Readout Issues for the New Waveform Digitizer

Edward Kearns
Department of Physics, Boston University

February 20, 1994 (revised 4/10/94)

Abstract

This MACRO note reviews issues regarding readout of the new 200 MSPS waveform digitizer (WFD). The work was done during my trip to LNGS during January 1994. I verified the general feasibility of WFD readout using the VIC interface and made some studies of readout speed and data size.

1 Introduction

The new waveform digitizer operates by running constantly and writing data into memory storage for each channel. The WFD hardware fills memory backwards: from high address values to low address values, so that readout may take place forwards- with incrementing addresses. Each wfd cycle fills 32 bits in the ADC memory bank (address offset 0x0000 to 0x7fff) and 32 bits in the time-word/discriminator memory bank (offset 0x8000 from the data in the ADC memory bank). The memory is continuously overwritten by cycling through the address-space of each channel in a circular fashion: when the digitizer reaches the beginning (offset 0x0000), the address pointer wraps around and continues again from the end of the address-space (offset 0x7ffc). When the WFD is stopped by an external trigger signal, one executes a special control function to determine the last address (STOP ADDRESS) to which data has been stored. Then one reads forwards, beginning at that address until sufficient data has been read-out. If the STOP ADDRESS occurs near the end of the address space (near 0x7ffc), one will have to read data until the end of channel-memory is reached and jump to the beginning of channel-memory to finish reading.

For our purposes, the amount of data to readout is most usefully expressed in units of time; however, due to hardware zero suppression it is non-trivial to relate a specific amount of time to a quantity of memory storage.

The data acquisition group has provided a variety of CAMAC-list based readout commands [1]. For example, using `VME_RD_EX`, one may read a fixed amount of memory beginning at a fixed memory location. Although these commands are useful for WFD debugging [2] and for reading from other VME modules, they do not satisfy the WFD readout requirements as

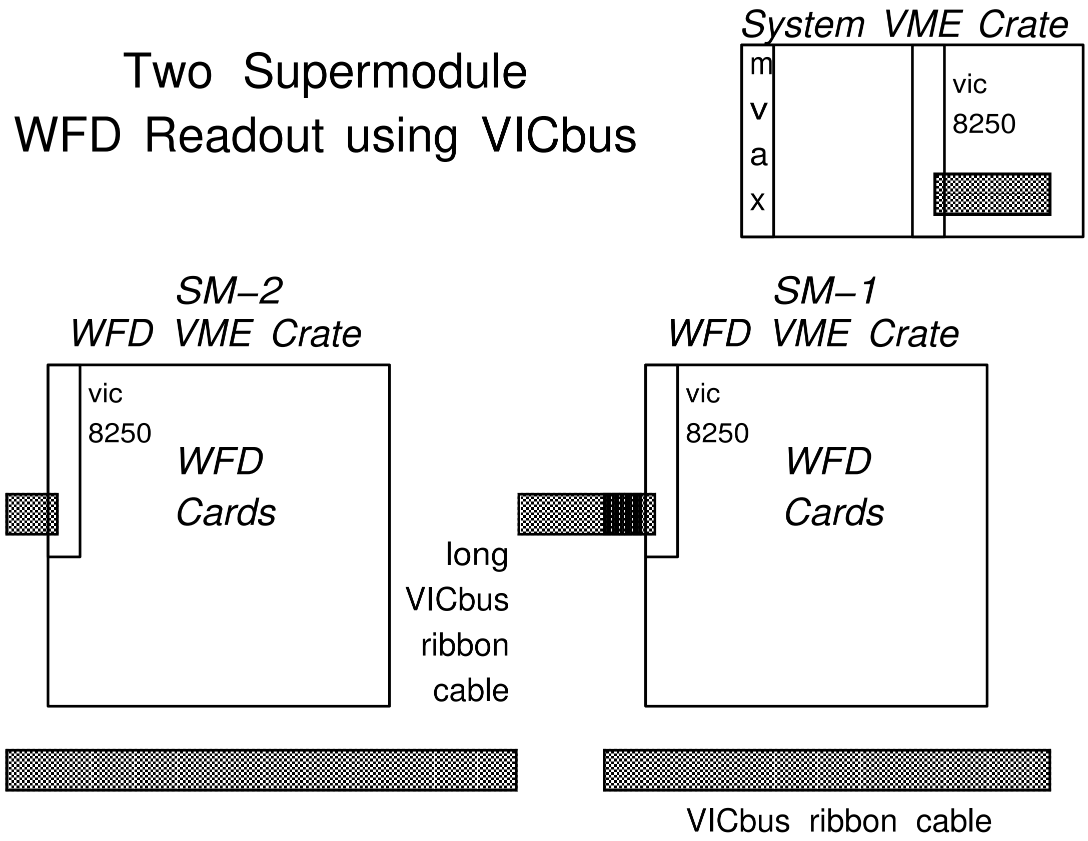


Figure 1: VICbus configuration for WFD readout. For two supermodules per μVaX, we will daisy chain the VICbus from the WFD crate nearest the system crate.

described above. There is no obvious provision for executing the STOP ADDRESS control function and reading-out based on the result. In addition, it is very desirable to specify an amount of time rather than a number of bytes. I have come up with working solutions to these issues, which will be described in this note. Of course these algorithms are subject to refinement and even possibly very different alternatives; but it was important to prove the feasibility of the system before continuing further with the hardware production and purchase of readout modules.

