CONF-781033--19

THE EVENT HANDLER
A FAST, PROGRAMMABLE, CAMAC-COUPLED DATA ACQUISITION INTERFACE

David C. Hensley*

Summary

The purpose of this paper is to describe the architecture and performance of the Event Handler, a fast, programmable data acquisition interface which is linked to and through CAMAC. The special features of this interface make it a powerful tool in implementing data acquisition systems for experiments in nuclear physics.

I. Philosophy--what is needed

The data acquisition needs of a large part of nuclear physics fit into an unfortunate space of CAMAC interfacing. On the one hand, it is clear that CAMAC provides a fairly adequate and even fertile mechanism for data acquisition and data acquisition interfacing. On the other hand, it is painfully clear that the major impact on CAMAC has been the demands of industry, and most of the readily available equipment reflect this reality. Nuclear physics style ADCs, TDCs, and like equipment in CAMAC are now appearing, but a module which might act as the brains of a data acquisiton interface has not really appeared. The ubiquitous micro-processor (μP) provides a locally smart crate, able to go through the appropriate motions for the desired data acquisition, able in fact to handle easily a portion of data acquisition. Unfortunately the current CAMAC based μP systems lack the necessary speed to handle the high rates and burst character of the data acquisition of many of the currently significant types of experiments in nuclear physics. Something at least an order of magnitude faster than the current μP systems is urgently required, though it will require only a small fraction of the processing capability of the μP.

Generally, high-rate multiparameter experiments scatter relevant information into a few of many detectors. A pressing need is to have a system which can rapidly (less than 10 μsec) decide if the current event is of possible interest and, if not, to restore the system immediately for further data acquisition. If the event is to be accepted, the system must be able to transmit "relevant" information at rates greater than 200 kHz (<5 μsec/word); "irrelevant" information could be ignored, i.e., not transmitted. Notice that no arithmetic capability is generally required. Information is transmitted in an untransformed form, though irrelevant numbers may have been omitted. All arithmetic capability resides with the downstream computer. A further quality of a system is that its interaction capability should be strongest on the experiment side of the interfacing. On the host computer side it need do little other than transmit data and recognize the availability of the computer. On the experiment side, however, the system must recognize the status of switches, of ADCs and other modules, and it must respond to any steering logic applied to it by the experimenter. That is, the system must be fairly capable on the experiment side of the interface, but it can be quite dumb on the computer side. Finally, to be generally usable, the system should be easily programmable and should be largely independent of the host computer.

II. The Event Handler

An "Event Handler" (EH) module has been developed which incorporates most of the philosophical considerations (prejudices) discussed above, and the experience from six months of using it in experiments has demonstrated its ability to go faster than a speeding bullet and to jump tall buildings with a single bound. The name "Event Handler" has been chosen because the primary purpose of the interface is to handle the equipment and information associated with experimental events.

It made sense to divide the EH into two components, a processor and an auxiliary controller (Aux), as shown in Figure 1. The processor is physically independent of CAMAC; a 36 line bus (24 data lines and 12 logic lines) connects it to the Aux. The Aux can be accessed by the crate as a slave module. It will respond with a true Q only if the EH is disabled; then the following functions are implemented:

F(0)A(0 or 1)	read processor memory or address
F(16)A(0 or 1)	write processor memory or address
F(24)A(0)	disable EH; Q=1, if achieved
F(26)A(0)	enable EH; Q should be false

These functions allow one to disable the EH, to load and verify a program, and to re-enable the EH, all from the host computer. The EH has an enable switch which overrides the Aux and protects the EH from erroneous crate requests to the Aux. When the Aux operates as an auxiliary crate controller, it accepts an NAF request from the processor, executes the NAF, and signals the processor that it is through. It will put its I/O register on the connecting data bus upon request of the processor. The Q-response of the current NAF operation is latched and available to the processor.

The processor is inactive if the EH is disabled; this requires both that the front-panel enable switch

*Oak Ridge National Laboratory, Oak Ridge, TN 37830. Operated by Union Carbide Corp. for U.S. Department of Energy, under contract W-7405-eng-26.

