencer Internal
BA = Base Address
CA = Current Address
WC = Word Counter

Fig. 5. Flowchart for a PAX read sequence.

2.2 The Greyhound Interface

The 168/$_E$ processors at UA1 need an interconnection that allows the PAX to transfer data over a distance of $\approx$5 meters at 500 ns per 16-bit word. The interface on each processor must allow either of two such buses access to 168/$_E$ memory and control functions. Without this feature there would be a dead time of $\approx$100 ms whenever an accepted event is read from 168/$_E$ memory. The two buses do not need simultaneous access to 168/$_E$ memory. However, the 168/$_E$ Control Status Register (CSR) needs a true dual port read function so each PAX and the corresponding support tasks on the host computer can obtain 168/$_E$ status independently.

The early Fastbus interface was inadequate and a more parallel structure called the Greyhound Bus has been developed for UA1. The Greyhound bus is carried over two 50 way twisted pair flat cables - two cables for each PAX. As described in Table 3, there are 18 address lines for the half megabyte 168/$_E$ data memory space and 16 data lines. The wordsize for Remus data transfers at UA1 is always 16 bits. The LSB of the address field set to 0 (1) selects the upper (lower) halfword in 168/$_E$ memory.

There is no protocol on the Greyhound bus. To the PAX the 168/$_E$'s appear as elements in its memory map as accessed through Greyhound address and control lines. The interface in a 168/$_E$ crate DC-follows the PAX which has asserted mastership and write operations are triggered by the Greyhound S1 signal. Since there is no protocol, the host computer and 168/$_E$ programs are responsible for checking data transfer integrity.

A block diagram of Greyhound dual port structure is shown in Fig. 6. Both PAX ports have access to program or data memory and to the CSR. The CSR described in Table 4 provides all control and status functions required to operate a 168/$_E$ processor. The host computer selects Greyhound control space by loading PAX control register bits 5-6 with "Control" and "Program" options as defined in Table 1. When control space is selected any even address will access the CSR and odd addresses access the lower 16 bits of the 168/$_E$ program counter. The host computer can start a 168/$_E$ program with the following operations:

Table 3. Greyhound Bus

Signal		Definition
A1-18	*	Eighteen 168/E address lines.
D0-15		Sixteen bi-directional data lines.
R/W	*	Control line indicating read or write cycle.
D/C	*	Control line indicating control or data space access.
P/D	*	Control line indicating program or data space access.
PRESEL	*	Pulse generated in the PAX by setting PAX CR ready to write into processor data space. The pulse triggers the selected 168/E into a mode in which it alone will respond to PAX operations.
S1	*	Timing signal used in write operations. S1 is true 250 ns after validation of the address lines and lasts for 250 ns. There is no timing during cycles. The interface 'DC-follows' the address lines until CYCVAL is negated.
RESET	*	Master Clear.
HALT		Wired 'OR' of all 168/E halt interrupts. This sets PAX status register bit 12 and is available on the PAX front panel.
PN0-2	*	Encoded processor select.
START	*	A pulse of not less than 100ns on this line will start the selected processor. The user must ensure that the required PC value is set before enabling this action. It is only generated by the PAX sequencer.
		* => unidirectional

(a) load an odd address in the PAX address register,

(b) use F16 A6 to write zero to the 168/E program counter. This function increments the PAX address register so the next write will access the CSR.

(c) Write 5 (D) to start the processor with halt interrupts disabled (enabled).

Logic for 168/E clock signals on the Greyhound interface is identical to the Fastbus interface.

While the CSR can be read simultaneously from both ports, a PAX must secure mastership of a processor before any other I/O. This is accomplished with the following sequence of PAX functions:

(a) F16 A2 - load PAX control register with target CPU# and all other bits set to zero,

(b) F16 A2 - load PAX control register with same CPU# and bits 4-6 set to write-data-memory.

This sequence generates a PRESELECT pulse on the Greyhound bus. If both PAX's do this simultaneously, then CONFLICT bit 9 is set in the Greyhound CSR. This indicates a potentially fatal error in the control task on the host computer which must be programmed to avoid such contention. The clear bit in the PAX control register will reset all processors on the bus and clear the conflict condition.