2 DAQ Configurations

We have discussed two basic schemes for reading out the WFD cards. The planned scheme is to use the same μVaX and data acquisition software as is used by the rest of the MACRO equipments. This requires some communication between the system VME crate containing the KAV30 μVaX and the VME crate containing the WFD. This is accomplished by a Vertical Inter-Crate bus or VICbus [3]. The modules are built by Creative Electronic Systems (CES). The bus is constructed with a VIC-8250 [4] in the DAQ VME crate and a double ribbon cable connecting it to a VIC 8250 in the WFD VME crate. These modules may be daisy chained, so for two supermodules per μVaX, we will configure the system as shown in figure 1.

The second scheme we have discussed is one where we have some sort of "local intelligence" in the WFD VME crate. In this case, the readout algorithm would be executed by a local processor, such as the Motorola MVME-167 (68040) that I use for testing. The data still needs to be transmitted to the DAQ; this has not been solved, but there are several possibilities. For example, the VIC configuration discussed above may still be used, but now with a simple readout algorithm. An alternative is using Ethernet and IP sockets; Jiangtao Hong has made some preliminary studies of how to do this.

We are proceeding with the first scheme because it will be ready the fastest, and it is less expensive. Also, it is unproven that the second scheme will have any net advantage in readout speed. But the using a local processor is still viable as a future upgrade.

For the tests described below, I had two DAQ configurations operating simultaneously. I used the test μVaX (MVMA10) running the mini-data-acquisition system connected through a VICbus as described above. In the WFD VME crate, I also had the MVME-167 computer running the OS9 realtime operating system, which I talked to via a serial terminal line. This allowed me to stop the WFD card and read data using my debugged programs and compare it to the same data as read by the algorithms transported to VaXELN on the μVaX.

3 Equipment Definition

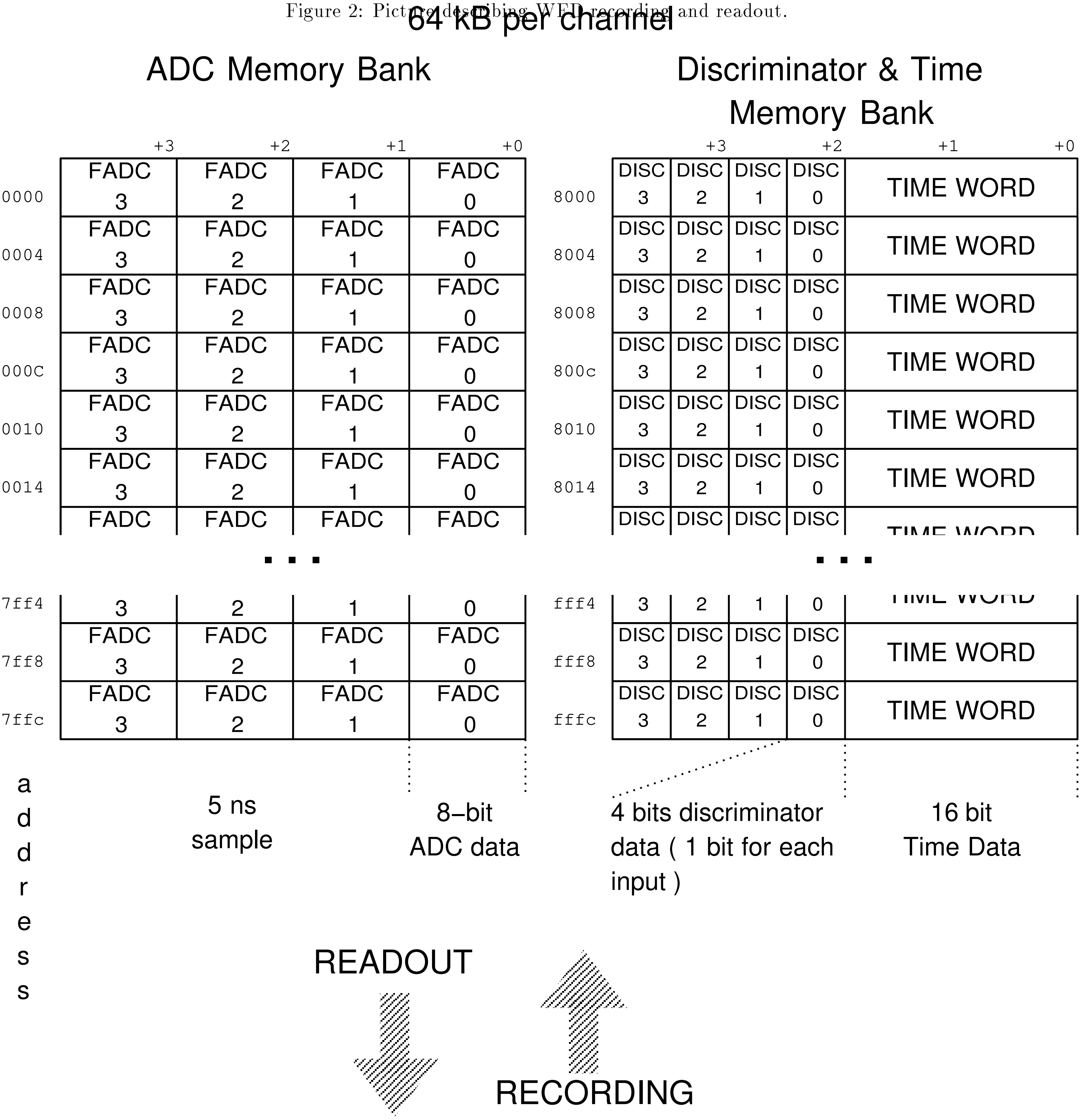
An "equipment", in MACRO terminology, is the structure of the raw data and corresponds to the sequence of readout commands present in the CAMAC list. When an equipment is configured for real data acquisition, it requires decoding and diagnostic software that matches that data structure. Sometimes, we are lax in how we set this up and it becomes necessary to handle special cases when words are added to an equipment. I have attempted to anticipate future modifications by beginning with a flexible structure, at the expense of a few bytes of overhead.

