CONF-790549--7

MASTER

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

David C. Hensley*

Summary

The purpose of this paper is to describe the architecture of the Event Handler II, 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.

Philosophy--what is needed

Nuclear physics style ADCs, TDCs, and like equipment in CAMAC are now appearing, but a module which might act as the brains of a general, but fast, 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; 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 of other modules, and it must respond to any steering logic applied to it by the experimenter. Finally, to be generally usable, the system should be easily programmable and should be largely independent of the host computer.

An "Event Handler" (EH) module has been developed[1] which incorporates most of the philosophical considerations (prejudices) discussed above, and a year's highly successful experience with the device has demonstrated its power, flexibility, and general ease of use. The Event Handler II is the next model of this device and includes an enhanced capability for sparse data scans and transmission verification. 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.

Overview of the Event Handler II

The Auxiliary Controller (Aux)

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 data and logic bus 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)	read processor memory
F(16)A(0 or 1)	write processor memory or address
F(24)A(0)	disable EH; Q=1, if achieved
F(24)A(1 or 2)	set PAUSE or set PAUSE/HALT
F(26)A(0)	enable EH; Q should be false
F(26)A(1 or 2)	reset PAUSE or reset PAUSE/HALT

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. 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

The processor is inactive if the EH is disabled; this requires both that the front-panel enable switch 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 accessed through the Aux. When the EH is enabled, the processor, after a 410 μsec delay, begins to execute its program starting at the last address specified.

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. The processor is driven by a 10 MHz clock and has a 24 bit wide random access memory. The beginning of a cycle clocks the memory into the pipeline register and at the same time

*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.

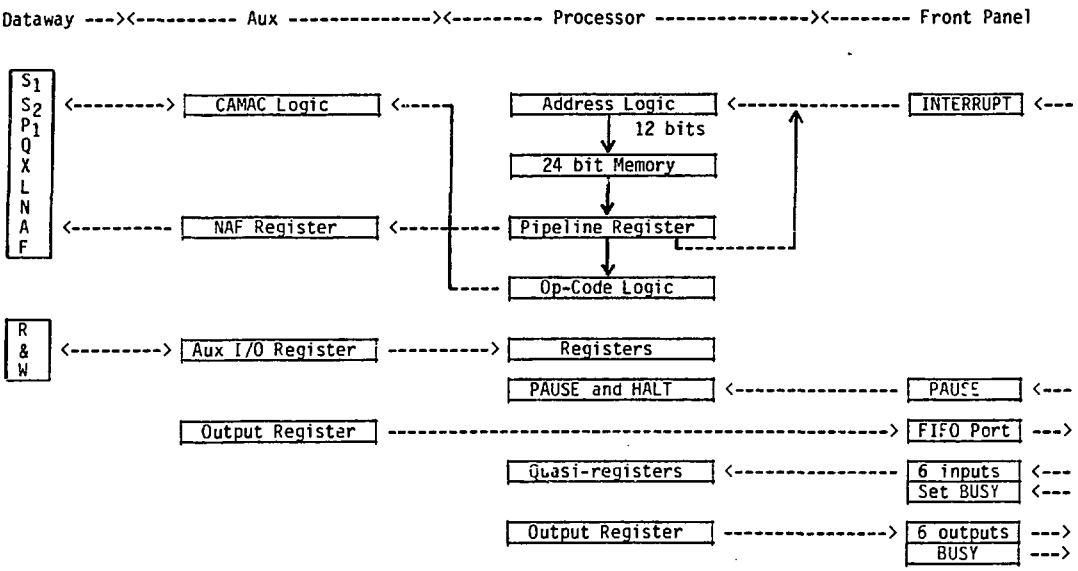

Figure 1. Schematic Layout of the Event Handler II

increments the address. The next cycle starts typically 0.4 or 0.5 μsec later, longer than the access time of the memory. Four 24-bit registers are available for saving the Aux I/O register, and these registers are available to the processor's bit-checking instructions.

The FIFO 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, the FIFO port should be used to handle any high-rate data transmission to the computer, when feasible.

External Control Logic

The processor has several sources of external control logic. It can sense and test 6 front panel (FP) inputs with the SKIP instruction. This allows the experimenter to provide steering logic. In addition, there are three FP inputs; SET BUSY, INTERRUPT, and PAUSE. The BUSY latch drives a FP BUSY OUTPUT and can be sensed/tested by the processor. An INTERRUPT signal will generate a standard processor interrupt, if enabled. The processor can also drive any of 6 FP outputs.

Two basic control lines are used with the FIFO output; RECEIVER-AVAILABLE, to indicate that the downstream device is available, and SHIFT-IN, to indicate that a word is being transmitted.

There are two additional controls, PAUSE and HALT, which are recognized by the processor. HALT indicates that the downstream control computer is through; it will accept no more data. PAUSE is an indication that processing should terminate as soon as possible--usually at the end of the current cycle. Generally the control computer will indicate PAUSE, wait a bit for everything to clear out, and then indicate HALT. The control computer can set PAUSE and/or HALT through the AUX or through the FIFO. PAUSE is, in addition, generated by the FIFO when it is 3/4 full (for example) or by the experimenter through the FP input when he wishes to shut off the data acquisition front end.