Each Greyhound has three lemo outputs with the following positive true TTL signals:

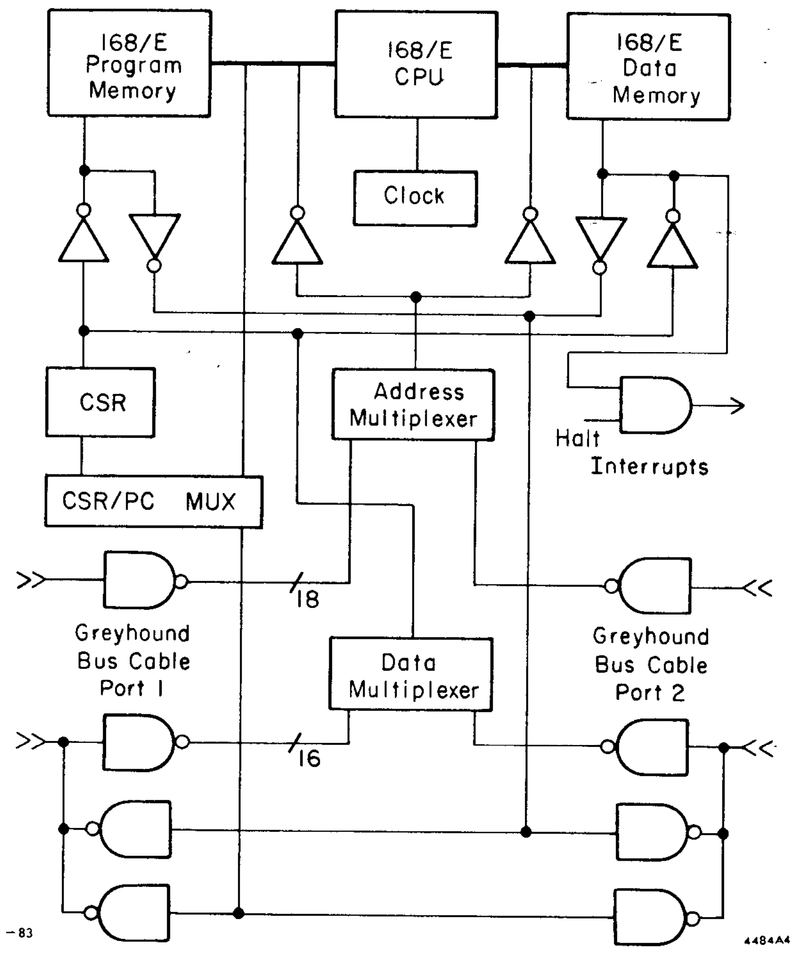


Fig. 6. Greyhound interface block diagram.

Table 4. Greyhound Status/Control Register

Bit	Name		Definition
0	START	R/W	Setting this bit enables the clock generator and starts the 168/E. When the clock starts this bit is cleared. A PAX Seq-Internal can set this bit.
1	FLAG0	R/W	Reserved for software use.
2	MPU	R/W	When set the 168/E has control over it's own memories. If clear the interface may have memory access.
3	INTENB	R/W	True enables a 168/E halt to set the front panel TTL status signals and the 'OR' halt on the Greyhound bus.
4	HALT	R/-	True when CPU halted.
5	RUN	R/-	True between START and HALT.
6	RESET	-/W	Setting this bit generates an interface reset which also resets itself.
7	SPY	R/W	If set the interface will spy on and duplicate all write operations from PAX1 irrespective of their destination.
8	HEREIAM	R/-	This bit is always 1 when the PAX is reading the CSR from a real processor. A non-existant processor returns zero.
9	CONFLICT	R/-	Set if interface receives simultaneous tries to preset from both ports. This is potentially fatal since it is not clear which PAX had mastership. The bit is only cleared by a reset function.
10	P1OPS	R/W	True means PAX1 has preset the 168/E and it will now obey PAX1. If 'SPY' is not set then it is cleared when PAX1 presets another 168/E. It is always set to zero if PAX2 selects the processor or if an external START is received.
11	P2OPS	R/W	True means PAX2 has preset the 168/E and it will now obey PAX2. If 'SPY' is not set then it is cleared when PAX2 presets another 168/E. It is always set to zero if PAX1 selects the processor or if an external START is received.
12	FLAG1	R/W	Reserved for software use.
13	FLAG2	R/W	Reserved for software use.
14	PC17	R/-	Program counter bit#17.
15	PC18	R/-	Program counter bit 18.

$$\text{Lemo}\,1 = \text{CPU Halted}$$

$$\text{Lemo}\,2 = \text{CPU Halt} \cdot \text{AND} \cdot \text{DM0}$$

$$\text{Lemo}\,3 = \text{CPU Halt} \cdot \text{AND} \cdot \text{DM1}$$

CSR bit 4 true enables these levels when the processor halts and clearing this bit resets the lemo outputs. Lemo outputs 2-3 can be used as trigger flags set by an algorithm write to Data Memory immediately before the program halts. An additional lemo output provides a TTL "RUN" signal which is the envelope between processor start and halt.