I am proposing a general data structure *type* for the new waveform digitizer. It can handle any number of WFD channels and has a header which may be expanded to include special words such as pattern registers. We cannot yet anticipate exactly what will go in the header; but by allocating space for it we can provide low level decoding routines which are guaranteed to read this WFD equipment type without getting lost or, more importantly, requiring a priori knowledge of exactly how many channels and how much data per channel is being readout for a given trigger.

First, let me remind you of the WFD data structure and introduce some terminology. See figure 2 for a graphic illustration, and refer to the WFD VMEbus Specification memo [5] for more about the data structure and control operations. Each WFD card has four channels with 64 kB of memory storage. This memory is divided into two sections: the first 32 kB contains 8-bit ADC samples; the second 32 kB contains 16-bit time words and 4-bit discriminator words. One WFD "cycle" consists of a total of 64-bits of information. Although it is feasible to readout all of the time/discriminator data at once followed by all the ADC data, the algorithms described in this report read the data by switching back and forth between the two 32 kB memory banks.

The equipment definition is shown in table 3. It starts with an integer that contains the

Figure 2: Picture describing WFD recording and readout.



NHEAD	number of header words
HEAD(1)	first header word
...	more header words
HEAD(NHEAD-1)	(n-1)st header word
NCHAN	number of wfd channels
CHADDR 1 TSTART 1 ...	first channel of wfd data (see right)
CHADDR 2 ...	second channel of wfd data
... ...	more channels of wfd data
CHADDR NCHAN ...	last channel of wfd data

⟵ 32 bits ⟶

CHADDR	address of wfd channel
TSTART	tics from STOP to first data
NDATA	amount of wfd data (32 bits)
ADC(1) TIM(1)	4 bytes of adc data time + disc words
ADC(2) TIM(2)	4 bytes of adc data time + disc words
...	more wfd data
ADC(NDATA) TIM(NDATA)	4 bytes of adc data time + disc words

⟵ 32 bits ⟶

Table 1: General data structure for WFD-type equipment. The left table represents the overall structure of the equipment, consisting of a header followed by NCHAN channels of WFD data. The details of the header words will be defined later. The right table describes the structure of a single channel of WFD data.

length of the header section. The minimum header length is one, in which case the only word in the header is the number of WFD channels (NCHAN). NCHAN should always be the last word in the header, so that decoding routines can work without modification. Additional data, such as a pattern register, should be inserted between the header-length word and the number-of-channels word.

Following the header are the WFD channel data. These are also structured, consisting of a channel-id, a t_0 word (TSTART) and the number-of-cycles of WFD data. The WFD data is 64 bits wide and matches the structure explained in section 5 and figure 2.

TSTART contains the number of WFD clock tics between the STOP signal and the first WFD data cycle. For slow monopole triggers, we will save the entire time window from $t = 0$ (the STOP signal) back to $t = -1$ millisecond. In this case, TSTART = 0. For exclusive muon triggers, we wish to save only a small time-window. But we will still delay the STOP signal for about 1 msec [6]. Assume we decide to save 10 μsec around the muon. In this case, we will readout a window from approximately $-.99$ msec to -1 msec, but to account for zero suppression, the exact time of the first cycle saved will be recorded by the readout algorithm. This readout code is a small variation on the `WFD_RDTIM` algorithm; it has not been written and tested yet, but I believe it will be a straightforward modification.

Sample FORTRAN decoding of this equipment structure may be found in SERVER_LIB.FOR in VXMACB::[BOSTONPUB.KEARNS.COM]. These routines eventually will be migrated to the official version of the mini-DAQ.

4 CAMAC List Commands

The code developed for this report resides on VXMACB in [BOSTONPUB.KEARNS.PAS]. This directory contains selected source code from [MACROELN.VE3_1] for modification. To add the new CAMAC list commands, I have modified CAMACPROCEDURE.PAS to add the command names to the list of valid commands, and CADRIVER.PAS to add new readout code to the Pascal case statement driven by this list of commands. Obviously, this directory is a temporary repository for current work; the reader should expect the exact details of this and other code to be changing not long after the release of this report.

Below is a list of the relevant VME commands that may be placed in a CAMAC-list. For completeness, I am including the routines provided by our Italian DAQ software experts [7], as well as an anticipated future command that saves the data from a time-window where one also specifies the starting time of the time-window. The VIC hardware is configured similar to CAMAC branches, with branch and crate number determined by thumb-wheels on the VIC 8250 modules. The address is specified as a hex constant (although `WFD_RD*` will only use this argument as a base address for calculating the actual STOP ADDRESS).

- `VME_RD_ST branch, crate, vme_address, ndata`
  Read ndata from vme_address using D16 transfers.
- `VME_RD_EX branch, crate, vme_address, ndata`
  Read ndata from vme_address using D32 transfers.
- `VME_WR_ST branch, crate, vme_address, data`
  Write single word (data) to vme_address using D16 transfers.
- `VME_WR_EX branch, crate, vme_address, data`
  Write single word (data) to vme_address using D32 transfers.
- `WFD_RDMEM branch, crate, wfd_base_addr, ndata`
  Read ndata from channel corresponding to wfd_base_addr, starting at STOP ADDRESS using D32 transfers.
- `WFD_RDTIM branch, crate, wfd_base_addr, ntics`
  Read time window of length ntics from channel corresponding to wfd_base_addr, starting at STOP ADDRESS using D32 transfers.
- `WFD_RDWIN branch, crate, wfd_base_addr, ntics, tic0`
  Read time window of length ntics from channel corresponding to wfd_base_addr, starting tic0 after the STOP ADDRESS, using D32 transfers.