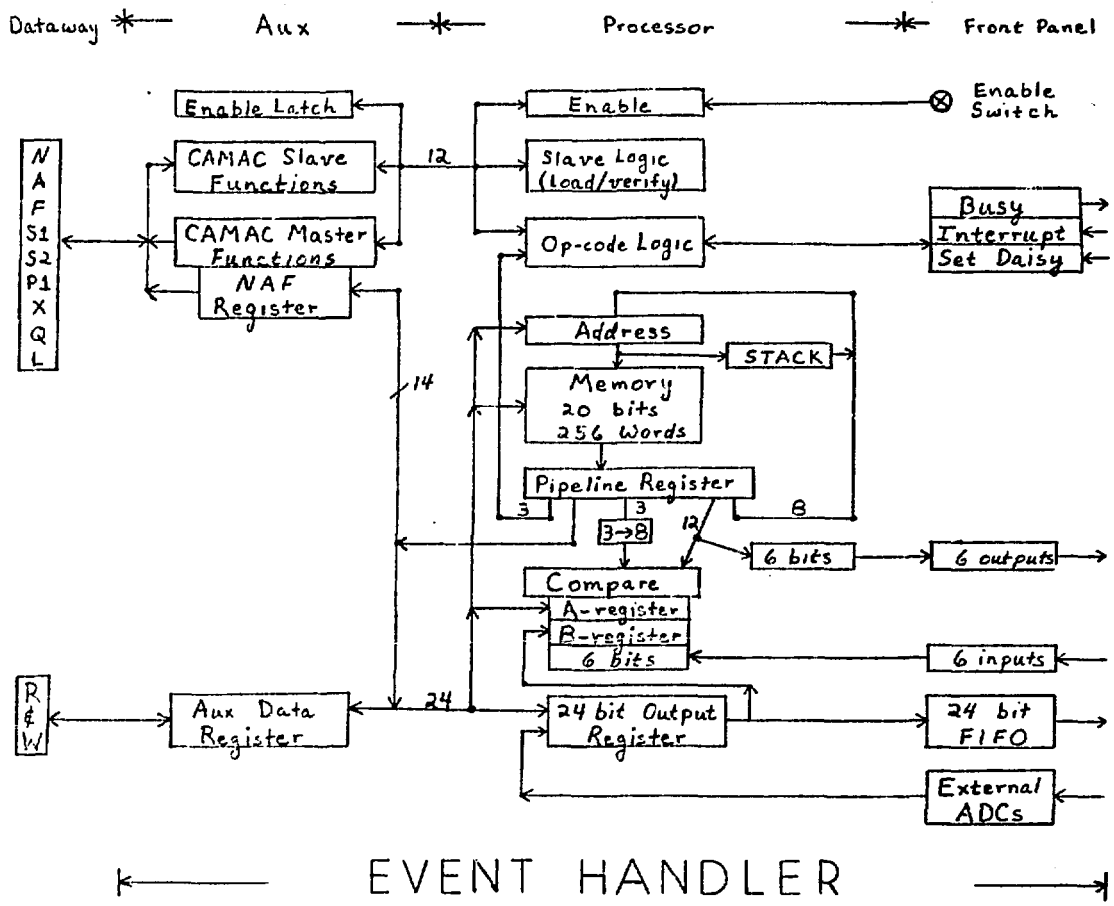

Figure 1. Schematic Layout of the Event Handler

is off and also that the enable latch has been reset in the Aux. When the processor is inactive, its memory and address may be set or read through the Aux. When the EH is enabled, the processor, after a 410 usec delay, begins to execute its program starting at the last address specified. Whenever any instruction takes more than 410 usec to execute, the processor automatically executes the next instruction. (This might happen if the output port were busy.)

The processor works on a pipeline scheme whereby the current instruction is clocked into a pipeline register for decoding and implementation while the address logic is fetching the next instruction. (See Figure 1.) Two 24 bit registers are available for saving the Aux data register and the output register, respectively, and these registers are available to the processor's bit-checking instructions. The processor can check the status of 6 front panel inputs and can drive 6 front panel outputs. Two other inputs initiate an interrupt or set an internal latch (DAISY), respectively.