2.3 A Modified RWBD

The PAX can generate external CAMAC cycles of 0.5 or 1.0 μs with control register bit 19 set to 1 or 0. A 0.5 μs cycle allows PAX to access each of the five UA1 branches every 2.5 μs which is the average Remus data rate at the LSS5 control room. In this preferred mode of operation the CAMAC S_1 pulse is 200 ns and there is no S_2 generated by the PAX. The RWBD Remus branch drivers[9] used with the PAX can operate with either CAMAC cycle selected by a hardware switch at the rear of the module. For a 0.5 μs cycle each RWBD generates its own S_2 internally. In this non-CAMAC standard mode of operation the Seq-Branch "wait cycle" can be used to guarantee that successive accesses to the same RWBD do not exceed Remus specifications.

The PAX sequencer can issue the "Prepare and Go" command to initiate branch read-out but Remus specifications require a minimum delay of 9 μs before the first RWBD read instruction. The host computer must meet this requirement or the sequencer can be programmed to provide the minimum delay with wait cycles. All RWBD's used with the PAX generate an "End LAM" defined as LAM = 1 at the end of a transfer. If a Seq-External read is followed by Seq-Branch with "test on Q," the two instructions must be separated by 1 (2) NOP's for a cycle of 0.5 (1.0) μs. The NOP (an unconditional branch to next sequencer instruction) allows sufficient time for the RWBD to respond and set Q on the CAMAC bus. A Seq-Branch immediately after a Seq-External is executed before the external S_1. There have been no problems for RWBD operation with the 0.5 μs PAX-CAMAC cycle.

2.4 168/$_E$ MEMORY

The 168/$_E$ processors at UA1 use a memory board developed at DPHE, Saclay[10] which allows each processor to be loaded with a complete event. Each 168/$_E$ has 8 boards with a full data memory space of 512K bytes. Program memory is loaded on two of the 8 boards so 168/$_E$'s at UA1 have the original design limit of 32K micro-instructions - sufficient for a practical real-time algorithm. Expanding data memory by a factor of four required 2 more bits in memory address logic on the 168/$_E$ integer CPU board. The data memory address adder was increased from 16 to 18 bits demanding a second level in the carry look-ahead, but with the 55 ns access time of the INMOS IMS1400 this was accomplished without difficulty.

The original 168/$_E$ memory test developed at SLAC checked for unsatisfactory address transitions between every pair of addresses in the memory test interval. This is a strong test but execution time $t \alpha N^2$ where N is the number of test words and with 64K byte memory boards a single card takes 2 hours and a full processor ≈128 hours. A Random MEMory test (RMEMV6) was developed using a simple algorithm to select random addresses and check a contiguous region as follows:

(a) write complement of a fullword pattern to a random test address,

(b) write exclusive OR of pattern and address to memory above and below the test address,

(c) read entire test interval using non-zero displacements to exercise address calculation.

RMEMV6 also has a burst write test for worst case conditions in successive memory accesses. This diagnostic was essential in tracing an early problem with the INMOS IC. Before installation at LSS5 each processor was required to pass all CPU and memory tests at 4.8-5.2 volts and execute the random memory test for ≈15 hours. After installation the only memory failure observed was a hard error on a single INMOS IC - an error easily detected by less sophisticated diagnostics.

3. NORD DAS WITH THE 168/*E*

The CAMAC and Greyhound configuration for the multi-processor system with PAX interface to NORD-A is shown in Fig. 7. The original system in Fig. 2 has been modified to include an additional set of Remus routers to a CAMAC crate with five RWBD's and PAX-A. The original RWBD's remain in the primary NORD-A system crate with PAX-B allowing DAS to bypass 168/*E* processors. The PAX-A crate resides on a branch from a second system crate which can be accessed by both NORD-A and NORD-B. For DAS using 168/*E*'s the routers are configured with RWBD's in the PAX-A crate as masters. The host computer transfers a new event to a preselected 168/*E* by executing a sequence in PAX-A. An accepted event is read from 168/*E* memory using PAX-B and recorded on tape by ZREAD, the main DAS task. The transition