To place constants, such as the number-of-channels word, in the equipment, I also added a CAMAC list command, `WRITEHEX`, which takes as a single argument a hexadecimal constant that will be written to the data (occupying 32 bits). See the example CAMAC list (appendix A) to see how this is used to construct the above data structure. Note! It is the responsibility of the CAMAC list creator to make the number-of-header-words and number-of-channels match the number of CAMAC list commands in the equipment. Although the equipment length will always be calculated correctly from the individual CAMAC list commands, decoding software that assumes this structure may get lost. But that is generally true for any equipment definition that changes.

5 WFD Readout Algorithm

Appendix B contains the VaXELN Pascal for `WFD_RDTIM` that has been incorporated into the mini-DAQ system. Note! This code will certainly be different in the final version. There are several things I plan to try to optimize in the source code (concentrating on the readout loops), but overall I do not expect a large change in speed. This can be seen from the relative transfer rates for the simple readout compared to the time-window readout as discussed in section 7.

I will describe in words the readout algorithm. Actual VME operations are marked with a letter and may be cross referenced with the code in the appendices. The first step in the readout algorithm is to determine which address to start readout from. This is done by **(A)** setting bit 4 of the WFD control register and then **(B)** reading from any address in the WFD channel. When done, **(C)** set the control register back to the default configuration. The datum read during **(B)** contains the lower 16 bits of the address of the WFD data cycle that was digitizing when the STOP signal occurred. Using this datum, one constructs address pointers to the data in the ADC section and time/discriminator section of memory.

To conform to the data structure described in section 3, we record the address of the channel we are reading from, which acts like a channel id. We also record t_0 (TSTART) and the amount of data that follows (NDATA). The algorithms in this report begin saving data at $t = 0$. Future algorithms will save only a small window of data and will fill TSTART when t_0 is determined by reading through the data. NDATA is calculated after all of the data has been readout.

The rest of the readout consists of a loop over the WFD memory, beginning from the STOP ADDRESS. Data is read alternately from the ADC **(D)** and time **(E)** sections of memory and copied to the data array stored as the equipment. The loop must stop if it reaches the end of channel memory and restart at the top of memory. In `WFD_RDMEM`, the loop is over a specified number of memory locations. Since no additional calculation is necessary the readout loop is nearly as compact as in VME_RD_EX (the fixed location VME read routine, see CADRIVER.PAS on VXMACB::). The difference is that it performs two read operations for each loop iteration.

In the case of `WFD_RDTIM`, we need additional code to calculate the elapsed time in units of clock tics (5 ns at 200 MHz). After the 32-bits of time plus discriminator information is stored in DAQ memory, the upper 16-bits, which contain the time-word, is compared against the previous time-word. If the new time-word is larger than the old time-word, the 16-bit ASIC timer has rolled over and the elapsed time must be calculated accordingly. The loop over memory terminates when the requested amount of time has been read-out.

Reminder: originally, each roll-over was going to be marked by the ASIC in a special digitization cycle. However the ASIC design has borderline timing in generating that cycle and this feature is deemed unreliable. See my transparencies from the Martina Franca collaboration meeting for more information [8]. The work-around is to put external pulses into the NIM_CYCLE input to generate a reliable stream of time-words and find roll-overs by the comparison described above.

6 Data Size

To study the amount of data in 1 ms of waveform digitization, I took data with 2:1 fan-in of MACRO horizontal tanks set at -4 mV/s.p.e. Because we have not completely solved the noise problems present when the pmt signals are connected through the fanout to the WFD [9], I bypassed the fanout and used the end of the pmt signal cables directly. The signals are very quiet in this configuration, and I was able to set the WFD discriminator levels (which control zero suppression) to approximately -2 mV threshold. Figure 3 shows 1 ms of background data. You can discern the baseline provided by the regular NIM_CYCLE pulses, produced here every 35 μsec.

Assuming 2:1 fan-in of the horizontal tanks, and a mix of 3:1 and 4:1 fan-in of the vertical tanks, I estimate there will be 3300 bytes of data for every channel (per msec). This assumes no noise problems, and NIM_CYCLE data (pedestal) will take up about 13% of the data. The amount of NIM_CYCLE can be turned down once we gain confidence in the data, perhaps to less than 5% of the data. There are about 65 channels per supermodule, for a total of .21 Mbytes per full supermodule of WFD channels.

For comparison, current MACRO data files are approximately 74 Mbytes for a full-length run (21000 events) of the lower 6 supermodules. A run of this type has approximately 480 two-face slow monopole triggers and 6000 single-face slow monopole triggers. Scaling this to attico running, we can expect 9700 single-face triggers and 700 two-face triggers.

In addition to the scintillator monopole triggers, we wish to read out some of the WFD for streamer tube monopole triggers, muons and the LIP trigger. A trigger budget must be worked out that will balance reading a lot of the detector for rare triggers versus reading a little of the detector for high-rate triggers. In any case, we must prepare for a significant increase in data size, assuming we can solve the readout speed issues.

7 Readout Speed

The readout speed was measured by using a digital storage scope to record the pattern of TTL address strobes ($\overline{AS}$), a VME backplane signal available on U60 pin 4 of the WFD card. Using a CAMAC-list similar to the example in this note, it is possible to see the complete structure of the readout times, including the overhead when going from one CAMAC-list command to the next (about 40 μsec).