The FIFO Output vs. CAMAC Output

The output of the EH has been made to resemble that of a 24 bit wide FIFO (First In, First Out buffer). There are essentially only two control lines: RECEIVER-AVAILABLE and SHIFT-IN. The fact that the EH looks like a FIFO means that it may be connected directly to a FIFO to provide buffering and time-derandomizing. If the data acquisition crate is remotely positioned, the output may go into a data transmitter. While, of course, the EH could use the CAMAC dataway for its data transfer, this would slow it up and could involve a CAMAC serial highway for data transfer, a possibly restrictive choice. The use of CAMAC for data output from the EH may become valuable when mixed event types are handled. In that case, the output could be steered through CAMAC to any of several devices as, for instance, to a scaler display module or to an LED Activity/Status panel. The FIFO port should be used, however, to handle any high-rate data transmission to the computer, when feasible.

External ADCs

The one poor design decision in the current EH concerns the handling of our external ADC system. Our current non-CAMAC ADC system dumps a specified number of 24 bit words (parameters) onto a data bus using three control lines. The decision was to bring the bus directly into the EH and mix it with the output from the EH. As it turns out, while this reduces

the logic overhead somewhat, it reduces significantly the ability of the EH to manage the data stream. Consequently, further improvements to the system will surely see the external ADCs brought in through a separate CAMAC module.

III. The Processor

General Op-Code Operation

The following general operations were incorporated into the processor. They fit comfortably into the architecture of the processor while permitting a fair degree of flexibility and power in the data acquisition configuration.

1) NOP -- this no-operation instruction uses one timing cycle; the next instruction is then executed.

2) DELAY -- the delay instruction has a duration of one timing cycle plus up to 4095 0.1 usec time intervals. This is used to specify the duration of an output pulse or to wait for a device to complete its task.

3) BRANCH -- unconditional jump to a specified address. The address may be specified in the pipeline register (normal branch) or may be fetched from a one-deep stack (return from subroutine). The current location may be stored in the stack (call subroutine).

4) LOAD -- this instruction stores information in the A or B registers or stores the pipeline register in the AUX I/O register. It can also set a 6 bit output register which drives the 6 front panel outputs shown in Figure 1.

5) SKIP -- this instruction will skip the next instruction if a specified condition is met, otherwise the next instruction is executed. The condition is specified by 12 bits given in the pipeline register. It compares this pattern against that of the (upper or lower) 12 bits of the A or B registers or of the 6 front panel inputs or of certain internal flag latches (e.g., CAMAC Q-response). It will skip if any (or none) of the 12 bits is set in the specified place.

6) NAF -- this instruction passes NAF, a CAMAC I/O request to the Aux Controller, where N is the slot number, A is the subaddress, and F is the function number. The Q-response is available for checking with the SKIP instruction.

7) OUT -- this instruction causes the pipeline register or the result of a CAMAC read request (Aux I/O register) to be loaded into the output register. At the same time, the instruction can request that the output register be transmitted. Further, the OUT instruction can load the 12 least significant bits of the Aux register into the 12 most significant bits of the output register. (A special feature which recognizes that most CAMAC ADCs have 12 or fewer output bits and which packs two ADCs into one transmission.)

8) INTERRUPT -- this psuedo-instruction generates a return-from-subroutine and store-currrent-position operation after receipt of a front panel pulse. All instructions except DELAY are interrupted at the end of their execution. The DELAY instruction is interrupted immediately. Normal response time (except during an NAF instruction) is less than 1 usec.

Detailed Hardware

The processor is driven by a 10 MHz clock and has a 20 bit wide by 256 word deep random access memory. The major components of the processor are shown in Figure 1. The beginning of a cycle clocks the memory into the pipeline register and at the same time increments the address. The next cycle starts typically 0.4 or 0.5 usec later, longer than the access time of the memory.

The op-code is 3 bits wide and is processed by a 3->8 decoder which selects 1 of 7 functions (2 of the outputs are trapped as NOPs). The schematic for the decoding is shown in Figure 2. The pipeline register is clocked at time zero (T0), and an extra bit of this register is used to enable the decoder. This ensures that the decoder has settled before its outputs are enabled. At time T300 (300 nsec later), the pipeline op-code is cleared and the decoder is disabled. Any function lasting longer than 300 nsecs must latch the op-code.