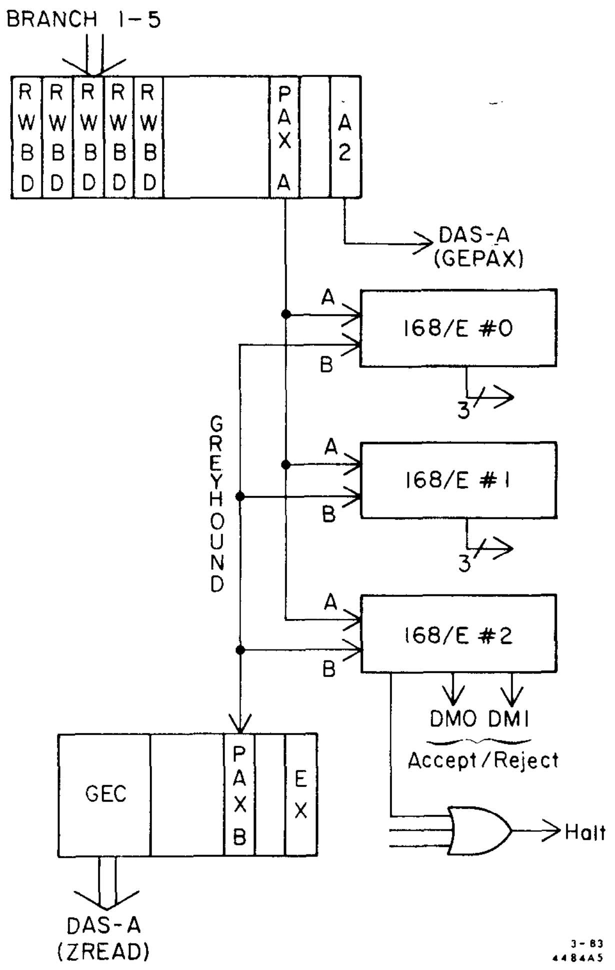


Fig. 7. NORD configuration with PAX and 168/*E*.

between data acquisition without and with 168/$_E$'s is accomplished in software from menus available at the NORD-A console.

For DAS without 168/$_E$'s the only related CAMAC interrupt to NORD-A is the 1st/2nd level trigger. To operate a 3rd level trigger the system must also handle interrupts generated by PAX sequencer and 168/$_E$ halts. The system responds to these three interrupts as follows:

1. Hardware trigger
   - disable hardware trigger and find next free 168/$_E$,
   - start PAX-A executing a small RWBD "Prepare and Go" sequence,
   - preselect target 168/$_E$, set PC=0 and set CSR with halt interrupt enabled,
   - set PAX-A control register for write-data-memory,
   - enable PAX-A LAM,
   - start main PAX-A sequence reading branches 1-5.
2. PAX-A Halt
   - disable PAX-A LAM,
   - check PAX-A status register for an abnormal sequencer condition,
   - read 168/$_E$ CSR and check for Greyhound conflict,
   - enable hardware trigger if another processor is free.
3. 168/$_E$ Halt
   - read CAMAC status register to determine which CPU halted,
   - read CAMAC status register with accept/reject flag,
   - ignore rejected events and add 168/$_E$ to list of free processors,
   - instruct ZREAD to record accepted events on tape.

If PAX control register has bit 9 set to enable time-out, sequencer execution stops after an elapsed time of about 500 ms. Similarly, with the overflow bit set the sequencer stops if a 168/$_E$ address CA(N) overflows when incremented after reading station N. If the PAX continued to read station N under this condition it would overwrite algorithm local constants. Either of these conditions can produce a CAMAC interrupt. A direct task called GEPAX running at NORD level 6 was developed to handle all interrupts associated with DAS using PAX-A (interrupt response time $\approx$100 μs). In the case of an abnormal PAX termination or Greyhound conflict, GEPAX was coded to ignore the event, update a run summary table and continue after resetting the PAX and/or 168/E for another trigger. After a 168/E halt, GEPAX reads CAMAC status registers which provide the CPU# and algorithm accept/reject flag. Note that this information is available whenever a processor halts even if both PAX modules are busy. The most recent version of GEPAX ignored events rejected by the filter algorithm. The processors provide a 3rd level trigger but the host computer can still make the final decision concerning which events are recorded on tape. For example, DAS could easily be programmed to record a random sample of rejected events for off-line analysis.

For accepted events GEPAX generates an interrupt to direct task ZREAD running at level 8. ZREAD uses PAX-B to read a small algorithm summary, PAX-A address file data written to 168/E memory and input from the five Remus branches. Since PAX-B is in a NORD system crate, this transfer proceeds at the maximum CAMAC rate of 1.7 μs per 16-bit word for DMA. All control operations performed by GEPAX are done with programmed I/O since the number of PAX-A instructions required is small. The sequencer in PAX-B is not used in this configuration but the modules are identical.