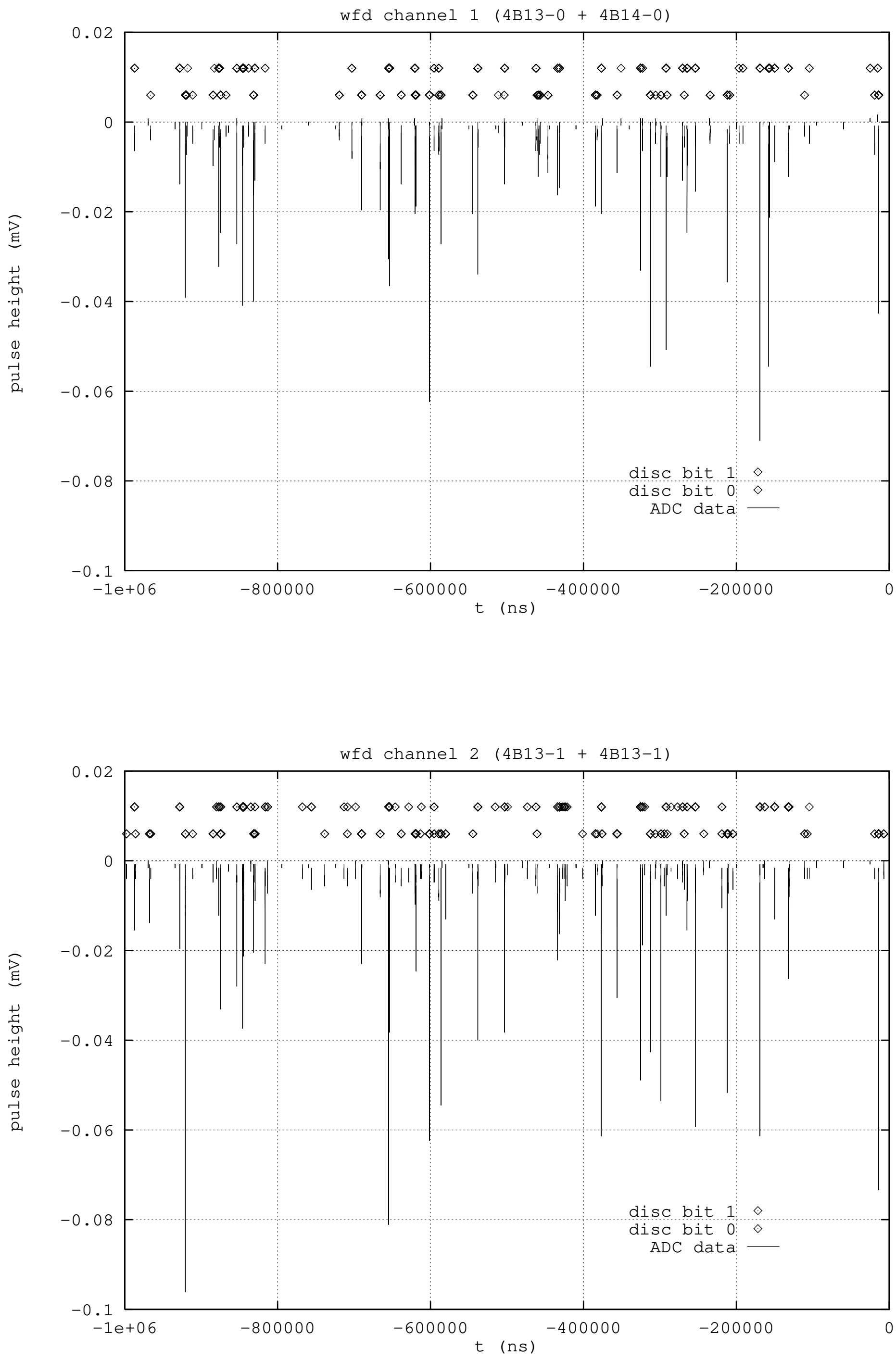


Figure 3: One msec of background radioactivity from 2:1 fan-in of two horizontal tanks. The rows of plot symbols are used to indicate which of the discriminator bits is present for a given ADC sample.

Figure 4 reproduces the $\overline{AS}$ timing for two different readout algorithms (`WFD_RDMEM` and `WFD_RDTIM`) executed on two different DAQ systems (VaXELN on the μVaX and OS9 on the MVME-167 in-crate computer). Each pulse corresponds to a 32-bit VME operation. The first three pulses correspond to executing the STOP ADDRESS read-back function, steps **(A)**,**(B)** and **(C)** of the algorithm described above. The rest of the pulses represent reading 5 WFD data cycles, steps **(D)** and **(E)** repeated 5 times.

In the case of `WFD_RDTIM`, the first WFD data cycle is an initialization step that is outside of the loop; therefore one expects a slight difference in timing between the first two pulses and the rest that follow. Curiously, there is also a noticeable effect for the `WFD_RDMEM` algorithm, where all the WFD cycles are read inside a loop. I do not fully understand this effect; I surmise it is due to some address pre-fetch architecture of the VME control chip; but it is not important. I just want to point out that the slight variation in the pattern is as recorded on the scope and is not a mistake.

Because of the large amount of data we are reading out, we need only compare the top readout speed of the algorithms (the overhead is at worst a 3% effect). The transfer rates are tabulated below, including the elapsed time for reading out a full supermodule of data:

algorithm	DAQ	transfer rate	Δt 1 SM
`WFD_RDTIM`	μVaX + VIC	.54 Mbytes/sec	.39 sec
`WFD_RDMEM`	μVaX + VIC	.80 Mbytes/sec	.27 sec
`WFD_RDTIM`	MVME-167	.84 Mbytes/sec	.25 sec
`WFD_RDMEM`	MVME-167	2.54 Mbytes/sec	.08 sec

The notable feature is that the fixed length readout is very fast on the in-crate computer, while all other algorithm/system combinations are considerably slower. This can be understood empirically. Each computer can execute VME operations at a specific top-speed. That top-speed is characterized by the time between the pairs of address strobes shown in figure 4. Adding code (`WFD_RDTIM`) to the tight readout loop of `WFD_RDMEM` spaces the pairs of address strobes further apart, but leaves the time within the pair unchanged. At its fastest, the μVax/VIC combination generates read cycles every 3 μsec, and it appears that adding code has a smaller percentage effect compared to this interval. Note that there need be no obvious relationship between the incremental time for an extra microprocessor instruction and the characteristic time of the VME readout. The VME operations are controlled by a separate chip, not the main microprocessor.