The cycle request and timing generation are shown in Figure 3. A cycle request sets a latch which will be cleared by the first timing pulse (T0). Note that T0 is not generated until after the request goes away. Each timing pulse is 100 nsec wide. The pipeline register is clocked by T0, and the op-code decoding appears midway between T0 and T100. Any output enabling is generally accomplished before T100. Then any register clocking or loading is done at time

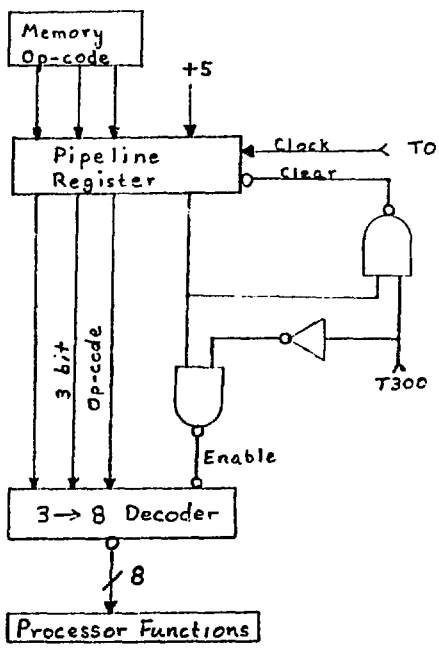

Figure 2. Op-code Decoding

T200, allowing at least 100 nsec of settling time. T200 is used to generate the cycle request for 3 of the functions. Since the request is honored only after it is taken away, T0 for the next cycle corresponds to T400 for the current cycle. Those functions generating an address change (BRANCH, SKIP) change the address at T200 and the next cycle begins at T600, allowing 400 nsec access time.

The interrupt operation, which is asynchronous with the timing of the processor, is handled as shown in Figure 3. An interrupt request from the front panel is latched into INTL; this will occur only on the front edge of the pulse and only if the latch is enabled. INTL will propagate to the second latch INTL2 only if the processor is currently generating a cycle request or if a DELAY instruction is currently being executed. INTL also holds off the propagation of cycle request. INTL2 is latched synchronously by the 10 MHz clock. (INTL2 is held off if REQUEST has already reached the shift register.) INTL2 prepares for a swap of the current address and the return address and enables INTL3. INTL3 sets 100 nsec later, swaps addresses, disables and clears the interrupt latch INTL, and generates a delayed cycle request. The request is delayed 400 nsec so that the memory can set up.

IV. Auxiliary Controller (Aux)

The Aux is currently compatible with a KS-3960 or an L2 serial crate controller. An "NAF" request (where NAF indicates slot number, subaddress, and function, respectively) from the processor stores N, A, and F in a special register and sets a NAF-BUSY latch. The Aux initiates a CAMAC cycle according to the status of an internal switch which indicates the kind of crate controller.

If the crate controller is a KS-3960 CC, the Aux generates a LAM and transfers NAF on the read lines in response to the F(1)A(12) inquiry from the 3960. At the end of this inquiry it drives/senses the I/O lines while the 3960 drives the NAF and other control lines. The Aux resets on the end of S2, except for a read cycle when it resets on the end of S1. (Thus on a read request the Aux takes a little over 1.2 μsec to finish, 0.6 μsec for the F(1)A(12) and 0.6 μsec to the end of S1.) The 3960 CC generates S2, of course, so that a proper cycle is seen by crate modules.

If the crate controller is an L2 serial crate controller (SCC), the Aux checks the status of the SCC. If the SCC is using the crate (SCC BUSY), the Aux waits until SCC-BUSY goes away or CRATE BUSY goes away. Then it initiates its own CAMAC cycle, driving the A, F, S1, S2, BUSY lines directly and driving the N line through the SCC, and it senses/drives the data lines as is appropriate. The Aux resets on the end of S2.

A request/grant protocol will be implemented next. Since most of the hardware for this protocol already exists as part of the SCC function, few additions should be required to affect this improvement.

V. General Applications

The crucial test of a programmable interface such as the Event Handler is its acutal application. My experience with the device indicates that it is nicely tailored for many-experiments.