The host computer loads PAX-A sequencer memory and address file once at run initialization. The host must also maintain a table with the status of each processor and a free 168/E is found by searching this table. In Fig. 7 all the processors are functionally equal. If a 168/E failed DAS could disable the processor in the status table and continue acquisition with the remaining pair of processors. This equality can be a powerful diagnostic tool for checks on algorithm plus data validity. PAX-Greyhound design allows simultaneous transfer of the same event to any pair of processors. The host computer can then read and compare algorithm results checking for consistency or against some standard.

4. 168/E ALGORITHMS

A FORTRAN algorithm can access data from any of the Remus branches using labelled common blocks defined in Table 5. Arrays for each branch are dimensioned somewhat larger than the maximum branch size for normal operation. The 168/E data memory addresses in Table 5 locate Remus input common blocks in high memory and algorithm local constants and commons in low memory. This protects the algorithm against a Remus data transfer error since PAX addresses increment for each word transferred and the sequencer halts if an address overflows. Absolute addresses for all algorithm common blocks are fixed by "locate" data cards provided to the translator program. Clearly the PAX address file must be loaded with the same Remus branch target addresses as used for algorithm translation to 168/E microcode. However, the support task for accepted events need only know the address of COMMON /RESULT/ in Table 5.

Table 5. Algorithm Data Memory

BA(bytes)	COMMON	Contents
0		FORTRAN algorithm local constants and common blocks.
20000	/RESULT/HW(2048)	PAX address file and algorithm results.
21000	/DPRINC/FW(4096)	FORTRAN debug print buffer.
25000		not used
2A000	/BR1/HW(12288)	Branch 1 = calorimeter + muon
30000	/BR2/HW(16384)	Branch 2 = forward detector
38000	/BR3/HW(49152)	Branch 3 = central detector
50000	/BR4/HW(49152)	Branch 4 = central detector
68000	/BR5/HW(49152)	Branch 5 = central detector

HW => Half Word; FW => Full Word

The main routine in a UA1 algorithm has the following basic structure:

```
      COMMON/RESULT/IABC(2,5),IBERR,IRYES,IALG(106),IPBC(2,5)
      INTEGER*2      IBERR,IRYES,IALG
C
C     IABC  = BA and CA for branches 1-5 loaded by algorithm,
C     IBERR = algorithm branch data error flag,
C     IRYES = algorithm accept/reject flag,
C     IALG  = algorithm results,
C     IPBC  = BA and CA for branches 1-5 loaded by PAX.
C
      CALL EVPROC
      CALL ALG
      IF(IRYES.EQ.1) I=1
      IF(IRYES.EQ.0) I=2
      IF(IBERR.NE.1) I=3
      IANS = I
      STOP
      END
```

While addresses in array IPBC are 32-bit words, only the least significant 18 bits are defined by the PAX. Subroutine EVPROC masks off the upper 14 bits of each address, copies them to array IABC and sets flag IBERR if there is an obvious transfer error, e.g. $BA > CA$ or overlapping data windows. The algorithm and/or NORD-A also check integrity of Remus branch structure for each event. If there are no errors in address data, the filter algorithm is executed and a short summary of results for output to tape is saved in array IALG.

Immediately before it halts the algorithm executes an instruction (IANS=I) which accesses $168/_E$ data memory and sets bits 0-1 as follows:

DM1	DM0	Definition
0	0	Undefined => fault,
0	1	Event accepted,
1	0	Event rejected,
1	1	Error in Remus data.

For accepted events NORD-A reads COMMON /RESULT/ to obtain $168/_E$ addresses for each Remus buffer and algorithm results for output to tape. Branch commons must be read from PAX-B in five DMA transfers since sections with data from the last event are not contiguous. Ordinarily the $168/_E$ Remus commons are not cleared between successive events since this would take about 80 ms for this block of 352K bytes.

A UA1 filter algorithm is developed and tested off-line, translated to $168/E$ microcode, transferred over CERNET to disk at LSS5 control room and downloaded to each processor at the start of a run sequence. After a program is loaded the host computer always reads back $168/E$ memory and compares with the disk file - no errors were observed during Fall 1982 runs. The user developing an IBM FORTRAN algorithm need only adhere to the basic structure described above for input data and results. Absolute addresses in Table 5 are seldom changed and the translation step is a standard procedure requiring very little user input for this type of program.