The Processor Instructions

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-μsec time intervals. This is used to specify the duration of an output pulse or to wait for a device to complete its task.

3) JUMP, JSUB, RETURN -- unconditional jump to a specified address. The address may be specified in the pipeline register (normal JUMP) or may be fetched from a four-deep stack (RETURN). The current location may be stored in the stack (JSUB).

4) LOAD -- this instruction loads the A through D registers with the contents of 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 SKIP instruction first masks (ANDs) the designated register with the pipeline register. The instruction can skip if the result has any or no bits set, or if the bits are the same as those in the mask. Or the result can be compared with a number-register and the instruction can then skip if the result is <, =, or > the number.

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) XMIT -- this instruction causes the 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.

8) INTERRUPT -- this psuedo-instruction generates a JSUB REGISTER instruction upon 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 is less than 1 μsec.

Op-code Special Features

In order to facilitate sparse data scans, the JUMP+FSBR instruction has been implemented to allow a JUMP to a specified location plus a number (from 0 to 12) from the FSBR (First Significant Bit Register) which indicates the position of the first non-zero masked bit in a particular register.

```
     LOAD    FSBR,A,0770
```

will set FSBR to the position of the first significant bit common to the A-register and the mask (0770, in octal). FSBR is set to zero if there is no such bit. For example;

BITS	1	2	3	4	5	6	7	8	9	10	11	12
A	1	0	1	0	0	1	1	1	0	1	1	1
MASK	0	0	0	1	1	1	1	1	1	0	0	0

+ FSBR = 6

JUMP 10+FSBR will jump to 16.

In order to facilitate multiplicity measurements, an 8 bit multiplicity Adder is implemented:

```
NOP     MULTIPLICITY   Clear the Adder
LOAD    MULT,A,MASK    Count bits in mask
LOAD    MULT,UA,MASK   Add to MULT-Register
```

The MULT-register now contains the count of the masked bits in the 24 bit A+UA register. This register can be checked with the SKIP instruction.

The NAF instruction can reuse the previous N and F but increment A.

```
NAF     1,2,0
NAF     INCREMENT  -->  NAF 1,3,0
```

The A in the NAF instruction can be tested with the SKIP instruction for looping purposes.

XLOAD moves the datum in the AUX I/O register to the output register but does not transmit it.

XMIT moves the datum and transmit it.

XUP moves the lower 12 bits of the AUX register to the upper 12 bits of the output register and transmits the result.

XLOAD+NA or XMIT+NA move the 14 lowest bits of the AUX register to the output register. Bits 15 through 23 are replaced by N and A, and bit 24 is set to zero.

IF two words are read, ADC1 and ADC2, respectively, the following transmissions are possible:

```
(1)  (ADC1)          XMIT
     (ADC2)          XMIT
(2)  (ADC2,ADC1)     XLOAD, then XUP
(3)  (NA1,ADC1)      XMIT+NA
     (NA2,ADC2)      XMIT+NA
```

Transmission (2) simply recognizes that most CAMAC ADCs use no more than 12 bits and packs the two words into one 24-bit transmission. Finally, either the NAF or NAF INCREMENT instruction may be combined with XMIT, XLOAD, and XUP and with NA, if desired.

General Applications

It will be assumed that 96 ADCs are to be read and that they are packed 16 per module in sequential slots.

Simple data list transmission (96 instructions)

```
(PROGRAM)                (COMMENTS)
NAF+XMIT   1,0,0         1st in slot 1
NAF+XMIT   1,1,0
...
NAF+XMIT   1,15,0        last in slot 1
NAF+XMIT   2,0,0         1st in slot 2
...
NAF+XMIT   6,15,0        last in slot 6
STOP
```

"Do-Loop" data list transmission (16 instructions)

```
NAF+XMIT   1,0,0
JSUB       XT
NAF+XMIT   2,0,0
JSUB       XT
...
NAF+XMIT   6,0,0
JSUB   XT
STOP
```

```
XT    NAF+XMIT   INCREMENT   subroutine XT
      SKIP       NAF.NE.15   check if A=15
      RETURN                 yes, it is
      JUMP   XT
```

Q-scan data list transmission (18 instructions)

```
      NAF    1,0,0
      JSUB   XT
      ...
      NAF    6,15,0
      JSUB   XT
      STOP
XT    SKIP   Q.NOT.1        if bad Q, don'transmit
      XMIT+NA               Tag output with NA
      SKIP   NAF.NE.15
      RETURN
      NAF    INCREMENT
      JUMP   XT
```

Sparse Data Scans

Two cases will be treated for the two modes of sparse data scans. It is assumed that the pattern of relevant data is stored as 96 bits in the A, B, C, and D registers. This is the sort of information that is generally stored in a coincidence buffer. Each bit in the buffer is set if its respective detector fired during the gating interval. Only if a bit is on will the corresponding ADC be transmitted. For the case of 2 ADCs per detector (i.e., per bit), the second set of ADCs are assumed to be in modules 7-12. Simple examples of 2 ADCs are 2-element detectors or analog and time measurements for each detector.