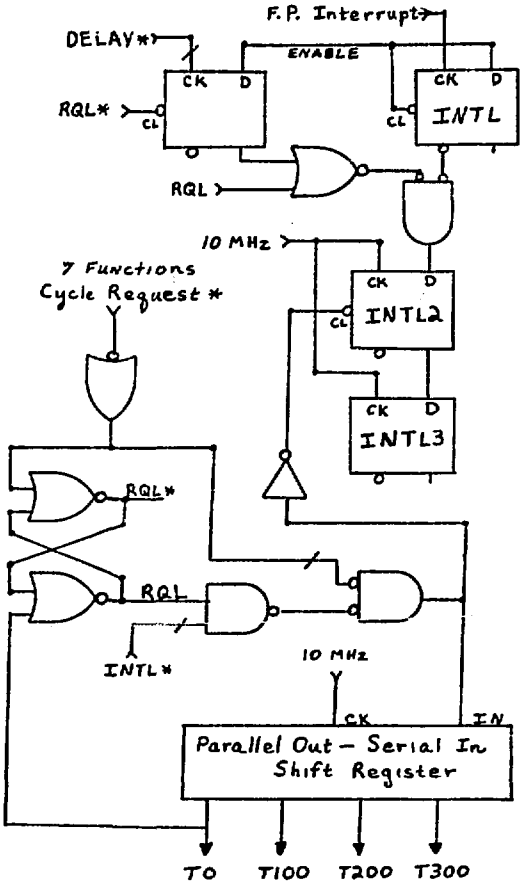

Figure 3.
Details of the Cycle Request, of the Cycle Timing Generator, and of the Interrupt Logic

Setting-up

This is a critical process for most multiparameter experiments. A CAMAC system, especially one not on-line with the host computer as during setup time, is generally a blind system. The EH can be programmed readily to execute any data acquisition sequence and to display various converted parameters on a local LED display or on a slave multichannel analyzer. Even though the input to an ADC can be checked on a scope, the conversion gain and lower cutoff can be determined only by running the ADC and displaying the results, and this is particularly true of a TDC (time-to-digital converter) which digitizes a time interval. It has proved to be very easy to program the EH to interrogate a 24-switch module, to read then the desired parameter, and to display or transmit the result. Or it can read a coincidence buffer and then read the parameters indicated by the buffer pattern, as it might during an experiment.

General Parameter List

The simplest (most straight-forward) application is to have the EH read and transmit a prescribed list

of modules. This it can do with an average speed of about 2 μsec per word. It can then clear the modules and wait for the next event. This is the same sort of cability that a programable crate controller might have, but it does not rely on Q-scan techniques.

Selective Parameter List

The most common aspect of multiparameter experiments is that few of the parameters within a given event will be relevant (i.e., meaningfully nonzero). If there are several master detectors, only one may have detected anything. If there are many slave detectors, few (or none) of them may have detected anything. In this case communications overhead can be greatly reduced, since the EH can first read and store a coincidence buffer (giving the relevant coincidence pattern) and then transmit only those parameters which the buffer pattern indicates are relevant. Since the host computer receives the buffer pattern also, it knows which and how many parameters to expect in this transmission. In many cases this information, in this culled form, is ready to list on magnetic tape (or disk), although the computer may still want to do some immediate coarse processing for on-line monitoring purposes. Front end dead-time should be reduced somewhat if fewer words are read and transmitted. This is especially true if the "irrelevant" data to be ignored were to come from a module with a long conversion time -- the system can proceed with the faster units and re-enable the data acquisition system much sooner.

"Crash" Mode

Because there is a relatively long digitizing time associated with many ADCs, especially CAMAC ADCs, it is often desirable to accept an event and start the ADCs digitizing and then, after deciding that the event is in fact not wanted, to stop and clear (crash) the ADCs and to start over again. This is the case when there is a high "singles" count rate but a low coincidence count rate. For example, with a "singles" input rate of 10 kHz and an ADC digitizing time of 100 μsec, the front end of the acquisition system has a deadtime over 50%. The host computer may also have trouble keeping up the high rate and this could add to the deadtime. Often, unfortunately, it is clumsy or impractical to decide ahead of time with logic circuits that an event is unwanted. Consequently the EH can be most valuable for reducing front end deadtime. As soon as an event is initiated, the EH reads the coincidence buffer and determines whether the pattern is potentially useful; e.g., that there were any slave detectors in coincidence, that there was more than one master detector. If not, the EH can stop and clear the ADCs and other equipment and restart the acquisition process -- this it can do comfortably in less than 10 μsec. Thus the front end deadtime will improve from over 50% dead to about 10% dead, and the desired coincidence throughput efficiency will nearly double. A detailed description will be given below.