The first algorithm tested at LSS5 searched for high momentum charged track candidates near 90^o to the beam axis in the vertical (bending) plane. The algorithm searches four CD drift volumes above and four below the horizontal beam plane. The search is limited to a single drift volume at a time, selecting eight wires in each volume with a uniform 6 cm separation. An essential feature of this and any algorithm using CD data is fast selective decoding of a few wires with minimal unpacking - in this case only the drift time. Execution time for accepted and rejected events is shown in Fig. 8a-b. An event is accepted as soon as a track candidate is found in any of the 8 volumes. Consequently rejected events are more time consuming which seems to be a general characteristic of $168/E$ trigger algorithms. Comparison with the same events run on IBM shows the $168/E$ is 2.6 times slower than a 370/168 for this program[8].

Another FORTRAN algorithm searches for muon track candidates using a procedure logically equivalent to that of the 2nd level trigger which is written in M68000 assembly language.[11] This subroutine searches all 20 muon modules on the top, front and sides of the UA1 detector and the 10 bottom modules. Each projection of each module is searched as follows:

(a) prepare a table of cone candidates from hits in the two planes closest to the beam line,

(b) prepare a table of drift times - raw data is not sorted,

(c) for each plane use lookup tables to construct the logical OR of patterns consistent with a cone from the interaction region,

(d) require a track candidate to have at least 3 hits in the 4 planes intersecting the cone.

Equivalent 168/*E* execution time measured on IBM is shown in Fig. 8c. The distribution for all events peaks at 2 ms and most events above 5 ms have one or more tracks. About 50% of this execution time is required to prepare tables of cone candidates and drift times for each module to be searched.

An algorithm using a more refined estimate of energy deposited in the central electromagnetic calorimeter (Gondolas) has also been tested. A muon or calorimeter algorithm could be executed first to define dynamically a region of the central detector to be searched. During December 1982 the PAX-168/*E* system was used with a variety of test algorithms recording 28K events on tape. An additional 8400 events were accumulated using a prototype physics algorithm (muon + CD + calorimeter) with results included on tape. From this sample 6000 events were later checked and found to be in complete agreement with the same algorithm run on IBM.

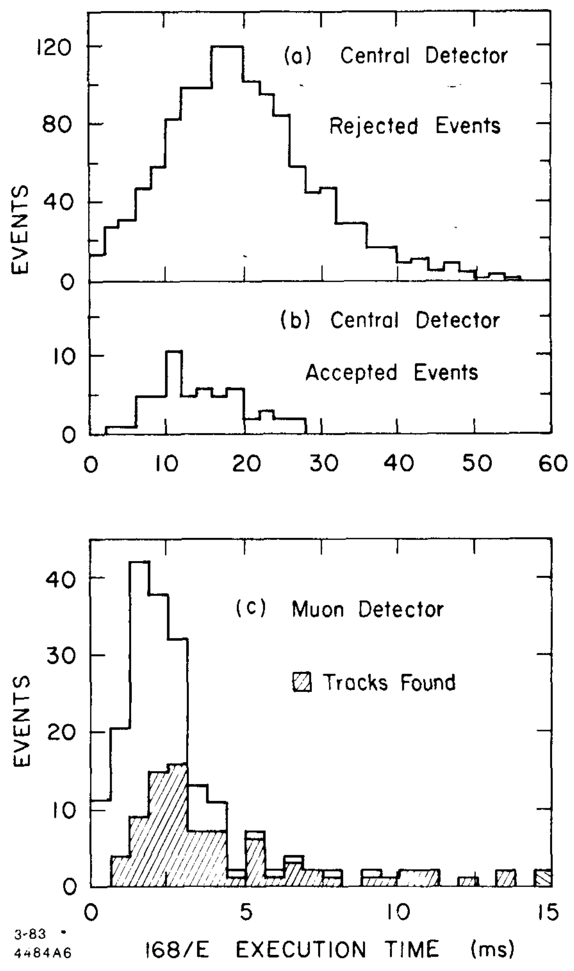


Fig. 8. 168/*E* algorithm execution time.

The end of the 1982 collider schedule was followed by a few days of central detector calibration runs using cosmic ray data. In only a few hours a very simple algorithm was coded, tested on IBM and transferred to the $168/E$. In addition to requesting a minimum energy deposit in the forward and backward sections of the calorimeter, it required a minimum signal in the central detector. Without the PAX-$168/E$ trigger it would not have been possible to make such correlations between data from parallel Remus branches.