SKIP data scans

(One ADC transferred per bit) (192 instructions)

```
      (PROGRAM)              (COMMENTS)
      SKIP   A.NOT.1
      NAF+XMIT+NA   1,0,0
      SKIP   A.NOT.2         2nd bit
      NAF+XMIT+NA   1,1,0
      ...
      SKIP   A.NOT.400D      12th bit
      NAF+XMIT+NA   1,11,0
      SKIP   UA.NOT.1        13th bit
      ...
      SKIP   UD.NOT.4000     96th bit
      NAF+XMIT+NA   6,15,0
      STOP
```

(Two ADCs transferred per bit) (384 instructions)

```
      SKIP   A.NOT.1
      JUMP   X2
      NAF+XMIT+NA   1,0,0
      NAF+SMIT+NA   7,0,0
X2    SKIP   A.NOT.2
      JUMP   X3
      ...
X96   SKIP   UD.NOT.4000
      JUMP   STOP
      NAF+XMIT+NA   6,12,0
      NAF+XMIT+NA   12,12,0
STOP  STOP
```

JUMP+FSBR data scans (384 instructions)

(One ADC transferred per bit)

```
      LOAD   FSBR,A,7777
      JUMP   TAB1+FSBR
TAB1  JUMP   DOUA
      JUMP   X1
      JUMP   X2
      ...
      JUMP   X12
X1    NAF+XMIT+NA   1,0,0    1st word
      LOAD   FSBR,A,7776     omit bit 1
      JUMP   TAB1+FSBR
      ...
X11   NAF+XMIT+NA   1,10,0
      SKIP   A.ANY.4000
X12   NAF+XMIT+NA   1,11,0
DOUA  LOAD   FSBR,UA,7777
      JUMP   TAB2+FSBR
TAB2  JUMP   DOB
      JUMP   X13
      JUMP   X14
      ...
      JUMP   X24
X13   NAF+XMIT+NA   1,12,0
      LOAD   FSBR,UA,7776    omit bit 13
      JUMP   TAB2+FSBR
      ...
X23   NAF+XMIT+NA   2,6,0
      SKIP   UA.ANY.4000
X24   NAF+XMIT+NA   2,7,0
      ...
      STOP
```

(Two ADCs transferred per bit) (488 instructions)

```
      LOAD   FSBR,A,7777
      JUMP   TAB1+FSBR
TABI  JUMP   DOUA
      JUMP   X1
      ...
X1    NAF+MIT+NA   1,0,0
      LOAD   FSBR,A,776      omit bit 1
      NAF+MIT+NA   7,0,0
      JUMP   TAB1+FSBR
      ...
      STOP
```

Conclusions

The desire to reduce both the front-end dead time and the downstream communications overhead has motivated the particular architecture of the Event Handler. The currently available ADCs and TDCs densely packed (8 or more) in CAMAC modules have conversion times from 50 to 200 μsecs. It is desirable to keep the scan time comparable to or less than this conversion time so that the front-end dead time will not be dominated by the speed of the Event Handler. Table I compares the sparse-data scan times for 96 detectors for the three main modes available to the EH when 100%, 30%, and 0% of the detectors are significant. When one ADC per bit is transmitted, the SKIP and JUMP modes are equally fast when about 29% of the ADCs are significant. For two ADCs, equal speed occurs when about 63% of the ADCs are significant. The Q-scan technique has not been especially optimized in the EH and, consequently, does more poorly than it might; but, in any case, the Q-scan

technique must be inherently slower since it relies on a CAMAC data-way transaction to make its decision of relevance.

For scans of very sparse data involving many parameters the JUMP-mode is clearly superior and very powerful. For smaller numbers of parameters the SKIP-mode will often be found to be superior.

Table I

Time (μsec) for 96 detector scans

	1 ADC Time			2 ADCs Time		
	100%	30%	0%	100%	30%	0%
SKIP MODE	115	79	67	230	145	106
JUMP+FSBR	235	81	14	283	95	14
Q-SCAN	173	166	163	317	236	202
SKIP=JUMP		29%			63%	

Even though special attention has been given in this paper to the problem of sparse data scans, it is clear that the overall features of the Event Handler satisfy nicely a current need for a general, fast, data acquisition equipment. This CAMAC based programmable interface is much faster than CAMAC μP systems and is much better suited 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-wired 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 which supports CAMAC. Such a device must be a major part of most complex multiparameter nuclear physics experiments.

Reference

1. D. C. Hensley, "The Event Handler - A Fast, Programmable, CAMAC-coupled Data Acquisition Interface," IEEE Trans. Nucl. Sci. NS-26, No. 1, 710-716 (Feb. 1979).