Handling Mixed Events

Often experiments may be comprised of logically quite separate parts, each generating quite different data patterns. An example would be a set of LED stablized NaI detectors. Regular events would have interspersed in them calibration/stabilization events from the LED driver. Or it might be required that status reports from the equipment be periodically transmitted, reports on rates, temperatures, positions, scalers, etc. There are many ways the EH can handle these cases.

```
        BEGIN     What event am I
        TYPE 1    CALL A
                  STOP
        TYPE 2    CALL B
                  CALL C
                  STOP
        TYPE 3    CALL A
                  CALL C
                  STOP
        A         READ Experiment ADCs
                  READ TDCs
                  RETURN
        B         READ LED ADCs
                  RETURN
        C         READ SCALERS
                  READ DETECTOR ANGLES
                  RETURN
                  END
```

This schematic program shows a possible program flow--the logic steering could come from the front panel or could come from a latch module. The protocols for A, B, C are written only once in the program even though they are used more than once. This is especially valuable if the protocols are long or involved, since program space is reduced and debugging problems may be reduced. The subroutines could be any of the types listed above including the crash mode. Note finally that the information from the different sections need not be sent to the same place or even over the same path. Status information might be routed to a video display and stabilization information might be routed to a local stabilizing unit.

In-crate Tester

Since it can request any CAMAC function (NAF), the EH can be used as a device to drive a CAMAC module that needs testing. This can be very useful, particularly when the host computer is too busy to act as a testing instrument.

VI. Specific Application -- Software "Crash"

The simple program for the EH listed below is an example of a selective parameter list with a crash feature.

```
        (PROGRAM)                    (COMMENTS)
WAIT    SKIP ANY  FP,1       Is F.P. 1 on
        BRANCH  WAIT         No - no event yet
        LOAD  BUSY,ON        Yes -- inhibit the system
        NAF  BUFF,0,0        Read coincidence buffer
        LOAD  A,DATA         Store buffer in A-register
CHEK    SKIP ANY  A,77       Are any of 6 detectors set
        BRANCH  NONE         None set -- terminate
        OUT  DATA            Transmit buffer
DLAY    DELAY  1000          ADCs 100  sec to convert
        NAF  MAST,0,0        Read Master
        OUT  DATA            Transmit
X1      SKIP ANY  A,1        Is detector 1 set
        BRANCH  X2           No
        NAF  DET,0,0         Read 1
        OUT DATA             Transmit it
X2      SKIP ANY  A,2        Is detector 2 set
        BRU X3               No
        NAF  DET,1,0         Read 2
        .....
        .....                Check and read 3,4,5
        .....
```

```
X6    SKIP ANY  A,40      Is detector 6 set
      BRANCH  NONE       No
      NAF  DET,5,0       Read 6
      OUT  DATA          Transmit
NONE  NAF  MAST,...      Clear master ADC
      NAF  Det,...       Clear slave ADCs
      NAF  Buff,...      Clear coincidence buffer
      LOAD  EXT,1,ON     F.P. 1 on
      LOAD  EXT,1,OFF    1 off ~ 0.5 μsec clear pulse
      LOAD  BUSY,OFF     Turn system busy off
      BRANCH  WAIT
      END
```

1) The processor sits in a wait loop while it polls front panel input-1. This is a simple method to recognize that an event cycle has been initiated. Maximum response time for this method should be ~ 1.3 μsec. BUSY should appear within ~ 1.5 μsec from setting of input-1.

2) Next, the coincidence buffer module is read and loaded into the A-register. (It could also be transmitted at this time.)

3) At CHEK, the A-register is checked to see if any slave detector is set. If none are, the processor jumps to NONE where it clears and starts again.

4) If there are relevant data, the EH transmits the buffer so the host computer will know what is going on.

5) The 100 μsec delay at DLAY is arbitrary. One could do a Q-test on the respective ADCs or could sense an external conversion-complete line. A good choice here can significantly reduce the waiting time.

6) The master detector is then read and transmitted.

7) Beinning with X1 through X6 the appropriate bit of the A-register (buffer) is checked. If the bit is set, the (slave) ADC is read and transmitted. If the bit is not set, the next bit is checked, on through the last bit. It takes ~ 1.3 μsec to check a bit and jump to the next; it takes ~ 2.6 μsec to check, read, and transmit.