5. NEW DEVELOPMENTS

As discussed in the Introduction, it takes several milliseconds to complete the CD read-out and a 2nd level muon trigger is available during this time. After CD data is transferred to Remus buffer memories, a single event cannot be cleared. However, the Read-Out Processor (ROP) in each CD crate has a 1K 16-bit buffer and it takes about 8 ms to fill this buffer. Data from a CD selective read-out could be transferred to a $168/E$ providing another 2nd level trigger. For example, reading only drift time after zero suppression for two wires in each CTD module (12 wires per module) should yield about 1K bytes per event. Although the quantity of CD data and transfer time to $168/E$ is small for this scenario, a CD algorithm would probably still require access to muon or calorimeter results to scan a selected region of the central detector in this 2nd level time interval.

A major improvement to the 3rd level trigger system is already in progress and expected to be operational at the start of the next collider cycle. The layout in Fig. 7 has been modified to include two more processors, a 3rd PAX, color graphics and Super CAVIAR.[12] As shown in Fig. 9, the Super CAVIAR labelled DACQ runs the "GEPAX" task supervising read-out by PAX-A and controlling processors 1-4. This reduces the number of interrupts which must be processed by NORD-A. A sample of events are transferred simultaneously to $168/E$ #0 and one of the four trigger processors. Using PAX-C, the Super CAVIAR labelled DISP can display algorithm results on the MATROX or transfer a full event from processor 0 to NORD-B for analysis by monitoring programs (previously NORD-B could only spy on one branch).

When an event display algorithm runs on $168/E$ #0, the processor prepares a buffer with pixel coordinates and color for each image, e.g. CD tracks or points. The buffer is transferred by PAX-C in block mode via a CAMAC module to the MATROX 512*512 color display. Optionally the same results can go to NORD-B. The display algorithm also maintains statistics which may be output to the MATROX display or NORD-B. The DACQ CAVIAR monitors the complete acquisition system with color displays showing status of PAX and 168/E activity, algorithm results, data errors, PAX read-out and 168/E processing time, etc.

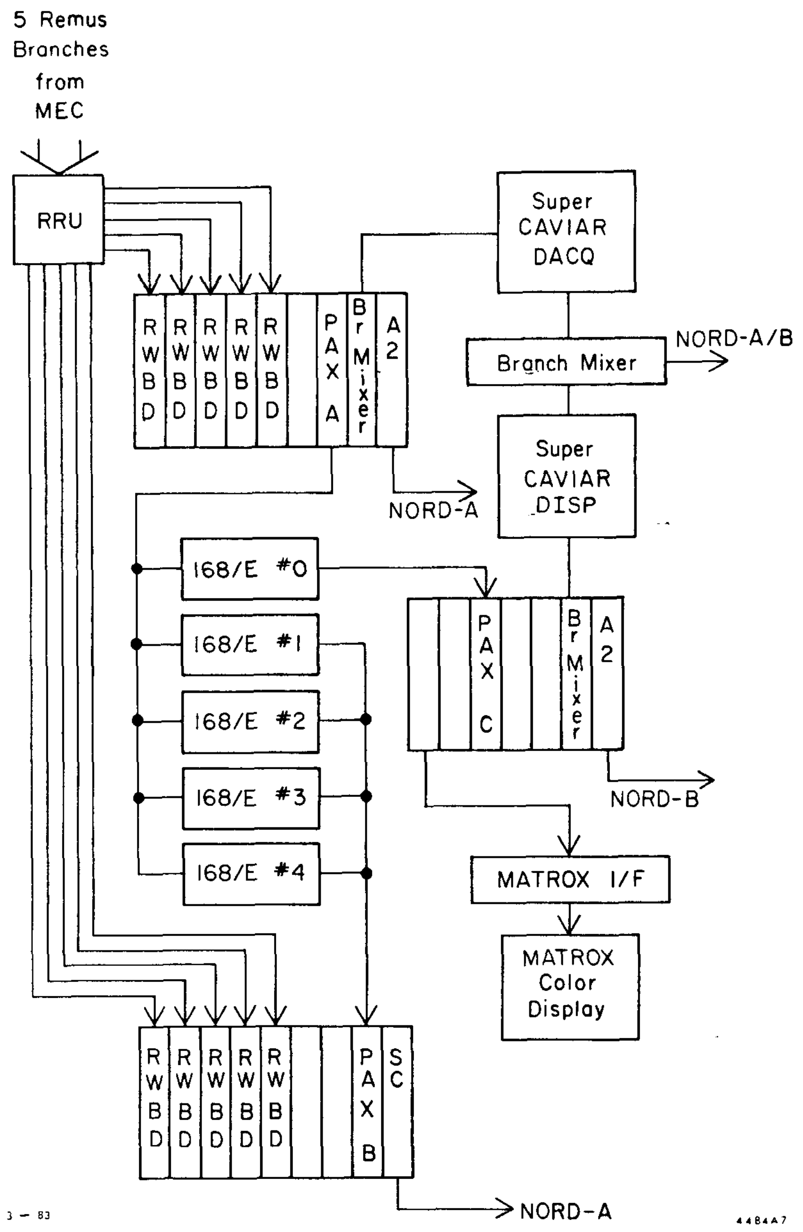


Fig. 9. PAX-168/E with Super CAVIAR.