There are several effects that can contribute to the overall slowness of the μVaX/VIC configuration (compared to the in-crate computer); but I do not know the relative magnitudes. Consider the delay across the entire VICbus system. Any VME read cycle begins when the master sets the address and control lines and produces an address strobe ($\overline{AS}$), and ends when the slave (WFD) acknowledges that the transfer is done ($\overline{DTACK}$) [10]. Since the master waits until it receives the acknowledgement before starting the next cycle, the transfer rate is limited by the round-trip time of the signal from master to slave to master. In the MACRO DAQ, the μVaX generates a VME operation, which is accepted by the VIC-8250 in

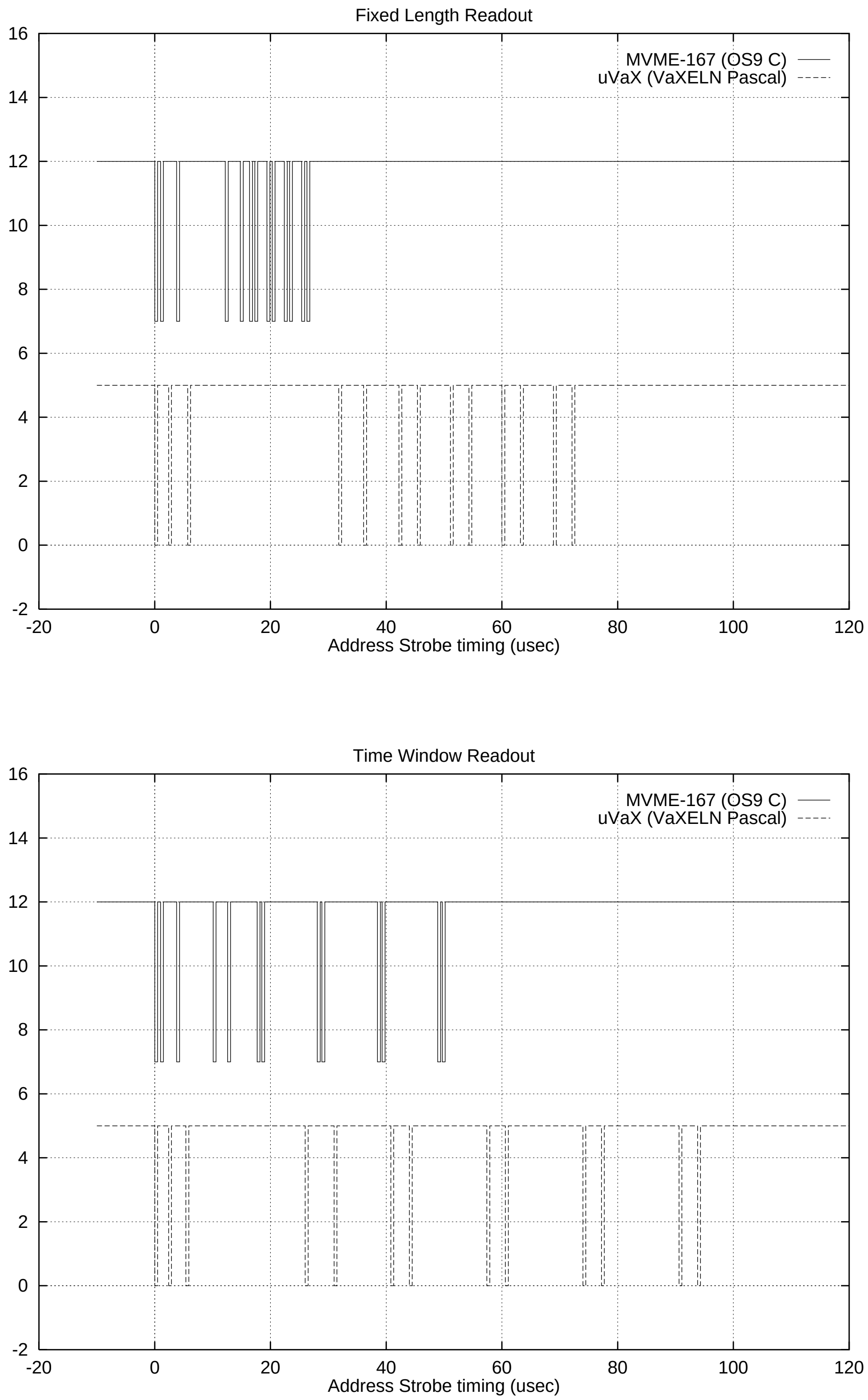


Figure 4: Address strobe timing of the readout algorithms. The strobe pulses match the annotations of the algorithm described in the text as follows: **(A) (B) (C) (D) (E) (D) (E) (D) (E) (D) (E) (D) (E)**
.

the system VME crate, which translates it into VICbus protocol and sends it down a long ribbon cable to the VIC-8250 receiver in the WFD VME crate which translates it back into a VME protocol operation on the WFD backplane. The WFD board acknowledges this in 150 ns. The VME acknowledge is recognized by the VIC-8250 in the WFD crate and turned into VICbus protocol which is sent back along the ribbon cable to the VIC-8250 in the system VME crate, which translates it back to VME protocol and sends it on to the μVaX. Finally, the μVaX must recognize that the previous cycle is completed and begin the next cycle.