8) The NONE section clears, through CAMAC, the buffer, the master ADC, and the slave ADCs. Then it generates an ~ 0.5 μsec external pulse to clear any external modules. Finally it turns the BUSY line off and goes to wait for the next event.

If the processor takes the "BRANCH NONE" jump at CHEK, the full time from sensing the beginning of event to clearing of BUSY is about 9 μsec. This can be reduced to about 5 μsec if the modules can be cleared via the front panel rather than through the crate.

VII. The Past, The Future

The Past

The Event Handler has been in use for approximately six months and has handled an interesting variety of experiments, mainly many parameter experiments, particularly those requiring CAMAC ADCs. In all of these experiments we have chosen to build the steering logic for the experiments around a gated latch (coincidence buffer) module. This module can record the firing pattern of the master and slave detectors, and the EH determines from the pattern which if any detectors to read. The host computer also uses the gated latch pattern to determine what words have been received. I shall describe just two of the many experiments to illustrate the current application of the EH. (Note, a detector telescope produces two signals, one identifying the particle, the other giving a measure of its energy. TDCs and TACs measure time intervals.)

Experiment-A (31 parameters)

2	Heavy Ion Telescopes (Masters)
8	Light Ion Telescopes (Slaves)
8	TDCs
2	TACs
1	Gated Latch

Experiment-B (21 parameters)

1	Electron detector (Master)
1	Magnet current
1	X-ray detector (Master)
3	NaI detectors (Masters)
9	NaI detectors (Slaves)
5	TDCs
1	Gated Latch

Experiment-A was run in a selective list mode. If either master telescope fired, all relevant words were read and transmitted. Most of the time only one master telescope and no slaves would fire, so only the telescope, its TAC, and the gated latch would be read and transmitted -- 4 parameters instead of 31. Also, the slave ADCs were slow, so the EH would stop and clear them as soon as it saw they were not needed. This cut the deadtime by about a factor of two.

Experiment-B was more complicated. If either the electron or x-ray detectors fired, the full event was taken. However if one (or more) of the master NaI detectors fired but no slave detector fired, then the system was "crashed" (stopped and cleared). This was very important as the NaI detectors counted at a combined rate of 6 kHz solely from room background; if the background contribution had not been quickly rejected the deadtime would have been > 60%. By crashing unwanted events, the EH held the deadtime under 10% and increased the effective counting rate by a factor of two.

The Future

Experience with the current Event Handler indicates that certain new features and improvements would be desirable, and most of these listed below are planned for the next model of the EH.

1) The Aux needs a Request/Grant protocol (and it would help if serial crate controllers honored this protocol).

2) The processor memory will be expanded to at least 1024 words and at least 4 levels of subroutines will be implemented. At the same time, the interrupt vector will be kept separate from the subroutine return stack.

3) The bit checking instructions will include an "exact" check, that is, whether a pattern of on-bits matches exactly those of the designated register. An instruction checking the number of bits on may also be implemented.

4) Four save-and-check registers, instead of two, will be implemented, permitting a check of up to 96 bits.

5) An "A" counter for NAF requests will be implemented so that A-scans can be easily implemented.

6) N and A will be merged into the answer from an NAF request, if desired, so that the host computer can quickly identify the source of a word.

7) The External ADC port will be dropped and will be implemented through a separate CAMAC module.

VIII. Conclusions

The Event Handler satisfies nicely an apparent lack in current data acquisition equipment. This CAMAC based programmable interface is much faster than CAMAC μP systems and is better geared for a wide variety of experiments in nuclear physics. Because it is programmable and already linked to CAMAC, it is much more adaptable and versatile than hard wire logic for event managing, particularly if event managing includes the manipulation of CAMAC modules. Because it is essentially totally independent of the host computer, the EH can be immediately implemented into any system possessing a CAMAC interface. It is clear that a fair fraction of the multiparameter experiments at the Holifield Heavy Ion Research Facility (and possibly several other laboratories) will be built around the Event Handler.

I am deeply indebted to N. B. Hickman for his heroic efforts in rapidly and artistically fabricating the prototype Event Handler discussed in this paper.