Super CAVIAR firmware has been extended to include resident routines that perform the following tasks:

(a) load and verify a 168/E program from any CAVIAR peripheral,

(b) read-write-display 168/E program and data memory in several formats including a transparent two way conversion between IBM and CAVIAR floating point representation,

(c) control loading and execution of PAX sequences, and

(d) test-debug a complete PAX-168/E system.

In addition an interactive monitor has been developed which allows 168/E data memory to be described in terms of symbols representing variables or arrays of any type, e.g. a FORTRAN program COMMON statement. The 168/E memory can be set/displayed via these symbols. A simple protocol for 168/E subroutine call with parameter passing has also been implemented, and the parameters may be modified, routines executed, and results retrieved via the CAVIAR. This development could be used for CPU bound problems with limited data I/O, e.g. vector rotation and function minimization.

6. CONCLUSION

The 168/E processors have been integrated into the UA1 data acquisition system with an interface supporting optimum Remus event transfer rates. The PAX sequencer and Greyhound dual port structure are reliable and versatile. Sophisticated algorithms with access to a complete event can now be written in a high level language, giving the possibility of triggers which correlate data from parallel branches and improve effective luminosity by a significant factor. At the same time, the event monitoring and display capabilities of the system have been improved.

7. ACKNOWLEDGEMENTS

The design of the final system emerged only after many discussions, and we would like to thank D. Lord, D. Jacobs, and P. Kunz for their contributions. We are indebted to J. Prevost and G. Seite, not only for their design, but also for providing many of the 64K 168/E memory boards, and to L. McCulloch for the modified version of the Remus branch driver. C. Bertuzzi participated in designing of the PAX module and made the initial implementation. Responsibility for construction and testing of the $168/E$ processors was taken by N. Bosco, R. Hinton and M. Kudla. Finally we gratefully acknowledge the continued support and encouragement of Prof. C.Rubbia and the UA1 collaboration.

REFERENCES

1. A. Astbury et al., "A 4π Solid-Angle Detector for the SPS Used as a $p\bar{p}$ Collider at c.m. Energy of 540 GeV," CERN/SPSC/78-06 (1978).
2. P. J. Ponting, "A Guide to Romulus/Remus Data Acquisition Systems," CERN EP-Electronics Note 80-01 (1980).
3. S. Cittolin, "The UA1 Data-Acquisition System," International Conference on Instrumentation for Colliding Beam Physics, Stanford (1982); CERN DD/80/03 (1982).
4. L. O. Hertzberger et al., "The Fast Amsterdam Multiprocessor (FAMP) System Hardware," Proceedings of the EPS Conference on Computing in High Energy and Nuclear Physics, Bologna (1980).
5. P. F. Kunz et al., "The LASS Hardware Processor," Proceedings of the 11th Annual Microprogramming Workshop, SIGMICRO Newsletter 9, 25 (1978).
6. D. Lord et al., "The $168/E$ at CERN and the MARK II: An improved processor design," Proceedings of Topical Conference on the Application of Microprocessors to High-Energy Physics Exp., Geneve, CERN 81-07 (1980).
7. J. T. Carroll et al., "On-Line Experience with the $168/E$," Proceedings of Topical Conference on the Application of Microprocessors to High-Energy Physics Exp., Geneve, CERN 81-07 (1980).
8. J. T. Carroll et al., "On-Line Use of $168/E$ for UA1," UA1 Technical Note TN 82/08 (1982).
9. C. Jacobs and L. McCulloch, "CAMAC Read Only Branch Driver Type 243," CERN CAMAC Note 63-00 (1976); Remus Development Note 30.5.79.
10. Private communication, J. Prevost and G. Seite, DPHE, Saclay.

11. D. J. Holthuizen and D. Samyn, "FAMP-UA1/Software," CERN FAMP/82-10 (1982).

12. S. Cittolin and B. G. Taylor, "SUPER CAVIAR: Memory mapping the general-purpose microcomputer," Proceedings of Topical Conference on the Application of Microprocessors to High-Energy Physics Exp., Geneve, CERN 81-07 (1980).