Although the VICbus protocol advertises the potential for high transfer rates (in excess of 8 Mbytes/sec), it appears that the rates we are seeing are in line with our hardware [11]. The KAV30 has a rather old VME chipset and the VIC-8250 supports only a subset ("VMV") of the VICbus features. The high transfer rates reported in [3] were measured using a 68040 computer and the VICbus compliant VIC-8251. In thi sconfiguration, it is possible to set up a "non-compelled protocol" where the master does not have to wait for the acknowledgement before starting the next cycle (a form of pipelining).

Therefore, I do not expect the transfer rate using the current hardware to improve beyond 1 Mbyte/sec. Future DAQ upgrades may alleviate this limitation, but for now the only near-term alternative would be a WFD VME crate processor to prepare the data for block transfer. When considering the advantages of local preprocessing of the data, keep in mind that the somewhat faster performance of the in-crate computer is negated by the need to transfer the data twice: once from the WFD cards to a local buffer and again to the DAQ via VIC or ethernet. However, the second transfer of a single block of memory should approach the top speed for the network.

8 Summary

- The general feasibility of VIC readout of the WFD system has been proven. A new CAMAC-list command has been added to execute time-window readout of a WFD channel.
- A general purpose data structure has been defined for WFD-type equipments.
- One supermodule of 1 ms waveform data is estimated to occupy approximately .2 Mbytes (despite hardware zero-suppression). This is expected to increase the net size of MACRO data by a factor of 4 to 10, depending on how we budget space for each trigger.
- The transfer rate of the time-window algorithm is approximately .5 Mbytes per second. Improving the code is not likely to speed it up beyond .8 Mbytes per second.
- Reading out a full supermodule at the above rate will take approximately .4 seconds. This is acceptable for very infrequent triggers such as slow monopole coincidence. But most other triggers must request a limited amount of WFD data.
- To do: modify the readout algorithm to save a narrow time-window of data for exclusive muon triggers.

- To do: modify the readout algorithm to process CAMAC data stored in the equipment header. We want to be able to read out only selected channels, as recorded by a pattern register. We also want to be able to extend the readout time beyond 1 ms on an event by event basis to account for the variable delay produced by (and read-back from) the Stop Master (see [6]).
- To do: optimize the algorithm where possible and continue general testing for robustness.
- To do: develop data budgets for each trigger. These should be based on measured trigger rates, accounting for correlations. Define prototype CAMAC-list configurations for MACRO data running.

References

[1] talk by Surdo, Caltech Collaboration Meeting Transparencies, MACRO note 1003/93 page 528.

[2] The CAMAC-list commands were tested and used for initial studies at Gran Sasso by Hong, Walter, Lu and Sanzgiri.

[3] C. Parkman, "VICbus: VME Inter-Crate Bus, A Versatile Cable Bus", IEEE0018/9499 1992.

[4] Creative Electronic Systems SA, "VIC-8250 Users Manual", 1991. Note: the CES manuals refer to "VMVbus", which is their proprietary bus system that was used as the basis for the VICbus standard. In fact, the VIC-8250 does not support the full functionality of VICbus, such as non-compelled prototcol.

[5] W. Earle, "VMEbus Specification for the MACRO Waveform Digitizer", MACRO note 1030/91. Note: this memo has a few discrepancies with respect to the final WFD design, and will be superseded by new documentation.

[6] See email notes on the informal meeting of 3/26/93, as well as the STOP master messages from R. Liu. These are available from the WFD listserver archive. For an index, send email to listserv@cithe501.cithep.caltech.edu with "RUN WFD GRANSASSO SCAN" in the message body. The motivation is also briefly recapitulated in MACRO memo 1023/94 by B. Nolty.

[7] I wish to explicitely thank A. Surdo and F. Ronga for their assistance with the DAQ software for these studies.

[8] talk by Kearns, Bari Collaboration Meeting Transparencies, MACRO note 26/93 page 380.

[9] There appears to be noise on the wfd inputs that is related to the switching-style power supply. This has been reduced to about 10 mV as of now, and A. Sanzgiri and the TAMU group are working on reducing it to the 1 mV level.

[10] VITA, “VMEBus Specification”, IEEE1014-1987. For a more lucid explanation of some of the details, see “Understanding the VMEbus Specification”, by W. Earle, distributed at the Cape Cod WFD meeting and available on request from Boston U.

[11] A. Lounis, Creative Electronic Systems, private communication.

A Sample CAMAC List

```
c Example CAMAC+VME list showing WFD readout (ETK 26-Jan-1994)
RDTRIG 1,1,3,0,0
INIT
SETNOQ
c Remove Inhibit and Z on crate 1
t 1,1,28,8,26
t 1,1,30,9,24
c SET24
c write on a Borer display
F 1,1,3,0,16,1
EOI
c
c ----------------------------
c Equipment definition       |
c ----------------------------
c
EQUIP 1
c ---------------------------------------------------------------
c WFD equipment has well defined structure. See comments.
c Obey this structure and print/plot routines will work. *** ETK
c (Data can be added to header as we decide what is necessary)
c ---------------------------------------------------------------
SET24
VME32
c *** First word of WFD equip is length of header ***
WRITEHEX $0001
c *** Currently, the only header word, is number of channels. ***
c *** _Last_ word of WFD equip header is number of channels ***
WRITEHEX $0003
c *** Now we have n-channels of readout (WFD_RDTIM or WFD_RDMEM) ***
WFD_RDTIM 1,2,$700000,200000
WFD_RDTIM 1,2,$710000,200000
WFD_RDTIM 1,2,$720000,200000
EOE
C
c
c ----------------------------
c     Triggers definition    |
c ----------------------------
c
TRIG 1
EQUIP 1
EOT
C
```

B WFD_RDTIM in VaXELN Pascal

```
{ WFD Time Window Read ------------------------------------------------------}
{ *** This is test code for wfd intelligent readout ************************* }
{ *** In the future, this may become a user_procedure ********************** }
{ ************************************************************************** }
{ This section of code reads wfd data words (2 x 32 bits) until a certain     }
{ amount of time (in units of 5 ns at 200 MSPS) has elapsed. It begins by     }
{ using the special wfd function that returns the address pointer that        }
{ records the last memory location digitized when the STOP signal happens.    }
{ The code then read "n" wfd data, from the adc bank and time bank. The code  }
{ calculates the elapsed time by looking at the WFD timeword as it is stored. }
{ When the requested time has elapsed (passed in point^.spare), it calculates }
{ the length of the data and stores that at the beginning of the equipment,   }
{ as well as in evt_word (used by the daq). This routine handles wrap-around  }
{ reading of the wfd memory.                                                  }
{ ETK: 25-Jan-1994                                                            }

55:            begin

                { base address is CAMAC list argument }
                wfd_base_add := point^.l_mem_add;
                { calculate address of control register }
                wfd_creg_add::integer := wfd_base_add::integer + %xFFE0;

                { Put wfd card into address readback mode: }
                wfd_creg_add^ := %x08080808;                                (A)
                { Readback stop address: }
                wfd_address::integer := wfd_base_add^;                       (B)
                { Put wfd card back into standard operating mode: }
                wfd_creg_add^ := %x41414141;                                (C)
                { Calculate wfd stop address: }
                { This formula is complicated because WFD returns }
                { address of *next* cycle. So I have to add 4 but }
                { be careful of memory boundary at base+0xfffc.   }
                wfd_address::integer := wfd_base_add::integer
                    + (((wfd_address::integer DIV %x10000) +4) MOD %x8000);
                wfd_addr_t::integer  := wfd_address::integer + %x8000;
                { Calculate loop starting value: }
                start := (wfd_address::integer - wfd_base_add::integer) DIV 4;
                count := start;

                { pointer setup (see VME Long Read 50:) }
                evt_word := evt_word+1; {1}
                tmp_buf_p := address(evt_buffer[id]^[evt_word]);
                tmp_buf_ptr::integer := tmp_buf_p::integer;
                { We will calculate data length from address of tmp_buf_p }

                { Header of this data structure:                }
                {     [ starting address of wfd data       ]     }
                {     [ tics from stop to first cycle (t0) ]     }
                {     [ length of wfd data (32 bit words)  ]     }
                tmp_buf_ptr^ := wfd_address::integer;
                tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
                tmp_buf_ptr^ := 0;
                tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
                wfd_len_ptr := tmp_buf_ptr;   { <-- Fill in later }
```

```
tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;

{ Read first data to initialize tlast: }
{ This is nearly the same as in the loop below. }
begin
   tmp_buf_ptr^ := wfd_address^;                            (D)
   tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
   tmp_buf_ptr^ := wfd_addr_t^;                             (E)

   { Calculate tlast: }
   tlast := tmp_buf_ptr^ DIV %x10000;

   tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
   wfd_address::integer := wfd_address::integer+4;
   wfd_addr_t::integer := wfd_addr_t::integer+4;

   count := count + 1;
end;

{ Loop over memory and alternate reading ADC and time data }
{ The constant 8191 represents 7ffc in units of four bytes }
t := 0;
while (count < 8191) AND (t < point^.spare) do
  begin
     tmp_buf_ptr^ := wfd_address^;                          (D)
     tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
     tmp_buf_ptr^ := wfd_addr_t^;                           (E)

     { Calculate elapsed time: }
     tnow := tmp_buf_ptr^ DIV %x10000;
     if (tlast < tnow) then
        t := t + tlast + 65536 - tnow
     else
        t := t + tlast - tnow;
     tlast := tnow;

     tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
     wfd_address::integer := wfd_address::integer+4;
     wfd_addr_t::integer := wfd_addr_t::integer+4;

     count := count + 1;
  end;

{ This second loop is only executed if we are near the }
{ end of memory for this channel and we must wrap to   }
{ the beginning of memory to finish reading. The       }
{ condition (count < start) ensures that we read at    }
{ most the entire 64K memory only once.                }
wfd_address := wfd_base_add;
wfd_addr_t::integer  := wfd_address::integer + %x8000;
count := 1;
while (count < start) AND (t < point^.spare) do
  begin
     tmp_buf_ptr^ := wfd_address^;                          (D)
     tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
     tmp_buf_ptr^ := wfd_addr_t^;                           (E)

     { Calculate elapsed time: }
```

```
      tnow := tmp_buf_ptr^ DIV %x10000;
      if (tlast < tnow) then
         t := t + tlast + 65536 - tnow
      else
         t := t + tlast - tnow;
      tlast := tnow;

      tmp_buf_ptr::integer := tmp_buf_ptr::integer+4;
      wfd_address::integer := wfd_address::integer+4;
      wfd_addr_t::integer := wfd_addr_t::integer+4;

      count := count + 1;
    end;

  { Increment totals. }
  wfd_len_ptr^ := (tmp_buf_ptr::integer -
                   wfd_len_ptr::integer - 4) DIV 4;
  evt_word := evt_word + 2*(wfd_len_ptr^) + 6 - 1; {1}
  eqp_word := eqp_word + 2*(wfd_len_ptr^) + 6;

end;
```